DATA STRUCTURE

Introduction to Data Structures:

Data structure is a representation of logical relationship existing between individual elements of data.

In other words, a data structure defines a way of organizing all data items that considers not only the elements stored but also their relationship to each other.

The term data structure is used to describe the way data is stored.

- To develop a program of an algorithm we should select an appropriate data structure for that algorithm. Therefore, data structure is represented as:

  Algorithm + Data structure = Program

- A data structure is said to be linear if its elements form a sequence or a linear list.
- The linear data structures like an array, stacks, queues and linked lists organize data in linear order.
- A data structure is said to be non linear if its elements form a hierarchical classification where, data items appear at various levels

- Trees and Graphs are widely used non-linear data structures.
- Tree and graph structures represents hierarchial relationship between individual data elements.
- Graphs are nothing but trees with certain restrictions removed.

- Data structures are divided into two types:
  - Primitive data structures.
  - Non-primitive data structures.

- Primitive data structures.
- the basic data structures that directly operate upon the machine instructions.
- They have different representations on different computers.
- Integers, floating point numbers, character constants, string constants and pointers come under this category.

- Non-primitive data structures.
- more complicated data structures and are derived from primitive data structures.
- They emphasize on grouping same or different data items with relationship between each data item.
- Arrays, lists and files come under this category.

Figure 1.1 shows the classification of data structures.

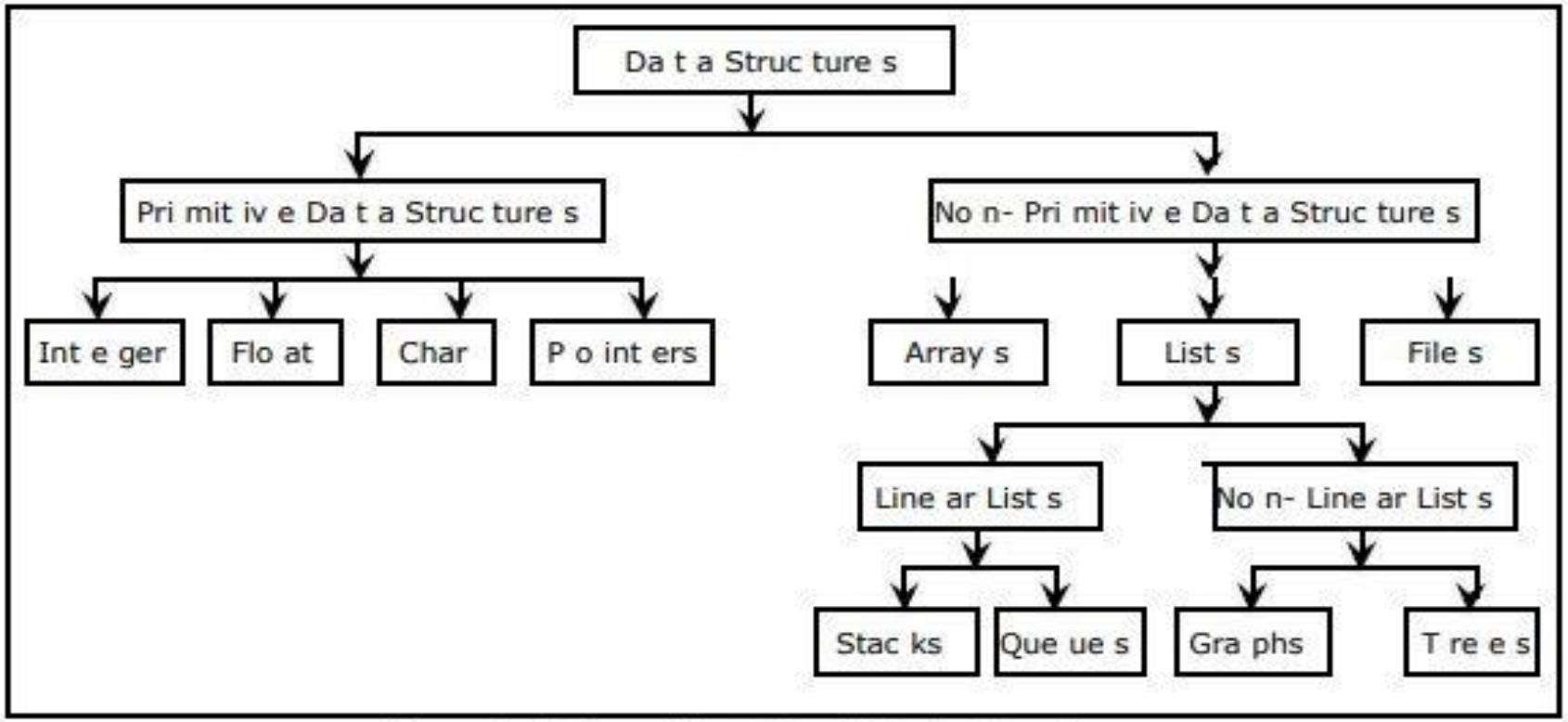


Figure 1. 1. Cla s s if ic at io n of Da t a Struc ture s

Linear data structures

- A data structure is said to be Linear, if its elements are connected in linear fashion by means of logically or in sequence memory locations.
- There are two ways to represent a linear data structure in memory,
  - Static memory allocation
  - Dynamic memory allocation
- The possible operations on the linear data structure are: Traversal, Insertion, Deletion, Searching, Sorting and Merging.
- Examples of Linear Data Structure are Stack and Queue.
- Stack: Stack is a data structure in which insertion and deletion operations are performed at one end only.
  - The insertion operation is referred to as 'PUSH' and deletion operation is referred to as 'POP' operation.
  - Stack is also called as Last in First out (LIFO) data structure.
- Queue: The data structure which permits the insertion at one end and Deletion at another end, known as Queue.
  - End at which deletion is occurs is known as FRONT end and another end at which insertion occurs is known as REAR end.
  - Queue is also called as First in First out (FIFO) data structure.

Nonlinear data structures

- Nonlinear data structures are those data structure in which data items are not arranged in a sequence.
- Examples of Non-linear Data Structure are Tree and Graph.
- **Tree:** A tree can be defined as finite set of data items (nodes) in which data items are arranged in branches and sub branches according to requirement.
  - Trees represent the hierarchical relationship between various elements.
  - Tree consist of nodes connected by edge, the node represented by circle and edge lives connecting to circle.

- **Graph:** Graph is a collection of nodes (Information) and connecting edges (Logical relation) between nodes.
  - A tree can be viewed as restricted graph.
  - Graphs have many types:
    - Un-directed Graph
    - Directed Graph
    - Mixed Graph
    - Multi Graph
    - Simple Graph
    - Null Graph
    - Weighted Graph

Difference between Linear and Non Linear Data Structure

Linear Data Structure	Non-Linear Data Structure
Every item is related to its previous and next time.	Every item is attached with many other items.
Data is arranged in linear sequence.	Data is not arranged in sequence.
Data items can be traversed in a single run.	Data cannot be traversed in a single run.
Eg. Array, Stacks, linked list, queue.	Eg. tree, graph.
Implementation is easy.	Implementation is difficult.

Operation on Data Structures

Design of efficient data structure must take operations to be performed on the data structures into account. The most commonly used operations on data structure are broadly categorized into following types

1. **Create**
   The create operation results in reserving memory for program elements. This can be done by declaration statement. Creation of data structure may take place either during compile-time or run-time. malloc() function of C language is used for creation.

2. **Destroy**
   Destroy operation destroys memory space allocated for specified data structure. free() function of C language is used to destroy data structure.

3. **Selection**
   Selection operation deals with accessing a particular data within a data structure.

4. **Updation**
   It updates or modifies the data in the data structure.

5. **Searching**
   It finds the presence of desired data item in the list of data items, it may also find the locations of all elements that satisfy certain conditions.

6. **Sorting**
   Sorting is a process of arranging all data items in a data structure in a particular order, say for example, either in ascending order or in descending order.

7. **Merging**
   Merging is a process of combining the data items of two different sorted list into a single sorted list.

8. **Splitting**
   Splitting is a process of partitioning single list to multiple list.

9. **Traversal**
   Traversal is a process of visiting each and every node of a list in systematic manner.

Time and space analysis of algorithms

Algorithm

- An essential aspect to data structures is algorithms.
- Data structures are implemented using algorithms.
- An algorithm is a procedure that you can write as a C function or program, or any other language.
- An algorithm states explicitly how the data will be manipulated.

Algorithm Efficiency

- Some algorithms are more efficient than others. We would prefer to choose an efficient algorithm, so it would be nice to have metrics for comparing algorithm efficiency.
- The complexity of an algorithm is a function describing the efficiency of the algorithm in terms of the amount of data the algorithm must process.
- Usually there are natural units for the domain and range of this function. There are two main complexity measures of the efficiency of an algorithm
- **Time complexity**
  - **Time Complexity** is a function describing the amount of time an algorithm takes in terms of the amount of input to the algorithm.

- **Space complexity**
  - **Space complexity** is a function describing the amount of memory (space) an algorithm takes in terms of the amount of input to the algorithm.
  - We often speak of "extra" memory needed, not counting the memory needed to store the input itself. Again, we use natural (but fixed-length) units to measure this.
  - We can use bytes, but it's easier to use, say, number of integers used, number of fixed-sized structures, etc. In the end, the function we come up with will be independent of the actual number of bytes needed to represent the unit.
  - Space complexity is sometimes ignored because the space used is minimal and/or obvious, but sometimes it becomes as important an issue as time.

Worst Case Analysis

In the worst case analysis, we calculate upper bound on running time of an algorithm. We must know the case that causes maximum number of operations to be executed. For Linear Search, the worst case happens when the element to be searched is not present in the array. When x is not present, the search () functions compares it with all the elements of array [] one by one. Therefore, the worst case time complexity of linear search would be.

Average Case Analysis

In average case analysis, we take all possible inputs and calculate computing time for all of the inputs. Sum all the calculated values and divide the sum by total number of inputs. We must know (or predict) distribution of cases. For the linear search problem, let us assume that all cases are uniformly distributed. So we sum all the cases and divide the sum by (n+1).

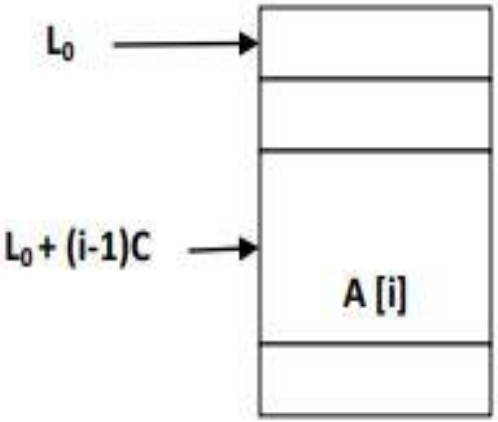
Best Case Analysis

In the best case analysis, we calculate lower bound on running time of an algorithm. We must know the case that causes minimum number of operations to be executed. In the linear search problem, the best case occurs when x is present at the first location. The number of operations in worst case is constant (not dependent on n). So time complexity in the best case would be.

Explain Array in detail

One Dimensional Array

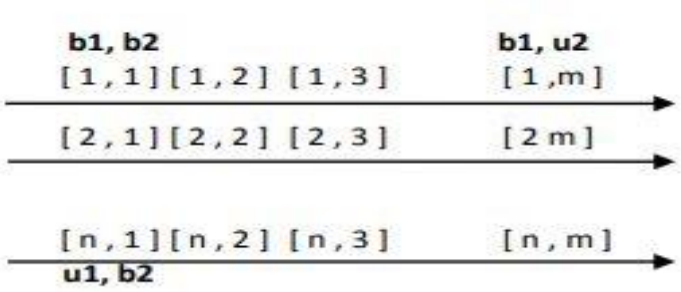
- Simplest data structure that makes use of computed address to locate its elements is the one-dimensional array or vector; number of memory locations is sequentially allocated to the vector.
- A vector size is fixed and therefore requires a fixed number of memory locations.
- Vector A with subscript lower bound of "one" is represented as below....



- L_0 is the address of the first word allocated to the first element of vector A.
- C words are allocated for each element or node
- The address of A_i is given equation **Loc (A_i) = L_0 + C (i-1)**
- Let's consider the more general case of representing a vector A whose lower bound for it's subscript is given by some variable b. The location of Ai is then given by **Loc (A_i) = L_0 + C (i-b)**

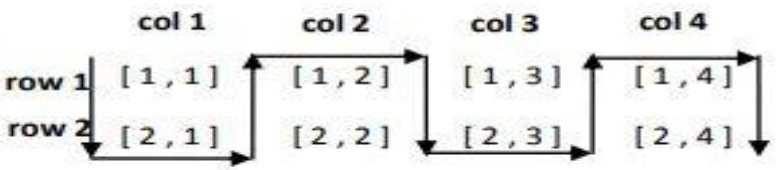
Two Dimensional Array

- Two dimensional arrays are also called table or matrix, two dimensional arrays have two subscripts
- Two dimensional array in which elements are stored column by column is called as column major matrix
- Two dimensional array in which elements are stored row by row is called as row major matrix
- First subscript denotes number of rows and second subscript denotes the number of columns
- Two dimensional array consisting of two rows and four columns as above Fig is stored sequentially by columns : A [1, 1], A [2 , 1], A [1 , 2], A [2 , 2], A [1 , 3], A [2 , 3], A [1, 4], A [2, 4]
- The address of element A [i , j] can be obtained by expression **Loc (A [i , j]) = L_0 + (j-1)*2 + i-1**
- In general for two dimensional array consisting of n rows and m columns the address element A [i , j] is given by **Loc (A [i , j]) = L_0 + (j-1)*n + (i – 1)**
- In row major matrix, array can be generalized to arbitrary lower and upper bound in its subscripts, assume that b1 ≤ I ≤ u1 and b2 ≤ j ≤u2



Row major matrix

No of Columns = m = u2 – b2 + 1



Column major matrix

- For row major matrix : **Loc (A [i , j]) = L_0 + (i – b1) *(u2-b2+1) + (j-b2)**

Applications of Array

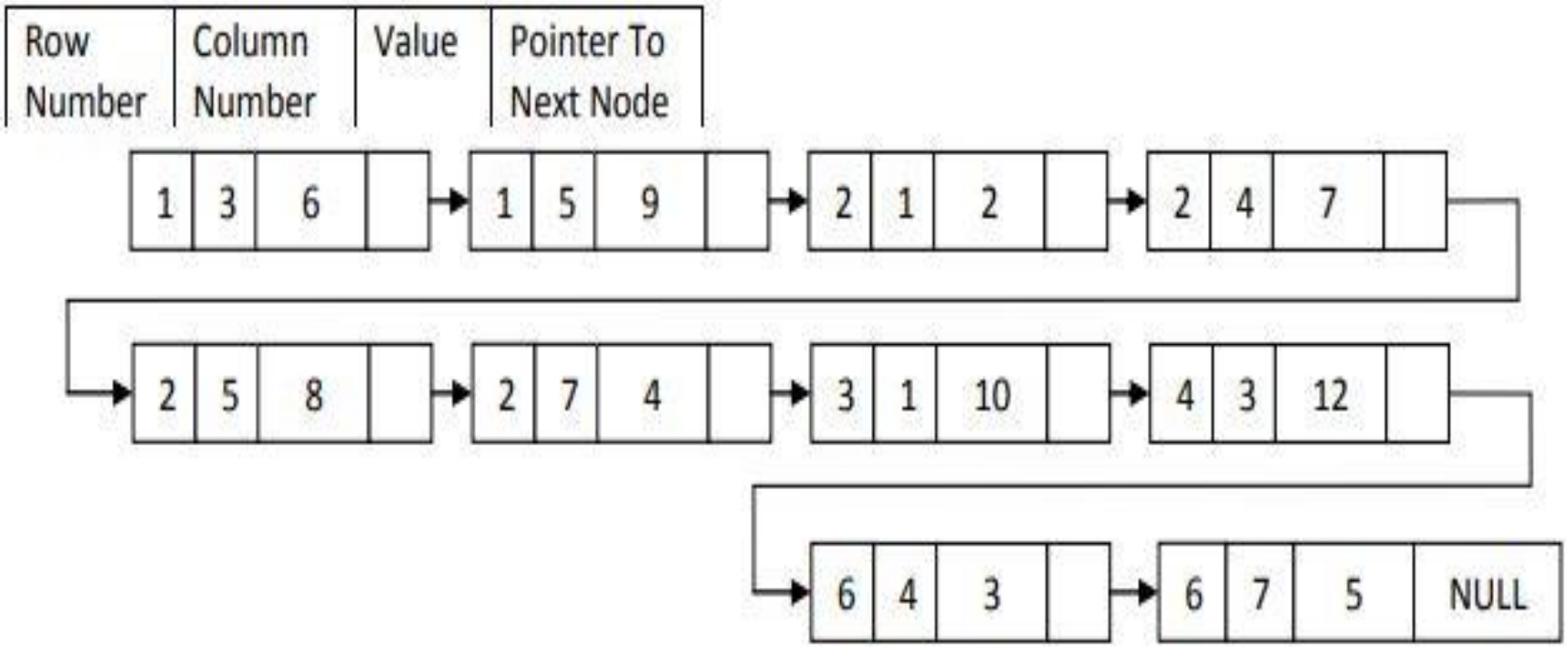
1. Symbol Manipulation (matrix representation of polynomial equation)
2. Sparse Matrix

What is sparse matrix? Explain

- An mXn matrix is said to be sparse if "many" of its elements are zero.
- A matrix that is not sparse is called a dense matrix.
- We can device a simple representation scheme whose space requirement equals the size of the non-zero elements.

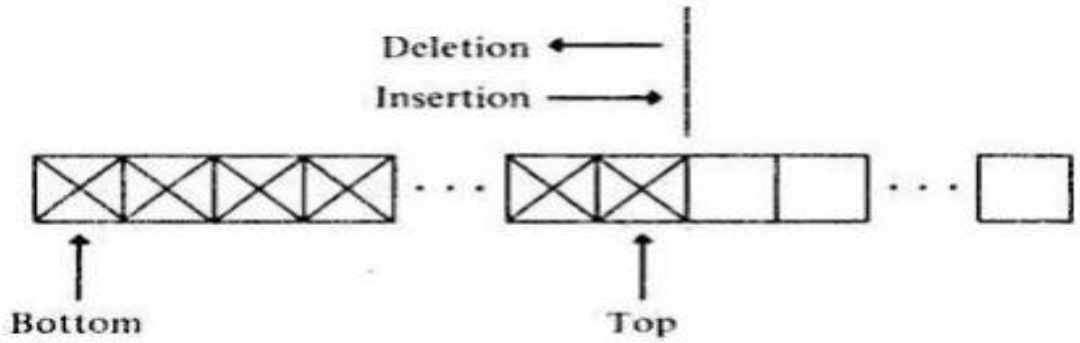
Linked Representation of Sparse matrix

Typical node to represent non-zero element is



Write algorithms for Stack Operations – PUSH, POP, PEEP

- A linear list which allows insertion and deletion of an element at one end only is called stack.
- The insertion operation is called as **PUSH** and deletion operation as **POP**.
- The most and least accessible elements in stack are known as top and bottom of the stack respectively.
- Since insertion and deletion operations are performed at one end of a stack, the elements can only be removed in the opposite orders from that in which they were added to the stack; such a linear list is referred to as a LIFO (last in first out) list.



Alternative representation of a stack

- A pointer TOP keeps track of the top element in the stack. Initially, when the stack is empty, TOP has a value of "one" and so on.
- Each time a new element is inserted in the stack, the pointer is incremented by "one" before, the element is placed on the stack. The pointer is decremented by "one" each time a deletion is made from the stack.

Applications of Stack

- Recursion
- Keeping track of function calls
- Evaluation of expressions
- Reversing characters
- Servicing hardware interrupts
- Solving combinatorial problems using backtracking.

Procedure: PUSH(S,TOP,X)

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

Function : POP (S, TOP)

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

1. [Check for underflow of stack]

   If TOP = 0

   Then Write ('STACK UNDERFLOW ON POP')

   Take action in response to underflow

   Return

2. [Decrement Pointer]

   TOP ← TOP – 1

3. [Return former top element of stack]

   Return (S[TOP + 1])

Give difference between recursion and iteration

Iteration	Recursion
In iteration, a problem is converted into a train of steps that are finished one at a time, one after another	Recursion is like piling all of those steps on top of each other and then quashing them all into the solution.
With iteration, each step clearly leads onto the next, like stepping stones across a river	In recursion, each step replicates itself at a smaller scale, so that all of them combined together eventually solve the problem.
Any iterative problem is solved recursively	Not all recursive problem can solved by iteration
It does not use Stack	It uses Stack

Trace the conversion of infix to postfix form in tabular form.

(i) (A + B * C / D - E + F / G / (H + I))

Input Symbol	Content of stack	Reverse polish	Rank
	(		0
(	((		0
A	((		0
+	((+	A	1
B	((+ B	A	1
*	((+ *	A B	2
C	((+ * C	A B	2
/	((+ /	A B C *	2
D	((+ / D	A B C *	2
-	((-	A B C * D / +	1
E	((- E	A B C * D / +	1
+	((+	A B C * D / + E -	1
F	((+ F	A B C * D / + E -	1
/	((+ /	A B C * D / + E – F	2
G	((+ / G	A B C * D / + E – F	2
/	((+ /	A B C * D / + E – F G /	2
(	((+ / (	A B C * D / + E – F G /	2
H	((+ / (H	A B C * D / + E – F G /	2
+	((+ / (+	A B C * D / + E – F G / H	3
I	((+ / (+ I	A B C * D / + E – F G / H	3
)	((+ /	A B C * D / + E – F G / H I +	3
)	(	A B C * D / + E – F G / H I + / +	1
)		A B C * D / + E – F G / H I + / +	1

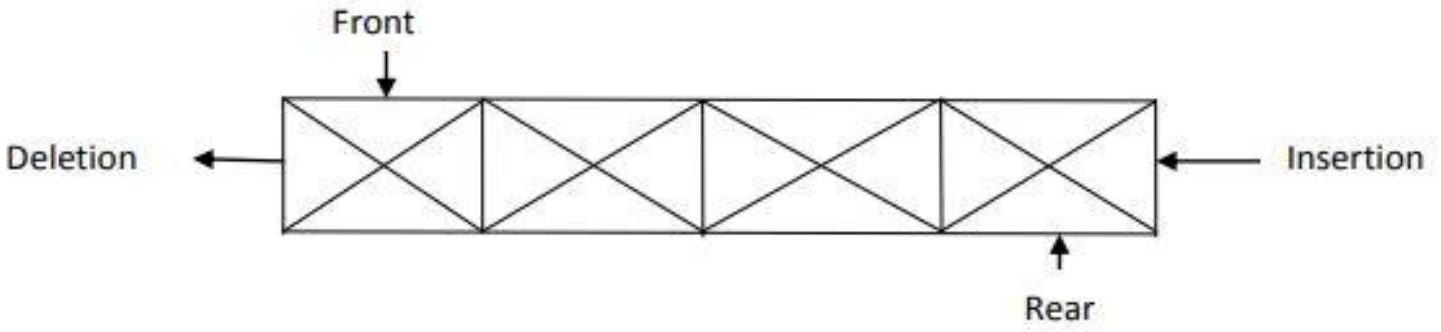
Postfix expression is: **A B C * D / + E – F G / H I + / +**

Explain Difference between Stack and Queue.

Stack	Queue
A Linear List Which allows insertion or deletion of an element at one end only is called as Stack	A Linear List Which allows insertion at one end and deletion at another end is called as Queue
Since insertion and deletion of an element are performed at one end of the stack, the elements can only be removed in the opposite order of insertion.	Since insertion and deletion of an element are performed at opposite end of the queue, the elements can only be removed in the same order of insertion.
Stack is called as Last In First Out (LIFO) List.	Queue is called as First In First Out (FIFO) List.
The most and least accessible elements are called as TOP and BOTTOM of the stack	Insertion of element is performed at FRONT end and deletion is performed from REAR end
Example of stack is arranging plates in one above one.	Example is ordinary queue in provisional store.
Insertion operation is referred as PUSH and deletion operation is referred as POP	Insertion operation is referred as ENQUEUE and deletion operation is referred as DQUEUE
Function calling in any languages uses Stack	Task Scheduling by Operating System uses queue

(i) Queue

- A linear list which permits deletion to be performed at one end of the list and insertion at the other end is called queue.
- The information in such a list is processed FIFO (first in first out) of FCFS (first come first served) pattern.
- Front is the end of queue from that deletion is to be performed.
- Rear is the end of queue at which new element is to be inserted.
- The process to add an element into queue is called **Enqueue**
- The process of removal of an element from queue is called **Dequeue**.
- The familiar and traditional example of a queue is Checkout line at Supermarket Cash Register where the first person in line is usually the first to be checkedout.

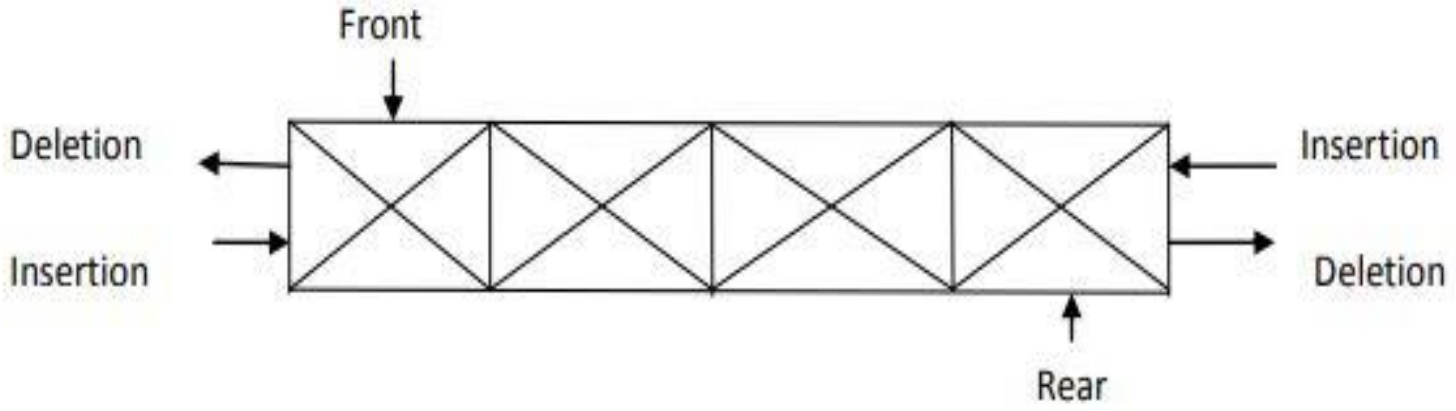


(ii) Circular Queue

- A more suitable method of representing simple queue which prevents an excessive use of memory is to arrange the elements Q[1], Q[2]….,Q[n] in a circular fashion with Q[1] following Q[n], this is called circular queue
- In a standard queue data structure re-buffering problem occurs for each **dequeue** operation. To solve this problem by joining the front and rear ends of a queue to make the queue as a circular queue
- Circular queue is a linear data structure. It follows FIFO principle.
- In circular queue the last node is connected back to the first node to make a circle.
- Circular linked list fallow the First In First Out principle
- Elements are added at the rear end and the elements are deleted at front end of the queue
- Both the front and the rear pointers points to the beginning of the array.
- It is also called as "Ring buffer".

(iii) Dequeue

- A dequeue (double ended queue) is a linear list in which insertion and deletion are performed from the either end of the structure.
- There are two variations of Dqueue
  - Input restricted dqueue- allows insertion at only one end
  - Output restricted dqueue- allows deletion from only one end
- Such a structure can be represented by following fig.



(iv) Priority Queue

- A queue in which we are able to insert remove items from any position based on some property (such as priority of the task to be processed) is often referred as priority queue.
- Below fig. represent a priority queue of jobs waiting to use a computer.
- Priorities of 1, 2, 3 have been attached with jobs of real time, online and batch respectively. Therefore if a job is initiated with priority i,it is inserted immediately at the end of list of other jobs with priorities i. Here jobs are always removed from the front of queue

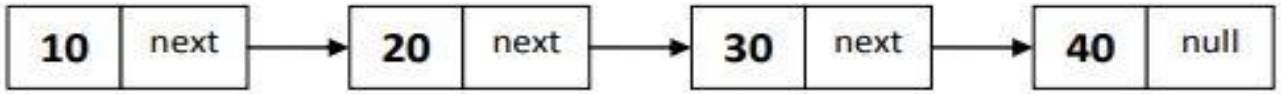
1. Linear Data Structure and their linked storage representation.

There are many applications where sequential allocation method is unacceptable because of following characteristics

- Unpredictable storage requirement
- Extensive manipulation of stored data

The linked allocation method of storage can result in both efficient use of computer storage and computer time.

- A linked list is a non-sequential collection of data items.
- The concept of a linked list is very simple, for every data item in the linked list, there is an associated pointer that would give the memory allocation of the next data item in the linked list.
- The data items in the linked list are not in a consecutive memory locations but they may be anywhere in memory.
- Accessing of these data items is easier as each data item contains within itself the address of the next data item.



A Linked List

2. What is linked list? What are different types of linked list? OR Write a short note on singly, circular and doubly linked list. OR Advantages and disadvantages of singly, circular and doubly linked list.

- A linked list is a collection of objects stored in a list form.
- A linked list is a sequence of items (objects) where every item is linked to the next.
- A linked list is a non-primitive type of data structure in which each element is dynamically allocated and in which elements point to each other to define a linear relationship.
- Elements of linked list are called nodes where each node contains two things, data and pointer to next node.
- Linked list require more memory compared to array because along with value it stores pointer to next node.
- Linked lists are among the simplest and most common data structures. They can be used to implement other data structures like stacks, queues, and symbolic expressions, etc...

Operations on linked list

- Insert
  - Insert at first position
  - Insert at last position
  - Insert into ordered list
- Delete
- Traverse list (Print list)
- Copy linked list

Types of linked list

Singly Linked List

- It is basic type of linked list.
- Each node contains data and pointer to next node.
- Last node's pointer is null.
- Limitation of singly linked list is we can traverse only in one direction, forward direction.

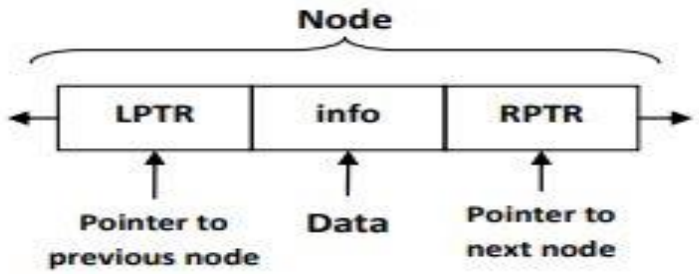
Singly Linked List

Circular Linked List

- Circular linked list is a singly linked list where last node points to first node in the list.
- It does not contain null pointers like singly linked list.
- We can traverse only in one direction that is forward direction.
- It has the biggest advantage of time saving when we want to go from last node to first node, it directly points to first node.
- A good example of an application where circular linked list should be used is a timesharing problem solved by the operating system.

Doubly Linked list

- Each node of doubly linked list contains data and two pointers to point previous (LPTR) and next (RPTR) node.




```
// C Structure to represent a node
struct node
{
        int info
        struct node *lptr;
        struct node *rptr;
};
```

- Main advantage of doubly linked list is we can traverse in any direction, forward or reverse.
- Other advantage of doubly linked list is we can delete a node with little trouble, since we have pointers to the previous and next nodes. A node on a singly linked list cannot be removed unless we have the pointer to its predecessor.
- Drawback of doubly linked list is it requires more memory compared to singly linked list because we need an extra pointer to point previous node.
- L and R in image denote left most and right most nodes in the list.
- Left link of L node and right link of R node is NULL, indicating the end of list for each direction.

3. Discuss advantages and disadvantages of linked list over array.

Advantages of an array

1. We can access any element of an array directly means random access is easy
2. It can be used to create other useful data structures (queues, stacks)

Disadvantages of an array

1. Its size is fixed
2. It cannot be dynamically resized in most languages
3. It is hard to add/remove elements
4. Size of all elements must be same.
5. Rigid structure (Rigid = Inflexible or not changeable)

Advantages of Linked List

1. **Linked lists are dynamic data structures:** That is, they can grow or shrink during execution of a program.
2. **Efficient memory utilization:** Here memory is not pre-allocated. Memory is allocated whenever it is required. And it is deallocated (free) when it is no longer needed.
3. **Insertion and deletions are easier and efficient:** Linked list provide flexibility in inserting a data item at a specified position and deletion of a data item from the given position.
4. **Elements of linked list are flexible:** It can be primary data type or user defined data types

Disadvantages of Linked List

1. Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists.
2. It cannot be easily sorted
3. We must traverse 1/2 the list on average to access any element
4. More complex to create than an array
5. Extra memory space for a pointer is required with each element of the list

3. What are the advantages and disadvantages of stack and queue implemented using linked list over array?

Advantages and disadvantages of stack & queue implemented using linked list over array is described below,

Insertion & Deletion Operation

- Insertion and deletion operations are known as push and pop operation in stack and as insert and delete operation in queue.
- In the case of an array, if we have n-elements list and it is required to insert a new element between the first and second element then n-1 elements of the list must be moved so as to make room for the new element.
- In case of linked-list, this can be accomplished by only interchanging pointers.
- Thus, insertion and deletions are more efficient when performed in linked list then array.

Searching a node

- If a particular node in a linked list is required, it is necessary to follow links from the first node onwards until the desired node is found.
- Where as in the case of an array, directly we can access any node

Join & Split

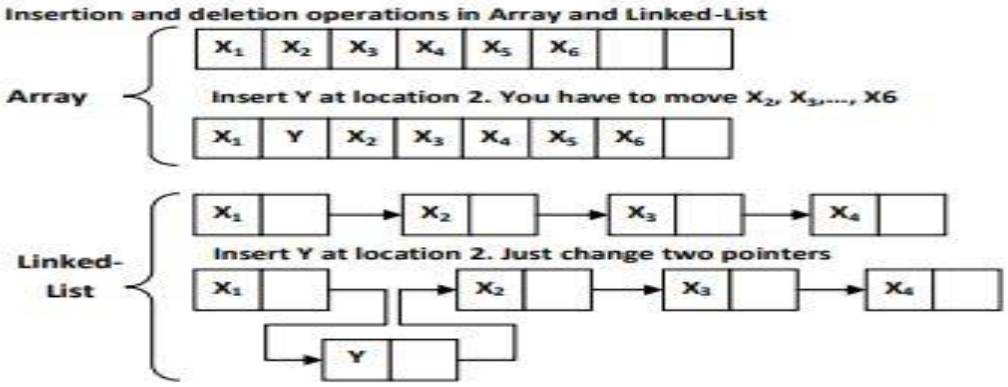
- We can join two linked list by assigning pointer of second linked list in the last node of first linked list.
- Just assign null address in the node from where we want to split one linked list in two parts.
- Joining and splitting of two arrays is much more difficult compared to linked list.

Memory

- The pointers in linked list consume additional memory compared to an array

Size

- Array is fixed sized so number of elements will be limited in stack and queue.
- Size of linked list is dynamic and can be changed easily so it is flexible in number of elements



Non-linear Data Structure(graph&tree)

1. **Graph**
   - A graph G consist of a non-empty set V called the set of nodes (points, vertices) of the graph, a set E which is the set of edges and a mapping from the set of edges E to a set of pairs of elements of V.
   - It is also convenient to write a graph as G=(V,E).
   - Notice that definition of graph implies that to every edge of a graph G, we can associate a pair of nodes of the graph. If an edge X ∈ E is thus associated with a pair of nodes (u,v) where u, v ∈ V then we says that edge x connect u and v.
2. **Adjacent Nodes**
   - Any two nodes which are connected by an edge in a graph are called adjacent node.
3. **Directed & Undirected Edge**
   - In a graph G=(V,E) an edge which is directed from one end to another end is called a directed edge, while the edge which has no specific direction is called undirected edge.
4. **Directed graph (Digraph)**
   - A graph in which every edge is directed is called directed graph or digraph.
5. **Undirected graph**
   - A graph in which every edge is undirected is called undirected graph.

6. Mixed Graph

- If some of the edges are directed and some are undirected in graph then the graph is called mixed graph.

7. Loop (Sling)

- An edge of a graph which joins a node to itself is called a loop (sling).

8. Parallel Edges

- In some directed as well as undirected graphs, we may have certain pairs of nodes joined by more than one edges, such edges are called Parallel edges.

9. Multigraph

- Any graph which contains some parallel edges is called multigraph.

10. Weighted Graph

- A graph in which weights are assigned to every edge is called weighted graph.

11. Isolated Node

- In a graph a node which is not adjacent to any other node is called isolated node.

12. Null Graph

- A graph containing only isolated nodes are called null graph. In other words set of edges in null graph is empty.

13. Path of Graph

- Let G=(V, E) be a simple digraph such that the terminal node of any edge in the sequence is the initial node of the edge, if any appearing next in the sequence defined as path of the graph.

14. Length of Path

- The number of edges appearing in the sequence of the path is called length of path.

15. Degree of vertex

- The no of edges which have V as their terminal node is call as indegree of node V
- The no of edges which have V as their initial node is call as outdegree of node V
- Sum of indegree and outdegree of node V is called its Total Degree or Degree of vertex.

16. Simple Path (Edge Simple)

- A path in a diagraph in which the edges are distinct is called simple path or edge simple.

17. Elementary Path (Node Simple)

- A path in which all the nodes through which it traverses are distinct is called elementary path.

18. Cycle (Circuit)

- A path which originates and ends in the same node is called cycle (circuit).

19. Directed Tree

- A directed tree is an acyclic digraph which has one node called its root with in degree 0, while all other nodes have in degree 1.
- Every directed tree must have at least one node.
- An isolated node is also a directed tree.

20. Terminal Node (Leaf Node)

- In a directed tree, any node which has out degree 0 is called terminal node or leaf node.

21. Level of Node

- The level of any node is the length of its path from the root.

22. Ordered Tree

- In a directed tree an ordering of the nodes at each level is prescribed then such a tree is called ordered tree.

23. Forest

- If we delete the root and its edges connecting the nodes at level 1, we obtain a set of disjoint tree. A set of disjoint tree is a forest.

24. M-ary Tree

- If in a directed tree the out degree of every node is less than or equal to m then tree is called an m-ary tree.

25. Full or Complete M-ary Tree

- If the out degree of each and every node is exactly equal to m or 0 and their number of nodes at level **i** is $\mathbf{m}^{(i-1)}$ then the tree is called a full or complete m-ary tree.

26. Positional M-ary Tree

- If we consider m-ary trees in which the m children of any node are assumed to have m distinct positions, if such positions are taken into account, then tree is called positional m-ary tree.

27. Height of the tree

- The height of a tree is the length of the path from the root to the deepest node in the tree.

28. Binary tree

- If in a directed tree the out degree of every node is less than or equal to 2 then tree is called binary tree.

29. Strictly binary tree

- A strictly binary tree (sometimes proper binary tree or 2-tree or full binary tree) is a tree in which every node other than the leaves has two children.

30. Complete binary tree

- If the out degree of each and every node is exactly equal to 2 or 0 and their number of nodes at level **i** is $2^{(i-1)}$ then the tree is called a full or complete binary tree.

31. Sibling

- Siblings are nodes that share the same parent node.

32. Binary search tree

- A binary search tree is a binary tree in which each node possessed a key that satisfy the following conditions

1. All key (if any) in the left sub tree of the root precedes the key in the root.
2. The key in the root precedes all key (if any) in the right sub tree.
3. The left and right sub tree sub trees of the root are again search trees.

33. Height Balanced Binary tree (AVL Tree)

- A tree is called AVL (height balance binary tree), if each node possesses one of the following properties
  1. A node is called left heavy if the longest path in its left sub tree is one longer then the longest path of its right sub tree.
  2. A node is called right heavy if the longest path in the right sub tree is one longer than path in its left sub tree.
  3. A node is called balanced, if the longest path in both the right and left sub tree are equal.

Explain the Preorder, Inorder and Postorder traversal techniques of the binary tree with suitable example.

- The most common operations performed on tree structure is that of traversal. This is a procedure by which each node in the tree is processed exactly once in a systematic manner.
- There are three ways of traversing a binary tree.
  1. Preorder Traversal
  2. Inorder Traversal
  3. Postorder Traversal

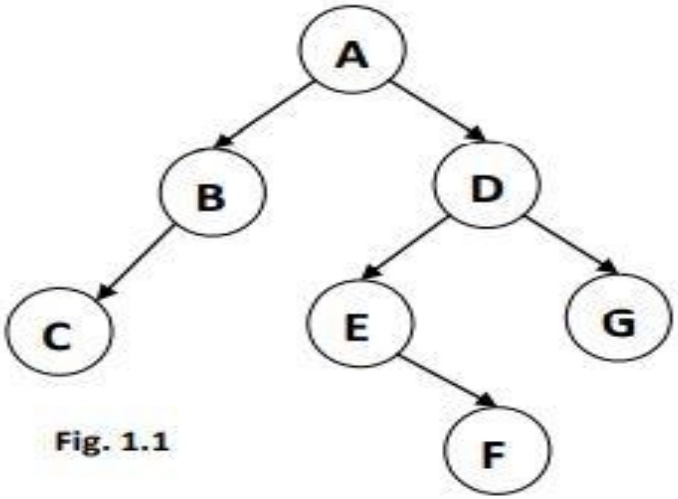


Fig. 1.1

Preorder traversal : A B C D E F G

Inorder traversal : C B A E F D G

Postorder traversal : C B F E G D A

Converse Preorder traversal : A D G E F B C

Converse Inorder traversal : G D F E A B C

Converse Postorder traversal : G F E D C B A

Preorder

- Preorder traversal of a binary tree is defined as follow
  - Process the root node
  - Traverse the left subtree in preorder
  - Traverse the right subtree in preorder
- If particular subtree is empty (i.e., node has no left or right descendant) the traversal is performed by doing nothing, In other words, a null subtree is considered to be fully traversed when it is encountered.
- The preorder traversal of a tree (Fig. 1.1) is given by A B C D E F G

Inorder

- The Inorder traversal of a binary tree is given by following steps,
  - Traverse the left subtree in Inorder
  - Process the root node
  - Traverse the right subtree in Inorder
- The Inorder traversal of a tree (Fig. 1.1) is given by C B A E F D G

Postorder

- The postorder traversal is given by
  - Traverse the left subtree in postorder
  - Traverse the right subtree in postorder
  - Process the root node
- The Postorder traversal of a tree (Fig. 1.1) is given by C B F E G D A

Converse ...

- If we interchange left and right words in the preceding definitions, we obtain three new traversal orders which are called
  - Converse Preorder (A D G E F B C)
  - Converse Inorder (G D F E A B C)
  - Converse Postorder (G F E D C B A)

Threaded Binary Tree

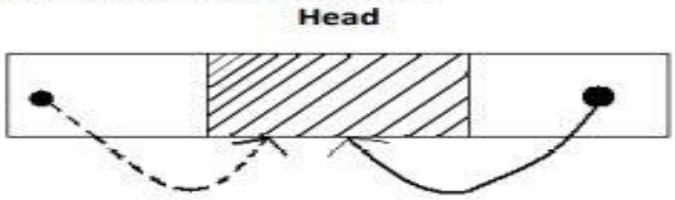
- The wasted NULL links in the binary tree storage representation can be replaced by threads.
- A binary tree is threaded according to particular traversal order. e.g.: Threads for the inorder traversals of tree are pointers to its higher nodes, for this traversal order.
  - If left link of node P is null, then this link is replaced by the address of its predecessor.
  - If right link of node P is null, then it is replaced by the address of its successor
- Because the left or right link of a node can denote either structural link or a thread, we must somehow be able to distinguish them.
- Method 1:- Represent thread a –ve address.
- Method 2:- To have a separate Boolean flag for each of left and right pointers, node structure for this is given below,

LPTR	LTHREAD	Data	RTHREAD	RPTR

Alternate node for threaded binary tree.

- LTHREAD = true = Denotes leaf thread link
- LTHREAD = false = Denotes leaf structural link
- RTHREAD = true = Denotes right threaded link
- RTHREAD = false = Denotes right structural link

- Head node is simply another node which serves as the predecessor and successor of first and last tree nodes. Tree is attached to the left branch of the head node



Advantages

- Inorder traversal is faster than unthreaded version as stack is not required.
- Effectively determines the predecessor and successor for inorder traversal, for unthreaded tree this task is more difficult.
- A stack is required to provide upward pointing information in tree which threading provides.
- It is possible to generate successor or predecessor of any node without having over head of stack with the help of threading.

Disadvantages

- Threaded trees are unable to share common subtrees

What is graph? How it can be represented using adjacency matrix, what is path matrix? How path matrix can be found out using adjacency matrix .

Graph

- A graph G consist of a non empty set V called the set of nodes (points, vertices) of the graph, a set E which is the set of edges and a mapping from the set of edges E to a set of pairs of elements of V.
- It is also convenient to write a graph as G=(V,E).
- Notice that definition of graph implies that to every edge of a graph G, we can associate a pair of nodes of the graph. If an edge X ∈ E is thus associated with a pair of nodes (u,v) where u, v ∈ V then we says that edge x connect U and V.

Adjacency matrix

Let G = (V, E) be a simple diagraph in which V = $\{v_1, v_2, \ldots, v_n\}$ and the nodes are assumed to be ordered from v_1 to v_n. An n x n matrix A whose elements are a_{ij} are given by

$$a_{ij} = \begin{cases} 1 & if (Vi, Vj) \in E \\ 0 & otherwise \end{cases}$$

is called adjacency matrix of the graph G.

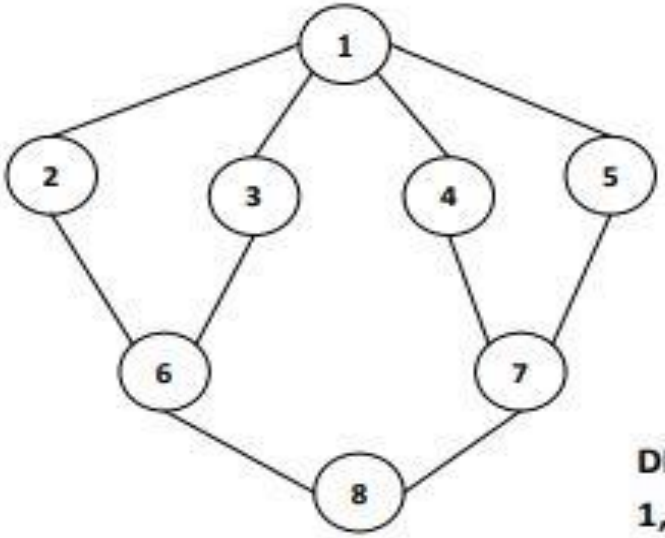
- Any element of the adjacency matrix is either 0 or 1.
- For a given graph G =m (V, E), an adjacency matrix depends upon the ordering of the elements of V.
- For different ordering of the elements of V we get different adjacency matrices.

Which are the basic traversing techniques of the Graph? Write the algorithm of them.

- Most graph problems involve traversal of a graph. Traversal of a graph means visit each node exactly once.
- Two commonly used graphs traversal techniques are
  1. Depth First Search (DFS)
  2. Breadth First Search (BFS)

Depth First Search (DFS)

- It is like preorder traversal of tree.
- Traversal can start from any vertex vi
- Vi is visited and then all vertices adjacent to vi are traversed recursively using DFS



Graph G

DFS (G, 1) is given by

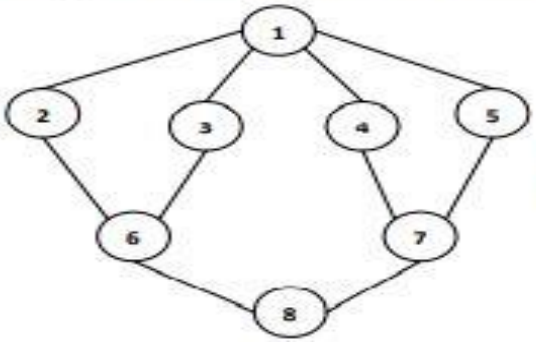
a) Visit (1)
b) DFS (G, 2)
DFS (G, 3)
DFS (G, 4)
DFS (G, 5)

DFS traversal of given graph is:
1, 2, 6, 3, 8, 7, 4, 5

- Since graph can have cycles, we must avoid re-visiting a node. To do this when we visit a vertex V, we marks it visited as visited should not be selected for traversal.

Breadth First Search (BFS)

- This methods starts from vertex v_0
- V_0 is marked as visited. All vertices adjacent to v_0 are visited next
- Let vertices adjacent to v_0 are v_1, v_2, v_3, v_4
- v_1, v_2, v_3 and v_4 are marked visited.
- All unvisited vertices adjacent to v_1, v_2, v_3, v_4 are visited next.
- The method continuous until all vertices are visited
- The algorithm for BFS has to maintain a list of vertices which have been visited but not explored for adjacent vertices. The vertices which have been visited but not explored for adjacent vertices can be stored in queue.
- Initially the queue contains the starting vertex.
- In every iteration, a vertex is removed from the queue and its adjacent vertices which are not visited as yet are added to the queue.
- The algorithm terminates when the queue becomes empty.

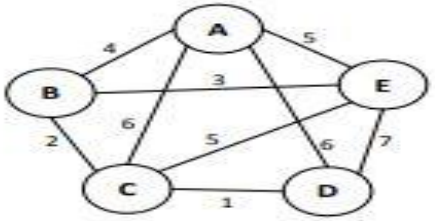


Graph G

BFS traversal of given graph is:
1 | 2, 3, 4, 5 | 6, 7 | 8

What is spanning tree?

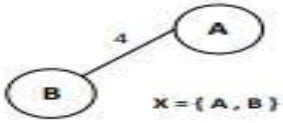
- A Spanning tree of a graph is an undirected tree consisting of only those edges necessary to connect all the nodes in the original graph
- A spanning tree has the properties that
  - For any pair of nodes there exists only one path between them
  - Insertion of any edge to a spanning tree forms a unique cycle
- The particular Spanning for a graph depends on the criteria used to generate it.
- If DFS search is use, those edges traversed by the algorithm forms the edges of tree, referred to as Depth First Spanning Tree.
- If BFS Search is used, the spanning tree is formed from those edges traversed during the search, producing Breadth First Search Spanning tree.



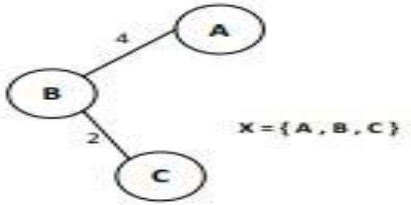
Using Prim's Algorithm:

Let X be the set of nodes explored, initially X = { A }

Step 1: Taking minimum weight edge of all Adjacent edges of **X = { A }**

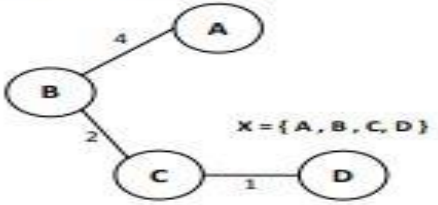


Step 2: Taking minimum weight edge of all Adjacent edges of **X = { A , B }**

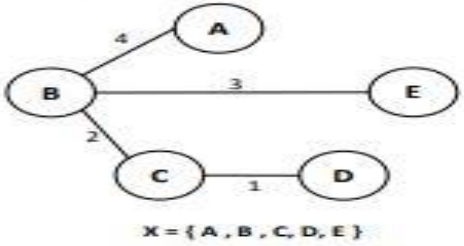


~~A – B | 4~~
A – E | 5
A – C | 6
A – D | 6
~~B – E | 3~~
~~B – C | 2~~
C – E | 6
~~C – D | 1~~
D – E | 7

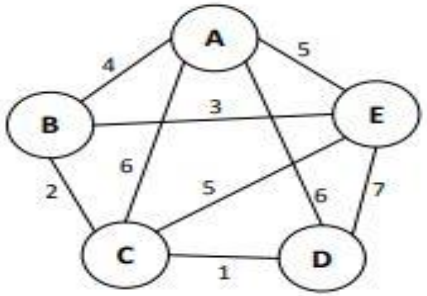
Step 3: Taking minimum weight edge of all Adjacent edges of **X = { A , B , C }**



Step 4: Taking minimum weight edge of all Adjacent edges of **X = { A , B , C , D }**

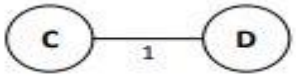


All nodes of graph are there with set X, so we obtained minimum spanning tree of cost: **4 + 2 + 1 + 3 = 10**

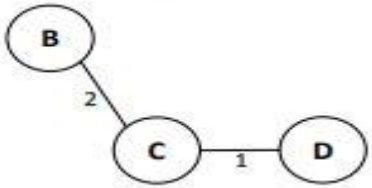


Using Kruskal's Algorithm

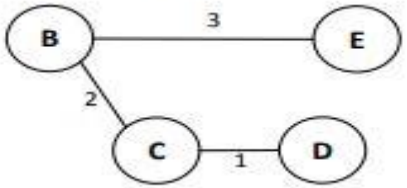
Step 1: Taking min edge (C,D)



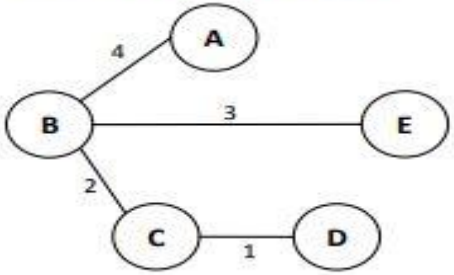
Step 2: Taking next min edge (B,C)



Step 3: Taking next min edge (B,E)



Step 4: Taking next min edge (A,B)



Step 5: Taking next min edge (A,E) it forms cycle so do not consider
Step 6: Taking next min edge (C,E) it forms cycle so do not consider
Step 7: Taking next min edge (A,D) it forms cycle so do not consider
Step 8: Taking next min edge (A,C) it forms cycle so do not consider
Step 9: Taking next min edge (E,D) it forms cycle so do not consider

All edges of graph has been visited,
so we obtained minimum spanning tree of cost:
4 + 2 + 1 + 3 = 10