Unit 1: Introduction to Data Structures

Unit at a glance:

1.1 Data Structure

Data may be organized in many different ways. The logical or mathematical model of a particular organization of data is called data structure.

Linear data structure: A linear data structure traverses the data elements sequentially, in which only one data element can directly be reached. Ex: Arrays, Linked Lists

Non-Linear data structure: Every data item is attached to several other data items in a way that is specific for reflecting relationships. The data items are not arranged in a sequential structure. Ex: Trees, Graphs

1.2 Classification of Data Type

Data type can be classified as ***primitive data type*** and ***abstract data type.***

- **Primitive data type:** In the phrase *primitive data type* the word **primitive** means "a fundamental component that is used to create other, larger parts." In terms of the programming language, **primitive data type** is either of the following:
- **Abstract data type** (ADT): *Abstract Data Types* are a set of data values and associated operations that are precisely independent of any particular implementation. The term abstract signifies that the data type will only set the rule of it's usage but how it will be used depends on the implementation. For example stacks and queues are called abstract data types. The stack data type defines two abstract methods PUSH and POP.
- **Atomic data types:** An atomic data type can contain content whose values are not composed of values of other data types.

1.3 Algorithm

An algorithm is a finite sequential set of instructions which, if followed, accomplish a particular task or a set of tasks in finite time. Algorithms are used for calculation, data processing, and automated reasoning.

1.4 Complexity

There are two types of complexities of an algorithm, time complexity and space complexity.

- The ***time complexity*** of an algorithm is the amount of time the computer requires to execute the algorithm.
- The ***space complexity*** of an algorithm is the amount of memory space the computer requires, completing the execution of the algorithm.

1.5 Big OH Notation

The Big Oh (the "O" stands for "order of") notation is used to classify functions by their asymptotic growth function and hence finding the time complexity of an algorithm

1.6 Fixed-Point and Floating-Point Representations

Fixed-Point Method

This assumes that the binary point is always fixed in one specific point in the register i.e. either the binary point lies in the ***extreme left*** of the register thus making the stored number a ***fraction***, or it lies in the *extreme* ***right*** of the register making the stored number an ***integer.***

Though in either of the above cases, the binary point is not present physically but its presence is assumed to treat a stored number as fraction or as integer.

Floating-Point Method

Such a representation uses a second register to store the number that designates the position of the decimal point stored in the first register.

IEEE Representation of Floating-Point Numbers

All modern computers use the floating-point representation that was specified in IEEE standard 754.

Short Answer Type Questions

A. Choose the correct answer from the given alternatives in each of the following:

1. If $f(n)=5n^3+4n^2-8n+100$, then $f(n)=$ _____ [WBSCTE 2017]

   (a) O(1) (b) O(n) (c) $O(n^2)$ (d) $O(n^3)$

Answer: (d)

2. Which data structure has fixed size? [WBSCTE 2018]

   (a) Arrays (b) Linked lists (c) Trees (d) Graphs

Answer: (a)

B. Fill in the blanks in the following statements:

3. ***Integer*** is a primitive data type. [WBSCTE 2007]

4. In 2s complement form, the highest integer that can be stores with n bits is 2^{n-1} [WBSCTE 2007]

5. A data structure suitable for data remaining in secondary storage devices is ***structured***. [WBSCTE 2007]

6. A programming technique that never uses a GOTO statement is called ***structured*** programming. [WBSCTE 2007]

7. Complexity of the order O (n^2) is ***greater*** than O (log n). [WBSCTE 2009]

8. The clrscr() function is kept in conio.h header file. [WBSCTE 2022]

9. Breaking a program into several functions is called ***modular programming***. [WBSCTE 2022]

10. The #define pi 3.14 is a ***macro*** statement. [WBSCTE 2022]

11. The pointer without a data type is ***void***. [WBSCTE 2022]

12. A loop inside another loop is called ***nested***. [WBSCTE 2022]

13. ***Structure*** stores the non-homogeneous data elements. [WBSCTE 2022]

C. State whether the following statements are True or False:

14. A data structure is an actual implementation of a particular abstract data type. [WBSCTE 2009]

Answer: True

15. Dynamic memory allocation Obtain and release memory during execution. [WBSCTE 2009]

Answer: True

16. Allocate the memory location at runtime is known as static memory allocation. [WBSCTE 2013]

Answer: False

17. Time complexity of Insertion sort O ($\log_2 n$). [WBSCTE 2013]

Answer: False

D. Answer the following questions:

18. Differentiate between Storage structures and Data structures. [WBSCTE 2011]

Answer:

The representation of a particular data structure in the memory of a computer is called a storage structure whereas a storage structure representation in auxiliary memory is often called a data structure.

19. What is data structure? [WBSCTE 2012, 2014, 2015, 2018, 2019,2022]

Answer:
A data structure is a mathematical or logical way of organizing data in the memory that consider not only the items stored but also the relationship to each other and also it is characterized by accessing functions.

20. How linear data structure differs from non-linear data structure? [WBSCTE 2012, 2022]

Answer:
Linear data structure is linear if element is adjacent to each other. It has exactly two neighbors elements to which it is connected as its previous and next member. 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.

21. What are the methods available in storing sequential files? [WBSCTE 2012, 2014, 2019]

Answer:
Straight merging,
Natural merging,
Polyphase sort,
Distribution of Initial runs.

22. What do you mean by garbage collection? [WBSCTE 2013]

Answer:
Garbage collection is the process by which unused memory is reclaimed by the system so that it can be referenced later on. In Java this process of garbage collection is automatic, which involves marking, then deleting unreferenced objects. Automatic garbage collection avoids memory leaks. However, garbage collector can be called explicitly by a programmer also.

23. Prove the following" $n^a=O(n^b)$ if and only if a<=b. [WBSCTE 2013]

Answer:
By definition $n^a= O(n^b)$ if there exists a $C > 0$ and an $N \in N$ such that $n^a \leq C\, n^b$ for all $n \geq N$. We first show that if $a \leq b$ then $n^a= O(n^b)$. Let $C := 1$ and $N := 1$. If $a \leq b$ then $n^a \leq Cn^b$ for all $n \geq N$ and thus by above definition $n^a= O(n^b)$.
Next, we show that if $n^a= O(n^b)$ then $a \leq b$. Let $n^a= O(n^b)$. By above definition there exists a $C > 0$ and an $N \in N$ such that $n^a \leq Cn^b$ for all $n \geq N$. Then, $n^{a-b} \leq C$ for all $n \geq N$. For contradiction assume $a > b$. Then the sequence n^{a-b} diverges. This is a contradiction to $n^{a-b} \leq C$, thus $a \leq b$.

24. Discuss the significance of Big-Oh notation. [WBSCTE 2013, 2016]

Answer:
The Big Oh (the "O" stands for "order of") notation is used to classify functions by their asymptotic growth function and hence finding the time complexity of an algorithm.
The different computing functions are measured as:

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

Let f(n) and g(n) are two nonnegative functions. The function f(n) = O(g(n)) (read as f of n equals big-oh of g of n) iff there exist two constants c & n_0 such that

$$|f(n)| \leq c * |g(n)| \text{ for all } n \geq n_0.$$

f(n) will normally represent the computing time of some algorithm. When we say that the computing time of some algorithm is O(g(n)) we mean that its execution takes no more than a constant time g(n), where n is a parameter which characterizes input/output. For example n might be the number of inputs or number of outputs or their sum or the magnitude of one of them.

25. Differentiate data type and data structure? [WBSCTE 2013]

Answer:

Data type:

1. A **data type** in a programming language is a set of data with values having predefined characteristics.
2. Data type can be classified as ***primitive data type*** and ***abstract data type.***
3. Primitive data type: int, float, char, pointer are the primitive data types of C.
4. Abstract data type: For example stacks and queues are called abstract data types.

Data Structure:

1. Data may be organized in many different ways. The logical or mathematical model of a particular organization of data is called data structure.
2. Data structure can be classified as ***linear*** and ***non-linear.***
3. Linear data structure: Array, linked list.
4. Non-Linear data structure: tree and graph.

26. List out the areas in which data structures are applied extensively? [WBSCTE 2014, 2019]

OR,

Write some applications of data structure. [WBSCTE 2018]

Answer:

1. Compiler Design,
2. Operating System,
3. Database Management System,
4. Statistical analysis package,
5. Numerical Analysis,
6. Graphics,
7. Artificial Intelligence,
8. Simulation

27. What is a primitive data type? [WBSCTE 2015]

Answer: ***Refer to 1.2 of Unit at a Glance.***

28. What do you mean by abstract data type? [WBSCTE2016]

Answer: ***Refer to 1.2 of Unit at a Glance.***

29. Define an algorithm? [WBSCTE2016]

Answer: *Refer to 1.3 of Unit at a Glance.*

30. What is big 'O' notation? [WBSCTE2016]

Answer: *Refer to 1.5 and Question No. 18 of Short Answer Type Questions.*

31. How recursion is different from iteration? [WBSCTE2016]

Answer:

Basis for Comparison	Recursion	Iteration
Basic	The statement in a body of function calls the function itself.	Allows the set of instructions to be repeatedly executed.
Format	In recursive function, only termination condition (base case) is specified.	Iteration includes initialization, condition, execution of statement within loop and update (increments and decrements) the control variable.
Termination	A conditional statement is included in the body of the function to force the function to return without recursion call being executed.	The iteration statement is repeatedly executed until a certain condition is reached.
Condition	If the function does not converge to some condition called (base case), it leads to infinite recursion.	If the control condition in the iteration statement never become false, it leads to infinite iteration.
Infinite Repetition	Infinite recursion can crash the system.	Infinite loop uses CPU cycles repeatedly.
Applied	Recursion is always applied to functions.	Iteration is applied to iteration statements or "loops".
Stack	The stack is used to store the set of new local variables and parameters each time the function is called.	Does not uses stack.
Overhead	Recursion possesses the overhead of repeated function calls.	No overhead of repeated function call.
Speed	Slow in execution.	Fast in execution.
Size of Code	Recursion reduces the size of the code.	Iteration makes the code longer.

32. Define null pointer. [WBSCTE 2016]

Answer:

In computer programming, a null pointer is a pointer that does not point to any object or function.

33. Explain the features of a good program. [WBSCTE 2018]

Answer:

A good computer program should have following characteristics:

- **Portability:** Portability refers to the ability of an application to run on different platforms (operating systems) with or without minimal changes. Due to rapid development in the hardware and the software, nowadays platform change is a

common phenomenon. Hence, if a program is developed for a particular platform, then the life span of the program is severely affected.

- **Readability:** The program should be written in such a way that it makes other programmers or users to follow the logic of the program without much effort. If a program is written structurally, it helps the programmers to understand their own program in a better way. Even if some computational efficiency needs to be sacrificed for better readability, it is advisable to use a more user-friendly approach, unless the processing of an application is of almost importance.
- **Efficiency:** Every program requires certain processing time and memory to process the instructions and data. As the processing power and memory are the most precious resources of a computer, a program should be laid out in such a manner that it utilizes the least amount of memory and processing time.
- **Structural:** To develop a program, the task must be broken down into a number of subtasks. These subtasks are developed independently, and each subtask is able to perform the assigned job without the help of any other subtask. If a program is developed structurally, it becomes more readable, and the testing and documentation process also gets easier.
- **Flexibility:** A program should be flexible enough to handle most of the changes without having to rewrite the entire program. Most of the programs are developed for a certain period and they require modifications from time to time. For example, in case of payroll management, as the time progresses, some employees may leave the company while some others may join. Hence, the payroll application should be flexible enough to incorporate all the changes without having to reconstruct the entire application.
- **Generality:** Apart from flexibility, the program should also be general. Generality means that if a program is developed for a particular task, then it should also be used for all similar tasks of the same domain. For example, if a program is developed for a particular organization, then it should suit all the other similar organizations.
- **Documentation:** Documentation is one of the most important components of an application development. Even if a program is developed following the best programming practices, it will be rendered useless if the end user is not able to fully utilize the functionality of the application. A well-documented application is also useful for other programmers because even in the absence of the author, they can understand it.

34. What is Backtracking?

Answer: **[WBSCTE 2022]**

Backtracking is an algorithmic technique whose goal is to use brute force to find all solutions to a problem.

35. What will be the value of A (1,5) by using Ackerman function. **[WBSCTE 2022]**

Answer:
In computability theory, the Ackermann function, named after **Wilhelm Ackermann**, is one of the simplest and earliest-discovered examples of a total computable function that is not primitive recursive. All primitive recursive functions are total and computable, but the Ackermann function illustrates that not all total computable functions are primitive recursive.
It's a function with two arguments each of which can be assigned any non-negative integer.
Ackermann function is defined as:

$$A(m,n)=\begin{cases} n+1 & \text{if } m=0 \\ A(m-1,1) & \text{if } m>0 \text{ and } n=0 \\ A(m-1,A(m,n-1)) & \text{if } m>0 \text{ and } n>0 \end{cases}$$

where m and n are non-negative integers

So value of **A(1,5) = 7.**

Long Answer Type Questions

➲ **1. What is algorithm?** **[WBSCTE 2004, 2006, 2009, 2016, 2017]**
What do you mean by complexity of an algorithm? Show that the function $f(x)$ defined by: $f(1)=1$, $f(n)=f(n-1)+1$ has the complexity $0(n)$ **[WBSCTE 2004, 2006, 2009]**

Answer:
1st part:
Algorithm: An algorithm is a finite sequential set of instructions which, if followed, accomplish a particular task or a set of tasks in finite time, is an effective method for solving a problem expressed as a finite sequence of steps.

2nd part:
There are two types of complexities of an algorithm, time complexity and space complexity.

Time Complexity: The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the size of the input to the problem. This commonly expressed using big O notation, which suppresses multiplicative constants and lower order terms. When expressed this way, the time complexity is said to be described *asymptotically*
The ***space complexity*** of an algorithm is the amount of memory space the computer requires, completing the execution of the algorithm.

3rd part:
Applying the recursion up to n-2 times that is:
f(n) = 1+1+1+1+1+1+.....n-2 times + f(1) +1
= (n-1) + f(1)
=(n-1) +1 where f(1) = 1
F(n) = n
Or, | f(n) | <= 2*| g(n)| where c = 2 & g(n) = n
Or, f(n) = O(g(n))
Or, f(n) = O(n)

➲ **2. Explain the terms: Best, Average and worst case complexities of an algorithm.** **[WBSCTE 2009]**

Answer:
There are different types of time complexities which can be analysed for an algorithm:

- Best-case time complexity, Average-case time complexity AND Worst-case time complexity.

The best-case time complexity of an algorithm is a measure of the minimum time that the algorithm will require for an input of size *n*. The running time of many algorithms varies not only for the inputs of different sizes but also for the different inputs of same size. The average-case time complexity of an algorithm is a measure of the average time that the algorithm will require for an input of size *n*.

The worst-case time complexity of an algorithm is a measure of the maximum time that the algorithm will require for an input of size *n*. Therefore, if various algorithms for sorting are taken into account and say, *n* input data items are supplied in reverse order for any sorting algorithm, then the algorithm will require n^2 operations to perform the sort which will correspond to the worst-case time complexity of the algorithm.

The work done by an algorithm for the execution of the input size n defines the time analysis as function $f(n)$ of the input data items.

➲ **3. Explain Abstract Data Types with any example.** **[WBSCTE 2009, 2022]**

Answer:
Abstract Data type: In computing, an **abstract data type** is a mathematical model for a certain class of data structures that have similar behavior; or for certain data types of one or more programming languages that have similar semantics. An abstract data type 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.

Abstract data types are purely theoretical entities, used (among other things) to simplify the description of abstract algorithms, to classify and evaluate data structures, and to formally describe the type systems of programming languages.

Example: Struct, class in C, C++, and using record in Pascal, programmer can define their own data types.

➲ **4. Differentiate between linear and non-linear data structure.** **[WBSCTE 2009]**

Answer:

1) Linear data structure: The data structure is said to be linear, if its element forms a sequence or linear list. There are two basic ways of representing such structure in memory. Here the elements are traversed sequentially starting from the beginning.

(a) Linear relationship between data elements is represented by means of sequential memory locations. ***Ex: Arrays.***

(b) Linear relationship between data elements is represented by means of pointers and links. ***Ex: Linked list.***

2) Non-linear data structure: They are used to represent the data having a hierarchical relationship between elements. Here the elements are not traversed sequentially, rather they are traversed in a non-linear fashion. For example, in case of trees, we have to start from the root but we have to traverse either left subtree or right subtree, but not the both.

Ex: - Trees, Graphs.

➲ **5. The Ackerman function, for all non-negative of m and n recursively as defined as**

$$A(m,n)=\begin{cases} n+1 & if\ m=0 \\ A(m-1,1) & \text{if } m!=0 \text{ but } n=0 \\ A(M-1,A(m,n-1)) & if\ m!=0 \text{ and } n!=0 \end{cases}$$

Find value of A (1, 3). **[WBSCTE 2012]**

Answer:

Value of A (1, 3)

A(1, 3) = A(0,A(1,2)) = A(0, A(0,A(1,1)))

= A(0, A(0, A(0,A(1,0)))) = A(0, A(0, A(0, A(0,1))))

= A(0, A(0, A(0, 2))) = A(0, A(0, 3)) = A(0, 4) =5

➲ **6. The Ackerman function, for all non-negative of m and n recursively as defined as**

$$A(m,n)=\begin{cases} n+1 & \text{if } m=0 \\ A(m-1,1) & \text{if } m!=0 \text{ but } n=0 \\ A(m-1,A(m,n-1)) & \text{if } m!=0 \text{ and } n!=0 \end{cases}$$

Therefore what will be the value of $A(1, 5)$? **[WBSCTE 2013]**

Answer:

$$A(1,5)=A(0,A(1,4))$$

$= A(0, A(0, A(1,3))) = A(0, A(0, A(0, A(1,2))))$

$= A(0, A(0, A(0, A(0, A(1,1))))) = A(0, A(0, A(0, A(0, A(0, A(0,1))))))$

$= A(0, A(0, A(0, A(0, A(0,2))))) = A(0, A(0, A(0, A(0,3))))$

$= A(0, A(0, A)(0A(0,4))) = A(0, A(0,5))$

$= A(0,6) = 7$

➲ **7. What is time Complexity? Explain Big 'O' Notation.** [WBSCTE 2014, 2015]

Answer:

1st part: ***Refer to 1.4 of Unit at a Glance.***

2nd part: ***Refer to Question No. 18 of Short Answer Type Questions.***

➲ **8. (a) What do you mean by Complexity of Algorithm?**
(b) What do you mean by primitive data type?
(c) Distinguish between data type and data structure. [WBSCTE 2016]

Answer:

a) ***Refer to Question No. 1 of Long Answer Type Question.***
b) ***Refer to 1.2 of Unit at a Glance.***
c) ***Refer to Question No. 19 Short Answer Type Questions.***

➲ **9. The Ackerman function, for all non-negative of m and n is recursively as defined as**

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,\ 1) & \text{if } m! = 0 \text{ but } n = 0 \\ A(m-1,\ A(m,n-1)) & \text{if } m! = 0 \text{ and } n! = 0 \end{cases}$$

Therefore,
What will be the value of A(1, 4)?
Answer: [WBSCTE 2019]

$A(1,4) = A(0, A(1,3))$

$= A(0, A(0, A(1,2)))$

$= A(0, A(0, A(0, A(1,1))))$

$= A(0, A(0, A(0, A(0,1))))$

$= A(0, A(0, A(0, A(0,2))))$

$= A(0, A(0, A(0,3)))$

$= A(0, A(0,4))$

$= A(0,5) = 6$

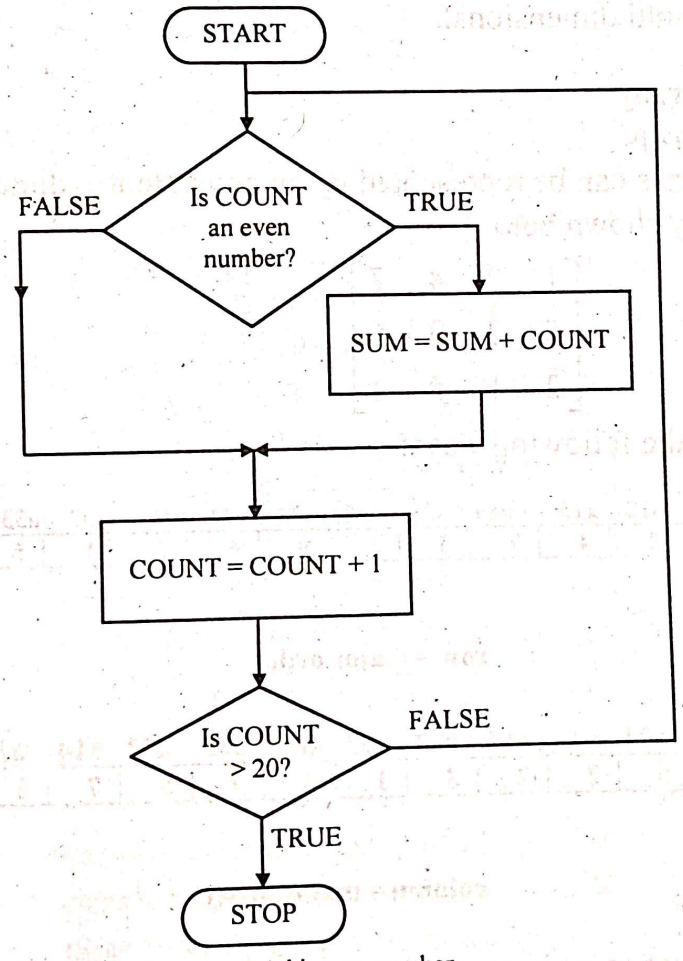
The value of $A(1,4)$ will be 6:

➲ **10. By which features a program is not an algorithm? How can you describe an algorithm with the help of a flowchart?** **[Model Question]**

Answer:

Actually, Algorithm + Data Structure = Program

The algorithm of addition of even numbers are shown below using a flow-chart.



Add even number

Unit 2: Linear Data Structure

Unit at a glance:

2.1 Array

An array is a finite set of ordered, homogeneous elements. By homogeneity it means that an array can hold more than one values of the same data type. It is a linear data structure, in which consecutive memory locations are used to store the data. Typically the elements stored in an array are accessed by the index of the array. Array can be one dimensional, two dimensional or multi dimensional.

Representation of Array

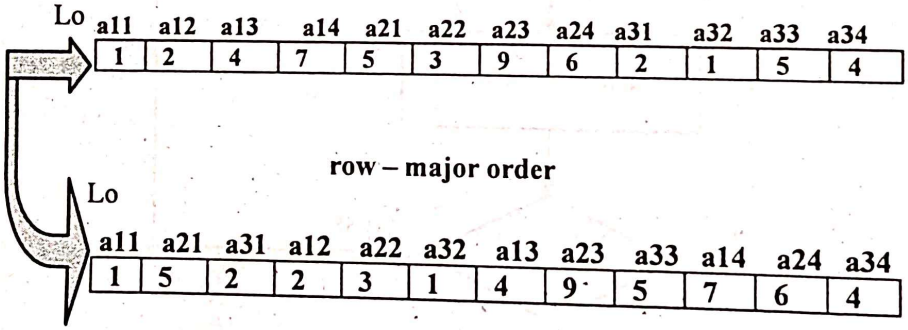
Two Dimensional Array

All 2-dimensional arrays can be represented by an equivalent 1-dimensional array.

For example, the array shown belo

$$\begin{bmatrix} 1 & 2 & 4 & 7 \\ 5 & 3 & 9 & 6 \\ 2 & 1 & 5 & 4 \end{bmatrix}$$

can be represented in the following ways:



A 2D array can be represented in two different ways: ***row-major order*** (where elements are stored row-wise in 1D form) and ***column major order*** (where elements are stored column wise in 1D form).

2.2 Representation of Polynomials Using Array

In general any polynomial A(x) can be written as

$$A(x) = a_0 + a_1x + a_2x^2 + \text{-----} + a_{n-1}x^{n-1} + a_nx^n.$$

Each $a_i x^i$ is i^{th} term of the polynomial, where x is variable, a, is its coefficient & e is the exponent.

If $a_i = 0$, then the term is zero term, otherwise it is nonzero term.

2.3 Sparse Matrix

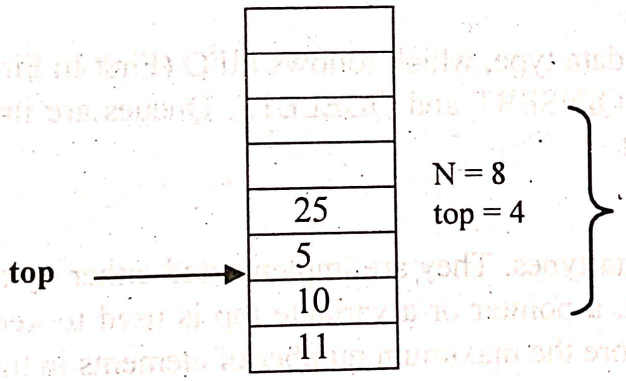
A matrix is represented in memory as 2D array, like A [i, j], where i is a row and j is a column. If there are m rows and n columns, then total elements in a matrix or size of a matrix is m x n elements. This m x n elements requires m x n units of storage.
A matrix, which is filled with mostly zeroes (approx more than 2/3 elements are zero), is called as a ***sparse matrix.***

2.4 Stack

A stack is an abstract data type which follows LIFO (Last In First Out) data structure. It declares two methods PUSH and POP. Stacks are implemented either by an array or by a linked list.

2.5 Array Representation

Stacks are abstract data types. They are implemented either by an array or a linked list. In array representation, a pointer or a variable top is used to keep track of the stack. A variable N is used to store the maximum number of elements in the stack.



2.6 Various Types of Expressions

A mathematical expression involves constants (operands) and operations (operators).

- Infix notation: **operand1 *operator* operand2,** **A + B**
- Prefix notation: ***operator* operand1 operand2,** **+ A B**
- Postfix notation: **operand1 operand2 *operator*, A B +**

Conversion from INFIX to POSTFIX

In order to convert the infix to its corresponding postfix form; we have to follow the following steps:

i) Fully parenthesize the expression according to the priority of different operators.
ii) Move all operators so that they replace their corresponding right parentheses.
iii) Delete all parentheses.

The priorities of different operators are given below:

Operators	Priority
Unary -, unary +, not (!)	4
*, /, %, and (& / &&)	3
+, -, or (| / ||)	2
<, <=, >, >=, ==, !=	1

Let us consider the following expression:

4 / 2 + 3 * 7 – 12 % 2

As per the priority table, after parenthesizing the expression we get

(((4 / 2) + (3 * 7)) - (12 % 2))

After moving the operators corresponding to the parentheses we get

(((4 2 /) + (3 7 *)) - (12 2 %))

= 4 2 / 3 7 * + 12 2 % -

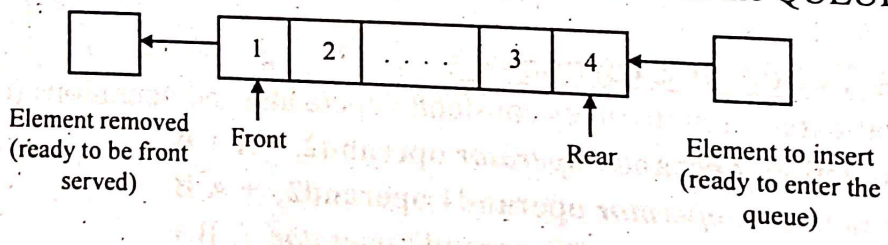
which is the required postfix form.

2.7 Queues

A queue is an abstract data type, which follows FIFO (First In First Out) data structure. It declares two methods QINSERT and QDELETE. Queues are implemented either by an array or by a linked list.

Array Representation

Queues are abstract data types. They are implemented either by an array or a linked list. In array representation, a pointer or a variable top is used to keep track of the stack. A variable N is used to store the maximum number of elements in the QUEUE.



2.8 Circular Queues, Dequeue, Priority Queues

In a queue if all the elements are arranged CQ [1] ,CQ [2] , ... CQ [N] in a circular fashion with CQ [1] following CQ [N] then ,these type of queues are called ***circular queues.***

A ***priority queue*** is a data structure used for storing a set S of elements, based on a key value, which denotes the priority of that element. The priority determines the order in which they exit the queue. The element, having the highest priority, will be removed first.

Dequeue means double-ended queue. In these types of queues insertion and deletion can be done from both ends.

Short Answer Type Questions

A. Choose the correct answer from the given alternatives in each of the following:

1. If TOP=MAX–1, then the stack is [WBSCTE 2017]
   (a) Empty (b) Full
   (c) Contains some data (d) None of these
   Answer: (b)

2. Queue is a [WBSCTE 2017, 2021]
   (a) Linear data structure (b) Non-linear data structure
   (c) Both (a) and (b) (d) none of these
   Answer: (a)

3. Queue is [WBSCTE 2017, 2021]
   (a) LIFO (b) FIFO (c) FILO (d) LILO
   Answer: (b)

4. Disks piled up one above the other represents a [WBSCTE 2017, 2021]
   (a) Stack (b) Queue (c) Linked list (d) Array
   Answer: (a)

5. Stack is: [WBSCTE 2018]
   (a) LIFO (b) FIFO (c) FILO (d) LILO
   Answer: (a)

6. The end at which an element gets removed from the queue is called [WBSCTE 2018]
   (a) front (b) rear (c) top (d) bottom
   Answer: (a)

7. Which function places an element on the stack? [WBSCTE 2018]
   (a) Pop() (b) Push() (c) Peek() (d) Peep()
   Answer: (a)

8. Which data structure is defined as a collection of similar data elements? [WBSCTE 2022]
   (a) Arrays (b) Structs (c) Trees (d) Graphs
   Answer: (a)

9. Which function places an element on the stack? [WBSCTE 2022]
(a) Pop (b) Push (c) Peek (d) isEMpty
Answer: (b)

10. How do you initialize an array in C? [WBSCTE 2022]
(a) int arr[3] = (1, 2, 3) (b) int arr(3) = {1, 2, 3};
(c) int arr[3] = {1, 2, 3} (d) int arr(3) = (1, 2, 3)
Answer: (c)

11. Reverse polish notation is the other name of [WBSCTE 2022]
(a) Infix expression (b) Prefix Expression
(c) Postfix expression (d) None of these
Answer: (c)

12. If TOP=MAX –1, then the stack is [WBSCTE 2021, 2022]
(a) full (b) empty (c) contains some data (d) none of these
Answer: (a)

13. Pushing an element into a stack already having five elements and a stack size of 5, then the stack becomes _____. [WBSCTE 2022]
(a) Overflow (b) Crash (c) Underflow (d) User flow
Answer: (a)

B. Fill in the blanks in the following statements:

14. ***Queue*** is a FIFO type data structure. [WBSCTE 2007, 2008]

15. Dequeue process removes data from the ***either end*** of a queue. [WBSCTE 2008]

16. Insertion in a queue is done in the ***rear*** of a queue. [WBSCTE 2008]

17. Most of the entries in a sparse matrix are ***null***. [WBSCTE 2008]

18. Array cells are located ***linearly*** in the memory. [WBSCTE 2009]

19. We can access an element of the stack only from the ***top*** of the stack. [WBSCTE 2009]

20. Sparse matrix consists of more ***majority (null)*** elements than actual data. [WBSCTE 2009]

21. Queue is a ***Linear*** data structure. [WBSCTE 2010]

22. Removes item from stack is possible from the ***pop*** of a stack. [WBSCTE 2010]

23. Process of removing an element from stack is called ***POP***. [WBSCTE 2022]

24. Circular Queue is also known as ***Ring Buffer***. [WBSCTE 2022]

25. In the stack, if a user tries to remove an element from the empty stack , then it called ***underflow***. [WBSCTE 2022]

26. ***Node*** is the logical container of a data item. [WBSCTE 2022]

27. The malloc function returns ***null*** when the allocation fails. [WBSCTE 2022]

C. State whether the following statements are True or False:

28. Linked list is the suitable data structure to construct tree. [WBSCTE 2010]
Answer: True

29. Sparse matrix contains all zero elements at lower half. [WBSCTE 2010]
Answer: False

30. In a dequeue elements are inserted only at the rear. [WBSCTE 2010]
Answer: False

31. Queue maintains Last-in-Last-out fashion. [WBSCTE 2011]
Answer: True

32. Array is an abstract data type. [WBSCTE 2012]
Answer: True

33. Pointer is a derived data type. [WBSCTE 2012]
Answer: True

34. Array is an abstract data type. [WBSCTE 2013]
Answer: True

35. Size of the character pointer is one byte in C. [WBSCTE 2013]
Answer: True

D. Answer the following questions:

36. **An array A consists 10 keys. If the percentage of requests for the keys in A[1], A[2], ...A[10] be 5, 10, 5, 25, 20, 9, 11, 11, 2, 2 respectively, determine the average number of comparisons for a sequential search.** [WBSCTE 2011]

Answer:

The average number of comparisons for a sequential search

$$= \frac{1\times5+2\times10+3\times5+4\times25+5\times20+6\times9+7\times11+8\times11+9\times2+10\times2}{5+10+5+25+20+9+11+11+2+2} = 4.97$$

37. What is a priority queue? **[WBSCTE 2011, 2012, 2013, 2014, 2015, 2016, 2019]**

OR, **[WBSCTE 2022]**

Explain Priority queue and its types.

Answer:

Priority queue is a queue in which inserting an item or removing an item can be performed from any position based on some priority.

There are two types of priority queues based on the priority of elements.

- If the element with the smallest value has the highest priority, then that priority queue is called the min priority queue.
- If the element with a higher value has the highest priority, then that priority queue is known as the max priority queue.
- Furthermore, you can implement the min priority queue using a min heap, whereas you can implement the max priority queue using a max heap.

38. Explain the terms, row major order and column major order. **[WBSCTE 2011]**

Answer:

In row-major storage, a multidimensional array in linear memory is organized such that rows are stored one after the other.

Column-major order is a similar method of flattening arrays onto linear memory, but the columns are listed in sequence.

39. When does the sparse matrix perform efficiently? **[WBSCTE 2011]**

Answer:

A sparse matrix works more efficiently when all data values (i.e. '1') are is a band.

40. What is de-queue? **[WBSCTE 2012, 2013, 2014, 2019, 2022]**

Answer:

Dequeue is the Double Ended Queue. Which is used to insert and delete operations are performed at the both the ends.

41. How circular queue is better than ordinary queue?

[WBSCTE 2012, 2013, 2014, 2019]

OR,

Explain briefly why we prefer circular queue instead of normal queue with example.

Answer: **[WBSCTE 2015]**

It is better than a normal queue because in this we can effectively utilise 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 then also we cannot insert any new element because the insertion has to be done from one side only (i.e., rear or tail) and deletion has to be done from another side (i.e., front or head).But in case of circular queue the front and rear are adjacent to each other.

42. Differentiate between an array and a stack. [WBSCTE 2012, 2017, 2019]

Answer:

Firstly, an array can be multi-dimensional, while a stack is strictly one-dimensional. Secondly, an array allows direct access to any of its elements, whereas with a stack, only the 'top' element is directly accessible; to access other elements of a stack, you must go through them in order, until you get to the one you want.

43. What is sparse matrix? [WBSCTE 2012, 2015, 2019]

Answer:

A sparse matrix is a matrix that allows special techniques to take advantage of the large number of zero elements.

44. What is pointer? [WBSCTE 2013]

Answer:

A pointer is a variable which contains the address in memory of another variable.

int *a, b=25;

a=&b; Here a is pointer to an integer which holds the address of b.

45. What is the difference between the HEAP and STACK? [WBSCTE 2013]

Answer:

STACK:

1. Variables allocated on the stack are automatic variables
2. Access to this memory is very fast.
3. Used in recursion and has local scope.

HEAP:

1. Variables allocated on the heap, or dynamic variables.
2. Accessing this memory is a bit slower, but the heap size is only limited by the size of virtual memory.
3. Heap is also used for global or shared access of data. Reference to memory can be automatically or explicitly reused later on.

46. What is the minimum number of queues needed to implement the priority queue? [WBSCTE 2014]

Answer:

Two. One queue is used for actual storing of data and another for storing priorities.

47. Which data structure is used to perform recursion? [WBSCTE 2014]

Answer:
Stack. Because of its LIFO (Last In First Out) property it remembers its 'caller' so knows whom to return when the function has to return. Recursion makes use of system stack for storing the return addresses of the function calls.

48. Convert the expression ((A*B)*C/(D–E)^(F+G)) to equivalent Prefix and Postfix notations. [WBSCTE 2014]

Answer:
Prefix Notation: ^ / * *ABC - DE + FG
Postfix Notation: AB *C * DE - /FG + ^

49. Which data structure is used to perform recursion and why? [WBSCTE 2015, 2019]

Answer:
Stack. Because of its LIFO (Last In First Out) property, it remembers its 'caller', so the function knows where to return when the function has to return.

50. Convert the expression ((A*B)*C/(D – E) ∧ (F + G)) to equivalent Prefix and Postfix notations. [WBSCTE 2015]

Answer:
1) **Prefix Notation** : /**ABC ∧-DE+FG
2) **Postfix Notation**: AB*C*DE-FG+∧/

51. In a dequeue where the elements are inserted? [WBSCTE 2015]
Answer: ***Refer to 2.8 of Unit at a Glance.***

52. If a two-dimensional array A[2][3] is stored in row-major form with the A[0][0] element at address 2000 then what will be the address of the A[i][j], assuming each element requires 2 bytes for storage? [WBSCTE 2015]

Answer:
If a two-dimensional array A[2][3] is stored in row-major form with the A[0][0] element at address 2000 then 2000+2[(i-1)3+(j-1)] will be the address of the A[i][j], assuming each element requires 2 bytes for storage.

53. During which operation stack underflow condition is to be checked?

Answer: The stack underflow condition is to be checked during the pop operation. [WBSCTE 2015]

54. The memory address of the first element is called?
Answer: The memory address of the first element is called base address. [WBSCTE 2015]

55. Write down two disadvantages of array to implement stack. [WBSCTE 2016]

Answer:
Two disadvantage of array to implement stack:
In case of array, the size is to be declared at first to implement stack which cause problem if the programmer wants to increase the size of the stack afterwards.
In case of array, due to contiguous memory allocation, proper utilization of the computer memory can never be alone to implement stack.

56. Write down the infix expression (3/2*5)/(3*2+3)+5 into its equivalent postfix form. **[WBSCTE 2016]**
Answer: (3 2/5*)(3 2.3*)5+

57. What are the operations that can be performed on a stack? **[WBSCTE 2017, 2021]**

Answer:
PUSH, POP, PEEK are the operations performed on stack. **Push** to insert element into a stack, **Pop** to delete the last element inserted and **Peek** to display the top element.

58. Write prefix form of the expression: (A+B*C)–(D/E) **[WBSCTE 2017, 2021, 2022]**
Answer: -+A*BC/DE

59. Write postfix form of the expression: $(A+B*C)-(D/E)$. **[WBSCTE 2018]**
Answer:
The postfix form is: operand1 operand2 operator
According to the priority table the expression and be written as: (A+(B*C))-(D/E)
So, the postfix form is: ABC*+DE/-

60. List the applications of a queue. **[WBSCTE 2018]**
Answer:
Queue, as the name suggests is used whenever we need to manage any group of objects in an order in which the first one coming in, also gets out first while the others wait for their turn, like in the following scenarios:

1. Serving requests on a single shared resource, like a printer, CPU task scheduling etc.
2. In real life scenario, Call Center phone systems uses Queues to hold people calling them in an order, until a service representative is free.
3. Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive i.e. First come first served.

61. What are the disadvantages of linear queue? **[WBSCTE 2021]**
Answer:
In a linear queue, the traversal through the queue is possible only once,i.e.,**once an element is deleted, we cannot insert another element in its position.** This disadvantage of a linear queue is overcome by a circular queue, thus saving memory.

62. What is FIFO? [WBSCTE 2022]

Answer:

FIFO stands for **First-in, First-out**. In the FIFO technique, as the name suggests, the data entered first will also exit first. In data structures, the **queue** data structure uses the FIFO technique for maintaining the data.

63. Memory space for an array is allocated in compile-time or in run time? [WBSCTE 2022]

Answer:

The memory is allocated during **compile time**. Dynamic Memory Allocation: Memory allocation done at the time of execution (run time) is known as dynamic memory allocation. Array can be represented in both way.

64. Explain about dummy header? [WBSCTE 2022]

Answer:

When the head of the Linked List doesn't point to any Node, you create a Dummy Head (Node) pointed from that head. So that you would always be able to reach e.g. head. val or head. next without doing any extra null checks.

65. What is the difference between a PUSH and a POP? [WBSCTE 2022]

Answer:

There are two basic operations that can be performed on a stack to modify its contents, which are called PUSH and POP. The main difference between PUSH and POP is what they do with the stack. PUSH is used when you want to add more entries to a stack while POP is used to remove entries from it.

66. Define queue full condition. [WBSCTE 2022]

Answer:

Queue is completely full when rear is at last array position i.e. (MaxSize -1). Now no more elements can be inserted in queue even if the queue has some empty spaces.

67. What is rear of a queue? [WBSCTE 2022]

Answer:

A queue is an ordered collection of items where the addition of new items happens at one end, called the "rear," and the removal of existing items occurs at the other end, commonly called the "front."

68. Write two applications of queue. [WBSCTE 2022]

Answer:

1. Queues are widely used as waiting lists for a single shared resource like printer, disk, CPU.
2. Queues are used in asynchronous transfer of data (where data is not being transferred at the same rate between two processes) for eg. pipes, file IO, sockets.

69. What sparse matrix? [WBSCTE 2022]

Answer:

A sparse matrix is a matrix that is comprised of mostly zero values. Sparse matrices are distinct from matrices with mostly non-zero values, which are referred to as dense matrices. A matrix is sparse if many of its coefficients are zero.

70. Give infix notation with an example. [WBSCTE 2022]

Answer:

Infix: The typical mathematical form of expression that we encounter generally is known as infix notation. In infix form, an operator is written in between two operands.

For example:

An expression in the form of **A * (B + C) / D** is in infix form. This expression can be simply decoded as: *"Add B and C, then multiply the result by A, and then divide it by D for the final answer."*

71. What do you understand by stack underflow and stack overflow? [WBSCTE 2022]

Answer:

Underflow happens when we try to pop an item from an empty stack. Overflow happens when we try to push more items on a stack than it can hold.

72. What are the disadvantages of linked list? [WBSCTE 2022]

Answer:

Disadvantages of Linked Lists:

- Use of pointers is more in linked lists hence, complex and requires more memory.
- Searching an element is costly and requires O(n) time complexity.
- Traversing is more time consuming and reverse traversing is not possible in singly linked lists.

Long Answer Type Questions

➲ **1. (a) Write an algorithm to multiply two matrices $A=[a_{MP}]$ and $B=[b_{PM}]$**

(b) Determine the complexity of your algorithm. [WBSCTE 2004, 2007, 2009]

Answer:

a) Algorithm:

MATMULTI (A,B,C,D,P,I)

Let A be an $M \times P$ matrix and let B be a $P \times N$ matrix. The algorithm stores the product of A and B in an M N matrix array C.

1. Repeat Steps 2 to 4 for I = 1 to M:
2. Repeat Steps 3 and 4 for J = 1 to N:
3. Set C[I, J]:=0, [Initializes C[I, j]]

4. Repeat for K = 1 to P:
   C[I, J]: = C[I, J] + A[I, K] * B[K,J]
   [End of inner loop.]
   [End to Step 2 middle loop]
   [End of step 1 outer loop.]
5. Exit.

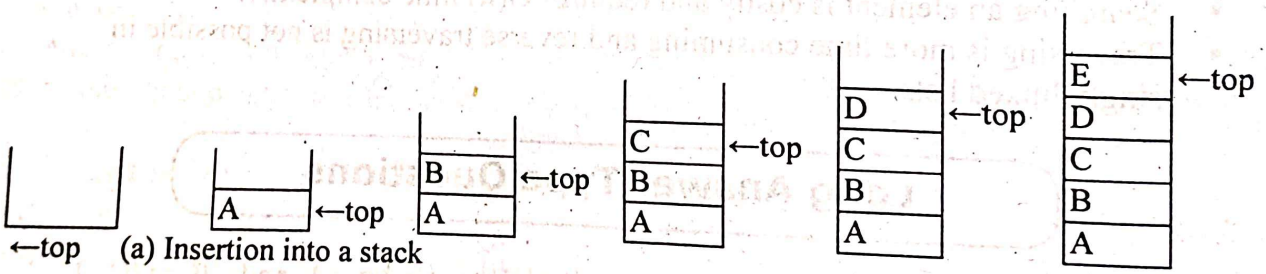
b) Complexity of a matrix multiplication algorithm is measured by counting the number C of multiplications. The reason that additions are not counted in such algorithms is that computer multiplication takes much more time than computer addition. The complexity of the above Algorithm is equal to $C = m \cdot n \cdot p$

➲ **2. What is stack? Why a stack is called LIFO? Describe the operations of stack with proper diagram and sequence of codes.** **[WBSCTE 2004, 2009]**

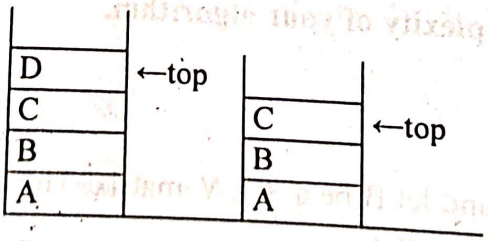
Answer:

A stack is a linear structure in which items may be added or removed only at one end. An item may be added or removed only from the top of the stack. This means that the last item to be added to a stack is the first item to be removed. Accordingly, stacks are also called last in first out (LIFO) lists. Other names used for stacks are pile and push-down lists. Although the stack may seem to be a very restricted type of data structure, it has many important applications in computer science.

Figure 1(a) depicts the expansion and shrinking of a stack. Initially the stack is empty. As the elements A, B, C, D, E are inserted into it, the stack grows continually upward. For example, the first element deleted from the stack in figure 1(a) was E, which had been the last element inserted into the stack. This is why a stack is often called a Last in First out (LIFO) list.



(a) Insertion into a stack



(b) After deletion of two elements of stack shown in (a)

Fig 1: Operations on a stack

```
void eval(char x[])
{
  int op1, op2, result, i = 1;
  while (x[i]! = '# ')
  {
    if (isdigit(x[i]))
      push(x[i]);
    else
    {
      op2 = pop(); /* pop the top element top will be decremented
by 1 */
      op1 = pop(); /* next pop will take the topmost element
again*/
      switch (x[i])
      {
        case '+':
          result = op1 + op2;
          break;
        case '-':
          result = op1 - op2;
          break;
        case '*':
          result = op1 * op2;
          break;
        case '%':
          result = op1 % op2;
          break;
      }
      push(result);
    }
    i++;
  }

  printf(" The result is= %d ", pop());
}
```

➲ **3. Write an algorithm for transforming an infix expression to its equivalent postfix expression. Trace your algorithm with following expression.** **[WBSCTE 2004]**

$A+B*(C+D)/F$

OR,

Write an algorithm to convert infix notation to postfix. **[WBSCTE 2008, 2009, 2010]**

Answer:

Input: E is an arithmetic expression in infix notation delimited the right parenthesis.

Output: An arithmetic expression in postfix notation.

Data Structure: Stack

```
InfixToPostfix(E)
1. Push ('(')
        [Push left parenthesis '(' on to stack]
2. Repeat Steps 3 to 20 While stack is not empty
3. Set item = E.scan_ch()
        [scan the symbol from infix expression]
4. If (item = operand)
5.      Write(item)
        [write the symbol into the output expression]
  [End of Step 4 If structure.]
6. If (item = '(')
7.      Push(item)
                [Push symbol '(' to the stack]
  [End of Step 6 If structure.]
8. If (item = operator)
9.      x = Pop()
10.     If precedence(x) ≥ precedence(item)
11.     Repeat Steps 12 and 13 while (precedence(x) ≥ precedence(item))
12.             Write(x)
13.             x = Pop()
                [End of Step 11 loop.]
        [end of Step 10 If structure.]
      [End of Step 8 If structure.]
14. Push(x)
15. Push(item)
16.     If (item = ')')
17.     x = Pop()
18.     Repeat Steps 19 and 20 while x ≠ '('
19.             write(x)


20.             x = Pop()
                [End of Step 18 loop.]
        [end of Step 16 If structure.]
   [End of Step 2 loop.]
21. Exit.
```

Example: Consider the following arithmetic infix expression E:

Equation A + B * (C + D)/F:

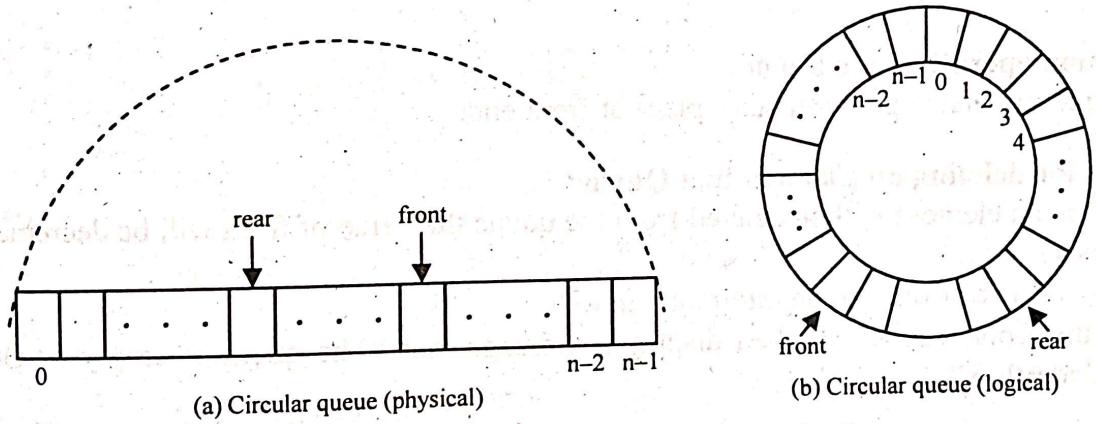
	Next Symbol	Stack	O/P
(1)	A	(	A
(2)	+	(+	AB
(3)	B	(+	AB
(4)	*	(+ *	AB
(5)	C	(+ * (	AB

(6)	C	(+ * (	ABC
(7)	+	(+ * (+	ABC
(8)	D	(+ * (+	ABCD
(9)	)	(+*	ABCD+
(10)	/	(+/	ABCD+*
(11)	F	(+/	ABCD+*F
(12)	)		ABCD+*F/+

4. (a) Describe briefly about - (i) Circular Queue, (ii) Priority Queue.
(b) Write an algorithm for insertion and deletion operations of a queue.
[WBSCTE 2004, 2008]

Answer:

a) (i) Circular Queue: Circular queue is a bounded queue, which implements arrays. 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 then also we cannot insert any new element because the insertion has to be done from one side only (i.e., rear or tail) and deletion has to be done from another side (i.e., front or head).But in case of circular queue the front and rear are adjacent to each other.



(a) Circular queue (physical) (b) Circular queue (logical)

Fig: Logical and physical view of a circular queue

(ii) Priority Queues

A **priority queue** is an abstract data type used for storing a set S of elements, based on a key value, which denotes the priority of that element. The priority determines the order in which they exit the queue. The element, having the highest priority, will be removed first.

A **priority queue** supports the following three operations:

- **insertWithPriority**: add an element to the queue with an associated priority
- **getNext**: remove the element from the queue that has the *highest priority*, and return it (also known as "PopElement(Off)", or "GetMinimum")
- **peekAtNext** (optional): look at the element with *highest priority* without removing it.

b) *Insertion operation of a queue:*

In a queue, insert operation takes place at rear end.

Steps for inserting an Element in a Queue:

a. Initialize both the front and rear as -1, which means that the queue is empty.
b. When the first element will be inserted then the rear and front both will be incremented by 1. But for the second element onwards only the rear will be incremented.
c. The value of rear can be maximum up to Max-1, where Max is the maximum number of elements a queue can hold.
d. If the rear reaches Max-1, then display a message that "The queue is full or Queue Overflow".

Algorithm for Insert Operation:

If rear=MAX
Print "Queue is full"
Else
rear=rear+1
Queue[rear]=value
END

Deletion operation of a queue:

In a queue, delete operation takes place at front end.

Steps for deleting an Element in a Queue:

a. When an element will be deleted from the queue the value of front will be decremented by 1.
b. The value of front can be minimum up to 0.
c. If the front reaches -1, then display a message that "The queue is empty or Queue Underflow".

➲ **5. Write an algorithm to reverse a single linked list.** **[WBSCTE 2008, 2012, 2013]**

Answer:

```
Node* ReverseList( Node ** List )
{
    Node *temp1 = *List;
    Node * temp2 = NULL;
    Node * temp3 = NULL;
    while ( temp1 )
    {
          *List = temp1; //set the head to last node
temp2= temp1->pNext; // save the next ptr in temp2
          temp1->pNext = temp3; // change next to privous
          temp3 = temp1;
          temp1 = temp2;
```

```
    }
    return *List;
}
```

6. Write algorithms to insert and delete elements from a circular queue. [WBSCTE 2009]

OR,

Write a routine for implementing insertion and deletion of element in circular queue. [WBSCTE 2014, 2015]

OR,

Write the advantages of circular queue over linear queue. Write an algorithm to delete an item from a circular queue. Explain Deque and its types. [WBSCTE 2017]

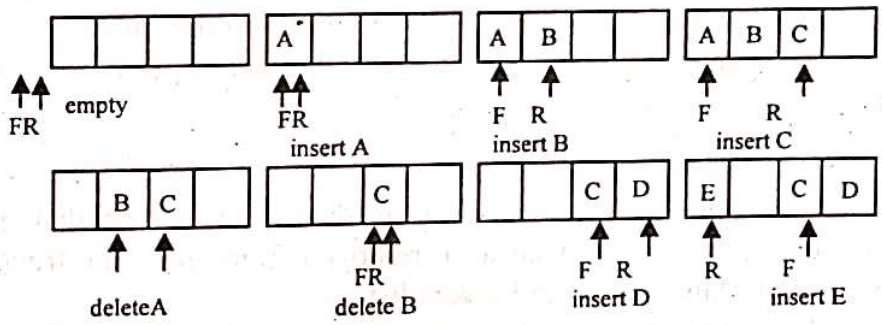
OR,

Write pseudocode to implement a circular queue using an array. [WBSCTE 2022]

Answer:

1st part:

The sequence of operations can be represented are shown below (in circular queue).



As we can see that the overflow issue in the previous case while inserting E is resolved by redirecting the rear to the front. This is the essence of circular queue.

Let us now write the insert and delete routine of the circular queue implementation. Let us also assume that F and R represent front and rear pointers respectively.

```
CQINSERT( item)
1. if (rear == N-1)
   1.1 printf("Queue Is Full");
   1.2     return;
   1.3 End of if structure.
2. CQ[++rear] = item;
3. if (front == - 1)
   3.1 front = 0;
   3.2 End.
CQDELETE()
 1. X be a variable declared;
```

```
2. if (front == - 1)
   2.1 printf("Queue Is Empty");
   2.2 exit(0);
   2.3 End of if structure.
3. x = CQ[front];
4. CQ[front] = - 1;
5. if (front == rear)
   5.1 front = rear = - 1;
   5.2 else
   5.3 front++;
6. return x;
7. End.
```

2nd Part:

When any element is inserted in linear queue then rear will be increased by 1. Let, assume after insertion operations rear is shifted to last position in queue. It means, now queue is full. Now if a new element is inserted then overflow condition will occur.

Now, if we delete some elements from queue then front will be increased by 1. After deletion memory space occupied by those elements will be blank and it should be reused by other new element. But it is not possible because rear is still pointing to the last position. Hence, we are not able to reuse free memory space present in linear/static queue and memory is not effectively used. This is overcome by circular queue hence memory is effectively used.

3rd Part:

A double-ended queue (abbreviated to deque) is an abstract data type that generalizes a queue, for which elements can be added to or removed from either the front (head) or back (tail).It is also often called a head-tail linked list.

This differs from the queue abstract data type or first in first out list (FIFO), where elements can only be added to one end and removed from the other. This general data class has following sub-types:

1) An **input-restricted deque** is one where deletion can be made from both ends, but insertion can be made at one end only.

2) An **output-restricted deque** is one where insertion can be made at both ends, but deletion can be made from one end only.

➲ **7. Given two sorted lists L1 and L2 write algorithm to compute L1 union L2. L1 and L2 are to be implemented as arrays.** **[WBSCTE 2009, 2010]**

Answer:

Input: Two sorted lists L1 and L2, such that no element occurs more than once in either list.

Output: A sorted list of the elements that occur in at least one of the lists L1 and L2 (and which contains no duplicates).

```
list union(list L1, list L2)
```

```
{
list result;
position L1pos = first(L1), L2pos = first(L2), resultPos =
first(result);
elementType insertElement;
while( L1pos != NULL && L2pos != NULL ) //while not at the end of
either list
{
if( L1pos->element < L2pos->element )
{
insertElement = L1Pos->element;
L1pos = L1pos->next;
}
else if( L1pos->element > L2pos->element )
{
insertElement = L2Pos->element;
L2pos = L2pos->next;
}
else
{
insertElement = L1Pos->element;

L1pos = L1pos->next;
L2pos = L2pos->next;
}
insert(insertElement, result, resultPos);
resultPos = resultPos->next;
}
//At most one of the following two while loops gets executed
while( L1pos != NULL ) //while not at the end of L1
{
insert(L1pos->element, result, resultPos);
L1pos = L1pos->next;
resultPos = resultPos->next;
}
while( L2pos != NULL ) //while not at the end of L2
{
insert(L2pos->element, result, resultPos);
L2pos = L2pos->next;
resultPos = resultPos->next;
}
return result;
}
```

➲ **8. Write an algorithm to remove repetitive entries from an array, what is the running time of this procedure?** [WBSCTE 2010]

Answer:

```
void rmdup(int *array, int length)
{
    int *current , *end = array + length - 1;
    for ( current = array + 1; array < end; array++, current =
array + 1 )
    {
        while ( current < end )
        {
            if ( *current == *array )
            {
                *current = *end--;
            }
            else
            {
                current++;
            }
        }
    }
}
```

The running time of this algorithm should be O(n^2) or less.

➲ **9. Mention two advantages of sequential allocation scheme of sparse matrix over linked-allocation scheme.** **[WBSCTE 2011]**

Answer:

Two advantages of sequential allocation scheme of sparse matrix over linked-allocation scheme

1. It allows faster execution of matrix operation.
2. It is more storage efficient

➲ **10. Explain the terms overflow and underflow in the context of stack data structure. Write a program to implement a stack that stores character data.** **[WBSCTE 2012, 2019]**

OR,

Explain the terms overflow and underflow. **[WBSCTE 2014, 2015]**

Answer:

Suppose a new data is to be inserted or 'push' in a stack. Stack overflow is a condition arises when the stack is completely full (so, no room for the new data).
Similiarly if we want to 'pop' or delete data from an empty stack, the situation refers as stack underflow.

Implementation of stack

```
#define N 10                    // maximum elements in a stack
int top = - 1  // indicates empty stack
```

```
char stck[N] // initially all the elements are -1.
void PUSH(char item)
{
  if (top == N - 1)
  {
    printf("STACK IS FULL");
    exit(0);
  }
  stck[++top] = item;
}
int POP()
{
  char x;
  if (top ==  - 1)
  {
    printf("STACK IS EMPTY");
    exit(0);
  }
  x = stck[top--];
  stck[top + 1] =  - 1;
  return x;
}
```

➲ **11. Convert the prefix expression - -/ab*+bcd into infix expression and then draw the corresponding expression tree.** **[WBSCTE 2012. 2013, 2019]**

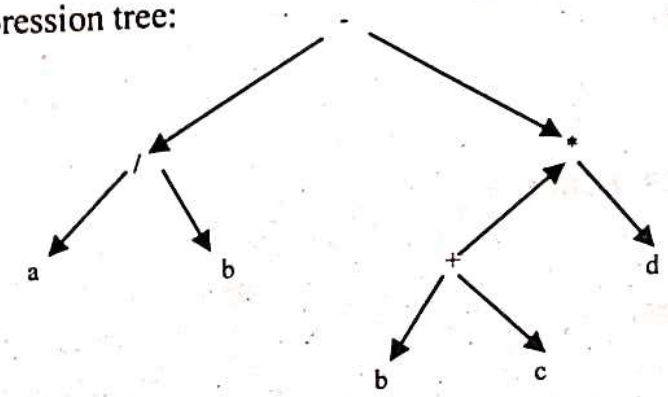
Answer:

Prefix expression: -/ab*+bcd

Symbol scanned	Stack	Output
-	-	
/	-/	
A	-/	a
B	-	a/b
*	-*	a/b
+	-*+	a/b
B	*+	a/b-b
C	*	a/b-b+c
D		a/b-b+c*d
		(a/b)-(b+c)*d

Infix expression: (a/b)-(b+c)*d

Corresponding expression tree:



➲ **12. Write algorithms to implement the queue insertion and deletion using linked list.** **[WBSCTE 2012]**

Answer:

The queue can be declared as follows:

```
typedef struct linked_list
{
  int data;
  struct linked - list * next;
} queue;
queue *front,  *rear;
```

LQinsert: An element can be inserted at the rear of the list. This concept is very much similar like the stack but the difference is that element will be added at the end of the list.

```
void LQinsert(int x)
{
  // initially both rear & front points to NULL.
  queue *p;
  p = (queue*)malloc(size of(queue));
  p -> data = x;
  if (front == NULL)
  {
    front = p;
    rear = p;
  }
  else
  {
    rear -> next = p;
    rear = rear -> next;
    rear ->next = NULL;
  }
}
```

Lqdelete: An element to be deleted from the beginning of the list. This concept is very much similar like the stack. Always we have to delete the first node.

```
int Lqdelete()
```

```
{
  queue *p;
  int x;
  if (front == NULL)
  {
    printf("Queue empty");
    exit(0);
  }
  p = front;
  if (p != NULL)
  {
    x = p -> data;
    front = front -> next;
  }
  free(p);
  return (x);
}
```

13. Consider the following two dimensional array: A(–5:5, 7:33) if Base address of A is 700 and there are w = 4 words per memory location. Find the effective address of A (4, 20) in row as well as column – major order. [WBSCTE 2013]

Answer:

No. of elements of row of A: $Nr_=$ (UB-LB)-1=(5+5)+1=11

No. of elements of column of A: $Nc_=$ (UB-LB)+1=(33-7)+1=27

B=700 ;W=4

Row Major Effective address calculation: A(i,j)=A(4,20)

B+ W*(Nc*(i-Lr)+(j-Lc))=700+4*(27*(4+5) +(20-7))=700+4*(27*9 +13)=1724

Column Major Effective address calculation:

B+W*((i-Lr) + Nr*(j-Lc))=700+4((4+5) + 11(20-7))=700+4(9+143)=1308.

14. a) Differentiate between an iterative function and recursive function. Which one will be preferred to which circumstances? [WBSCTE 2014, 2015, 2019]

b) What are the different types of recursion? [WBSCTE 2014, 2015]

OR,

What are the different types of recursion? Explain with example. [WBSCTE 2019]

c) Write a recursive routine to implement the factorial of *n* numbers. Explain with example. [WBSCTE 2014]

Answer:

a) 1st part:

Recursion vs. Iteration

- If a recursive method is called with a base case, the method returns a result. If a method is called with a more complex problem, the method divides the problem into two or more conceptual pieces: a piece that the method knows how to do and a slightly smaller version of the original problem. Because this new problem

looks like the original problem, the method launches a recursive call to work on the smaller problem.

- For recursion to terminate, each time the recursion method calls itself with a slightly simpler version of the original problem, the sequence of smaller and smaller problems must converge on the base case. When the method recognizes the base case, the result is returned to the previous method call and a sequence of returns ensures all the way up the line until the original call of the method eventually returns the final result.
- Both iteration and recursion are based on a control structure: Iteration uses a repetition structure; recursion uses a selection structure.
- Both iteration and recursion involve repetition: Iteration explicitly uses a repetition structure; recursion achieves repetition through repeated method calls.
- Iteration and recursion each involve a termination test: Iteration terminates when the loop-continuation condition fails; recursion terminates when a base case is recognized.
- Iteration and recursion can occur infinitely: An infinite loop occurs with iteration if the loop-continuation test never becomes false; infinite recursion occurs if the recursion step does not reduce the problem in a manner that converges on the base case.

Recursion repeatedly invokes the mechanism, and consequently the overhead, of method calls. This can be expensive in both processor time and memory sp.

2nd part:

Let us consider the recursive and iterative version of the same function fact() to determine which strategy is better in programming. Apparently just by looking the functions, we can conclude that recursion is better. But this is not the actual fact. We know internal stack operation is done by the language compiler to implement recursion. From the recursion tree, we can conclude that the time requirement is directly proportional to 2*n whereas the space requirement is proportional to n. But if we consider the iterative version, the time requirement is directly proportional to n and space requirement is constant. So, here the iterative version is better.

Iterative

```
int fact(int n)
    {
    int i, result = 1;
    if(n == 0)
        result = 1;
    else
     {
     for(i = 1; i <= n; i ++)
     result * = i;
     }
     return(result);
    }
```

Recursive

```
int fact(int n)
{
   if (n = = 0)
      return (1);
    else
   return ( n * fact(n-1));
}
```

There are two key requirements to make sure that the recursion is successful:

- Every recursive call must simplify the computation in some way.
- There must be special cases to handle the simplest computations.

b) 1st part:

Different types of recursion

- cdr resursion
- car-cdr recursion
- Linear recursion
- Flat recursion
- Deep recursion
- Mutual recursion
- A special recursion -- tail recursion (Efficiency)

2nd Part:

Linear recursion example:

We are given an array arr of n decimal integers and we have to find the maximum element from the given array. We can solve this problem using linear recursion by observing that the maximum among all integers in arr is arr[0], if n==1, or the maximum of first n-1 in arr and the last element in arr.

```
#include <stdio.h>
#define SIZE 10
int recursiveMax (int *, int );
int max (int, int);
int main ()
{
    int arr[SIZE] = {1, 3, 5, 4, 7, 19, 6, 11, 10, 2};
    int max = recursiveMax(arr, SIZE);
    printf("Maximum element among array items is: %d\n", max);
}
int recursiveMax (int *arr, int n)
```

```
{
    if (n == 1)
        return arr[0];
    return max (recursiveMax (arr, n-1), arr[n-1]);
}
/* helper function to compute max of two decimal integers */
int max (int n, int m)
{
    if (n > m) return n;
    return m;
}
```

OUTPUT

```
======
Maximum element among array items is: 19
```

c)

```
#include<stdio.h>
 long factorial(int);
 int main()
 {
   int n;
   long f;
    printf("Enter an integer to calculate it's factorial\n");
  scanf("%d", &n);
   if (n < 0)
    printf("Negative integers are not allowed.\n");
  else
  {
    f = factorial(n);
   printf("Factorial of %d = %ld\n", n, f);
  }
   return 0;
}
 long factorial(int n)
{
  if (n == 0)
    return 1;
  else
    return(n * factorial(n-1));
}
```

Output of code:

```
Enter a number to calculate it's factorial
6
Factorial of 6 = 720
Process return 0 (0x0) execution time : 2.891 s
Press any key to continue.
```

15. a) Describe a situation where storing items in an array is clearly better than storing items on a linked list.

b) Write an algorithm to implement the insertion and deletion of element into double linked list.

[WBSCTE 2014]

Answer:

a) An array is very good when it comes to index accesses. If our application needs to access elements at specific positions very often, we should rather use an array, because cannot access linked list through [ ].

b) Algorithm to implement the insertion and deletion of element into double linked list:

Insertion

Let us assume that x is data value to be inserted after the node containing the data value y. p initially contains the address of the head node.

The algorithm for the same is given below:

Step 1: start traversing the linked list from the head node

Step 2: Identify the address of the node containing data value y

Step 2a) after identification let p contains the address of data value y

Step 3: create a new node q

Step 4: place x to the data field of the new node

Step 5: place the next address of the p node to the next address of the q node

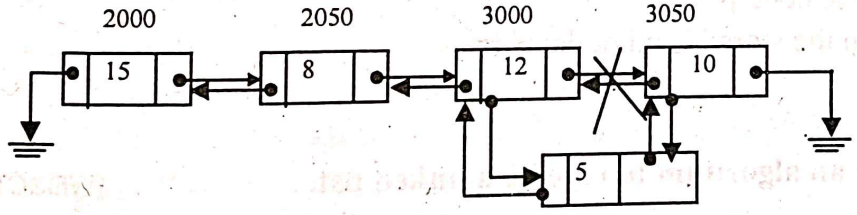
Step 6: place p to the prev of q

Step 7: change the prev of next of p as q

Step 8: change the next address of the p node by the q

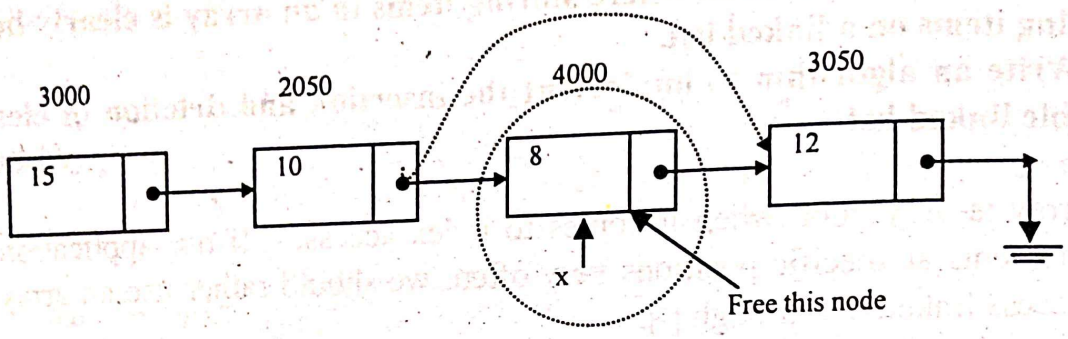
Step 9: End

This is shown in the following diagram.



Deletion

Before deleting the specific node, our first task is to identify the position of that node. Here the location of the previous node is not required because this is a doubly linked list. We can traverse the list in backward direction or forward direction from any node. When we are deleting a node, the general convention is to free the node. Another thing is that, if we delete the head node, we have to store the address of the new head node. The diagram is shown below.



The algorithm for deleting a node in the above manner is given below:
Let us assume that node containing data value x is to be deleted. p initially contains the address of the head node.

Step 1: store the address of the head node then start traversing the linked list from the head node
Step 2: if the data value of the head node is not equal to x goto step 7
Step 3: make the next node of head as new head node
Step 4: make the prev of the new head node as NULL
Step 5: free the previous head node
Step 6: return the address of the new head node & Goto Step 12
Step 7: Identify the address of the node containing data value x as follows
Step 7a) if the data value of p is equal to x
Step 7b) else change p by its next node address
Step 8: change the prev of next of p by prev of p
Step 9: change the next of prev of p by next of p
Step 10: free the node p
Step 11: return the stored head node address
Step 12: End

➲ 16. Write an algorithm to reverse a linked list. **[WBSCTE 2014, 2019]**

Answer:

Reverse a linked list (iterative algorithm)

1. The head node's next pointer should be set to NULL since the head will become the tail. This is an exception for the head node, and can be done outside the while loop. But, before we do this we will need a temp variable to point to the 2nd node (the node after the head node), because the only way to reference the 2nd node is through the head node's next pointer.

2. The 2nd node (the node after the head node) should have it's own next pointer changed to point to the head node. This will reverse the order of the nodes. But, the 2nd node's next pointer will at first be pointing to the 3rd node. This means that before we change the 2nd node's next pointer, we have to save a reference to the 3rd node otherwise we

will have no way of referencing the 3rd node. So, we simply store a reference to the 3rd node in a variable before we change the 2nd node's next pointer.

3. The 3rd node then becomes the "first" node in the while loop and we repeat the process of changing pointers described in step 2.

4. Continue step 3 until we come across a node that has a next pointer set to NULL. When we do come across a NULL next pointer we just set the head node to point to the node that has the NULL next pointer. This node was previously the tail node, but is now the head node because we are reversing the linked list.

➲ **17. Write a recursive and a non-recursive routine to implement the tower of Hanoi.** **[WBSCTE 2015]**

Answer:

Recursive Routine

```
FUNCTION MoveTower(disk, source, dest, spare):
IF disk == 0, THEN:
   move disk from source to dest
ELSE:
   MoveTower(disk - 1, source, spare, dest)   // Step 1 above
   move disk from source to dest              // Step 2 above
   MoveTower(disk - 1, spare, dest, source)   // Step 3 above
END IF
```

Non Recursive Routine

```
Step 1: start
Step 2: declare the no
Step 3: read the no value
Step 4: if (no<1) Print nothing to move
        Else Print nonrecursion
Step 5: Hanoi non recursion(no,.A','B','C')
Step 6: stop

Sub program:
Step 1: Declare num,sndl,indl,dndl,
        stkn[ ],stksndl[ ],stkdndl [ ],stkadd[ ],
         temp,top,add
Step 2: declare the top=NULL

Step 3: one:
                If(num==1)then
                Print the output value
```

```
            Goto four
Step 4: two:
            Top=top+1
Stkn[top]=num
Stksndl[top]=sndl
Stkindl[top]=indl

Stkdndl[top]=dndl
Stkadd[top]=3
Num=num-1
Sndl=sndl
Temp=indl
Indl=dndl
Dndl=temp
Goto one. Goto step 3
Step 5:
      Three:
            Print the output
            Top=top+1
Stkn[top]=num
Stksndl[top]=sndl
Stkindl[top]=indl
Stkdndl[top]=dndl
Stkadd[top]=5
Num=num-1
temp=sndl
sndl=indl
Indl=temp
Dndl=dndl
Goto one. Goto step 3
Step 6:
      Four:
            If(top==NULL)
            Return to main program
            Num= stkn[top]
Sndl= stksndl[top]
Indl= stkindl[top]
Dndl=stkdndl[top]
Add=stkadd[top]
Top=top-1
            If(add==3)
            Goto three. Goto step 5
```

Else
If(add==5)
Goto four. Goto step 6.

Step 7: return to main program

➲ **18. (a) Write down the functions PUSH( ) an POP( ) in C to implement a stack.**
(b) Explain the terms 'underflow' and 'overflow' with respect to a stack?
[WBSCTE 2016]

Answer:

a) The two function push() and pop () of stack in C are as follows:
Here n-1 is the maximum size of the stack and top=-1 means no element in the stack.

```
void push()
{
    if(top>=n-1)
    {
        printf("\n\tSTACK is over flow");
        getch();
    }
    else
    {
        printf(" Enter a value to be pushed:");
        scanf("%d",&x);
        top++;
        stack[top]=x;
    }
}
void pop()
{
    if(top<=-1)
    {
        printf("\n\t Stack is under flow");
    }
    else
    {
        printf("\n\t The popped elements is %d",stack[top]);
        top--;
    }
}
```

b) ***Refer to Question No. 10 Long Answer Type Questions.***

➲ **19. (a) What do you mean by 'Circular Queue'?** **[WBSCTE 2016]**

Answer:

Refer to Question No. 4(a) of Long Answer Type Questions.

➲ 19. (b) How the limitations in linear Queue can be avoided using Circular Queue? [WBSCTE 2016]

Answer:

Limitation of linear queue:

The linear queue suffers from serious drawback that performing some operations, we cannot insert items into queue, even if there is space in the queue. Suppose we have queue of 5 elements and we insert 5 items into queue, and then delete some items, then queue has space, but at that condition we cannot insert items into queue. This led to the discovery of the circular queue where the end is linked to the front or head thus the free space is utilized.

➲ 19. (c) Explain 'Queue Full' and 'Queue Empty' conditions for a Circular Queue using code of implementation. [WBSCTE 2016]

Answer:

Circular Queue: 'Queue full'

```
Void  insert( )
{ int num ;
 print ("\n Enter the number to be inserted in the queue");
Scanf  ("%d", & num);
if (front = = 0&& rear = = MAX-1)
print f ('Queue is full');
else if (front = = -1 & & rear = = -1)
{
 front = rear =0;
queue[rear] = num;
}
else if (rear = MAX-1 & & front !=0)
{
rear =0;
 queue [rear] = num;
}
else {
 rear ++;
queue [rear]=num;
}
}
```

Queue Empty condition for circular queue

```
int  delete_element( )
{
int  val;
if (front = = -1 &&rear = = -1)
{
print f ("\n underflow");
 return -1;
```

```
}
val = queue [front];
if (front = = rear)
front  =  rear  =  -1;

else {
 if (front  =  max-1)
front  =  0 ;
else
 front ++;
}
return val;
}
```

➲ **20. Write down the algorithm to implement to convert the infix expression into postfix expression.** **[WBSCTE 2016]**

Answer:

Refer to Question No. 3 of Long Answer Type Questions.

➲ **21. What are the various applications of a stack? Draw the stack structure in each case when the following operations are performed on an empty stack. (a) Add A, B, C, D, E, F, (b) Delete two letters, (c) Add G, H, (d) Delete four letters, (e) Add I.** **[WBSCTE 2017]**

Answer:

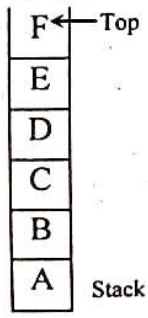
1st Part:

Application of Stack:

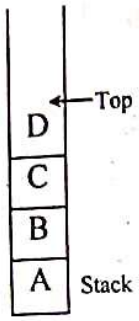
1) Expression evaluation and syntax parsing
2) Backtracking
3) Compile time memory management
4) Efficient Algorithm

2nd Part:

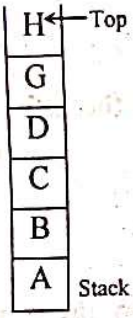
a) Add A, B, C, D, E, F



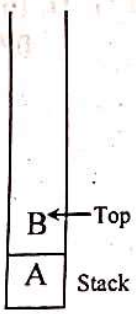
b) Delete two letters.. F & E deleted



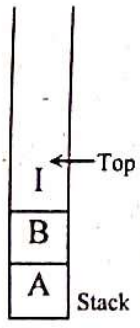
c) Add G, H



d) Delete four letters..H,G,D,C deleted



e) Add I



22. a) Explain Stack with push and pop algorithm.
b) Write some applications of stack data structure.
c) Evaluate the postfix expression. 4 2 $ 3 * 3 – 8 4 / 1 1 + / + [WBSCTE 2018, 2021]

Answer:

a) Push

This is the insertion operation.when the value is entered it is inserted into an array.

Algorithm

Push(S,TOP,X).This procedure inserts an element X to the top of a stack which is represented by a vector S containing N elements with a pointer TOP denoting the top element of the stack.

1. [check for stack overflow]
   if TOP >=N
   then Write("stack overflow")
   Return
2. [increment Top]
   TOP←TOP+1
3. [insert element]
   S[TOP] ←X
4. [finished]
   Return

The first step of this algorithm checks for overflow condition.If such a condition exits then the insertion operation cannot be performed.

Pop

This operation is used to remove a data item form the stack.

Algorithm

Pop(S,TOP). This function removes the top element from the stack which is represented by the vector S and returns this element. TOP is a pointer to the top of the stack.

1. [check for underflow of the stack]
   if TOP=0
   then Write("stack underflow on pop)
   Exit
2. [decrement pointer]
   TOP←TOP-1
3. [return former top element of the stack]
   Return(S[TOP+1])

An underflow condition is checked in the first step. If there is an underflow then appropriate actions should take place.

b) Application of stack data structure includes:

Expression Evaluation: Stack is used to evaluate prefix, postfix and infix expressions.

Expression Conversion: An expression can be represented in prefix, postfix or infix notation. Stack can be used to convert one form of expression to another.

Syntax Parsing: Many compilers use a stack for parsing the syntax of expressions, program blocks etc. before translating into low level code.

Backtracking: Suppose we are finding a path for solving maze problem. We choose a path and after following it we realize that it is wrong. Now we need to go back to the beginning of the path to start with new path. This can be done with the help of stack.

Parenthesis Checking: Stack is used to check the proper opening and closing of parenthesis.

String Reversal: Stack is used to reverse a string. We push the characters of string one by one into stack and then pop character from stack.

Function Call: Stack is used to keep information about the active functions or subroutines.

c) The postfix expression. 4 2 $ 3 * 3 – 8 4 / 1 1 + / +

Symbol	Operand 1	Operand 1	Value	Operand Stack
4				4
2				4,2
$	4	2	16	16
3				16,3
*	16	3	48	48
3				48,3
-	48	3	45	45
8				45,8
4				45,8,4
/	8	4	2	45,2
1				45,2,1
1				45,2,1,1
+	1	1	2	45,2,2
/	2	2	1	45,1
+	45	1	46	46

Answer = 46.

➲ **23. a) Write Insertion and Deletion algorithm of circular queue.**
b) Explain input and output restricted deque. **[WBSCTE 2018, 2021]**

Answer:

a) ***Refer to Question No. 6 (1st part) of Long Answer Type Questions.***

b) ***Refer to Question No. 6 (3rd part) of Long Answer Type Questions.***

➲ **24. Write a routine to reverse a string using recursion.** **[WBSCTE 2019]**

Answer:

```
# include <stdio.h>
 /* Function to print reverse of the passed string */
void reverse(char *str)
{
   if (*str)
   {
       reverse(str+1);
       printf("%c", *str);
   }
}
 /* Driver program to test above function */
int main()
{
   char a[] = "I am happy";
   reverse(a);
   return 0;
}
```

Output: yppah ma I.

➲ **25. Suppose multidimensional arrays *A* and *B* are declared as $A(-2:2,\ 2:22)$ and $B(1:8,\ -5:5,\ -10:5)$.**

(i) Find the length of each dimension and the number of elements is *A* and *B*.

(ii) Consider the element B[3.3.3] in B. Fine the effective indices E1, E2, E3 and the address of the element assuming Base (B) = 400 and that there are W = 4 words per memory location. **[Model Question]**

Answer:

(i) For Array A (-2:2, 2:22)

Length of the dimension $L_1 = (UB - LB) + 1 \quad L_1 = 2 - (-2) + 1 \quad L_1 = 5$

Length of the dimension $L_2 = (UB - LB) + 1 = 22 - 2 + 1 \quad = 21$

For Array B (1:8,-5:5, - 10 : 5)

Length of the dimension $L_1 = (UB - LB) + 1 \quad = 8 - 1 + 1 \quad = 8$

Length of the dimension $L_2 = (UB - LB) + 1 \quad = 5 - (-5) + 1 = 11$

Length of the dimension $L_3 = (UB - LB) + 1 = 5 - (-10) + 1 = 16$

(ii) The effective index E_i is obtained from $E_i = k_i - \text{LB}$, where k_i is the given index and LB is the lower bound. Hence

$E_1 = 3 - 1 = 2 \qquad E_2 = 3 - (-5) = 8 \qquad E_3 = 3 - (-10) = 13$

The address depends on whether the programming language stores B in row-major order or column-major order. Assuming B is stored in column-major order, we use the formula

$$Base(C) + w\left[\left(\left(\left(\ldots\left(E_N L_{N-1} + E_{N-1}\right)L_{N-2}\right) + \cdots + E_3\right)L_2 + E_2\right)L_1 + E_1\right]$$

$E_3 L_2 = 13 \cdot 11 = 143$ $\quad E_3 L_2 + E_2 = 143 + 8 = 151$

$(E_3 L_2 + E_2) L_1 = 151 \cdot 8 = 1208$ $\quad (E_3 L_2 + E_2) L_1 + E_1 = 1208 + 2 = 1210$

Therefore, LOC(B[3, 3, 3]) = 400 + 4(1210) = 400 + 4840 = 5240

➲ 26. What is sparse matrix? Explain the procedure involved in representing sparse matrix. **[Model Question]**

Answer:

A matrix is represented in memory as 2D array, like A [i, j], where i is a row and j is a column. If there are m rows and n columns, then total elements in a matrix or size of a matrix is m x n elements. This m x n elements requires m x n units of storage. A matrix that is filled with mostly zeroes (approx more than 2/3 elements are zero) is called as a ***sparse matrix.***

Now, if there is a sparse matrix of order m x n, and if we use a 2D array of order m x n to store this matrix, then there is basically large amount of wastage of memory.

A sparse matrix can be stored as a list of three tuples of the form (i, j, value)

The representation of the sparse matrix will be an array consisting 3 columns and (t+1) rows where t = no. of non-zero elements. The first row contains information about the number of rows, columns and non-zero elements in the sparse matrix. Subsequent rows are maintained, one row for each non-zero element, and they contain information about the row, column and non-zero value in the respective 3 columns.

In this way we can store only non-zero elements.

Let us consider a matrix M (5 x 6)

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & -3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 30 \end{bmatrix}$$

Here the number of non-zero elements t = 5.

The array A [0...t, 1...3] can store all the non zero elements as given below:

	1	2	3
A[0	5	6	5
A[1	1	5	1
A[2	2	1	2
A[3	3	2	-3
A[4	4	4	7
A[5	5	6	30

The element A [0, 1] and A [0, 2] contain the number of rows and columns of the matrix. A[0, 3] contains total number of non-zero items.

27. Write a function to evaluate value of the determinant of any square matrix of order n. [Model Question]

Answer:

Let f[20][20] is the matrix and n is the order of the matrix

```
int determinant(int f[20][20],int n)
{
  int pr,c[20],d=0,b[20][20],j,p,q,t;
  if(n==2)
  {
    d=0;
    d=(f[1][1]*f[2][2])-(f[1][2]*f[2][1]);
    return(d);
   }
  else
  {
    for(j=1;j<=n;j++)
    {
      int r=1,s=1;
      for(p=1;p<=n;p++)
        {
          for(q=1;q<=n;q++)
            {
              if(p!=1&&q!=j)
              {
                b[r][s]=f[p][q];
                s++;
                if(s>n-1)
                 {
                   r++;
                   s=1;
                  }
               }
             }
        }
      for(t=1,pr=1;t<=(1+j);t++)
      pr=(-1)*pr;
      c[j]=pr*determinant(b,n-1);
      }
      for(j=1,d=0;j<=n;j++)
      {
        d=d+(f[1][j]*c[j]);
       }
      return(d);
  }
```

Unit 3: Linked List

Unit at a glance:

3.1 Dynamic and Static Memory Allocation

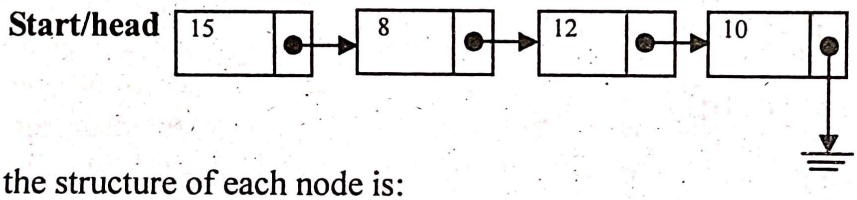
Dynamic memory allocation: memory space for any variable is allocated at the time of execution of the program.

Static memory allocation: this memory is allocated during the compile time of a program.

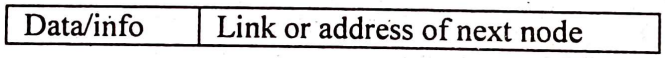
3.2 Linked List

Linked list is a linear data structure consisting of a group of nodes which together represent a sequence. Each node consists of at least two fields, one is to store the data & the other is the link field to store the address of the next node. The data field can also be called as information field and the link field is also called as the next address field.

The first node of the list is called as Header Node. It holds information about the entire list. The link field of the last node in the list contains a special value known as NULL, which signifies the end of the list. A list of integers (in linked list representation) is shown below:



the structure of each node is:



3.3 Type of Linked List

- Singly Linked List
- Doubly Linked List
- Multilinked Linked List
- Circular Linked List

Operations on Single Linked List

- Creation
- Insertion

In order to insert an element in a singly linked list three possibilities are there.

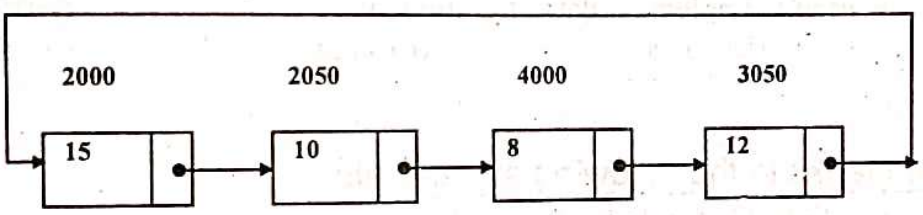
- Insert at the beginning of the list
- Insert at the end of the list
- Insert at the nth position of the list.
- Deletion
- Counting
- Printing In Reverse Order

Others operations on Linked List

- *checking* whether the list is empty;
- *accessing* a node to modify it or to obtain the information in it;
- *traversing* the list to access all elements (e.g., to print them, or to find some specific element);
- determining the *size* (i.e., the number of elements) of the list;
- *inserting* or *removing* a specific element (e.g., the first one, the last one, or one with a certain value);
- creating a list by *reading* the elements from an input stream;
- *converting* a list to and from an array, string, etc.

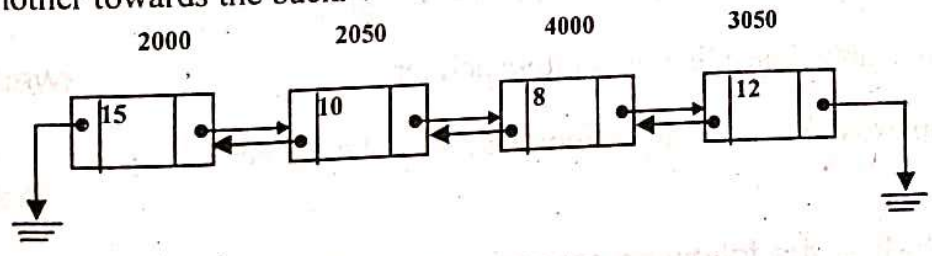
Circular Linked List

In case of the single linked list the address field of the last node contains NULL, to denote the end of the list. But in case of the circular linked list, the address of the last node contains the address of the first node, as shown in the figure below:



Doubly Linked List

The node of a doubly linked list contains two link fields, instead of one. One link is used to point to the predecessor node, i.e., the previous node & the other link is used to point to the successor node, i.e., the next node. So, the two links are directed, one towards the front and another towards the back.



3.4 Linked Stacks and Queues

A stack may be represented by a linked list. The first node of the list will be the topmost element of the stack and is pointed by a top pointer. This type of stack representation is called **linked stack.**

Short Answer Type Questions

A. Choose the correct answer from the given alternatives in each of the following:

1. Which type of linked list does not store NULL in next field? [WBSCTE 2017, 2021]
   (a) Singly linked list (b) Circular linked list
   (c) Doubly linked list (d) All of these
Answer: (b)

2. A linked list is a [WBSCTE 2017]
   (a) Random access structure (b) Sequential access structures
   (c) Both (d) none of these
Answer: (b)

3. Linked list is used to implement data structures like [WBSCTE 2018]
   (a) stack (b) queue (c) Trees (d) All of these
Answer: (d)

B. Fill in the blanks in the following statements:

4. Each entry in a linked list is defined as ***node***. [WBSCTE 2008, 2010]

5. A node in Double linked lists use ***right link & left link*** pointers. [WBSCTE 2008]

6. A dummy node at the start of a link list is known as ***header***. [WBSCTE 2009]

7. The link pointer of the last node of a link list is ***null***. [WBSCTE 2009]

8. Nodes in doubly linked lists use ***double*** pointers. [WBSCTE 2010]

9. The complexity to delete a node from the end of the linked list is ***O(n)***. [WBSCTE 2022]

C. State whether the following statements are True or False:

10. In a double linked list a mode consists of one pointer. [WBSCTE 2009]
Answer: False

11. Linked list is a non-linear data structure. [WBSCTE 2010]
Answer: False

12. Singly linked list is a type of linear data structure [WBSCTE 2011]
Answer: True

13. Data is nothing but an information. [WBSCTE 2011]
Answer: False

14. A node in singly linked can be point to only one node at a time. [WBSCTE 2012]
Answer: True

D. Answer the following questions:

15. "A linked list provides a natural tool for the representation as well as the manipulation of polynomials". Justify or reject the statement giving reasons. [WBSCTE 2011]

Answer:
Link list will be the best possible data structure to represent polynomial in computer.
A Polynomial has mainly two fields, exponent and coefficient.
Node of a Polynomial:

Exp	Coef	link

In real life applications we need to deal with integers that are larger than 64 bits (the size of a long). To manipulate with such big numbers, we will be using a linked list data structure.

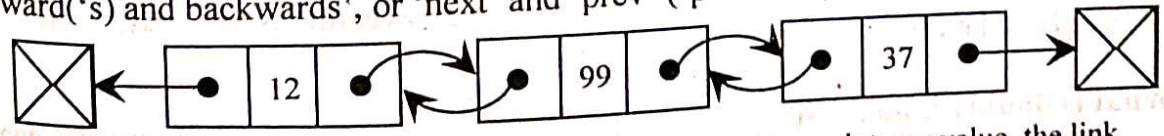
16. What are the advantages of link list over array? [WBSCTE 2012, 2013, 2014, 2015, 2019]

Answer:
A. Array size is fix and cannot change at run time, But in link list we can create memory according to requirement.
B. In array we define but at the run time it is not used so in that case memory is waste.

17. What pointer type is used to implement the heterogeneous linked list in C? [WBSCTE 2014]

Answer:
The heterogeneous linked list contains different data types in its nodes and we need a link, pointer to connect them. It is not possible to use ordinary pointers for this. So we go for void pointer. Void pointer is capable of storing pointer to any type as it is a generic pointer type.

18. Draw a node of doubly linked list and state its fields. [WBSCTE 2015]
Answer:
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('s) and backwards', or 'next' and 'prev' ('previous').

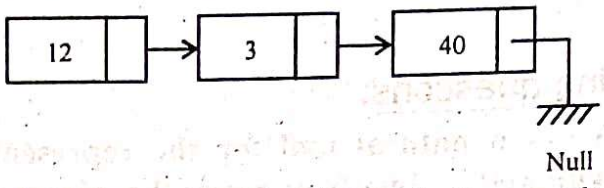


A doubly linked list whose nodes contain three fields: an integer value, the link forward to the next node and the link backward to the previous node

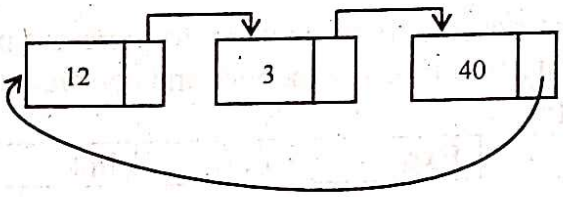
19. Distinguish between single linked list and circular linked list. **[WBSCTE 2016]**

Answer:

Single linked lists contain nodes which have a data field as well as a next field, which points to the next node in the line of nodes.



In the case of a circular linked list, the only change that occurs is that the end or 'tail' of the said list is linked back to the front or head of the list or vice- versa.



20. Write the advantages of doubly linked list. **[WBSCTE 2017, 2021]**

Answer:

Advantages/Disadvantages of Doubly Linked List:

Doubly linked list offers easy implementation of many operations, whereas singly linked list requires more info for the same operation. Like for the deletion of a node in a singly linked list. We definitely need the ptr to node previous to it. But in case of doubly linked list, we just need the ptr to the node being deleted.

But then some tradeoffs are there. Doubly linked list occupy more space and often more operations are required for the similar tasks as compared to singly linked lists.

The primary advantage of a doubly linked list is that given a node in the list, one can navigate easily in either direction.

21. What are the advantages of a circular linked list? **[WBSCTE 2018]**

Answer:

Refer to Question No. 14 (a) 2nd part of Long Answer Type Questions.

22. What are the disadvantages of linked list? **[WBSCTE 2022]**

Answer:

Disadvantages of Linked Lists:

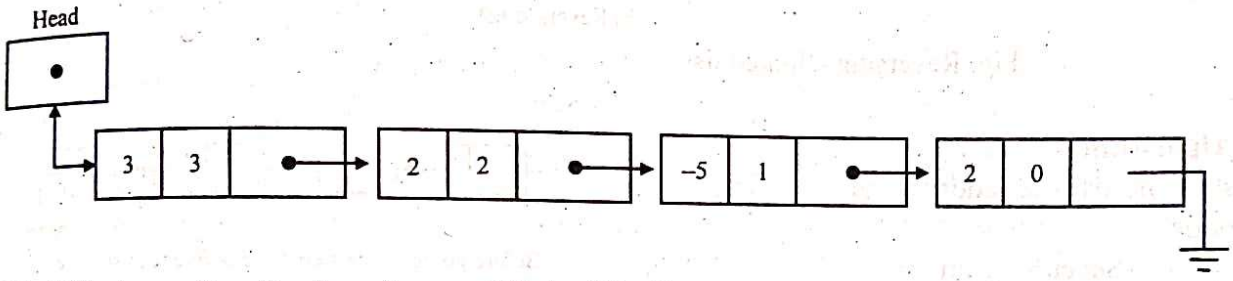
- Use of pointers is more in linked lists hence, complex and requires more memory.
- Searching an element is costly and requires O(n) time complexity.
- Traversing is more time consuming and reverse traversing is not possible in singly linked lists.

23. What is doubly linked list? **[WBSCTE 2022]**

Answer:
Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence.

24. Explain the Polynomial equation of linked list: $3x^3 + 2x^2 - 5x + 2 = 0$. [WBSCTE 2022]

Answer:



25. What are the disadvantages of linked list? [WBSCTE 2022]

Answer:

Disadvantages of Linked Lists:

- Use of pointers is more in linked lists hence, complex and requires more memory.
- Searching an element is costly and requires O(n) time complexity.
- Traversing is more time consuming and reverse traversing is not possible in singly linked lists.

Long Answer Type Questions

1. Describe a suitable data structure to store a variable length string of characters. Then write an algorithm to reverse the given string. Describe the functioning of your algorithm for the string append. What is the complexity of your algorithm? [WBSCTE 2006]

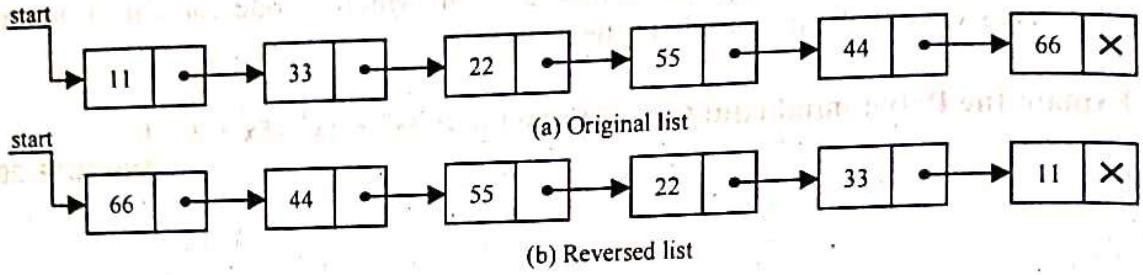
OR,

Write an algorithm to reverse the string "dasarpibed" using linked list. What is the computing time of your algorithm? [WBSCTE 2015]

Answer:

Link list will be the best possible data structure to store a variable length string of characters.

Figure (a) and (b) illustrate before and after reversing a single linked list respectively.



(a) Original list

(b) Reversed list

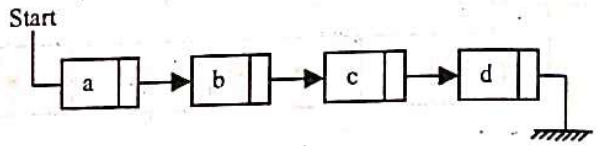
Fig: Reversing a linked list

Algorithm:

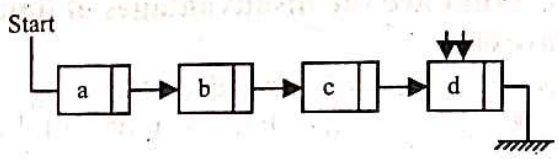
start holds the address of the first node.

1. Set curr = start
2. If next[curr] = NULL, then Exit
3. Set prev = next[curr]
4. Set next[curr] = NULL
5. Repeat Steps 6 to 9 while next[prev] ≠ NULL
6. Set ptr = next[prev]
7. Set next[prev] = curr
8. Set curr = prev
9. Set prev = ptr
   [End of Step 5 loop.]
10. Set next[prev] = curr
11. Set start = prev
12. Exit

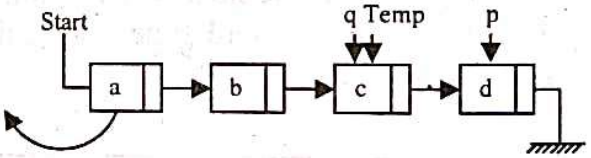
Complexity is O (n)



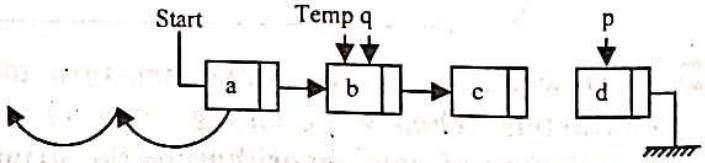
Before going to the step 4 the pointer are set like



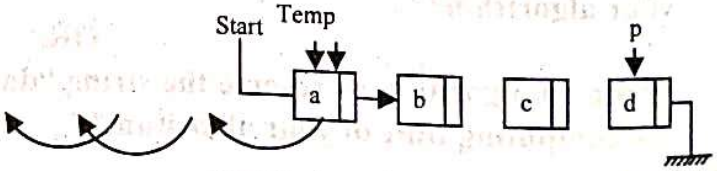
Now pass 4 q ≠ start True



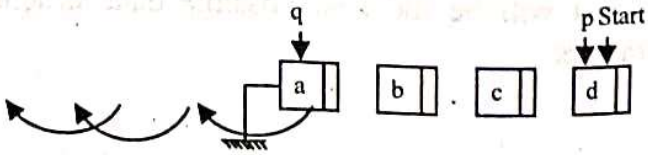
Pass II q ≠ start True



Pass III q ≠ start



Pass IV q ≠ start false



➲ **2. Show with the help of neat diagrams, how to insert and delete an element into/and from existing linked list. Write corresponding procedures.**

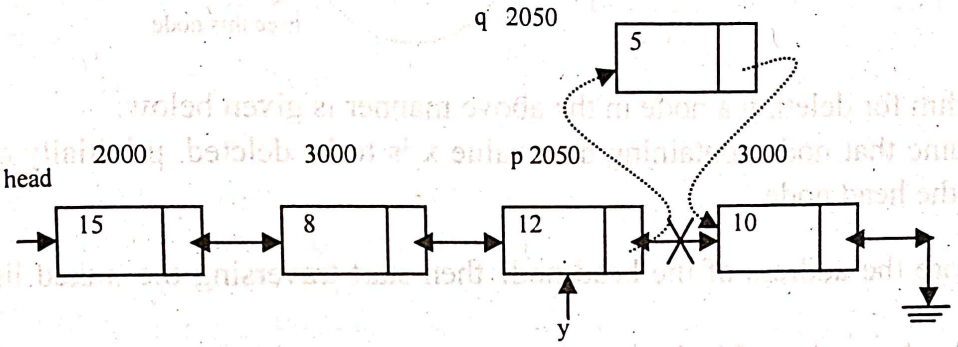
[WBSCTE 2007]

Answer:

Insert at nth position of the list

The algorithm for inserting in the above manner is given below:

Let us assume that x is data value to be inserted at nth position of the list. p initially contains the address of the head node. pos contains the position where the data to be inserted. In this case as there is a possibility that the head node address may change so the algorithm will return the address of head node at the end.



Step 1: store the address of the head node then start traversing the linked list from the head node

Step 2: if the position is not equal to 1 goto step 8

Step 3: create a new node

Step 4: place x to the data field of the new node

Step 5: change the next address of the new node by the head node address

Step 6: make this new node as head node

Step 7: Goto Step 13

Step 8: Identify the address of the node at n-1 th position

Step 9: create a new node

Step 10: place x to the data field of the new node

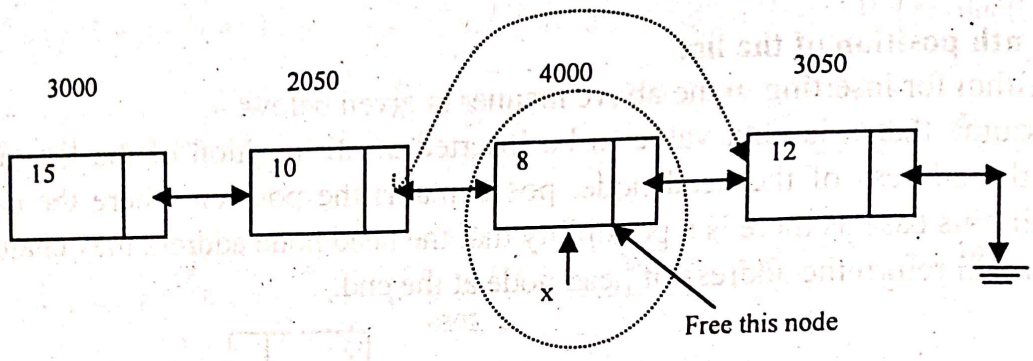
Step 11: place the next address of n-1th node to the next address of new node

Step 12: change the next address of the n-1th node by the new node address

Step 13: End

Deletion

Before deleting the specific node, our first task is to identify the position of not only that node, but also the location of the previous node. The reason is that, this is a singly linked list. We cannot traverse the list in backward direction. So, we have start checking the data value of the next node. So, when we have found match, our location will be just at the previous node of the specified node. When we are deleting a node, the general convention is to free the node. Another thing is that, if we delete the head node, we have to store the address of the new head node. The diagram is shown below.



The algorithm for deleting a node in the above manner is given below:
Let us assume that node containing data value x is to be deleted. p initially contains the address of the head node.

Step 1: store the address of the head node then start traversing the linked list from the head node
Step 2: if the data value of the head node is not equal to x goto step 7
Step 3: make the next node of head as new head node
Step 4: free the previous head node
Step 5: return the address of the new head node
Step 6: Goto Step 12
Step 7: Identify the address of the node containing data value x as follows
Step 7a) if the data value of the next node of p is equal to x
Step 7b) else change p by its next node address
Step 8: store the next node address of p in q
Step 9: change the next node address of p by next to next node address of p
Step 10: free the node q
Step 11: return the stored head node address
Step 12: End

➲ **3. Mention a suitable data structure to represent a polynomial in a computer. Write and explain an algorithm to add two polynomials, each represented through the data structure mentioned above.** **[WBSCTE 2007, 2011]**

Answer:

Link list will be the best possible data structure to represent polynomial in computer.
In order to write the algorithm polynomial addition using linked list, we have to define a structure as follows:

```
typedef struct poly
{
  int deg;
  int coeff;
  struct poly *next;
} term;
```

Now, we can declare three pointer variables a, b, c of term type.

```
term *a, *b, *c;
```

At first we have to store degree and coefficient of A(x) and B(x) through a and b, a contains the address of head node of the linked-list containing all the terms of A(x). Similar is the case for b and c. All these a, b and c are declared globally.

polyadd(term *a, term *b)

```
// a and b are the two polynomial which will be added and will be
store in r.//
  term *p,  *q,  *r,  *m;
  p = a;
  q = b;
  r = (term*)malloc(size of(term));
  1. Repeat step while (p != NULL && q != NULL)
      if (p->deg == q->deg)
        r->coeff = p->coeff + q->coeff;
        r->deg = p->deg;
        p = p->next;
        q = q->next;
    else if (p->deg >= q->deg)
        r->coeff = p->coeff;
        r->deg = p->deg;
        p = p->next;
     else if (p->deg <= q->deg)
       r->coeff = q->coeff;
       r->deg = q->deg;
       q = q->next;
    if (p != NULL && q != NULL)
      m = (term*)malloc(sizeof(term))
      r -  > next = m;
      r = m;
 End of if statement.
 End of repeat while.
2. Repeat step  while (p ! = NULL)
    m = (term*)malloc(sizeof(term));
    r -  > next = m;
    r = m;
    r->coeff = q->coeff;
    r->deg = q->deg;
    p = p->next;
  End of repeat while.
3.  Repeat step while (q ! = NULL)
    m = (term*)malloc(sizeof(term));
    r -  > next = m;
    r = m;
```

```
    r->coeff = q->coeff;
    r->deg = q->deg;
    q = q->next;
  End of repeat while.
4. End.
```

➲ **4. Swap two adjacent elements by adjusting only the pointers (and not the data) using – (a) singly linked lists, (b) double linked lists.** [WBSCTE 2008]

Answer:

a) singly linked lists

```
void swap_with_nest(Posiiton P, before P, List L)
{
Position P, After P;
P=before P→Next;
After P=P→Next;
P→Next=After P→Next;
Before P→Next=After P;
After P→Next=P;
}
```

b) double linked lists

NODE SWAP
METHOD SWAP_NODE (NODE A. NODE B, NODE PREVA)
A→NEXT – B→NEXT
B→NEXT – A
PREV →NEXT=B
End

➲ **5. Write the routines to implement queue using – (a) linked lists, (b) arrays. Discuss the advantages of using circular queue.** [WBSCTE 2008, 2009]

Answer:

a) Linked list representation of a queue

Here, each node is divided into two fields.

- info which holds the data element
- next which hoods the address of the successor node

Apart from that we need two more variables

- front which holds the address of the first node
- rear which holds the address of the last node

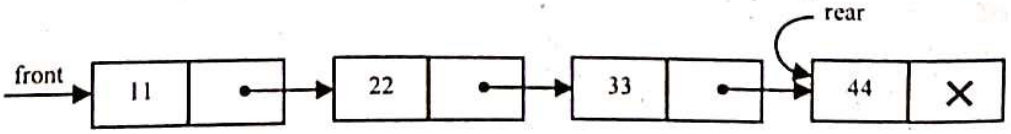
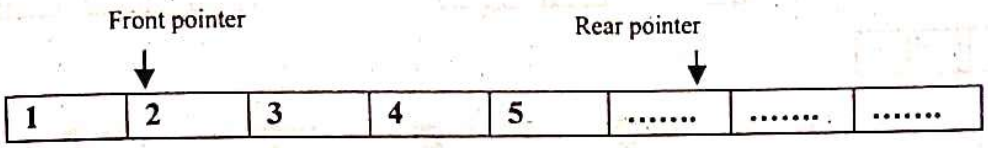


Fig: Linked list representation of queue

Under the list representation of q, two pointers named as front and rear are used. The operations IsEmpty(q) and delete(q) are completely analogous to the IsEmpty(s) and Pop(s) operations of stack. However, a special attention is required when the last element is removed from a queue. In that case, rear must also set to NULL, since in an empty queue both front and rear must be NULL.

b) Array Representation of Queue:

Queues are abstract data types. They are implemented either by an array or a linked list. In array representation, a pointer or a variable top is used to keep track of the stack. A variable N is used to store the maximum number of elements in the QUEUE.



The following are the necessary declarations:

```
# define MAXQ 100
typedef struct
{
      int front, rear;
      int items[MAXQ];
}queue;
queue q;
```

Advantages of circular queue:

(1) This is the list having the last node pointing the first node. Circular link list can be traversed from any node of the list means any node can be accessed from other node.
(2) Circular queue have less memory consumption as compared to doubly linked list. In circular queue the first is used as it comes immediate after the last. In circular queue the memory of the deleted process can be used by some other new process.
(3) Here each node has two parts: (i) Data (ii) Next address.

➲ **6. Write and explain an algorithm to add a node to a doubly linked list.** **[WBSCTE 2009]**

Answer:

Insertion of a node in a double linked list:

There are various positions where node can be inserted

i) Insert at beginning
ii) Insert at end
iii) Insert at a specific location
iv) Insert after a given node.

Insert at beginning

Figure 1(a) and 1(b) illustrate the double linked list before and after insertion respectively. The previous and next pointer field should be adjusted during insertion.

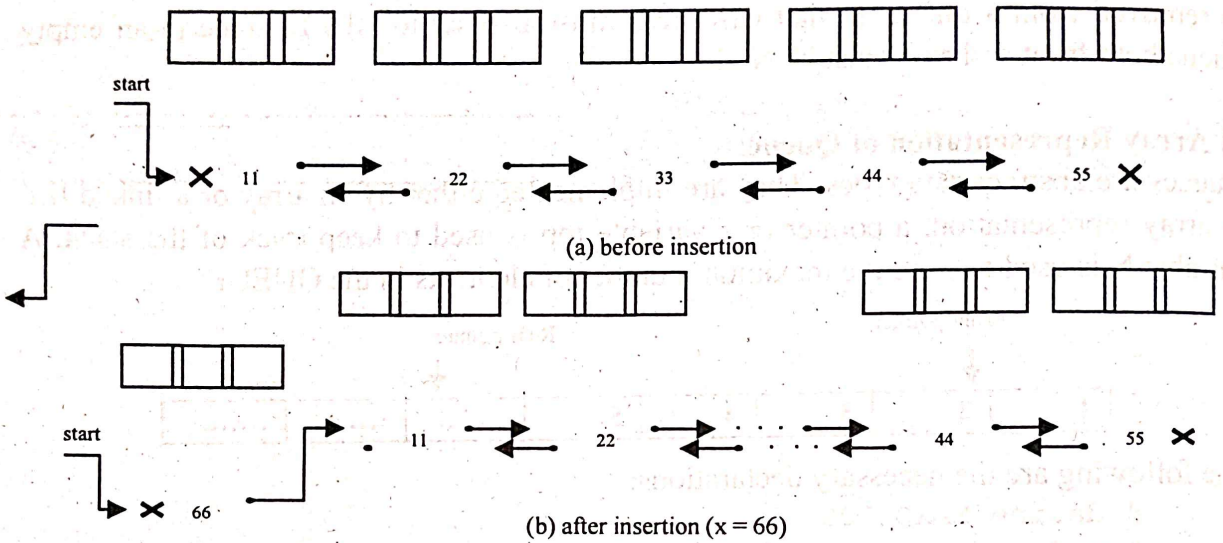


Fig: 1 Insert at the beginning

Algorithm:

Start holds the address of the first node.

1. Create a new node named as node
2. If node = NULL then Write 'Out of memory space' and Exit
3. Set info[node] = x [copies new data into new node]
4. Set next[node] = start [new node now points to original first node]
5. Set prev[node] = NULL
6. Set prev[start] = node
7. Set start = node
8. Exit

➲ **7. Write an algorithm to insert an element at last position of a singly linked list.** **[WBSCTE 2010]**

Answer:

C function which inserts at the end of a list

```
/* Node definition */
struct node
{
      int data;
      struct node *next;
};
struct node *BaseNode = '\0';
/*Adds at the end of the list */
void AddAtEnd(struct node **BaseNode, int data)
{
struct node *temp, *temp1;
temp = *BaseNode;
```

```
if(temp == '\0')
{
      /*List is empty - Create first node*/
      temp1 = (struct node *)malloc(sizeof(struct node));
      if(temp1 != '\0')
      {
            temp1->data = data;
            temp1->next = '\0';
            *BaseNode = temp1;
      }
}
else
{
      /*Node is existing already, So create another node*/
      while(temp->next != '\0')
      {
            temp = temp->next;
      }
      temp1 = (struct node *)malloc(sizeof(struct node));
      if(temp1 != '\0')
      {
            temp1->data = data;
            temp1->next = '\0';
            temp->next = temp1;
      }
}
}
```

➲ **8. (a) Write an algorithm to delete the middle element from the singly lined list.**
(b) Will there be any advantage in searching the number n if the values in the data fields in the linked list are sorted in ascending order. Explain?
[WBSCTE 2011]

Answer:

a) Algorithm:

Step 1: find the middle element of the linked list (i.e. n/2, if n is the total no. of the nodes) and let it say p^{th} address

Step 2: store the address of the head node then start traversing the linked list from the head node

Step 3: traverse upto $(p-1)^{th}$ node (store in pointer q)

Step 4: change the next node address of q by next to next node address of q

Step 5: return the stored head node address

Step 6: End

b) Yes, using a sorted list we can use a binary search which is less complex in nature and less time consuming. In Binary Search the entire sorted list is divided into two parts. We first compare our input item with the mid element of the list and then restrict our attention to only the first or second half of the list depending on whether the input item comes left

or right of the mid-element. In this way we reduce the length of the list to be searched by half. Let us consider a sorted array with the following.

a [ ] = 13 23 27 33 51 66 85

We find the mid-element by the formula

mid = a [left + right] / 2; left and right are the leftmost and rightmost index of the array. In our example left = 0 and right = 6.

Therefore mid = a [0 + 6] / 2 = a[3] = 33.Thus the array now looks like

Left array = 13 23 27 (33) **right array** = 51 66 85

Let us consider our input search item is **23.**

Step 1:

We compare 23 with our mid element i.e. 33. We find 23 < 33. Hence we move to the left of 33. The left side of 33 contains all the elements lesser than 33. In this way we restrict our search to one side of an array.

Step 2:

We again compute the mid of left array by the formula a [left + right] / 2 = a [0 + 2] / 2 = 1. We compare 23 with our new mid element i.e. 23. We found the match. Hence the program ends here. If the match were not found we would have continued with step 2. The C function, which accomplishes the above task, is as follows.

➲ 9. Represent the sparse matrix using link list. **[WBSCTE 2012, 2014, 2019]**

Answer:

In Figure, fields *i* and *j* store the row and column numbers for a matrix element respectively. DATA field stores the matrix element at the *i*-th row and the *j*-th column, i.e., a_{ij}. The ROWLINK points to the next node in the same row and COLLINK points the nest node in the same column. Thus, for a sparse matrix of order $m \times n$, we have to maintain m headers for all rows and n headers for all columns.

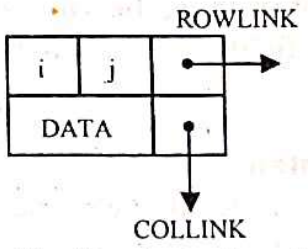
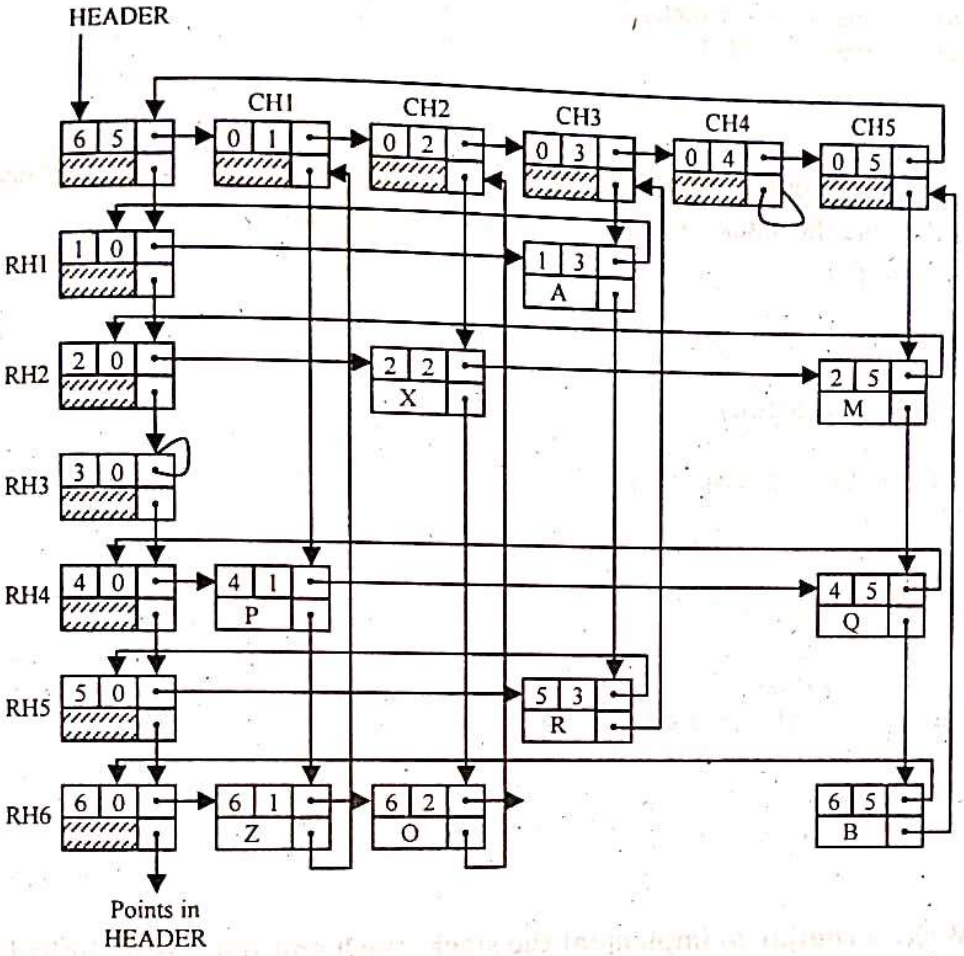


Fig: Structure of a node to represent sparse matrices

Column

Row	1	2	3	4	5
1	•	•	*A*	•	•
2	•	*X*	•	•	*M*
3	•	•	•	•	•
4	*P*	•	•	•	*Q*
5	•	•	*R*	•	•
6	*Z*	*O*	•	•	*B*

Fig: A sparse matrix of order 6 × 5 containing 9 elements only



➲ **10. Write a routine to implement the queue (insertion and deletion) using linked list.** **[WBSCTE 2013, 2014, 2019]**

Answer:

LQinsert: An element can be inserted at the rear of the list. This concept is very much similar like the stack but the difference is that element will be added at the end of the list.

```
void LQinsert(int x)
{
 // initially both rear & front points to NULL.
  queue *p;
  p = (queue*)malloc(size of(queue));
  p -> data = x;
  if (front == NULL)
  {
    front = p;
    rear = p;
  }
  else
  {
    rear -> next = p;
```

```
    rear = rear -> next;
    rear ->next = NULL;
  }
}
```

Lqdelete: An element to be deleted from the beginning of the list. This concept is very much similar like the stack. Always we have to delete the first node.

```
int Lqdelete()
{
  queue *p;
  int x;
  if (front == NULL)
  {
    printf("Queue empty");
    exit(0);
  }
  p = front;
  if (p != NULL)
  {
    x = p -> data;
    front = front -> next;
  }
  free(p);
  return (x);
}
```

➲ 11. Write a routine to implement the stack (push and pop) using linked list. **[WBSCTE 2015]**

Answer:

The required algorithms for PUSH and POP operations using Linked list as follows:

```
Algorithm PUSH(top,item,Avail)
{ // Avail is the list of free nodes which are maintained as pool
of stack//
   New: =Avail
   Avail: =Link[Avail]
   Info[New]: = item
   Link[New]: =top
   Top: =New
}
Algorithm POP(top.item.Avail)
{ // Avail is the list of free nodes which are maintained as pool
of stack//
   If(top-Null] then
            { write(stack underflow)
               return
            }
  item:= Info[top]
```

```
  ptr:=top
  top:=Link[top]
  Link[ptr]:= Avail
  Avail:=ptr
}
```

Programme is implemented using following two functions:

```
void stack: : push( )
{
node *p;
p=new node( )
cout<<"Enter the number to be pushed:";
cin>>p->no;
p->link=NULL
if (top= =NULL)
{
top=p;
return;
}
   top->link=p
   top=p;
   return;
}
void stack: : Pop( )
{
   node *p ;
if (top= =NULL)
{
cout<<"Underflow";
return;
}
    cout<<top->no
    p=top;
    top=top->link;
    delete p;
    return;
    }
```

➲ **12. a) Given two sorted linked lists L1 and L2, write an algorithm to implement the Merge of two sorted list.**

b) Write an algorithm to insert an element after a node having a given value in a double linked list. [WBSCTE 2015]

Answer:

a) Approach:

Without Recursion:

- Create a new node say result
- Navigate through both the linked lists at the same time, starting from head
- Compare the first node values of both the linked lists
- Whichever is smaller, add it to the result node
- Move the head pointer of the linked list whose value was smaller
- Again compare the node values
- Keep doing until one list gets over
- Copy the rest of the nodes of unfinished list to the result

With Recursion:

- Base Case:
  If List A gets finished, return List B.
  If List B gets finished, return List A.
- Create a result node and initialize it as NULL
- Check which node (List A or List B) has a smaller value.
- Whichever is smaller, add it to the result node.
- Make recursive call and add the return node as result.next

b) Adding a Node at the middle of the list.

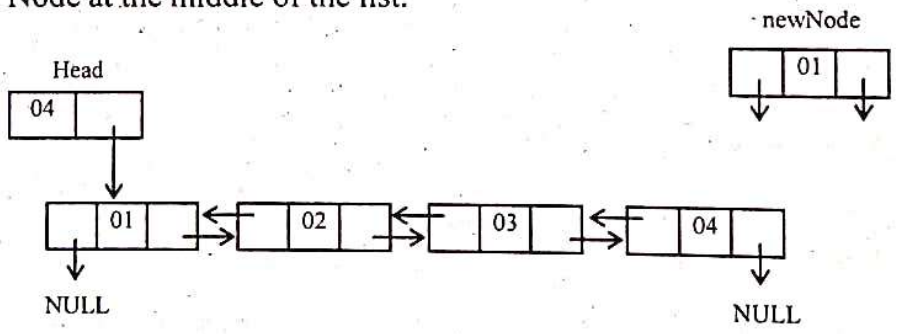


Fig: Current preview of the Doubly Linked List

Let us add a newNode at Location 3.

Traverse the Doubly Linked List until the Location 2. So our curPtr is pointing to Node 2. In newNode – PREV points to NODE 2 (newNode->PREV = curPtr) and NEXT points to NODE 3 (newNode->NEXT = curPtr->NEXT).

In NODE3 – PREV points to newNode (newNode->NEXT->PREV = newNode) and NEXT remains unchanged.

In NODE2 – NEXT points to newNode (curPtr->NEXT = newNode) and PREV remains unchanged.

Increment the LENGTH variable in HEAD to maintain the count of number of Nodes in the List.
Here 'curPtr' points to current pointer.

So after addition the output will be as follows:

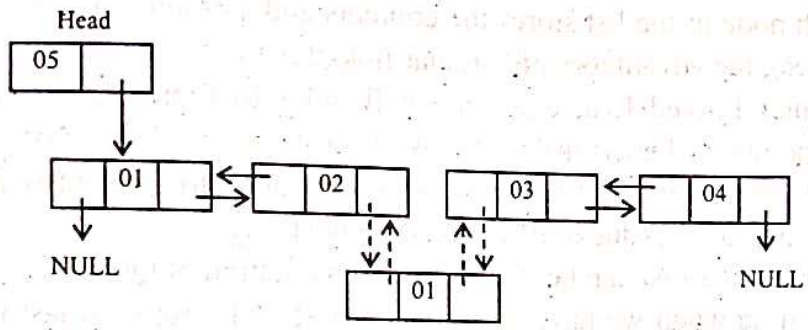


Fig: After adding newNode at Location 3 (Changes in Dotted)

13. Write down a function to insert a node at the beginning of a linked list. [WBSCTE 2016]

Answer:

Insert a node at the beginning of the list:

```
void insertNodeAtBeginning(int data)
{
    struct node *newNode;
    newNode = (struct node*)malloc(sizeof(struct node));
    if(newNode == NULL)
    {
        printf("Unable to allocate memory.");
    }
    else
    {
        newNode->data = data; //Links the data part
        newNode->next = head; //Links the address part
        head = newNode; //Makes newNode as first node
        printf("DATA INSERTED SUCCESSFULLY\n");
    }
}
```

14. (a) What do you mean by circular linked list? What is the advantage of it over simple linked list?
(b) Write down the algorithm to insert a node at a specific position in the linked list.
(c) Write down a function to delete a node from a specific position in a linked list? [WBSCTE 2016]

Answer:

a) 1st part: ***Refer to 3.3 of Unit at a Glance.***

2nd Part:

Linked list, each node in the list stores the contents and a pointer or reference to the next node in the list. So, the advantages of Circular linked list are:

1. In Circular Linked List, end node will point to first Node (doesn't contain a NULL pointer) whereas in singly linked list it won't point to first Node.
2. Circular list is very useful in case of Game play, to give turns for each player without any failure (due to its circular connectivity).
3. Circular Linked List can be used for implementation of Queue.
4. It saves time when we have to go to the first node from the last node. It can be done in single step because there is no need to traverse the in between nodes.

b)

algorithm:

step 1: if avail = Null
Write overflow
go to step 12
[end of if]
Step 2: Set new_ node = avail
Step 3: Set avail = avail → Next
Step 4: Set new_node → data = val
Step 5: Set ptr = start
Step 6: Set preptr = ptr
Step 7: Repeat step 8 and 9 while preptr → data 1! = num
Step 8: Set preptr = ptr
Step 9: Set ptr = ptr → Next
[End of loop]
Step 10: preptr → Next = New _ node
Step 11: Set new_Node → Next = ptr
Step 12: Exit

c)

```
/* Given a reference (pointer to pointer) to the head of a list
   and a key, deletes the first occurrence of key in linked list
*/
void deleteNode(struct Node **head_ref, int key)
{
    // Store head node
    struct Node* temp = *head_ref, *prev;
    // If head node itself holds the key to be deleted
```

```
    if (temp != NULL && temp->data == key)
    {
        *head_ref = temp->next;    // Changed head
        free(temp);                // free old head
        return;
    }
     // Search for the key to be deleted, keep track of the
    // previous node as we need to change 'prev->next'
    while (temp != NULL && temp->data != key)
    {
        prev = temp;
        temp = temp->next;
    }
    // If key was not present in linked list
    if (temp == NULL) return;
    // Unlink the node from linked list
    prev->next = temp->next;
    free(temp);  // Free memory
}
```

➲ **15. Write algorithms to perform the following operations on a linear singly linked list.**

a) Append an item

b) Display the list

What are the disadvantages of linked list? **[WBSCTE 2017]**

Answer:

(I) Algorithm to insert at beginning and delete at beginning

a)

```
void push(struct Node** start, int new_data)
{
    /* 1. allocate node */
    struct Node* new_node = (struct Node*) malloc(sizeof(struct
Node));
    /* 2. put in the data  */
    new_node->data  = new_data;
    /* 3. Make next of new node as head */
    new_node->next = (*start);
    /* 4. move the head to point to the new node */
    (*start)    = new_node;
}
```

b)

Step 1: If START= NULL then

Write "Linked List is Empty"

Step 2: If START->NEXT = NULL then

Return START->INFO
START =NULL
Else
Return START ->INFO
START = START ->NEXT

Step 3: Exit

(II) Few disadvantages of linked lists are:

1) They use more memory than arrays because of the storage used by their pointers.
2) Difficulties arise in linked lists when it comes to reverse traversing. For instance, singly linked lists are cumbersome to navigate backwards and while doubly linked lists are somewhat easier to read, memory is wasted in allocating space for a back-pointer.
3) Nodes in a linked list must be read in order from the beginning as linked lists are inherently sequential access.
4) Nodes are stored incontiguously, greatly increasing the time required to access individual elements within the list, especially with a CPU cache.

➲ **16. Write algorithms to perform the following operations on a linear singly linked list –**
(a) add an item after a given node (b) count number of nodes.

[WBSCTE 2018, 2021]

Answer:

a) Given a node prev_node, insert a new node after the given prev_node

ALGORITHM:

1. Check if the given prev_node is NULL
2. If not NULL then allocate new node (new_node)
3. Put in the data for that new_node
4. Make next of new_node as next of prev_node
5. Move the next of prev_node as new_node

b) Let: 'phead' is a head pointer pointing to the first node, 'last' is a pointer to Single Node and 'count' is an integer variable initially set to zero.

ALGORITHM:

1. Check if the list is empty or not: if phead ==NULL then OUTPUT 'Zero nodes' and exit else go to step 2
2. Set the head pointer to a temporary variable: last = phead
3. Traverse till the last node:

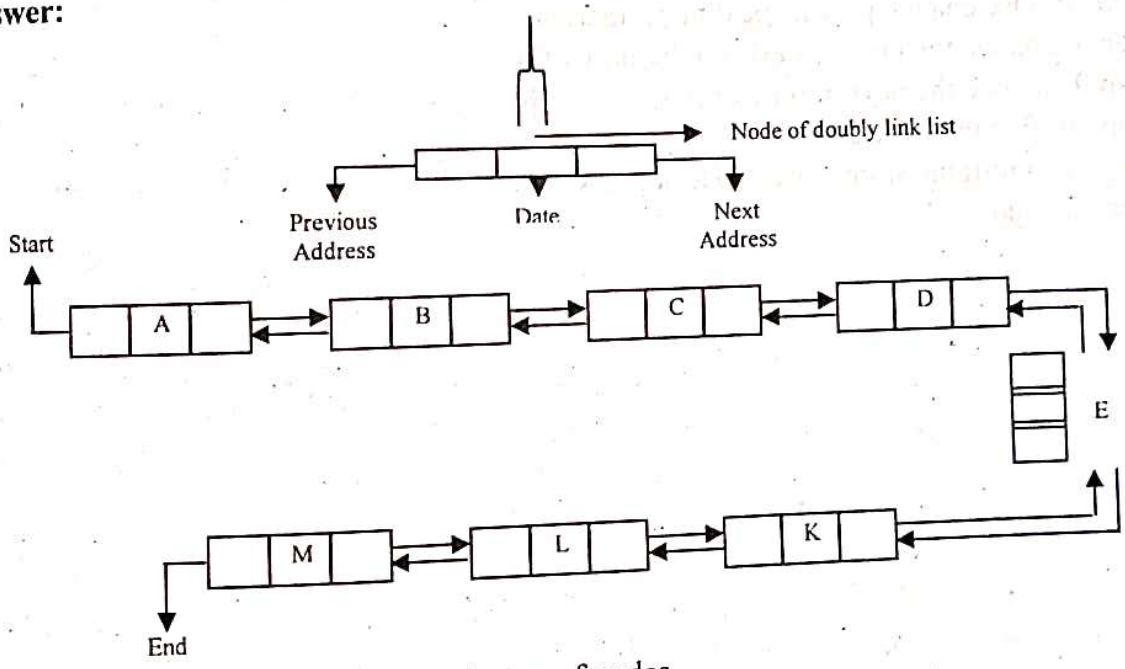
SET while (last->next! = NULL)
{

Increase the node count by 1
SET count++
Point to the next node in the list
SET last=last->next
}

4. Increase the value of count by 1 for last node if the list has more than one node otherwise this condition is applicable if the list has exactly one node.
5. Display the total count of nodes: OUTPUT count

➲ **17. (a) Construct a doubly linked list (only diagram) with following data items.**
A, B, C, D, E, K, L, M
(b) Write an algorithm to traverse a doubly linked list of a *n* data item.
(c)Write a function to delete any node from a doubly linked list.
(d) Write the difference between circular queue and doubly linked list.
[Model Question]

Answer:
a)



b) Let, head = Header node and n is the no. of nodes

Step 1. Start
Step 2. temp = head
Step 3. $i \leftarrow 0$;
Step 4. While $(i < n)$, repeat step 5 – 6
Step 5. if (temp $\rightarrow$ data = x)
print 'Data is present'
break
Step 6. $i \leftarrow i+1$;

Step 7. if *(i = n)*

print (Data is not presut)

Step 8. Stop.

c) The algorithm for deleting a node in the above manner is given below:

Let us assume that node containing data value x is to be deleted. p initially contains the address of the head node.

Step 1: store the address of the head node then start traversing the linked list from the head node

Step 2: if the data value of the head node is not equal to x goto step 7

Step 3: make the next node of head as new head node

Step 4: make the prev of the new head node as NULL

Step 5: free the previous head node

Step 6: return the address of the new head node & Goto Step 12

Step 7: Identify the address of the node containing data value x as follows

Step 7a) if the data value of p is equal to x

Step 7b) else change p by its next node address

Step 8: change the prev of next of p by prev of p

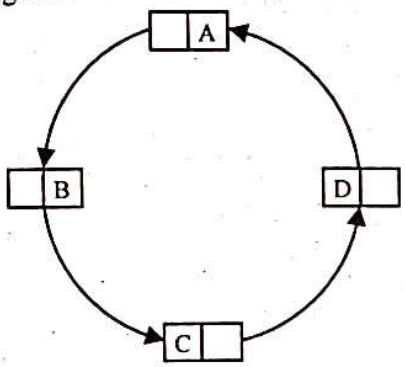
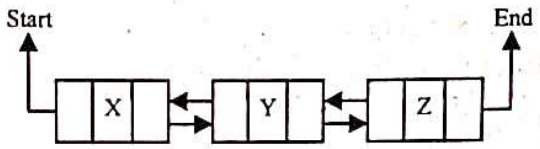
Step 9: change the next of prev of p by next of p

Step 10: free the node p

Step 11: return the stored head node address

Step 12: End

d)

Circular Queue	Doubly Link List
1. This is the list having the last node pointing the first node. 2. Circular link list can be traversed from any node of the list means any node can be accessed from other node. 3. Here each node has two parts: (i) Data (ii) Next address. 4. Diagram: 	1. Doubly link list has two distinct ends (i) start (ii) end. 2. A doubly linked list can be traversed in both directions (forward and backward). 3. Here each node has three parts: (i) Data (ii) Previous address (iii) Next address. 4. Diagram: 

➲ **18. What do you mean by dynamic data structure? Write advantages and disadvantages of linked list over array.** **[Model Question]**

Answer:

Dynamic data structure: A data structure whose organizational characteristics may change during its lifetime. The adaptability afforded by such structures, e.g. linked lists, is often at the expense of decreased efficiency in accessing elements of the structure.

Linked Lists versus Arrays

Similar data items can be stored in memory with the use of array of a linked list.

Arrays are simple data structures that are easy to understand but have the following limitations:

- The size of arrays cannot be increased or decreased during execution. They have a fixed dimension. The elements in an array are stored in contiguous memory locations, but in many cases it may be possible that the contiguous memory space is not available.
- The operations like insertion of a new element in an array or deletion of an existing element after the specified position is tedious.
- Linked list can be used to overcome all these disadvantages.

19. Describe circular doubly linked list data structure write procedures to insert and delete an element from such a list. [Model Question]

Answer:

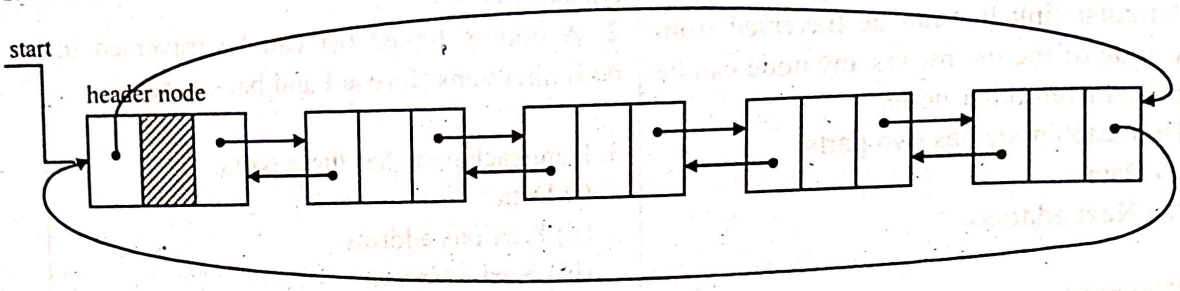


Fig: Double circular linked list with header node

Data Structure:

Structure of the node, link_node

1. Left
2. Right
3. Data

End link_node definition

Left Right Data

Algorithm:

To insert node into the circular doubly link list.

INSERT_CIRCULAR_LIST(START, item, search_item)

1. If (START = NULL) then
   - (i) Print "list is empty"
   - (ii) Scan the user comment in Ch as either 'Y' or 'N'
   - (iii) If (Ch = 'Y') then
     - START = Creat_node ( )
     - START.left = START
     - START.right = START
     - Return

   End of if structure

   End of if structure
2. P = START
3. IF (START.Data = search_item) then
   - P = Create_node ( );
   - P.Data = item
   - P. left = START
   - P.right = START.right
   - START.right = P
   - START.left = P
   - Return

   End of if structure.

4. P = START.right
5. if (P = START) then
   Print "no match found"
   Return
   End of if structure
6. While P START
   If (P.DATA= Search_item) then
   q = Creat_node ( )
   q.Data = item.
   q.left = P
   q.right = P.right
   P.right.left = q
   P.right = q
   Returns
   End of if structure.
   End of Repeat while
7. Print "position not found to insert
8. end.

DEL_CIRCULAR_LIST (START.item)

Algorithm:

To delete node into the circular doubly link list.

DEL_CIRCULAR_LIST (START.item)

1. if (START.left = START) then
   If (START.left = item) then
   START.left = Null
   START.right = Null
   Return
   End of if structure
   Print "Match not found to delete"
   Return
   End of if structure.
2. P = Start
3. While P.right START
   if (P.Data = item) then
   P.right = P.right.right
   P.right.left = P
   Return
   End of if structure
   P = P.Right
   End of while loop.
4. if (P.Data = item) then,
   P.right = P.right.right.right
   P.right.left = P

Return
End of if structure
5. Print "item not match for delete"
6. End.

Unit 4: Non Linear Data Structure

Unit at a glance:

4.1 Tree

A tree is a finite set of one or more nodes connected by edges such that
i) There is a specially designated node called the root
ii) The remaining nodes are partitioned into n (>=0) mutually exclusive disjoint sets

4.2 Binary Tree

A binary tree is either empty or consists of one single vertex called the root, together with two disjoint binary trees called left subtree & right subtree. Number of branches of any node is either zero or one or at most two.

4.3 Different Ways of Traversing a Binary Tree

There are three ways of traversing a binary tree.
i) Inorder ii) Preorder iii) Postorder.

Inorder:

a) The leftmost child is visited first
b) The parent is visited the next.
c) Lastly, the right child is visited.

In short we can say LPR (left, parent, right).

Preorder:

a) The parent is visited first.
b) Then its subsequent left child (if any)
c) Then the right child (if any)

In short we can say PLR (parent, left, right).

Postorder:

a) The leftmost child is visited first (if any).
b) The corresponding rightmost child is then visited.
c) The parent of the two children is visited at last.

In short we can say LRP (left, right, parent).

4.4 AVL Tree

A binary search tree structure that is balanced with respect to the heights of subtrees is called an AVL tree. As a result of the balanced nature of this type of tree, dynamic retrievals can performed in O(log n) times if the tree has n nodes in it.
If T is a non-empty binary search tree with T_L & T_R as its left & right subtrees respectively, then T is height balanced if

i) T_L & T_R are height balanced and
ii) $|h_L - h_R| <= 1$, where h_L & h_R are the heights of T_L & T_R respectively.

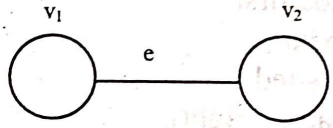
The Balanced Factor, BF (T), of a node T in a binary search tree is defined to be $h_L - h_R$ i.e. the height of its left sub tree minus the height of its right sub tree.

4.5 B Tree

The concept of B-Trees is applied in case of external information retrieval, where we wish to locate and retrieve records stored in a disk file. In case of internal retrieval, all data are kept in high-speed memory. The time required to access and retrieve a word from high speed memory is a few microseconds at most. The time required to locate a particular record on a disk is measured in milliseconds. When a record is located on a disk, the normal practice is not to read only one word, but to read in a large page or block of information at once. Our goal in external searching must be to minimize the number of disk accesses, since each access takes so long compared to internal computation. Hence multiway trees or m-way trees are especially appropriate for external searching.

4.6 Graph

A graph is a mathematical tool used to represent a physical problem. It is also used to model networks, data structures, scheduling, computation and a variety of other systems where the relationship between the objects in the system plays a dominant role. Mathematically a graph consists of two sets V and E where, V is a finite, non-empty set of vertices and E is a set of pairs of vertices. The line joining a pair of vertices is called an edge.



Graph (G)

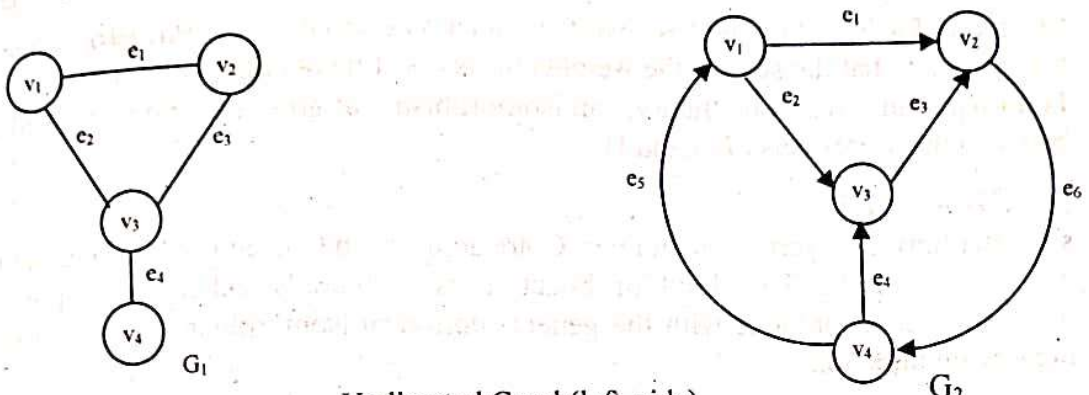
The above figure represents a graph with v_1 , v_2 as the set of vertices and e , an edge, joining the two vertices v_1 and v_2 . v_1 , v_2 and e together form graph . The notation v1 - v_2 means that the edge e has v1 and v2 as end points. In other words e connects vertices v1 and v2 and that v1 and v2 are adjacent.

4.7 Types of Graph

A graph often symbolized as G can be of two types:

- **Undirected graph:** where a pair of vertices representing an edge is unordered.
- **Directed graph:** where a pair of vertices representing an edge is ordered.

The figures shown below represent both types.



Undirected Graph(left side)
Directed Graph(right side)

4.8 Some important definitions

- **Weighted/ Un-weighted Graph:** A *weighted graph* is a graph for which each edge has an associated *weight*, usually given by a *weight function w:* $E \rightarrow R$
- An *un-weighted graph* is a graph for which each edge has no associated *weight*.
- **Sub-graph:** A sub graph of G=G(V,E) is a subset W of the vertex set V together with all of the edges that connect pairs of vertices in W.
- **Degree:** ***Degree*** of a vertex in an undirected graph is the number of edges incident on it. In a directed graph, the ***out degree*** of a vertex is the number of edges leaving it and the ***in degree*** is the number of edges entering it.
- **Cut Vertex/ Articulation Point:** In graph theory, a bi-connected component (or 2-connected component) is a maximal bi-connected sub-graph. Any connected graph decomposes into a tree of bi-connected components called the block tree of the graph. The blocks are attached to each other at shared vertices called cut vertices or articulation points.
- **Pendant node:** A leaf vertex is also known pendent vertex, is a vertex with degree one.
- **Clique:** A clique in an undirected graph is a subset of its vertices such that every two vertices in the subset are connected by an edge.
- **Complete Graph:** A complete graph is an undirected/directed graph in which every pair of vertices is adjacent. If (u, v) is an edge in a graph G, we say that vertex v is adjacent to vertex u.
- **Connected Components:** A connected component of an undirected graph is a sub graph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices.
- **Strongly Connected Component:** A directed graph is called strongly connected if there is a path from each vertex in the graph to every other vertex.
- **Weakly Connected Component:** A weakly connected graph can be thought of as a digraph in which every vertex is "reachable" from every other but not necessarily following the directions of the arcs.
- **Path:** A simple path is a walk with no repeated nodes

- **Shortest Path:** The shortest path is a path between two vertices(or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.
- **Isomorphism:** n graph theory, an isomorphism of graphs G and H is a bijection between the vertex sets of G and H

$f: V(G) \rightarrow V(H)$

- such that any two vertices u and v of G are adjacent in G if and only if f(u) and f(v) are adjacent in H. This kind of bijection is commonly called "edge-preserving bijection", in accordance with the general notion of isomorphism being a structure-preserving bijection.

4.9 Graph Traversals

A graph can be traversed in two ways:

- ***Depth first search traversal:*** often analogous to preorder traversal of an ordered tree.
- ***Breadth first search traversal:*** often analogous to level-by-level traversal of an ordered tree.

Apart from the two types of graph i.e. the undirected type and the directed type, a graph can also have other forms.

Empty graph: A graph G with no edges is called an empty graph.

Null graph: A graph G with no vertices (and hence no edge) is called a null graph.

Connected graph: A graph G is connected if there exists a path between every pair of vertices. In other words, we can also say that if we can traverse every other node from one node by means of any traversal algorithm, then also we can say that the graph is connected.

Simple graph: **A** graph G is simple if it has no parallel edge or self loops.

4.10 Spanning Tree

- Given a connected, undirected graph, a spanning tree of that graph is a sub graph which is a tree and connects all the vertices together. A single graph can have many different spanning trees. We can also assign a *weight* to each edge, which is a number representing how unfavorable it is, and use this to assign a weight to a spanning tree by computing the sum of the weights of the edges in that spanning tree.
- A **minimum spanning tree** (MST) or **minimum weight spanning tree** is then a spanning tree with weight less than or equal to the weight of every other spanning tree. More generally, any undirected graph (not necessarily connected) has a **minimum spanning forest**, which is a union of minimum spanning trees for its connected components.

4.11 Shortest path

- In graph theory, the shortest path problem is the problem of finding a path between two vertices in a weighted graph 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 cost needed to travel that segment.

- There are several variations according to whether the given graph is undirected, directed, or others. For undirected graphs, the shortest path problem can be formally defined as follows. Given a weighted graph (that is, a set V of vertices, a set E of edges, and a real-valued weight function $f: E \rightarrow R$), and elements v and v' of V, find a path P from v to a v' of V so that

$$\sum_{p \in P} f(p)$$

is minimal among all paths connecting v to v' .

The problem is also sometimes called the single-pair shortest path problem. There are several other variations of this problem.

- **The single-source shortest path problem:** In this problem we have to find shortest paths from a source vertex v to all other vertices in the graph.
- **The single-destination shortest path problem**: Here we have to find shortest paths from all vertices in the directed graph to a single destination vertex v. This can be reduced to the single-source shortest path problem by reversing the arcs in the directed graph.
- **The all-pairs shortest path problem**: In this problem, we have to find shortest paths between every pair of vertices v, v' in the graph.

The most important algorithms for solving this problem are:

- **Dijkstra's algorithm** solves the single-source shortest path problems.
- **Bellman-Ford algorithm** solves the single source problem if edge weights may be negative.
- **A* search algorithm** solves for single pair shortest path using heuristics to try to speed up the search.
- **Floyd-Warshall** algorithm solves all pairs shortest paths.
- **Johnson's algorithm** solves all pairs shortest paths, and may be faster than Floyd-Warshall on sparse graphs.

Perturbation theory finds (at worst) the locally shortest path.

Short Answer Type Questions

A. Choose the correct answer from the given alternatives in each of the following:

1. The adjacency matrix of an undirected graph is **[WBSCTE 2003, 2008, 2010]**
   (a) Unit matrix
   (b) Asymmetric matrix
   (c) Symmetric matrix
   (d) None of these

Answer: (c)

2. Adjacency matrix of a digraph is [WBSCTE 2007, 2012]
(a) identity (b) symmetric (c) asymmetric (d) none of these

Answer: (b)

3. Maximum number of edges in a n-node undirected graph without self loop is [WBSCTE 2010]

(a) n^2 (b) $\frac{n(n-1)}{2}$ (c) $n-2$ (d) $\frac{(n+1)(n)}{2}$

Answer: (b)

4. BFS constructs [WBSCTE 2010]
(a) a minimal cost spanning tree of a graph (b) a depth first spanning tree of a graph
(c) a breath first spanning tree of a graph (d) none of these

Answer: (a)

5. A complete directed graph of 5 nodes has..... [WBSCTE 2011]
(a) 5 (b) 10 (c) 20 (d) 25

Answer: (b)

6. A vertex with degree one in a graph is called [WBSCTE 2012]
(a) leaf (b) pendant vertex (c) end vertex (d) none of these

Answer: (b)

7. The memory use of an adjacency matrix is [WBSCTE 2017, 2021]
(a) O(n) (b) $O(n^2)$ (c) $O(n^3)$ (d) O(log n)

Answer: (b)

8. 8. Degree of a leaf node of a binary tree is [WBSCTE 2017, 2021]
(a) 0 (b) 1 (c) 2 (d) 3

Answer: (a)

9. 9. Total number of nodes at the n^{th} level of a binary tree can be given as

[WBSCTE 2017, 2021]
(a) 2^n (b) 2n (c) 2^n+1 (d) 2n + 1

Answer: (c)

10. Preorder is nothing but [WBSCTE 2017, 2021]
(a) Depth first search (b) Breadth first search
(c) Topological order (d) Linear order

Answer: (a)

11. A binary tree of height h has at least [WBSCTE 2018]
(a) 1 node (b) h nodes (c) 2h nodes (d) 2h nodes

Answer: (b)

12. The diagonal of the adjacency matrix of a graph without any self loop contains only [WBSCTE 2018]

(a) 1 (b) ∞ (c) 0 (d) –1

Answer: (c)

13. An array consists of n elements. We want to create a heap using the elements. The time complexity of building a heap will be in order of _____. [WBSCTE 2022]

(a) O(n*n*logn) (b) O(n*logn) (c) O(n*n) (d) O(n*logn*logn)

Answer: (b)

B. Fill in the blanks in the following statements:

14. Binary tree can have maximum ***two*** children. [WBSCTE 2008, 2010]

15. The prefix form of the algebraic expression A*B-C/D is ***-*AB/CD.*** [WBSCTE 2008]

16. ***Kruskal's Algorithm*** algorithm can be used to find minimal spanning tree. [WBSCTE 2008]

17. One of the tree traversal technique is ***inorder***. [WBSCTE 2008]

18. Number of leafs in a binary with 10 nodes are ***3***. [WBSCTE 2008]

19. A node in a tree having no child is called ***leaf node***. [WBSCTE 2009]

20. Tree is a ***Non Linear*** data structure. [WBSCTE 2010]

21. In binary search the list of elements should be arranged in sorted order. [WBSCTE 2010]

22. In Depth first search order, the binary tree traversal the root node comes ***first***. [WBSCTE 2010]

23. Number of binary trees can be constructed with 5 nodes are ***135***. [WBSCTE 2010]

24. The postfix form of the expression $-B+\left(\left(B^2-4*A*C\right)^{1/2}\right)\Big/\left(2*A\right)$ is [WBSCTE 2010]

B-B2^4AC**-12/^+2A*/.

25. A tree is ***non linear*** data structure. [WBSCTE 2022]

26. A tree node that has no children is called a ***leaf*** node. [WBSCTE 2022]

27. The height of a binary tree is the maximum number of edges in any root to leaf path. The maximum number of nodes in a binary tree of height h is $2^{h+1}-1$. [WBSCTE 2022]

28. The leaf nodes of a tree has ***smaller*** children. [Model Question]

29. Tree has ***No*** cycle. [Model Question]

30. Tree structure is implemented by ***Dynamic*** data structure. [Model Question]

31. B-tree is used to build ***Hierarchal Directors.*** [Model Question]

32. The height of a complete binary tree with k number of nodes is $\lfloor \log_2 n \rfloor + 1$. [Model Question]

33. Binary tree traversal method that processes the root after processing the left and the right sub-trees is called ***Post order traversal.*** [Model Question]

34. A balanced m-way search tree is called ***B*Tree*** tree. [Model Question]

35. The clrscr() function is kept in ***conio.h*** header file. [Model Question]

C. State whether the following statements are True or False:

36. Left → Node → Right is the inorder traversal of Binary Search Tree. [WBSCTE 2009]
Answer: True

37. A Tree is also a Graph. [WBSCTE 2009]
Answer: True

38. A sub tree is a tree that has the child of a node as its root. [WBSCTE 2009]
Answer: True

39. Maximum number of nodes in a binary tree of height h is 2^h+1. [WBSCTE 2009]
Answer: False

40. Tree is a connected graph with cycle. [WBSCTE 2010]
Answer: False

41. AVL tree is a height balanced tree. [WBSCTE 2010]
Answer: True

42. Definition of complete binary tree and full binary trees are same. [WBSCTE 2010]
Answer: False

43. Binary search is always better than linear search. [WBSCTE 2010]
Answer: True

44. Heap is a complete binary tree. [WBSCTE 2010]
Answer: True

45. Spanning tree is a type of tree that contains repetitive nodes. [WBSCTE 2010]
Answer: False

46. ADT is same as data structure. [WBSCTE 2010]
Answer: False

47. Binary tree consists at most one children. [WBSCTE 2011]
Answer: False

48. Graphs are the example of linear data structure. [WBSCTE 2012]
Answer: False

49. Graphs are the example of linear data structure. [WBSCTE 2013]
Answer: False

50. A complete tree of height 5 has 63 nodes. [Model Question]
Answer: True

51. Reverse polish expression is evaluated without consideration of majority of operation. [Model Question]
Answer: True

52. Every tree can be represented by a binary tree. [Model Question]
Answer: True

53. A binary tree could be constructed form post order traversal sequence. [Model Question]
Answer: False

54. Given the preorder and post order traversal traces of a binary tree, it is possible to reconstruct the tree. [Model Question]
Answer: False

D. Answer the following questions:

55. Convert the following expression in Reverse polish notation to in-fix notation. [WBSCTE 2011]
A B C / D E //
Answer:
Note: The given expression is incomplete. We correct it as A B C / D E ///
So, Now
The in-fix notation of the above Reverse polish notation is, A/ ((B/C) / (D/E))

[WBSCTE 2011]

56. Define ADT.
Answer:
An ADT is a set of operation. A useful tool for specifying the logical properties of a datatype is the abstract data type. ADT refers to the basic mathematical concept that defines the datatype. Eg. Objects such as list, set and graph along their operations can be viewed as ADT's.

57. What will be the maximum number of children in a node of a B-tree of order M? [WBSCTE 2011]

Answer: Node has at most *m* children

58. Define strictly binary tree. [WBSCTE 2011]

Answer:

A **Strictly Binary Tree** is a binary tree where each non leaf has exactly two children.

59. Convert the expression ((A + B) * C – (D – E) ^ (F + G)) to equivalent Prefix and Postfix notations. [WBSCTE 2012, 2019]

Answer:

1. **Prefix Notation:** - * +ABC ^ - DE + FG
2. **Postfix Notation:** AB + C * DE - FG + ^ -

60. How many null branches are there in a binary tree with 20 nodes? [WBSCTE 2012, 2014, 2019]

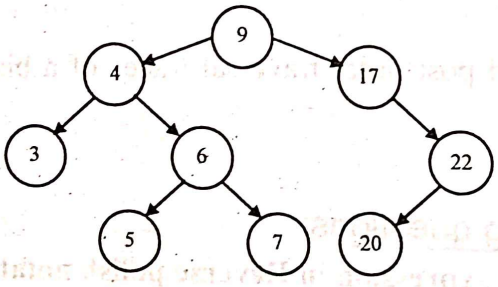
Answer:

A binary tree with n nodes has exactly n+ 1 null node or Null Branches. So answer is 21.

61. Explain the concept of binary search tree. [WBSCTE 2012, 2013, 2014, 2015, 2019]

Answer:

A binary search tree (BST), also known as an ordered binary tree, is a node-based data structure in which each node has no more than two child nodes. Each child must either be a leaf node or the root of another binary search tree. The left sub-tree contains only nodes with keys less than the parent node; the right sub-tree contains only nodes with keys greater than the parent node.



The BST data structure is the basis for a number of highly efficient sorting and searching algorithms, and it can be used to construct more abstract data structures including sets, multisets, and associative arrays.

62. Discuss the advantages of an AVL tree. [WBSCTE 2012, 2019]

Answer:

An *AVL tree* is a self-balancing binary tree. It is similar in concept to a Red-Black tree. AVL tree does not waste much memory.

63. B-trees of order 2 are full binary trees. Justify your answer. [WBSCTE 2012, 2014, 2019]

Answer:
B-tree of order 2 is
2-way search tree,
Not empty => root has at least 2 children.
Remaining internal nodes (if any) have at least ceil(m/2) children.
So, B-tree of order 2 is full binary tree.

64. Does the minimum spanning tree of a graph give the shortest distance between any two specified nodes? [WBSCTE 2013]

Answer:
No. A **minimum spanning tree** (MST) or **minimum weight spanning tree** is then a spanning tree with weight less than or equal to the weight of every other spanning tree. But it doesn't mean that the distance between any two nodes involved in the minimum-spanning tree is minimum.

65. Why large value of M (possible branches of each node) needed in a B-tree? [WBSCTE 2013, 2014, 2019]

Answer:
B trees contain data with each key, frequently accessed nodes can lie closer to the root, and therefore can be accessed more quickly so large value of M (possible branches of each node) needed in a B-tree.

66. What are some of the applications for the tree data structure? [WBSCTE 2014, 2015]

Answer:
The list is as follows:
- The manipulation of Arithmetic expression,
- Symbol Table construction,
- Syntax analysis.

67. In an AVL tree, at what condition the balancing is to be done? [WBSCTE 2014]
Answer: ***Refer to Question No. 6 of Long Answer Type Questions.***

68. In an AVL tree, what are the allowable balancing factor (BF) of a node? [WBSCTE 2015]

Answer: The highest possible balancing factor (BF) a node of an AVL tree can have is 1.

69. What is the suitable data structure to construct tree? [WBSCTE 2015]
Answer: Array is the suitable data structure to construct tree.

70. Give definition of complete binary tree and full binary tree. [WBSCTE 2015]
Answer:
A full binary tree (sometimes proper binary tree or 2-tree) is a tree in which every node other than the leaves has two children. A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

71. How many children can a n-ary tree have? [WBSCTE 2015]
Answer: An n-ary tree is a tree where node can have between 0 and n children.

72. How many binary trees can be constructed with 5 nodes? [WBSCTE 2015]
Answer: Number of binary trees can be constructed with 5 nodes are 135.

73. What is max – heap and min – heap? [WBSCTE 2015]
Answer: A Binary Heap is either Min Heap or Max Heap. In a Min Binary Heap, the key at root must be minimum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree. Max Binary Heap is similar to Min Heap, the key at root must be maximum among all keys.

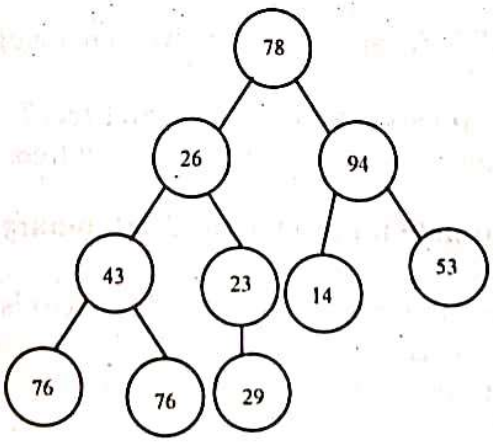
74. Convert the expression ((A*B)*C/(D – E) ∧ (F + G)) to equivalent Prefix and Postfix notations. [WBSCTE 2015]
Answer:
1) **Prefix Notation :** /**ABC ∧-DE+FG
2) **Postfix Notation:** AB*C*DE-FG+∧/

75. Write down the implementing structure of a tree in C. [WBSCTE 2016]
Answer:
Implementing structure of a tree in C:
Ans: Struct node {
int data;
struct node * left;
struct node * right;};

76. Define the term 'Polish notation'. [WBSCTE 2016]
Answer: ***Refer to Question No. 12 of Long Answer Type Questions.***

77. Define tree traversal and its type. [WBSCTE 2016]
Answer: ***Refer to 4.3 of Chapter at a Glance.***

78. Construct a Binary tree with the elements 78, 26, 94, 43, 23, 14, 53, 76, 76, 29 [WBSCTE 2016]
Answer:



79. Define height balanced tree. [WBSCTE 2016]
Answer: *Refer to 4.4 of Chapter at a Glance.*

80. Define BFS. [WBSCTE 2016]
Answer:
Breadth first search is a graph traversal algorithm that starts traversing the graph from root node and explores all the neighboring nodes. Then, it selects the nearest node and explores all the unexplored nodes. The algorithm follows the same process for each of the nearest node until it finds the goal.

81. What is the purpose for AVL trees? [WBSCTE 2017, 2021]
Answer:
The purpose of the AVL tree is to avoid the poorly shaped trees and ensure that the tree is always well balanced so that we are guaranteed of getting good search, insert, and delete times. It is a self-balancing binary search tree.

82. What is a leaf node? [WBSCTE 2022]
Answer:
Those nodes in the tree which don't have any child are known as leaf nodes i.e., A node is a leaf node if both left and right child nodes of it are null.

83. What is BST? [WBSCTE 2022]
Answer:
A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties –

- The value of the key of the left sub-tree is less than the value of its parent (root) node's key.
- The value of the key of the right sub-tree is greater than or equal to the value of its parent (root) node's key.

84. What is a degree of a node of a tree? [WBSCTE 2022]
Answer:
Degree of a node is the total number of children of that node. Degree of a tree is the highest degree of a node among all the nodes in the tree.

85. Write the full form of MST. [WBSCTE 2022]
Answer: Minimum Spanning Tree (MST).

86. What is complete binary tree? [WBSCTE 2022]
Answer:
A complete binary tree is a special type of binary tree where all the levels of the tree are filled completely except the lowest level nodes which are filled from as left as possible.

87. 10, 5, 1, 7, 40, 50 is given preorder traversal of a binary search tree. Find out the post-order traversal of the same tree? [WBSCTE 2022]

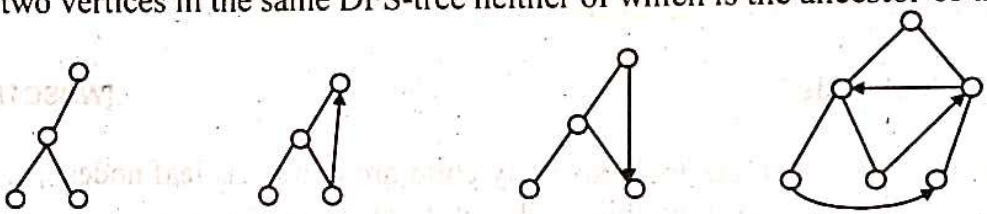
Answer: 1, 7, 5, 50, 40, 10

88. Describe the Edge classification of DFS algorithm. [Model Question]

Answer:

Consider a directed graph G = (V, E). After a DFS of graph G we can put each edge into one of four classes:

1. A ***tree edge*** is an edge in a DFS-tree.
2. A ***back edge*** connects a vertex to an ancestor in a DFS-tree. Note that a self-loop is a back edge.
3. A ***forward edge*** is a non-tree edge that connects a vertex to a descendent in a DFS-tree.
4. A ***cross edge*** is any other edge in graph G. It connects vertices in two different DFS-tree or two vertices in the same DFS-tree neither of which is the ancestor of the other.



1. A Tree Edge | 2. A back edge to an ancestor | 3. A forward edge to a decendant | 4. A cross edge to a different node

Any particular DFS or BFS of a directed or un-directed graph, each edge gets classified as one of the above.

In a DFS of an undirected graph, every edge is either a tree edge or back edge. Here no cross edge goes to a higher numbered or rightward vertex. The reason behind the DFS algorithm is so important is that it defines a very nice ordering of edges of the graph.

Long Answer Type Questions

➲ **1. What is a complete graph. Show that the sum of degree of all the vertices in a graph is always even.** [WBSCTE 2003, 2009]

Answer:

A graph is said to be complete if there exist an edge between every pair of vertices.

At first we have to know, the degree of a vertex. Degree of a vertex is defined as no. of edges incident on a particular vertex with the self loop counted twice. Now, let us take the sum of degree of all the vertices. Now, this summation is an even number, because each edge contributes twice when we are calculate degree of different vertices. Now, if we take the summation of all the degrees that is nothing but the twice the number of edges $\sum_{i=1}^{n} d(v_i) = 2e \Rightarrow$ even number

2. Write a note on Sequential and linked representation of graphs in memory of computer. [WBSCTE 2003]

OR,

Discuss the two different ways of representing a graph. [WBSCTE 2009]

Answer:

There are three ways by which graphs can be represented in memory.

i) Adjacency matrices.
ii) Adjacency list.
iii) Adjacency multilists.

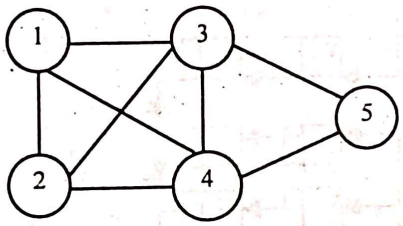
i) ***Adjacency matrices:*** For a graph G with n vertices the adjacency matrix is a 2D array with n x n element.

For undirected graphs

A [i, j] = 1, if there is an edge between i & j
A [i, j] = 0, if there is no edge between i & j.

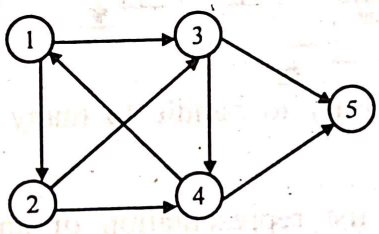
For directed graphs

A [i, j] = 1, if there is an edge directed i & j
A [i, j] = 0, otherwise



Undirected graph

	1	2	3	4	5
1	0	1	1	1	0
2	1	0	1	1	0
3	1	1	0	1	1
4	1	1	1	0	1
5	0	0	1	1	0



Directed graph

	1	2	3	4	5
1	0	1	1	0	0
2	0	0	1	1	0
3	0	0	0	1	1
4	1	0	0	0	1
5	0	0	0	0	0

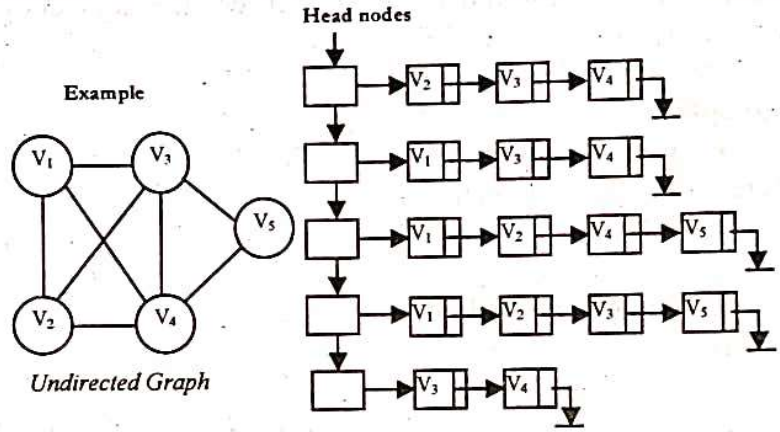
The adjacency matrix for an undirected graph is symmetrical i.e., diagonal elements are zero.

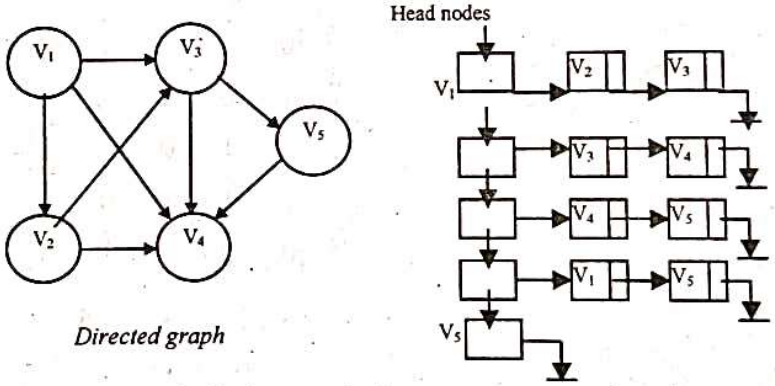
Advantage: It is possible to determine whether an edge is connecting two vertices or not.

Disadvantage:

- It requires n x n memory locations. If the graph is not complete, there will be a number of zero entries & this will waste the memory area. To avoid this, the adjacency list method is used.
- There will be two entries for the same edge. This information is redundant.

ii) ***Adjacency List:*** It is a linked list representation of a graph where each vertex represents a head node. Each head node has a list associated with it which represents the edges. The nodes of the list form a path between the head node and all other reachable vertices in the graph from that head node.





Drawbacks: When a graph is large, it is necessary to handle to many numbers of pointers, which can be complex.

iii) ***Adjacency Multilist:*** From the adjacency list representation of an undirected graph, it is clear that each edge (Vi, Vj) is represented by two entries, one is list Vi and another is list Vj. This is unnecessary wastage of the memory as the information present is same at both the nodes (1,2)= (2,1).

To avoid this, the adjacency multilist is used. Using this representation, the nodes are shared amongst the several lists.

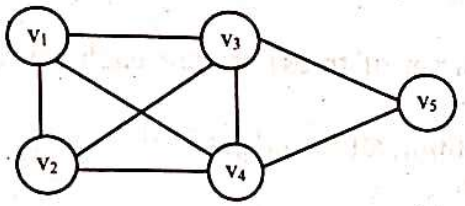
In this method for each edge of the graph, there will be exactly one node. But this node will provide the information about two more nodes to which it is incident.

The node structure with this representation will be:

m	V_1	V_2	P_1	P_2

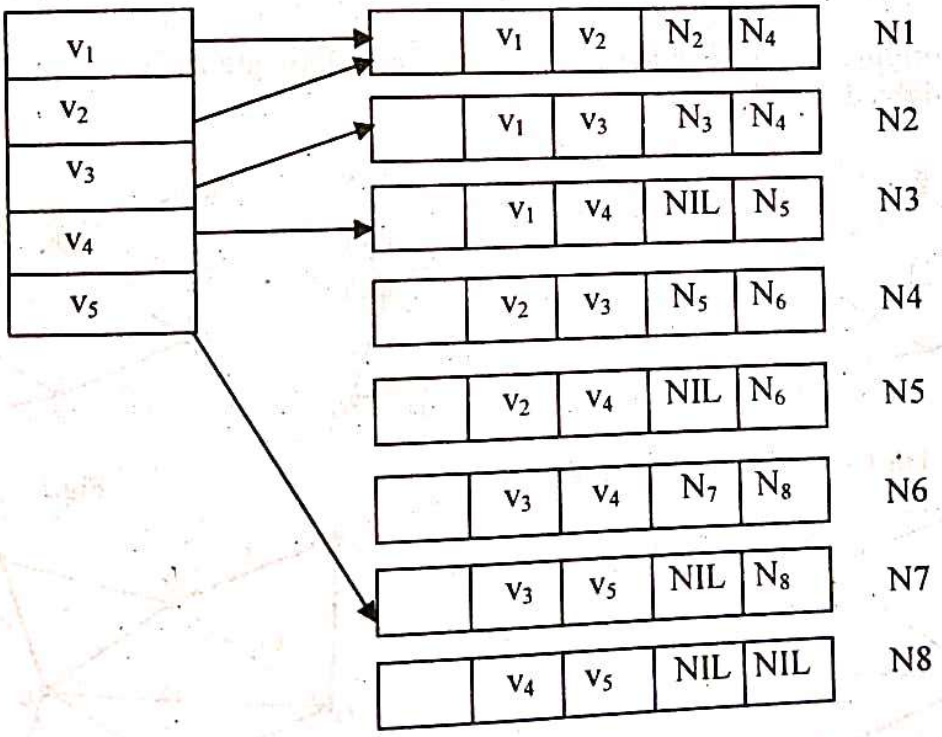
m --- tagfield
V_1, V_2 ---- Vertex
P1, P_2----- Incident paths
m is a one bit mark field that is used to indicate whether or not the edge has been examined.

Consider the following graph.



Total distinct edges are:
$\{v_1, v_2\}$, $\{v_1, v_3\}$, $\{v_1, v_4\}$, $\{v_2, v_3\}$, $\{v_2, v_4\}$, $\{v_3, v_4\}$, $\{v_3, v_5\}$ & $\{v_4, v_5\}$. All other edges represent the same information, because this is an undirected graph.

The adjacency multilist representation is as follows



The lists are: -
Vertex 1 : N1 → N2 → N3
Vertex 2 : N1 → N4 → N5
Vertex 3 : N2→ N4 → N6 → N7
Vertex 4 : N3 → N5 → N6 → N8
Vertex 5 : N7 → N8

3. Define spanning tree. Write a method or algorithm which will generate a minimum cost spanning tree. Explain your method with an example. [WBSCTE 2005, 2006, 2007]

Answer:

Spanning Tree: Given a connected, undirected graph, a spanning tree of that graph is a sub graph, which is a tree and connects all the vertices together. A single graph can have many different spanning trees.

A **minimum spanning tree** (MST) or **minimum weight spanning tree** is then a spanning tree with weight less than or equal to the weight of every other spanning tree.

Kruskal's algorithm:

- create a forest F (a set of trees), where each vertex in the graph is a separate tree
- create a set S containing all the edges in the graph
- while S is nonempty
  - remove an edge with minimum weight from S
  - if that edge connects two different trees, then add it to the forest, combining two trees into a single tree
  - Otherwise discard that edge.
  - End

Example

This is our original graph. The numbers near the arcs indicate their weight. None of the arcs is highlighted.

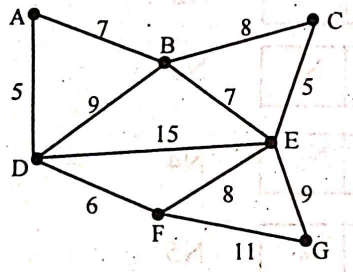


Fig.1

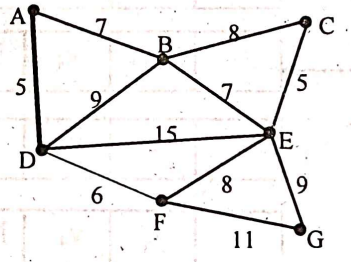


Fig.2

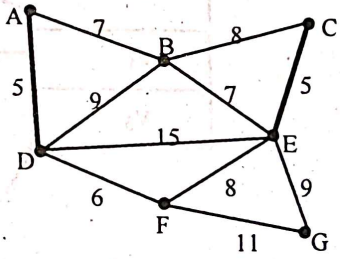


Fig.3

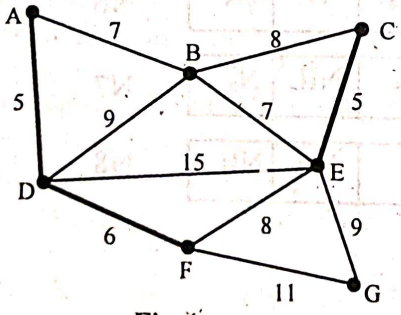


Fig.4

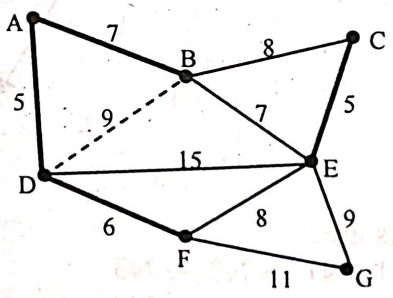


Fig.5

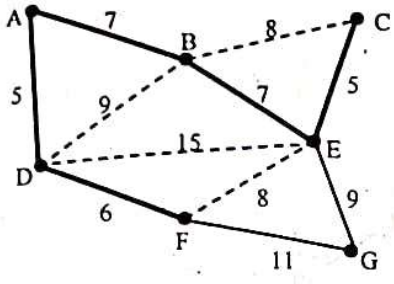
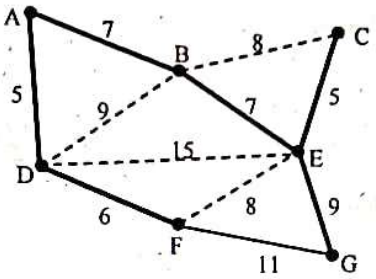


Fig.6



Final figure

➲ **4. What is an AVL tree?** **[WBSCTE 2006, 2011]**
Write its advantages over Binary Search tree. Construct an AVL tree with the following data:

50, 33, 44, 22, 77, 35, 60, 40

Mention the kind of rotation you are applying at different steps. [WBSCTE 2006]

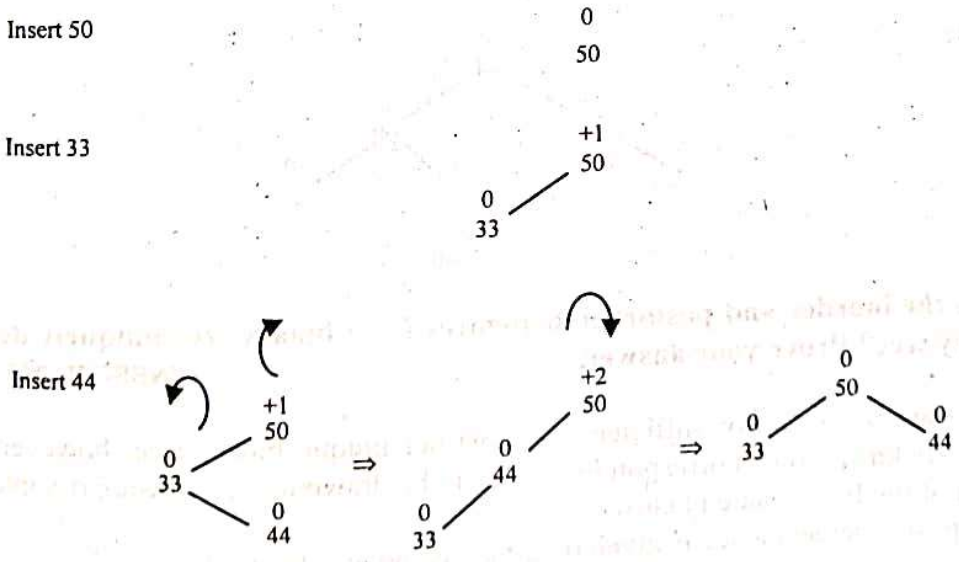
Answer:

AVL Tree:

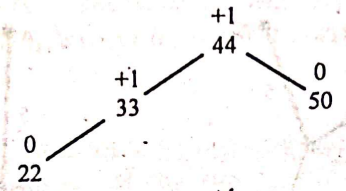
A binary search tree structure that is balanced with respect to the heights of subtrees is called an AVL tree. As a result of the balanced nature of this type of tree, dynamic retrievals can performed in O(log n) times if the tree has n nodes in it.
Each node in a balanced binary tree has a balance factor of 0, +1, -1,

Advantages of AVL Tree over Binary Search Tree:

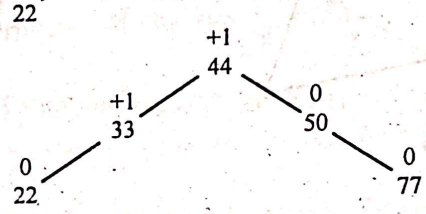
In a complete binary search tree, the height of the left and right sub trees of any node is equal. Such trees are ideal for efficient searching because the height of such a tree with n nodes is O(log n) . But when insertion and deletion are performed randomly on a binary search tree, the tree often turns out to be far from ideal. A close approximation to an ideal binary search tree is achievable through the idea of AVL tree or balanced trees.



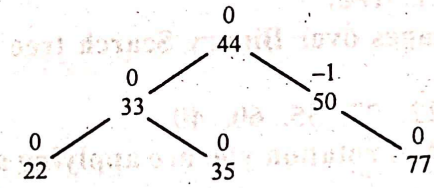
Insert 22



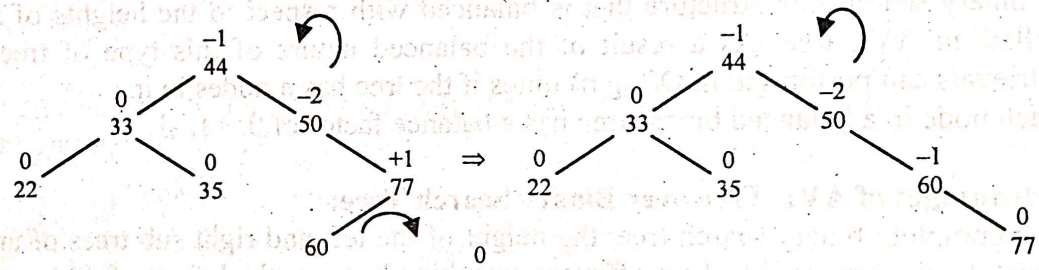
Insert 77

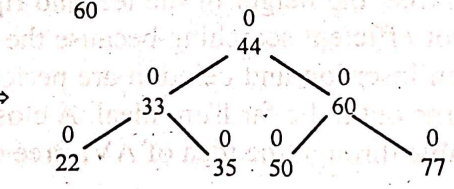


Insert 35

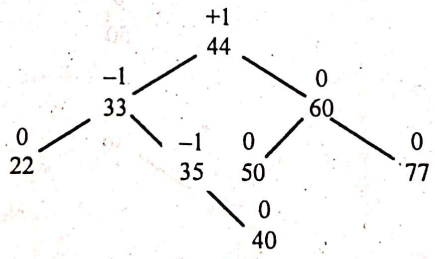


Insert 60





Insert 40



➲ 5. Do the inorder and postorder sequences for a binary tree uniquely define the binary tree? Prove your answer. **[WBSCTE 2008, 2010]**

Answer:

A single traversal it is not sufficient to construct unique binary tree, however, if two traversals are known then corresponding tree can be drawn unique. Basic principle for the formation of the tree is stated below: -

(1) If the post order sequence is given then the last node is the root.

(2) Once the root node is identified, all the nodes in the left sub-trees and right sub-trees of the root node can be identified from the in-order sequence.

Example: Construction of the tree from its given postorder and inorder traversal.

Postorder : D F E B G L J K H C A

Inorder : D B F E A G C L J H K

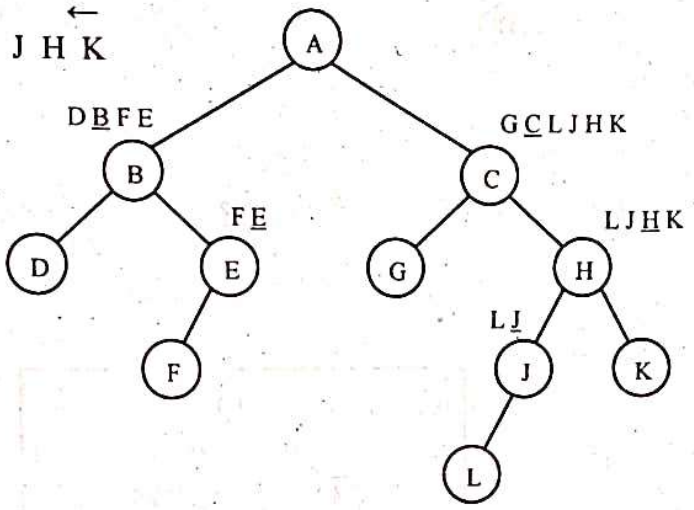


Fig: Construction of a tree from its postorder and inorder traversal

➲ **6. Write brief note on AVL tree.** [WBSCTE 2008, 2009, 2013]

OR,

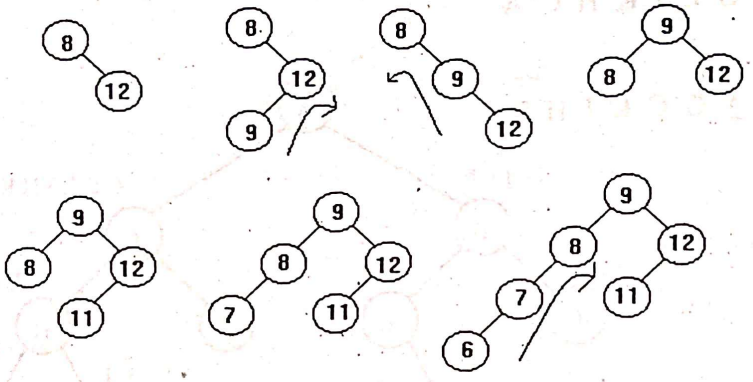
Height Balanced Tree. [WBSCTE 2010]

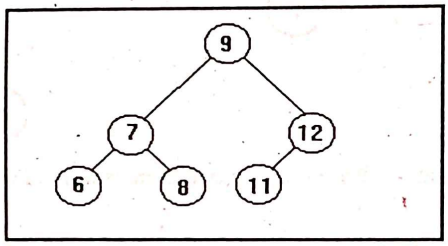
Answer:

The Balanced Factor, BF (T), of a node T in a binary search tree is defined to be $(h_L - h_R)$ i.e. the height of its left sub tree minus the height of its right sub tree

Each node in a balanced binary tree has a balance factor of 0, +1, -1, depending on whether the height of its left sub tree is equal, greater than or less than to the height of its right sub tree .If the balance factor of a node is +1 then the node is called left heavy, if the balance factor of a node is –1 then the node is called right heavy. A node is perfectly balanced if it has a balanced factor of 0.

Example:
8, 12, 9, 11, 7, 6.





The required AVL tree as per question

➲ **7. Describe Kruskal's minimal spanning tree algorithm.** **[WBSCTE 2008]**

Answer:

Kruskal's algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a minimum spanning forest (a minimum spanning tree for each connected component). Kruskal's algorithm is an example of a greedy algorithm.

It works as follows:

- create a forest F (a set of trees), where each vertex in the graph is a separate tree
- create a set S containing all the edges in the graph
- while S is nonempty
  - remove an edge with minimum weight from S
  - if that edge connects two different trees, then add it to the forest, combining two trees into a single tree
  - Otherwise discard that edge.

At the termination of the algorithm, the forest has only one component and forms a minimum spanning tree of the graph.

8. Define minimal spanning tree. Discuss Prim's algorithm with example. [WBSCTE 2009]

Answer:

A **minimal spanning tree** (MST) or **minimal weight spanning tree** is then a spanning tree with weight less than or equal to the weight of every other spanning tree. More generally, any undirected graph (not necessarily connected) has a **minimal spanning forest**, which is a union of minimum spanning trees for its connected components.

Prim's algorithm:

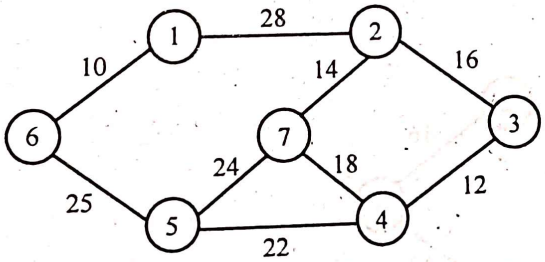
Prim's algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it will only find a minimum spanning tree for one of the connected components. It works as follows:

- create a tree containing a single vertex, chosen arbitrarily from the graph
- create a set containing all the edges in the graph
- loop until every edge in the set connects two vertices in the tree
  - remove from the set an edge with minimum weight that connects a vertex in the tree with a vertex not in the tree
  - add that edge to the tree

Prim's algorithm can be shown to run in time, which is O (m + n log n) where m is the number of edges and n is the number of vertices.

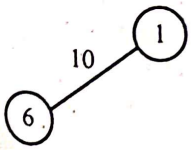
For example,

Construct the minimum spanning tree (MST) for the given graph using Prim's Algorithm-

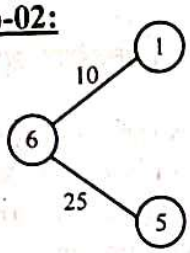


The above discussed steps are followed to find the minimum cost spanning tree using Prim's Algorithm-

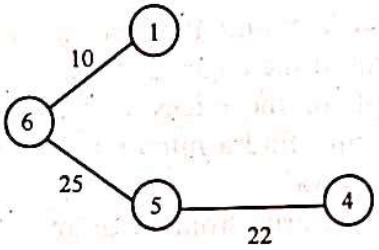
Step-01:



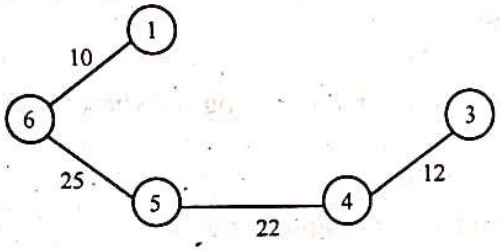
Step-02:



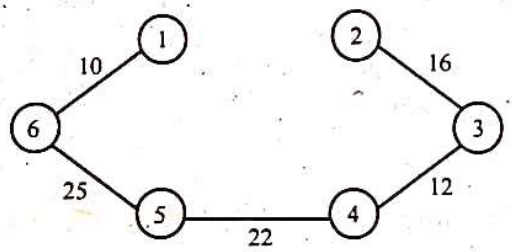
Step-03:



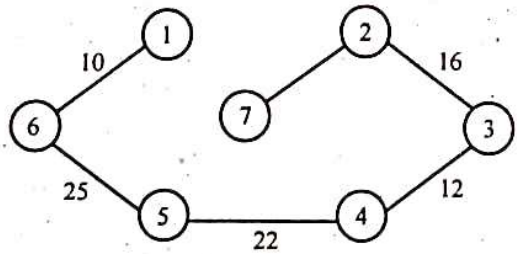
Step-04:



Step-05:



Step-06:



Since all the vertices have been included in the MST, so we stop.
Now, Cost of Minimum Spanning Tree
= Sum of all edge weights
= 10 + 25 + 22 + 12 + 16 + 14
= 99 units

➲ **9. Write an algorithm that takes only a pointer to the root of a binary tree T and computes the number of nodes in T.** **[WBSCTE 2009]**

Answer:
In order to count the number of leaf nodes in a binary tree, we have to traverse it to find out these nodes, which do not have any lchild & rchild.

```
int COUNT(bintree *p)
{
 Let i be a variable and i = 0;
  BINTREE *n[100];
  int leaf_count = 0;
  while (1)
  {
    while (p != NULL)
    {
      n[++i] = p;
      if (p -> lchild == NULL && p -> rchild == NULL)
        leaf_count = leaf_count + 1;
      p = p->lchild;
    }
    if (i ! = 0)
    {
      p = n[i--];
      p = p->rchild;
    }
    else
      break;
  } // end of first while(1)
  return (leaf_count);
}
```

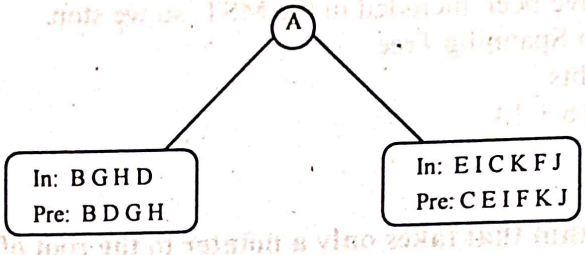
➲ **10. The order of nodes of a binary tree in Preorder and Inorder traversal are as under:**

Preorder: A B D G H C E F I K J
Inorder: B G H D A E I C K F J

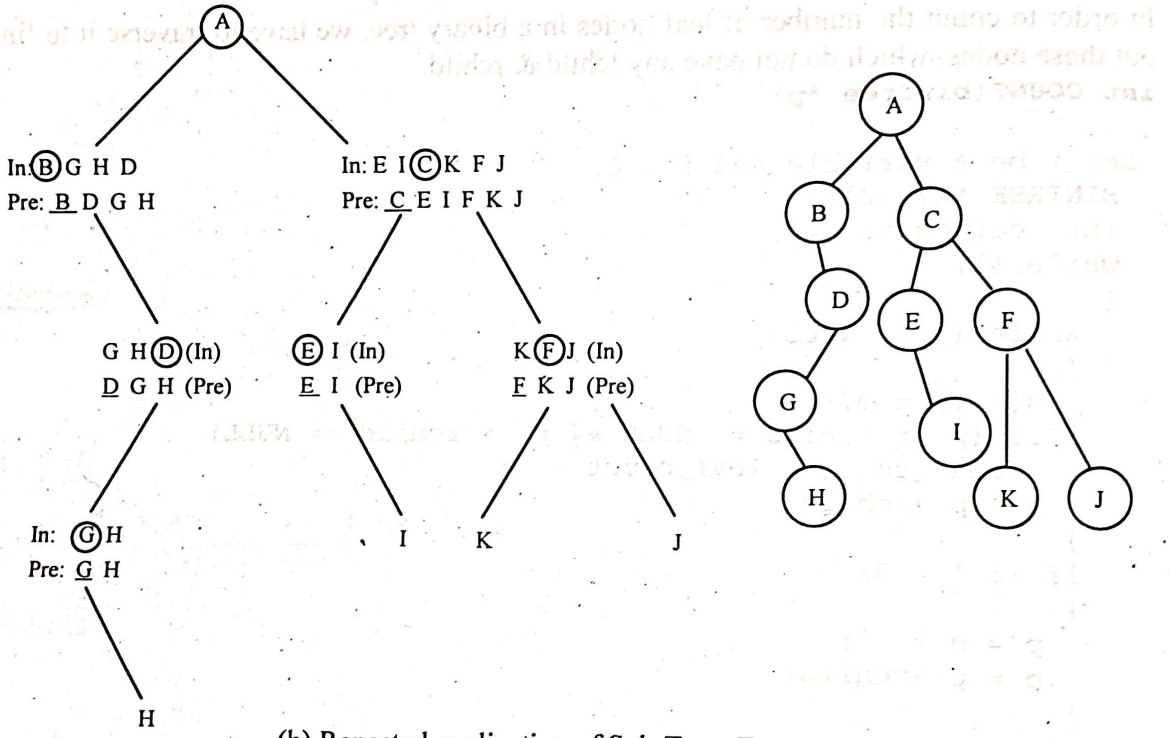
[WBSCTE 2009]

Draw the corresponding binary tree.

Answer:
Basic principle for the formation of the tree is stated below:
(1) If the post order sequence is given then the last node is the root.
(2) Once the root node is identified, all the nodes in the left sub-trees and right sub-trees of the root node can be identified from the in-order sequence.



(a) Two Sub-Trees as A being the root from two traversals



(b) Repeated application of Sub-Trees Formation

➲ 11. Write a note on Tree balancing. **[WBSCTE 2009]**

Answer:

In practice, one can however not always guarantee that binary search trees are built randomly. Binary search trees are thus only interesting if they are "relatively complete." So we must look for specialisations of binary search trees whose worst-case performance on the basic tree operations can be guaranteed to be good, that is $O(\log_2 n)$ time

A balanced tree is a tree where every leaf is "not more than a certain distance" away from the root than any other leaf. The various balancing schemes give actual definitions for "not more than a certain distance" and require different efforts to keep the trees balanced:

- AVL trees
- Red-black trees.

Inserting into, and deleting from, a balanced binary search tree involves transforming the tree if its balancing property — which is to be kept invariant — is violated.

These re-balancing transformations should also take $O(\log_2 n)$ time, so that the effort is worth it.
These transformations are built from operators that are independent from the balancing scheme.

➲ 12. Define Polish notation. [WBSCTE 2010, 2011]

Answer:
Polish notation, also known as prefix notation, is a form of notation for logic, arithmetic, and algebra. Its distinguishing feature is that it places operators to the left of their operands. In simple word, if you are prefixing the operator to the number, it is called Polish notation.

Example,
Polish notation: +22

➲ 13. Define extended binary tree. [WBSCTE 2010]
Derive an expression for the minimum external path length of an extended binary tree with n internal nodes. [WBSCTE 2010, 2011]

Answer:
1st part: Extended binary tree: A binary tree can be converted to complete binary tree by replacing every NULL in the original binary tree with a dummy node. If n is the no. of elements present in the binary tree, then n+1 dummy nodes are required to make it a complete binary tree.

2nd part: The external path length, E, of an extended binary tree to be the sum of the path lengths from the root to each of the square boxes. If an unsuccessful search for a key K occurs in an extended binary tree, T, then the search lands in one of the square boxes of the extended tree. Moreover, if a particular box is at level l in the extended tree, it takes exactly 2*l comparisons to determine that the search is unsuccessful. If the extended tree, T, has n internal nodes, then it has (n+1) boxes as leaves. Therefore, if each of the boxes in T is an equaly likelt target for an unsuccessful search, then the total number of comparisons required for an average unsuccessful search is 2*E/(n+1).

➲ 14. Design an algorithm that find minimal spanning tree in polynomial time. [WBSCTE 2010]

Answer: ***Refer to Question No. 7 of Long Answer Type Questions.***

➲ 15. Write a note on B-tree and its use. [WBSCTE 2010, 2013]

Answer:
B-trees are balanced trees that are optimized for situations when part or the entire tree must be maintained in secondary storage such as a magnetic disk. Since disk accesses are expensive (time consuming) operations, a B-tree tries to minimize the number of disk accesses. Each node of a b-tree may have a variable number of keys and children. The keys are stored in non-decreasing order. Each key has an associated child that is the root of a subtree containing all nodes with keys less than or equal to the key but greater than

the preceding key. A node also has an additional rightmost child that is the root for a subtree containing all keys greater than any keys in the node.
Magnetic disks (secondary memory) are cheaper than RAM and have higher capacity. But they are much slower because they have moving parts. B-trees try to read as much information as possible in every disk access operation.

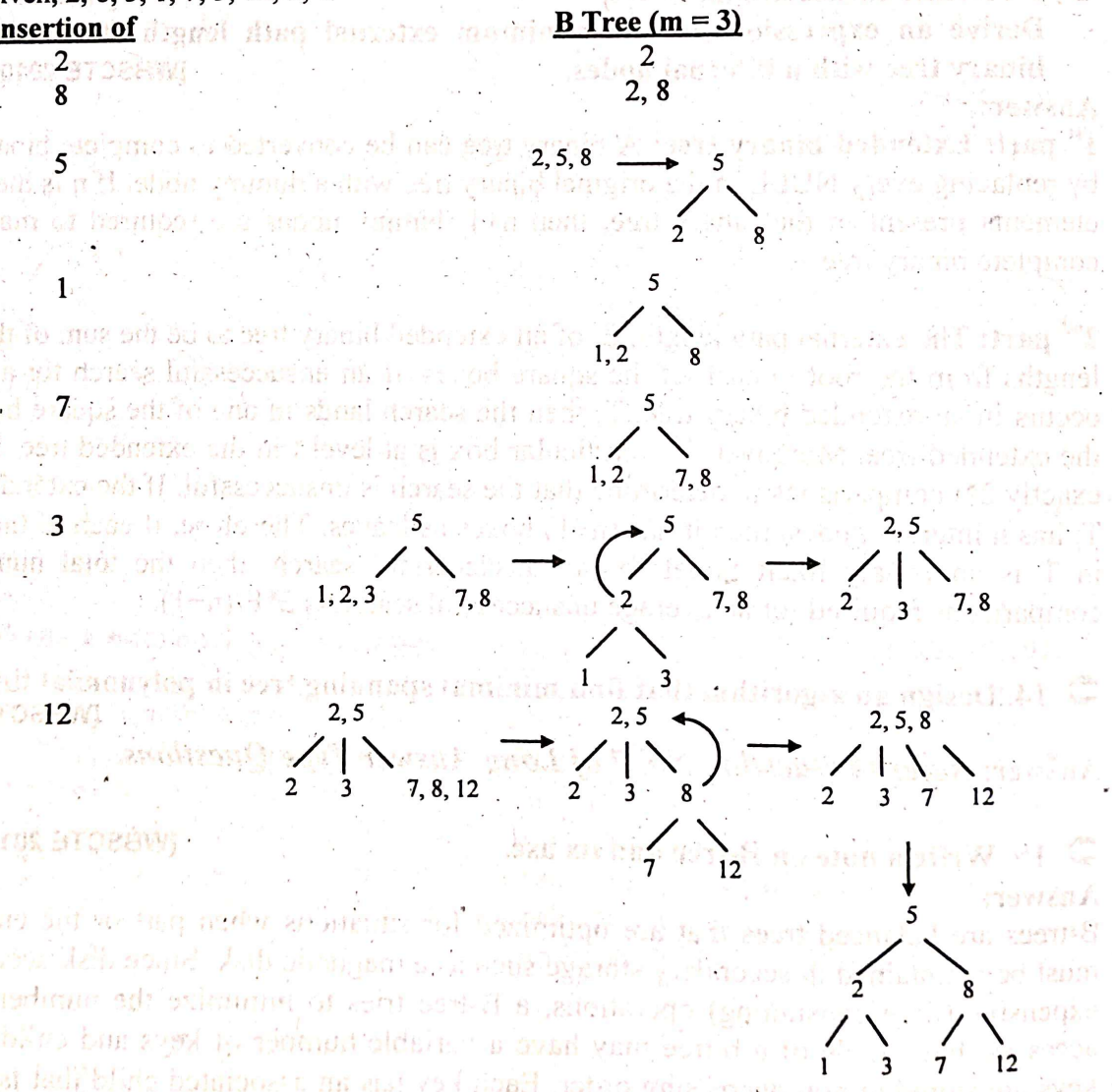
16. Define B-tree of order K that is used to access a key field to search records in a data file. Insert the following records in sequence in B-Tree of order p=3 and P leaf = 2, 8, 5, 1, 7, 3, 12, 9,6 Show each stage on insertion. [WBSCTE 2011]

Answer:

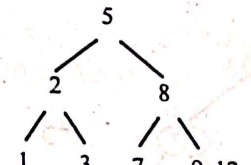
Insertion of B-Tree of order 3

Order m = 3, means each node expect root contains $\left\lceil \frac{m}{2} \right\rceil$ to $(m-1)$ names.

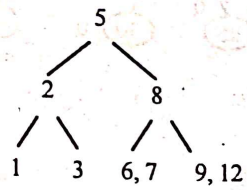
P leaf given, 2, 8, 5, 1, 7, 3, 12, 9, 6



9



6



➲ **17. What is a tree structure? Where is it frequently used?** **[WBSCTE 2011]**

Answer:

A tree is a finite set of one or more nodes connected by edges such that

i) There is a specially designated node called the root

ii) The remaining nodes are partitioned into n (>=0) mutually exclusive disjoint sets

Tree is frequently used for searching of non linear data structure.

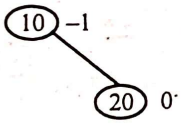
➲ **18. Construct an AVL tree with the input given below:**

10, 20, 30, 25, 27, 7, 4, 23, 26, 21 **[WBSCTE 2012, 2019]**

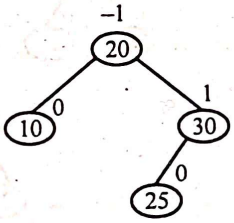
Answer:

Insert 10

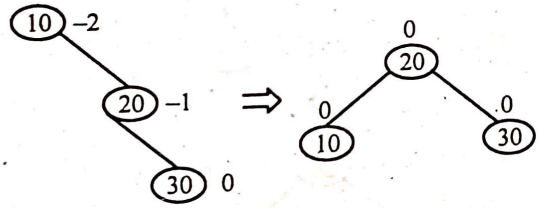
Insert 20



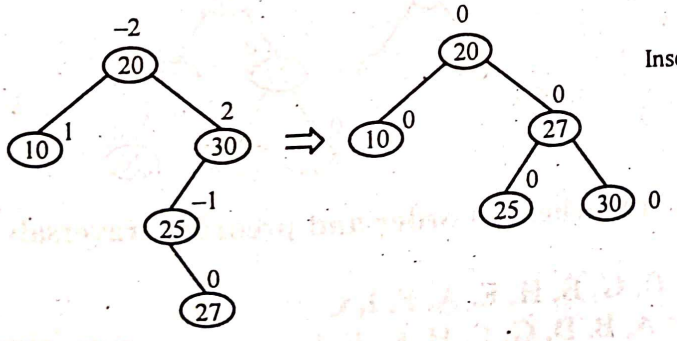
Insert 25



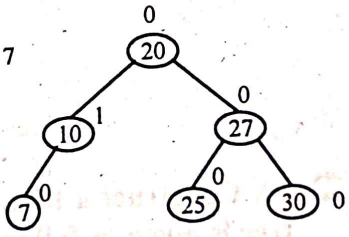
Insert 30



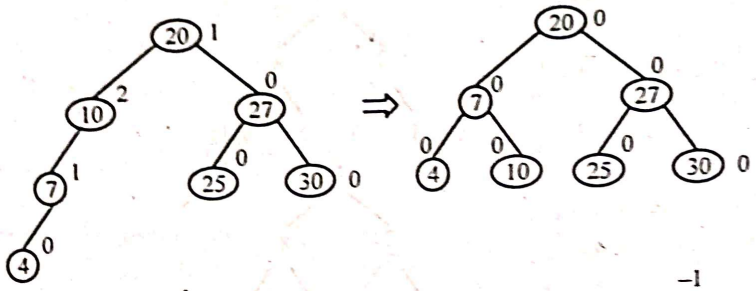
Insert 27



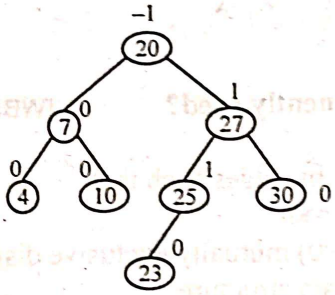
Insert 7



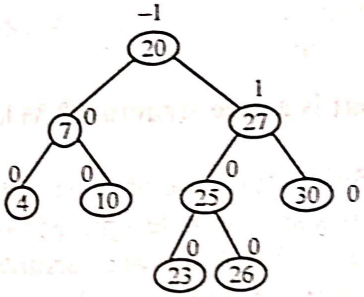
Insert 4



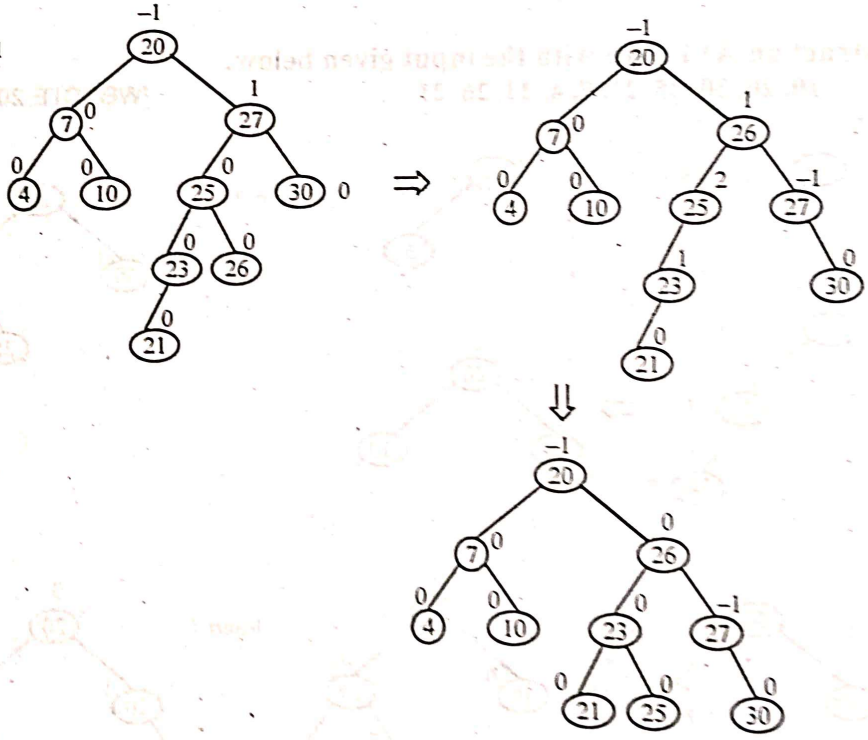
Insert 23



Insert 26



Insert 21



➲ **19. Construct a binary tree where in order and preorder traversals of a binary tree is given as follow:**

In order Sequence: D, G, B, H, E, A, F, I, C

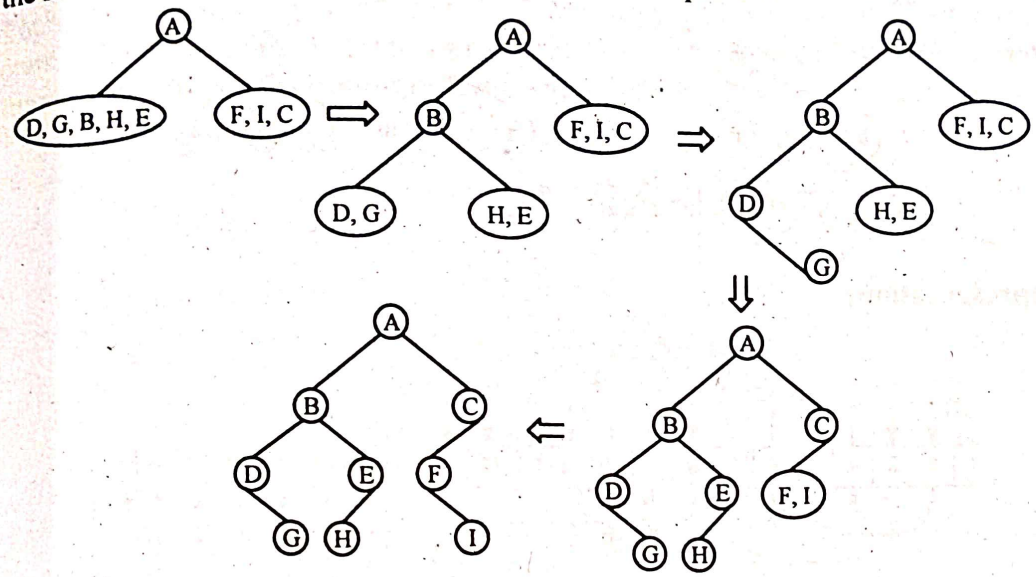
Preorder Sequence: A, B, D, G, E, H, C, F, I **[WBSCTE 2012, 2019]**

Answer:

Basic principle for the formation of the tree is stated below: -

(1) If the post order sequence is given then the last node is the root.

(2) Once the root node is identified, all the nodes in the left sub-trees and right sub-trees of the root node can be identified from the in-order sequence.



➲ **20. What are the advantages of threaded binary tree?**
[WBSCTE 2012, 2013, 2014, 2015, 2019]

Answer:

Advantages of threaded binary tree

1. By doing threading we avoid the recursive method of traversing a Tree, which makes use of stack and consumes a lot of memory and time .
2. The node can keep record of its root.

➲ **21. What are the ways of representing binary trees in memory? Which one do you prefer and why?**
[WBSCTE 2012, 2013, 2014, 2019]

Answer:

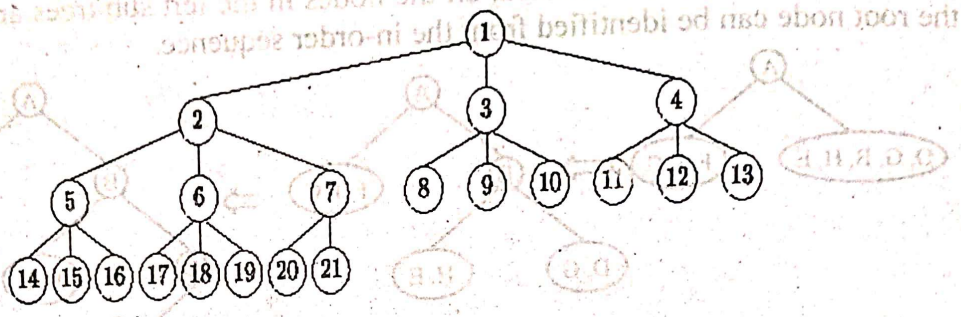
There are two ways by which we can represent a binary tree.

- Linked representation of a binary tree
- Array representation of a binary tree

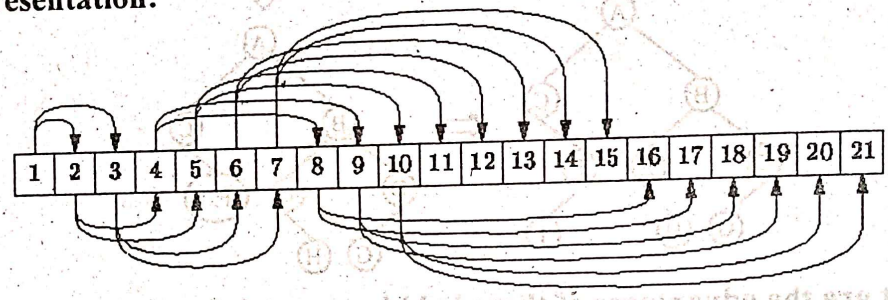
Array representation of binary trees:

When a binary tree is represented by arrays three separate arrays are required. One array stores the data fields of the trees. The other two arrays represent the left child and right child of the nodes.

Binary tree:



Array representation:



Linked representation of binary trees

Binary trees can be represented by links, where each node contains the address of the left child and the right child. These addresses are nothing but kinds to the left and right child respectively. A node that does not have a left or a right child contains a NULL value in its link fields.

Linked representation uses three parallel arrays, INFO, LEFT and RIGHT and a pointer variable ROOT. Each node N of T will correspond to a location K such that –

- INFO[K] contains the data at node N
- LEFT[K] contains the location of left child node N
- RIGHT[K] contains the location of right child node N
- ROOT will contain the location of root R of T

This representation is simpler than array representation, that's why it is more preferable way.

22. (a) Explain how a node can be inserted to a binary search tree. [WBSCTE 2012]

(b) Prove that the maximum number of nodes in a binary tree of height "k" is $2^{k+1}-1$. [WBSCTE 2012, 2019]

(c) Construct an order 5 B-tree with the input given below: [WBSCTE 2012, 2019]
1, 7, 6, 2, 11, 4, 8, 13, 10, 5, 19, 9, 18, 24, 3, 12, 14, 20, 21, 16

Answer:

a) Insertion begins as a search would begin; if the key is not equal to that of the root, we search the left or right subtrees as before. Eventually, we will reach an external node and add the new key-value pair (here encoded as a record 'newNode') as its right or left child, depending on the node's key. In other words, we examine the root and recursively insert

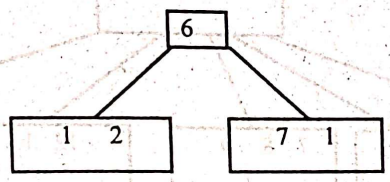
the new node to the left subtree if its key is less than that of the root, or the right subtree if its key is greater than or equal to the root.

b) The maximum number of nodes in a binary tree in level i is 2^i (from the previous theorem). Hence the maximum number of nodes in a binary tree of depth k is, summation of maximum number of nodes from level 0 upto level k , i.e.,

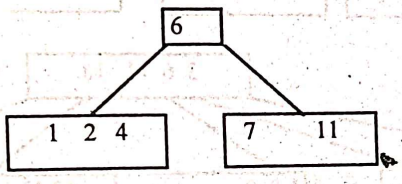
$$\sum_{i=0}^{k} 2^i = 2^0 + 2^1 + 2^2 + \ldots\ldots\ldots + 2^k = 2^0 * \left(\frac{2^{k+1}-1}{2-1}\right) = 2^{k+1} - 1$$

c)

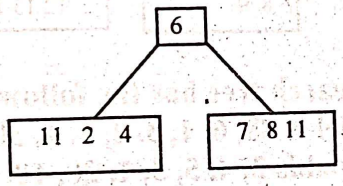
Insert 1:



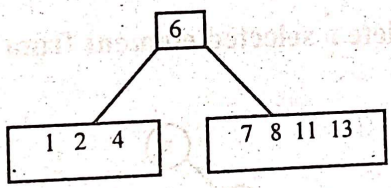
Insert 4:



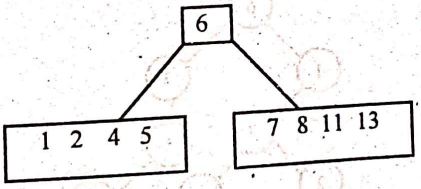
Insert 8:



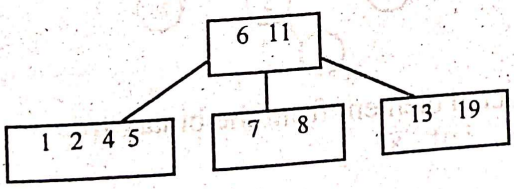
Insert 13:



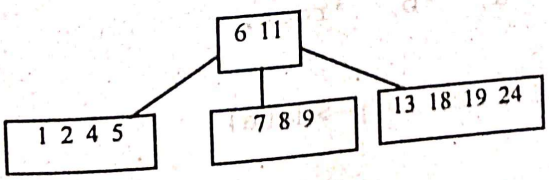
Insert 5:



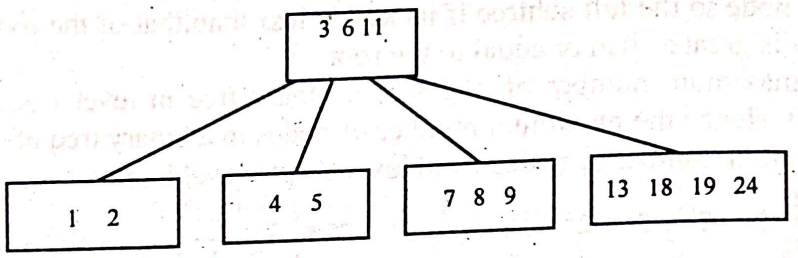
Insert 19:



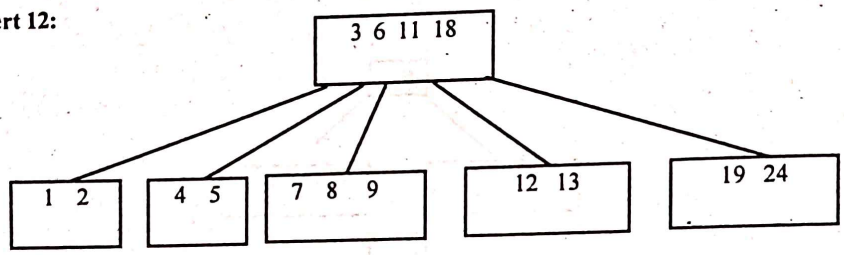
Insert 9, 18, 24:



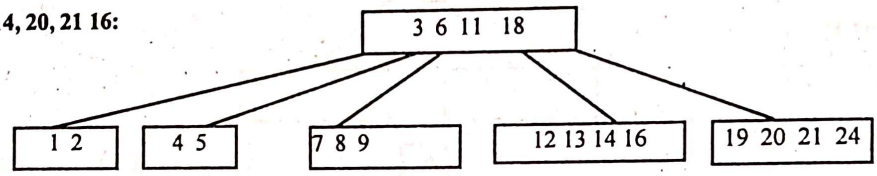
Insert 3:



Insert 12:



Insert 14, 20, 21 16:

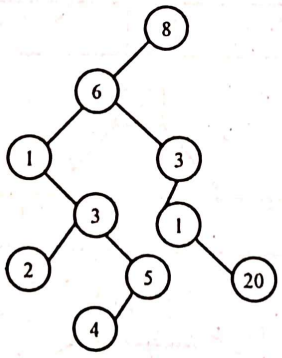


➲ **23. a) A particular binary search tree has the following known about it:**
A pre-order traversal yields 88, 6, 1, 3, 2, 5, 4, 30, 10, 20.
A post-order traversal yields 2, 4, 5, 3, 1, 20, 10, 30, 6, 88.
Draw this tree.
b) Write a routine to delete a selected element from the binary tree.

[WBSCTE 2013]

Answer:

a)



b) A routine to delete a selected element from the binary tree

```
node *delete(node *p, int x)
{
  node *q,  *rp,  *a,  *b,  *root;
  q = NULL;
  root = p;
  while (p != NULL & &x != p->data)
  {
    q = p;
    if (x < p->data)
```

```
    p = p->lc;
  else
    p = p->rc;
}
if (p == NULL)
{
  printf("key not present");
  exit(0);
}
if (p->lc == NULL)
 /* node p has no left child*/
  rp = p->rc;
else if (p->rc == NULL)
  /* p has no right child*/
  rp = p->lc;
else
{
  a = p; /* a contains the parent node p */
  rp = p->rc;
  b = rp->lc; /* b is always left child of rp */
  while (b ! = NULL)
   /* to find the inorder successor with no left child */
  {
    a = rp;
    rp = b;
    b = rp->lc;
  }
     /* At this point rp is the inorder successor of p */
  if (a ! = p)
  {
    a->lc = rp->rc;
    rp->rc = p->rc;
  }
  if (q == NULL)
    nroot = rp;
  else if (p == q->lc)
  {
    q->lc = rp;
    rp->lc = p->lc;
  }
  else
  {
    q->rc = rp;
    rp->lc = p->lc;
  }
  free(p); // delete the node
  return (root);
}
```

24. a) Construct a binary tree where in-order and pre-order traversals of a binary tree produce the following sequences.

Post order: DCBGFEA

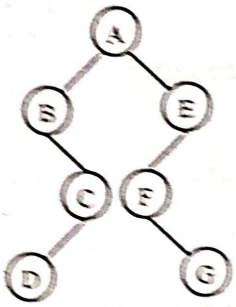
In order: BDCAFGE

b) Construct an AVL tree with the input given below:

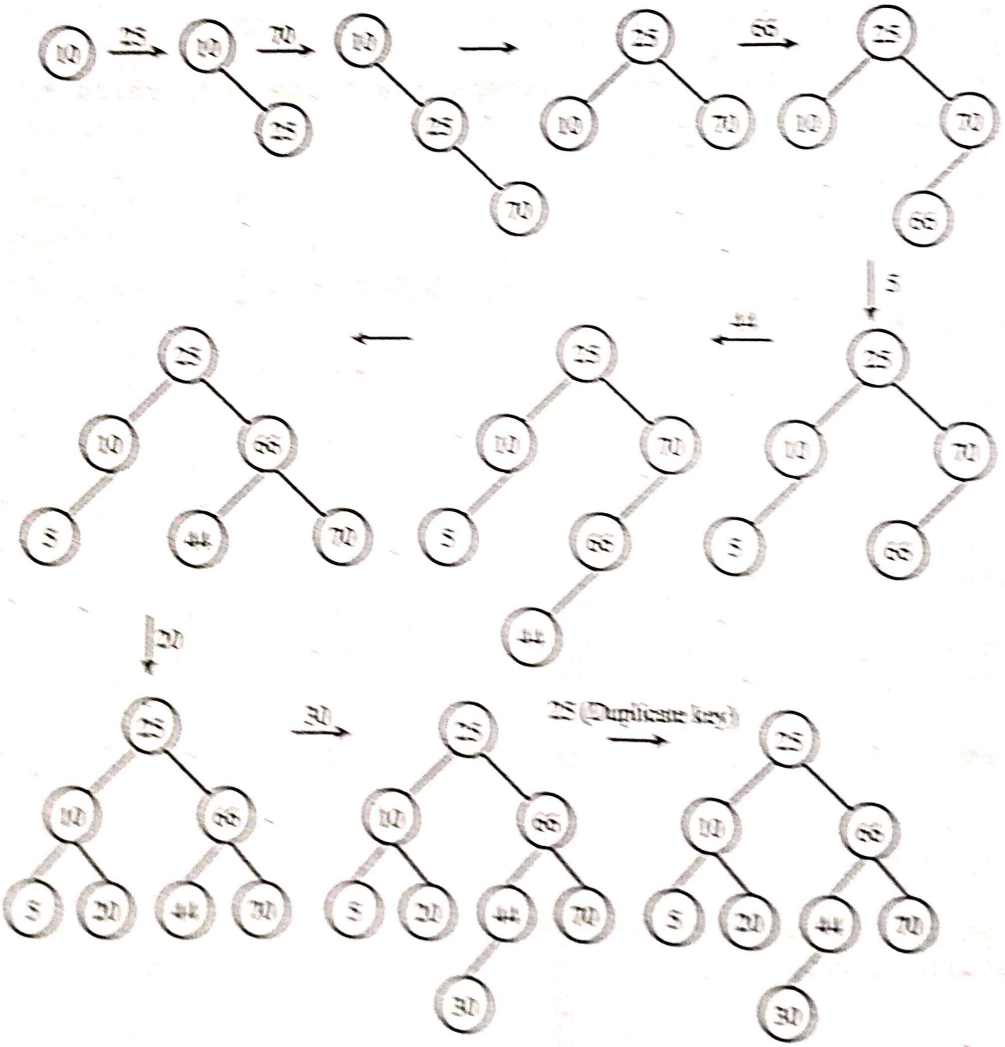
10, 25, 70, 65, 5, 44, 20, 30, 25, 27, 7, 4, 23, 26, 21 [WBSCTE 2014]

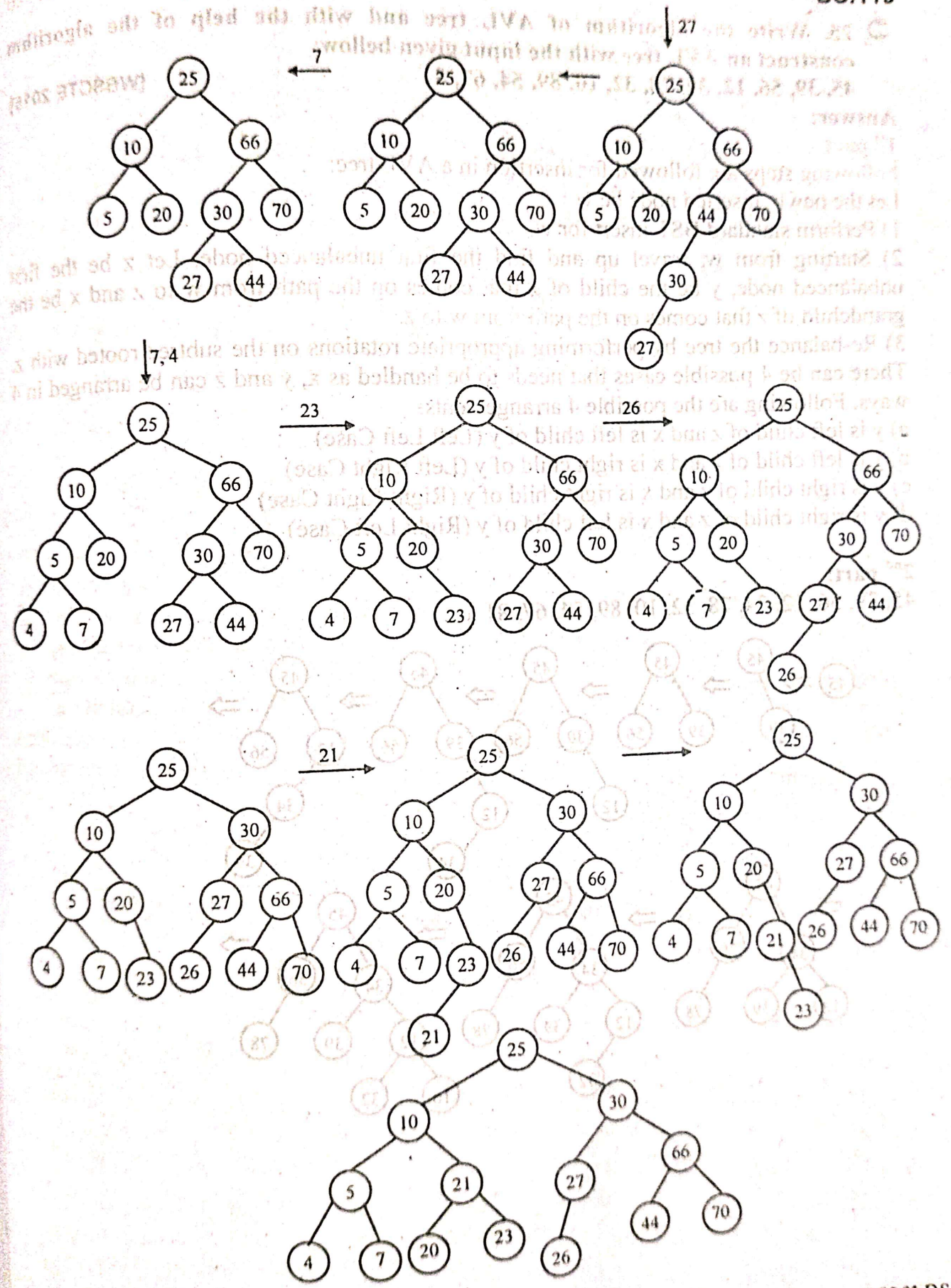
Answer:

a)



b)




27
7
25
10
66
5
20
30
70
27
44
25
10
66
5
20
30
70
27
44
25
10
66
5
20
44
70
30
27
7, 4
25
10
66
5
20
30
70
4
7
27
44
23
25
10
66
5
20
30
70
4
7
23
27
44
26
25
10
66
5
20
30
70
4
7
23
27
44
26
21
25
10
30
5
20
27
66
4
7
23
26
44
70
25
10
30
5
20
27
66
4
7
23
26
44
70
21
25
10
30
5
20
27
66
4
7
21
26
44
70
23
21
25
10
30
5
21
27
66
4
7
20
23
26
44
70

➲ 25. Write the algorithm of AVL tree and with the help of the algorithm construct an AVL tree with the input given bellow: [WBSCTE 2015]
45, 39, 56, 12, 34, 78, 32, 10, 89, 54, 67, 81

Answer:

1st part:

Following steps are followed for insertion in a AVL tree:

Let the newly inserted node be w

1) Perform standard BST insert for w.

2) Starting from w, travel up and find the first unbalanced node. Let z be the first unbalanced node, y be the child of z that comes on the path from w to z and x be the grandchild of z that comes on the path from w to z.

3) Re-balance the tree by performing appropriate rotations on the subtree rooted with z. There can be 4 possible cases that needs to be handled as x, y and z can be arranged in 4 ways. Following are the possible 4 arrangements:

a) y is left child of z and x is left child of y (Left Left Case)

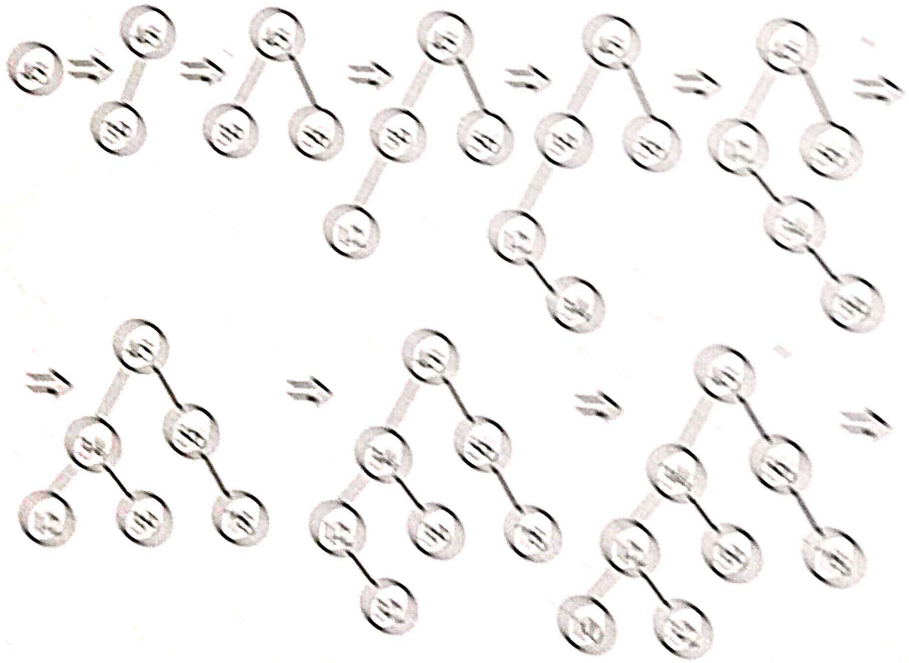
b) y is left child of z and x is right child of y (Left Right Case)

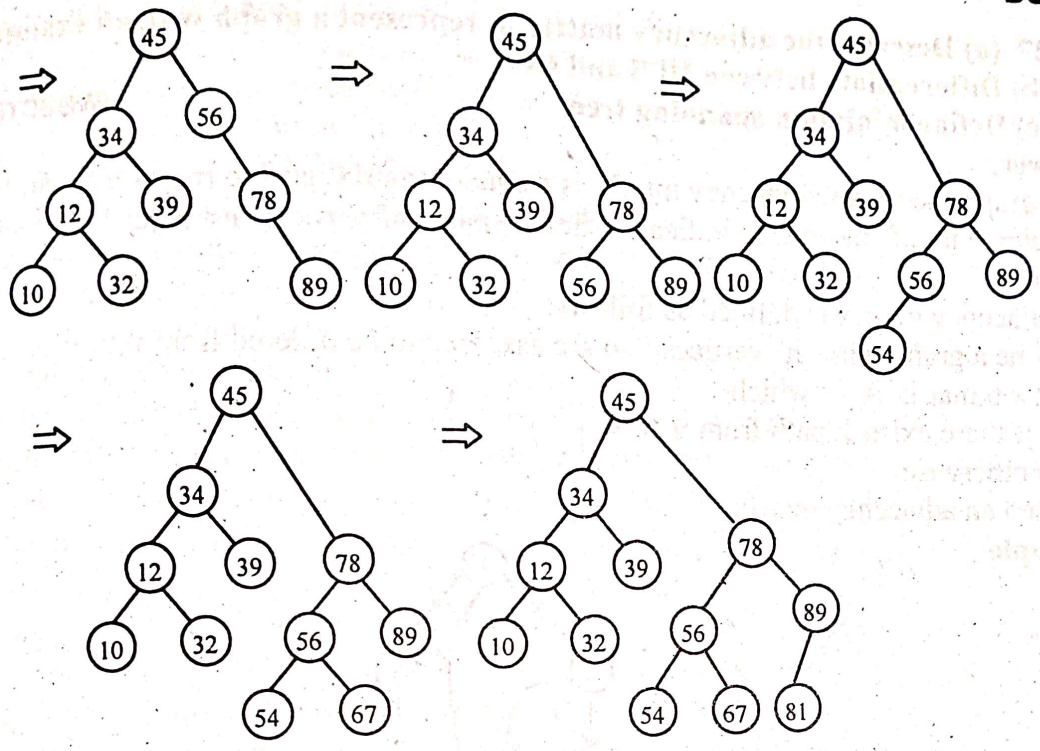
c) y is right child of z and x is right child of y (Right Right Case)

d) y is right child of z and x is left child of y (Right Left Case)

2nd part:

45, 39, 56, 12, 34, 78, 32, 10, 89, 54, 67, 81





➲ **26. Construct a binary tree where in-order traversals of a binary tree produce the following sequences.**

Post Order: D B H I G F E C A

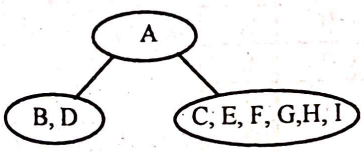
In Order: B D A E H G I F C

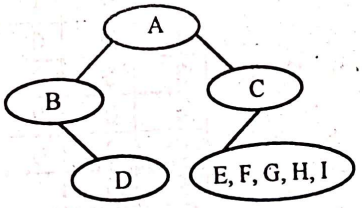
[WBSCTE 2015]

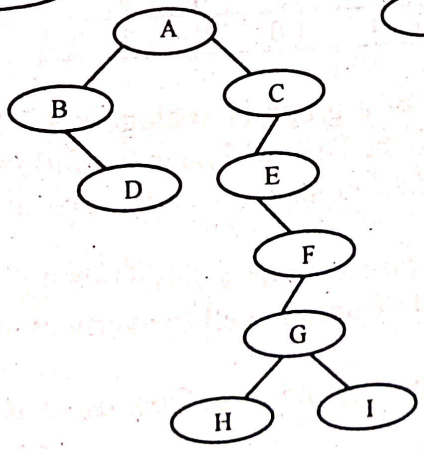
Answer:

Post order: D B H I G F E C A

In. Order: B D A E H G I F C







27. (a) Describe the adjacency matrix to represent a graph with an example.
(b) Differentiate between DFS and BFS.
(c) Define minimum spanning tree. **[WBSCTE 2016]**

Answer:

a) In graph theory an adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph.

An adjacency matrix is defined as follows:

Let G be a graph with "n" vertices that are assumed to be ordered from v_1 to v_n.

The n x n matrix A, in which

$a_{ij} = 1$ if there exists a path from v_i to v_j

$a_{ij} = 0$ otherwise

is called an adjacency matrix.

Example

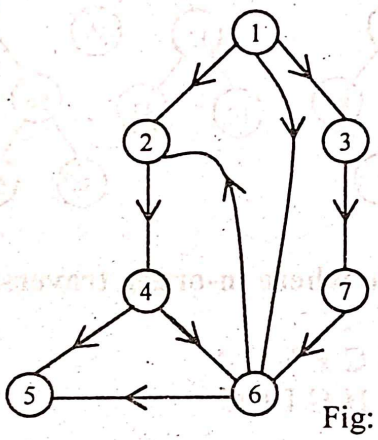


Fig: 1

The adjacency matrix of the above graph is

	1	2	3	4	5	6	7
1	0	1	1	0	0	1	0
2	0	0	0	1	0	0	0
3	0	0	0	0	0	0	1
4	0	0	0	0	1	1	0
5	0	0	0	0	0	0	0
6	0	1	0	0	1	0	0

b) ***Depth First Search:*** DFS of a graph is analogous to the per-order traversal of an ordered tree. For a given graph the DFS will have the following rules.

Rule 1: If possible, visit an adjacent unvisited vertex, mark it visited, and push it on the stack.

Rule 2: If Rule 1 fails, then if possible pop a vertex off the stack follow Rule 1 from it.

Rule 3: Repeat Rule 1 and Rule 2 until the all the vertices are visited.

Breadth First Search: BFS of a graph is analogous to level-by-level traversal of an ordered tree.

For a given graph the BFS will have the following rules.

Rule 1: Visit all the **next** unvisited vertices (if any) that is adjacent to the current vertex, and insert them into a queue one at a time on every visit.
Rule 2: If there is no unvisited vertex, remove a vertex from the Queue and make it the current vertex and then follow Rule 1.
Rule 3: Repeat Rule 1 and Rule 2 until the all the vertices visited.
Breadth-first search (BFS) is a graph search algorithm that begins at the root node and explores all the neighboring nodes. Then for each of those nearest nodes, it explores their unexplored neighbor nodes, and so on, until it finds the goal.
The time complexity can also be expressed as $O(|E|+|V|)$ since every vertex and every edge will be explored in the worst case.
Depth-first search (DFS) is an algorithm for traversing or searching a tree, tree structure, or graph. Intuitively, 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. The time complexity is also given by $O(|E|+|V|)$.

c) ***Refer to 4.10 of Chapter at a Glance.***

➲ **28. (a) What do you mean by depth of the tree?**
(b) Write down the algorithm to implement preorder traversal in a binary tree.
(c) Write down an algorithm to insert an element into a binary tree.
[WBSCTE 2016]

Answer:
a) The **depth** of a node is the number of edges from the node to the **tree's** root node. A root node will have a **depth** of 0. The height of a node is the number of edges on the longest path from the node to a leaf. A leaf node will have a height of 0.

b)
```
Algorithm Preorder(tree)
  1. Visit the root.
  2. Traverse the left subtree, i.e., call Preorder(left-
subtree)
  3. Traverse the right subtree, i.e., call Preorder(right-
subtree)
```

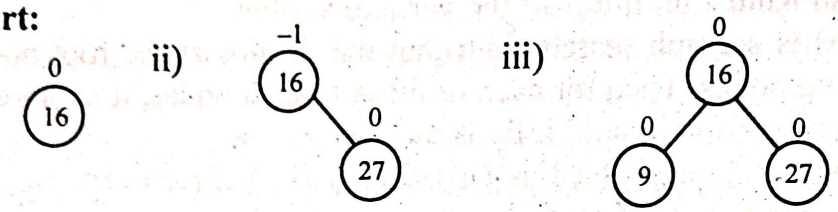
c)
```
struct node* insert(struct node* node, int key)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(key);
    /* Otherwise, recur down the tree */
    if (key < node->key)
        node->left  = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);
    /* return the (unchanged) node pointer */
    return node;
}
```

29. Create an AVL tree using the following sequence of data: 16, 27, 9, 11, 2, 15, 99, 36, 54, 81, 63, 72. Define Binary Search Tree with an example. [WBSCTE 2017]

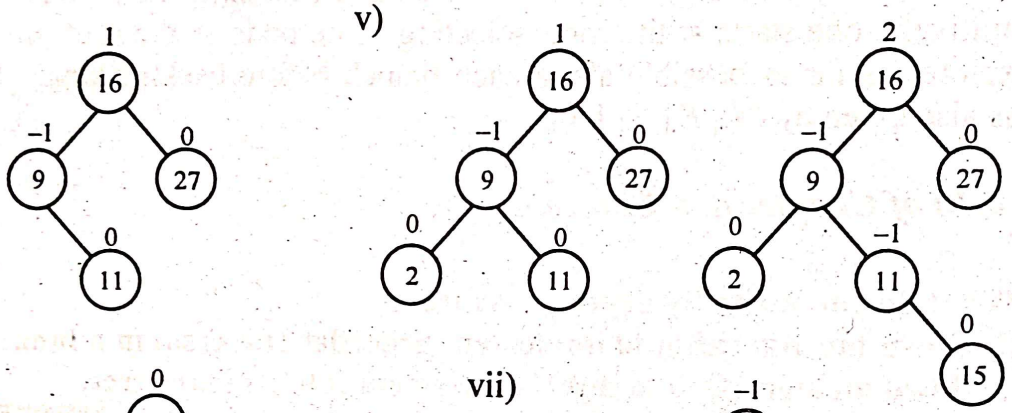
Answer:

1st Part:

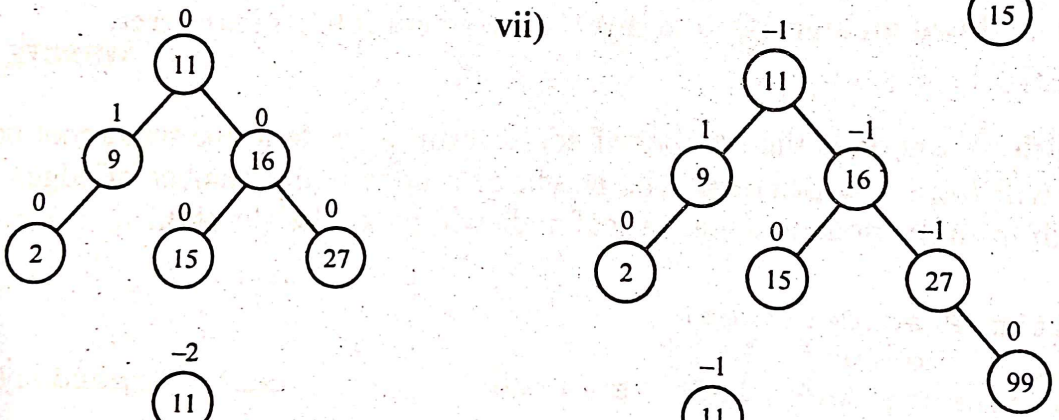
i) ii) iii)



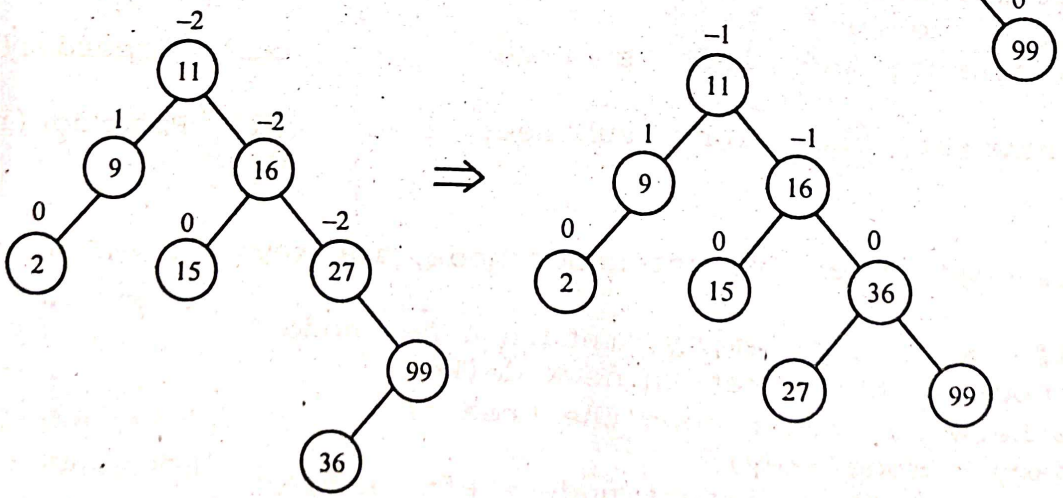
iv) v)



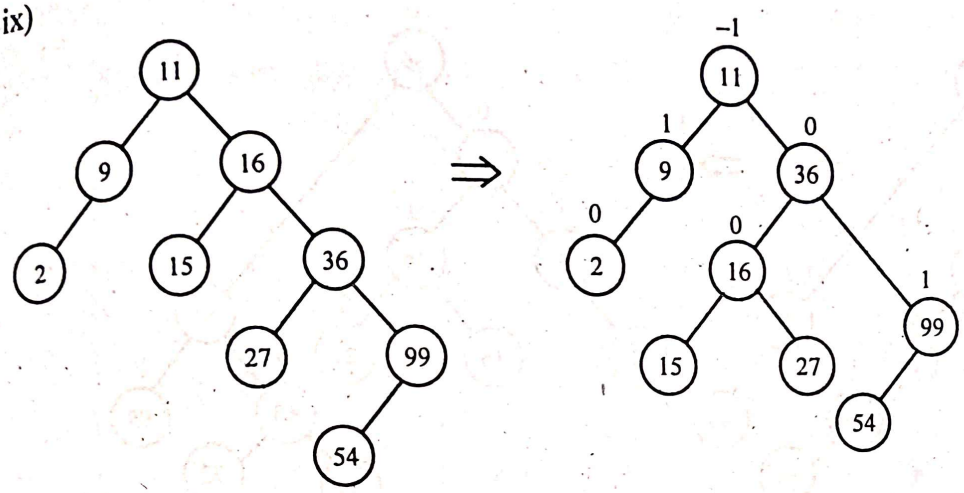
vi) vii)



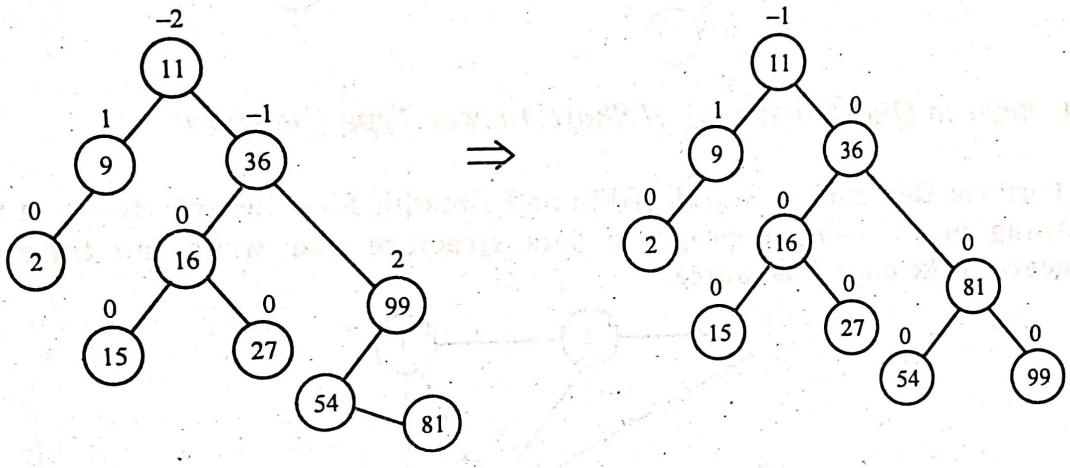
viii)



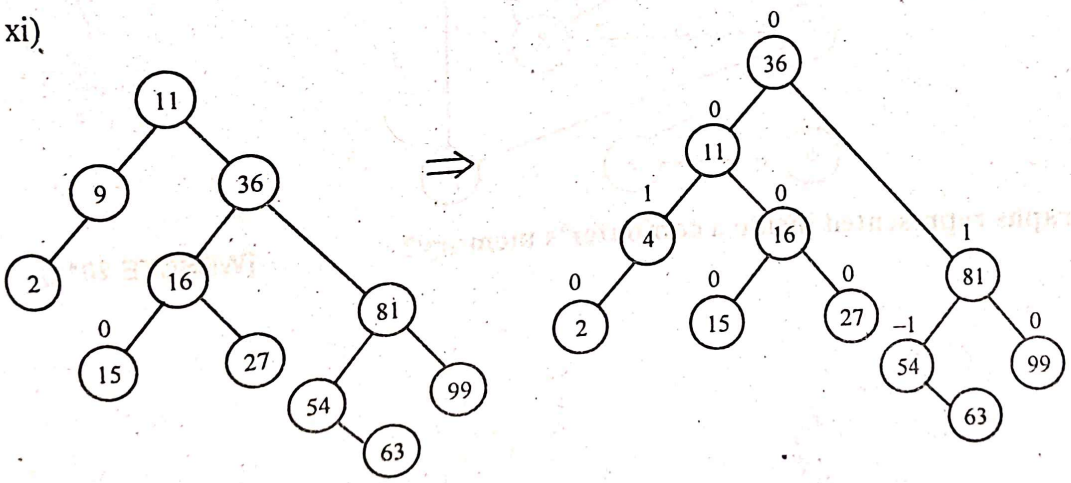
ix)


11
9
16
2
15
36
27
99
54
⇒
−1
11
1
0
9
36
0
0
2
16
1
99
15
27
54

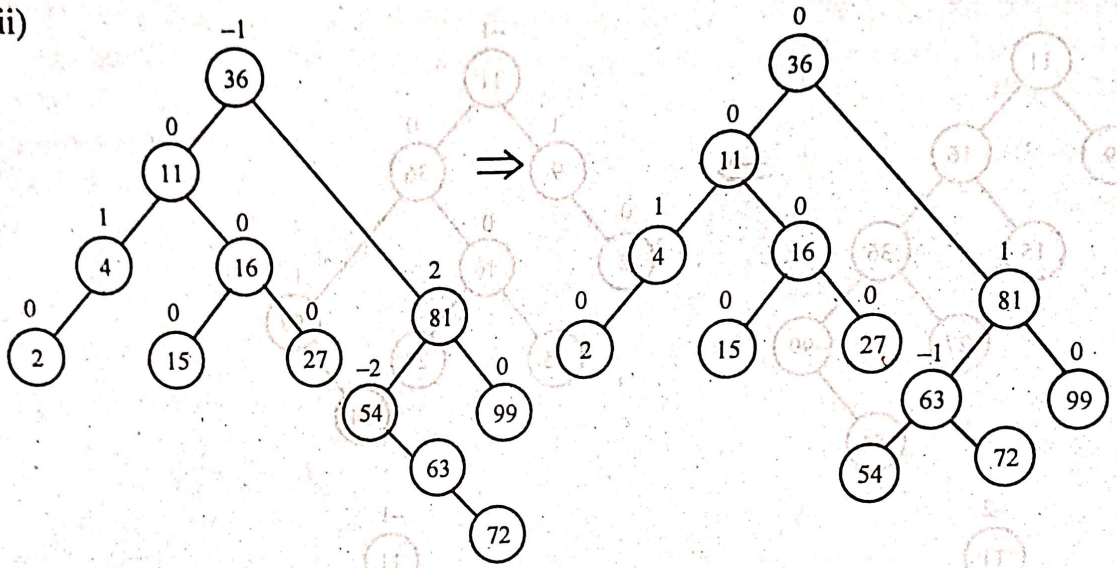
x)


−2
11
1
−1
9
36
0
0
2
16
2
0
0
99
15
27
54
81
⇒
−1
11
1
0
9
36
0
0
2
16
0
0
0
81
15
27
0
0
54
99

xi)

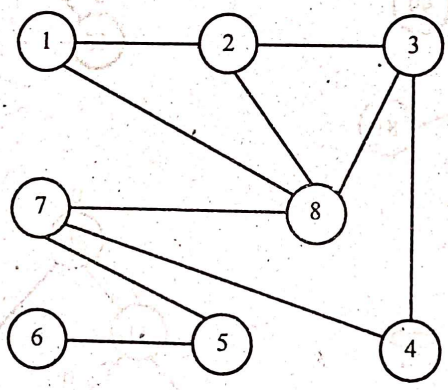

11
9
36
2
16
81
0
15
27
54
99
63
⇒
0
36
0
11
1
0
4
16
1
0
0
0
81
2
15
27
−1
0
54
99
63

xii)



2nd Part: ***Refer to Question No. 61 of Short Answer Type Questions.***

➲ **30. Perform Depth First Search (DFS) and Breadth First Search (BFS) on the following graph using appropriate data structure and write the traversal sequence. Take node 1 as source.**

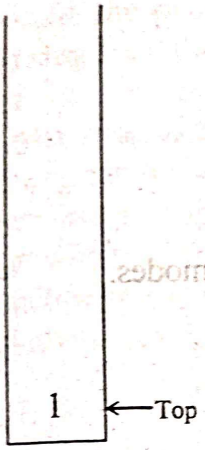


How are graphs represented inside a computer's memory? **[WBSCTE 2017]**

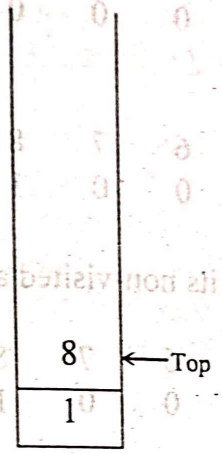
Answer:

1) DFS Stack data structure is used.

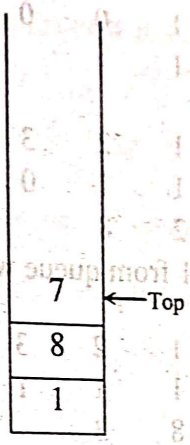
1)



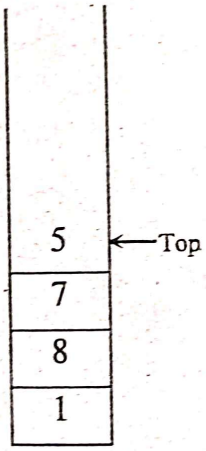
2)



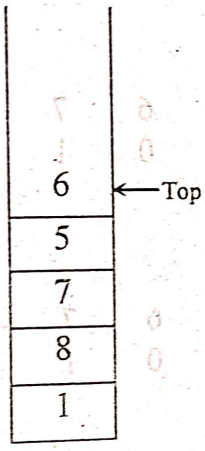
3)



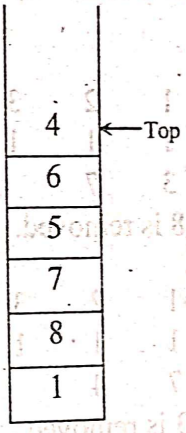
4)



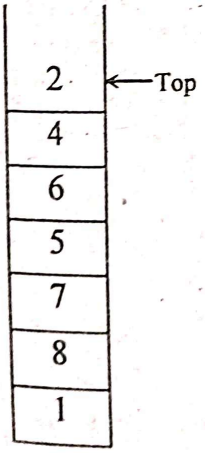
5)



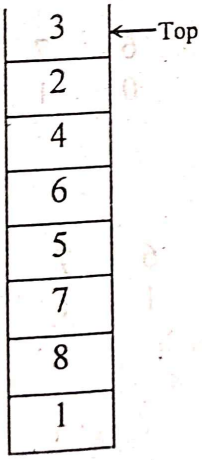
6)



7)



8)



Traversal Sequence

1→8→7→5→6→4→2→3

BFS Queue data structure is used

1.

	1	2	3	4	5	6	7	8
Visited:	1	0	0	0	0	0	0	0
Queue:	1							

2.

	1	2	3	4	5	6	7	8
Visited:	1	1	0	0	0	0	0	1
Queue:	2	8						

After removing 1 from queue we can queue its non-visited adjacent modes.

3.

	1	2	3	4	5	6	7	8
Visited:	1	1	1	0	0	0	0	1
Queue:	8	3						

2 is removed.

4.

	1	2	3	4	5	6	7	8
Visited:	1	1	1	0	0	0	1	1
Queue:	3	7						

8 is removed.

5.

	1	2	3	4	5	6	7	8
Visited:	1	1	1	1	0	0	1	1
Queue:	7	4						

3 is removed

6.

	1	2	3	4	5	6	7	8
Visited:	1	1	1	1	1	0	1	1
Queue:	4	5						

7 is removed.

7.

	1	2	3	4	5	6	7	8
Visited:	1	1	1	1	1	1	1	1
Queue:	5	6						

4 is removed.

(6 is explored after traversing 5)
Traversal sequence: 1→2→8→3→7→4→5→6

II) An adjacency list contains, for each vertex, a list of vertices that are connected to it by an edge. This list can either be in an array or in a linked list. If the vertices are ordered

and stored in arrays, a binary search can be used to look up an edge. If the vertices are stored in linked list, then new edges can be added efficiently

31. State the properties of heap data structure. Construct a max heap with the following set values: 5, 4, 15, 12, 19, 13, 8, 16, 20, 6. [WBSCTE 2017]

Answer:

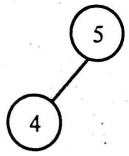
(I) A binary heap is a complete binary tree which satisfies the heap ordering property. The ordering can be one of two types:

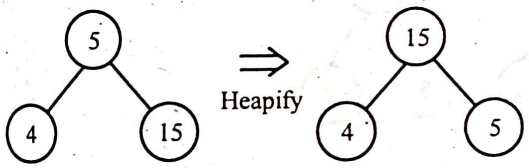
1) **the min-heap property**: the value of each node is greater than or equal to the value of its parent, with the minimum-value element at the root.

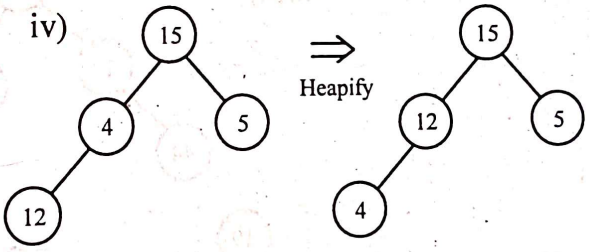
2) **the max-heap property**: the value of each node is less than or equal to the value of its parent, with the maximum-value element at the root.

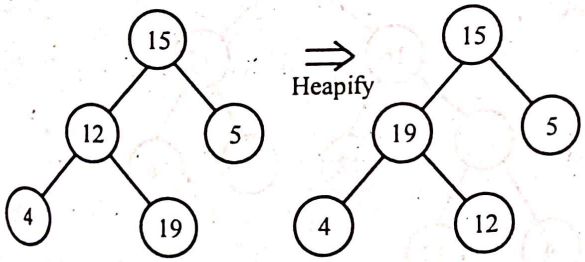
(II)

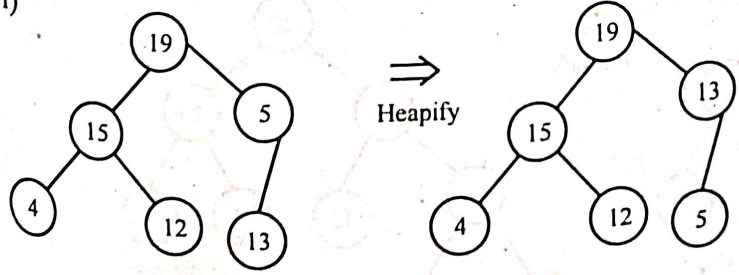
i) 

ii) 

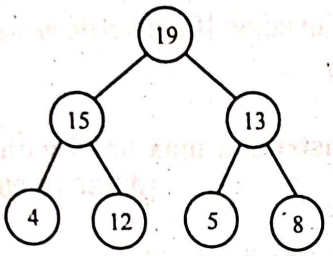
iii) 

iv) 

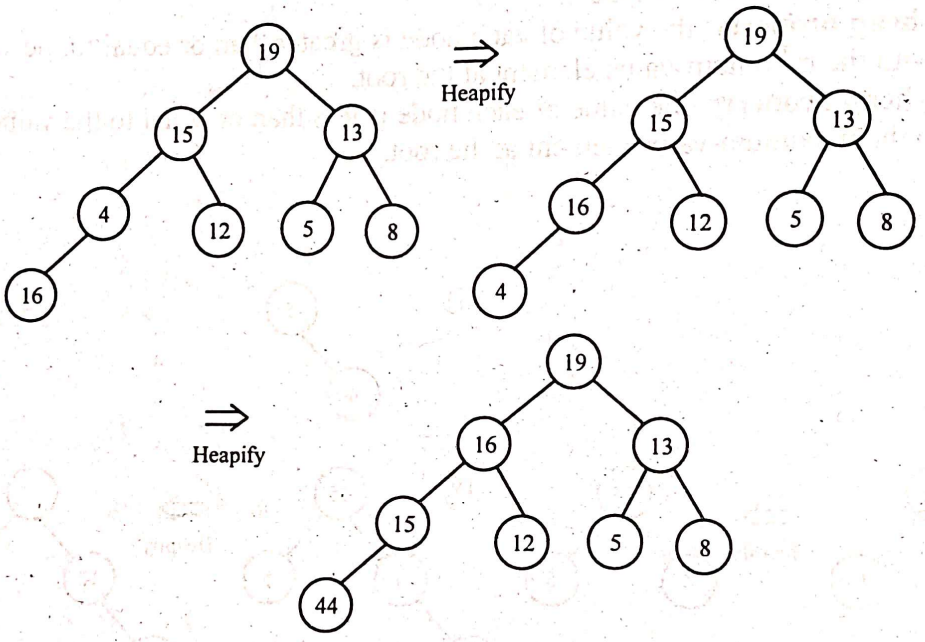
v) 

vi) 

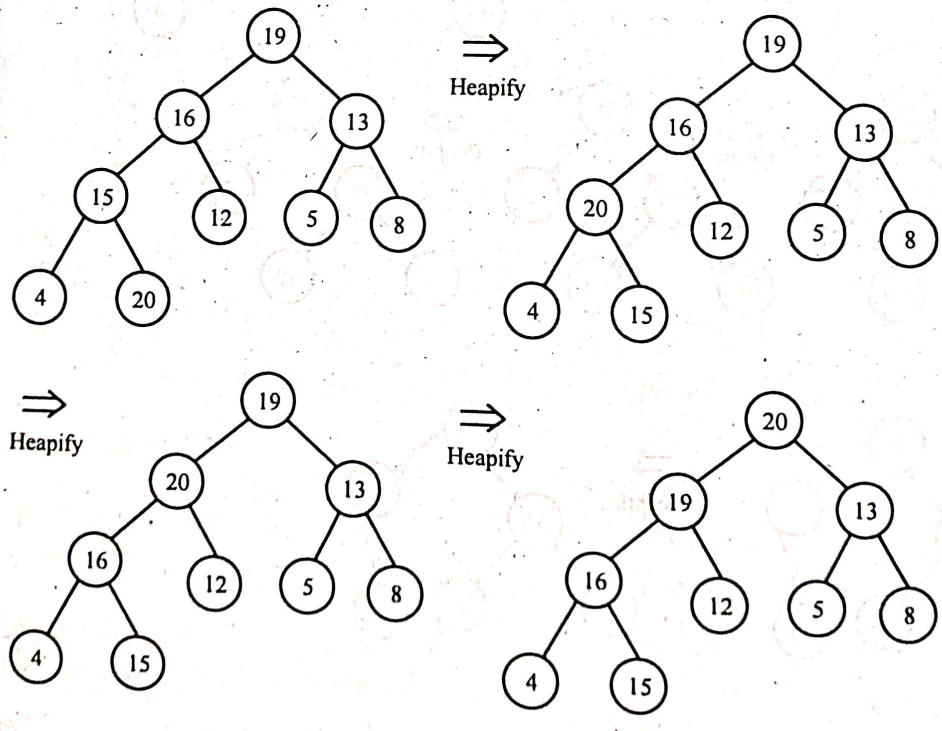
vii)


19
15
13
4
12
5
8

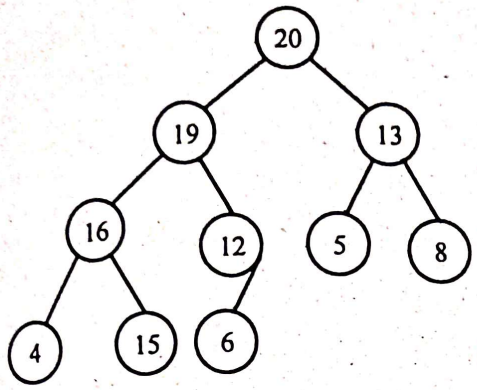
viii)


19
15
13
4
12
5
8
16
⇒
Heapify
19
15
13
16
12
5
8
4
⇒
Heapify
19
16
13
15
12
5
8
44

ix)


19
16
13
15
12
5
8
4
20
⇒
Heapify
19
16
13
20
12
5
8
4
15
⇒
Heapify
19
20
13
16
12
5
8
4
15
⇒
Heapify
20
19
13
16
12
5
8
4
15

x)

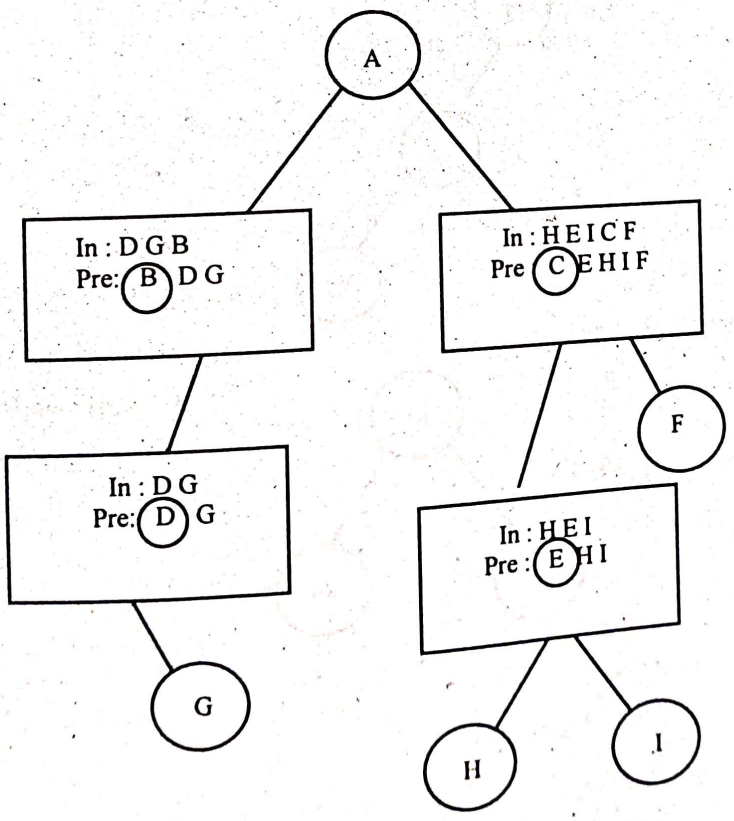


→ Man Heap tree

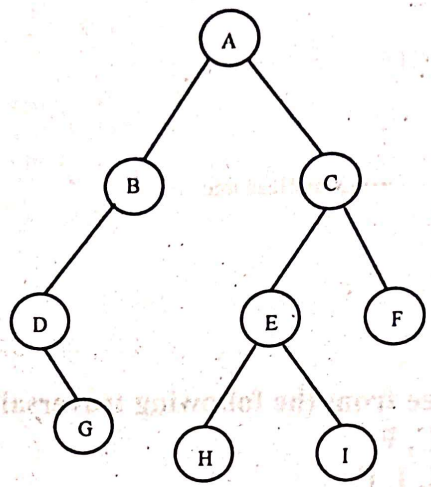
32. a) Reconstruct a binary tree from the following traversal sequences.
In-order: D, G, B, A, H, E, I, C, F
Pre-order: A, B, D, G, C, E, H, I, F
b) Create an AVL tree using the following sequence of data:
12, 23, 89, 11, 9, 1, 99, 36, 54, 81

[WBSCTE 2018, 2021]

Answer:
a) In-order: D G B A H E I C F
Pre-order: A B D G C E H I F
So, root is – A



So, the tree is –

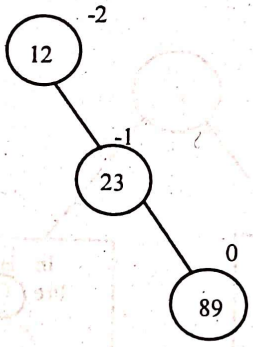


b)

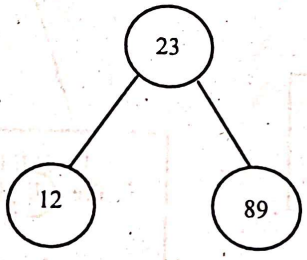
Sequence: 12, 23, 89, 11, 91, 99, 36, 54, 81

Step-1.

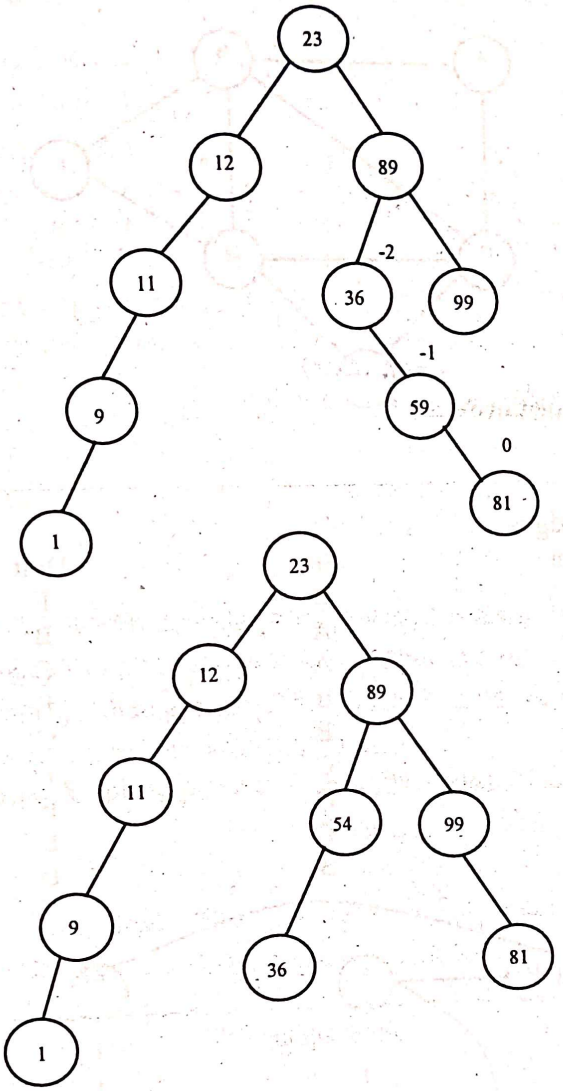
Step-2.



Step-2.1.

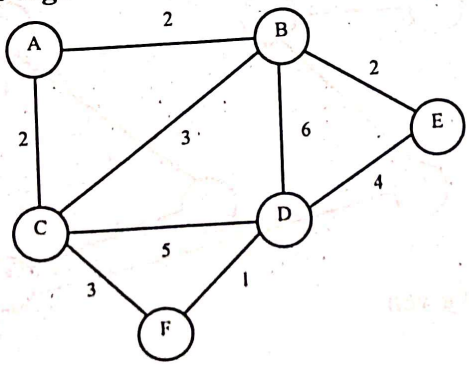


Step-3.

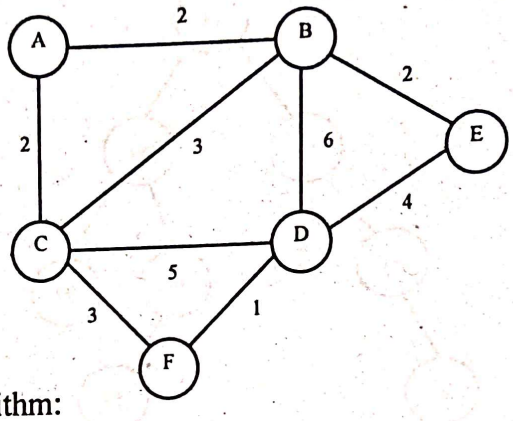


This is the final AVL tree.

➲ **33. Consider the graph given below. Find the minimum spanning tree of this graph using Kruskal's algorithm.** [WBSCTE 2018, 2021]

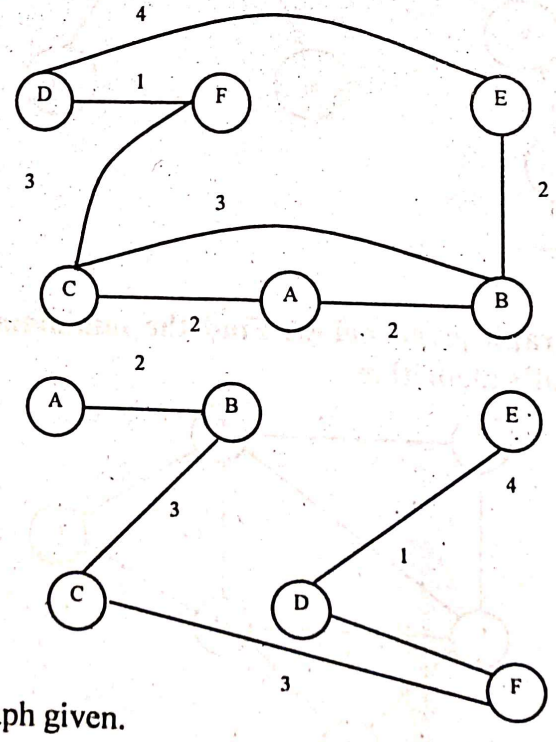


Answer:



Applying Kruskal's algorithm:
Total vertices = 6
Total edges = 9
∴ MST will have = 5 edges

Weight	Src	Dest
1	D	F
2	A	B
2	A	C
2	B	E
3	B	C
3	C	F
4	D	E
5	C	D
6	B	D



It is the MST of the graph given.

➲ 34. Write an algorithm to insert a node in AVL tree. [WBSCTE 2019, 2022]

Answer:

Following steps are followed for insertion in a AVL tree:

Let the newly inserted node be w

1) Perform standard BST insert for w.

2) Starting from w, travel up and find the first unbalanced node. Let z be the first unbalanced node, y be the child of z that comes on the path from w to z and x be the grandchild of z that comes on the path from w to z.

3) Re-balance the tree by performing appropriate rotations on the subtree rooted with z. There can be 4 possible cases that needs to be handled as x, y and z can be arranged in 4 ways. Following are the possible 4 arrangements:

a) y is left child of z and x is left child of y (Left Left Case)

b) y is left child of z and x is right child of y (Left Right Case)

c) y is right child of z and x is right child of y (Right Right Case)

d) y is right child of z and x is left child of y (Right Left Case)

➲ 35. Compare BFS and DFS. [WBSCTE 2019]

Answer:

Refer to Question No. 27.b) of Long Answer Type Questions.

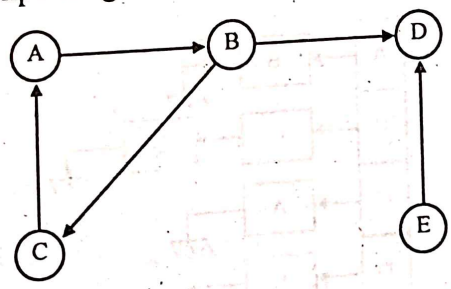
➲ 36. Define a graph. Explain different representation of graph. [WBSCTE 2022]

Answer:

1st part: ***Refer to 4.6 of Unit at a Glance.***

2nd part:

The graph is a non-linear data structures. This represents data using nodes, and their relations using edges. A graph G has two sections. The vertices, and edges. Vertices are represented using set V, and Edges are represented as set E. So the graph notation is G(V,E). Let us see one example to get the idea.



In this graph, there are five vertices and five edges. The edges are directed. As an example, if we choose the edge connecting vertices B and D, the source vertex is B and destination is D. So we can move B to D but not move from D to B.

The graphs are non-linear, and it has no regular structure. To represent a graph in memory, there are few different styles. These styles are –

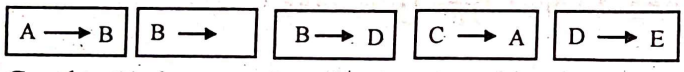
- Adjacency matrix representation
- Edge list representation
- Adjacency List representation

Adjacency Matrix Representation

We can represent a graph using Adjacency matrix. The given matrix is an adjacency matrix. It is a binary, square matrix and from ith row to jth column, if there is an edge, that place is marked as 1. When we will try to represent an undirected graph using adjacency matrix, the matrix will be symmetric.

	A	B	C	D	E
A	0	1	0	0	0
B	0	0	1	1	0
C	1	0	0	0	0
D	0	0	0	0	1
E	0	0	0	0	0

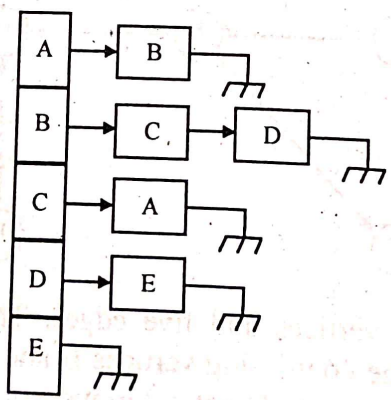
Edge List Representation



Graphs can be represented using one dimensional array also. This is called the edge list. In this representation there are five edges are present, for each edge the first element is the source and the second one is the destination. For undirected graph representation the number of elements in the edge list will.be doubled.

Adjacency List Representation

This is another type of graph representation. It is called the adjacency list. This representation is based on Linked Lists. In this approach, each Node is holding a list of Nodes, which are Directly connected with that vertices. At the end of list, each node is connected with the null values to tell that it is the end node of that list.

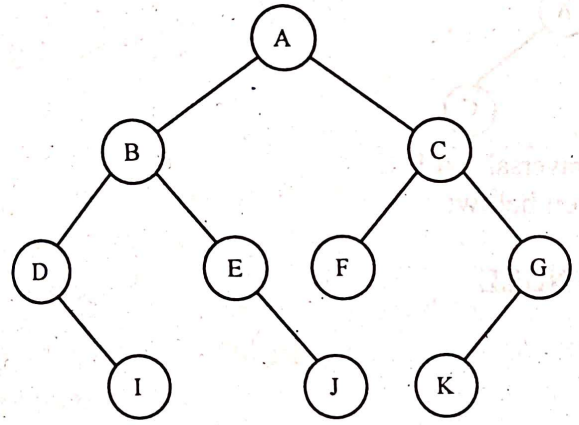


➲ **37. Given the Pre-order and In-order traversal of binary tree. Braw the tree representation and write its Post-order traversal-**

Pre-order:	**A**	**B**	**D**	**I**	**E**	**J**	**C**	**F**	**G**	**K**
In-order:	**D**	**I**	**B**	**E**	**J**	**A**	**F**	**C**	**K**	**G**

[WBSCTE 2022]

Answer:



Post order traversal is I D J E B F K G C A

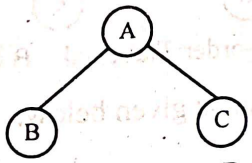
➲ **38. Write a recursive algorithm for binary tree traversal with an example.**
[WBSCTE 2022]

Answer:

In-order Traversal of Binary Tree

The following operations are done recursively at each node to traverse a non-empty binary tree in order.

- Traverse the left subtree of R in inorder.
- Process the root, R.
- Traverse the right subtree of R in inorder.



In-order Traversal – B A C

The algorithm for in-order traversal is given below:

Algorithm: inorder

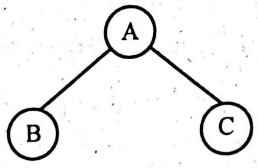
Step 1: Repeat Steps 2 to 4 **while** TREE != **NULL**
Step 2: inorder(TREE -> LEFT)
Step 3: Write TREE -> DATA
Step 4: inorder(TREE -> RIGHT)
[end of LOOP]

Step 5: **end**

Pre-order Traversal of Binary Tree

The following operations are done recursively at each node to traverse a non-empty binary tree in pre-order. Pre-order traversal is also called **depth-first traversal.**

- Process the root, R.
- Traverse the left subtree of R in preorder.
- Traverse the right subtree of R in preorder.



Pre-order Traversal – A B C

The algorithm for pre-order traversal is given below:

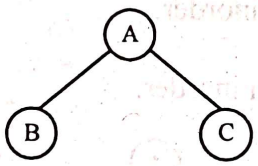
Algorithm: preorder

Step 1: Repeat Steps 2 to 4 **while** TREE != **NULL**
Step 2: Write TREE -> DATA
Step 3: preorder(TREE -> LEFT)
Step 4: preorder(TREE -> RIGHT)
[**end of** LOOP]
Step 5: **end**

Post-order Traversal of Binary Tree

The following operations are done recursively at each node to traverse a non-empty binary tree in post-order.

- Traverse the left subtree of R in postorder.
- Traverse the right subtree of R in postorder.
- Process the root, R.



Post-order Traversal – B C A

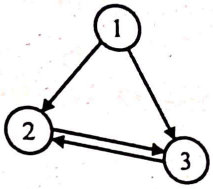
The algorithm for post-order traversal is given below:

Algorithm: postorder

Step 1: Repeat Steps 2 to 4 **while** TREE != **NULL**
Step 2: postorder(TREE -> LEFT)
Step 3: postorder(TREE -> RIGHT)
Step 4: Write TREE -> DATA
[**end of** LOOP]
Step 5: **end**

➲ **39. Define a directed graph. Provide an example.** **[Model Question]**

Answer:
A directed graph is a graph whose edges are *ordered* pairs of vertices. That is, each edge can be followed from one vertex to another vertex.



Directed graph (V_2, E_2)

$V_2 = \{1, 2, 3\}$

$E_2 = \{(1, 2), (2, 3), (3, 2), (1, 3)\}$

Formal Definition: A graph G is a pair (V,E), where V is a set of vertices, and E is a set of edges between the vertices $E \subseteq \{(u,v) \mid u, v \in V\}$. If the graph does not allow *self-loops*, adjacency is *irreflexive*, that is $E \subseteq \{(u,v) \mid u, v \in V \wedge u \neq v\}$.
For example, edge (2,3) is directed from 2 to 3 , which is different than the directed edge (3,2) from 3 to 2. Directed graphs are drawn with arrowheads on the links, as shown in the figure.

➲ **40. Discuss about the following terminology** **[Model Question]**

a) **Indegree**
b) **Sink**
c) **Cycle**
d) **Network**

Answer:
a) In a directed graph the **indegree** of a vertex v is the number of edges terminating at v.

b) A local sink is a node of a directed graph with no exiting edges – it is also called a terminal. A global sink (often simply called a sink) is a node in a directed graph which is reached by all directed edges.

c) In a graph, a cycle is a sequence $v_0, e_1, v_1, \ldots, e_k, v_k (k \geq 1)$ of alternately vertices and edges (where e_i is an edge joining v_{i-1} and v_i), with all the edges different and all the vertices different, except that $v_0 = v_k$.

d) A network is a set of items (nodes or vertices) connected by edges or links. A network is represented by a graph using adjacency matrix ususally.

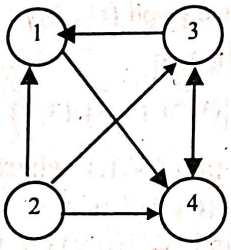
➲ **41. Define adjacency matrix corresponding to a digraph.** **[Model Question]**

Answer:
Suppose that D is a digraph with n vertices, numbered 1, 2, . . . , n. Then the **adjacency matrix** A(D) of D is a square n × n array, with rows and columns numbered 1, 2, . . . , n, such that the entry in row i and column j is the number of arcs from vertex i tovertex j.

➲ **42. Draw the graph corresponding to the following bit matrix:** **[Model Question]**

$$A = \begin{matrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{matrix}$$

Answer:



➲ **43. What is path matrix?** **[Model Question]**

Answer:

It is a 2-dimensional matrix. At each step in the **Floyd–Warshall algorithm,** path[i][j] is the shortest path from i to j using intermediate vertices (1..k−1). Each path[i][j] is initialized to *edgeCost*(i,j) or infinity if there is no edge between *i* and *j*.