Chapter 1

Introduction

Contents

Basic Terminologies : Elementary Data Organizations, Data Structure Operations : Insertion, deletion, traversal etc.; Analysis of an Algorithm, Asymptotic Notations, Time-Space Trade off.

Searching : Linear Search and Binary Search Techniques and their Complexity analysis.

POINTS TO REMEMBER

- Data refers to a value or a set of values which may represent some observation from an experiment or some facts/figures gathered systematically for one or more specific purposes.
- A data item is the single unit of values of certain type that have meaning to its user.
- A file is a collection of related records. A record is a collection of related fields. A field holds a particular kind of data.
- Each entity has certain properties also known as its attributes that describes it.
- Entity set is a collection of entities of same type that share the same properties or attributes.
- The key which is chosen to uniquely identity a record is known as primary key and other key (s) are known as alternate key (s).
- Information is defined as the processed summarized or organized data which when used by its recipient helps in taking decisions.
- Data structure is a logical or mathematical model of a particular arrangement or organization of data. An efficient data structure uses minimum memory space and execution time to process the structure as possible.
- A linked list is a dynamic data structure containing of nodes where each node is composed of data and a pointer to the next node.
- A stack is a linear data structure in which all insertions and deletions are restricted only at one end called the top of the stack. A stack is often described as FIFO. The basic operations performed on a stack are 'Push' and 'Pop'.
- A queue is a linear data structure in which elements are inserted at one end called the rear end and deleted from the other end called the front end. A queue is said to have FIFO behaviour. Enqueue and dequeue are two basic operations that can be applied on a queue.

- A graph is a non-linear data structure that consists of set of nodes (or vertices) and set of edges (or arcs), which each edge going from one node to another. Each edge in the graph depicts a relationship between pair of nodes.
- Traversing, searching, insertion, deletion, sorting, merging, copying, concatenation are key operations performed on a data structure.
- An algorithm is a sequence of steps or instructions required to solve a given problem.
- The analysis of algorithm based on how much memory an algorithm needs to solve a particular problem is called the space complexity of an algorithm.
- Measure of time complexity of an algorithm where we disregard certain terms of a function to express the efficiency of algorithm is called asymptotic complexity.
- For a given input, the efficiency of an algorithm is complexity is measured for average case, best case and worst case.
- Searching is an operation in which a given item is found in a list of numbers.
- Linear search search the element sequentially starting from the first element.
- Binary search is used to find an element of a sorted list only.
- In binary search, element present in the middle of the list is determined
- In Linear Search data must be stored in any order.
- The Binary Search data may be stored in order.
- Time complexity of Linear Search is O (n).
- Time complexity of Binary Search is O (log n).
- BSM search elements through middle element.
- LSB search elements one by one.

QUESTION-ANSWERS

Q 1. List various non-linear data structures. **(PTU, May 2004)**

Ans. Some of the non linear data structures are graphs, trees, multilinked structures like sparse matrices etc.

Q 2. What is the difference between data structure and data types? **(PTU, May 2004)**

Ans. Difference between data structure and data types are as follow :

Data Structure	Data Types
Data structure is a logical or mathematical model of a particular organization of data. e.g. STACK, QUEUE, LINK LIST etc.	A data type is a term which refers to the kind of data that variable may hold in a programming language. e.g. Integer, float, character etc.

Q 3. What is meant by merging? **(PTU, Dec. 2004)**

Ans. Merging means combining the records in two different sorted files into a single

sorted file. In other words, merging means combining the records from two different lists into a single list.

Q 4. What are non-linear data structures? **(PTU, Dec. 2006 ; May 2007, 2004)**

OR

What do you mean by non-linear data structure? Give examples.

(PTU, May 2011 ; Dec. 2008)

Ans. The non-linear data structure are those which are capable of expressing more complex relationships than that of physical adjacency. Examples of these data structures are trees, graphs, etc.

Q 5. Compare the running time of linear search algorithm with binary search algorithm. **(PTU, Dec. 2004)**

Ans. In case of linear search algorithm, the time required to execute the algorithm is proportional to no. of comparisons. The average no. of comparisons for a file with 'n' records is $= \frac{n}{2}$, i.e. the complexity of linear search algorithm is given by $C(n) = \frac{n}{2}$.

In case of binary search algorithm, complexity, $C(n) = \log_2 n$.

Q 6. What are data items? **(PTU, Dec. 2005)**

Ans. Data are simply values or sets of values. A data item refers to a single unit of values. Data items that are divided into sub items are called group items, those that are not are called elementary items. For e.g. an employee's name may be divided into 3 subitems – first name, middle initial and last name.

Q 7. For a linear search algorithm, calculate complexity of the algorithm for best care. **(PTU, Dec. 2005)**

Ans. Complexity of algorithm for the best case in a linear search algorithm, occur when item to be searched is, the first element that appears in the list, i.e. $C(n) = 1$.

Q 8. What is data structure? **(PTU, May 2006)**

Ans. Data structure is a logical or mathematical model of a particular organisation of data. The study of DSs includes :

1. Description of DS
2. Implementation of DS
3. Quantitative analysis of structure.

Q 9. What is sorting and merging? **(PTU, Dec. 2006)**

Ans. Sorting : Sorting means arranging the records in some logical order like ascending or descending order. It may be bubble sort, quicksort, etc.

Merging : Merging means combining the records from two different lists into a single list.

Q 10. Explain operations of Data Structure. **(PTU, Dec. 2006)**

Ans. 1. Traversing : It means accessing each element atleast/exactly once in order.

2. Searching : It means finding the location of a record with a given key value or finding locations of all records which satisfy one or more conditions.

3. Inserting : Adding a new record refers to insertion.

4. Deleting : Deleting refers to removing a record.

5. Sorting : Arranging the records in some logical order.

6. Merging : Combining records from two different lists into a single list.

Q 11. Write a short note on linear search. **(PTU, Dec. 2006)**

Ans. In case of linear search, each item is compared with a particular item until the required item is successfully searched. Here, no. of comparisons is equal to the number of elements in the list. Hence, complexity,

$C(n) = O(n)$.

Q 12. What do you mean by linear data structure? Give examples.

(PTU, May 2008)

Ans. A data structure is said to be linear if its elements form a sequence or in other words, a linear list. There are two basic ways of representing such linear structures in memory. One way is to have the linear relationship between the elements represented by means of sequential memory locations. The other way is to have the linear relationship between the elements by means of pointers or links. Examples of linear data structure are arrays, stack, queues etc.

Q 13. What will be the complexity of the linear search algorithm for both the worst case and average case? **(PTU, May 2019, 2008)**

Ans. (i) Worst Case : The worst case occurs when ITEM is the last element in the array DATA or is not there at all. In either situation. We have,

$$C(n) = n$$

According $C(n) = n$, the worst case complexity of the linear scalar algorithm.

(ii) Average Case : In this case, ITEM does appear in DATA and that it is equally likely to occur at any position in the array.

Accordingly, the number of comparisons can be any of numbers 1, 2, 3, ..., n and each number occur with probability.

$$P = \frac{1}{n} \text{ then}$$

$$C(n) = 1.\frac{1}{n} + 2.\frac{1}{n} + ... + n.\frac{1}{n}$$

$$= (1 + 2 + ... + n)\frac{1}{n}$$

$$= \frac{n(n+1)}{2}.\frac{1}{n}$$

$$= \frac{n+1}{2}.$$

This aggress with over initiative feeling that the average number of comparisons needed to find location of ITEM is approximately equal to half the number of elements in the data list.

Q 14. Define complexity of an algorithm. **(PTU, Dec. 2008)**

Ans. The complexity of an algorithm is the function that gives the running time and space in terms of input size.

Therefore time and the space it uses are two major measures of the efficiency of a program. Now each of our algorithm will involve a particular data structure. The choice of data structure depends upon many things, including the type of data structure and frequency with which various operations applied.

"The complexity of an algorithm M is the function f (n) which gives the running time and/or storage space required by an algorithm for processing the input data of size n.

Q 15. Define : Algorithm. **(PTU, Dec. 2009)**

Ans. An algorithm is a finite set of steps defining the solution of a particular problem. An algorithm can be written in English like language, called pseudocode. The coding of algorithms exposed in pseudocode is very simple and straight forward. Algorithm can also be expressed in the form of a flow chart.

Every algorithm mostly satisfy the following conditions :

1. Input : There must be zero or more values which are externally supplied to the algorithm.

2. Output : At least one value is produced.

3. Definiteness : Each step must be clear and unambigous.

4. Finiteness : The algorithm must terminate after a finite number of steps.

Q 16. What is an asymptoic notation? Mention its types. **(PTU, Dec. 2009)**

Ans. Asymptoic notation is a way to express an algorithm's efficiency. It is called asymptotic because it deals with the behaviour of the algorithm as the input size approaches the asymptotic limit of infinity. It is used to describe the running time of an algorithm defined in terms of function whose domain the set of natural numbers.

Types of asymptotic notation :

1. Big-O Notation : It is the formal method of expressing the upper bound of an algorithm's running time. It is a measure of the longest amount of time it could possibly take for the algorithm to complete.

2. Big-Ω Notation : It is same as big-O, but it express the lower bound of an algorithm, instead of upper bound. It describes the best that can happen for a given data size.

3. 0 Notation (Theta Notation) : This provides simultaneously both asymptotic upper bound and asymptotic lower bound for a given function.

4. Little-O Notation : It represents a loose boundary version of Big-O, function bounds from top, but it does not bound the bottom.

5. Little-Ω (Little Omega) : It is much like big omega but its express a loose lower boundary of the function. It bounds from the bottom but not from the top.

Q 17. What do you mean by complexity? **(PTU, Dec. 2009)**

Ans. The analysis of algorithms is a major task in computer science. In order to compare algorithms, we must have some criteria to measure the efficiency of our algorithms. The complexity of an algorithm is the function which gives the running time and/or space in terms of the input size.

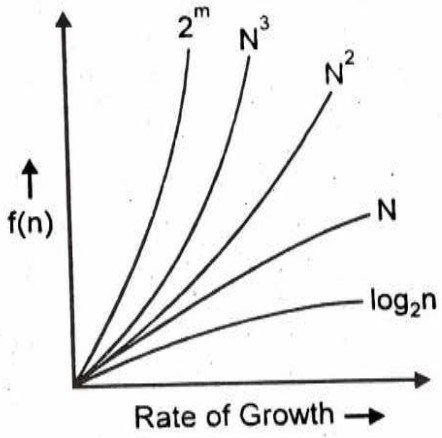
Analysis of space complexity of an algorithm or program is the amount of memory it needs to run for its completion.

The time complexity of an algorithm or a program is the amount of time it needs to run for its completion.

Q 18. What is the complexity of an algorithm? Also explain time space trade off. **(PTU, Dec. 2004)**

Ans. Complexity of an algorithm is the function f(x) which gives the running time and/or space in terms of size and of input data. Rate of growth is the analysis which shows at what rate complexity of algorithm increases with increase of input size (n).

This is done by comparing f(n) with some standard functions such as $\log_2 n$, n, $n \log_2 n$, n^2, n^3, 2^n.



$\log_2 n$ grows most slowly and the exponential function, i.e. 2^n growth most rapidly. Polynomial function, i.e. n^c grows according to exponent 'c'.

Complexities of various algorithm are as follows :

Linear search → o (n)

Binary search → o ($\log_2 n$)

Bubble sort → o (n^2)

Merge sort → o ($n \log_2 n$).

Time space trade off : It specifies that by increasing the amount of space, one may be able to reduce the time needed for processing data or vice versa. The time is measured by counting the no. of key operations. The space is measured by counting max. memory needed for algorithm complexity can be considered in 3 cases :

(i) Worst case : The max. value of f (n) for any possible input.

(ii) Average case : It is the expected value of f(n).

(iii) Best case : It is the minimum possible value of f(n).

Q 19. Write an algorithm for linear search. Also write its complexity.
(PTU, May 2019 ; Dec. 2008)

Ans. Algorithm :

Linear Search : A liner search array DATA with N elements and a specific ITEM of information are given. This algorithm finds the location LOC of ITEM in the array DATA or sets LOC = 0.

Step 1. [Initialize] set K = 1 and LOC = 0

Step 2. Repeat steps 3 and 4 while LOC = 0 and $K \leq N$

Step 3. If ITEM = DATA [K] then set LOC = K.

Step 4. Set K = K + 1 [increment counter]
[End of step 2 loop]

Step 5. [Successful]
It LOC = 0 then
write : ITEM is not in the array DATA
ELSE :
Write LOC is the location of ITEM
[End of If structure]

Step 6. Exit

Complexity

(i) Worst case C (n) = n

(ii) Average case $C(n) = 1.\frac{1}{n} + 2.\frac{1}{n} + ... + n.\frac{1}{n}$

$$= (1 + 2 + ... + n)\frac{1}{n}$$

$$= \frac{n(n+1)}{2}.\frac{1}{n}$$

$$= \frac{n+1}{2}.$$

Q 20. What is Data Structures? What are linear and non-linear data structures?
(PTU, May 2005)

OR

What is data structures? What are different data structures operations?
(PTU, May 2006 ; Dec. 2005)

Ans. Data structure is a logical or mathematical model of a particular organisation of data.

study of data structure includes :
1. Description of data structure.
2. Implementation of structure
3. Quantitative analysis of structure.

The choice of a particular data model depend upon the following considerations :
(i) It should represent the relationships of the data in the real world.
(ii) It should be simple enough so that it can be efficiently processed when required.

The data structures can be divided into two broad categories :

1. Linear : The elements in this type of structure forms a sequence. e.g. Array, linked lists, stacks, queues.

2. Non-linear : The elements in this type of structure doesn't form any sequence. For e.g. Trees and graphs.

Data Structure Operations : There are following operations performed by Data Structure :

1. Transversing : Accessing each element atleast exactly once in order.

2. Searching : Finding the location of record with a given key value of finding locations of all the records which satisfy one or more conditions.

3. Insertion : Adding a new record.

4. Deletion : Removing a record.

5. Sorting : Arraying records in some logical order.

6. Merging : Combining records from two different lists into a single list.

Q 21. With an example, explain how will you measure the efficiency of an algorithm. (PTU, Dec. 2009)

Ans. A algorithm is a null defined list of steps for solving a particular problem. The time and space it uses are two major measures of the efficiency of an algorithms. The complexity of an algorithm is the function which gives the running time and/or space in terms of input size.

Suppose M is an algorithm and suppose n is the size of the input data. Clearly the complexity of (n) of M increases and n increases. It is usually the rate of increase of f (n) that we want to examine. This is usually done by comparing f (n) with some standard function, such as

$$\log_2 n,\ n,\ n\log_2 n,\ n^2,\ n^3,\ 2^n$$

Suppose f (n) and g (n) are functions defined on the positive integers with the property that f (n) is bounded by some multiple of g (n) for almost all n. Suppose there exist a position integer no and a positive number M such that, for all $n > n_0$, we have

$$|g(n)| \le M\ |g(n)|$$

Then we may write

$$f(n) = O(g(n))$$

This is called the "big O notation

For example, for any polynomial P (n) of degree m,

$$P(n) = O(n^m)\ ;\ eg$$

$$8n^3 - 576\ n^2 + 800\ n - 245 = O(n^3)$$

The complexity of well known searching algorithms are :
1. Linear search : O (n)
2. Binary search : O (log n)
3. Bubble search : O (n^2)
4. Merge search : O (n log n)

Q 22. Write algorithm for linear search. Discuss its complexity. (PTU, May 2005)

Ans. LINEAR (DATA, N, ITEM, LOC)

Here, DATA is a linear array with N elements and ITEM is a given item of information. This algorithm finds the location LOC of item in DATA, or sets LOC = 0, if search is unsuccessful.

1. Set DATA [N+1] = ITEM
2. Set LOC = 1
3. Repeat while DATA [LOC] ≠ ITEM
   Set LOC = LOC + 1
4. If LOC = N + 1, then
   Set LOC = 0
5. Exit

Here, complexity, C = O (n).

Q 23. What is time space trade off? (PTU, May 2011)

Ans. In computer science, a space time or time memory trade off is a situation where the memory use can be reduced at the cost of slower program execution. As the relative costs of CPU cycles, ram space has for some time been getting cheaper at a much faster rate than other components of computer the appropriate choices for space-time trade offs have changed radically. Often, by exploiting a space-time tradeoff a program can be made to run much faster.

Q 24. Define data type.

Ans. A data type is a term, which is used to refer the kinds of data that variable may hold in a programming language.

Q 25. What are the various levels of data structure?

Ans. There are three-levels of data structure :
(a) Abstract level
(b) Implementation level
(c) Application level.

Q 26. Write the name of commonly used data structures.

Ans. Commonly used data structure are stacks, queues, lists, trees and graphs.

Q 27. What do you mean by primitive and non-primitive data structure?

Ans. Primitive data structure are those which are provided by the language itself such as int, float, double, char etc. Data structure which are composed of primitive data structure are called as non-primitive data structures.

Q 28. Explain the concept of data type.

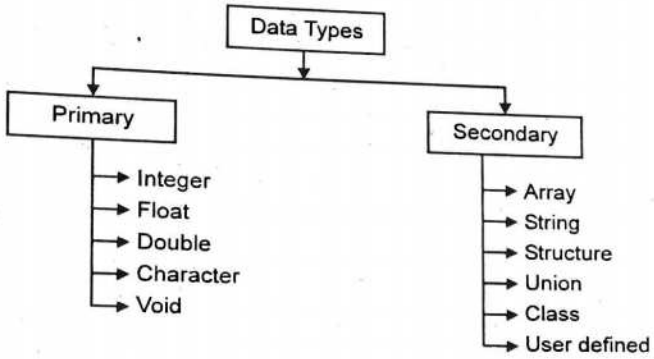
Ans. A data type is a term, which is used to refer the kinds of data that variable may hold in a programming language. In other words the general form of a class of data items is known as data type.

In programming, we store the variable in computer's memory. So there is a need to tell the computer, what type of data we are going to store in computer because for different kind of data, different amount of memory is reserved. So this is done with the help of data type.

Data types can be classified as :

(a) Primary data types

(b) Secondary data types.



Q 29. What are the various types of data structure?

Ans. Data structure are mainly of two types :

1. Linear data structure
2. Non-linear data structure.

Q 30. Write the various applications of data structure.

Ans. The applications of data structure are as follow :

1. It gives considerable help in compiler design.
2. It provides the methods of representing the data elements in the memory of computer efficiently.
3. It describe the physical and logical relationship between the data items.
4. Database management system.
5. It provides the considerable help in managing the operating system.
6. It help in graphics applications.
7. Data structure find its major application in artificial intelligence.
8. It is useful in simulation.

Q 31. Define algorithm.

Ans. An algorithm is a finite set of instructions which if followed, accomplish a particular task.

Q 32. Explain the complexity of algorithms.

Ans. An algorithm is a sequence of statements, each of which has a clear meaning and can be performed with a finite amount of effort is a finite length of time. An algorithm is supposed to do two things.

(a) Compute the correct answer given valid input data.

(b) Perform the computation in a reasonable time.

A correct algorithm is useless if it is too slow. In such a case, we must find another algorithm or any be we will have to accept some approximation which allow us to use another algorithm, which genrates an approximately correct results.

These analysis are done without reference to any specific computer or programming language, the issue is how many calculation steps are needed to obtain the result and how this depends on the size of the problem. One of the most important properties of algorithm is how its execution time increases as the problem is made large. By a large problem, we mean sequences to align or longer sequences to align. This is so called complexity of algorithm.

Q 33. Explain the concept of Big-O notation.

Ans. Big-O is the formal method of expressing the upper bound of an algorithm's running time. It is a measure of the longest amount of time it could possibly take for the algorithm to complete.

Mathematical description of Big-O notation

Assume f (n) and g (n) be two arbitrary functions, such that

f (n) = 0 (g (n)), read as f (n) is "big-0" of g (n) or f (n) is of order g (n), if there exists positive constants n_0 and C, such that $f(n) \leq C^* g(n)$ for all $n > n_0$.

Example :

(a) $100\ n^3 \Rightarrow 100 * n^3$ is O (n^3)

(b) 1000 is 0 (1)

(c) n + log n is O (n) because log n < n

(d) $2n^2 + 3n + 4$ is O (n^2).

Q 34. What is Big 'O' notation? Explain its significance. **(PTU, May 2012)**

Ans. In mathematics, **big O notation** is used to describe the limiting behaviour of a function when the argument tends towards a particular value or infinity, usually in terms of simpler functions. It is a member of a larger family of notations that is called **Landau notation, Bachmann-Landau notation** (after Edmund Landau and Paul Bachmann), or **asymptotic notation**. In computer science, big O notation is used to classify algorithms by how they respond (e.g., in their processing time or working space requirements) to changes in input size.

Big O notation characterizes functions according to their growth rates : different functions with the same growth rate may be represented using the same O notation. A description of a function in terms of big O notation usually only provides an upper bound on the growth rate of the function. Associated with big O notation are several related notations, using the symbol o, Ω, ω, and , to describe other kinds of bounds on asymptotic growth rates.

Big O notation is also used in many other fields to provide similar estimates.

Example : Suppose an algorithm is being developed to operate on a set of n element Its developers are interested in finding a function T (n) that will express how long the algorith

will take to run (in some arbitrary measurement of time) in terms of the number of elements in the input set. The algorithm works by first calling a subroutine to sort the elements in the set and then perform its own operations. The sort has a known time complexity of O (n^2), and after the subroutine runs the algorithm must take an additional $55n^3 + 2n + 10$ time before it terminates. Thus, the overall time complexity of the algorithm can be expressed as

$$T(n) = O(n^2) + 55^3 + 2n + 10.$$

This can perhaps be most easily read by replacing O (n^2) with "some function that grows asymptotically no faster than n^2." Again, this usage disregards some of the formal meaning of the "=" and "+" symbols, but it does allow one to use the big O notation as a kind of convenient place holder.

Q 35. Define the term data structure and discuss the criteria used for evaluating the suitability of a particular data structure for a given application. (PTU, Dec. 2012)

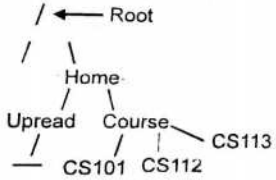
Ans. Data Structure : Data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently.

Types of data structure

1. Linear data structure
2. Non-linear data structure

Application : Array and linked list, which are linear data structures, tree is hierarchical (or non-linear) data structure.

One reason to use tree might be because user wants to store information that naturally forms a hierarchy. For example, the file system on a computer.

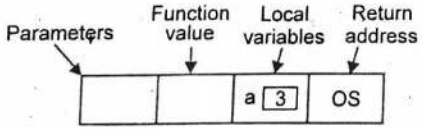


Q 36. What data structure is used by compiler to handle function calls and how? (PTU, May 2014)

Ans. A call stack is used by complier to handle function calls. A call stack is a stack data structure that stores information about the active subroutine of a computer program. This kind of stack is also known as an execution stack, control stack, run-time stack or machine stack, and is often shortened to just "the stack".

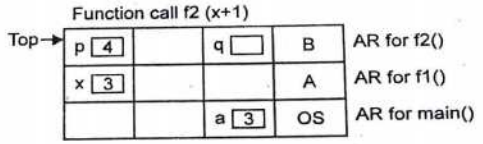
Consider events when a function begins execution

- ❑ Activation record or stack frame is created.
- ❑ Store the current environment for that function



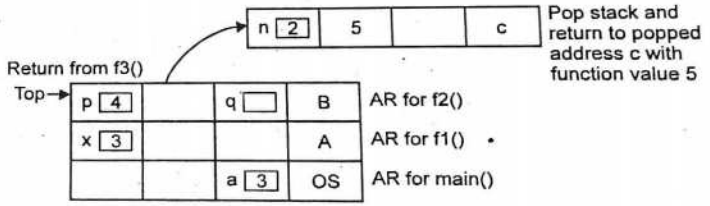
When a function is called...

(i) Copy of activation record pushed onto call stack or run-time stack.
(ii) Arguments copied into parameter spaces.
(iii) Control transferred to starting address of body of function.



When function terminates

(i) Run-time stack popped
- ❑ Remove activation record of terminated function.
- ❑ Exposes activation record of previously excuting function.



(ii) Activation record used to restore environment of interrupted function.
(iii) Interrupted function resumes execution.

Q 37. What do you mean by searching?

Ans. Searching is an operation in which a given item is found in a list of numbers.

Q 38. What are the various kinds of searching in data structure?

Ans. Searching are basically of two types :

(a) Linear Search
(b) Binary Search

Q 39. What is linear search?

Ans. Linear search is the simplest form of search. It searches for the element sequentially starting from the first element. This search has a disadvantage if the element is located at the end. Advantage lies in the simplicity of the search. Also it is most useful when the element are arranged in a random order.

Q 40. What is binary search?

Ans. Binary search is most useful when the list is sorted. In binary search, eleme present in the middle of the list is determined. If the key (number to search) is smaller tha the middle element, the binary search is done on the first half. If the key (number to searc

is greater than the middle element, the binary search is done on the second half (right). The first and the last half are again divided into two by determining the middle element.

Q 41. Differentiate between linear search and binary search.

Ans. The main difference between linear search and binary search are as follow :

Linear Search	Binary Search
1. In Linear Search data must be stored in any order.	1. In Binary search data may be stored in order.
2. Linear Search method is slow.	2. Binary search method is fast.
3. Time complexity is O (n).	3. Time complexity is O (log n).
4. LSM search elements one by one.	4. BSM search elements through middle element.
5. LSM is best when data is sorted or unsorted but short in length.	5. BSM is best when data is sorted.

Q 42. What do you mean by searching? What are the various types of searching?

Ans. Searching is an operation in which a given item is found in a list of numbers.

Types of Searching : Searching are basically of two types :

(a) Linear Search

(b) Binary Search

(a) Linear Search : Linear search is the simplest form of search. It searches for the element sequentially starting from the first element. This search has a disadvantage if the element is located at the end. Advantage lies in the simplicity of the search. Also it is most useful when the elements are arranged in a random order.

(b) Binary Search : Binary search is most useful when the list is sorted. In binary search, element present in the middle of the list is determined. If the key (number to search) is smaller than the middle element, the binary search is done on the first half. If the key (number to search) is greater than the middle element, the binary search is done on the second half (right). The first and the last half are again divided into two by determining the middle element.

Q 43. What is the advantage of binary search algorithm? **(PTU, May 2004)**

Ans. Binary search is an extremely efficient algorithm. Binary search technique searches data in minimum possible comparisons with the time complexity of O ($\log_2 n$).

Q 44. Write an algorithm for Binary search. **(PTU, May 2008)**

Ans. BINARY (DATA, LB, UB, ITEM, LOC)

The DATA is sorted array with lower bound LB and upper bound UB and ITEM is a given item of information. The variables BEG, END and MID denotes respectively begining, end and middle locations of a segment of elements of DATA. This algorithm finds the location LOC of ITEM in DATA or sets LOC = NULL.

Step 1 : [Initialize segment variable]
set BEG = LB, END = UB and MID = INT [(BEG + END)/2]

Step 2 : Repeat steps 3 and 4 while BEG ≤ END and
DATA [MID] ≠ ITEM

Step 3 : If ITEM < DATA [MID] then
set END = MID–1
else
set BEG = MID+1
[END of if structure]

Step 4 : Set MID = INT ((BEG + END) /2)
[END of step 2 loop)

Step 5 : If DATA [MID] = ITEM then
Set LOC = MID
else
set LOC = NULL
[End of if structure]

Step 6 : Exit

Complexity of Binary search algorithm

$$f(n) = [\log_2 n] + 1$$

Q 45. Compare the linear search with binary search. **(PTU, Dec. 200**

Ans. In case of linear search, the elements are searched one by one, i.e. each eleme is compared with a given element one by one.

Here, complexity is given as :

$$1.\frac{1}{n} + 2.\frac{1}{n} + 3.\frac{1}{n} + \ldots\ldots + n.\frac{1}{n}$$

$$(1 + 2 + 3 + \ldots\ldots + n).\frac{1}{n}$$

$$n\left(\frac{n+1}{2}\right).\frac{1}{n}$$

$$= O\left(\frac{n+1}{2}\right) = O\left(\frac{n}{2}\right)$$

i.e. $C(n) = \frac{n}{2}$.

In case of binary search, there are 2 primary conditions :

1. The data must be in sorted order.
2. One must have direct access to middle element.

$$MID = INT\left(\frac{BEG + END}{2}\right)$$

1. If A [MID] = ITEM, then Location = MID
2. If A [MID] > ITEM, BEG = 1, END = MID – 1
ELSE
BEG = MID + 1, END = N.

In case of binary search, complexity i.e.

$C(n) = \log_2 n$.

Write an algorithm for binary search and discuss its limitations. **(PTU, Dec. 2005)**

Ans. BINSEARCH (DATA, LB,UB, ITEM, LOC)

Here, DATA is sorted with lower bound LB and upper bound UB and ITEM is any given item of information. This algorithm finds the location LOC of ITEM in DATA or sets LOC = NULL

1. Set BEG = LB, END = UB and
   MID = INT [(BEG + END)/2]
2. Repeat steps 3 and 4 while BEG ≤ END
   and DATA [MID] ≠ ITEM.
3. If ITEM < DATA [MID], then :
   Set END = MID – 1.
   Else.
   Set BEG = MID + 1
4. Set MID = INT [(BEG + END)/2]
5. If DATA [MID] = ITEM, then
   Set LOC = MID
   ELSE :
   Set LOC = NULL
6. Exit.

Limitations of Binary search : The primary conditions for binary search are :

1. The data must be in sorted order, thus, it can't be used in case data is unsorted.
2. One must have direct access to middle element. Thus, they aren't used for searching in a link list as there is no direct access to middle element.

Q 47. What do you mean by binary search? Write an algorithm that demonstrates binary search. **(PTU, Dec. 2011)**

Ans. In computer science, a binary search algorithm (or binary chop) is a technique for locating a particular value in a sorted list. The method makes progressively better guesses, and closes in on the location of the sought value by selecting the middle element in the span which, because the list is in sorted order, is the median value), comparing its value to the target value, and determining if it is greater than, less than, or equal to the target value. A guessed index who value turns out to be too high becomes the new upper bound of the span, and if its value is too low that index becomes the new lower bound. Only the sign of the difference is inspected : there is no attempt at an intepolaton search based on the size of the difference. Pursuing this strategy iteratively, the method reduces the search span by a factor of two each time, and soon finds the target value or else determines that it is not in the list at all. A binary search is an example of a dichotomic divide and conquer search algorithm.

```
// recursive binary search
// returns index of found element
// or the ones complement of where it should be if not found
int binsearch (int*a, int n, int low, int high) {
int mid = (high + low) / 2;
if (high < low) return – low – 1 ;
```



```
if (a [mid] = = n) return mid ;
if (a [mid] < n) return binsearch (a, n, mid + 1, high) ;
else return binsearch (a, n, low, mid-1) ;
}
int a [ ] = {2, 3, 5, 7, 11, 13, 17, 23, 27} ;
int n, r ;
for (; ;) {
cout <<end/ << "Enter search term :" ;
cin >> n ; if (n = = 0) break ;
r = binsearch (a, n, 0, 8) ;
cout << end / << "Result :" ;
if (r > = 0) cout << "Found : " << r << " " << a [r] << end / ;
else {
cout <<"Not Found :" << – r – 1 << " " ;
if (– r – 1 < 9) cout << a [– r – 1] << end / ; else cout << "?" << end/ ;
}
} ;
```

Q 48. Suppose a sequence of numbers is given like :

5, 10, 12, 18, 56, 68, 52, 85, 95

(a) What are the various steps in which the number 52 will be found by the Binary search?

(b) In how many steps the number 52 will be found in the linear search?

(c) In how many steps it will be found in the binary search that the number 8 does not exist in this array?

Explain the algorithm involved in each of the problems a, b, c. **(PTU, May 201**

Ans. (a) 5, 10, 12, 18, 56, 68, 52, 85, 95

Beg = 5

End = 95

$$\text{Mid} = \frac{\text{Beg} + \text{End}}{2} = \frac{1+9}{2} = \frac{10}{2} = 5$$

5 10 12 18 (52) 56 68 85 95

Beg = 56

End = 95

The step is only 1 for 52 will be found.

5
10
12
18
56
68
52
85
95
52

(b) Step 1. Add 52 in the last
then match it with first no

Step 2. 52 match with 5

Step 3. 52 match with 10

Step 4. 52 match with 12

Step 5. 52 match with 18

Step 6. 52 match with 56

Step 7. 52 match with 68

Step 8. 52 match with 52
It takes total 8 steps.

(c) 5 10 12 18 52 56 68 92
Now
Beg = 5 (1)
End = 92 (9)

Step 1.

$$\text{Mid} = \frac{1+9}{2} = \frac{10}{2} \text{ i.e. } 52$$

5 10 12 18 (52) 56 68 85 95

Left = 56 End 95

Left = Mid + 1 = 5 + 1 = 6

Step 2.

56 68 85 95

$$\frac{\text{Beg}+\text{End}}{2} = \frac{1+4}{2} = \frac{5}{2} = 2.5$$

It will take 3 steps.

Q 49. What are the mathematical notation used to define the complexity of an algorithm. **(PTU, Dec. 2013)**

Ans. The Big O notation is used to define the complexity of an algorithm.

Q 50. Define the terms: static and dynamic data structures. List some of the static and dynamic data structure in C. **(PTU, Dec. 2014)**

Ans. Data structures which have fixed size and have their memory allocated during the time of compilation of the program are called static data structures.

Typically, the elements (nodes) of a static data structure will have consecutive addresses in the memory. An array is an example of the static data structure.

A data structure that can grow and shrink, with memory allocated during the execution of the program is known as a dynamic data structure.

Typically, the elements are stored whenever there is a free space. A linked list is an example of the dynamic data structure.

Q 51. What criteria is used for evaluating the suitability of a particular data structure for a given application. **(PTU, Dec. 2014)**

Ans. By employing a series of evaluation criteria, the suitability of different data structures for different tasks can be explored. The following criteria are :

- **(a) Completeness :** The proportion of entities which can be represented.
- **(b) Robustness :** The ability of the model or structure to handle special circumstances.
- **(c) Efficiency :** The compactness and speed of use ;
- **(d) Versatility :** The ease with which the model or system can be applied to new situations.
- **(e) Ease of generation :** The ease with which raw data can be transformed into a particular structure.
- **(f) Functionality :** The range of operations which may be performed on data in this form.
- **(g) Utility :** ease of use of the data model or structure.

Q 52. What do you mean by the time complexities of an algorithm ? **(PTU, May 2019, 2015)**

Ans. The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the string representing the input. The time complexity of an algorithm is commonly expressed using big O notation, which excludes coefficients and lower order terms.

Q 53. Differentiate between linear and non-linear data structure. (PTU, May 2015)

Ans. Main difference between linear and non-linear data structures lie in the way they organize data elements. In linear data structures, data elements are organized sequentially and therefore they are easy to implement in the computer's memory. In nonlinear data structures, a data element can be attached to several other data elements to represent specific relationships that exist among them. Non-Linear data structure is that if one element can be connected to more than two adjacent element then it is known as non-linear data structure.

Q 54. What are the objectives of studying data structures ? (PTU, Dec. 2015)

Ans. ❑ To identify and create useful mathematical entities and operations to determine what classes of problems can be solved using these entities and operations.

❑ To determine the representation of these abstract entities and to implement the abstract operations on these concrete representation.

Q 55. Why complexity of linear search is of the order of O(n) ? (PTU, May 2016

Ans. Linear search is obtained by next←low. So worst-case access time in O(n) fo linear search.

Q 56. A sorting method with "Big-Oh" complexity O(n log n) spends exact 1 millisecond to sort 1,000 data items. Assuming that time T(n) of sorting n items directly proportional to n log n, that is, T(n) = cn log n, derive a formula for T(n), giv the time T(N) for sorting N items, and estimate how long this method will sort 1,000,0 items. **(PTU, May 20**

Ans. Because processing time is T(n) = cn logn

the constant factor $c = \frac{T(N)}{N \log N}$,

and $T(n) = T(N) \frac{n \log n}{N \log N}$.

Ratio of logarithms of the same base is independent of the base, hence any approp base can be used.

Therefore, n = 1000000 the time is
T (1,000,000) = T (1,000)

$$\frac{1.000000 \log_{10} 1000000}{1000 \log_{10} 1000} = 1.\frac{1000000.6}{1000.3} = 2{,}000 \text{ ms}$$

Q 57. What are non recursive procedures ? **(PTU, Dec**

Ans. Non recursive procedures are the subrouting or functions implement programming language whose implementation does not reference itself.

❑❑❑

Stacks and Queues

Contents

ADT Stack and its Operations : Algorithms and their complexity analysis, Applications of Stacks : Expression Conversion and Evaluation - Corresponding algorithms and complexity analysis. ADT queue, Type of Queue : Simple Queue, Circular Queue, Priority Queue; Operations on each types of Queues : Algorithms and their analysis.

POINTS TO REMEMBER

- Stack is a linear data structure in which insertion and deletion of an element can occur only at one end called the top of the stack.
- Push is the term used to insert an element into a stack.
- Pop is the term used to delete an element from a stack.
- Stack can be traversed from top to bottom or bottom to top.
- In stack, elements can be inserted at any place by shifting pointers.
- The element can be deleted by shifting pointer.
- Stacks can be maintained in memory using two methods :
  (i) Using Array
  (ii) Using linked list
- If operator symbol is placed between two operands, it is called infix expression.
- If operator symbol is placed before its two operands it is called as prefix expression or polish expression.
- Stack words an the principle of LIFO.
- The stack top or pick operation returns the element at the top of the stack without altering the stack.
- Stacks are used to convert infix expression to post fix expression.
- Stacks are used to sort the elements using quick sort.
- Top pointer contains the location of the top element of the stack.
- A queue is a linear data structure in which new elements are inserted at one end and elements are deleted from the other end.

- A queue follows FIFO (First-in, First-out) property in which elements leave the queue in the order in which they enter.
- The enqueue operation is used to insert a new element to the rear of the queue. The newly inserted element becomes the rear.
- The dequeue operation is used to remove an existing element from the front of the queue.
- The front peek operation returns the element from the front of the queue without altering the queue.
- The rear peek operation is used to return the element from the rear of the queue without altering the queue.
- The circular queue is an improved array representation of a queue. It handles the problem when you want to insert a new element in the queue when rear reaches the end of array and there are empty locations at the beginning of the array representing queue.
- In a circular queue, elements are arranged in such a way that the last element is logically followed by the first element.
- In the linked queue, the actual data item is stored in the INFO part of the node and link part of a node contains a pointer that points to the next element of the queue.
- A deque (double-ended queue) is a linear list that allows insertion and deletion at either end i.e. at the front or rear of the queue but not in the middle.
- Insert left, insert right, delete right and delete left are the four basic operations performed on a deque.
- The deque generalizes both the stack and queue. If you restrict yourself to insert left and delete left operations then deque acts as a stack.
- A priority queue is a variation of simple queue in which each element has been assigned a key called priority. Elements are ordered by priority in the sense that elements with the highest priority are removed first.
- In the linked representation of priority queue, all the nodes are sorted according to the priority values of the elements.
- Queues are used in simulation, some sorting and searching techniques, round robin algorithm etc.

QUESTION-ANSWERS

Q 1. Write procedure to POP top element from STACK. **(PTU, May 2005)**

Ans. POP (STACK, TOP, ITEM)

This procedure deletes the top element of STACK and assigns it to variable ITEM.

1. If TOP = 0, then : Print : Underflow and return :
2. Set ITEM : = STACK [TOP].
3. Set TOP : = TOP – 1
4. Return.

Q 2. What data structure operations can be applied to stacks? (PTU, Dec. 2005)

Ans. Data structure operations that can be applied to stacks are :

1. Push, i.e. inserting an item into stack.
2. Pop, i.e. deleting an item from stack.

These operations are performed only on top of the stack.

Q 3. Write a function to PUSH a new item to STACK. (PTU, Dec. 2005)

Ans. PUSH (STACK, TOP, MAX, ITEM)

1. If TOP = MAX, then :
   Print : overflow & Return.
2. Set TOP = TOP + 1
3. Set STACK [TOP] = ITEM
4. Return.

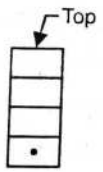
Q 4. What are uses of stacks? (PTU, Dec. 2005)

Ans. 1. Stacks are used to convert infix expression to post fix expression.
2. Stacks are used to evaluate postfix expression.
3. Stacks are used to sort the elements using quicksort.

Q 5. What is stack? (PTU, Dec. 2006 ; May 2006)

Ans. A stack is a list of elements in which an element may be inserted or deleted only at one end called top of the stack.

It is a LIFO system, i.e. last element entered is the first element to delete. Two operations, i.e. push and pop operations are used for insertion and deletion in stacks.



Q 6. Distinguish between stack and queue. (PTU, May 2008, 2007)

Ans.

	Stack		Queue
1.	As stack is a list of elements in which an element may be inserted or deleted only at one end, called, top of the stack.	1.	A queue is a linear list of elements in which deletions can take place only at one end, called, front and insertions can take place at other, called rear end.
2.	It is also known as LIFO, i.e. last In first out.	2.	It is also known as FIFO, i.e. First In First Out.
3.	**e.g.** Stack of coins are above the other.	3.	**e.g.** Queue of people waiting for a bus.

Q 7. Write any two applications of stack. (PTU, May 2008)

OR

List few applications of stack. (PTU, May 2019, 2009)

Ans. Application of Stack : The main applications of STACK are as follows :

1. Evaluation of Postfix expression : The stacks are commonly employed in evaluating of postfix expression.

2. Conversion of infix expression to post fix expression : Stack can be used to convert infix expression into the post fix expression.

Q 8. Write the prefix notation for the expression.

(A + B) * C – (D – E) ^ F (PTU, May 2008)

Ans. (A + B) *C – (D – E) ^ F

= [+AB] * C – [–DE] ^ F

= [* +ABC] – [^ –DEF]

= –*+ ABC ^ – DEF

Q 9. Define ADT and give an example. (PTU, May 2009)

Ans. In computing, an abstract data type or abstract data structure is a mathematical model for a certain class of data structures that have similar behaviour, or for certain data types of one or more programming languages that have similar semantics. An ADT is defined indirectly,only by the operations that may be performed on it and by mathematical constraints on the effects (and possibly cost) of those operations.

For example, an abstract stack data structure could be defined by two operations ; push, that inserts some data item into the structure,and pop, that extracts an item from it;

Example : abstract stack (imperative).

Q 10. Write the postfix notation for the following expression :

(A + B) * C – (D – E) ∧ F (PTU, Dec. 2008)

Ans. (A + B) * C –˙(D – E) ∧ F

The positive notation for the given expression is as follows :

[AB+] * C – [DE–] ^ F

[AB + C*] – [DE – F^]

= AB + C * DE – F^ –

Q 11. What do you understand by polish notation? Explain. (PTU, May 2010)

Ans. Polish notation refers to the notation in which the operator symbol is placed before its operands in order from left to right. If an operand is an operaiton with operands, then the same rules apply.

For example :

(X + Y) * Z = [+XY] *Z = * + XYZ

X + (Y * Z) = X + [*YZ] = +X * YZ

Q 12. Give an example that shows how a stack is used by a computer system. (PTU, May 2010

Ans. Stack is internally used by compiler when we execute any recursive functior When a function calls a function, its arguments, return address and local variables are pushe

onto the stack. Since each function runs in its own environment, it becomes possible for a function to call itself. When the function completes its execution these parameters are popped off from the stack to execute the next nested call.

Q 13. Consider the following post fix expression, P : 5, 6, 2, + , *, 12, 4, /, –. Write the procedure and evaluate this expression. **(PTU, Dec. 2004)**

Ans.

1. Add a right parenthesis at the end of post fix expressions.
   P : 5, 6, 2, +, *, 12, 4, /, –)
2. Scan P from left to right and repeat the following steps for each element of 'P' until right parenthesis is encountered.
3. If an operand is encountered, put it on the stack.

P : | 5 | 6 | 2 |

4. If an operator is encountered, then, remove the top 2 elements, apply operator and place the result back on stack.

∴ P : | 5 | 6 | 2 | + |

P : | 5 | 8 | * | → | 40 |

P : | 40 | 12 | 4 | / |

P : | 40 | 3 | – |

P : | 37 |

∴ Value = 37

Symbols scanned	Stack
5	5
6	5, 6
2	5, 6, 2
+	5, 8
*	40
12	40, 12
4	40, 12, 4
/	40, 3
–	40, 3
)	37

#Q 14. What is a top pointer of a stack? **(PTU, Dec. 2010)**

Ans. Top pointer which contains the location of the top element of the stack ; and a variable MAXSTK which gives the maximum number of elements that can be held by the stack. The condition TOP = 0 or TOP = NULL will indicate that the stack is empty.

Q 15. Consider the postfix expression :
P : 12, 7, 3, –, /, 2, 1, 5, +, *, +.
Write the procedure and evaluate the expression. **(PTU, Dec. 2005)**

Ans.

1. Add a right parenthesis at the end of post fix expression.
2. Scan P from left to right and repeat to following steps for each element of 'P' until right parenthesis is encountered.
3. If an operand is encountered, put it on stack.
4. If an operator is encountered, then remove the top two elements, apply operator and place the result back on stack.

Symbol	Stack
12	12
7	12, 7
3	12, 7, 3
–	12, 4
/	3
2	3, 2
1	3, 2, 1
5	3, 2, 1, 5
+	3, 2, 6
*	3, 12
+	15
)	15

∴ Value = 15.

Q 16. Consider the following infix expression :
((A + B) * D) ↑ (E – F).
Write the expression and convert into the equivalent postfix expression.

Ans. **(PTU, May 20**

Symbol	Stack	P
(	((	
(	(((	
A	(((	A
+	((+	A
B	((+	AB

)	((	AB +
*	((*	AB +
D	((*	AB + D
)	(	AB + D*
↑	(↑	AB + D*
(	(↑ (	AB + D*
E	(↑ (	AB + D*E
–	(↑ (–	AB + D*E
F	(↑ (–	AB + D*EF
)	(↑	AB + D* EF –
)		AB + D* EF – ↑

Hence, postfix expression.

P : AB + D * EF – ↑

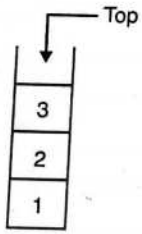
Q 17. Explain concept of stack along with its applications. (PTU, Dec. 2006)

Ans. A stack is a list of elements in which an element may be inserted or deleted only at one end, called top of the stack. Here, the elements are removed from a stack in reverse order of that in which they were inserted into the stack. Stack is also called a Last-in First-out (LIFO) structure.

Two basic operations associated with stacks :

(i) "Push" is used to insert an element into stack.

(ii) "Pop" is the term used to delete an element from a stack.



Application of stacks :

1. Evaluation of postfix expressions : Stacks are commonly employed in evaluation of postfix expression P. Various steps involved are :

(i) Add a right parenthesis at the end of postfix expression.

(ii) Scan 'P' from left to right and repeat the following steps for each element until right parenthesis is encountered.

(iii) If an operand is encountered, put it on stack.

(iv) If an operator is encountered, then remove the top two elements, apply the operator and place the result back on stack.

2. Conversion of infix expresssion to postfix : Stacks are also used to convert an infix expression to postfix expression.

e.g. If an infix expression is :

5* (6 + 2) – 12/4

Symbol	Stack	P
5	(	5
*	(*	5
6	(* (	56
+	(* (+	562
2	(* (+	562
,	(*	562+
–	(–	562 + *
12	(–	562 + *12
/	(–/	562 + * 12
4	(–/	562 + * 124
,		562 + * 124/-

∴ Equivalent postfix expression is

562 + * 124/-

3. Quicksort : Quicksort is an algorithm which uses divide and conquer technique sort the elements. Stacks are used for sorting. In this algo, we position the first element applying some iteration at its correct position and divide the complete list in two parts and using stack, we keep track of these two lists.

Q 18. Write a program for implement stack using arrays. (PTU, May 20

Ans.

```
#include <iostream.h>
#include <process.h>
void push (int a [ ], int n) ;
void pop (int a [ ] );
void display (int a [ ], int) ;
int top = –1;
void main ( )
{
int a [51], n ;
char ch = 'y';
while (ch = = 'y' ¦¦ ch == 'Y')
{
cout << "Insert element into stack",
cin >> n ;
push (a, n) ;
cout << "Want to insert more?" ;
cin >> ch ;
```

```
    }
    do
    {
    cout <<"Current stack is" ;
    display (a, top) ;
    cout <<"want to delete item?" ;
    cin >> ch;
    if (ch = = 'y' || ch = = 'Y')
    pop (a) ;
    else
    ch = 'n' ;
    } while (ch = = 'y' || ch = 'Y') ;
    cout << "stack is" ;
    display (a, top) ;
    }
    void push (int a [ ], int n)
    {
    if (top = = 51)
    {
    cout <<"overflow" ;
    exit (0) ;
    }
    top = top + 1 ;
    a [top] = n ;
    }
    void pop (int a [ ])
    {   if (top = = –1)
        {  cout <<"under flow" ;
           exit (0) ;
        }
        int them = a [top] ;
        cout <<" Item deleted : "<< item ;
        top = top – 1 ;
}
void display (int c [ ] , int top)
{
for (int i = top ; i > = 0 ; i – –)
cout <<" "<< a [i] ;
}
```


Q 19. What is the polish notation representation of the following expression?

(A * (b + C) + (b/d)* a + z

(a + (b + c* (d + e))) + f **(PTU, May 2010)**

Ans.

1. (A * (b + C)) + (b/d) * a + z
   ≈ (A* [+bC]) + [/bd] * a + z
   ≈ [* A + bC] + [*/bda] + z
   ≈ [+*A + bC*/bd a] + Z
   ≈ ++ *A + bC*/bdaz

Hence, the expression in polish notation is :

++*A + bC */ bdaz

2. (a + (b + C* (d + e))) + f
   ≈ (a + (b + C* [+de])) + f
   ≈ (a + (b + [*C + de])) + f
   ≈ (a + [+b*c + de]) + f
   ≈ [+a + b * c + de] + f
   ≈ ++a + b *c + def

Hence, the expression in polish notation is :

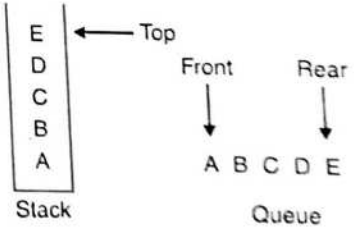
++ a + b*c + def

Q 20. Compare stacks and queues and discuss their limitations.

(PTU, Dec. 2007)

Ans. Two of the more common data objects found in computer algorithms are stacks and queues. Both of these objects are special cases of the more general data object, an ordered list.

A stack is an odered list in which all insertions and deletions are made at one end called the top. A queue is an ordered list in which all insertions take place at one end, th rear, while all deletions take place at the other end, the front. Given a stack S = (a [1], a [2 ..... a [n]) then we say that a1 is the bottom most element and element a [i]) is on top element a [i – 1], 1 < i < = n. When viewed as a queue with a [n] as the rear element one sa that a [i + 1] is behind a [i], 1 < i < =n.



as Last In First Out (LIFO) lists. The restrictions on queue imply
A is the first letter to be removed, and queues are known as First In First Out (FIFO) lists.
element which is inserted into the queue will be the first one to be removed. Thus
Note that the data object queue as defined here need not necessarily correspond to the
mathematical concept of queue in which the insert/delete rules may be different.

Q 21. What is the post fix and prefix representation of the following expression (A * (b + C) + (b/d)* a + z **(PTU, Dec. 2010)**

Ans. Step 1. Fully parenthesize the expression,

I = ((A * (b + c)) + ((b/d) * (a + z)))

Step 2. Convert to postfix expression.

I = ((A * (bc+)) + ((b/d) * (a + z)))
I = ((A * (bc+)) + ((bd/) * (a + z)))
I = ((A * (bc+)) + ((bd/) * (az+)))
I = ((A (bc+*) + ((bd/) * (az+)))
I = ((A (bc+)*) + ((bd/) (az +)*))
I = ((A (bc+)*) ((bd/) (az+)*) +)

Q 22. Give application of stacks, queues and trees. Also list the operations possible on stack. **(PTU, May 2004)**

Ans. The application of stacks are as follow :

1. Evaluation of post fix expression :

Stacks are commonly employed in evaluation of a postfix expression.

2. Conversion of infix expression to postfix expression :

Stacks are used to convert an infix expression to postfix expression.

e.g. It infix expression is :

5* (6 + 2) – 12/4

Then, using stacks, the postfix expression is :

562 + *124/-

3. Quick sort : Quicksort is an algorithm which uses divide and conquer technique to t the elements. Stacks are used for sorting. Here, we position the Ist element by applying ne iteration at its correct position and divide the complete list in 2 parts and by using stack, keep track of these 2 lists.

The application of queues are as follow :

1. Breadth First Search (BFS) : The main application of queues is in traversing the using Breadth First Search (BFS), i.e. to find path between and 2 vertices in graph.

2. Priority queues : Priority queues are used to assign priority, s.t. an element of er priority is processed before any element of lower priority.

$L_E = L_I + 2 \cdot n$

Where, $n \rightarrow$ no. of internal nodes
$L_E \rightarrow$ length of external nodes
$L_J \rightarrow$ length of internal nodes.

2. Huffman's algorithm is used to find a tree with minimum weighted path. Operations possible on stack are :

1. **Push :** PUSH operation is used to insert an element at the top of the stack.
PUSH (STACK, TOP, MAX, ITEM)
1. If TOP = MAX, then :
Print : overflow and return
2. Set TOP = TOP + 1
3. Set STACK [TOP] = ITEM
4. Return.

2. **POP :** Pop operation is used to delete an item for Top of the stack.
POP (STACK, TOP)
1. If TOP = NULL, then :
write : underflow and Exit
2. Set ITEM = STACK [TOP]
3. Set TOP = TOP – 1
4. Exit.

Q 23. Consider the following mathematical expression written in postfix : 12, 9, 3, –/, 3, 2, 5, +, *, +.
Write the procedure and evaluate the expression. **(PTU, May 2005**

Ans.

Symbols scanned	Stack
12	12
9	12, 9
3	12, 9, 3
–	12, 6
/	2
3	2, 3
2	2, 3, 2
5	2, 3, 2, 5
+	2, 3, 7
*	2, 21
+	23
)	

∴ Value = 23.

Q 24. Explain about infix, prefix and postfix and write an algorithm to convert infix to postfix expression. **(PTU, May 2006)**

Ans. For most common arithmetic operation, operator symbol is placed between its two operands. For e.g. A + B, C – D, E * F etc.

This is called infix notation.

(i) Polish notation, i.e. prefix notation refers to notation in which the operator symbol is placed before its two operands. **e.g.**
+AB, –CD, *EF etc.

(ii) Reverse polish notation, i.e., postfix expression is an analogous notation in which operator symbol is placed after its two operands. **e.g.**
AB+, CD–, EF*, etc.

Polish (Q, P)

Suppose 'Q' is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression P.

1. Push "("onto STACK, and add")" to end of Q.
2. Scan 'Q' from left to right and repeat steps 3 to 6 for each element of 'Q' until STACK is empty.
3. If an operand is encountered, add it to P.
4. If a left parenthesis is encountered, push it onto stack.
5. If an operator (X) is encountered, then (a) Repeatedly pop from STACK and add to P each operator which has same precedence or higher precedence then X (b) Add X to STACK.
6. If a right parenthesis is encountered, then :
   (a) Repeatedly pop from STACK and add to 'P' each operator until a left parenthesis is encountered.
   (b) Remove the left parenthesis.
7. Exit.

Q 25. What are the various operations possible on stacks? Explain the algorithm for each of them. **(PTU, May 2019 ; Dec. 2010)**

Ans. Stacks : A stack is a list of elements in which an element may be inserted or deleted only at one end, called the top of the stack. This means, in particular, that elements are removed from a stack in the reverse order of that in which they were inserted into the stack.

Special terminology is used for two basic operations associated with stacks :

(a) "Push" is the term used to insert an element into a stack.

(b) "Pop" is the term used to delete an element from a stack.

PUSH (STACK, TOP, MAXSTK, ITEM)

This procedure pushes an ITEM onto a stack.

1. [stack already filled?]
   If TOP = MAXSTK, then : Print : OVERFLOW and Return.
2. Set TOP = TOP + 1 [Increases Top by 1]
3. Set STACK (TOP) = ITEM [Inserts ITEM in new TOP position]
4. Return.

POP (STACK, TOP, ITEM)

This procedure deletes the top element of STACK and assigns it to the variable ITEM.

1. [Stack has an item to be removed?]
   If TOP = 0, then : Print : UNDERFLOW, and Return.
2. Set ITEM = STACK [TOP]. [Assigns TOP element to ITEM]
3. Set TOP = TOP – 1 . [Decreases Top by 1]
4. Return.

Push an Item into Stack :

```
/ * C implementation of pop algorithm *.
int pop (int stack [ ], int stack – top) / * stack – top is a local variable indicating top o
stack * /
{int item ;
   if (stack_top <10)
   {   print f ("\ n stack underflow") ;
       return – 1 ;
   }
else
   {   item = stack [stack – top] ;
       stack – top ..... ;
       top = stack – top ; / * update global variable
       top * /
       return item ;
   }
}
```

POP from stack :

```
/ * C implementation of push algorithm * /
int push (int stack [ ], int stack – top, int maxstack, int item)
{  if (stack – top = = maxstack)
   {   print f (" \n stack overflow") ;
       return – 1 ;
   }
else  {   stack – top ++ ;
          stack [stack – top] = item ;
          top = stack – top ; /* update global variable top* /
          return 0 ;
       }
}
```

(PTU, May 2009)

The postfix expression is :

((a + b) + c * (d + e) + f) * (g + h)

= ((ab+) + C * (de+) + f) * (gh+)

= ((ab+) + (cde + *) + f) * (gh+)

= ((ab + cde + * +) + f) * (gh+)

= (ab + cde + * + f +) * (gh+)

= (ab + cde + * + f + gh + *)

The prefix expression is :

((a + b) + c * (d + e) + f) * (g + h)

= ((+ab) + c * (+de) + f) * (+gh)

= ((+ab) + (* c + de) + f) * (+gh)

= ((++ ab * c + de) + f) * (+gh)

= (+++ ab * c + def) * (+gh)

= (* +++ ab * c + def + gh)

Q 27. Write an algorithm to convert infix expression to postfix expression. Give example and apply algorithm. (PTU, May 2007)

Ans. POLISH (Q, P)

1. Push "("onto STACK & add")" to end of Q.
2. Scan 'Q' from left to right and repeat steps 3 to 6 for each element of until STACK is empty.
3. If an operand is encountered, add it Ⓧ to P.
4. If a left parenthesis is encountered, push it onto stack.
5. If an operator Ⓧ is encountered, then :
   (i) Repeatedly pop from STACK and add to P each operator which has same or higher procedence than Ⓧ
   (ii) Add Ⓧ to stack.
6. If a right parenthesis is encountered, then :
   (i) Repeatedly pop from STACK and add to P each operator until a left parenthesis is encountered.
   (ii) Remove the left parenthesis.
7. Exit.

Symbol		
(	((	
(	(((	
A	(((	A
+	(((+	A
B	(((+	AB
)	((	AB+
*	((*	AB+
D	((*	AB+D
)	(	AB+D*
↑	(↑	AB+D*
(	(↑ (	AB+D*
E	(↑ (	AB + D*E
–	(↑(–	AB+D*E
F	(↑(–	AB + D* EF
)	(↑	AB + D*EF –
)		AB + D*EF–↑

∴ Post fix expressison is :

AB + D * EF – ↑

Q 28. Convert the following infix expression to postfix. (PTU, Dec. 2007)

A + (B * C – (D / E ↑ F) * G) *H

Ans. Q : A + (B * C – (D / E ↑ F) * G) *H

A	+	(	B	*	C	–	(	D	/	E	↑	F	)	*	G	)	*	H	)
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)	(16)	(17)	(18)	(19)	(20)

The elements of Q have now been labelled from left to right for easy reference. The diagram shows status of STACK and of the string P as each element of Q is scanned.

1. Each operand is simply added to P and does not change STACK.
2. The subtraction operator (–) in row 7 sends * from STACK to P before it (–) is pushed onto STACK.
3. The right paranthesis in row 14 sends ↑ and then / from STACK to P, and then removes the left parenthesis from TOP of the STACK.
4. The right paranthesis in row 20 sends * and thus + from STACK to P and the removed left paranthesis from TOP of the STACK.

After step 20 is executed, the STACK is empty one

P : ABC * DEF ↑ / G * – H * +

which is requierd postfix equivalent of Q.

Symbol scanned	STACK	Expression P
(1) A	(	A
(2) +	(+	A
(3) (	(+ (	A
(4) B	(+ (*	A B
(5) *	(+ (*	A B C
(6) C	(+ (–	A B C *
(7) –	(+ (– (	A B C *
(8) (	(+ (– (	A B C * D
(9) D	(+ (– (	A B C * D
(10) /	(+ (– (/	A B C * D E
(11) E	(+ (– (/	A B C * D E
(12) ↑	(+ (– (/ ↑	A B C * D E F
(13) F	(+ (– (/ ↑	A B C * D E F ↑ /
(14))	(+ (–	A B C * D E F ↑ /
(15) *	(+ (– *	A B C * D E F ↑ / G
(16) G	(+ (– *	A B C * D E F ↑ / G * –
(17))	(+	A B C * D E F ↑ / G * –
(18) *	(+ *	A B C * D E F ↑ / G * – H
(19) H	(+ *	A B C * D E F ↑ / G * – H
(20))		* +

Q 29. Write the postfix notation for the following expression :
(A + B) * C – (D – E) ^ F. **(PTU, May 2011)**

Ans. Postfix notation of (A + B) * C – (D – E) AF is equal to AB + C* DE – FA

Q 30. Write any two applications of stack. **(PTU, May 2011)**

Ans. The two applications of stack are :

1. Towers of hanoi
2. Recursion
3. Conversion of expression.

Q 31. Write down the algorithms for various operations possible on stacks.
(PTU, May 2011)

Ans. Stacks : Stacks is a linear structure which consist of list of data items in which insertions and deletion are made only at one end called the top of the stack. Also, arc the data item or elements can be removed or added only from the top, it is defined as LAST-IN-FIRST-OUT (LIFO) structure. The terms used in operation associated with stack are PUSH and POP.

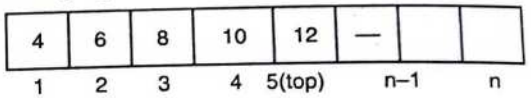
1. Push : Push is used to insert an element into a stack.

2. Pop : Pop is used to delete an element from a stack.

The stack structure functions in the same manner as a spring loaded stack of plates or ray function in canteen. When we add plates by pushing the top of the stack and to remove plate from the top of the stack the spring causes the next plate to pop up.

Representation of Stacks :

e.g. If the following 4 elements are pushed into empty stack
, 6, 8, 10, 12
then the stack is written as : 4, 6, 8, 10, 12
where right most element is 12 which is at the top of the stack.
It shown is the following fig.



Algorithm :

Step 1. If top = MAX
Print "Overflow" and return
Step 2. Set top = top + 1 Shows increment of value.
Step 3. Set stack (top) = D It adds data in the new top position
Step 4. Return

Algorithm for the pop (Deletion of element from the STACK)

Step 1. If top = 0
Print "Underflow" and return – It shows that stack has no value to remove
Step 2. Set D = Stack (Pop) It assign the top element of the stack to DATA
Step 3. Set top = top = 1 Increases the value of pointer variable top.
Step 4. Return.

Q 32. What are the various operations an stack?

Ans. There are two operations can be applied an stack :

(a) PUSH operation
(b) POP operation

(a) Push Operation : Push means to insert a new item at top of stack. In executing the procedure PUSH, we must first check whether there is a space in stack to insert new item or not. If there is a space then we insert new element otherwise the condition of overflow occurs.

(b) Pop Operation : Pop means to delete the top element from stack. In Pop operation one must first test whether there is an element in the stack to be deleted, if not then we have the condition known as underflow.

Q 33. Convert the infix notation 1 + (2*3 – (4/5 ↑ 6)*7) *8 into postfix notation.
(PTU, Dec. 201

Ans. Q is the infix notation and transform it into its equivalent postfix expression P.
First we push "(" onto stack, and then we add ")" to the end of Q to obtain.
Q : 1 + (2*3 – (4/5 ↑ 6)*7) *8)
(1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13) (14) (15) (16) (17) (18) (19) (20)
The elements of Q have now labeled from left to right. Table shows the status of sta and of the string P as each element of Q is scanned.
After step 20 is executed, the stack is empty and.
P : 123* 456↑ /7 * – 8* +

Which is the required postfix equivalent of Q.

	Symbol Scanned	Stack	Expression 'P'
(1)	1	(	1
(2)	+	(+	1
(3)	(	(+ (	1
(4)	2	(+ (	12
(5)	*	(+ (*	12
(6)	3	(+ (*	123
(7)	–	(+ (–	123*
(8)	(	(+ (–(	123*
(9)	4	(+ (– (	123*4
(10)	/	(+ (– (/	123*4
(11)	5	(+ (– (/	123 * 45
(12)	↑	(+ (– (/ ↑	123*45
(13)	6	(+ (– (/ ↑	123*456
(14)	)	(+ (–	123*456↑/
(15)	*	(+ (– *	123*456↑/
(16)	7	(+ (– *	123*456 ↑/7
(17)	)	(+	123*456↑ 7*–
(18)	*	(+ *	123*456 ↑ /7* –
(19)	8	(+ *	123*456↑/7*–8
(20)	)	*	123*456↑/7*–8*+

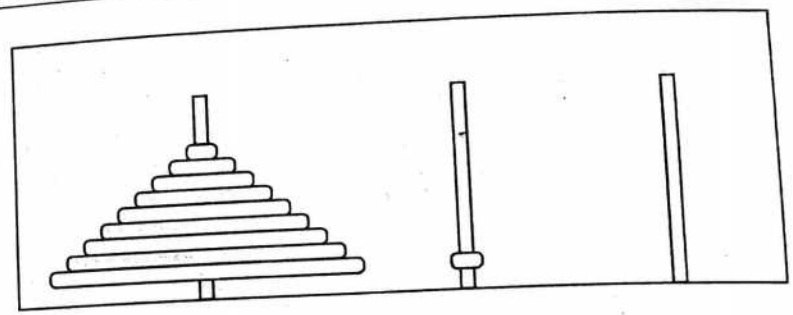
Q 34. Briefly explain Tower of Hanoi problem. **(PTU, Dec. 2011)**

Ans. The Tower of Hanoi puzzle was invented by the French mathematician Edouard Lucas in 1883. We are given a tower of eight disks (initially four in the applet below), initially stacked in increasing size on one of three pegs. The objective is to transfer the entire tower to one of the other pegs (the rightmost one in the applet below), moving only one disk at a time and never a larger one onto a smaller.

The puzzle is well known to students of Computer Science since it appears in virtually any introductory text on data structures or algorithms. Its solution touches on two important topics :

- Recursive functions and stacks
- Recurrence relations

The applet has several controls that allow one to select the number of disks and observe the soluton in a Fast or Slow manner. To solve the puzzle drag disks fron one peg to another following the rules. You can drop a disk on to a peg when its center is sufficiently close to the center of the peg. The applet expects you to move disks from the leftmost peg to the rightmost peg.



Q 35. List types of queues. **(PTU, May 2004)**

Ans. The queues are of the types : simple queue, circular queue, dequeue and priority queues.

Q 36. A deque is maintained by a circular array with N memory cells. When an element is added to deque, how is LEFTand RIGHT 2 variables to indicate 2 ends of a deque changed? **(PTU, Dec. 2004)**

Ans. If the element is added on the left, then LEFT is decreased by 1 (mod N). On the other hand, if the element is added on the right, then RIGHT is increased by 1 (mod N).

Q 37. What are limitations of queues? **(PTU, Dec. 2005)**

Ans. 1. The major drawbacks is that one cannot have access to middle element, i.e. insertion and deletion can take place only at the two ends called front end and rear end.

2. In this case, last element in a queue will be last element out of queue i.e. if we want to access only last element, we have to transverse all the elements before it.

Q 38. What is priority queue. **(PTU, Dec. 2009 ; May 2014, 2006)**

Ans. Priority queue is the collection of elements such that each element has been assigned a priority and the order in which the elements are deleted and processed comes from following rules :

1. An element of higher priority is processed before any element of lower priority.
2. Two elements with same priority are processes according to order in which they were added to queue.

Q 39. Write down application of queues. **(PTU, May 2007)**

Ans. 1. The main application of queues is that it is used in traversing the graph using Breadth First Search (BFS) i.e. to find shortest path between two vertices.

2. Priority queues are used to assign priority s.t., an element of higher priority is processed before any element of lower priority.

Q 40. What are priority queues? How they are implemented? **(PTU, Dec. 2007**

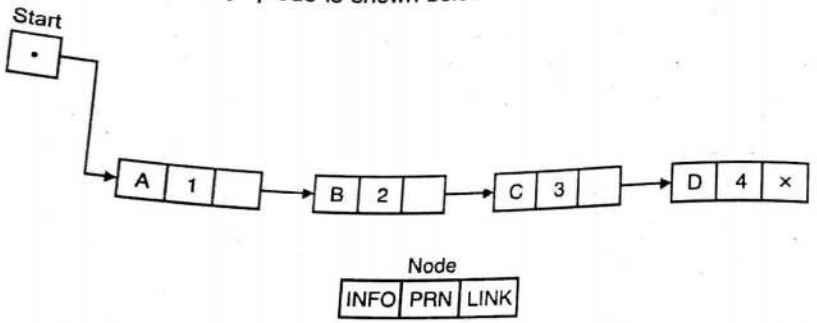
Ans. Priority Queues : A priority queue is a collection of elements such that the orde in which elements are deleted and processed comes from the following rules :

1. An element of higher priority is processed before any element of lower priority.

2. Two elements with same priority are processed according to order in which they are added to the queue.

There are various ways of maintaining a priority queue in memory : one uses a one-way list and other uses multiple queues. The ease or difficulty in adding elements to or deleting them from a priority queue clearly depends on representation that one choosen.

Example : The priority queue is shown below :



Priority queue with 4 node

Q 41. What is a queue? Discuss operations on queue. **(PTU, Dec. 2007)**

Ans. Queue : A queue is a linear data structure in which new elements are inserted at one end and elements are deleted from the other end.

Operation on Queue : A number of operation can be performed on a queue. These functions are :

Enqueue : Used to insert a new element to rear of queue.

Dequeue : Used to remove an existing element to front of queue.

Create : Used to create an empty queue.

Destroy : Used to delete all the element of the queue.

Isempty : Check and report is queue empty or not.

Isfull : Check and report is queue full or not.

Also perform front peek and rear peek.

Q 42. What are Dequeues? How are they maintained in memory? (PTU, May 2005)

Ans. Dequeue is a linear list in which elements can be added or removed at either end, but not in the middle.

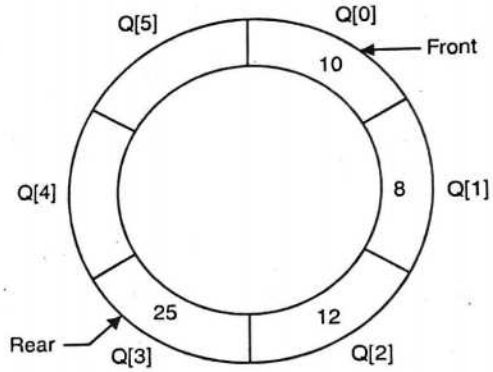
Dequeues is maintained by a circular queue with pointers 'LEFT' and 'RIGHT', which points to 2 ends of a dequeue. The variations of dequeue are :

(i) I/P restricted dequeue : In this type, insertion can only be possible at one end, but deletion can be done on both ends.

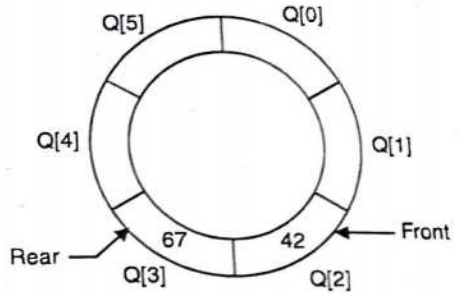
(ii) O/P restricted dequeue : In this type, deletion is possible only at one end, but insertion can be done at both ends.

Q 43. What is a circular queue? **(PTU, May 2010)**

Ans. In circular queues the elements are represented in a circular fashion so that last node points back to the first node. In circular queue, the insertion of a new node or element is done at the very first location of the queue if the last location at the queue is full. Suppose Q is a queue array of 6 elements. Insertion and deletion operation can be performed on circular queue. The following figures illustrate the insertion and deletion operations.



A circular queue after inserting 10, 8, 12, 25



A circular queue after deletion of 10, 8

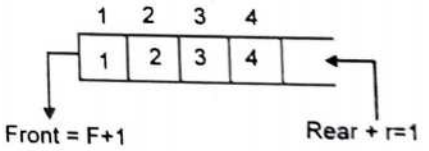
Q 44. What are queues? Compare with dequeues. How are they represented in memory? **(PTU, Dec. 2004)**

OR

How queues are represented in memory? **(PTU, May 2019 ; Dec. 2008)**

Ans. A queue is a linear list of elements in which deletions can take place only at one end called the front and insertion can take place only at the other end called the rear end.

Queues are also called FIFO, since Ist element in a queue will be Ist element out of queue.



With every insertion, rear is incremented by 1. With every deletion, front is incremented by 1.

Dequeues : It is a linear list in which elements can be added or removed at either end but not in the middle. Dequeue is maintained by circular array with pointers 'LEFT' and 'RIGHT' which points to the two ends of a deque. The variations of dequeue are :

(i) Input restricted dequeues : In this type, insertion can only be possible at one end but deletion can be done on both the ends.

(ii) O/P restricted dequeues : In this type of dequeues, deletion is possible only at one end, but insertion can be done at both the ends.

Q 45. What are priority queues? Explain how are priority queues represented by using arrays? **(PTU, Dec. 2005)**

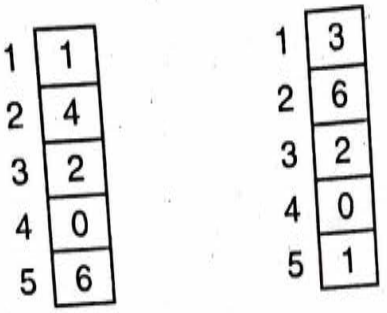
Ans. Priority queue is the collection of elements such that each element has been assigned a priority and the order in which the elements are deleted and processed comes from following rules :

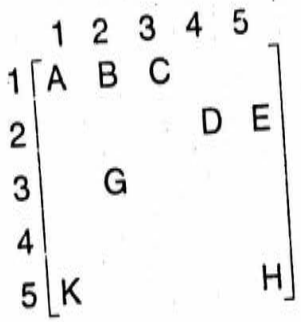
1. An element of higher priority is processed before any element of lower priority.
2. Two elements with same priority are processed according to order in which they were added to the queue.

Array representation

It uses separate queue for each level of priority. Each such queue will appear in its own circular array and must have its own pair of points FRONT and REAR.

e.g. **FRONT** **REAR**





Q 46. Write suitable routines to perform insertion and deletion operations in a queue. **(PTU, May 2009)**

Ans. The operations of insertion and deletion in a queue as follows :

An algorithm for insertion of an element in a queue.

Q INSERT(QUEUE, N, FRONT, REAR, ITEM)

Step 1: [Queue alread filled?]
If FRONT = 1 and REAR = N or if FRONT = REAR + 1, then :
Write : OVERFLOW and Return

Step 2 : [Find new value of REAR]
If FRONT = NULL then [queue initially empty]
Set FRONT = 1 and REAR = 1

Else if REAR = N then
Set REAR = 1
Else
Set REAR = REAR +1
[End of If data structure]

Step 3 : Set QUEUE [REAR] = ITEM This insert new line

Step 4 : Return.

An algorithm for deletion from queue

QUEUE (QUEUE, N, FRONT, REAR, ITEM)

Step 1: [Queue already empty 1]
If FRONT = NULL the write UNDERFLOW and Return.

Step 2 : Set ITEM = QUEUE [FRONT]

Step 3 : [Find new value of FRONT]
If FRONT = REAR then
Set FRONT = NULL and REAR = NULL
Else if FRONT = N then
set FRONT = 1
Else
Set FRONT = FRONT + 1
[End of If structure]

Step 4 : Return.

Q 47. What are the various operations possible on queue? Explain the algorithm for each of them. **(PTU, May 2010)**

Ans. A queue is logically a first-in first-out (FIFO) linear data structure. It is a homogeneous collection of elements in which new elements are called at one end called rear, and the existing elements are deleted from other end called front.

The basic operations that can be performed on queue are :

1. Insert (add) an element to the queue.
2. Delete (remove) an element from a queue.

Let Q be the array of some specified size say size.

(a) Following in an algorithm for inserting an element into the queue :

1. Initialize front = 0, rear = –1
2. Input the value to be inserted and assign to variable "data".
3. If (Rear > = SIZE)
   (a) Display "Queue overflow"
   (b) Exit.
4. Else
   (a) Rear = Rear + 1
5. Q [Rear] = data
6. Exit.

(b) Following is an algorithm for deleting an element from queue

1. If (Rear < Front)
   (a) Front = 0, rear = −1
   (b) Display "The queue is empty"
   (c) Exit.
2. Else
   Data = Q [Front]
3. Front = Front + 1
4. Exit.

Q 48. Explain the procedure for insertion of an item into a queue. **(PTU, May 2005)**

Ans. Suppose we want to insert an element ITEM into a queue at the time the queue does occupy the last part of the array, i.e. when REAR = N.

As the array QUEUE is circular, i.e. QUEUE [i] comes after QUEUE [N] in aray with this assumption, we insert ITEM into queue by assigning ITEM to QUEUE [1]. Thus, instead of increasing REAR to N + 1, we reset REAR = 1

QUEUE [REAR] = ITEM

Now, whenever an element is added to queue, the value of REAR is increased by 1, i.e.

REAR = REAR + 1

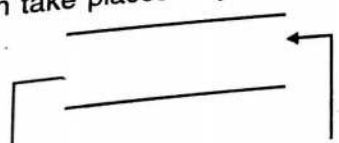
The condition FRONT = NULL indicates that queues is empty.

INSQUEUE (FRONT, REAR, QUEUE)

1. If FRONT = 1, REAR = N, then :
   Write : overflow & Exit.
2. If FRONT = REAR = NULL
   Set FRONT = REAR = 1.
   ELSE IF REAR = N & FRONT ≠ 1,
   Set REAR = 1
   ELSE
   Set REAR = REAR + 1
3. Set QUEUE [REAR] = ITEM
4. Exit.

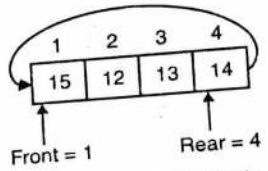
Q 49. Explain various types of queues with examples and write an algorithm to mplement circular queue. **(PTU, May 2006)**

Ans. A queue is a linear list of elements in which deletions can take place only at one nd called front and insertion can take places only at other end called rear end.



Types of queues :

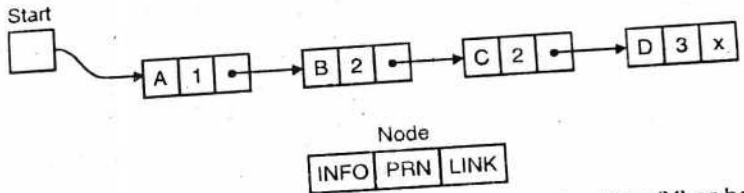
1. Circular queue : These are used to better utilize the memory location. In this case, Q [1] comes after Q [N], i.e. last element. If FRONT = N and element is deleted, we reset FRONT = 1 instead of increasing FRONT = N + 1



If Front = Null and Rear = Null, their, queue is empty. Suppose we have only 1 element, then, Front = Rear ≠ Null.

(ii) Priority Queues : Priority queue is the collection of elements s.t. each element has been assigned a priority and the order in which elements are deleted and processed comes from following rules.

1. An element of higher priority is processed before any element of lower priority.
2. Two elements with same priority are processed according to order in which they were added to queue.



A node 'X' preceeds a node 'Y' when 'X' has priority greater than 'Y' or both have san priority, but 'X' was added to list before 'Y'.

3. Dequeues : It is a linear list in which elements can be added or removed at eith end, but not in the middle. Dequeue is maintained by a circular queue with pointers 'LEF and 'RIGHT' which points to two ends of dequeue. These are of two types :

1. I/P restricted dequeue
2. O/P restricted dequeue.

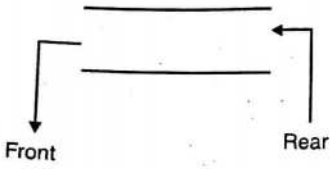
QINS (Q, F, R, ITEM)

1. If F = 1 and R = N, then :
   write : overflow and Return
2. If F = NULL then :
   Set F = 1 and R = 1
   Else if R = N, then :
   Set R = 1 [concept of circular array]
   Else
   Set R = R + 1

3. Set Q [R] = ITEM
4. Return.

Q 50. Define queue. How queues are implemented using double link list? **(PTU, May 2006)**

Ans. A queue is a linear list of elements in which deletions can take place only at one end called front and insertion can take place only at other end called rear end.



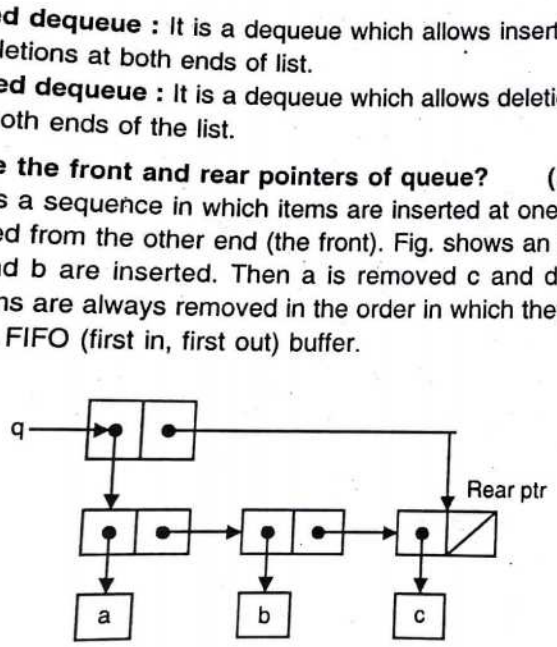
Queues implemented using Double link list are called dequeues. A dequeue is a linear list in which elements can be added or removed at either end but not in middle. Dequeue is maintained in memory by a circular array with pointers LEFT and RIGHT, which point to two ends of dequeue. Here DEQUE [1] comes after DEQUE [N]. There are two variations of a dequeue.

1. I/P restricted dequeue : It is a dequeue which allows insertions at only one end of the list but allows deletions at both ends of list.

2. O/P restricted dequeue : It is a dequeue which allows deletions at only one end but allows insertions at both ends of the list.

Q 51. What are the front and rear pointers of queue? **(PTU, May 2013, 2011)**

Ans. A queue is a sequence in which items are inserted at one end (called the rear of the queue) and deleted from the other end (the front). Fig. shows an initially empty queue in which the items a and b are inserted. Then a is removed c and d are inserted and b is removed.Because items are always removed in the order in which they are inserted, a queue is sometimes called a FIFO (first in, first out) buffer.



Resulting Queue

Operation
(define q (move-queue)
(insert – queue ! q'a) a
(insert – queue ! q' b) ab

Queue operations

(delete – queue ! q) b
(insert – queue ! q' c) bc
(insert – queue ! q' d) bcd
(delete – queue ! q) cd

Q 52. Define queue. What are its various types?

Ans. A queue is an ordered collection of items from which items may be deleted at one end (called the front of the queue) and into which items may be inserted at the other end (called the rear of the queue).

Types of Queue : There are four types of queue :

1. Normal Queue/Linear Queue
2. Circular Queue
3. De-Queue
4. Priority Queue.

Q 53. Explain the various features of queue.

Ans. 1. A list structure will two access points called the front and rear.
2. All insertions occur at the rear and deletions occurs at the front.
3. Varying length.
4. Homogeneous components.
5. It has a (FIFO) first. In, first-out characteristic.

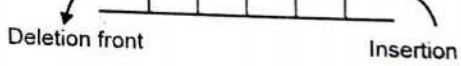
Q 54. How queues can be represented?

Ans. The queues can be represented or stored in computer memory by two ways :

1. By using arrays
2. By using linked list.

Q 55. Explain linear queue.

Ans. In linear queue no circular operation is performed that is last position is full a first cell is empty even then first position will not be occupied by an element. In linear que the overflow condition occurs when FRONT = 1 and REAR = N.



Q 56. What are the various operations performed on circular queue?

Ans. The following operations are performed an circular queue :

1. Create operation
2. Enqueue operation/insertion
3. Dequeue operation/deletion.

Q 57. Write insertion and deletion algorithms an linear queue.

Ans. Insertion/Enqueue Algorithm :

Enqueue (queue, item)

1. Check whether the queue is full or not. If the queue is empty then flash an error message
"Queue Overflow" and
goto step 5, otherwise go to step 2.
2. Increment the value of 'rear' as
rear = rear + 1
3. Store the 'item' at rear postion as queue [rear] = item
4. Set front = 0 only when (front = = −1)
5. Exit.

Deletion/Dequeue Algorithm :

Dequeue (Queue)

Here 'queue' contains the base address of the array queue.

1. Check whether the queue is empty or not. If the queue is empty then flash an error message "Queue underflow" and go to step 4, otherwise go to step 2.
2. Access the front element as item = queue [front]
3. Reset the value of front and rear if (front = rear) ; otherwise increment the value of front = front + 1
4. Exit.

Q 58. What are queues and their applications? Write an algorithm to demonstrate insertion of a node in a queue. **(PTU, May 2019 ; Dec. 2011)**

Ans. Queue : A queue is a linear data structure in which new elements are inserted at one end and elements are deleted from the other end.

Applications :

1. The main application of queues is that it is used in traversing the graph using Breadth First Search (BFS) i.e. to find shortest path between two vertices.

2. Priority queues are used to assign priority s.t., an element of higher priority is processed before any element of lower priority.

An algorithm for insertion of an element in a queue.

Q INSERT(QUEUE, N, FRONT, REAR, ITEM)

Step 1: [Queue alread filled?]
If FRONT = 1 and REAR = N or if FRONT = REAR + 1, then :
Write : OVERFLOW and Return

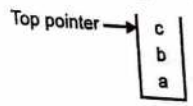
Step 2 : [Find new value of REAR]
If FRONT = NULL then [queue initially empty]
Set FRONT = 1 and REAR = 1
Else if REAR = N then
Set REAR = 1
Else
Set REAR = REAR +1
[End of If data structure]

Step 3 : Set QUEUE [REAR] = ITEM This insert new line

Step 4 : Return.

Q 59. What is a top pointer of stack? **(PTU, May 2013)**

Ans. Top pointer is a pointer to the stack from where operations are performed i.e. starting pointer where push and pop operations can be performed easily.

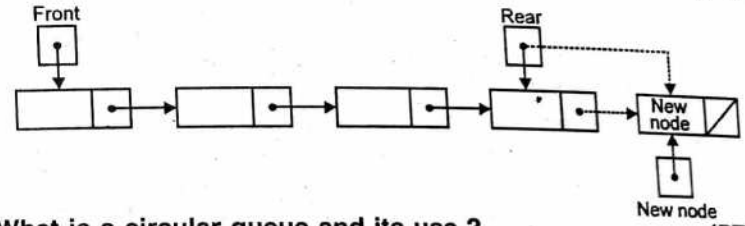


Q 60. Post-fix Expression. **(PTU, May 2014)**

Ans. In postfix expression if operator symbol is placed after its two operands, it is called postfix or reverse polish expression and this way of writing an expression is called as postfix or reverse polish notation of an expression.

Q 61. Linked representation of queue. **(PTU, May 2014)**

Ans.



Q 62. What is a circular queue and its use ? **(PTU, Dec. 2014)**

Ans. It is better than a normal queue because in this we can effectively utilize the memory space. If we have a normal queue and have deleted some elements from there then empty space is created there and even if the queue has empty cells than also we cannot insert any new element because the insertion has to be done from one side only and deletion has to be done from another side.

Application : Circular queue is used in VB.net
Polynomials
Sequential atomic operations

Q 63. Evaluate : (a) 1 24 3 + * 41 − (b) 25 7 * 14 − 6 + **(PTU, Dec. 2014**

Ans. (a) 1 24 3 + * 41 −

Token read	Content of stack
1	1
24	1, 24
3	1, 24, 3
+	1, 27
*	27
41	27, 41
−	14
)	14

(b) 25 7 * 14 − 6 +

Token read	Content of stack
25	25
7	25 7
*	175
14	175, 14
−	161
6	161, 6
+	167
)	167

Q 64. What is the usage of stack in recursive algorithm implementation ? **(PTU, Dec. 2**

Ans. In recursive algorithms, stack data structures is used to store the return add

when a recursive call is encountered and also to store the values of all parameters essent to the current state of the procedure.

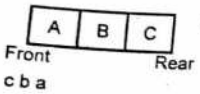
Q 65. What is the postfix form of the following prefix expression ? **(PTU, May 201**

*+ab–cd.

Ans. ab+cd–*

Q 66. Assume that a queue is available for pushing and popping elements. Give an input sequence a, b, c, (c be the first element), give the output sequence of element if the rightmost element given above is the first to be popped from the queue. **(PTU, May 201**

Ans.



c b a

Q 67. What are Circular Queues ? Write down routines for inserting and deletin elements from a circular queue implemented using arrays. **(PTU, May 201**

Ans. Circular Queue : *Refer to Q.No. 43*

Static queue have a very big drawback that once the queue is FULL, even though we delete few elements form the "front" and relieve some occupied space, we are not able to add anymore elements as the "rear" has already reached the Queue's rear most position. The solution lies in a queue in which the moment "rear" reaches its end; the "first" element will become the queue's new "rear". This type of queue is known as circular queue having a structure like this in which, once the Queue is full the "first" element of the Queue becomes the "Rear" most element, if and only if the "Front" has moved forward;

insert(queue, n, front, rear, item)

This procedure inserts an element item into a queue.

1. *If front = 1 and rear = n, or if front = rear + 1, then :*
   write : overflow, and return
2. *[find new value of rear]*
   If front = NULL, then : [queue initially empty.]
   set front = 1 and rear = 1.
   Else if rear = n, then:
   Set rear = 1.
   Else :
   Set rear = rear + 1.
   [end of structure.]
3. *Set queue[rear] = item. [this inserts new element.]*
4. *Return.*

delete(queue, n, front, rear, item)

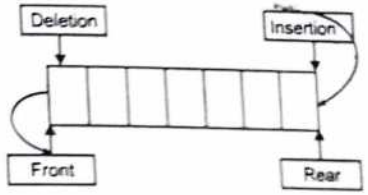
This procedure deletes an element from a queue and assigns it to the variable item.

1. *[queue already empty?]*
   if front = NULL, then :
   write : underflow, and return.
2. *Set item = queue[front].*

3. [find new value of front].
   If front = rear, then : [queue has only one element to start].
   Set front = NULL and rear = NULL.
   Else if front = n, then :
   Set front = 1.
   Else :
   Set front = front + 1.
   [end of structure.]
4. Return.

Q 68. Explain the terms Front and Rear for queue. **(PTU, May 2015)**

Ans. A queue is a non-primitive, linear data structure, and a sub-class of list data structure. It is an ordered, homogeneous collection of elements in which elements are appended at one end called **rear** end and elements are deleted at other end called **front** end. The meaning of front is face side and rear means back side.



Queue

Q 69. Explain application of Stack in recursive functions with example. **(PTU, May 2015)**

Ans. Recursive function uses stack. A stack is a Last In First Out (LIFO) data structure This means that the last data item to get stored on the stack (often called push operation) is the first data item to get out of the stack (often called pop operation). This is similar to that of stacking plates-the last plate that goes on the stack is the first one to get out of it.

- When function call itself, a new set of variables and parameters are stored on the stack and the function code is executed from the top with these new variables.
- As each recursive call returns, the old variables and parameters declared in the function are removed from the stack, and execution resumes immediately after the recursive call inside the function.
- The recursive function must have a conditional statement, such as if, somewhere to force the function to return without the recursive call being executed.

We can also say that, in the recursive function the call to self function is made. The recursive call means invoking the push operation, and when the control is returned from recursive function to main the pop operation is supposed to be performed. The recursion uses the concept of internal stack.

Q 70. What is the role of Priority queue in operating system design ? **(PTU, Dec. 201**

Ans. The typical example of priority queue is scheduling the jobs in operating syste Priority queues are used in well-known computer science application is job schedul algorithms in the operating system design where the job or tasks with highest priorities ha to be processed first. Typically operating system allocates priority to jobs. The jobs are plac in the queue and position of the job in priority queue determines their priority. There are th kinds of jobs.

1. Real time
2. Foreground
3. Background

The operating system always schedules the real time job first. If there is no real time job pending then it schedules foreground jobs. Lastly it schedules the background jobs.

Q 71. Why do we need Queues ? Write the algorithms/programs for EmptyQ, FullQ, InsertQ and DeleteQ operations. **(PTU, Dec. 2015)**

Ans. A queue is logically a first in first out (FIFO) type of list. In our everyday life we come across many situations where we ought to wait for the desired service, there we have to get into a waiting line for our turn to get serviced. This waiting queue can be thought of a queue. Queue means a line. For example : At the railway reservation booth, we have to get into the reservation queue. Thus a queue is a non-primitive data structure. It is an homogeneous collection of elements in which new elements are added at one end called the rear end and the existing elements are deleted from other end called the front end :

1. This algorithm is used to check whether a queue is empty or not
   EMPTY-CHECK (QUEUE, FRONT, REAR, EMPTY)
   1. If (FRONT=REAR + 1) then
      a . EMPTY : = true;
   2. Otherwise
      a . EMPTY : = false;
   3. Return
2. This algorithm is used to check whether a queue is full or not.
   FULL-CHECK (QUEUE, FRONT, REAR, MAX, FULL)
   1. If (REAR = MAX) then
      a. FULL : = true;
   2. Otherwise
      a. FULL : = false;
   3. Return;
3. This algorithm is used to add or insert item to Queue.
   INSERT - ITEH (QUEUE, FRONT, REAR, MAX, ITEM)
   1. If (REAR = MAX) then
      a. Display "Queue overflow";
      b. Return;
   2. Otherwise
      a. REAR : = REAR + 1;
      b. QUEUE (REAR) : = ITEM;
   3. Return;
4. This algorithm is used to delete an item from queue.
   REMOVE - ITEM (QUEUE, FRONT, REAR, ITEM)
   1. If (FRONT = REAR + 1) then
      a. Display "Queue underflow";
      b. Return;

   2. Otherwise
      a. ITEM : = QUEUE (FRONT);
      b. FRONT : = FRONT + 1;
   3. Return;

Q 72. If the characters 'D', 'C', 'B', 'A' are placed in a queue (in that order), and then removed one at a time, in what order will they be removed ? **(PTU, May 2016)**

Ans. DCBA

Because As Queue follows FIFO structure.

Q 73. Suppose we have a circular array implementation of the queue with ten items in the queue stored at data [2] through data [11]. The current capacity is 42. Where does the insert method place the new entry in the array ? **(PTU, May 2016)**

Ans. data [12]

Q 74. What are the different representations of stacks ? How recursion functions can be implemented using stacks ? **(PTU, Dec. 2016)**

Ans. There are two ways to represent stacks in memory :

1. Array representation (Static data structure)
2. Linked list representation (Dynamic data structure)

A function calling itself or a call to another function, which in turn calls the first function is called a recursive function. These recursive function are executed using stacks. Return address and all local variables and formal parameters of the called method will be stored into the stack. Whenever any method is called all the elements stored in the stack will be restored after a return is executed. Recursion is implemented using the data structure stack.

Q 75. Explain how stack is applied for evaluating an arithmetic expression. **(PTU, May 2017)**

Ans. Evaluating Arithmetic expressions :

1. In order to evaluate an expression, standard precedence rules for arithmetic expression are applied.
2. The parenthesis are evaluate first followed by unary operator.
3. When unparenthesized operator of the same parenthesis are evaluated, the order of evaluation is from left to right except in the case of NOT (!), in which case the order is from right to left.

Q 76. Write an algorithm to convert infix to postfix expression and apply the same to convert the following expression from infix to postfix.

(a) (a * b) + c/d (b) (((a/b) – c) + (d * e)) – (a * c) **(PTU, May 2018)**

Ans. (a) (a * b) + c/d
(ab*) + c/d
ab * c/d +

(b) (((a/b – c) + (d*e)) – (a * c)
((ab/ – c) + (de *)) – (ac*)
ab/c – de * + ac * –

(PTU, May 2018)

Q 77. Write the drawbacks of DQUEUES.

Ans. With the DQUEUES a complication may arise :

(a) When there is a overflow, that is, when an element is to be inserted into a dequeue which is already full or

(b) When there is underflow that is when an element is to be deleted from a dequeue which is empty.

❑❑❑

Chapter 3A

Linked List

Contents

Singly linked lists : Representation in memory, Algorithms of several operations : Traversing, Searching, Insertion into, Deletion from linked list ; Linked representation of Stack and Queue, Header nodes, Doubly linked list : Operations on it and algorithms analysis ; Circular Linked Lists : All operations their algorithms and the complexity analysis.

POINTS TO REMEMBER

- A linked list is a linear data structure containing of elements called nodes where each node is composed of two parts : information part and a link part also called next pointer part.
- A pointer to the first node is called a start or head pointer. It contains the address of the first node of the linked list.
- Linked list is a dynamic data structure as the size of the linked list is not fixed and may expand or shrink as nodes are inserted or deleted.
- Traversing, searching, insertion, deletion and sorting are the key operations performed on a linked list.
- Searching is a process of finding the location of the node containing the desired item in the linked list. Search is considered successful if the node containing the item is found and unsuccessful otherwise.
- The availability list or free storage list is maintained to keep track of the free nodes. Whenever a node is to be inserted into a linked list, the first node is taken of from the available list and linked to the former list as required. On the other hand, whenever a node is deleted from a linked list, it is added to the front of the availability list from where it can be used for insertion purpose at some later stage.
- Overflow occurs during the insert operation on a linked list when AVAIL = NULL. The underflow occurs during the delete operation of linked list when HEAD = NULL.
- A circular linked list is a linked list in which the last node in the list points to the first node in the list.
- A doubly linked list is a linear collection of elements called nodes such that each node consists of three parts : information part, two pointers one pointing to the next node and other pointing to the previous node.

- Linked lists are used in a wide variety of applications such as polynomial representation, addition of long positive integers, symbol table creation, mailing lists, memory management etc.
- The memory locations that once were needed but become unnecessary and unused at some later time are called the garbage.

QUESTION-ANSWERS

Q 1. What data structure operations can be applied to linked lists? **(PTU, Dec. 2004)**

Ans. Data structure operations that can be applied to linked lists are :

1. Insertion
2. Deletion
3. Traversing
4. Searching (only linear searching).

Q 2. Which searching algorithms will you apply for a given item in S, when S is stored as a linked list? **(PTU, May 2005)**

Ans. In case of link list, only linear searching is applied. The worst case running time is proportional to no. 'n' of elements in LIST, and average case running time is approximately proportional to $\frac{n}{2}$. A binary search algorithm can't be applied to a sorted link list, since there is no way of indexing middle element in the list.

Q 3. What is underflow? **(PTU, May 2005)**

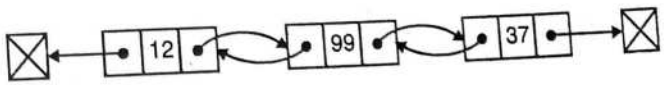
Ans. The term underflow refers to the situation where one wants to delete data from a data structure that is empty. The programmer may handle underflow by printing the message UNDERFLOW. Underflow will occur with our linked lists when START = NULL and there is a deletion.

Q 4. What do you mean by doubly linked list? **(PTU, May 2009)**

OR

What is double link list? **(PTU, May 2006)**

Ans. In computer science, a doubly-linked list is a linked data structure that consists o a set of data records, each having two special link fields that contain references to the previou and to the next record in the sequence. It can be viewed as two singly-linked lists formed fro the same data items, in two opposite orders.



A doubly-linked list whose nodes contain three fields : an integer value, the link to next node, and the link to the previous node.

The two links allow walking along the list in either direction with equal ease. Compared to a singly-linked list, modifying a doubly-linked list usually requires changing more pointers, but is sometimes simpler because there is no need to keep track of the address of the previous node.

Q 5. What is overflow? **(PTU, May 2011 ; Dec. 2005)**

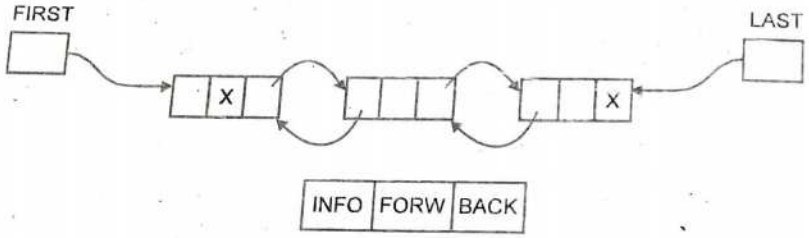
Ans. Sometimes new data are to be inserted into a data structure but there is no available space, i.e., the free-storage list is empty. This situation is usually called overflow. The programmer may handle overflow by printing the message OVERFLOW.

Q 6. What are doubly linked list? How are they represented in memory? **(PTU, May 2007)**

Ans. A doubly link list is a linear collection of data elements, called nodes, where each node 'N' is divided into three parts :

1. Info
2. FORW
3. BACK.

Thus, doubly link list can be traversed in two directions in forward direction from beginning to end and in backward direction from end to beginning.



Q 7. How linked lists are represented in memory? **(PTU, May 2008)**

Ans. Represented of linked list in memory : Let the list be a linked list. Then LIST will be maintained in memory, unless otherwise specified or implied, as follows. First of all, LIST requires two linear arrays – we have INFO and LINK – such that INFO [K] and LINK [K] contain respectively, the information part and next pointer field of a node of LIST. As LIST also requires a variable name – such as START – which contain the location of begining of list and a next pointer sentinal denoted by NULL – which indicates end of the list. Since subscript of the arrays INFO and LINK will usually be positive, we will use NULL = 0, unless otherwise started.

Q 8. What is the header linked list? **(PTU, May 2008)**

OR

What are header linked lists? What are advantages and disadvantages of using header node? **(PTU, Dec. 2007 ; May 2005)**

Ans. Header Linked Lists : A header linked list is a list which always contains a special ode, called header node, at the beginning of the list. The following are two kinds of widely ed header lists.

1. The grounded header list is a header list where the last node contain the null pointer. (see fig. 1)
2. The circular header list is a header list where the last node points back to header node. (see fig. 2)

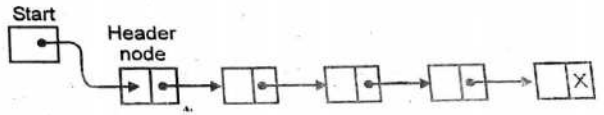


Fig. 1. Grounded header list

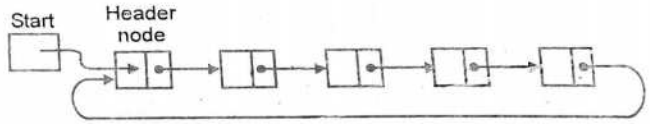


Fig. 2. Circular header list

Advantages :

1. The header linked lists are frequently used for maintaining polynomial in memory.
2. The header linked list are performing traversing, searches, deleting and inserting operations of elements in a list.

Q 9. What is need for Garbage collection? **(PTU, Dec. 2010, 2004)**

Ans. The operating system of a computer may periodically collect all the deleted space onto the free-storage list. Any techniques which does this collection is called garbage collection. Garbage collection usually takes place in two steps. First the computer runs through all lists, tagging those cells which are currently in use and then the computer runs through the memory, collecting all untagged space onto the free-storage list.

The garbage collection may take place where there is only some minimum amount of space or no space at all left in the free-storage list, or when the CPU is idle and has time to do the collection. Generally speaking, the garbage collection is invisible to the programmer. Any further discussion about this topic of garbage collection lies beyond the scope of this text.

Q 10. Write an algorithm for insertion of an item after the given node in the linked list. **(PTU, May 2008)**

Ans. INSLOC (INFO, LINK, START, AVAIL, LOC, ITEM)

This algorithm inserts ITEM so that ITEM follows the node with location LOC or inserts ITEM as first node when LOC = NULL.

Q 11. State the advantages of link list over array. **(PTU, May 2006)**

Ans. Advantages :

1. Less memory wastage : In an array, consecutive memory location are required so, there is wastage of memory, but in case of link list, there is no memory wastage.

2. Traversing : In case of arrays, traversing is linear, but in link list, traversing depends on the pointer used.

3. Insertion and Deletion : Insertion and deletion in an array is difficult to perform, while in case of link list, it is easier due to the concept of 'AVAIL'.

4. Memory allocation : Arrays require consecutive memory locations to be stored, but elements in link list can be stored at any location.

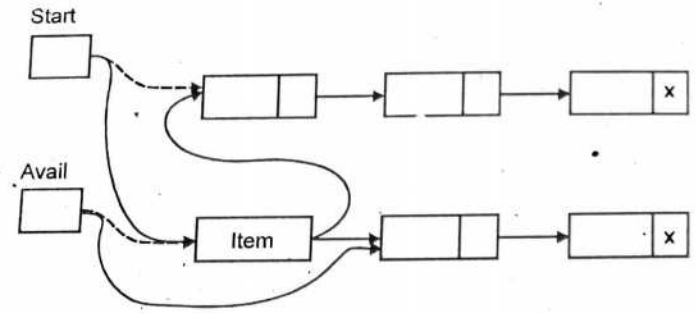
The only disadvantages of link list over arrays is that it is difficult to implement, while array is easy to implement and it is the simplest.

Q 12. What are the various operations possible on a singly link list? Explain with diagrams. **(PTU, May 2019, 2004)**

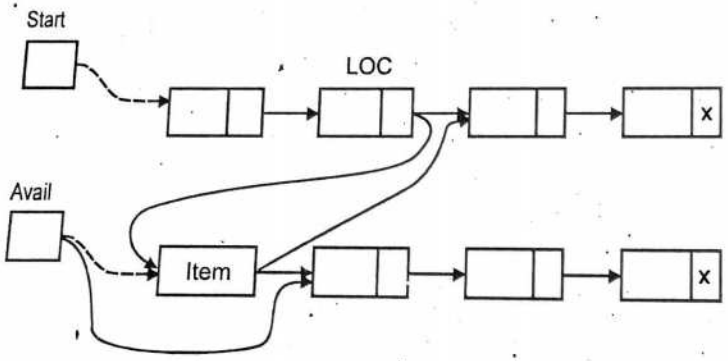
Ans. Various operations possible on a singly link list are :

1. Insertion : Insertion is possible in case of link list in 3 cases.

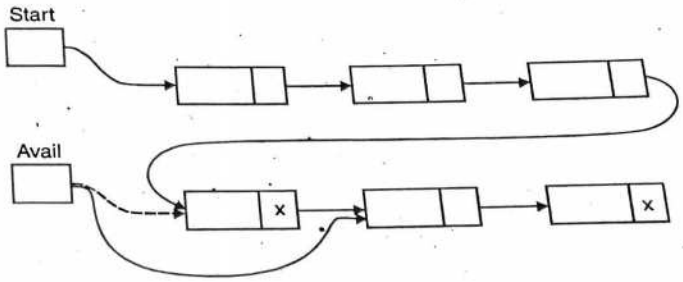
Insertion at the beginning : It means inserting a node and given item at the beginning.



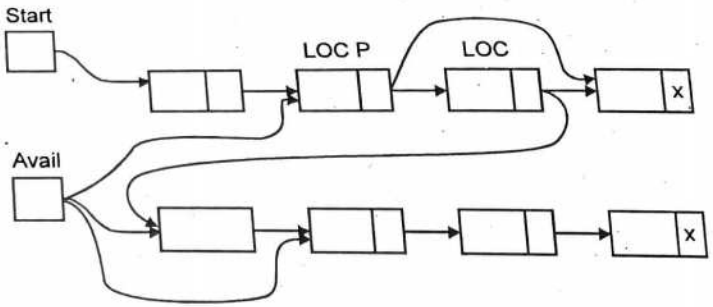
Insertion after a given node : Here, we insert a node following a given node.



3. Insertion at the end : In this, we insert a node of given item in the end.



(ii) Deletion : Deletion means deleting a node of given location.



Q 13. What are the various operations possible on a doubly link list? Explain with examples. **(PTU, May 2010, 2004)**

OR

What is linked list? Write an algorithm to insert, delete and modify operations on double link list. **(PTU, Dec. 2006)**

Ans. A linked list is a linear collection of data elements, called nodes, where the linear order is given by means of pointers. Here, each node is divided into two parts : the first part contains information of element and second part, called link part, contains the address of next node in list. In the case of doubly linked list, link part contains two parts, FORW and BACK. A pointer field FORW contains the location of the next node in the list and BACK contains the location of the preceding node in the list.

1. Insertion in double link list

INSTWL (INFO, FORW, BACK, START, AVAIL, LOCA, LOCB, ITEM)

1. If AVAIL = NULL, then :
   Write : overflow and exit.
2. Set NEW = AVAIL, AVAIL = FORW [AVAIL],
   INFO [NEW] = ITEM,

3. Set FORW [LOCA] = NEW, FORW [NEW] = LOCB
   BACK [LOCB] = NEW, BACK, [NEW] = LOCA
4. Exit.

2. Deletion in double link list

DELTWL (INFO, FORW, BACK, START, AVAIL, LOC)

1. Set FORW [BACK [LOC]] = FORW [LOC]
   and BACK [FORW [LOC]] = BACK [LOC]
2. Set FORW [LOC] = AVAIL
   AVAIL = LOC
3. Exit.

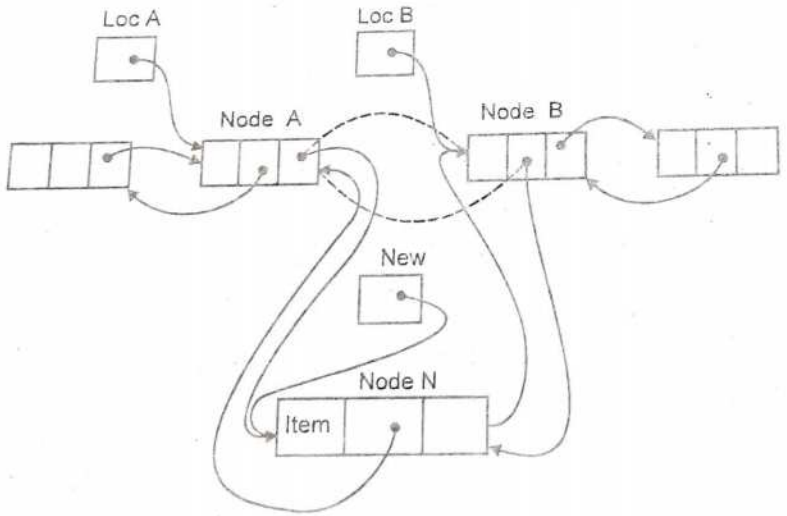
Q 14. Write and explain the algorithm for insertion of a node between two adjacent nodes A and B of a 2-way reader list. **(PTU, Dec. 2004)**

Ans. Suppose we are given location LOCA and LOCB of adjacent nodes A and B in LIST, and suppose we want to insert a given ITEM of information between nodes A and B. Firstly, we remove the lst node 'N' from AVAIL list, using the variable NEW to keep trace of this location, and copy the data ITEM into node N, i.e.,

NEW = AVAIL, AVAIL = FORW [AVAIL], INFO [NEW] = ITEM

Now, the node 'N' with contents ITEM is inserted into list by changing the following four pointers.

FORW [LOCA] = NEW, FORW [NEW] = LOCB
BACK [LOCB] = NEW, BACK [NEW] = LOCA



STWL (INFO, FORW, BACK, START, AVAIL, LOCA, LOCB, ITEM)

If AVAIL = NULL, then
Write : overflow, and Exit.

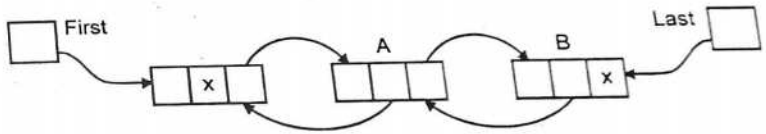
2. Set NEW : = AVAIL
   AVAIL = FORW [AVAIL]
3. Set INFO [NEW] = ITEM
4. Set FORW [LOCA] = NEW, FORW [NEW] = LOCB
   BACK [LOCB] = NEW, BACK [NEW] = LOCA
5. Exit.

Q 15. What are two way lists? What are main advantages of 2-way lists? **(PTU, Dec. 2005)**

Ans. A two-way list is a linear like list which can be transversed in two directions, i.e. forward and backward. For a given location of a node in the list, we now have immediate access to both next node and preceeding node.

In two-way list, each node is divided into three parts :

1. Info
2. BACK : Gives preceding node location.
3. FORW : gives next node location



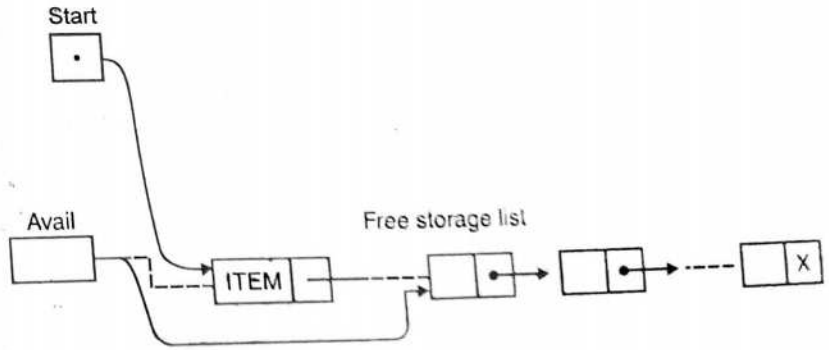
Suppose LOCA and LOCB are locations of nodes A and B in a two-way list. Then,
FORW [LOCA] = LOCB and BACK [LOCB] = LOCA

Advantages :

1. One have access to both preceding and succeeding node.
2. One can begin from last node to first node or first to list if desirable.

Q 16. Write an algorithm for insertion of an item at the beginning of the linked list. **(PTU, Dec. 2008)**

Ans.



Algorithm :

INS FIRST (INFO, LINK, START, AVAIL, ITEM)

This algorithm inserts ITEM as first node in the list.

Step 1. [OVERFLOW] It AVAIL = NULL then write : OVERFLOW and Exit.

Step 2. [Remove first node from AVAIL List]
Set NEW = AVAIL and AVAIL = LINK [AVAIL]

Step 3. Set INFO [NEW] = ITEM [Copies new data into new node]

Step 4. Set LINK [NEW] = START [New node now points to original first node]

Step 5. Set START = NEW [Change is START so it points to new node]

Step 6. Exit.

Q 17. Give an algorithm for creating and inserting a node in a linked list.
(PTU, May 2009)

Ans. *The following is algorithm for creating and inserting node is a linked list.*

INSLOC (INFO,LINK, START, AVAIL, LOC, ITEM)

Step 1 : *[OVERFLOW?] If AVAIL = NULL then write OVERFLOW and Exit.*

Step 2 : *[Remove first node from AVAIL list]*
Set NEW = AVAIL and AVAIL = LINK [AVAIL].

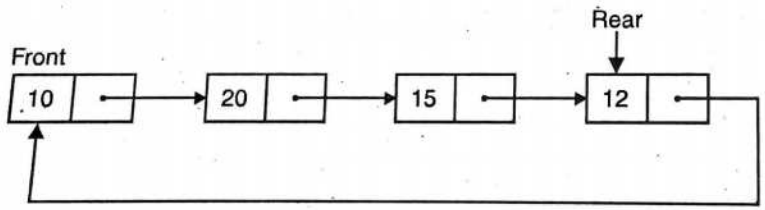
Step 3 : *Set INFO [NEW] = ITEM [copies new data into new node]*

Step 4 : *If LOC = NULL then [Insert as first node]*
Set LINK [NEW] = START and START = NEW
Else [Inset after node with location LOC]
Set LINK [NEW] = LINK [LOC] and
LINK [LOC] = NEW
[End of If Structure]

Step 5 : Exit.

Q 18. What is a circular list? Write an algorithm for inserting a node at the front.
(PTU, Dec. 2009)

Ans. *A circular list is a linear list in which the last node contains the address of first node. In some application, it is convenient to use circular linked list. A queue data structure can be implemented using a circular linked list, with a single pointer "rear" as the front node can be accessed through the rear node.*



Following is the algorithm for inserting a node at the front.

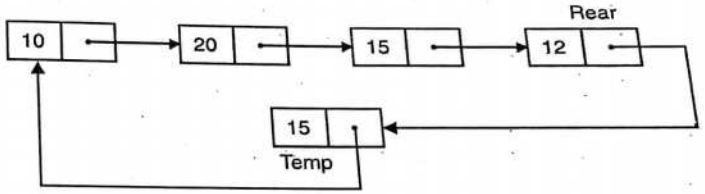
Add front (NUM)

Step 1. Temp = New NODE [Acquire memory location for new node]

Step 2. Temp → info = NUM [store NUM in newly acquired node]

Step 3. Temp → link = Rear → link [Store the address of first node into new node]

Step 4. Rear → link = Temp [Store the address of new node into the rear node]



Q 19. Write suitable routines to perform insertion and deletion operations in a linked list.
(PTU, Dec. 2009)

Ans. Suppose START is the first position in linked list. Let DATA be the element to be inserted in the new node. POS is the position where the new node is to be inserted. TEMP is a temporary pointer to hold the node address. TEMP is a temporary pointer to hold the node address. Following is the algorithm to insert a node at any specified position :

1. Input DATA and POS to be inserted.
2. Initilise TEMP = START, and K = 0.
3. Repeat the step 3 while (K is less than POS)
   (a) TEMP = TEMP → Next
   (b) If (TEMP is equal to NULL)
      (i) Display 'Number of nodes i the list are less than position'
      (ii) Exit.
   (c) K = K + 1
4. Creat a new node.
5. New node → Data = Data
6. New node → Next = TEMP → Next
7. TEMP → Next = New Node
8. Exit.

Algorithm for deleting a node :

1. Input the DATA to be deleted.
2. If (START → DATA = = DATA)
   (a) TEMP = START
   (b) START = START → NEXT
   (c) Free the node TEMP
   (d) Exit
3. PTR = START

4. While (PTR → Next → Next ≠ NULL)
   (a) If (PTR → NEXT → DATA = = DATA)
      (i) TEMP = PTR → NEXT
      (ii) PTR → Next = TEMP → Next
      (iii) Set tree the node TEMP, which is deleted
      (iv) Exit.
   (b) PTR = PTR → Next
5. Display "Data Not fond"
7. Exit.

Q 20. What are the various application of link list? How is it different from array as a data structure? Explain with examples. **(PTU, May 2010)**

Ans. A linked list is a linear collection of data elements linked to one another by means of pointers. Linked lists are used in a wide variety of application. Some of the application of the linked lists are :

1. Linked lists are used to implement other data structures such as stacks, queues, trees and graphs.
2. Linked lists can be used to maintain a directory of names.
3. Linked lists can be used to perform arithmetic operations on long integers.
4. Polynomials can be manipulated by using linked lists.
5. Sparse matrics can be represented with the help of linked lists.

The principal benefit of a linked list over an array is that the order of the linked items may be different from the order that the data items are stored in memory. For that reason, linked lists allow insertion and deletion of nodes at any point in the list, with a constant number of operations.

Arrays are of fixed size but linked is not fixed, it can grow as the data item increases. But in arrays data access is very fast became data item can be directly accessed by its index value.

Linked lists have an extra overhead in its each node to store pointers to its next node.

Q 21. What are the various operation possible on a doubly link list? Explain with the algorithm and graphically. **(PTU, Dec. 2010)**

OR

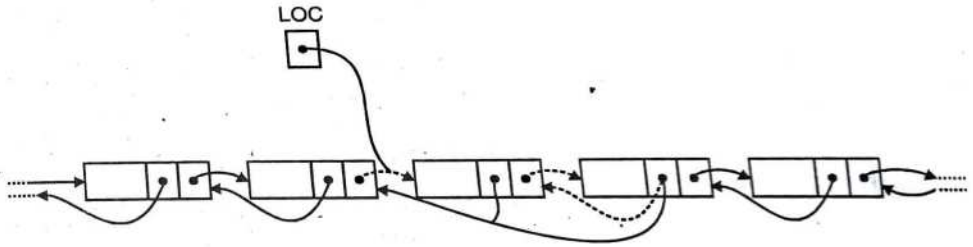
What are the various operations possible on a doubly link list? Explain with examples. **(PTU, May 2011)**

Ans. Operations on Two-Way Lists : Suppose LIST is a two-way list in memory. This subsection discusses a number of operations on LIST.

Traversing : Suppose we want to traverse LIST in order to process each node exactly once. Then we can use algorithm if LIST is an ordinary two-way list, or we can use algorithm if LIST contains a header node. Here it is of no advantage that the data are organized as a two-way list rather than as a one-way list.

Searching : Suppose we are given an ITEM of information a key value – and we want to find the location LOC of ITEM in LIST. Then we can use Algorithm if LIST is an ordinary two-way list, or we can use algorithm if LIST has a header node. Here the main advantage is that we can search for ITEM in the backward direction if we have reason to suspect that ITEM appears near the end of the list. For example, suppose LIST is a list of names sorted alphabetically. If ITEM = Smith, then we would search LIST in the backward direction, but if ITEM = Davis, then we would search LIST in the forward direction.

Deleting : Suppose we are given the location LOC of a node N in LIST and suppose we want to delete N from the list. We assume that LIST is a two-way circular header list. Note that BACK [LOC] and FORW [LOC] are the locations, respectively, of the nodes which precede and follow node N. According as pictured in fig., N is deleted from the list by changing the following pair of pointers :



Deleting Node N

FORW [BACK [LOC]] = FORW [LOC] and BACK [FORW [LOC]] : = BACK [LOC]

The deleted node N is then returned to the AVAIL list by the assignments :

FORW [LOC] = AVAIL and AVAIL = LOC

The formal statement of the algorithm follows :

DEL TWL (INFO, FORW, BACK, START, AVAIL, LOC)

1. [Delete node]
   Set FORW [BACK [LOC]] : = FORW [LOC] and
   BACK [FORW [LOC]] : = BACK [LOC].
2. [Return mode to AVAIL list]
   Set FORW [LOC] : = AVAIL and AVAIL : = LOC.
3. Exit.

Here we see one main advantage of a two-way list. If the data were organized as one way list then in order to delete. N, We would have to traverse the one way list to find the location of the hode preceding N.

Inserting : Suppose we are given the location LOCA and LOCB of adjacent nodes A and B in LIST and suppose we want to insert a given ITEM of information between nodes A and B. As with a one way list, first we remove the first node N from the AVAIL list, using the variable NEW to keep track of its location, and then we copy the data ITEM into the node N ; that is, we set :

NEW : = AVAIL, AVAIL : = FORW [AVAIL], INFO [NEW] : = ITEM

Now, as pictured in fig., the node N with contents ITEM is inserted into the list by changing the following four pointers :

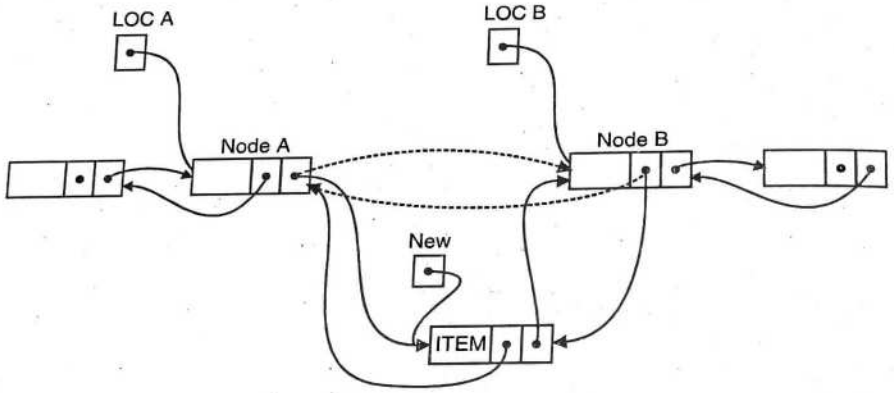
FORW [LOCA] : = NEW, FORW [NEW] : = LOC B

BACK [LOC B] : = NEW, BACK [NEW] : = LOC A

The formal statement of our algorithm follows :

INSTWL (INFO, FORW, BACK, START, AVAIL, LOCA, LOCB, ITEM)

1. [OVERFLOW?] if AVAIL = NULL, then : Write : OVERFLOW, and Exit.
2. [Remove node from AVAIL list and copy new data into node]
   Set NEW = AVAIL, AVAIL := FORW [AVAIL], INFO [NEW] : = ITEM.



3. [Insert node into list]
   Set FORW [LOC A] : = NEW, FORW [NEW] : = LOC B,
   BACK [LOC B] : = NEW, BACK [NEW] : = LOC A.
4. Exit.

Algorithm assumes that LIST contains a header node. Hence LOC A or LOC B may point to the header node. In which case N will be inserted as the first node or the last node. If LIST does not contain a header node then we must consider the case that LOCA = NULL and N is inserted as the first node in the list, and the case that LOCB = NULL and is inserted as ne last node in the list.

Remark : Generally speaking, starting data as a two-way list.Which requires extra space the backward pointers and extra time to change the added pointers rather than as a one y list is not worth the expense unless one must frequently find the location of the node ch proceeds a given node N as in the deletion above.

Q 22. Write an algorithm to concatenate two linked lists. **(PTU, May 2007)**

Ans. P = LISTA. Head → Prev ;

Q = LISTB. Head → Prev ;

P → Next = LISTB. Head ;

LISTB. Head → Prev = P ;

LISTA. Head → Prev = Q ;

Q → Next = LISTA. Head ;

Q 23. What is the need for garbage collection? **(PTU, May 2011)**

Ans. Garbage collection (gc) is a form of automatic memory management.The garbage collector or just collector, attempts to reclaim garbage or memory occupied by objects that are no longer in use by the program.

Benefits : Garbage collection frees the programmer from manually dealing with memory deallocation. As a result, certain categories of bugs are eliminated or substantially reduced.

- **Dangling Pointer Bugs :** Which occur when a piece of memory is freed while there are still pointers to it and one of those pointers is then used. By then the memory may have been re-assigned to another use, with unpredictable results.
- **Double Free Bugs :** Which occur when the program tries to free a region of memory that has already been freed and perhaps already been allocated again.
- Certain kinds of memory leaks in which a program fails to free memory occupied by object that will not be used again leading over time, to memory exhaustion. Some of these bugs can have security implications.

Q 24. What is circular linked list? **(PTU, May 2011)**

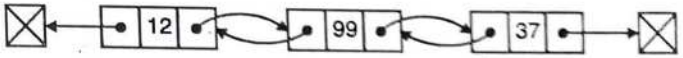
Ans. A circulary linked list may be a natural option to represent that are naturally circular, e.g. the corners of a polygon, a pool of buffers that are used and released in FIFO order or a set of processes that should be time-shared in round-robin order. In these applications a pointer to any node server as a handle to the whole list.

With a circular list a pointer to the last node gives easy access also to the first node, by following one link. Thus, in applications that require access to both ends of the list (e.g in the implementation of a queue), a circular structure allows one to handle the structure by a single poniter, instead of two.

A circular list can be split into two circular lists, in constant time, by giving the addresses of the last node of each piece.

Q 25. Give the advantages of doubly linked list over single linked list. **(PTU, Dec. 2012)**

Ans. In a doubly linked list, each node contains, besides the next-node link, a second link field pointing to the previous node in the sequence. The two links may be called forward and backward or next and previous.



In single link list there is no backward pointer.

Q 26. How a linked list is different from array as a data structure. What are the various applications of linked list? Explain with examples. **(PTU, May 2011)**

Ans. Linked lists have several advantages over arrays. Elements can be inserted into linked liks indefinitely while an array will eventually either fill up or need to be resized, an

expensive operation that may not even be possible. If memory is fragmented. Similarly, an array from which many elements we removed may become wastefully empty or need to be made smaller.

On the other hand, arrays allow random access, while linked lists allow any sequential access to elements. Singly linked lists in fact, can only be traversed in one direction. This makes linked lists unsuitable for applications where it's useful to look up an element by its index quickly such as heap sort. Sequential access on arrays is also faster than on linked lists on many machines due to locality of reference and data caches. Linked lists received almost no benefit from the cache.

Another disadvantage of linked listed lists is the extra storage needed for references, which often makes them impractical for lists of small data items such as characters or boolean values. It can also be slow and with a naive allocator, wasteful to allocate memory separately for each new element, a problem generally solved using memory pools.

The main applications of linked lists are :

- For representing polynomials it means in addition subtraction/multiplication of two polynomials. Eg. P1 = 2 × ^2 + 3X + 7 and
  $P^2 = 3 \times ^3 + 5x + 2P^1 + P^2 = 3 \times ^ 3 + 2 \times ^2 + 8x + 9$.
- In dynamic memory management in allocation and releasing memory at run time.
  *In symbol table in balancing parenthesis.
- Representing sparse matrix.

Q 27. What are the limitations of linked lists? **(PTU, May 2012)**

Ans. The limitations of linked list are :

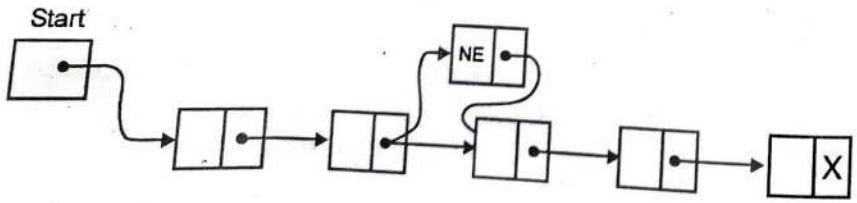
(a) Linked list insert data using technique for doubly linked list.

(b) Linked list can be insert data first, last.

Q 28. Suppose the names of few students of a class are as below :
Ram, Sham, Mohan, Sohan, Vimal, Komal
It is assumed that the names of the students is represented as a single link list.
ite a algorithm/program to insert the name of a student Raman between Sham and han. Represent it graphically also. **(PTU, May 2013)**

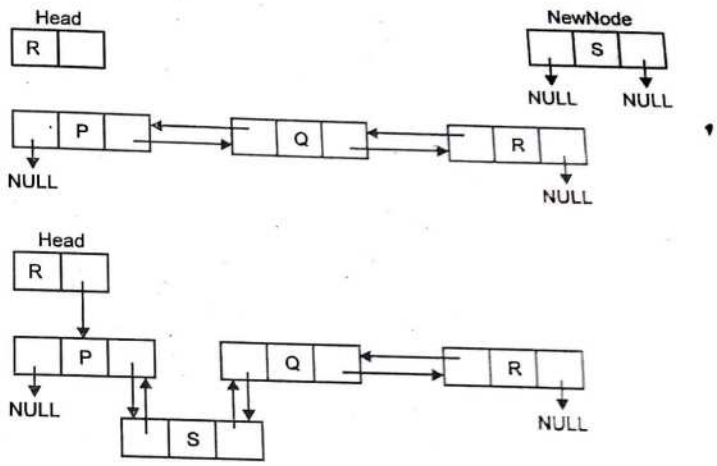
Ans. Insloc (Info, Link, Start, Avail, Loc, Item)

1. If avail = Null, then write overflow and exit.
2. Set NEW = Avail and Avail = Link [Avail]
3. Set Info [NEW] = Item
4. If LOC = Null then
   Set Link [NEW] = Start and Start = New
   Else
   Set Link [New] = Link [LOC] and Link [LOC] = New
5. Exit



Q 29. Let there be a doubly link list with three element, P, Q, R. How S can be added between P and Q. **(PTU, Dec. 2013)**

Ans.



Q 30. Implement a Single Linked List method that reverses the order of elements in Single Linked List. This method should run in O(n) time and shall not use recursion. **(PTU, May 2014)**

Ans. Implement an algorithm to reverse a singly linked list without recursion.

```
Node * RevSList (Node* pCur)
{
Node * pRev = NULL;
while (pCur)
{
Node * pNext = pCur → Next;
pCur → Next = pRev;
pRev = pCur;
pCur = pNext;
}
return pRev;
}
```

Q 31. What is garbage collection and how it can be implemented? **(PTU, May 2014)**

Ans. Garbage Collection : Garbage collection (GC), also known as automatic memory management, is the automatic recycling of dynamically allocated memory. Garbage collection is performed by a garbage collector which recycles memory that it can prove will never be used again. Systems and languages which use garbage collection can be described as garbage collected. Other words, Garbage collection (GC) is a dynamic approach to automatic memory management and heap allocation that processes and identifies dead memory blocks and

reallocates storage for reuse. The primary purpose of garbage collection is to reduce memory leaks.

GC implementation requires three primary approaches as follows :

1. Mark-and-sweep : In process when memory runs out, the GC locates all accessible memory and then reclaims available memory.

2. Reference counting : Allocated objects contain a reference count of the referencing number. When the memory count is zero, the object is garbage and is then destroyed. The freed memory returns to the memory heap.

3. Copy collection : There are two memory partitions. If the first partition is full, the GC locates all accessible data structures and copies them to the second partition, compacting memory after GC process and allowing continuous free memory.

Q 32. Define Linked list in data structure. How can we use linked list to implement Queue. Write suitable methods to perform enqueue, dequeue operations on queue using linked representation.

When implementing Queue with linked list, shall a pointer be kept to the end of the linked list also? What about when impelementing Stack using linked list?

(PTU, May 2019, 2014)

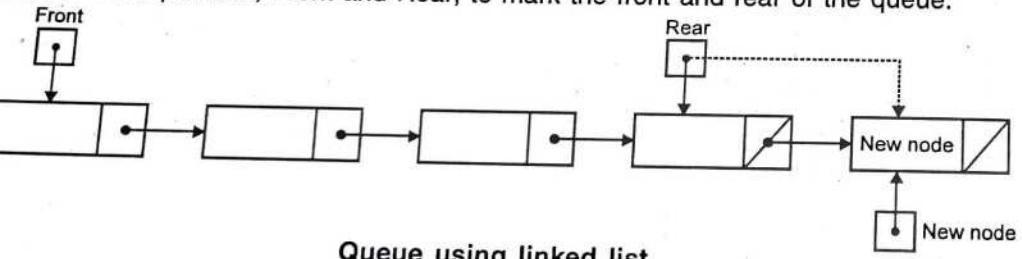
Ans. Linked List : A linked list is a set of nodes where each node has two fields "data" and "link".

The data field is used to store actual piece of information and link field is used to store address of next node.

An alternative and efficient representation of Queues is possible by using linked list for storing the data elements using dynamic memory.

Implemention of queues using linked list

- Allocate memory for each new element dynamically.
- Link the queue elements together.
- Use two pointers, Front and Rear, to mark the front and rear of the queue.



Queue using linked list

Enqueue Operation

Start
Read element
element → next = NULL
If (rear = NULL) then make following assignments
(i) rear = element
(ii) item = rear

5. (i) rear → next = element
   (ii) rear = element
6. Stop.

Dequeue operation

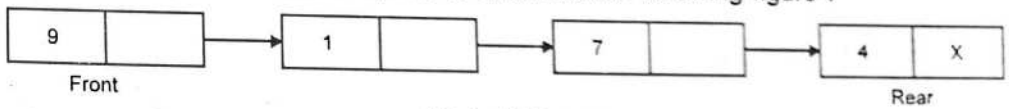
1. Start
2. If (item = NULL) then print that Queue is empty, go to step 7 after making assignment rear = item, otherwise proceed.
3. If (front = rear), then, there is only one element present in the list, remove it, go to step 7 after making an assignment front = rear = NULL, otherwise proceed.
4. element = item
5. item = item → next
6. Remove an element
7. Stop.

1. There is no need of knowing the stack size before implementing it, as dynamic memory allocation technique helps the programmer to declare the memory space at run time.
2. Data movement i.e. pushing or inserting and popping or deleting from stack is very easily handled with the concept of pointer in linked list.
3. The stack is never full as long as the system has enough space for Dynamic memory allocation.

Q 33. Explain linked representation of queue ?

Ans. The array implementation of queue cannot be used for the large scale applications one of the alternatives of array implementation is linked list implementation of queue. The storage requirement of linked representation of a queue with n elements is O (n) while the time requirement for operations is O (1). In the linked queue each node of the queue consists of two parts i.e. data part and the link past. Each element of the queue points to its immediate next element in the memory. In the linked queue, there are two pointer maintained in the memory i.e. front pointer and rear pointer. The front pointer contains the address of the element of the queue starting while the rear pointer contains the address of last element of the queue. Insertions and deletions are performed at rear and front end respectively. If front and rear both are NULL, it indicates that the queue of empty.

The linked representation of queue is shown in the following figure :



Linked Queue

Q 34. What is a header node ?

Ans. A header node is a special node at the beginning of the list in the header linked list. It is an extra node kept at the front of a list. Such a node does not represent an item in the list.

Q 35. What is B+ tree ? What are its advantages ?

Ans. B+ tree is an extension of B tree which allows efficient insertion, deletion and search operations. The leaf nodes of a B+ tree are linked together in the form of a singly

linked lists to make the search queries more efficient. B+ tree are used to store the large amount of data which cannot be stored in the main memory. Due to the fact that size of main memory is always limited,. the internal nodes of the B+ tree are stored in the main memory whereas, leaf nodes are stored in the secondary memory. The internal nodes of B+ tree are often called index nodes.

Advantages

1. Records can be fetched in equal number of disk accesses
2. Height of the tree remains balanced and less as compare to B tree.
3. We can access the data stored in a B+ tree sequentially as well as directly.
4. Keys are used for indexing.
5. Faster search queries as the data in stored only on the leaf nodes.

Q 36. Define stack. How stacks are implemented using linked list?

(PTU, Dec. 2006)

Ans. A stack is a list of elements in which an element may be inserted or deleted only at one end, called top of the stack. It uses LIFO operation, i.e. last in first out, two operations performed on stack are :

(i) Push, i.e. to insert an element to stack.

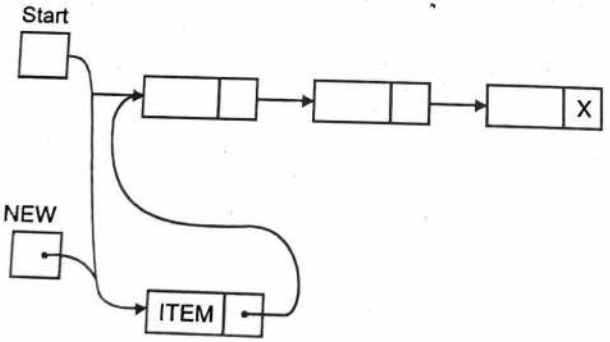
(ii) Pop, to delete an element from a stack.

Stacks can also be implemented using link list. In this case, we use a variable 'START' stead of 'TOP', rest all is same. In this case, insertion can only be possible at the beginning elements can be added only to top of the stack.

Insertion at the beginning

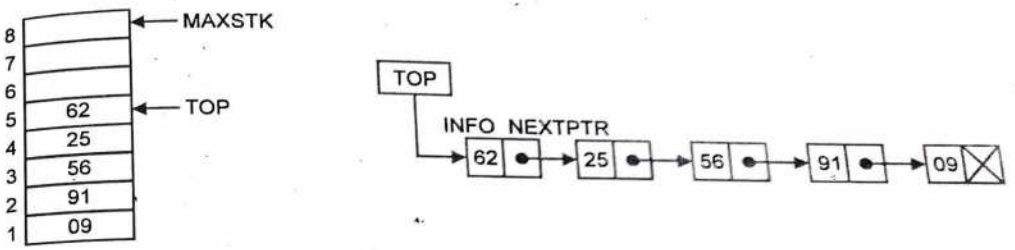
PUSH (TOP, AVAIL, INFO, ITEM, LINK)

1. If AVAIL = NULL, then ;
   Write overflow and Exit
2. Set NEW = AVAIL
   Set AVAIL = LINK [AVAIL]
3. Set INFO [NEW] = ITEM
4. Set LINK [NEW] = TOP
   TOP = NEW
5. Exit



Q 37. How stacks are represented using linked list a stack elements can also be using?

ns. Linked list : Such a representation is known as linked stack. The basic reason to ed stack is due to the fact that the size of array 1 stack must be known in advance and rray is limited that is it may cause over flow if we try to put an element in stack when ull.



Array representation **Linked stack**

TOP is the position of last element of stack.

MAXSTK is the maximum capacity of the stack.

Q 38. What is traversing ? Write an algorithm for traversing a link list ?

(PTU, Dec. 2014)

Ans. Traversing : Traversing a tree means processing it in such a way, that each node is visited only once.

Algorithm for traversing a linked list :

Step 1. Set Pointer = HEAD

Step 2. Repeat Steps 3 & 4 while Pointer ≠ NULL

Step 3. Apply Process to Data[Pointer]

Setp 4. Pointer = Link[Pointer]

Step 5. Stop

Q 39. Write the algorithms for the following:

(a) Deleting an element from a doubly link list.

(b) Inserting an element in a priority queue.

(c) To reverse a string of characters using stack.

(d) To search an element in a sorted array. **(PTU, Dec. 2014)**

Ans. (a) Deleting an element from a doubly link list :

1. Start
2. While list is not over
   Loop
   Check whether current node is not matching. If so go to the next node otherwise go to next step.
3. You reach this step only when matching node is found. Check whether matching node is first node. If so change starting address. Also change previous address of next node to null. Otherwise goto next step.
4. Check whether matching node is not the last node. If so set the next link of its previous node to next of matching node. Set the previous link of next node to previous of matching node. Otherwise goto next step.
5. You reach this step only when matching node is last node. So set the next link of previous node to null.
6. F[illegible] the memory of matching node and STOP. (End of loop)

7. Display error message that node is not found.
8. Stop.

(b) Inserting an element in a priority Queue :

1. Start
2. Read the value to be inserted and assign it to i.
3. Now insert the value at location point to by i, using an assignment PQ[i++] = value
4. Stop.

(c) To reverse a string of characters using stack :

Step 1. Create an empty stack.
Step 2. One by one push all characters of string to stack.
Step 3. One by one pop all characters from stack and put them back to string.

(d) To search an element in a sorted array :

1. *[Initialize]*
   Low = 0
   HIGH = n – 1 [LOW and HIGH denotes the lower and upper limit]
2. *[Perform search]*
   Repeat thru step 4 while LOW <= HIGH
3. *[Obtain index of midpoint of interval]*
   MIDDLE = INT ((Low + HIGH)/2)
4. *[Compare]*
   If ELEMENT < LIST[MIDDLE] [ELEMENT = given element, LIST = array]
   then HIGH = MIDDLE-1
   else
   If ELEMENT > LIST[MIDDLE]
   then
   LOW = MIDDLE + 1
   else
   Write('SUCCESSFUL SEARCH')
   POS = MIDDLE[POS = Position of given element]
   Write['UNSUCCESSFUL SEARCH']
   POS = NULL
   Exit.

0. Show the linked representation of the following two polynomials.

$7x^{80} + 5x^{50} + 3x^{30} + 1 = 0$

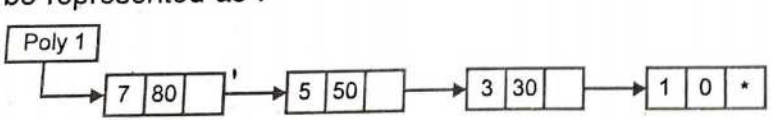
$9x^{90} + 6x^{60} + 2x^{20} - 1 = 0$

d a procedure for adding two polynomials stored in linked lists. Verify steps rocedure for the above two polynomials. (PTU, May 2015)

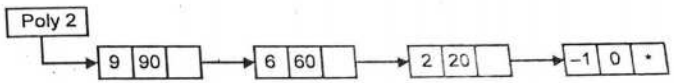
$7x^{80} + 5x^{50} + 3x^{30} + 1 = 0$ .....(1)

$9x^{90} + 6x^{60} + 2x^{20} - 1 = 0$ .....(2)

1 can be represented as :



Poly 2 can be represented as :



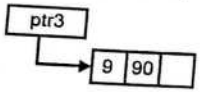
The node of the polynomial is of the form

Coefficient of the term	Power of x	Link to next node

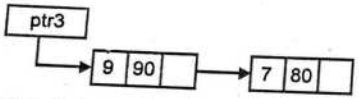
Procedure for Adding two Polynomials

```
typedef struct node_type
{
int coeff;
into power;
struct node_type *next;
} node;
node *poly;
void AddPolynomial(node *ptr1, node *ptr2, node **ptr3)
{
int powc;
int coef;
while((ptr1!= NULL) && (ptr2!= NULL)
{
if(ptr1 → power > ptr2 → power)
{
Coef = ptr1 → coeff;
powe = ptr1 → power;
ptr1 = ptr1 → next;
}
else if(ptr1 →power < ptr2 →power)
{
Coef = ptr2 →Coeff;
powe = ptr2 →power;
ptr2 = ptr 2 →next;
}
else
{
Coef = ptr1 →coeff + ptr2 →Coeff ;
powe  = ptr1 →power;
ptr1 = ptr1 →next;
ptr2 = ptr2 →next;
}
if(coef ! = 0)
add_node(ptr3, coef, powe);
```

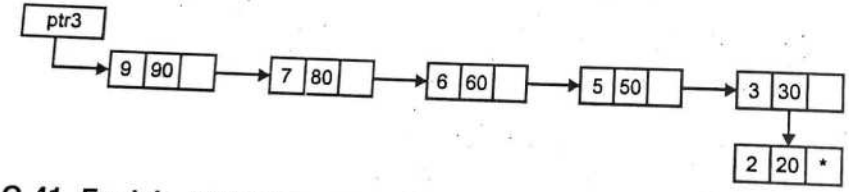
```
}
if(ptr1 = = NULL)
{
for (; ptr2 ! = (node*) NULL; ptr2 = ptr2 →next)
add_node(ptr3, ptr2 →Coef, ptr2 →powe);
}
else if(ptr2 = = NULL)
{
for (; ptr1 ! = NULL; ptr1 = ptr1= ptr1 →next)
add_node (ptr3, ptr1 →Coef, ptr1 →powe);
}
Coef = 9
powe = 90
add(ptr3, 9, 90)
```



Coef = 7
powe = 80
add(ptr3, 7, 80)



Like this we keep continuing and finally we get



Q 41. Explain why binary search cannot be performed on a linked list.

(PTU, May 2015)

Ans. If the list of items is unsorted, we must search the list using a sequential search whether the items are stored in an array or a linked list. If the list is sorted, it is possible to search the array using a binary search. Since binary search requires direct access to an element, we can not perform a binary search on a linked list. The only way to search a linked list is sequential.

Q 42. Write an algorithm to insert new node at the middle of a Singly Linked List.

(PTU, May 2015)

Ans. (a) 1. Get a memory block of type NODE and store the address of it in a pointer variable NEW.

i.e., NEW=GETNODE(NODE)

2. If (New is NULL) then
   Print "Memory is insufficient"
   Exit
3. Otherwise do the following

(a) Move the address of the HEADER node in a pointer variable ptr
   i.e., ptr = HEADER

(b) Get the data item of the Node key after which the new node has to be placed.

(c) Now traverse the single linked list from the HEADER node to find the node which contains key as the data item
   i.e., while(ptr.DATA is not equal to key) and (ptr.LINK is not equal to NULL)

(a) Go to the next node
   i.e., ptr = ptr.LINK

4. If(ptr.LINK is NULL)
   print "Key is not available in the list"
   Exit.
   Else

(a) Make the LINK field of the new node to point the node pointed out by ptr.
   i.e., NEW.LINK = ptr.LINK

(b) Place the data X in the DATA field of the NEW node.
   i.e., NEW.DATA=X

(c) Now Make the LINK field of the ptr node to point the NEW node.

5. Stop.

Q 43. What are the different ways to implement list ? What are the advantages in the array implementation of list ? **(PTU, Dec. 2015)**

Ans. Different ways to implement list : There are two different ways to implement list.

These are :

1. Simple array implementation of list
2. Linked list implementation of list.

Advantages in the array implementation of list :

1. Print list operation can be carried out at the linear time.
2. Find K^{th} operation takes a constant time.

Q 44. What does the following function do for a given Linked List ?

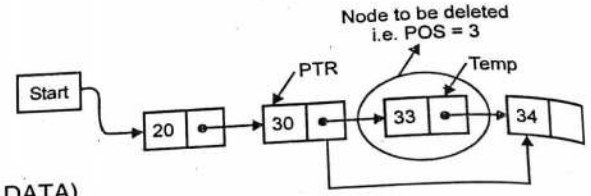
```
void datastructure(struct node*head)
{
If (head = = NULL)
return ;
datastructure(head→next);
printf("%d", head→data);
}
```

(PTU, May 201

Ans. datastructure( ) prints the given Linked List in reverse manner. For Linked List 1→2→3→4→5, datastructure( ) pri
5→4→3→2→ 1.

Q 45. Write an algorithm to delete a specific element in a single linked list. Doubly linked list takes more space than singly linked list for storing one extra address. In what condition could be a doubly linked list be more beneficial than singly linked list. (PTU, May 2016)

Ans. Suppose START is the first position in linked list. Let DATA be the element to be deleted. TEMP, HOLD is a temporary pointer to hold the node address



Deletion of a node

1. Input the DATA to be deleted
2. If (START→DATA) is equal to DATA)
   (a) TEMP = START
   (b) START = START→Next
   (c) Set free the node TEMP, which is deleted.
   (d) Exit.
3. HOLD = START
4. While ((HOLD→Next→Next) not equal to NULL))
   (a) If ((HOLD→NEXT→DATA) equal to DATA)
      (i) TEMP = HOLD→Next
      (ii) HOLD→Next = TEMP→Next
      (iii) Set free the node TEMP, which is deleted
      (iv) Exit
   (b) HOLD = HOLD→Next
5. If ((HOLD→next→DATA) = = DATA)
   (a) TEMP = HOLD→Next
   (b) Set free the node TEMP, which is deleted (C)
   (c) HOLD→Next = NULL
   (d) Exit
6. Display "Data not found"
7. Exit.

In doubly linked list, we can traverse list both i.e., forward and backward. The major advantage of doubly linked lists is that they make some operations (like the removal of a given node, or a right-to-left-traversal of the list) more efficient.

Q 46. What is a doubly-linked list ? Write an algorithm to create a doubly-linked list and also write a function to insert a node in doubly-linked list. (PTU, May 2016)

Ans. Doubly-linked list : *Refer to Q.No. 4 & 13*

The algorithm for creation of doubly linked list is given below :

1. Take a new node in pointer called first.
2. For→left Link = NULL.
3. Read Data (First)
4. back = First
5. Bring a new node in the pointer called for.
6. Read Data (Far)
7. back→right Link = Far

8. Far→left Link = Back
9. back = Far
10. Repeat step 5 to 9 till whole of the list is constructed.
11. For→right List = NULL
12. Stop.

Q 47. What are doubly linked lists and what are their advantages ? Write a progra to insert and delete a node in a doubly linked list. (PTU, Dec. 20

Ans. Doubly linked lists : *Refer to Q.No. 4*

Advantages : *Refer to Q.No. 25*

Program :

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
struct node
{
int data;
struct node *next, *prev;
}*New, *newl, *temp, *start, *dummy;
void add(void);
struct node * get_node( );
void display(void);
void delet(void);
int find(int);
int first = 1;
void main( )
{
char ans;
int choice, num, found = 0;
start = NULL;
do
{
clrscr( );
Printf("\n\t Program for Doubly Linked List \n");
Printf("\n\n 1. Insertion of Element \n\n");
Printf("2. Delection of element from the List\n\n");
Printf("3. Display of Doubly Linked List \n\n");
Printf("4. Searching of Particular node \n\n");
Printf("5. Exit");
Printf("\n Enter your choice");
scanf("%d", &choice);
switch(choice)
{
case 1 : add( );
break;
case 2 : delete( );
break;
case 3 : display( );
```

```
    break;
    case 4 : printf("\n Enter the number which is to be searched"):
    scanf("%d", &num);
    temp = start ;
    while((temp != NULL) && (found == 0))
    found = find(num);
    if(found)
    printf("\n The number is present");
    else
    printf("\n The number is not present");
    break;
    case 5 : exist(0);
    }
    printf ("\n Do you want to continue ?\n");
    ans = getche( );
    }while(ans == "y" || ans == 'y');
    getch( );
    }
    void add (void)
    {
    clrscr( );
    New = get_node( );
    printf("\n\n\n\t Enter the element\n");
    scanf("%d", &New→data);
    if(first==1)
    {
    start = New;
    first = 0;
    }
    else
    {
    dummy = start;
    while(dummy→next ! = NULL)
    dummy = dummy→next;
    dummy→next = New;
    New→prev = dummy;
    }
    }
    struct node *get_node ( )
    {
    new 1 = (node *) malloc (size of(struct node));
    new 1→next = NULL;
    new 1→ prev = NULL'
    return(new1);

    void display(void)
```

```
clrscr( );
temp = start;
if (temp == NULL)
printf("\n The Double Linked List is Empty\n");
else
{
while (temp != NULL)
{
printf ("%d ⇒ ", temp→data);
temp = temp→next;
}
printf("NULL");
}
getch( );
}
int find (int num)
{
if (temp→data == num)
return(1);
else
temp = temp > next;
return 0;
}
void delet (void)
{
int num, flag = 0;
int found;
int last = 0;
clrscr( );
temp = start;
if (temp== NULL)
printf("\n\n Sorry : DLL not created!\n");
else
{
printf("Enter the number to be deleted");
scanf ("%d", &num);
while((flag = = 0) & & (temp != NULL)
{
found = find(num);
flag = found;
}
if (found == 0)
printf ("\n Number not found");
else
{
if(temp == start)
{
start = start→next;
```

```
temp→next = NULL;
start→prev = NULL;
free(temp);
getch( );
printf("\n The starting node is deleted");
}
else
{
if (temp→next == NULL)
last = 1;
else
last = 0;
(temp→next) → prev = temp → prev;
(temp→prev) → nex = temp → next;
temp→prev = NULL;
temp→next = NULL;
free(temp);
if(last)
printf("\n The last mode is deleted");
else
printf("\n The intermediate node is deleted");
}
}
}
}
```

Q 48. Write an algorithm to insert new node at the end of a Double Linked List. **(PTU, May 2017)**

Ans. Insert (value)

1. Start
2. Set PTR = addressof (New node)
3. Set PTR→INFO = value;
4. If FIRST = NULL, then go to step 5 else goto step 7
5. Set FIRST = PTR and LAST = PTR
6. Set PTR→NEXT = PTR→PREVIOUS = NULL and goto step 8
7. Set LAST→NEXT = PTR, PTR→PREVIOUS = LAST, PTR→NEXT = NULL, and LAST = PTR
8. STOP.

Q 49. Define header nodes. **(PTU, May 2018)**

Ans. A header linked list is a linked list which always contains a special node called the header node at the beginning of the list.

Q 50. How pointers are used to manage address of memory ? **(PTU, May 2019)**

Ans. A pointer is a programming language object that stores the memory address of another value located in computer memory. A pointer references a location in memory and obtaining the value stored at that location is known as dereferening the pointer.

Q 51. What is dangling pointer give example ? **(PTU, May 2019)**

Ans. A dangling pointer arises when a program uses a memory resource after it has been freed.

❑❑❑

Chapter 3B

Trees

Contents

Basic Tree Terminology, Different types of Trees : Binary Tree, Threaded Binary Tree, Binary Search Tree, AVL Tree ; Tree operations on each of the trees and their algorithms with complexity analysis. Applications of Binary Trees. B Tree, B+ Tree : Definitions algorithms and analysis.

POINTS TO REMEMBER

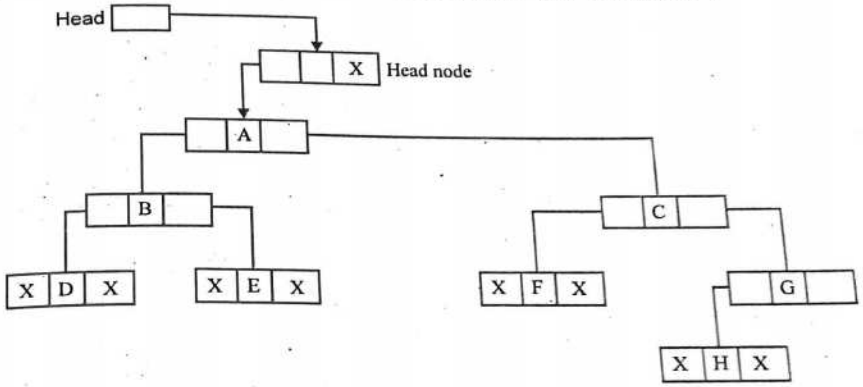
- A tree T is a finite set of nodes such that there is a special node called the root from which the remaining nodes can be partitioned into zero, one or more disjoint subsets T1, T2, ... Tn (N > O) each of which itself is a tree known as subtree of T.
- A node of tree stores the data element and may contain zero, one or more link (s) to other successor node (s) for connectivity. A node can be a parent, child or both.
- A directed line from one node to other successor node is called an edge of a tree.
- Two or more node with same parent are called siblings.
- The node that has no parent is called the root and the node that has no child is called the leaf node.
- A path is a resulting sequence of nodes when we traverse from one node to other node along the edges that connect them.
- The level of a node is an integer value that measures the distance of a node from the root
- The maximum number of children that are possible for a node is known as the degree o variety of node.
- The height of a node is an integer value that measures the distance of a node from th root.
- The depth of a node is a the length of unique path from the node to the root of the tr
- A binary tree is a tree in which no node can have more than two children.
- A binary is a full binary tree if each node has exactly zero or two children.
- A complete binary tree is a tree which is either full binary tree or one in which every l is fully occupied except possibly for bottomost level where all the nodes must be a left as possible.

- A binary tree traversal visits each node of the tree exactly once in a predetermined sequence.
- Breadth-first traversal and depth-first traversal are the two traversal approaches of a binary tree.
- Using depth first approach, we may traverse a binary tree in six different sequences, however, only three of these sequences namely preorder, inorder and postorder are the standard ones.
- In the preorder traversal of binary tree, we first process the root, followed by the left subtree and then the right subtree.
- In the inorder traversal of binary tree, we begin by first processing the left subtree, followed by the root node and then the right subtree of the root provided subtrees are not empty.
- In the post-order traversal of binary tree, we process the left subtrees, followed by the right subtree and then the root.
- An expression tree is a binary tree which is used to represent an arithmetic expression.
- A binary search tree is a binary tree whose node are arranged in such a way that for every node N, the values contained in all the nodes in its left subtree are less than value contained in node N and the values contained in all the nodes in its right subtree are greater than or equal to the value contained in node N.
- The inorder traversal of a BST produces an ordered list.
- To insert a node in a BST, we follow the left or right branch down the tree, depending on the value of new node, until we find a null subtree.
- A heap tree (max tree) is a complete binary tree in which data values stored in any node is greater than or equal to the value of its children.
- Insertion and deletion are the key operations performed on a heap tree.
- As heap tree is a complete binary tree so space is used efficiently when it is implemented as an array.
- Priority queues and heap sort are the key applications of heap tree.
- The balance factor of a node of binary tree is the difference in the height of its left and right subtrees.
- A binary tree is balanced if the height of its subtrees differ by no more than 1 and its subtrees are also balance.
- An AVL tree is a BST in which each node has a balance factor of +1, 0, or –1.
- In LL rotation, unbalanced AVL tree can be balanced by rotating the out of balance node to the right.

QUESTION-ANSWERS

Q 1. What are threads? **(PTU, Dec. 2004)**

Ans. Consider the linked representation of a binary tree T as below :



Approximately half of the entries in the pointer fields LEFT and RIGHT will contain null elements. This space may be more efficiently used by replacing null entries by some other type of information. Specifically, we will replace certain null entries by special pointers which point to nodes higher in tree. These special pointers are called threads.

Q 2. What is depth of a complete binary tree? **(PTU, May 2005 ; Dec. 2004)**

Ans. The depth or height of a tree 'T' is the max. no. of nodes in a branch of a tree 'T'. This is one more than maximum level no. of tree. The depth d_n of a complete binary tree with 'n' nodes is given by

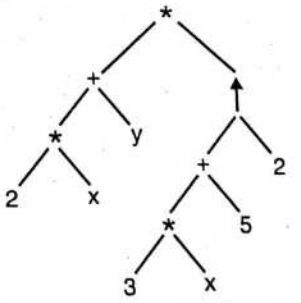
$$d_n = \log_2 n + 1$$

Q 3. Draw a tree corresponding to expression $(2x + y)(3x + 5)^2$.

(PTU, Dec. 2005)

Ans. The prefix expression of above tree is :

* + * 2 x y ↑ + * 3 x 52



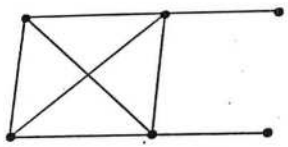
Q 4. What is a spaning tree? **(PTU, May 2006)**

OR

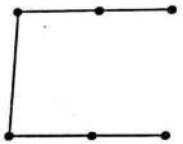
What do you mean by spaning tree? **(PTU, May 2007)**

Ans. *A spaning tree is a tree 'T', of a graph 'G', such that 'T' is connected and it contains all the vertices as in 'G' and it should contain no cycles.*

e.g.



'G'

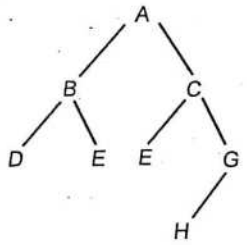


Spaning Tree 'T'

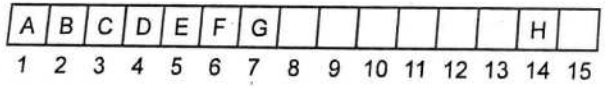
Q 5. How trees are represented in memory using arrays? **(PTU, May 2007)**

Ans. *In case of array representation of binary trees, the root 'R' of tree 'T' is stored in TREE [1]. If a node 'n' occupies TREE [K], then, it's left child is stored in TREE [2*K] and its right child is stored in TREE [2* K+1].*

e.g.



The above tree has sequential representation as :

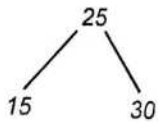


Q 6. What is binary search tree? Give example. **(PTU, May 2007)**

Ans. *A tree 'T' is called a Binary Search tree if each node 'N' of 'T' has the following properties :*

The value at 'N' is greater than every value in the left sub-tree and is less than every value right sub tree of 'N'.

e.g.



is a B.S.T. (Binary Search Tree)

Q 7. Discuss the sequential memory representation of binary trees. **(PTU, Dec. 2007)**

Ans. Linked Repersentation of Binary Trees : Consider a binary tree T. Unless otherwise stated or implied, T will be maintained in memory by means of a linked representation which uses three parallel arrays, INFO, LEFT and RIGHT pointer variable ROOT as follows :

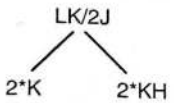
First of all each node N of T corresponds to location K such that

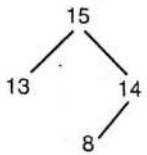
1. INFO [K] contains the data at the node N.
2. LEFT [K] contains the location of left child of node N.
3. RIGHT [K] contains the location of right child of node N.

Further more, ROOT will contain the location of root R of T. It may sub-tree is empty, then corresponding pointer will contain null value ; it the tree T is empty, then ROOT will contains NULL value.

Q 8. There are 8, 15, 13, 14 nodes were there in 4 different trees. Which of them could have formed a full binary tree. **(PTU, Dec. 2008)**

Ans. T : 8, 15, 13, 14





Q 9. What are siblings? **(PTU, Dec. 2009)**

Ans. Siblings are the nodes of the tree having the same father or parent node. For example, suppose N is a node in a tree T with left successor S_1 and right successor S_2. Then N is called the parent of S_1 and S_2. S_1 is called the left child or son of N_1 and S_2 is called the right child or son of N. S_1 and S_2 are said to be siblings or brothers.

Q 10. What is the best and average case of binary search? **(PTU, Dec. 2010)**

Ans. The binary search algorithm is a very efficient algorithm, it has some majo drawbacks. Specifically, the algorithm assumes that one has direct access to the middl name in the list or a sublist. This means that the list must be stored in some type of array Unfortunately, inserting an element in an array requires elements to be moved down the lis and deleting an element from an array requires element to be moved up the list.

The telephone company solves the above problem by printing a new directory eve year while keeping a separate temporary file for new telephone customers. That is, t telephone company updates its files every year. On the other hand, a bank may want insert a new customers in its files almost instantaneously. According, a linearly sorted l may not be the best data structure for a bank.

Q 11. What is tree? **(PTU, May 2006)**

OR

What is a binary tree? **(PTU, Dec. 2010, 2006)**

Ans. A binary tree T is defined as a finite set of elements, called nodes, such that ;

(a) T is empty (called the null tree or empty tree), or

(b) T contains a distinguished node R, called the root of T, and the remaining nodes of T form an ordered pair of disjoint binary trees T_1 and T_2.

If T does contain a root R, then the two trees T_1 and T_2 are called, respectively, the left and right subtrees of R. If T_1 is non empty, then its root is called the left successor of R ; similarly, if T_2 is nonempty, then its root is called the right successor of R.

Q 12. What is traversal method of a threaded binary tree? **(PTU, Dec. 2010)**

Ans. There are many ways to thread a binary tree T, but each threading will corresponding to a particular traversal of T. Also, one may choose a one-way threading or a two-way threading. Unless otherwise stated, our threading will correspond to the inorder transversal of T. Accordingly, in the one-way threading of T, a thread will appear in the right field of a node and will point to the next node in the inorder transversal of T ; and in the two-way threading of T, a thread will also appear in the LEFT field of a node and will point to the preceeding node in the inorder transversal of T.

Q 13. What is a AVL tree? **(PTU, May 2019 ; Dec. 2010)**

Ans. An empty binary tree is a an AVL tree. A non empty binary tree T is an AVL tree iff given T^L and T^R to be the left and right subtrees of T and h (T^L) and h (T^R) to be the heights of subtrees T^L and T^R respectively. T^L and T^R are AVL trees and \h (T^L) – h (T^R) ≤ 1.

h (T^L) – h (T^R) is known as the balance factor (BF) and for an AVL tree the balance factor of a node can be either 0, 1 or –1.

An AVL search tree is a binary seach tree which is an AVL tree.

Q 14. Write an algorithm for pre order, inorder and posorder traversal in a tree. **(PTU, May 2004)**

Ans.

1. PREORD (INFO, LEFT, RIGHT, ROOT)

1. Set TOP = 1, STACK [1] = NULL & PTR = ROOT
2. Repeat steps 3 to 5 while PTR ≠ NULL
3. Apply PROCESS TO INFO [PTR]
4. If RIGHT [PTR] ≠ NULL, them :
   Set TOP = TOP + 1, STACK [TOP] = RIGHT [PTR]
5. If LEFT [PTR] ≠ NULL, them :
   Set PTR : LEFT [PTR]
   ELSE
   Set PTR = STACK [TOP], TOP = TOP – 1
6. Exit.

2. INORD (INFO, LEFT, RIGHT, ROOT)

1. Set TOP = 1,STACK [1] = NULL & PTR = ROOT
2. Repeat while PTR ≠ NULL
   (a) Set TOP = TOP + 1 & STACK [TOP] = PTR
   (b) Set PTR = LEFT [PTR]
3. Set PTR = STACK [TOP] & TOP = TOP – 1
4. Repeat steps 5 yo 7 while PTR ≠ NULL
5. Apply Process to INFO [PTR]
6. If RIGHT [PTR] ≠ NULL, them :
   (a) Set PTR = RIGHT [PTR]
   (b) Go to step 2
7. Set PTR = STACK [TOP], TOP = TOP – 1
8. Exit.

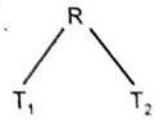
3. POSTORD (INFO, LEFT, RIGHT, ROOT)

1. Set TOP = 1, STACK [1] = NULL, PTR = ROOT
2. Repeat steps 3 to 5 while PTR ≠ NULL.
3. Set TOP = TOP + 1 & STACK [TOP] = PTR
4. If RIGHT [PTR] ≠ NULL, them
   Set TOP = top + 1 & STACK [TOP] = –RIGHT [PTR]
5. Set PTR = LEFT [PTR]
6. Set PTR = STACK [TOP] & TOP = TOP – 1
7. Repeat while PTR > 0 :
   (a) Apply Process to INFO [PTR]
   (b) Set PTR = STACK [TOP], TOP = TOP – 1
8. If PTR < 0, then
   (a) Set PTR : = –PTR
   (b) Go to step 2
9. Exit.

Q 15. What is tree? Explain the type branches of a tree. **(PTU, Dec. 200**

Ans. A binary tree (T) is defined as a finite set of elements called nodes, s.t.

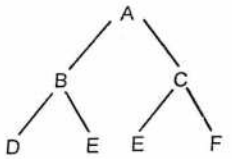
1. 'T' is empty.
2. 'T' contains a distinguished node 'R' called root of tree 'T' and the remaining no of 'T' forms an ordered pair of disjoint binary trees T_1 and T_2. T_1 and T_2 are ca left and right sub trees of 'R'. Any node 'N' in a binary tree has either 0, 1 sucessors.



Types of trees :

1. Complete binary tree : The tree 'T' is said to be complete if all its levels except possibly the last have the maximum no. of possible nodes and all the nodes at last level appear as far as possible, i.e. in complete binary tree, each node of a tree can have almost two nodes and level 'r' of 'T' can have atmost 2^r nodes.

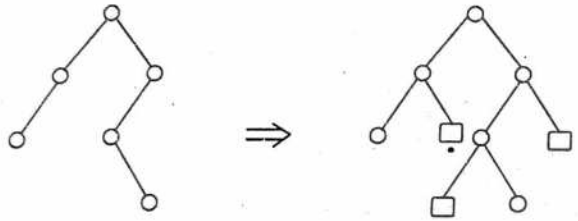
e.g.



The depth D_n of complete tree T_n with 'n' nodes is given by

$$D_n = [\log_2 n + 1]$$

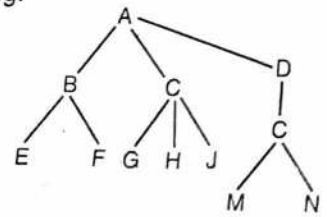
2. Extended binary tree : A binary tree 'T' is said to be a true tree or extended binary tree if each node 'n' has either 0 or 2 children. The nodes with 2 childrens are called internal and nodes with 0 childrens are called external nodes.



3. General tree : A general tree is defined to be non empty finite set of elements called nodes s.t.,

1. T contains a distinguished element 'R' called root of tree.
2. The remaining element of 'T' form an ordered collection of zero or more disjoint trees $T_1, T_2, \ldots\ldots, T_n$.

e.g.



Q 16. Write short notes on :

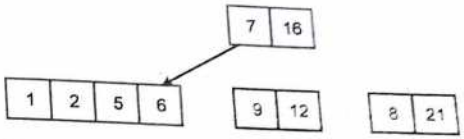
(a) B-Trees

(b) AVL Search Trees

(c) M-way search Trees **(PTU, Dec. 2011)**

Ans. (a) B-trees : In computer science, a B-tree is a tree data structure that keeps data

sorted and allows searches, sequential access, insertion and deletion in algorithm time. The B-tree is a generalization of binary search tree in that a node can have more than two children.



The database problem and B-tree solves the problem completely.

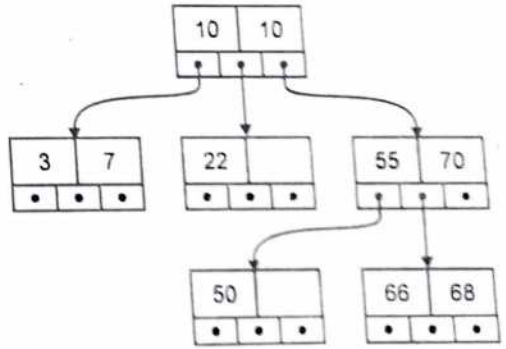
1. Time to search a sorted file.
2. An index speeds the search.
3. Insertion and deletion cause trouble.
4. The B-tree uses all those ideas.

(b) AVL Search Trees : In computer science, an AVL tree is a self-balancing binary search tree, and it was the first such data structure to be invented.[1] In an AVL tree, the heights of the two child subtrees of any node differ by at most one. Lookup, insertion, and deletion all take O (log n) time in both the average and worst cases, where n is the number of nodes in the tree prior to the operation. Insertions and deletions may require the tree to be rebalanced by one or more tree rotations.

(c) M-way search Trees : M-way Search Tree : A binary search tree has one value in each node and two subtrees. This notion easily generalizes to an M-way search tree, which has (M – 1) values per node and M subtrees. M is called the degree of the tree. A binary search tree, therefore, has degree 2. In fact, it is not necessary for every node to contain exactly (M – 1) values and have exactly M subtrees. In an M-way subtree a node can have anywhere from 1 to (M – 1) values, and the number of (non-empty) subtrees can range from 0 (for a leaf) to 1 + (the number of values). M is thus a fixed upper limit on how much data can be stored in a node.

The values in a node are stored in ascending order, V1 < V2 < ... Vk (k < = M – 1) and the subtrees are placed between adjacent values, with one dimensional subtree at each end. We can thus associate with each value a 'left' and 'right' subtree, with the right subtree of V (i) being the same as the left subtree of V (i + 1). All the values in V1's left subtree are less than V1 ; all the values in Vk's subtree are greater than Vk ; and all the values in the subtree between V (i) and V (i + 1) are greater than V (i) and less than V (i + 1).

For example, here is a 3-way search tree :



In our examples, it will be convenient to illustrate M-way trees using a small value of M. But bear in mind that, in practice, M is usually very large. Each node corresponds to a physical block on disk, and M represents the maximum number of data items that can be stored in a single block. M is maximized in order to speedup processing : to move from one node to another involves reading a block from disk – a very slow operation compared to moving around a data structure stored in memory.

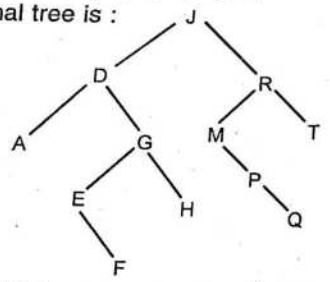
Q 17. Design a B.S.T. from following sequence J R D G T E M H P A F Q and perform operation in sequence. **(PTU, May 2007)**

(i) Node J is deleted (ii) Node S is inserted.

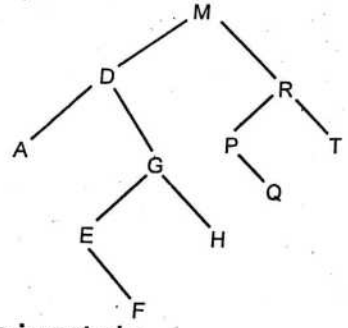
Ans. The in order traversal and 'T' is :

A, D, E, F, G, H, J, M, P, Q, R, T

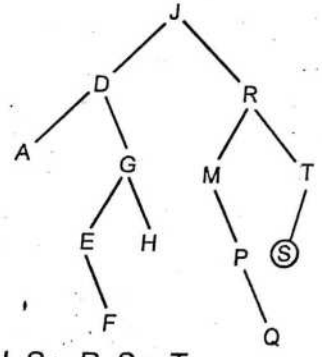
Now, final tree is :



(i) Node 'J' is deleted : Here, 'J' is root node and has 2 children. It is deleted by replacing it with its inorder success i.e. M and M is replaced by P.

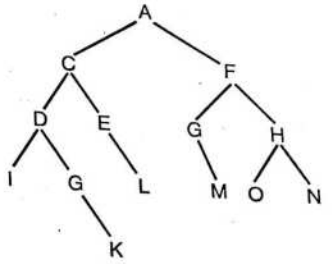


(ii) Node S is inserted :



Here, S > J, S > R, S > T.

Q 18. For the following tree write the preoder inorder and postorder traversals. **(PTU, Dec. 2007)**



Ans. The preorder traversal of tree T is

A C D I G K E L F G M H O N

The inorder traverse of tree T is

I D G K C E L A G M F O H N

The post order traversal of tree T is

I K G D L E C M G O N H F A

Q 19. Discuss different ways of representing a binary tree and suggest an application for each of the representations. **(PTU, Dec. 2009)**

Ans. There are two ways of representing a binary tree in the memory :

1. Sequential representation using arrays
2. Linked represetation

1. Array Representation : An array can be used to store of nodes of a binary tree. The nodes stored in an array of memory can be accessed sequentially.

Suppose T is a binary tree that is complete or nearly complete.

Then an efficient way of sequential representation of T is a linear array Tree as follows:

1. The root of R of T is stored in TREE [1].

2. If a node N occupies TREE [K], then its left child is stored in TREE [2*K] and its right child is stored in TREE [2* K + 1]

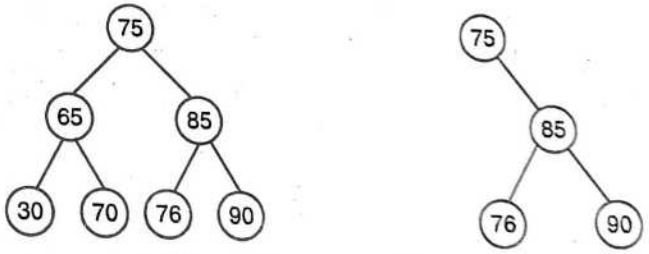


Fig. 1 Binary search tree

1	2	3	4	5	6	7
75	65	85	30	70	76	90

(a) Array representation of binary search tree of fig. 1 (a)

1	2	3	4	5	6	7
75		85			76	90

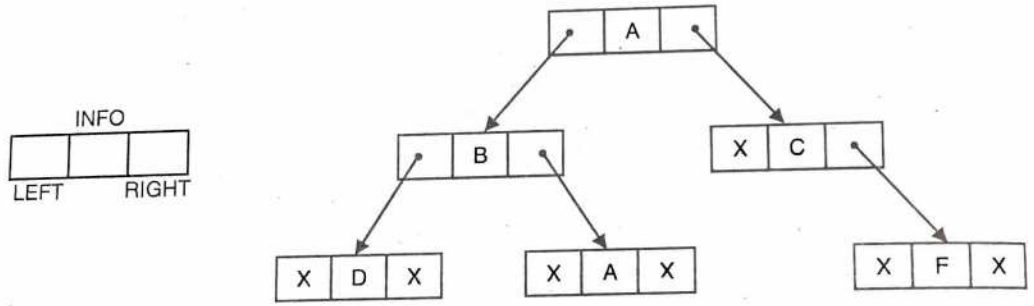
(b) Array representation of binary search tree of fig. 1 (b)

Generally speaking, the sequential representation of a tree with depth d will require an array with approximately 2^{d+1} elements. So, this sequential representation is usually in efficient unless the binary tree T is complete or nearly complete.

2. Linked Representation : The most popular and practical way of representing a binary tree is using linked list. In linked list, every element is represented as nodes. Each node consists of three fields such as :

(a) Left child [LEFT]
(b) Information of the node [INFO]
(c) Right child [RIGHT]

The LEFT links to the left child of the parent node, INFO holds the information of every node and RIGHT holdes the address of right child of the parent node.



(a) Node **(b) Binary Tree**

Fig. 2 Node and linked representation of binary tree

If a node has not left or/and right node, corresponding LEFT or RIGHT is assigned to ULL.

Q 20. Write program to traverse a binary tree in preorder and postorder fashion. (PTU, May 2010)

Ans. If a node has left or/and right node, corresponding L child or R child is NULL. The ode structure can be logically represented in C/C++ as :

```
struct node
{
    int Info ;
    struct node *Lchild ;
    stuct node *Rchild ;
} ;
typedef struct node*NODE ;
```


Pre-order Traversal : To tranverse a non-empty binary tree in pre order, following steps one to be processed :

1. Visit the root node
2. Traverse the left sub tree in preorder
3. Traverse the right sub tree in preorder.

It can be implemented in C/C++ function as below :

```
void preorder (NODE * Root)
{
    if (Root != NULL)
    printf ("%d\n", Root → info) ;
    preorder (Root → Child) ;
    preorder (Root → Child) ;
    }
}
```

Post-order Traversal : The post order traversal of a non-empty binary tree can be defined as:

1. Traverse the left sub tree in post order.
2. Traverse the right sub tree in post order.
3. Visit the root node.

In post order traversal, the left and right sub tree (s) are recursively processed before visiting the root.

```
void postorder (NODE * Root)
{
    if (Root ! = NULL)
    {
        postorder (Root → Lchild) ;
        postorder (Root → Rchild) ;
        printf ("%d\n", Root → info) ;
    }
}
```

Q 21. Suppose a binary tree T is in the memory. Write a recursive algorithm or a program which find the number of nodes in T and which find the depth of T. (PTU, Dec. 201

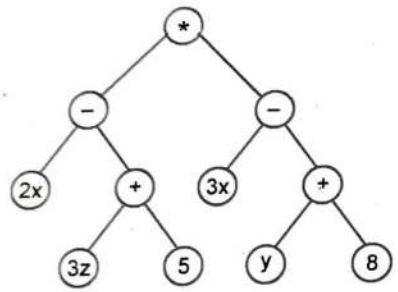
Ans. The number of nodes in any subtree is the number of nodes in its left subtre plus the number of nodes in its right subtree, plus one, so you can use a recursive algorit and start at the root.

```
Unsigned int binary tree_count_recursive (const node * root)
{
unsigned int count = 0 ;
if (root! = NULL) {
```

```
count = 1 + binary tree_count_recursive (root -> left)
+ binary tree_count_recursive (root -> right) ;
}
    return count ;
}
```

Q 22. Construct the binary tree for the following expression.
(2x – 3z + 5) (3x – y + 8) **(PTU, Dec. 2010)**

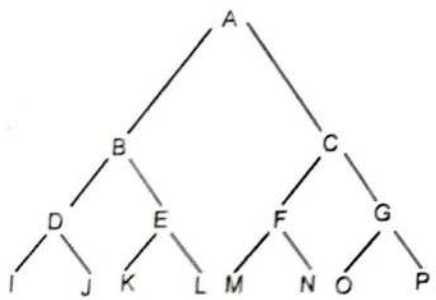
Ans. Construct the binary tree for the following expression :
(2x – 3z + 5) (3x – y + 8)



Q 23. When is a tree said to be a complete binary tree? How is it different from extended binary tree? **(PTU, Dec. 2004)**

Ans. The tree 'T' is said to be complete if all of its levels except possibly the last have the maximum no. of possible nodes and if all the nodes at the last level appear as far as possible.

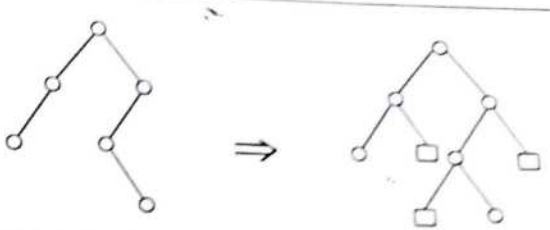
e.g.



The depth 'D_n' of complete the with 'n' nodes is given by,

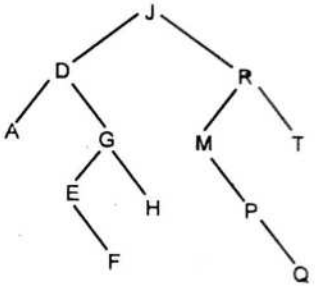
$$D_n = \log_2 n + 1$$

A binary tree 'T' is said to be a true tree or extended binary tree if each node 'n' has her 0 or 2 children. The nodes with 2 children are called internal nodes and nodes with 0 ldrens are called external nodes.



Q 24. Suppose the following list of letters are to be inserted into an empty binary search tree : J R D G T E M H P A F Q. Find the final tree. (PTU, Dec. 2004)

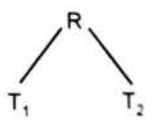
Ans.



Q 25. What are binary trees? Explain sequential representation of binary trees. (PTU, May 2005)

Ans. A binary tree 'T' is defined as a finite set of elements called nodes s.t.,

1. 'T' is empty (it is called null or empty tree)
2. 'T' contains a distinguished node 'R' all the root of tree 'T' and remaining nodes of 'T' forms an ordered pair of disjoint binary trees 'T_1' and 'T_2'. T_1 and T_2 are called left and right sub trees of 'R'. Any node 'N' in a binary trees has either 0, 1 or 2 successors.



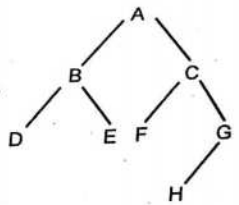
The nodes with no successors are called terminal nodes.

Sequential representation of trees :

Root 'R' of 'T' is stored in TREE [1]. If a node 'n' occupies TREE [K], then, it's left child i stored in TREE [2*K] and its right child is stored in TREE [2* K + 1].

TREE K =

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	B	C	D	E	F	G							H	



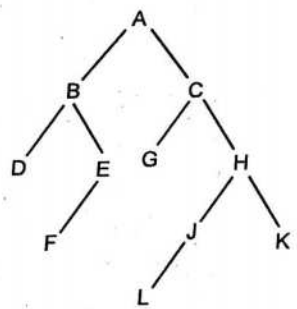
Q 26. A binary tree has 11 nodes. Inorder and postorder traversals are given below :

Inorder : DBFEAGCLJHK

Postorder : DFEBGLJKHCA

Draw the tree. **(PTU, May 2005)**

Ans.



Q 27. A binary tree has 9 nodes. The inorder and preorder traversal sequences are given below :

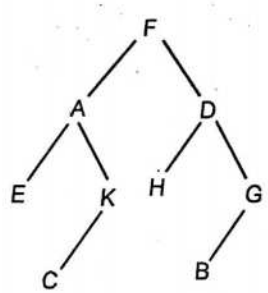
Inorder : E A C K F H D B G

Preorder : F A E K C D H G B

Draw the tree. **(PTU, Dec. 2005)**

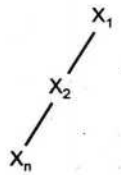
Ans. Here, Inorder : E A C K F H D B G

Preorder : F A E K C D H G B



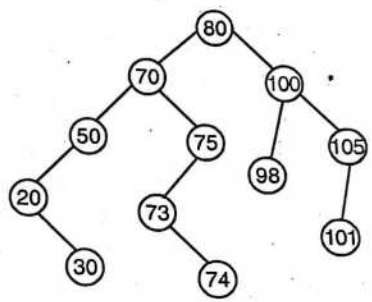
Q 28. Suppose that 'n' data elements X_1, X_2, ..., X_n are sorted in descending order, these elements are to be inserted into an empty binary search tree. Generate the final tree. What will be the depth of this tree? **(PTU, Dec. 2005)**

Ans. The tree will consist of one branch which extends as follows :



Since T has a branch with all n nodes, thus, depth, D = n.

Q 29. Write a procedure for inorder traversal of a binary tree. What will be inorder and post order traversals of following binary tree? **(PTU, May 2008)**



Ans. The procedure of inorder traversal in :

1. Tranverse the left subtree of R in inorder.
2. Process the root R.
3. Traverse the right subtree of R in inorder.

Preorder travesal

80, 70, 50, 20, 30, 75, 73, 74, 100, 98, 105, 101

Inorder traversal

20, 30, 50, 70, 73, 74, 75, 80, 98, 100, 101, 105

Post order traversal

30, 20, 50, 74, 73, 75, 70, 98, 101, 105, 100, 80

Q 30. What are the different ways for traversing a binary tree. Draw a binary tre for the following algebraic expression :

[a + (b – c)] * [(d – e)/ (f + g – h)]

Explain pre order and post order traversals of the binary tree (by using examp of constructed binary tree for the above expression). **(PTU, May 2011 ; Dec. 200**

Ans. Let E denote the following expression :

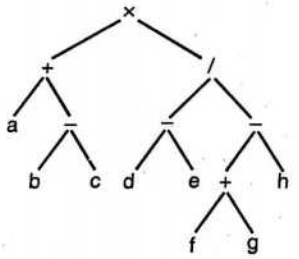
E : [a + (b – c)] * [(d – e)/ (f + g – h)]

The corresponding binary tree T appear in diagram given below. One can varify the inspection that the preorder and post order travesales of T are as follows :

(pre order) * + a – b c / – de – + f g h

(post order) a b c – + d e – f g + h – / *

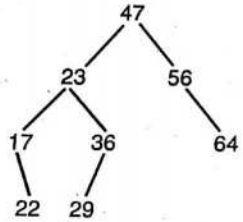
one can verify that these orders corresponds precisely to prefix and post fix polish notation of E.



Tree representation of E (expression)

Q 31. Construct a binary search tree to accommodate the given list of integers. 47, 56, 23, 17,64, 36, 29, 22. **(PTU, May 2009)**

Ans. 47, 56, 23, 17,64, 36, 29, 22



Q 32. Find the order, preorder and post order sequence of nodes of the above tree. **(PTU, May 2009)**

Ans.

Preorder

47 23 17 22 36 29 56 64

Inorder

17 22 23 29 36 47 56 64

Post order

22 17 29 36 23 64 6 47

Q 33. Write C functions which take a pointer to the binary tree T and compute the following :

The number of leaves in T.

The number of nodes in T that contain one NON Null child.

The number of nodes in tree that contain exactly two non null Children. **(PTU, May 2010)**

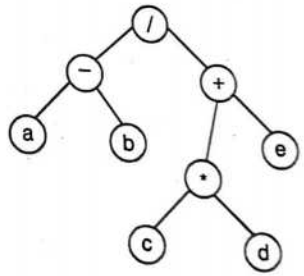
Ans. The number of leaf nodes in a tree is equal to the sum of leaf nodes in the left subtree and leaf nodes in the right subtree of a given node. Note that if the binary (search) tree is empty than the number of non-leaf is zero and if there is only one node in the tree, then the number of leaf nodes is equal to 1.

The following function in C language shows the implementation of various steps required.

```
typedef struct BST_node {
struct BST_node*left ;
int item ;
struct BST_node *right
} BST ;
int leaf Nodes (BST *tree)
{
    if (tree = = NULL)
        return 0 ;
    else if ((tree → left = = (BST*) NULL) && (tree → right = = (BST*) NULL))
        return 1 ;
    else
        return (leafnodes (tree → left) + leafnodes (tree → right)) ;
}
```

Q 34. Draw binary tree for (a – b) / ((c * d) + e)) and find out the inorder, preorder and post order traversals. **(PTU, Dec. 2007)**

Ans. The expression tree for the expression (a – b) / ((c*d) + e))



The traversals of the above expression tree gives the following result.

Preorder : (/ – ab + * cde)

This expression is same as the prefix notation of the original expression.

Inorder : (a – b) / ((c * d) + e)

Thus in order traversal gives the actual expression.

Thus the postorder traversal of this gives us the "Postfix notation" or the "Reverse polish notation" of original expression.

Q 35. What are the various binary tree traversal techniques? Discuss with exampl and algorithm. **(PTU, May 2019 ; Dec. 201**

Ans. Traversing Binary Trees : There are three standard ways of traversing a bina tree T with root R. These three algorithms, called preorder, inorder and postorder are as follow

Preorder :

1. Process the root R.
2. Transverse the left subtree of R in preorder.
3. Transverse the right subtree of R in preorder.

Inorder :

1. Traverse the left subtree of R in inorder.
2. Process the root R.
3. Tranverse the right subtree of R in inorder.

Postorder :

1. Transverse the left subtree of R in postorder.
2. Tranverse the right subtree of R in postorder.
3. Process the root R.

Observe that each algorithm contains the same three steps, and that the left subtree of R is always transversed before the right subtree. The difference between the algorithms is the time at which the root R is processed. Specifically, in the "pre" algorithm, the root R is processed before the subtrees are traversed, in the "in" algorithm, the root R is processed between the traversals of the subtrees and in the "post" algorithm, the root R is processed after the subtrees are traversed.

The three algorithms are sometimes called, respectively, the node-left-right (NLR) transversal, the left-node-right (LNR) traversal and the left-right-node (LRN) traversed.

Observe that each of the above traversal algorithms is recursively defined, since the algorithm involves traversing subtrees in the given order. Accordingly, we will expect that a stack will be used when the algorithms are implemented on the computer.

Example : Let E denote the following algebric expression :

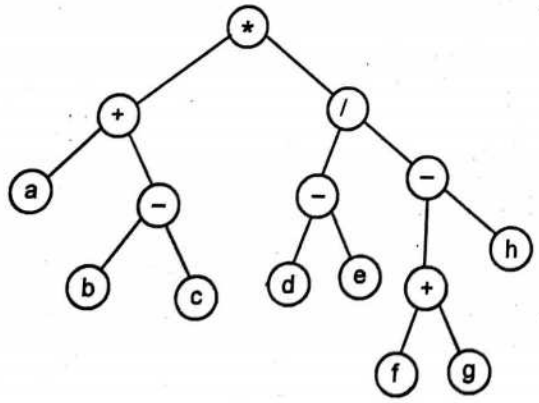
[a + (b – c)] * [(d – e) / (f + g – h)]

The corresponding binary tree T appears in fig. The reader can verify by inspection that e preorder and postorder traversals of T are as follows :

(Preorder) * + a – b c / - d e – + f g h

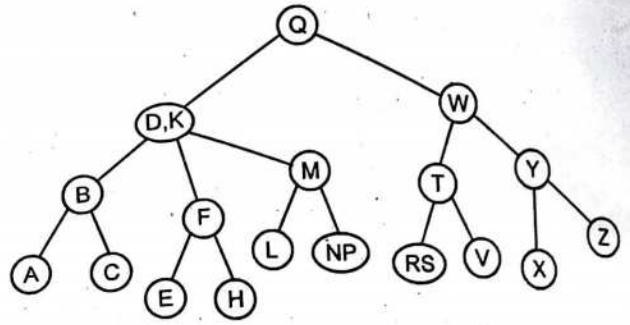
(Postorder) a b c – + d e – f g + h – / *

The reader can also verify that these orders corresponds precisely to the prefix and stfix polish notation of E as discussed in sec. We emphasize that this is true for any algebric ression E.



Q 36. Show the result of inserting F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B, X, Y, D, Z, E in the order to an empty binary tree of degree 3. (PTU, Dec. 2010)

Ans.



Q 37. What is complexity of binary search algorithm? (PTU, May 2011)

Ans. Average case performance O(log n)
Worst case performance O(log n)
Best case performance O(1)
Worst case space complexity O(1)

Q 38. Write an algorithm for binary search. What are its limitations?(PTU, May 2011)

Ans.

1. Initialize an ordered array, search array searchno.length
2. Initialize low = 0 and high = length
3. Repeat step 4 will low < = high.
4. Middle = (low + high) 12.
5. If search arrays [middle] = searchno
   Search is successful
   return middle
   else if
   search array [middle] > search no [high]
   high = middle – 1
   else
   low = middle + 1.

Q 39. How searching is beneficial using binary search trees? (PTU, Dec. 201

Ans. Binary search is an extremely efficient algorithm. A binary search tree is a bina tree in which the data in the nodes is ordered in a particular way.

Q 40. What do you mean by traversal? Also explain various ways of traversal trees. (PTU, Dec. 201

Ans. Tree Traversal refers to the process of visiting (examining and/or updating) ea node in a tree data structure, exactly once, in a systematic way. Such traversals are classifi by the order in which the nodes are visited. The are two types of traversal in trees are :

1. DFS
2. BFS

1. Depth-first search (DFS) is an algorithm for traversing or searching a tree, tree structure, or graph. One starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking.

2. Binary Tree : To traverse a non-empty binary tree in **preorder**, perform the following operations recursively at each node, starting with the root node :

1. Visit the root
2. Traverse the left subtree
3. Traverse the right subtree

To traverse a non-empty binary tree in **inorder** (symmetric), perform the following operations recursively at each node :

1. Traverse the left subtree
2. Visit the root
3. Traverse the right subtree.

To traverse a non-empty binary tree in **postorder**, perform the following operations recursively at each node :

1. Traverse the left subtree
2. Traverse the right subtree
3. Visit the root.

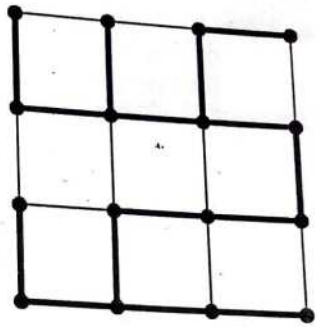
In graph theory, **breadth-first search (BFS)** is a strategy for searching in a graph when search is limited to essentially two operations : (a) visit and inspect a node of a graph ; (b) gain access to visit the nodes that neighbour the currently visited node. The BSF begins at a root node and inspect all the neighboring nodes. Then for each of those neighbour nodes in turn, it inspects their neighbor nodes which were unvisited, and so on.

Q 41. What do you mean by spanning trees? Explain with the help of diagrams.
(PTU, Dec. 2011)

Ans. A **spanning tree** T of a connected, undirected graph G is a tree composed of all the vertices and some (or perhaps all) of the edges of G. Informally, a spanning tree of G is a selection of edges of G that form a tree spanning every vertex. That is, every vertex lies in the tree, but no cycles (or loops) are formed. On the other hand, every bridge of G must belong to T.

A spanning tree of a connected graph G can also be defined as a maximal set of edges of G that contains no cycle, or as a minimal set of edges that connect all vertices.

In certain fields of graph theory it s often useful to find a minimum spanning tree of a weighted graph. Other optimization problems on spanning trees have also been studied, including the maximum spanning tree, the minimum tree that spans at least k vertices, the minimum spanning tree with at most k edges per vertex (Degree-Constrained Spanning Tree), the spanning tree with the largest number of leaves (closely related to the smallest connected dominating set), the spanning tree with the fewest leaves (closely related to the Hamiltonian path problem), the minimum diameter spanning tree, and the minimum dilation spanning tree.



Q 42. What is tree data structure? What are different ways of traversing a tree?
(PTU, May 2014)

Ans. A tree is a widely used abstract data type (ADT) or data structure implementing this ADT that simulates a hierarchical tree structure, with a root value and subtrees of children, represented as a set of linked nodes.

A tree data structure can be defined recursively (locally) as a collection of nodes (starting at a root node), where each node is a data structure consisting of a value, together with a list of references to nodes (the "children"), with the constraints that no reference is duplicated and none points to the root.

Different ways of traversing a tree :

(i) Depth first traversal :
  (a) Inorder
  (b) Preorder
  (c) Post order
(ii) Breadth first traversal

Q 43. What are the advantages and disadvantages of threaded trees?
(PTU, May 2014

Ans. Advantages :

1. By doing threading we neglect the recursive method of traversing a tree, which make use of stack and consumes many memory and time.
2. The node can keep record of its root.

Disadvantages :

1. This makes the tree more difficult.
2. More prone to errors when both the child are not present and both values of nod pointers to the ancestors.

Q 44. Design an algorithm to find out if the binary tree is :
Strictly binary **(PTU, May 20**

Ans. Strictly binary :

```
# define True 1
```

```
# define False 0
Int is strictbinary tree (struct tree & n)
{
    if (n = = null) return True ;
    if (n → left ! = Null & & n → right ! = Null)
    return (is strict Binary Tree (n → left) &&
            is strict Binary Tree (n → right) ;
    if (n → left = = Null and n → right = = Null)
    return True ;
    return False ;
}
```

Q 45. Name various tree traversal algorithms. Create a binary expression tree from the following expression and traverse it using all possible tree traversals. **(PTU, Dec. 2012)**

(A * B/C) * D + E + F / (G + H).

Ans. *Tree traversal refers to the process of visiting each node in a tree data structure, exactly once, in systematic way. Such traversals are classified by the order in which the nodes are visited.*

Pre-order

1. *Visit the root.*
2. *Traverse the left subtree.*
3. *Traverse the right subtree.*

In-order :

1. *Traverse the left subtree.*
2. *Visit the root.*
3. *Traverse the right subtree.*

Post-order :

1. *Traverse the left subtree.*
2. *Traverse the right subtree.*
3. *Visit the root.*

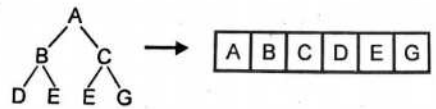
(A B/C) * D + E + F / (G + H)*

Pre-order : ***A/bc + + de + /fgh*

Post-order : *abc/*de + fg / h + /**

Q 46. How a binary tree can be represented as array structure? (PTU, May 2013)

Ans. *In case of array representation of binary trees, the root 'R' of tree 'T' is stored in tree [1]. If a node 'n' occupies tree [K], then it's left child is stored in tree [2* K] and its right child is stored in tree [2* K + 1].*



Q 47. How an element is searched in BST. **(PTU, Dec. 2013)**

Ans. *1. Start at the root node.*

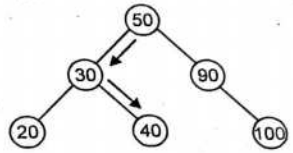
2. If the item that you are searching for is less than the root node, move to the left child the root node, if the item that you are searching for is more than the root node, move to the

right child of the root node and if it is equal to the root node, then you have found the item that you are looking for.

3. Now check to see if the item that you are searching for is equal to, less than or more than the new node that you are on. Again if the item you are searching for is less than the current node, move to the left child, and if the item that you are searching for is greater than the current node, move to the right child.

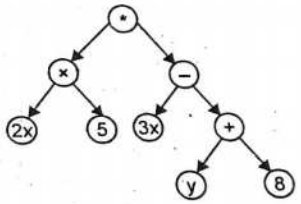
4. Repeat this process until you find the item that you are looking for or until the node doesn't have a child on the correct branch, in which case the tree doesn't contain the item which you are looking for.

Example : Suppose we want to search the element 40.



Q 48. Construct the binary tree for the following expression (2x+5) (3x–y+8). Give the sequence obtained when tree is traversed in post order form. **(PTU, Dec. 2013)**

Ans.



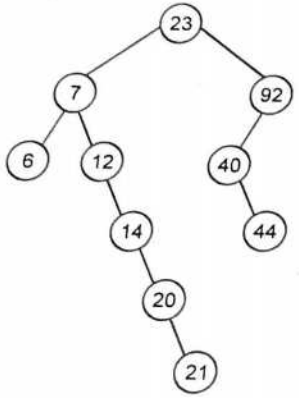
Q 49. Write the algorithm for post-order tree traversal. Also show the steps this algorithm on a set of numbers to shown an example. **(PTU, May 2**

Ans. Algorithm

1. Set Top = 1, stack [1] = Null and Ptr = Root.
2. Report steps 3 to 5 while Ptr ≠ null.
3. Set top = top + 1 and stack [top] = Ptr.
4. If Right [Ptr] ≠ Null, then :
   set Top = Top + 1 and stack [Top] = –Right [Ptr]
5. Set Ptr = Left [Ptr]
6. Set Ptr = stack [top] and Top = Top–1
7. Repeat while Ptr > 0;
   (a) Apply process to Info [Ptr]
   (b) Set Ptr = stack [top] and Top = Top–1
8. If Ptr < 0, then ;
   (a) Set PTR = –PTR
   (b) Go to step 2
9. Exit.

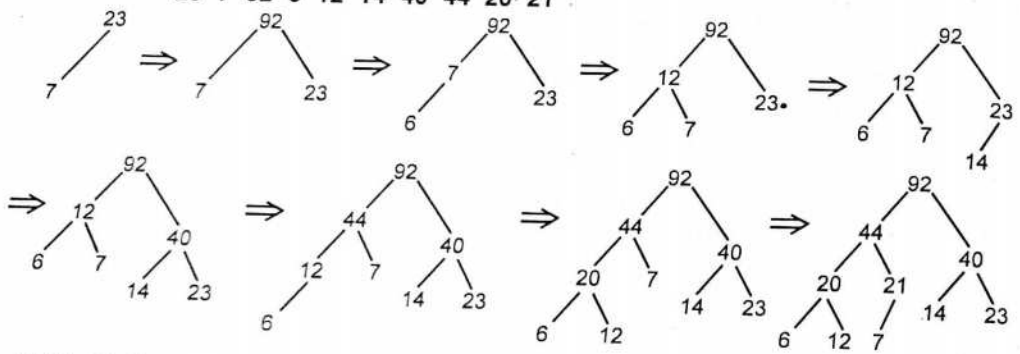
Q 50. Make a binary search tree and a heap tree from the given data :
23 7 92 6 12 14 40 44 20 21 **(PTU, Dec. 2014)**

Ans.



Heap tree :

23 7 92 6 12 14 40 44 20 21



Q 51. Define AVL and B-trees and their applications ? Explain various operations used for balancing a binary tree with the help of a suitable example ?
(PTU, Dec. 2014)

Ans. AVL tree : *Refer to Q.No. 13*

B-trees : *Refer to Q.No. 16 (a)*

AVL trees are applied in the following situations :

- *There are few insertion and deletion operations.*
- *Short search time is needed*
- *Input data is sorted or nearly sorted.*

AVL tree structures can be used in situations which require fast searching. But, the cost of rebalancing may limit the usefulness.

- *The main area for B-trees is databases (DBs)*

There are DBs larger than terabyte (~10^{12} bytes) and B-trees are useful in such applications.

- B-trees have also been used in file systems

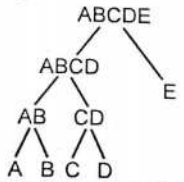
It is not uncommon that one has around 10000 files at home in a PC; some servers handles even more files.

Hence it has found that B-trees are even applicable in file systems.

When entries that are already sorted are stored in a tree, all new records will go the same route, and the tree will look more like a list. Therefore the tree needs balancing routines, making sure that under all branches are an equal number of records. This will keep searching in the tree at optimal speed. Specifically, if a tree with n nodes is a degenerate tree, the longest path through the tree will be n nodes, if it is a balanced tree, the longest path will be log n nodes.

Algorithms/Left_rotation : This shows how balancing is applied to establish a priority heap invariant in a Treap, a data structure which has the queueing performance of a heap, and the key look up performance of a tree. A balancing operation can change the tree structure while maintaining another order, which is binary tree sort order. The binary tree order is left to right with left nodes' keys less than right nodes' key whereas the priority order is up and down, with higher nodes priorities greater than lower nodes' priorities. Alternatively, the priority can be viewed as another ordering key, except that finding a specific key is more involved. The balancing operation can move nodes up and down a tree without affecting the left right ordering. A balanced binary tree has the minimum possible maximum height for the leaf nodes, because for any given number of leaf nodes the leaf nodes are placed at the greatest height possible.

Example :



Q 52. Suppose H is a complete binary tree with n elements then in what conditions, H is called a maxheap ? **(PTU, May 2015)**

Ans. Suppose H is a complete binary tree with n elements. Then H is called a heap or mexheap, if the value at N is greater than or equal to the value at any of the children of N.

Q 53. State different ways of traversing binary tree. **(PTU, May 2015)**

Ans. The binary tree can be traversed in three ways :

1. Inorder
2. Preorder
3. Postorder

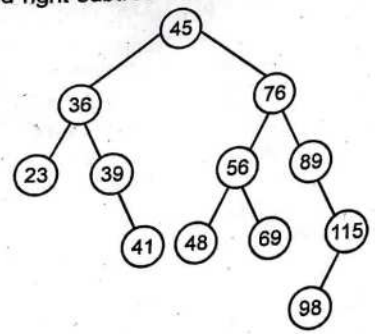
Q 54. What is Binary Search Tree (BST) ? Make a BST for the following sequence of numbers and Traverse the tree in Preorder.

45, 36, 76, 23, 89, 115, 98, 39, 41, 56, 69, 48 **(PTU, May 2015)**

Ans. A binary search tree B is a binary tree each node of which satisfies the following conditions.

1. The value of the left-subtree of 'x' is less than the value at 'x'.
2. The value of the right-subtree of 'x' is greater than the value at 'x'.

3. The left-subtree and right subtree of binary search tree are again binary search tree



Preorder :

23, 36, 39, 41, 45, 48, 56, 69, 76, 89, 98, 115

Q 55. What are the advantages of AVL tree and B-tree? (PTU, May 2015)

Ans. Advantages of an AVL tree :

(a) The height of an AVL tree is guaranteed to be < = 1.45 $\log_2 n$ where n is the number of nodes.

Hence, all operations are $O(\log_2 n)$.

(b) Any imbalance caused by an insertion or deletion can be corrected by just one or two rotations.

Advantages of B-Tree : B Trees take advantage of this by maintaining a balanced binary tree structure through the use of two files :

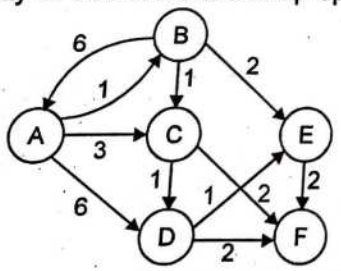
- **Index file :** It contain all the keys and tree's topology is represented by the organization of data in this file.
- **Data file :** A file that contains all the objects and information stored by the "tree". Objects contained here are referenced by block pointer references stored in the index file.

Q 56. Give two max heaps of size n each, what is the minimum possible time complexity to make a one max heap of size from elements of two max heaps ?

(PTU, May 2016)

Ans. O(n), because we can build a heap of 2n elements in O(n) time. Following are the steps. Create an array of size 2n and copy elements of both heaps to this array call build heap for the array of size 2n. Build heap operation takes O(n) time.

Q 57.



For the given Graph perform following operations :

(a) Find its adjacency list.

(b) Storage representation for adjacency list and edge list.

(c) Find its Path matrix. **(PTU, May 2016)**

Ans. (a) The adjacency list for the given graph is

Node	Neighbors
A	{B, C, D}
B	{A, C, E}
C	{D, F}
D	{E, F}
E	{F}
F	{–}

(b) Edge list :

L = {(A,B), (A,C), (A, D), (B, C), (B, E), (C, D), (C, F), (D, E), (D, F), (E, F)}

(c) Path matrix : Let G be a simple directed graph with m nodes $v_1, v_2, \ldots v_m$. The path matrix or reachability matrix of G is the m-square matrix $P = (p_{ij})$ defined as follows :

$$P_{ij} = \begin{cases} 1 & \text{If there is a path from } v_i \text{ to } v_j \\ 0 & \text{otherwise} \end{cases}$$

A procedure to delete a node from the graph as follows :

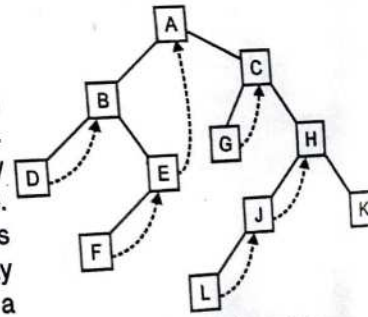
DELETE (INFO, LINK, START, AVAIL, ITEM, FLAG)

Q 58. What are the threaded binary trees ? Discuss different operations of node insertion and deletion in these trees. **(PTU, Dec. 2016)**

OR

What is a threaded binary tree? How this type of tree helps in traversal? **(PTU, May 2013)**

Ans. Threaded binary tree is used to remove the null pointers. Suppose in a tree half of the entries in the pointer fields LEFT and RIGHT will contain null elements. This space may be more efficiently used by replacing the null entries by some other type of information. Specifically, we will replace the certain null entries by special pointers which point to node higher in the tree. These special pointers are called threads. The threads in a threaded tree must be distinguished in some way from ordinary pointers. The threads in a diagram of a threaded tree are usually indicated by dotted lines. In computer memory an extra 1 bit TAG field may be used to distinguish threads from ordinary pointers.



One-way in order threading

The following operations can be defined on a threaded binary tree :

- Insertion of a node into a threaded binary tree.
- Deletion of a node from a threaded binary tree.

The insertion and deletion have to be carried out in such a way that after the operat the tree remains the inorder threaded binary tree.

Insertion :

1. When node X is inserted as left child of node Y and node Y has as empty left child.
2. When node X is inserted as right child of node Yand node Y has an empty right child.
3. When node X is inserted as left child of node Y and node Y has a non-empty left child.
4. When node X is inserted as right child of node Y and node Y has a non-empty right child.

Deletion :

1. When X is a left leaf node.
2. When X is a right leaf node.
3. When X is only having a right sub–tree
4. When X is only having a left sub–tree
5. When X is having both sub-trees.

Q 59. Write an algorithm to find minimum and maximum element from a binary search tree. **(PTU, May 2017)**

Ans. Algorithm to find minimum & maximum element in a binary search tree

For minimum

1. Start from root node
2. Go to left child
   - ❑ Keep on iterating (or recursively) till, we get left child as null
   - ❑ We found the minimum value in binary search tree.

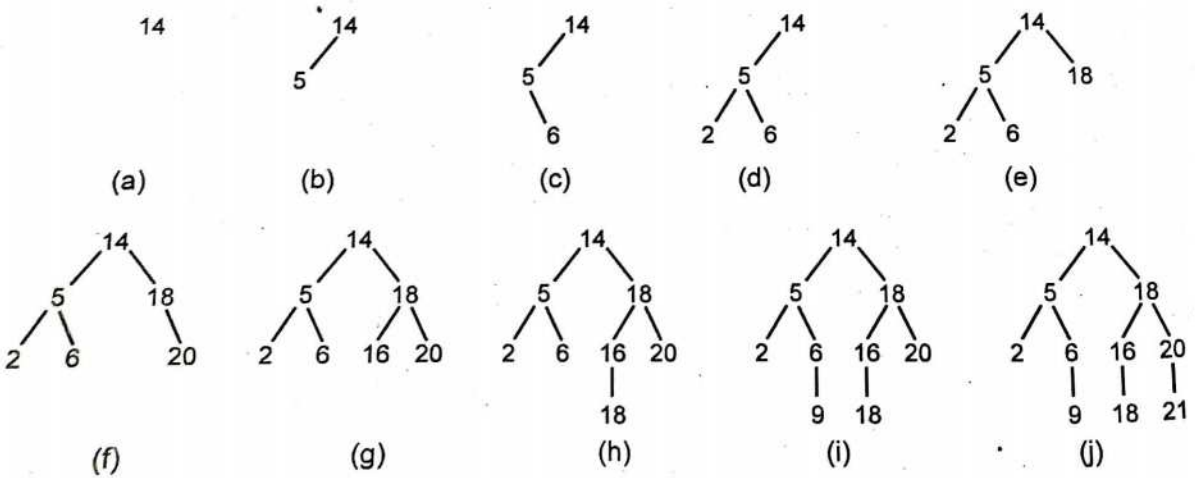
For maximum

1. Start from root node
2. Go to right child
   - ❑ Keep iterating (or recursively) till we found right child as null.
   - ❑ We found the maximum value in binary search tree.

Q 60. What is binary search tree? Draw the binary search tree for the following input:
14, 5, 6, 2, 18, 20, 16, 18, 9, 21 **(PTU, May 2018)**

Ans. Binary search tree : *Refer to Q.No. 6*

Binary search tree



Q 61. Supose a binary tree T is in memory. Write a procedure to delete all the terminal nodes. **(PTU, May 2018)**

Ans. Procedure to delete leaf node from bineary tree :

```
#include <bits/std C++.h>
using namespace std;
struct node{
int data;
struct Node * left;
struct Node * right;
};
struct Node * New Node (int data)
{
Struct Node * temp = New Node;
temp → data = data;
temp → left = temp → right = NULL;
return temp;
}
struct Node * insert (struct Node * int data)
{
If (root = = NULL)
return new Node (data);
If (data < root → data)
root → left = insert (root → left, data);
else If (data > root → data)
root → right = insert (root → right, data);
return root;
}
void inorder (struct Node * root)
{
If (root ! = NULL){
inorder (root → left);
cout <<root → data << " ";
inorder (root → right);
}
}
struct Node * leaf Delete (struct Node * root)
{
If (root → left = = NULL && root → right = = NULL) {
free (root);
return NULL;
}
root → left = leaf delete (root → left);
root → right = leaf delete (root → right);
return root;
}
int main ( )
```

```
{
struct Node * root = NULL;
root = insert (root, 20);
insert (root, 10);
insert (root, 5);
insert (root, 15);
insert (root, 30);
insert (root, 25);
insert (root, 35);
cout <<"Inorder before deleting the leaf node "<< endl;
inorder (root);
cout <<endl;
leafdelete (root);
cout <<"Inorder after deleting the leaf node"<<end;
inorder (root);
return 0;
}
```

Q 62. Are B trees of order 2 are full binary trees ? If yes, explain how. **(PTU, May 2018)**

Ans. B tree of order 2 is fully binary tree. In order for the B tree to function there needs be a choice in the number of keys. This means that the smallest possible B tree node is e that has either one or two keys. That's basically a (2, 3) tree and reportedly that's exactly v B trees were invented as a generalisation of (2, 3) trees.

Q 63. Discuss recursive procedure in trees. **(PTU, May 2019)**

Ans. A recursive procedure is an algorithm that handles a list of items, where each item be itself a list by decomposing the process into the handling of the first item of the list & w this by the handling of the remainder of the list. A recursive procedure is a way of ng problem that contain a number of item to be processed. The major problem that can r in recursive structures is circularity. A tree is not circular. There is no problem in these ures. Because circularity is an undesirable property, a program that verifies whether an ng recursive data structure contains circularity is required.

❑❑❑

Chapter 4

Sorting and Hashing

Contents

Objective and properties of different sorting algorithms : Selection Sort, Bubble Sort, Insertion Sort, Quick Sort, Merge Sort, Heap Sort ; Performance and Comparison among all the methods, Hashing.

POINTS TO REMEMBER

- Sorting actually refers to the operation of arranging data in some given order, such as increasing or decreasing, with numerical data, or alphabetically with character data.
- Sorting is a process of arranging the elements in a particular order.
- Sorting is generally classified as either internal or external. In an internal sorting, all the elements are held in primary storage during the sorting process. An external sorting uses primary storage for the elements currently being sorted and secondary storage for any data that does not fit in primary memory.
- In insertion sorting, we choose a particular element and then insert it at the appropriate location in the sorted subarray.
- In selection sort, we repeatedly find the next smallest element in the list and move it to its final position that it will occupy in the sorted array.
- Merge sort is based on the divide and conquer strategy in which we divide the data into smaller pieces, recursively conquer each piece and merge the result into a final result.
- Bubble sort algorithm is used for sorting a list.
- Bubble sort algorithm compares two numbers at a time and swaps them if they are in wrong order.
- Quick sort is one the fastest sorting algorithm used for sorting a list.
- Merge sort is a comparison based sorting algorithm.
- Selection sort is a sorting algorithm, specifically an in-place comparison sort.
- Time complexity of selection sort algorithm is O (n^2).
- Insertion sort is a simple sorting algorithm in which the sorted array (or list) is built o entry at a time.
- Hashing is the most efficient search technique in which we could probably find the o in only one comparison.

- Hash table is an array of some constant size which is used for sorting and maintaining data efficiently so that it can be retrieved quickly.
- Direct, division, midsquare, folding methods are the most commonly used hash function.

QUESTION-ANSWERS

Q 1. What do you mean by sorting?

Ans. Sorting actually refers to the operation of arranging data in some given order, such as increasing or decreasing, with numerical data, or alphabetically, with character data.

Q 2. What do you mean by external sorting?

Ans. External sorting is a term for a class of sorting algorithms that can handle massive amounts of data. External sorting is required when the data being sorted does not fit into the main memory of a computing device (usually RAM) and a slower kind of memory (usually a hard drive) needs to be used.

One example of external sorting is the external mergesort algorithm.

Q 3. Explain the bubble sort algorithm.

Ans. Bubble sort algorithm is used for sorting a list. It makes use of a temporary variable for swapping. It compares two numbers at a time and swaps them if they are in wrong order. This process is repeated until no swapping is needed. The algorithm is very inefficient if the list is long.

e.g. List : 7 4 5 3

1. 7 and 4 are compared
2. Since 4 < 7, 4 is stored in a temporary variable.
3. The content of 7 is now stored in the variable which was holding 4.
4. Now, the content of temporary variable and the variable previously holding 7 are swapped.

Q 4. What is quick sort? **(PTU, May 2015)**

Ans. Quick sort is one the fastest sorting algorithm used for sorting a list. A pivot point is chosen. Remaining elements are portioned or divided such that elements less than the pivot point are in left and those greater than the pivot are on the right. Now, the elements on the left and right can be recursively sorted by repeating the algorithm.

Q 5. Explain quick sort and merge sort algorithms.

Ans. Quick sort employs the 'divide and conquer' concept by dividing the list of elements into two sub elements.

The process is as follows :

1. Select an element, pivot, from the list.
2. Rearrange the elements in the list, so that all elements those are less than the pivot are arranged before the pivot and all elements those are greater than the pivot are arranged after the pivot. Now the pivot is in its position.
3. Sort the both sub lists - sub list of the elements which are less than the pivot and the list of elements which are more than the pivot recursively.

Merge Sort : A comparison based sorting algorithm. The input order is preserved in the sorted output.

Merge Sort algorithm is as follows :

1. The length of the list is 0 or 1, and then it is considered as sorted.
2. Otherwise, divide the unsorted list into 2 lists each about half the size.
3. Sort each sub list recursively. Implement the step 2 until the two sub lists are sorted.
4. As a final step, combine (merge) both the lists back into one sorted list.

Q 6. What is selection sort? **(PTU, May 2019)**

Ans. Selection sort is a sorting algorithm, specifically an in-place comparison sort. It has $O(n^2)$ time complexity, making it inefficient on large lists, and generally performs worse then the similar insertion sort. Selection sort is noted for its simplicity, and also has performance advantages over more complicated algorithms in certain situations, particularly where auxiliary memory is limited.

Algorithm :

1. Set first position as current position.
2. Find the minimum value in the list.
3. Swap it with the value in the current position.
4. Set next position as current position.
5. Repeat steps 2-4 until you reach end of list.

Q 7. What is insertion sort? **(PTU, Dec. 2015)**

Ans. Insertion sort is a simple sorting algorithm in which the sorted array (or list) is built one entry at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort. However, insertion sort provides several advantages:

- Simple implementation
- Efficient for (quite) small data sets
- Adaptive (i.e., efficient) for data sets that are already substantially sorted : the time complexity is O (n + d), where d is the number of inversions.
- More efficient in practice than most other simple quadratic (i.e., O (n^2)) algorithms such as selection sort or bubble sort ; the best case (nearly sorted input) is O (n).
- Stable ; i.e., does not change the relative order of elements with equal keys.
- In-place ; i.e., only requires a constant amount O (1) of additional memory space.
- Online ; i.e., can sort a list as it receives it.

Q 8. Comparison of insertion sort with selection sort algorithms.

Ans. Insertion sort is very similar to selection sort. As in selection sort, after k passes through the array, the first k elements are in sorted order. For selection sort these are k smallest elements, while in insertion sort they are whatever the first k elements were in the unsorted array. Insertion sort's advantage is that it only scans as many elements as needed to determine the correct location of the k+1st element, while selection sort must scan all remaining elements to find the absolute smallest element.

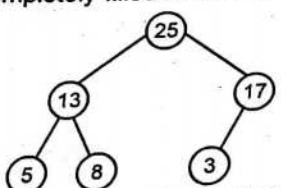
Calculations show the insertion sort will usually perform about half as many comparisons

as selection sort. Assuming the k+1st element's rank is random, insertion sort will on average require shifting half of the previous k elements, while selection sort always requires scanning all unplaced elements. If the input array is reverse-sorted, inserton sort performs as many comparisons as selection sort. If the input array is already sorted, insertion sort performs as few as n-1 comparisons, thus making insertion sort more efficient when given sorted or "nearly sorted" arrays.

(PTU, May 2019)

Q 9. Explain heap sort.

Ans. The binary heap data structures is an array that can be viewed as a complete binary tree. Each node of the binary tree corresponds to an element of the array. The array is completely filled on all levels except possibly lowest.

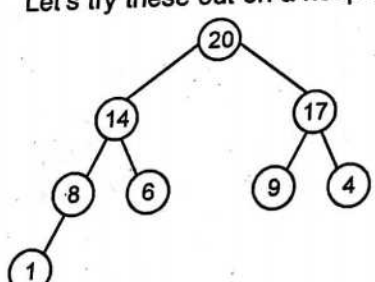


We represent heaps in level order, going from left to right. The array corresponding to the heap above is [25, 13, 17, 5, 8, 3].

The root of the tree A [1] and given index i of a node, the indices of its parent, left child and right child can be computed

```
PARENT (i)
    return floor (i/2)
LEFT ()
    return 2
RIGHT (i)
    return 2i + 1
```

Let's try these out on a heap to make sure we believe they are correct. Take this heap,



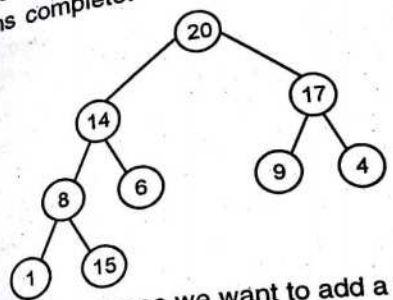
Which is represented by the array [20, 14, 17, 8, 6, 9, 4, 1].

We'll go from the 20 to the 6 first. The index of the 20 is 1. To find the index of the left child, we calculate 1*2 = 2. This takes us (correctly) to the 14. Now, we go right, so we calculate 2*2 + 1 = 5. This makes us (again, correctly) to the 6.

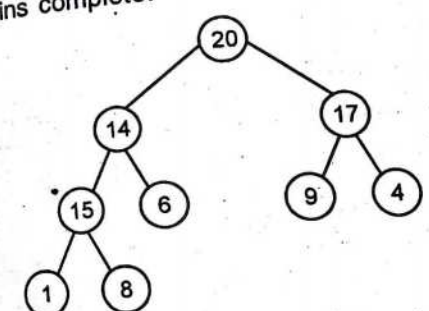
Q 10. How elements are inserted in heap?

Ans. Suppose we have a heap as follows :

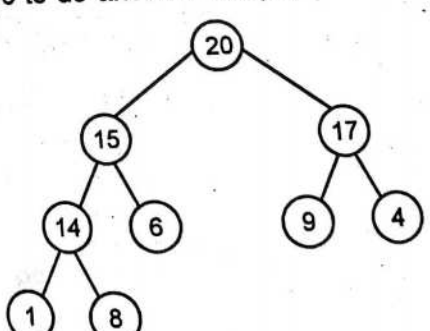
Let's suppose we want to add a node with key 15 to the heap. First, we add the node to the tree at the next spot available at the lowest level of the tree. This is to ensure that the tree remains complete.



Let's suppose we want to add a node with key 15 to the heap. First, we add the node to the tree at the next spot available at the lowest level of the tree. This is to ensure that the tree remains complete.



Now we do the same thing again, comparing the new node to its parent. Since 14 < 1 we have to do another swap :



Now element are inserted because 15 ≤ 20.

Q 11. What is the complexity of merge sort? **(PTU, Dec. 2005)**

Ans. Complexity of merge sort is as follows :

(i) Worst case : O (n log n)

(ii) Average case : O (n log n)

(iii) Best case : O (n log n).

Q 12. Write the time complexities of quick sorting method. **(PTU, May 2009)**

Ans. (i) Average case : O (n * log (n))

(ii) Best case : O (n * log (n))

(iii) Worst case : O (n^2).

Q 13. What is the need for external sorting? **(PTU, Dec. 2009)**

Ans. External sorting is a method used to sort elements which are two large to fit in the main memory of the computer. Any sorting algorithms that uses external memory, such as tape or disk, during sort is called external sort. Since most common sort algorithms assume high speed random access to all intermediate memory, they are unsuitable if the values to be sorted do not fit in main memory. The main concern with external sorting is to minimize external disk access since reading a disk block takes about a million times longer than accessing an items in RAM.

Q 14. Write a program to sort integer using selection sort. **(PTU, May 2006)**

Ans. In selection sort, first find the first position. Then find the second smallest element in the list and put it in the second position.

Suppose we want to sort the integers.

77, 33, 44, 11, 88, 22, 66, 55

Pass, LOC	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K = 1, LOC = 4	(77)	33	44	(11)	88	22	66	55
K = 2, LOC = 6	11	(33)	44	77	88	(22)	66	55
K = 3, LOC = 6	11	22	(44)	77	88	(33)	66	55
K = 4, LOC = 6	11	22	33	(77)	88	(44)	66	55
K = 5, LOC = 8	11	22	33	44	88	77	66	(55)
K = 6, LOC = 7	11	22	33	44	55	(77)	(66)	88
K = 7, LOC = 7	11	22	33	44	55	66	(77)	88
Sorted :	11	22	33	44	55	66	77	88

Thus algorithm sorts the array A with N elements.

SELECTION (A, N)

1. Repeat steps 2 and 3 for K = 1, 2, ..., N – 1
2. Call MIN (A, K, N, LOC)
   Set TEMP = A [K], A [K] = A [LOC] & A [LOC] = TEMP
3. Exit

MIN (A, K, N, LOC) This algorithm finds the location LOC of the smallest element among A[K], A[K+1], ... A[N]

1. Set MIN = A [K] & LOC = K
2. Repeat for J = K + 1, K + 2, ... N
   If MIN > A [J], then :
   Set MIN = A[J], LOC = J
3. Return.

Here, complexity = 0 (n^2)

Q 15. Sort the following list of numbers
52, 1, 27, 85, 66, 23, 13, 57
Using any efficient sorting algorithm. **(PTU, Dec. 2**

Ans. The list of given numbers sorting using bubble so

Pass 1 : (52) (1) 27 85 66 23 13 57
1 (52) (27) 85 66 23 13 57
1 27 (52) (85) 66 23 13 57
1 27 52 (85) (66) 23 13 57
1 27 52 66 (85) (23) 13 57
1 27 52 66 23 (85) (13) 57
1 27 52 66 23 13 (85) (57)
1 27 52 66 23 13 57 85

Pass 2 : (1) (27) 52 66 23 13 57 85
1 (27) (52) 66 23 13 57 85
1 27 (52) (66) 23 13 57 85
1 27 52 (66) (23) 13 57 85
1 27 52 23 (66) (13) 57 85
1 27 52 23 13 (66) (57) 85
1 27 52 23 13 57 66 85

Pass 3 : (1) (27) 52 23 13 57 66 85
1 (27) (52) 23 13 57 66 85

	1	27	(52)	(23)	13	57	66	85
	1	27	23	(52)	(13)	57	66	85
	1	27	23	13	52	57	66	85
Pass 4 :	(1)	(27)	23	13	52	57	66	85
	1	(27)	(23)	13	52	57	66	85
	1	23	(27)	(13)	52	57	66	85
	1	23	13	27	52	57	66	85
Pass 5 :	(1)	(23)	13	27	52	57	66	85
	1	(23)	(13)	27	52	57	66	85
	1	13	23	27	52	57	66	85
Pass 6 :	1	13	23	27	52	57	66	85

Pass 7 : All elements are solved.

Q 16. Sort the string DATA STRUCTURES using quicksort. **(PTU, Dec. 2004)**

Ans. (a) (D)ATA STRUCTURES

1. Scan from right to left, until finding a character which precedes D alphabetically. It is C. Interchange D & C to obtain :

   CATASTRU(D)TURES

2. Now, scan the list from left to right, until finding a character which succeeds D alphabetically. It is 'T'. Interchange D and T.

   CA(D)ASTRUCTURES

3. Again scan from right to left to obtain,

   CAA(D)STRUTTURES

   Sublist 1 Sublist 2

4. (C)AA

   Scan from right to left to obtain :

   AA(C)

5. (S)TRUCTURES

   Scan from right to left to obtain :

   STRUCTURE(S)

6. Scan from left to right

   S(S)RUTTURET

7. Scan from right to left :

   SERUTTUR(S)T

8. Scan from left to right

   SER(S)TTURUT

9. Scan from right to left

   SERRTTU(S)UT

10. Scan from left to right

    SERR(S)TUTUT

11. (S)ERR

    Scan from right to left to obtain :

    RER(S)

    (R)ER

12. Scan from right to left

    E(R)R

13. (T)UTUT

    Scan from right to left

    TUTU(T)

14. Scan from left to right

    T(T)TUU

    Hence the final result is

    AACDERRSSTTTUU

Q 17. Sort the following list of elements using Bubble Sort :

98 89 44 7 5 35 12 100 2 57

What is its complexity? **(PTU, M**

Ans.

Pass 1 :	(98)	(89)	44	7	5	35	12	100	2	57

	89	(98)	(44)	7	5	35	12	100	2	57
	89	44	(98)	(7)	5	35	12	100	2	57
	89	44	7	(98)	(5)	35	12	100	2	57
	89	44	7	5	(98)	(35)	12	100	2	57
	89	44	7	5	35	(98)	(12)	100	2	57
	89	44	7	5	35	12	(98)	(100)	2	57
	89	44	7	5	35	12	98	(100)	(2)	57
	89	44	7	5	35	12	98	2	(100)	(57)
	89	44	7	5	35	12	98	2	57	100
Pass 2 :	(89)	(44)	7	5	35	12	98	2	57	100
	44	(89)	(7)	5	35	12	98	2	57	100
	44	7	(89)	(5)	35	12	98	2	57	100
	44	7	5	(89)	(35)	12	98	2	57	100
	44	7	5	35	(89)	(12)	98	2	57	100
	44	7	5	35	12	(89)	(98)	2	57	100
	44	7	5	35	12	89	(98)	(2)	57	100
	44	7	5	35	12	89	2	(98)	(57)	100
	44	7	5	35	12	89	2	57	98	100
Pass 3 :	(44)	(7)	5	35	12	89	2	57	98	100
	7	(44)	(5)	35	12	89	2	57	98	100
	7	5	(44)	(35)	12	89	2	57	98	100
	7	5	35	(44)	(12)	89	2	57	98	100
	7	5	35	12	(44)	(89)	2	57	98	100
	7	5	35	12	44	(89)	(2)	57	98	100
	7	5	35	12	44	2	(89)	(57)	98	100
	7	5	35	12	44	2	57	89	98	100
Pass 4 :	(7)	(5)	35	12	44	2	57	89	98	100
	5	(7)	(35)	12	44	2	57	89	98	100
	5	7	(35)	(12)	44	2	57	89	98	100
	5	7	12	(35)	(44)	2	57	89	98	100
	5	7	12	35	(44)	(2)	57	89	98	100
	5	7	12	35	2	44	57	89	98	100

Pass 5 :	(5)	(7)	12	35	2	44	57	89	98	100
	5	(7)	(12)	35	2	44	57	89	98	100
	5	7	(12)	(35)	2	44	57	89	98	100
	5	7	12	(35)	(2)	44	57	89	98	100
	5	7	12	2	35	44	57	89	98	100
Pass 6 :	(5)	(7)	12	2	35	44	57	89	98	100
	5	(7)	(12)	2	35	44	57	89	98	100
	5	7	(12)	(2)	35	44	57	89	98	100
	5	7	2	12	35	44	57	89	98	100
Pass 7 :	(5)	(7)	2	12	35	44	57	89	98	100
	5	(7)	(2)	12	35	44	57	89	98	100
	5	2	7	12	35	44	57	89	98	100
Pass 8 :	(5)	(2)	7	12	35	44	57	89	98	100
	2	5	7	12	35	44	57	89	98	100

Pass 9 : All list are sorted.

Q 18. Write a algorithm for sorting numbers using heap sort. **(PTU, May 2014)**

Ans. Algorithm :

Heap Creation Algorithm

Step 1. [Create heap]
Repeat through Step 7 for k = 2, 3, ..., n

Step 2. [Initialize]
i = k
temp = data [k]

Step 3. [obtain parent of new element]
i = i/2

Step 4. [Place new element in the existing heap]
Repeat through step 6 while (i > 1) and (temp > data [j])

Step 5. [Interchange elements]
data [i] = data [j]

Step 6. [obtain next parent]
i = j
j = 1/2
if = (j <1) then j = 1

Step 7. [copy new element value into its proper place]
data [i] = temp

Step 8. Return.

Heap Sort Algorithm

Heap_sort (data, n)
where data represent the list of elements

n represents number of elements in the list.

Step 1. [Create initial heap]
call Heap-Creation [data, n]

Step 2. [Start sort]
Repeat through step 10 for k = n, n–1, ..., 2

Step 3. [Interchange elements]
data [1] = data [k]

Step 4. temp = data [1]
i = 1
j = 2

Step 5. [Find index of largest child of new element]
If j+1 <k then
If data [j+1] > data [j] then
j = j+1

Step 6. [Recreate the new heap]
Repeat through step 10 while j < = k –1 and data [j]> temp

Step 7. [Interchange element]
data [i] = data [j]

Step 8. [obtain left child]
i = j
j = 2 * i

Step 9. [obtain index of next largest child]
if j + 1 <k
If data [j+1]> data [j] then j = j + 1 else if j >n then j = 1

Step 10. [copy element into its proper place]
data [j] = temp

Step 11. Exit

Q 19. Write a short note on Hashing. **(PTU, May 2006)**

Ans. Hashing is a searching techniques which is independent of 'n'. Suppose there is a file 'F' of 'n' records with a set 'K' of keys which a uniquely determine records in 'F'. 'F' is maintained in memory by a table 'T' of 'n' memory locations and 'L' is the set of memory address of locations in 'T'.

$$H : K \rightarrow L$$

'H' is a hash function, which gives a location in memory by applying on 'K'. 'K' is uniquely determined element.

Q 20. Define hash function. **(PTU, May 2009 ; Dec. 2009, 2008, 2005)**

Ans. Hashing is a searching technique which is independent of 'n'. Suppose there is a file 'F' of 'n' records with a set 'K' of keys which uniquely determine the record in 'F'. 'F' is maintained in memory by a table 'T' of 'n' memory locations, 'L' is the set of memory addresses of location in 'T'.

$$H : K \rightarrow L$$

where, 'H' is a hash or hashing functions which gives a location in memory by applying it on 'K'. 'K' is any uniquely determined element.

The hash function can be determined by the following methods :

(i) Division method
(ii) Mid square method
(iii) Folding method.

Q 21. What does Hashing mean? Explain the technique in detail. **(PTU, May 2019, 2010, 2004 ; Dec. 2006)**

OR

What is hashing? Discuss basic hash functions with example. **(PTU, May 2007)**

OR

What is hashing? Give the characteristics of hash function. Name different hash functions. **(PTU, Dec. 2009)**

Ans. Hashing is a searching technique which is independent of 'n'. Suppose there is a file 'F' of 'n' records. With a set 'K'of keys, which techniquely determines records in 'F'. 'F' is maintained in memory by a table 'T' of 'n' memory locations and 'L' is the set of memory addresses of locations in'T'.

$$H : K \rightarrow L$$

'H' is a hash or hashing function which gives a location in memory by applying it on 'K'.

Hash functions :

1. Division method : In this method, choose a no. m > n of keys in 'K', which may be prime no. or a no. without small directions. The hashing function is

$$H(K) = K \pmod m$$
$$H(K) = K \pmod m + 1$$

2. Mid square method : The key 'K' is squared and 'L' is obtained by deleting digi from both ends of K^2.

3. Folding method : The key 'K' is portitioned into a no. of parts $K_1, K_2 \ldots\ldots K_r$, whe each part except the last has the same no. of digits as required address. Then, parts added together ignoring the last carry to get the address

$$H(K) = K_1 + K_2 + \ldots\ldots + K_r$$

e.g. Suppose there are 68 employees and 100 locations having 2-digit addresses f 00 to 99. Apply three methods on following keys.

Keys (K) : 3205 148 2345

(i) Division no :

Let m = 97

H (3208) = 4, H (7148) = 67, H (2345) = 17

(ii) Mid square method :

K^2 1027025 57093904 5499025

72,93 and 90 are addresses of digits.

(iii) Folding method :

H (3205) = 32 + 05 = 37

H (7148) = 71 + 48 = 19 (Ignore carry)

H (2345) = 23 + 45 = 68.

Q 22. What are the various types of hash function. How collision is handled while hashing. **(PTU, May 2010)**

OR

Explain the various collision resolution techniques used for hashing with example. **(PTU, Dec. 2007)**

Ans. Hashing is a searching technique where we can compute the location of the desired record in order to retrieve in a single access. The basic idea of hash function is the transformation of the key into the corresponding location in the hash table. A hash function H can be defined as a function that takes key as input and transforms it into a hash table index. Following are the most popular methods of hash functions :

1. Division Method : Table is an array of database file where the employee details are stored. Chose a number m, which is larger than the number of keys K i.e. m is greater than the total number of records the TABLE. The number m is usually chosen to be prime number of minimize the collision. The hash function H is defined by

$$H(K) = K \pmod m$$

Where H (K) is the hash address and here K (mod m) means the remainder when K is divided by m.

2. Mid Square Method : The key K is squared. Then the hash function H is defined by

$$H(K) = K^2 = L$$

Where L is obtained by digits from both the ends of K^2 starting from left. Same number of digits must be used for all of the keys.

e.g.

K	4147	3750	2103
K2	17199609	14062500	4422609
H (K)	97	62	22

3. Folding Method : The key K is partitioned into a number of parts K1, K2, K3 ..... Kr. The parts have same number of digits as the required hash address, except possibly for the last part. Then the parts are added together, ignoring the last carry. That is

$$H(K) = K1 + K2 + \ldots\ldots + Kr$$

Here we are dealing with a hash table with index from 00 to 99, i.e. two-digit hash table. So we divide the K numbers of two digits.

e.g.

K	2203	7248	12345
K1, K2, K3	22,03	72,48	12,34,5
H (K)	H (2203)	H (7248)	= 12 + 34 + 5
= K1 + K2 + K3	= 22 + 03 = 25	= 72 + 48 = 20	= 51

Hash Collision : It is possible that two non-identical keys K1, K2 are hashed into the same hash address. This situation is called hash collision. Collisions are almost impossible to avoid but it can be minimized considerably by introducing three techniques which are open addressing, chaining, bucket addressing. But in this question we will explain only one technique.

Open addressing : In open addressing method, when a key is colliding with another key, the collision is resolved by finding a nearest empty space by probing the cells.

Suppose a record R with key K has a hash address H (K) = h, then we will linearly search h + i (where i = 0, 1, 2, ... m) locations for free space (i.e. h, h + 1, h + 2, h + 3 .... hash address).

The position in which a key can be stored is found by sequentily searching all positions starting from the position calculated by the hash function until an empty cell is found. This type of probing is called linear probing.

Quadrating Probing : Suppose a record with R with key K has the hash address H (K) = h. Then instead of searching the location with address h, h + 1, h + 2 .... h + i, we search for free hash address h, h + 1, h + 4, h + 9 ..... $h + i^2$ .....

Q 23. What is hash table?

Ans. Hash table are common data structure. They consist of an array (the hash table) and a mapping (the hash function). The hash function maps keys into hash values. Items stored in a hash table must have keys. The hash function maps the key of an item to a hash value and that hash value is used as an index into the hash table for that item. This allow items to be inserted and located quickly. It is the best search method introduced for binary search.

Q 24. Why we use hash table?

Ans. Having new looked at arrays, linked list, stack, queue and trees we will conclude the concept of hash tables. The efficiency of storage, retrieval and sorting has been discussed elsewhere and not dealt with in detail when discussing the earlier data structure. Now, it become one of the main reason for using the hash tables.

Q 25. What is double hashing?

Ans. Double hashing is one of the best method available for open addressing because the permutations produced have many of the characteristics of randomly chosen permutations. Double hashing uses a hash functions of the form :

$$h(K, i) = [h, (K) + ih_2 (K)] \bmod m$$

where m is the size of hash table, h_1 (K) [=K modm] and h_2 (K) [=Kmodm) are tw auxiliary hash functions. Here m is chosen to be slightly less than m.

Q 26. What is rehashing? **(PTU, May 201**

Ans. In rehashing we find an alternative empty location by modifying the hash funct and applying the modified hash function to the colliding symbol.

Q 27. What are the various uses of hashing?

Ans. The uses of hashing are as follow :

1. CD Database
2. Drivers Licences/Insurance Cards

3. Sparse Arrays
4. File Signature
5. Game Boards
6. A.D.T Dictionary (Searching, Sorting)
7. Graphics
8. Associative Arrays
9. Database Indexing
10. Caches
11. Sets
12. Object Representation
13. Unique Data Representation.

Q 28. Why hashing is needed? **(PTU, Dec. 2011)**

Ans. While running collections of caching machines some limitatons are experienced. A common way of load balancing *n* cache machines is to put object O in cache maching number hash (O) mod *n*. But this will not work if a cache machine is added or removed because n changes and every object is hashed to a new location. This can be disastrous since the originating content servers are flooded with requests from the cache machines. Hence, consistent hashing is needed to avoid swamping of servers.

Q 29. List out the different types of hashing functions ? **(PTU, Dec. 2014)**

Ans. Different types of hash functions :

1. Division method
2. Mid square
3. Multiplicative hash function
4. Digit folding
5. Digit analysis

Q 30. What is the advantage and average efficiency of quick sort ? Apply Quick sort on the following data and show the contents of the array every pass : **(PTU, Dec. 2014)**

48 7 26 44 13 23 98 57 100 5 32

Ans. Advantages :

1. Quicksort is an in-place sort that needs no temporary memory.
2. Typically, quicksort is faster in practice than other Θ(n log n) algorithm, because its inner loop can be efficiently implemented on most architectures.
3. Quick sort can be easily parallelized due to its divide-and-conquer nature.
4. In most real-world data, it is possible to make design choice which minimize the probability of requiring quadratic time.
5. Quick sort tends to make excellent usage of the memory hierarchy like virtual memory of caches. It is well suited to modern computer architectures.

Quick Sort efficiency : Best case situation : Assuming that the list breaks into two equal halves, we have two lists of size N/2 to sort. In order for each half to be partitioned, (N/2) + (N/2) = N, comparisons are made. Also assuming that each of these lists breaks into two equal sized sublists, we can assume that there will be at the most log(N) splits. This will result in a best time estimate of O(N*log(N)) for quick sort.

Worst case situation : In the worst case, the list does not divide equally and is larger on one side than the other. In this case, the splitting may go on N-1 times. This gives a worst-case time estimate of $O(N^2)$.

Average time : The average time for quick sort is estimated to be O(N*log(N)) comparison.

```
48  7  26  44  13  23  98        57        100  5  32
48  7  26  44  13  23  32        57        100  5  98
48  7  26  44  13  23  32         5        100 57  98
48  7  26  44  13  23  32        48        100 57  98
 5  7  26  44  13  23  32
```

Now Sort two subarray

```
 5  7  26  44        13  23  32          100  57  98
 5  7  26  23        13  44  32           57 100  98
 5  7  13  23        26  44  32           57  98 100
 5  7  13  23            44  32
                      32  44
```

So Sorted array is :

5 7 13 23 26 32 44 48 57 98 100

Q 31. What is Quick Sort ? Sort the following array using quick sort method. **(PTU, May 2015)**

24 56 47 35 10 90 82 31

Ans. Quick Sort : *Refer to Q.No. 4*

The given data is

24 56 47 35 10 90 82 31

```
Pass 1 : (10)  24  (56  47  35  90  82  31)
Pass 2 :  10   24  (56  47  35  90  82  31)
Pass 3 :  10   24  (47  35  31)  56  (90  82)
Pass 4 :  10   24  (35  31)  47  56  (90  82)
Pass 5 :  10   24  (31)  35  47  56  (90  82)
Pass 6 :  10   24   31   35  47  56  (82)  90
Pass 7 :  10   24   31   35  47  56   82   90
Pass 8 :  10   24   31   35  47  56   82   90
```

Q 32. What is the advantage and average efficiency of Insertion sort ? Sort the following data using an insertion sort algorithm and show the contents of the array after every pass : **(PTU, Dec. 2015)**

23 7 92 6 12 14 40 44 20 21

Ans. Insertion sort : *Refer to Q.No. 7*

```
23  7  92  6  12  14  40  44  20  21
 7 23  92  6  12  14  40  44  20  21
 7 23  92  6  12  14  40  44  20  21
 6  7  23 92  12  14  40  44  20  21
 6  7  12 23  92  14  40  44  20  21
 6  7  12 14  23  92  40  44  20  21
 6  7  12 14  23  40  92  44  20  21
 6  7  12 14  23  40  44  92  20  21
 6  7  12 14  20  23  40  44  92  21
 6  7  12 14  20  21  23  40  44  92
```

Q 33. What is Heap ? How are they represented in memory ? Perform heap sort for the following items : **(PTU, May 2016)**

44, 30, 50, 22, 60, 55, 77, 55, 10.

Ans. Heap : A heap is a complete binary tree which leads to the idea of storing it using an array. Heap has the following properties :

(a) The value of the root is the smallest or largest value in the tree.

(b) Every sub tree is a heap.

A heap is represented in memory by sequential representation i.e., using linear array.

Build a heap H from the following list of numbers :

44, 30, 50, 22, 60, 55, 77, 55

1. 44

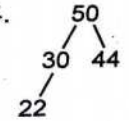
Item = 40

2. 

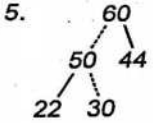
Item = 30

3. 

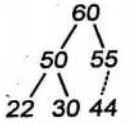
Item = 50

4. 

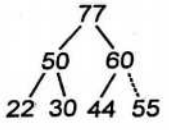
Item = 22

5. 

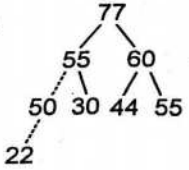
Item = 60

6. 

Item = 55

7. 

Item = 77

8. 

Item = 50

Q 34. Define the criteria for selecting a hash function. **(PTU, May 2015)**

Ans. The two principal criteria in selecting a hash function are that it should be easy and quick to compute and that it should achieve an even distribution of the keys that actually occur across the range of indices. If we know in advance exactly what keys will occur, then it is possible to construct has functions that will be very efficient, but generally we do not know in advance what keys will occur. Therefore, the usual way is for the hash function to take the key, chop it up, mix the pieces together in various ways, and thereby obtain an index that (like the pseudorandom numbers generated by computer) will be uniformly distributed over the range of indices.

Q 35. Consider the following numbers are stored in an array A :

32, 51, 27, 85, 66, 23, 13, 57

Apply Bubble sort algorithm to the array A and show each pass separately. **(PTU, May 2016)**

Ans.

(32)	(51)	27	85	66	23	13	57
32	(51)	(27)	85	66	23	13	57
32	27	(51)	(85)	66	23	13	57
32	27	51	(85)	(66)	23	13	57

	32	27	51	66	(85)	(13)	57
	32	27	51	66	23	(85)	(57)
	32	27	51	66	23	13	85
	32	27	51	66	23	13	57
Pass - 1	(32)	(27)	51	66	23	13	57
	27	(32)	(51)	66	23	13	57
	27	32	(51)	(66)	23	13	57
	27	32	51	(66)	(23)	13	57
	27	32	51	23	(66)	(13)	57
	27	32	51	23	13	(66)	(57)
	27	32	51	23	13	57	66
Pass - 2	(27)	(32)	51	23	13	57	
	27	(32)	(51)	23	13	57	
	27	32	(51)	(23)	13	57	
	27	32	23	(51)	(13)	57	
	27	32	23	13	51	57	
Pass - 3	(27)	(32)	23	13	51		
	27	(32)	(23)	13	51		
	27	23	(32)	(13)	51		
	27	23	13	32	51		
Pass - 4	(27)	(23)	13	32			
	23	(27)	(13)	32			
	23	13	27	32			
Pass - 5	(23)	(13)	27				
	13	23	27				

Pass - 6

So sorted array is

13, 23, 27, 32, 51, 57, 66, 85

Q 36. Consider the following numbers are stored in an array A :
31, 52, 28, 84, 65, 24, 14, 56
Apply Bubble sort algorithm to the array A and show each pass separately.
(PTU, May 2014)

Ans. Pass 1

	(31)	(52)	28	84	65	24	14	56	
	31	(52)	(28)	84	65	24	14	56	(Compare 31 and 52, no swap)
	31	28	(52)	(84)	65	24	14	56	(Compare 52 and 28, swap)
	31	28	52	(84)	(65)	24	14	56	(Compare 52 and 84, no swap)
	31	28	52	65	(84)	(24)	14	56	(Compare 84 and 65, swap)
	31	28	52	65	24	(84)	(14)	56	(Compare 84 and 24, swap)
	31	28	52	65	24	14	(84)	(56)	(Compare 84 and 14, swap)
	31	28	52	65	24	14	56	84	(Compare 84 and 56, swap)
Pass 2 :	(31)	(28)	52	65	24	14	56	84	
	28	(31)	(52)	65	24	14	56	84	
	28	31	(52)	(65)	24	14	56	84	
	28	31	52	(65)	(24)	14	56	84	
	28	31	52	24	(65)	(14)	56	84	
	28	31	52	24	14	(65)	(56)	84	
	28	31	52	24	14	56	65	84	
Pass 3 :	(28)	(31)	52	24	14	56	65	84	
	28	(31)	(52)	24	14	56	65	84	
	28	31	(52)	(24)	14	56	65	84	
	28	31	24	(52)	(14)	56	65	84	
	28	31	24	14	52	56	65	84	
Pass 4 :	(28)	(31)	24	14	52	56	65	84	
	28	(31)	(24)	14	52	56	65	84	
	28	24	(31)	(14)	52	56	65	84	
	28	24	14	31	52	56	65	84	
Pass 5 :	(28)	(24)	14	31	52	56	65	84	
	24	(28)	(14)	31	52	56	65	84	
	24	14	28	31	52	56	65	84	
Pass 6 :	(24)	(14)	28	31	52	56	65	84	
	14	24	28	31	52	56	65	84	

Pass 7 : All list are sorted.

Q 37. Write an algorithm to implement Quick sort. Write the steps to sort the following elements by quick sort method :
17, 28, 6, 87, 46 **(PTU, May 2018)**

Ans. Quick sort algorithm : *Refer to Q.No. 5*

The given data is

(17), 28, (6), 87, 46

Pass 1	:	6	28	(17)	87	46
Pass 2	:	6	17	(28)	87	46)
Pass 3	:	6	17	28	(87	46)
Pass 4	:	6	17	28	46	87

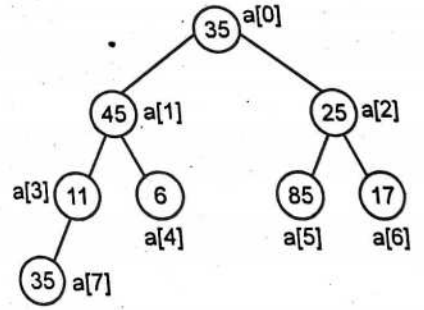
Q 38. Write ADT operations for heap sort. Using the above algorithm sort the following:
25, 45, 25, 11, 6, 35, 17, 35 **(PTU, May 2018)**

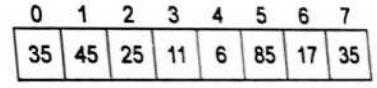
Ans. Heapsort is a comparison-based sorting algorithm to create a sorted array (or list), and is part of the selection sort family. Although somewhat slower in practice on most machines than a well-implemented quicksort, it has the advantage of a more favourable worst-case O (log n) run time. Heapsort is an inplace algorithm, but is not a stable sort.

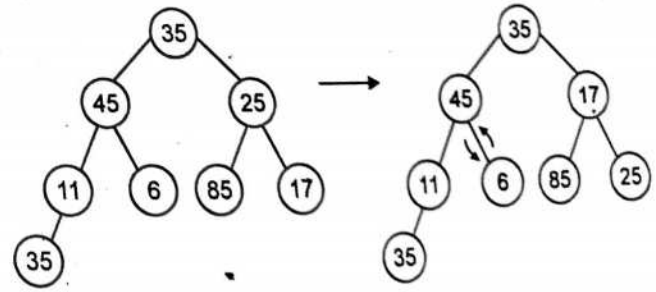
It is a two-step algorithm : The first step is to build a heap out of the data.

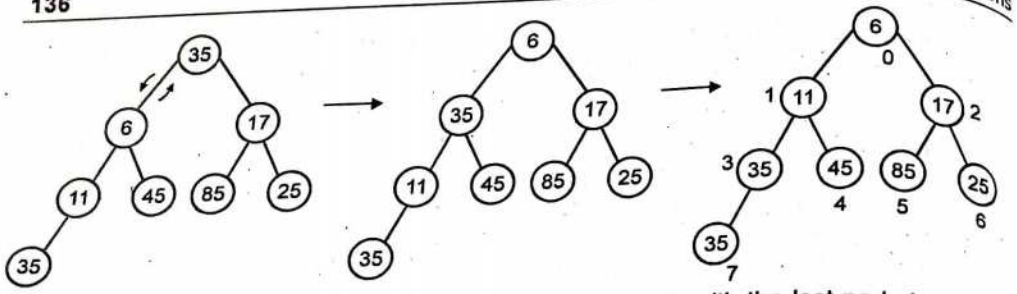
The second step begins with removing the largest element from the heap. We insert the removed element into the sorted array. For the first element, this would be position 0 of the array. Next we reconstruct the heap and remove the next largest item, and insert it into the array. After we have removed all the objects from the heap, we have a sorted array. We can vary the direction of the sorted elements by choosing a min-heap or max-heap in step one.

35, 45, 25, 11, 6, 85, 17, 35 be a list of elements we can construct a tree for these elements as follows :

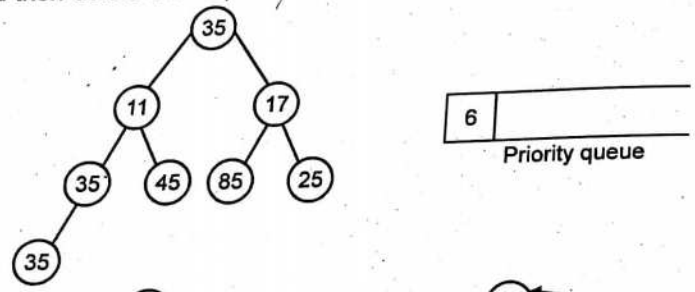


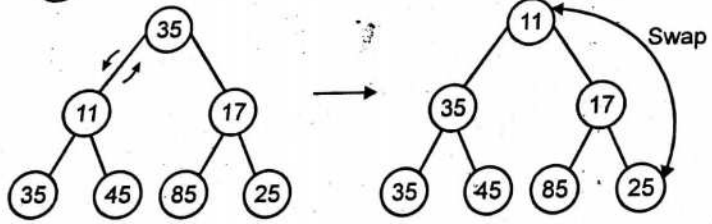


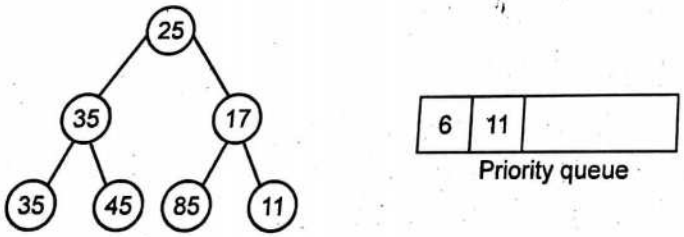


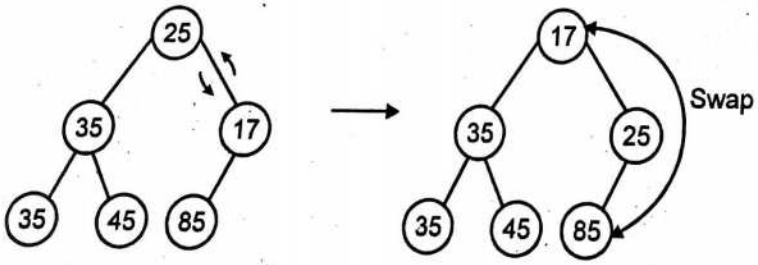


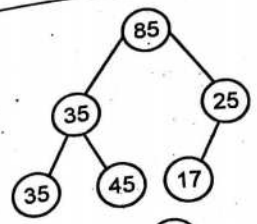
Thus the heapified tree is constructed. Now swap root node with the last node in the tree each time and then delete the last node and insert it in the priority queue.

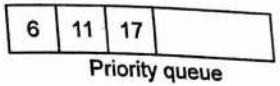


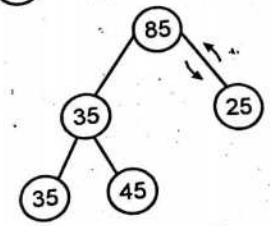


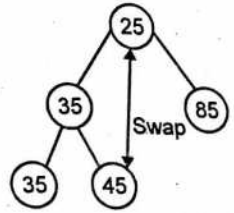


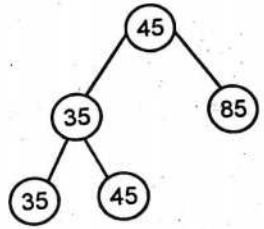


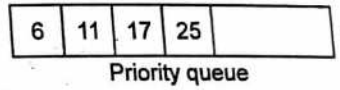


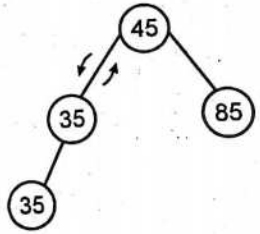


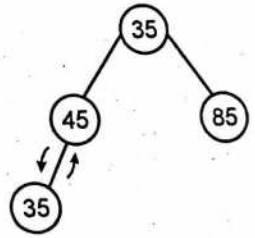


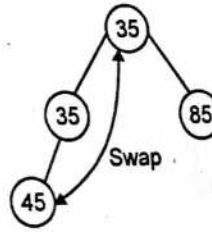


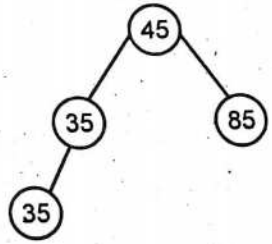


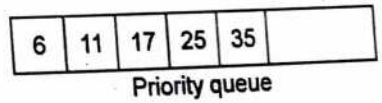


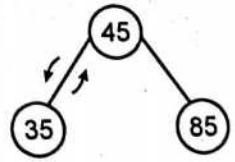


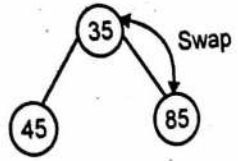


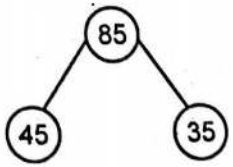


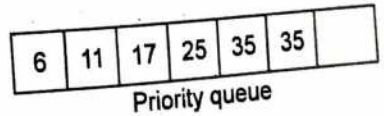


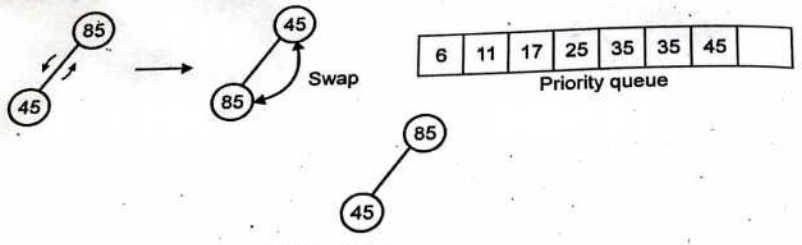












(85) Delete it and insert in priority queue

Finally we get priority queue as

6	11	17	25	35	35	45	85

If we delete the elements of queue one by one we will get the sorted list as

6, 11, 17, 25, 35, 35, 45, 85

Q 39. Write an algorithm to implement Merge sort. **(PTU, May 2018)**

Ans. Algorithm :

1. MERGING (A, R, B, S, C) : Let A and B be sorted arrays with R and S elements respectively. This algorithm merges A and B into an array C with N = R + S elements.

1. *[Initialize]* Set NA : = 1, NB : = 1 and PTR : = 1
2. *[Compare]* Repeat while NA ≤ R and NB ≤ S : If A [NA] < B [NB] then :
   (a) [Assign element from A to C] set C [PTR] : = A [NA]
   (b) [Update pointers] set PTR : PTR + 1 and NA : = NA + 1 else
   (a) [Assign element from B to C] Set C[PTR] : = B [NB]
   (b) [Update pointers] set PTR: = PTR + 1 and NB : = NB + 1 [End of if structure]
   [end of loop]
3. *[Assign remaining elements to C]*
   If NA > R then
   Repeat for K = 0, 1, 2, ... S – NB :
   Set ([PTR + K] : = B [NB + k]
   [End of loop]
   Else;
   Repeat for K = 0, 1, 2 ..... R – NA:
   Set ([PTR + k]; = A {NA + k]
   [End of Loop]
   [End of if structure]
4. Exit

2. MERGE (A, R, LBA, S, LBB, C, LBC) : This procedure merges the sorted array A and B into array C.

1. Set NA : = LBA, NB : = LBB, PRT : = LBC, UBA : = LBA + R' – 1
   UBB : = LBB + S – 1
2. Same as above algorithm except R is replaced by UBA and S by UBB.
3. Same as above algorithm except R is replaced by UBA and S by UBB.
4. Return.

3. MERGE PASS (A, N, L, B) : The N element array A is composed of sorted sub arrays

where each sub array has L elements except possibly the last sub array, which may have fewer than L elements. The procedure merges the pairs of sub arrays of A and assign them to the array B.

1. Set Q : INT (N/2*L) ; S = 2*L*Q and R : = N – S
2. [Use above procedure to merge the Q paris of subarrays]
3. Repeat for J = 1, 2, ... Q;
   (a) Set LB : 1 + (2 * J – 2) * L [Finds Lower bound of first array]
   (b) Call MERGE (A, L, LB, A, L, LB + L, B, LB)
   [End of Loop]
   (c) [Only one subarray left?]
   If R ≤ L then
   Repeat for J = 1, 2, ... R:
   Set B (S + L) : = A (S + J)
   [End of Loop]
   Else :
   Call MERGE (A, L, S + 1, A, R, L + S + 1, B, S + 1)
   [End of If structure]
4. Return

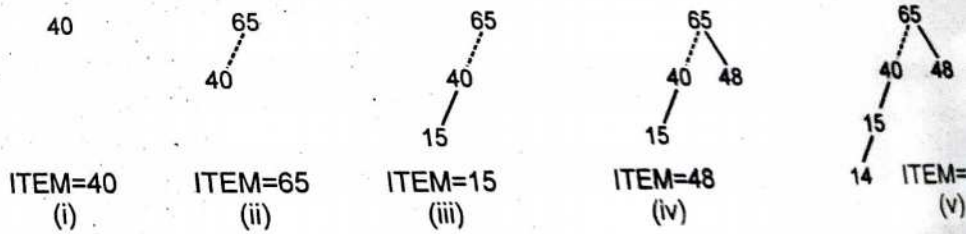
4. MERGE SORT (A,N) : This algorithm sorts the N element array A using an auxiliary array B.

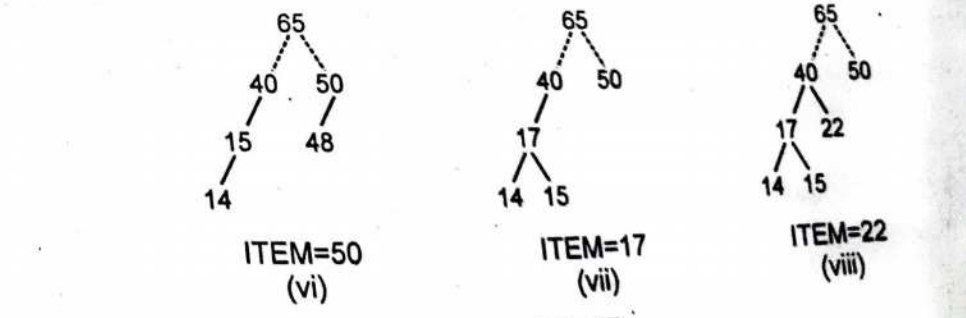
1. Set L : = 1 [Initializes the number of elements in the subarrays]
2. Repeat steps 3 to 6 while L < N
3. Call MERGE PASS (A, N, L, B)
4. Call MERGE PASS (B, N, 2 * L, A)
5. Set L : = 4 * L
   [End of step 2 Loop]
6. EXIT.

Q 40. Build a heap H from the following list of numbers :

40, 65, 15, 48, 14, 50, 17, 22 **(PTU, May 2**

Ans. Creating heap





❑❑❑

Chapter 5

Graph

Contents

Basic Terminology and Representations, Graphs search and traversal algorithms and complexity analysis.

POINTS TO REMEMBER

- A graph is a non-linear data structure that consists of a collection of vertices (or nodes) and a collection of edges, with each edge joining one vertex to another. It is represented as G = (V, E) where V is a set of vertices and E is a set of edges.
- A directed graph or digraph is a graph whose each edge is an ordered pair of vertices. Edges of a digraph have a direction associated with them which indicate how it may be traversed.
- A undirected graph is a graph in which there is a no direction associated with any of the edges. Presence of an edge connecting two nodes indicates that we can traverse in either direction.
- Two vertices in a graph are said to be adjacent it there exists an edge between them.
- A path is sequence of vertices traversed by following the edges between them. A path is simple it is has no repeating vertices with an exception that V_0 (starting vertex in path) may equal N_n (last vertex).
- A cycle is a simple path consisting of sequence of vertices such that the starting and ending vertex are the same.
- A loop is a special case of cycle in which an edge begins and ends with the same vertex.
- In an undirected graph, the degree of a vertex is the number of edges originating from it.
- Indegree of a vertex in a directed graph is the number of edges entering the vertex. Outdegree of a vertex in a directed graph is the number of edges leaving the vertex.
- A graph is said to be complete if there are edges from any vertex to all other vertices.
- An undirected graph is said to be connected if every vertex is reachable from others by following some path.
- A graph is termed as weighted graph if all the edges in it are labeled with some weights. A weighted edge between two vertices V_i and V_j that has a scalar value w associated with it is written as (V_i, V_j, W).
- A multi graph is a graph in which these are two or more edges connecting the same vertices of graphs.

- The entries of the adjacency matrix provides information regarding whether two vertices of a graph are adjacent or not.
- In a weighted graph, if an edge with a given weight exist between two vertices then we store that weight as the entry in the weighted adjacency matrix instead of 1 as in adjacency matrix.
- In an adjacency matrix of an undirected graph, the sum of all the entries of the adjacency matrix in twice the number of edges in the graph.
- In an adjacency matrix of a directed graph, the outdegree of a vertex is equal to the sum of the entries of the row corresponding to it and indegree of vertex is equal to the sum of the entries of the column corresponding to it.
- The path matrix determines whether there exists a path between any two vertices or not.
- Insertion, deletion and traversing are the basic operations that can be performed on a graph.
- The shortest path between any two vertices of a weighted graph is the sequence of connected vertices so that the sum of the cost of edges that interconnect them is minimum.
- Dijkstra's shortest path algorithm determines the shortest path from a single source vertex to all other vertices of a graph.
- Floyd's shortest path algorithm determines the path with the minimum cost from every vertex to every other vertex.
- A spanning tree in G is an acyclic subgroup of G that includes every vertex of G and is connected.
- A minimum spanning tree is a spanning tree whose sum of weights of all its edges is minimum.
- In a strongly connected graph, if there is a path from vertex U to vertex V then there should be another path from vertex V to vertex U.
- A connected graph is biconnected if there are no vertices whose removal disconnect the rest of the graph.
- A biconnected component of a graph is a maximum set of vertices that is biconnecte

QUESTION-ANSWERS

Q 1. What is Kruskal's algorithm used for in graphs? (PTU, May 20

Ans. The Kruskal's algorithm is used for finding the minimum spanning tree of the giv graph. In kruskal's algorithm, edges are added to the spanning tree in increasing order cost. If the edge form a cycle in the spanning, it is discarded.

(PTU, Dec. 20

Q 2. Explain representation of graph.

Ans. A graph 'G' consists of two things :

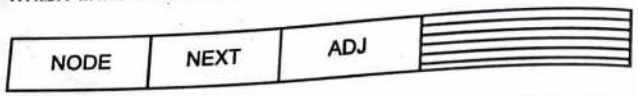
1. A set 'V' of elements called nodes.

2. A set 'E' of edges s.t. each edge in 'E' is identified with a unique pair of nodes in 'V' denoted by e = (u, v). Its representation is of two types :

1. Sequential representation : It is represented with adjacency matrix. It is a m x n matrix defined as $a_{ij} = \begin{cases} 1, & \text{if } v_i \text{ is adjacent to } v_j \\ 0, & \text{otherwise} \end{cases}$

2. Linked representation : It consists of two lists :

(a) A **node list** which links all the nodes of graph. The structure of node list is :



(b) An **edge list** which links all the adjacent nodes of a node. Structure of edge node is :

Q 3. Explain Depth-First Search. **(PTU, Dec. 2006)**

Ans. In Depth First Search i.e., DFS, firstly we examine starting node 'A'. Then, we examine each node 'N' along a path P, which begins at 'A', i.e. we process a neighbour of 'A' and so on. After coming to a "dead end", we back track on P until we can continue along another path P'. In this case, we use stack instead of queues.

Q 4. Distinguish between BFS and DFS. **(PTU, Dec. 2007)**

Ans. Breadth First-Search (BFS) : The general idea behind a breadth first search, beginning at a starting node A is as follows. First, we examine the starting node A. Then we examine all the neighbours of A. And so on. Naturally, we need to keep track of neighbours of a node and we need to guarantees that no node is processed more than once. This is accomplished by using a queue to hold nodes that are waiting to be processed, and by writing a field STATUS which tells us the current status of any node.

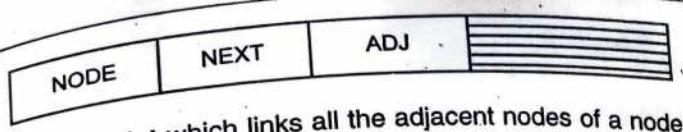
Depth First Search (DFS) : The general idea a depth first search begining at a storing ode A is as follows : First we examine the starting node A. Then we examine each node N ong a path P which begins at A ; that is, we procced a neighbour of A, then a neighbour of neighbour of A and so on. After coming to a "dead end" that is to be end of the path P, we ck track on P until we can continue along another path P. And so on.

In the nut shell results, DFS (depth first search) is similar to BFS (breadth first search) ept now we use a stack instead of queue for DFS (depth first search).

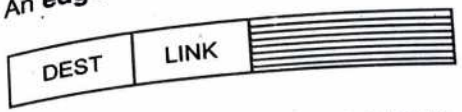
Q 5. Discuss how graphs are represented in memory using linked list. **(PTU, May 2007)**

Ans. The linked representation of graph contains two lists :

1. A **node list** which links all the nodes of the graph. The structure of node list is :

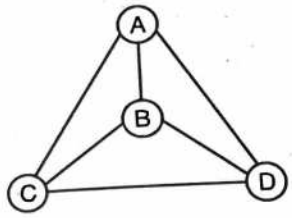


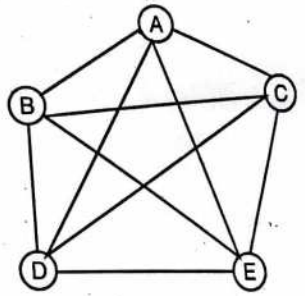
2. An **edge list** which links all the adjacent nodes of a node. Structure of edge list is :



Q 6. What are complete graph? Write any two applications of complete graph. **(PTU, Dec. 2007)**

Ans. A graph is said to be complete if there are edges from any vertex to all other vertices. In such type of graph, each vertex is adjacent to every other vertex. For example : The graph G1 and G2 are complete graph.





Complete Graph

Q 7. What is adjacency matrix representation of a graph in memory? **(PTU, Dec. 2008**

Ans. Suppose G is a simple directed graph with m nodes and suppose the nodes of have been ordered adn called $G_1, G_2, \ldots G_m$. Then adjacency matrix $A = (d_{ij})$ of graph G the m x m matrix defined as follows :

$$a_{ij} = \begin{cases} 1 & \text{if G is adjacent to } G_j, \text{that is, there is an edge } (G_i, G_j) \\ 0 & \text{otherwise} \end{cases}$$

such a matrix A, which contains entries of 0 and 1 is called bit matrix or a boole matrix.

for **example,**

Consider a graph G, suppose the nodes are stored is memory in a linear array DATA follows :

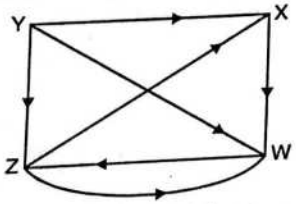
DATA : X, Y, Z, W

Then we assume that $G = X, G_2 = Y, G_3 = Z, G_1 = W$

The adjacency matrix A to G is as follows :

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Note that 1's in A is equal to number of edge G



Q 8. Define connected and weakly connected graph. **(PTU, May 2009)**

Ans. In mathematics and computer science, connectivity is one of the basic concepts of graph theory. It is closely related to the theory of network flow problems. The connectivity of a graph is an important measure of its robustness as a network.

In an undirected graph G, two vertices u and v are called connected if G contains a path from u to v. Otherwise, they are called disconnected. (Recall that vertices connected by an edge,i.e., by path of length 1, are called adjcent.) A graph is called connected if every pair of distinct vertices in the graph can be connected through some path.

A connected component is a maximal connected subgraph of G. Each vertex belongs to exactly one connected component, as does each edge.

A directed graph is called weakly connected if replacing all of its directed edges with undirected edges produces a connected (undirected) graph. It is strongly connected or strong if it contains a directed path from u and v and a directed path from v to u for every pair of vertices u, v. The strong components are the maximal strongly connected subgraphs.

Q 9. Explain topological sorting on graphs. **(PTU, May 2009)**

Ans. In graph theory, a topological sort or topological ordering of a directed acyclic graph (DAG) is a linear ordering of its nodes in which each node comes before all nodes to which it has outbound edges. Every DAG has one or more topological sorts.

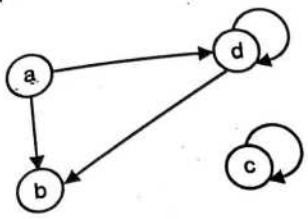
More formally,define the partial order relation R over the nodes of the DAG such that xRy if and only if there is a directed path from x to y. Then, a topological sort is a linear extension of this partial order, that is, a total order compatible with the partial order.

Q 10. What is directed graph? **(PTU, Dec. 2009)**

Ans. A directed graph G is defined as an ordered path where, V is a set of vertices and he ordered pairs in E are called edges on V. A directed graph can be represented geometrically s a set of marked points (called vertices) V with a set of arrows (called edges) E between airs of points (or vertices) so that there is at most one arrow from one vertex to another rtex. For example following figure shows a directed graph, where

Graph

$G = \{a, b, c, d\}, \{(a, b), (a,$



Q 11. How graph is represented in memory? **(PTU, May 2010)**

Ans. Graph is represented in memory by a linked representation, also called an adjacancy structure. First we store all the vertices of the graph in a list and then each adjacent vertices will be represented using linked list node. The terminal vertex of an edge is stored in a structure node and linked to a corresponding initial vertex in the list. For example, following directed graph of fig. 1 can be represented using linked list as in fig. 2.

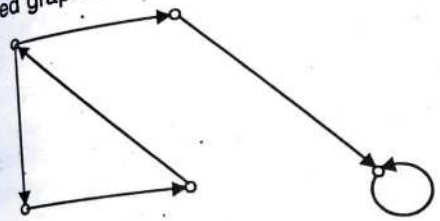


Fig. 1

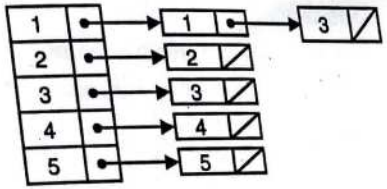


Fig. 2

Q 12. What are various applications of graphs? Write an algorithm for traversal in a graph. **(PTU, May 2004)**

Ans. Various applications of graphs include :

1. These are used to distinguish between chemical compounds having same molecular but difference structure formula.
2. Graphs are used to study the network of Internet, i.e. Worldwide web. Hence, these are used to establish connection between different system.
3. Graphs are used to implement a circuit on a circuit board.
4. Graphs are also used to implement transportation services and communication networks.

Traversal in a graph : A graph can be traversed in two ways : either Breadth First Search (BFS) or Depth First Search.

1. BFS : This algorithm executes a BFS on a graph G beginning at a starting node A.

1. Initialize all nodes to ready state (STATUS = 1).
2. Put the starting node 'A' in QUEUE and change its status to waiting state (STATUS = 2).
3. Repeat steps 4 and 5 until queue is empty.
4. Remove the front node N of QUEUE Process N and change the status of N to processed state (STATUS = 3).

5. Add to rear of QUEUE all the neighbours of N that are in steady state. (STATUS = 1), & change their status to waiting state (STATUS = 2).
6. Exit.

2. Depth First Search : This algorithm executes a DFS on a graph G beginning at starting node 'A'.

1. Initialize all nodes to ready state (STATUS = 1).
2. Push starting node 'A' onto STACK and change its status to waiting state (STATUS = 2).
3. Repeat steps 4 and 5 until STACK is empty.
4. Pop the top node N of STACK. Process N and change its status to processed state (STATUS = 3).
5. Push onto STACK all neighbours of N that are still in ready state (STATUS = 1) and change their status to waiting state (STATUS = 2).
6. Exit.

Q 13. How minimal spanning tree for a graph is generated? Explain with an algorithm. **(PTU, Dec. 2010)**

Ans. To obtain the minimum spanning tree for a graph, J.B Kruskal developed an algorithm in 1956 known as Kruskal's algorithm. This algorithm builds a minimum spanning tree by adding one edge at a time to a subgraph. Each time an edge with the lowest cost is chosen such that it does not create a cycle with the edges already chosen if it does, we reject that edge. This process of adding edges continues until (n – 1) edges are added to the n vertices spanning tree. If (n – 1) edges don't from a cycle then the resulting spanning tree is the minimum spanning tree.

Kruskel's Algorithm is as follow :

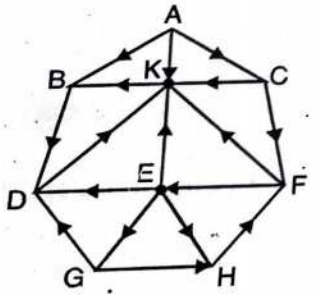
Step 1. Sort the edges of the graph is ascending order in accordance to their weight.

Step 2. Select the edge of least weight and add it to the tree which is initially empty.

Step 3. Select the edges not previously selected, the edge of the least weight that does form a cycle together with the edges already included. Add this edge into the tree.

Step 4. Repeat step 3 until the tree contains n – 1 edges or all the edges are exhausted. If the tree so generated contains (n – 1) edges than this tree is the minimum spanning otherwise no spanning tree is possible for the graph.

Q 14. Apply Depth First Search (DFS) to following algo. **(PTU, Dec. 2007)**



Adjoining List
A : B, C, K
B : D
C : K, F
D : K
E : K, D, G, H
F : E, K
G : D, H
H : E, F
K : B

(a) Initial push G into stack as follows :

STACK ; G

(b) Pop and print top element G and then push onto the stack all the neighbours of K as follows:

Print G : STACK D, H

(c) Pop and print top element H and then push onto the stack all the neighbours of H as follows:

Print H : STACK D, E, F

(d) Pop and print the top element F and then push onto stack all neighbours of F as follows :

Print F : STACK D, E, K

(e) Pop and print the top element K and then push onto stack all neighbours of K as follows :

Print K : STACK D, E, B

(f) Pop and print the top element B and then push onto stack all the neighbours of B as follows:

Print B : STACK D, E

(g) Pop and print the top element E and push onto stack all the neighbours of E as follows :

Print E : STACK D

(h) Pop and print top element K and push into stack

Print P : STACK

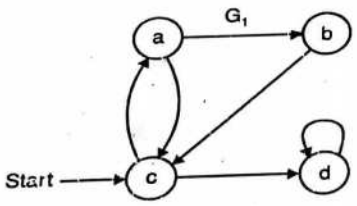
The stack is now empty, so the depth for search of G starting at G is now comple Accordingly now nodes will be printed as follows :

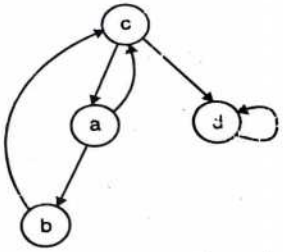
G, H, F, K, B, E, D

are precisely the nodes which are reacharable form G.

Q 15. Write the applications of depth first traversal of a graph. (PTU, May 2009)

Ans. Depth-First Traversal : A depth-first traversal of a tree always starts at the root of the tree. Since a graph has no root,when we do a depth-first traversal, we must specify the vertex at which to begin. A depth-first traversal of a tree visits a node and then recursively visits the subtrees of that node. Similarly, depth-first traversal of a graph visits a vertex and then recursively visits all the vertices adjacent to that node. The catch is that the graph may contain cycles, but the traversal must visit every vertex at most once.The solution to the problem is to keep track of the nodes that have been visited, so that the traversal does not suffer the fate of infinite recursion.





For example,fig. illustrates the depth-first traversal of the directed graph G_1 starting from vertex c. The depth-first traversal visits the nodes in the order

c.a.b.d.

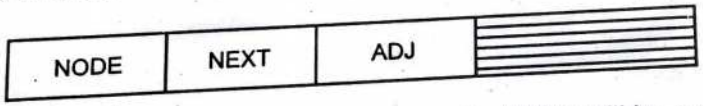
Applications : Algorithms where DFS is used :

- Finding connected components.
- Topological sorting.
- Finding 2- (edge or vertex)-connected components.
- Finding strongly connected components.
- Solving puzzles with only one solution, such as mazes. (DFS can be adapted to find all solutions to a maze by only including nodes on the current path in the visitede set.)

Q 16. Explain linked representation of graphs. **(PTU, Dec. 2004)**

Ans. The linked representation of graph contains two lists :

1. Node list : A node list links all the nodes of graph. The structure of node list is :



Here, NODE will be the name or key value of node, NEXT will be a pointer to the next de in the list NODE and ADJ will be a pointer to Ist element in adjacency list of node, ich is maintained in list EDGE. The shaded area indicates that there may be other rmation in the record, such as in degree and out degree of node, the STATUS of node, l so on.

2. Edge list : An edge li

The field DEST will point to the location in the list NODE of destination or terminal node of edge.The field LINK will link together the edges with the same initial node, i.e., the nodes in the same adjacency list. The shaded area indicates that there may be other information.

Q 17. Explain Warshell's algorithm for shortest paths. **(PTU, May 2005)**

Ans. Let 'G' be a directed graph with 'm' nodes $v_1, v_2, \ldots, v_m$. Suppose, we want to find the path matrix of graph 'G'. First, we define 'm' square matrices (mxm) P_0 to P_m as follows :

$$P_K[i, j] = \begin{cases} 1 & \text{if there is simple path from } v_i \text{ to } v_j \text{ which doen't use any other nodes except passing } v_1 \text{ to } v_K \\ 0 & \text{otherwise} \end{cases}$$

It means $P_0[i, j] = 1$, if there is an edge from v_i to v_j which doesn't use any other node except possibly v_1

$P_K[i, j] = 1$ can occur if one of following 2 cases occur :

1. There is a simple path from v_i to v_j which doesn't use any other node except only $v_1 \ldots v_{K-1}$, i.e. $P_{K-1}[i, j] = 1$.
2. There is a simple path from θ_i to v_j and a simple path from v_j to v_K, where each path doesn't use any nodes except possibly $v_1 \ldots v_{K-1}$.

$$P[i, j] = 1 \text{ or } (P_K[i, K] \text{ and } P_{K-1}[K, j])$$

Accordingly, the elements of matrix P_K can be obtained by

$$P_K[i, j] = P_{K-1}[i, j] \vee (P_{K-1}[i, K] \wedge P_{K-1}[K, j])$$

Where we use logical operations of ^ (AND) and V (OR).

Q 18. Write Warshall's algorithm for shortest path. **(PTU, Dec. 20**

Ans. Warshell's algorithm

A directed graph 'G' with 'M' nodes is maintained in memory by its adjacency matr

Thus algorithm finds the path matrix 'P' of graph 'G'.

1. Repeat for I, J = 1, 2, ... M

   If A [I, J] = 0, then :

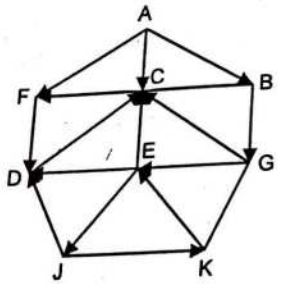
   set P [I, J] = 0 ;

   ELSE :

   Set P [I, J]= 1

2. Repeat steps 3 and 4 for K = 1, 2, ... M
3. Repeat step 4 for I = 1, 2, ... M
4. Repeat for J = 1, 2, ... M
   Set P [I, J] = P [I, J] V (P (I, K) ^ P (K, J))
5. Exit.

Q 19. Apply Breadth First Search (BFS) on following graphs. (PTU, May 2007)
Ans.



Ans. Adjacancy list :

A : F, C, B
B : G, C
C : F
D : C
E : D, C, J
F : D
G : C, E
J : D, K
K : E, G

1.	F = 1	QUEUE : A
	R = 1	0 : ϕ
2.	F = 2	Q : A, B, C, B
	R = 4	0 : ϕ, A, A, A
3.	F – 3	Q : A, F, C, B, D
	R = 5	0 : 0, A, A, A, F
4.	F = 4	Q : A, F, C, B, D
	R = 5	0 : 0 F A A F
	F = 5	Q : A, F, C, B, D, G
	R = 6	0 : 0, A, A, A, F, B
	F = 6	Q : A, F, C, B, D, G
	R = 6	0 : 0, A, A, A, F, B
7.	F = 7	Q : F, F, C, B, D, G, E
	R = 7	0 : 0, A, A, A, F, B, G
8.	F = 8	Q : A, F, C, B, D, G, E, J
	R = 8	0 : 0, A, A, A, F, B, G, E

We now back trace from J, using any origin. To find P. Thus,
J ← E ← G ← B ← A
is required path P.

Q 20. How graphs are represented in memory? Write a procedure to delete a node from the graph. (PTU, May 2008)

Ans. There are two standard ways of maintaining a graph G in memory of a computer.
1. Sequential representation of G, is by means of its adjacency matrix.
2. The other way is called the linked representation of G_1 is by means of linked list of neighbours.

Adjacency Matrix :

Suppose G is a simple directed graph with m nodes, and suppose the nodes of G have been ordered and are called $v_1, v_2, ... v_m$. Then adjacency matrix $A = (a_{ij})$ of the graph in G is the m × n matrix defined as follows :

$$a_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is adjacent to } v_j, \text{ that is, if there} \\ 0 & \text{is an edge } (v_i, v_j) \text{ otherwise} \end{cases}$$

such a matrix A, which contains entries of any 0 and 1 is called a bit matrix or boolean matrix.

PATH Matrix :

Let G be a simple directed graph with m nodes $v_1, v_2, ... v_m$. The path matrix or reachability matrix of G is the m-square matrix $P = (p_{ij})$ defined as follows :

$$P_{ij} = \begin{cases} 1 & \text{If there is a path from } v_i \text{ to } v_j \\ 0 & \text{otherwise} \end{cases}$$

A procedure to delete a node from the graph as follows :
DELETE (INFO, LINK, START, AVAIL, ITEM, FLAG)
This algorithm delete the first node in the list containing ITEM or sets FLAG = FALS when ITEM does not appear in the list.

Step 1 : [List Empty] : It START = NULL then set FLAG = FALSE and return.
Step 2 : [ITEM in the first node] If INFO [START] = ITEM then
Set PTR = START, START = LINK [START]
LINK [PTR] = AVAIL, AVAIL = PTR
FLAG = TRUE and return
[end of if structure]

Step 3 : Set PTR = LINK [START] and SAVE = START [initializes pointers]
Step 4 : Repeat steps 5 and 6 while PTR ≠ NULL
Step 5 : It INFO [PTR] = ITEM then
Set LINK [SAVE] = LINK [PTR], LINK [PTR] = AVAIL
AVAIL = PTR, FLAG = TRUE and return
[END of if structure]
Step 6 : Set SAVE = PTR and PTR = LINK [PTR] [updates pointers]
[step of step 4 loop]
Step 7 : Set FLAG = FALSE and return.

Q 21. Explain the Warshall's algorithm for finding the path in graph. **(PTU, Dec. 2008)**

Ans. Let G be a directed graph with m nodes G_1, G_2, G_n suppose we want to find the path matrix P of graph G.

Warshall gave algorithm for this purpose that is much more efficient than calculating powers of adjacency matrix.

According the elements of matrix P_k can be obtained by

$$P_k [i, j] = P_{k-1} [i j] \wedge [P_{k-1} [(i k] \wedge P_{k-1} (k,j)]$$

Warshall Algorithm : A directed graph G with M nodes is maintained memory by its adjacency matrix A. This algorithm finds the boolean path matrix P of graph G.

Step 1. Repeat for I, J = 1, 2, ... M [Initializing P]
If A [I, J] = 0 then set P [I, J] = 0
Else set P [I, J] = 1
[End of loop]
Step 2. Repeat step 3 and 4 for K = 1, 2, ... M [Updates P]
Step 3. Repeat step 4 for I = 1, 2, ... M.
Step 4. Repeat for J = 1, 2, ... M
Set p [I, J] = P [I, J] V (P [I, K] ∧ P [K, J])
[End of loop]
[End of step 3 loop]
[End of step 2 loop]
Step 5. Exit.

Q 22. Discuss the Dijkstra's algorithm for finding the shortest paths from a source all other vertices in a directed graph. What is its time complexity? (PTU, May 2009)

Ans. Algorithm : In graph theory, the shortest path problem is the problem of finding a path between two vertices (or node) such that the sum of the weights of its constituent edges is minimized. An example is finding the quickest way to get from one location to another on a road map; in this case, the vertices represent locations and the edges represent segments of road and are weighted by the time needed to travel that segment.

Dijkstra's algorithm, conceived by Dutch computer scientist Edsger Dijkstra in 1959, is a graph search algorithm that solves the single-source shortest path problem for a graph with nonnegative edge path costs, producing a shortest path tree. This algorithm is often used in routing. An equivalent algorithm was developed by Edward F. Moore in 1957

Let's call the node we are starting with an **initial node.** Let a **distance of a node Y** be the distance from the **initial node** to it. Dijkstra's algorithm will assign some initial distance values and will try to improve them step-by-step.

1. Assign to every node a distance value. Set it to zero for our initial node and to infinity for all other nodes.
2. Mark all nodes as unvisited. Set initial node as current.
3. For current node,consider all its unvisited neighbours and calculate their distance (from the initial node). For example, if current node (A) has distance of 6, and an edge connecting it with another node (B) is 2, the distance to B through A will be 6 + 2 = 8. If this distance is less than the previously recorded distance (infinitly in the beginning, zero for the initial node), overwrite the distance.
4. When we are done considering all neighbours of the current node, mark it as visited. A visited node will not be checked ever again; its distance recorded now is final and minimal.
5. Set the unvisited node with the smallest distance (from the initial node) as the next "current node" and continue from step 3.

Complexity of Dijkstra's Algorithm : With adjacency matrix representation, the running time is O (n2). By using an adjacency list representation and a partially order tree data structure for organizing the set V – S, the complexity can be shown to be

O (e log n)

where e is the number of edges and n is the number of vertices in the graph.

Q 23. How graph is represented in memory? **(PTU, May 2011)**

Ans. It is possible to represent graphs in computer memory with a variety of different data structures. One strategy is to use an dimensional array in which the row and column headers represent different vertices in the graph. A one way edge between for example, vertex one and there is denoted by a positive value in array position (1, 3). Another method for representing graphs is as a more complicated linked list structure. Each vertex in the graph is a node in a master linked list. Another linked list emantes from each vertex node and denotes the vertices directly adjacent to a given source vertex. This method called an adjacency list.

Q 24. Explain the following :
(a) Depth first search.
(b) Breadth first search. **(PTU, May 2011, 2010**

Ans. (a) Depth-First Search : The general idea depth-first search beginning a starting node A is as follows. First we examine the starting node A. Then we examine each node along a path P which begins at A : that is, we process a neighbour of A_1 then a neighbour a neighbour of A and so on. After coming to a "dead end", that is to the end of the path P,

backtrack on P until we can continue along another, path P'. And so on (This algorithm is similar to the in order traversal of a binary tree, and the algorithm is also similar to the array one might travel through amaze). The algorithm is very STATUS is used to tell us the current status of a node. The algorithm follows :

Algorithm : This algorithm executes a depth first search on a graph of beginning at a starting node A.

1. Initialize all nodes to the ready state (STATUS = 1).
2. Push the starting node A onto STACK and change its status to the waiting state (STATUS = 2).
3. Repeat steps 4 and 5 until STACK is empty.
4. Pop the top node N of STACK. Process N and change its status to the processed state (STATUS = 3).
5. Push onto STACK all the neighbours of N that are still in the ready state (STATUS = 1) and change their status to the waiting state (STATUS = 2).
   [End of the step 3 loop]
6. Exit.

(b) Breadth First Search : The general idea behind a breadth-first search beginning at a starting node A is as follows. First we examine the starting node A. Then we examine all the neighbours of A. Then we examine all the neighbours of the neighbors of A and so on. Naturally, we need to keep track of the neigbours of a node, and we need to guarantee that no node is processed, and by using a field STATUS which tells us the current status as any node. The algorithm follows.

Algorithm A : This algorithm executes a breadth first search on a graph a beginning at starting node A.

1. Initialize all nodes to the ready state (STATUS = 1).
2. Put the starting node A in queue and change its status to the waiting state (STATUS = 2).
3. Repeat steps 4 and 5 until queue is empty :
4. Remove the front node N of queue. Process N and change the status of N to the processed a state (STATUS = 3).
5. Add to the rear of QUEUE all the neighbors of N that are in the state (STATUS = 2)
   [End of step 3 loop]
6. Exit.

Q 25. Define data structures graph. How they are represented in memory?
(PTU, May 2019 ; Dec. 2007)

OR

What is graph? How they are different from trees? Describe in brief the various ods used to represent graphs in memory. **(PTU, Dec. 2012)**

Ans. A graph is an abstract data structure that is meant to implement the graph and graph concepts from mathematics.

A graph data structure consists of a finite (and possibly mutable) set of ordered pairs, edges or arcs, of certain entities called nodes or vertices. As in mathematics, an edge

(x, y) is said to point or go from x to y. The nodes may be part of the graph structure, or may be external entities represented by integer indices or references.

A graph data structure may also associate to each edge some edge value, such as a symbolic label or a numeric attribute (cost, capacity, length, etc.).

Difference : A tree is just a restricted form a graph. Tree have direction (parent/child relationship) and donot contain cycles. They fit within the category of Directed Acyclic graphs (or a DAG). So trees are DAGs with the restriction that a child can only have one parent. Graphs are generally search breadth first or depth first. The same applies to tree graphs are very useful and can be used to model an enormous amount of things.

Representations : Different data structures for the representation of graphs are used in practice :

- **Adjacency list :** Vertices are stored as records or objects, and every vertex stores a list of adjacent vertices. This data structure allows to store additional data on the vertices.
- **Incidence list :** Vertices and edges are stored as records or objects. Each vertex stores its incident edges, and each edge stores its incident vertices. This data structure allows to store additional data on vertices and edges.
- **Adjacency matrix :** A two-dimensional matrix, in which the rows represent source vertices and columns represent destination vertices. Data on edges and vertices must be stored externally. Only the cost for one edge can be stored between each pair of vertices.
- **Incidence matrix :** A two-dimensional Boolean matrix, in which the rows represent the vertices and columns represent the edges. The entries indicate whether the vertex at a row is incident to the edge at a column.

Q 26. What do you mean by path matrix? **(PTU, Dec. 2011)**

Ans. Let G be a simple directed graph with m nodes, $v_1, v_2, \ldots v_m$. The path matrix or reachability matrix of G is the m-square matrix defined as follows :

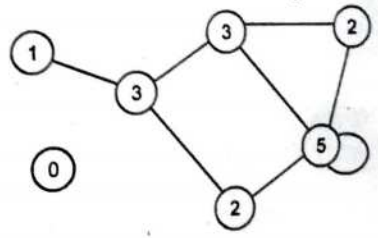
$$P_{ij} = \begin{cases} 1 & \text{there is path from } v_i \text{ to } v_j \\ 0 & \text{otherwise} \end{cases}$$

Suppose there is path from v_i to v_j. Then there must be simple path from v_i to v_j when $v_i! = v_j$. Since, G has m nodes, such a simple path must have length m-1 or less, or such a cycle must have length m or less. This means there is no zero ij entry in the matrix Bm, defined at the end of preceding subsection.

Q 27. What is a degree of a graph? **(PTU, May 2013)**

Ans. In graph theory, the degree of a vertex of a graph is the number of edges incident to the vertex, with loops counted twice.

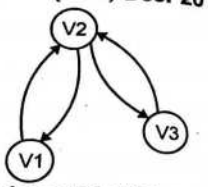
Maximum degree is 5.



Q 28. What is meant by strongly connected in a graph ? **(PTU, Dec. 2014)**

Ans. An undirected graph is connected, if there is a path from everyvertex to every other vertex. A directed graph with this property is called strongly connected.

A directed graph is said to be strongly connected if every pair of distinct vertices Vi, Vj, are connected. Thus if there exists a directed path from vi to vj then there also exists a directed path from vj and vi.



A strongly connected graph

Q 29. Discuss Depth First Search traversing techniques for graphs with the help of suitable example. Write program for the same. **(PTU, Dec. 2016)**

Ans. Depth First Search traversing technique : *Refer to Q.No. 14 & 24(a)*

```
int a[20][20], reach[20],n;
void dfs(int v)
{
int i;
reach[v] = 1;
for (i  = i; i < = n; i++)
if (a[v][i] && !reach[i])
{
printf("n% d → %d" , v , i);
dfs(i);
}
}
void main( )
{
int i, j, Count = 0;
clrscr( );
printf("n Enter number of vertices:");
scanf ("%d",&n);
for (i = 1, i<=n, i++)
{
reach[i] = 0;
for (j = 1; j < =n; j++)
a [i][j] = 0;
} .
printf("n Enter the adjacency matrix : n");
for (i = 1; i < = n; i++)
for (j = 1; j < = n j ++ )
scanf("%d", &a[i][j]);
dfs(1);
printf("n");
for(i = 1; i < = n ; i++)

(reach [i])
```



```
Count++;
}
if(count == n)
printf("n Graph is connected");
else
printf("n Graph is not connected");
getch( );
}
```

Q 30. Write an Algorithm to traverse a graph using Depth First search. **(PTU, May 2018)**

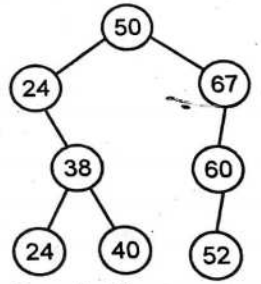
Ans. This algorithm executes a depth first search on a graph G beginning at a starting node A.

1. Initialize all nodes to the ready state (STATUS = 1)
2. Push the starting node A onto STACK and change its status to the waiting state (STATUS = 2)
3. Repeat steps 4 and 5 until STACK is empty.
4. Pop the top node N of STACK. Process N and change its status to the processed state (STATUS = 3)
5. Push onto STACK all the neighbours of N that are still in the ready state (STATUS = 1) and change their status to the waiting state (STATUS = 2) [End of step 3 loop]
6. Exit.

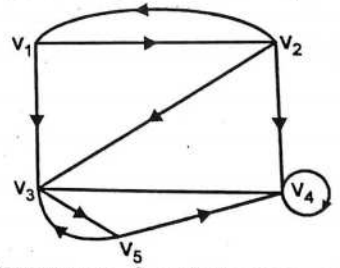
Q 31. Make a binary search tree by considering the following eight numbers. **(PTU, May 201**

50, 24, 38, 24, 67, 40, 60, 52.

Ans.



Q 32. Consider the directed Graph G.



(i) Find indegree and outdegree of each node.
(ii) Find number of simple paths.
(iii) Is there any source or sink ? **(PTU, Ma**

Ans. (i) Find indegree and outdegree of each node.

Indegree

$V_1 = 1$

$V_2 = 1$
$V_3 = 3$
$V_4 = 2$
$V_5 = 1$

Outdegree

$V_1 = 2$
$V_2 = 3$
$V_3 = 1$
$V_4 = 0$
$V_5 = 2$

(ii) Find number of simple paths

Simple paths are :

$V_1 \rightarrow V_5 \rightarrow V_4$
$V_2 \rightarrow V_4$
$V_2 \rightarrow V_3$
$V_3 \rightarrow V_5 \rightarrow V_4$
$V_5 \rightarrow V_4$

(iii) Is there any source or sink

V_4 in sink

Q 33. What is undirected graph ?

Ans. An undirected graph is graph i.e., a set of objects that are connected together, where all the edges are bidirectional. An undirected graph is sometimes called an undirected graph is sometimes called an undirected network. In contrast a graph where the edges point in a direction is called a directed graph. **(PTU, May 2019)**

Q 34. Write the procedure to implement the adjacent matrix. **(PTU, May 2019)**

Ans. The following is the procedure to implement the adjacent matrix.

```
/* get a list of all the nodes in our graph*/
$ array_nodes = $nonDirected Graph –> get Nodes ( );
/* This is where will save the adj. matrix */
$adj_matrix = array ( );
/* Reset the matrix to all '0' s */
for each ($nodes_names as $row){
for each ($nodes_names as $col){
$adj_matrix [$row] [$col] = 0;
}
}
/* Now build the adj.matrix */
for each ($array_nodes as $nd){
$row = $nd –> getData ( );
$neighbours = $nd –> getNeighbours ( );
for each ($neighbours as $neighbour)
{
$col = $neighbour –> getdata ( ) ;
$adj_matrix [$row] [$col] = 1;
}}
```

❑❑❑

LORDS MODEL TEST PAPERS

(Unsolved)

LORDS MODEL TEST PAPER – 1

Instructions to candidates :

1. Section A is compulsory.
2. Attempt any four questions from section B.
3. Attempt any two questions from section C.

SECTION – A

Q 1. (a) Define algorithm.
(b) Distinguish between stack and queue.
(c) What are the various operations on stack ?
(d) What are priority queues ? How are they implemented ?
(e) What is double link list ?
(f) What is the need for garbage collection ?
(g) How are trees represented in memory using arrays?
(h) What is a heap ? How heaps are implemented?
(i) What is an asymptotic rotation ? Mention its types.
(j) What is an AVL tree.

SECTION – B

Q 2. (a) What is the complexity of an algorithm ? Also explain time space trade off.
(b) Write an algorithm for Binary search.

Q 3. Consider the following in fix expression :
$(A + B \times D) \uparrow (E - F)$
Write the expression and convert into the equivalent post fix expression.

Q 4. Write an algorithm to insert new node at the end of a Double linked list.

Q 5. What is quick sort ? Sort the following array using quick sort method
24, 56, 47, 35, 10, 90, 82, 31.

Q 6. What is adjacency matrix representation of a graph in memory ?

SECTION – C

Q 7. What is Data Structures? What are different data structures operations.

Q 8. What are the various operations possible on a singly link list ? Explain with the diagrams.

Q 9. Sort the following list of elements using Bubble Sort :
98, 89, 44, 7, 5, 35, 12, 100, 2, 57
What is its complexity ?

LORDS MODEL TEST PAPER – 2

Instructions to candidates :

1. Section A is compulsory.
2. Attempt any four questions from section B.
3. Attempt any two questions from section C.

SECTION – A

Q 1. (a) Why Complexity of linear search is of the order of O (n) ?
(b) What is under flow?
(c) What is traversing? Write an algorithm for traversing a link list?
(d) What is a spanning tree ?
(e) Define hash function.
(f) Distinguish between BFS and DFS.
(g) What is time space trade off ?
(h) What is a top pointer of stack.
(i) What is degree of a graph ?
(j) Construct the binary tree for the following expression.
$(2x - 3z + 5)\ (3x - y + 8)$

SECTION – B

Q 2. Write a program for implementing stack using arrays.
Q 3. Write suitable routines to perform insertion and deletion operations in a linked list.
Q 4. Write short notes on :
(a) B–Trees
(b) AVL search Trees
(c) M–Way search Trees
Q 5. Sort the following list of numbers.
52, 1, 27, 85, 66, 23, 13, 57
Using any efficient sorting algorithm.
Q 6. Explain the various collision resolution techniques used for hashing with example.

SECTION – C

Q 7. Explain various types of queues with examples and write an algorithm to impleme circular queue.
Q 8. What are the various operations possible on a singly link list ? Explain with the diagram
Q 9. Write an algorithm for preorder, inorder and postorder traversal in a tree.

❑❑❑