3/1/2021

DSA - assignment 2

Name: Manvitha Vemulapalli
Rollno: 19131A0586
SEC: CSE-4

UNIT-3

1. a) Write a program for the implementation of double linked list.

C++ program to implement doubly linked list is as follows;

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;
Struct node
{
  Struct node *prev;
  int data;
  Struct node *next;
};
Struct node *head= NULL, *newnode, *head1;
void begininsert();
void endinsert();
void afterinsert();
void beforeinsert();
void begindelete();
void enddelete();
void deletenode();
void display();
int main()
{
  int ch;
 while(1)
 {
 cout<<"\n LIST : \n 1) insert at the beginning \n 2) insert at the ending \n 3) insert after a given node \n 4) insert before a given node \n 5) delete front node \n 6) delete end note \n 7) delete given node \n 8) display \n 9) exit \n"<<endl;
cin>>ch;
```

```
switch(ch)
{
case 1: begininsert();
        break;
case 2: endinsert();
        break;
case 3: afterinsert();
        break;
case 4: beforeinsert();
        break;
case 5: begindelete();
        break;
case 6: enddelete();
        break;
case 7: deletenode();
        break;
case 8: display();
        break;
case 9: exit(0);
  }
}
  return 0;
}
void begininsert()
{
  node* newnode = new node();
  cout << "\n enter element to be inserted" << endl;
  cin >> newnode -> data;
  newnode -> prev = NULL;
  newnode -> next = NULL;
  if (head == NULL)
  {
    head = head1 = newnode;
  }
 else
  {
    newnode -> next = head;
```

```
head->prev=newnode;
head=newnode;
}
}
void endinsert()
{
node * newnode=new node();
cout<<"\n enter element to be
                inserted"<<endl;
cin>>newnode->data;
newnode->next=NULL;
newnode->prev=NULL;
if(head==NULL)
{
head=head1=newnode;
}
else
{
head1=head;
while(head1->next!=NULL)
{
head1=head1->next;
}
head1->next=newnode;
newnode->prev=head1;
}
}
void afterinsert()
{
int num;
node * newnode=new node();
Cout<<"\n enter element to be
inserted"<<endl;
Cin>>newnode->data;
Cout<<"\n after which element";
Cin>>num;
newnode->next=newnode->prev
                =NULL;
if(head==NULL)
{
cout<<"Invalid"<<endl;
}
else
{
head1=head;
while(head1->data!=num
{
head1=head1->next;
}
newnode->next=head1->next
newnode->prev=head1;
head1->next=newnode;
if(newnode->next!=NULL)
newnode->next->prev=
                newnode;
}
}
void beforeinsert()
{
int num;
node* newnode=new node();
Cout<<"\nenter element to
be inserted"<<endl;
Cin>>newnode->data;
newnode->next=newnode->
                prev=NULL;
Cout<<"\n before which
element"<<endl;
Cin>>num;
if(head==NULL)
{
Cout<<"Invalid"<<endl;
}
```

```
else
{
head1=head;
while(head1->data!=num)
{
head1=head1->next;
}
newnode->next=head1;
newnode->prev=head1->next;
head1->prev=>next=newnode;
head1->prev=newnode;
}
}

void begindelete()
{
  head1=head;
 head=head->next;
if(head!=NULL)
head->prev=NULL;
free(head1);
}
void enddelete()
{
 head1=head;
if(head1->next!=NULL)
{
while(head1->next!=NULL)
{
  head1=head1->next;
}
head1->prev->next=NULL;
}
else
head=NULL;
free(head1);
}

void deletenode()
{
  int num;
  head1=head;
Cout<<"\nenter node to be
      deleted"endl;
Cin>>num;
while(head1->data!=num)
{
  head1=head1->next;
  if(head1==NULL)
  {
Cout<<"\nelement not found";
  goto x;
  }
}
if(head1->next!=NULL)
{
head1->next->prev=head1->
                    prev;
}
if(head1->prev!=NULL)
{
head1->prev->next=head1->
                    next;
}
else
{
  head=head1->next;
}
x: free(head1);
}
```

```
void display()
{
  if(head==NULL)
   {
   cout<<"\n nothing to display";

   }
 else
   {
    head1=head;
    while(head1!=NULL)
     {
     cout<<head1->data<<endl;
     head1=head1->next;
     }
    }
}
```

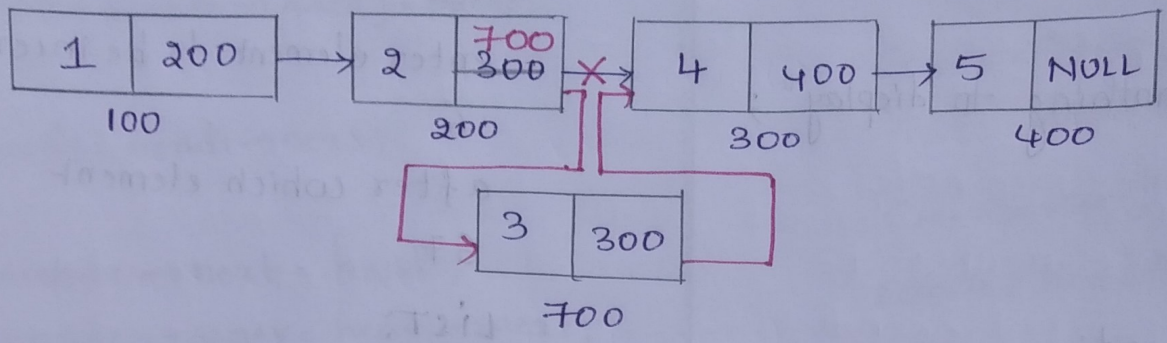
output:-

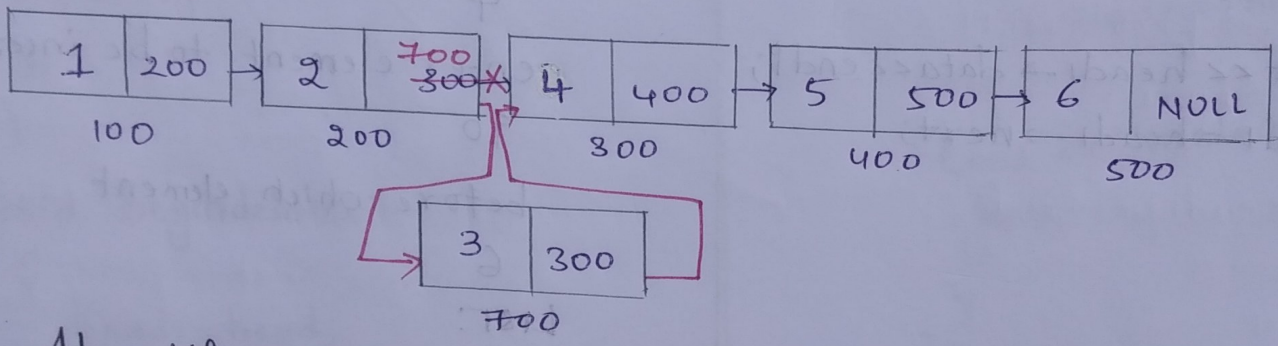
```
LIST:
 1) insert at the beginning
2) insert at the ending
3) insert after a given node
4) insert before a given node
5) delete front node
6) delete end node
7) delete given node
8) display
9) exit
1
enter element to be inserted
34
List:
 |
2
enter element to be inserted
67
List:
 |
 |
3
enter element to be inserted
6
after which element
67
List:
 |
 |
 |
4
enter element to be inserted
90
before which element
6
LIST:
 |
 |
 |
8

34
23
67
90
6
LIST:
 |
 |
5
LIST:
 |
 |
6
LIST:
 |
 |
7
enter node to be deleted
90
LIST:
 |
 |
8

23
67
9
PS D:\DSA>
```

b) Explain how to insert a node in the middle of a Singly linked list





Algorithm:-

Step1:- Assign head1 (A temporary pointer) to head

Step2:- Repeat the steps until head1 != NULL
i++
then step3

Step3:- len is the no.of nodes at that instant.
if(i==len/2) otherwise step4.
i, allocate memory dynamically to newnode of type struct node
ii, Read data (newnode→data) from the user
iii, new_node→next = head1→next; and
head1→next = newnode
then apply break statement to exit from while loop

Step4: assign head1 to its next value
head1 = head1→next;

Step5: display and exit

Program implementation in C++;

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;
Struct node
{
 int data;
 Struct node* next;
};
Struct node *head = NULL, *newnode, *head1;
int main()
{
 //creation of linked list
 int ch=1, len=0, i=0;
 while(ch)
 {
  len++;
  node* newnode = new node();
  Cout << "\nenter any value" << endl;
  Cin >> newnode → data;
  newnode → next = NULL;
  if (head == NULL)
  {
   head = head1 = newnode;
  }
  else
  {
   head1 → next = newnode;
   head1 = newnode;
  }
  Cout << "\n if you want to Continue enter 1 else 0" << endl;
  Cin >> ch;
 }
 //insertion at the middle of a Singly linked list
 head1 = head;
 while(head1 != NULL)
```

```
{
    i++;
  if(i=len/2)
  {
   node *newnode=new node();
   cout<<"\n this value will be inserted in "<<len/2+1<<
                                          " position"<<endl;

   cin>>newnode->data;
   newnode->next=head1->next;
   head1->next=newnode;
   break;
  }
  else
   {
     head1=head1->next;
   }
}
// display
head1=head;
while(head1!=NULL)
 {
  cout<<head1->data<<endl;
  head1=head1->next;
  }
 return 0;
}
```

2) a) What are the advantages and disadvantages of linked list?

Advantages of linked list:-

Dynamic data structure: A linked list is a dynamic arrangement so it can grow and shrink at run time by allocating & deallocating memory. So there is no need to give the initial size of linked list.

No memory wastage: In the linked list, efficient memory utilization can be acheived since the size of the linked list increase or decrease at run time so there is no memory wastage and there is no need to pre allocate the memory.

Implementation: Linear data structures like stack & queues are often easily implemented using a linked list.

Insertion & deletion operations: These operations are quite easier in the linked list. There is no need to shift other elements only address present in the next pointer is to be updated.

Disadvantages of linked list:-

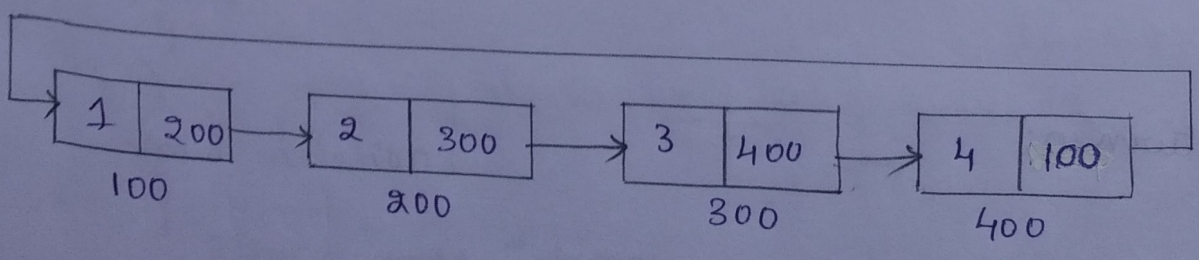
Memory Usage: more memory is required in the linked list as Compared to an array. Because in a linked list a pointer is also required to store the address of the next element and it requires extra memory foritself.

Traversal: This is time Consuming Compared to an array Direct access is not possible by index.

Reverse traversing: for Singly linked list reverse traversing isn't possible, where as in doubly linked list itis possible by using another pointer to node, here is the wastage of memory.

Random access: random access isn't possible due to dynamic memory allocation.

b) Write a program for circular linked list.



C++ program to implement circular linked list

```
#include<bits/stdc++.h>
#include<iostream>
using namespace std;
Struct node
{
  int data;
  struct node *next;
}*head=NULL,*head1,*newnode,*tail=NULL;
int len=0;
void insertion();
void deletion();
void display();
int main()
{
  int ch;
  while(1)
  {
  cout<<"\nLIST:\n1) Insertion at given location\n2) deletion
  at given location\n3) display\n4) exit"<<endl;
   cin>>ch;
   switch(ch)
   {
     case 1: insertion();
             break;
     case 2: deletion();
             break;
     case 3: display();
             break;
     Case 4: exit(0);
   }
  }
  return 0;
}
```

```
void insertion()
{
  int n;
  cout<< "\n no.of nodes are"<<len<<": enter position to
                                    insert"<<endl;

  cin>>n;
  head1=head;
  int i=1;
  while(i<n-1)
  {
    i++;
    head1=head1->next;
  }
  node *newnode=new node();
  cout<<"enter data to be inserted:"<<endl;
  cin>>newnode->data;
  len++;
  if(n==1)
  {
    head=newnode;
    newnode->next=head;
   if(tail!=NULL)
   tail->next=newnode;
   else
   tail=newnode;
  }
else
  {
   newnode->next=head1->next;
    head1->next=newnode;
    if(head1==tail)
     {
       tail=newnode;
     }
  }
 }
 void deletion()
  {
    Struct node *temp;
```

```
int n;
cout<<"\n no.of nodes are"<< len<<"; enter position to
                                    be deleted"<<endl;
cin>>n;
head1=head;
int i=0;
while (i<n-2)
{
  i++;
  head1 = head1->next;
}
len--;
if(n==1)
{
  if(len!=0)
   {
     head=head->next;
     tail ->next=head;
   }
  free(head1);
}
else
 {
   temp =head1->next;
   head1->next= head1->next->next;
  if ( head1 ->next==tail)
   { tail=head1;
  }
  free(temp);
 }
}
void  display ()
{
  if (len==0)
  { cout<<" there is ntmg to display";
  }
  else
  {
    head1 =head;
```

```
do
{
cout << head1 -> data << endl;
head1 = head1 -> next;
} while (head1 != head);
}
}
```

output:-

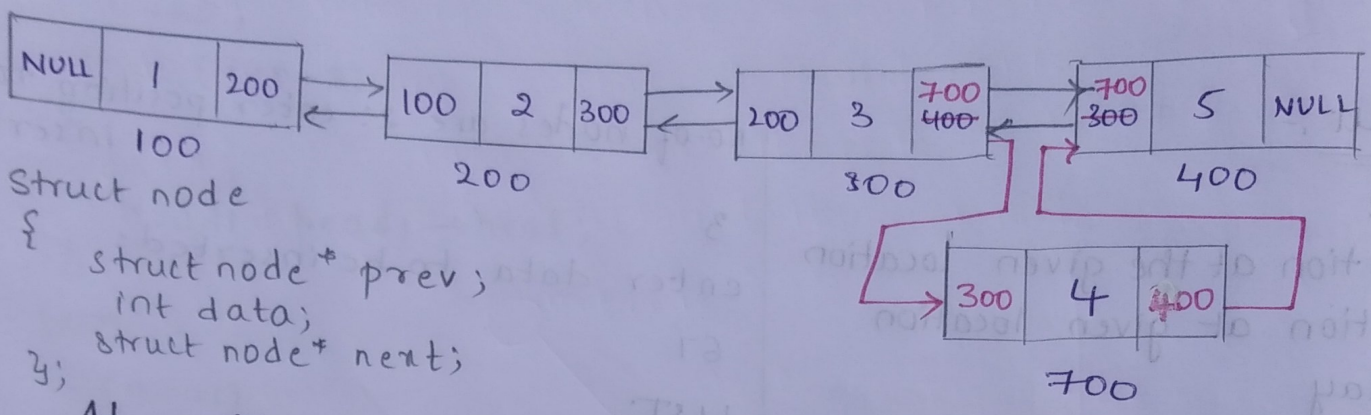
```
LIST:
1) insertion at the given location
2) deletion at given location
3) display
4) exit
1
no.of nodes are 0: enter position
                   to insert
1
enter data to be inserted:
45
LIST:
1
2
no.of nodes are 1: enter position
to be deleted
1
LIST:
1
3
there is ntng to display
LIST:
1
no.of nodes are 0: enterposition
                   to insert
2
enter data to be inserted:
45
LIST:
1
1
no.of nodes are 2: enter position to
                   insert
3
enter data to be inserted:
67
LIST:
1
3
45
67
LIST:
1
4
PS D:\DSA>
```

3) Given an algorithm to perform following operations in a double linked list.

(a) Insert a new node after a given node

(b) Delete last node.

(a)




```
Struct node
{
    struct node * prev;
    int data;
    struct node* next;
};
```

Algorithm:-

Step1: allocate memory to newnode of type struct node
(node * newnode = new node();)

Step2: Read data into the node and num of int type
(cin >> newnode → data)

Step3: Initialise newnode → next & new-node → prev to NULL
(newnode → next = newnode → prev = NULL;)

Step4: if (head == NULL) print Invalid and stop.

Step5: Initialise head1 (temporary pointer) with head

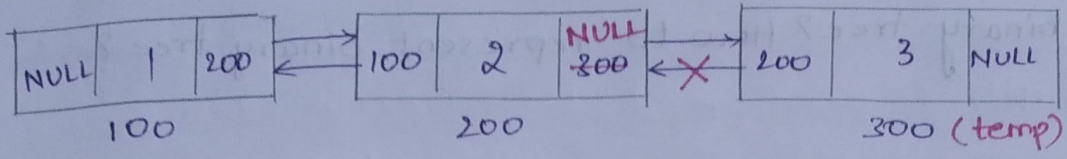
Step6: Repeat the head1 = head1 → next until the condition head1 → data != num becomes false.

Step7:
```
newnode → next = head1 → next;
newnode → prev = head1;
head1 → next = newnode;
```

Step8: if (newnode → next != NULL) do the following assignment
newnode → next → prev = newnode;

Step9: display & exit.

(b)



In order to delete last node there are two ways

(1) to set a tail pointer to last node while creation which makes the time complexity O(1)

(2) Traverse till end — O(n)

Here is the 2nd method algorithm.

Algorithm:-

Step 1: Initialise head1 (temporary pointer) with head

Step 2: if(head1→next!=NULL) evaluate step 3 otherwise goto step 5

Step 3: Repeat head1 = head1→next until head1→next!=NULL evaluates false.

Step 4: head1→prev→next = NULL;

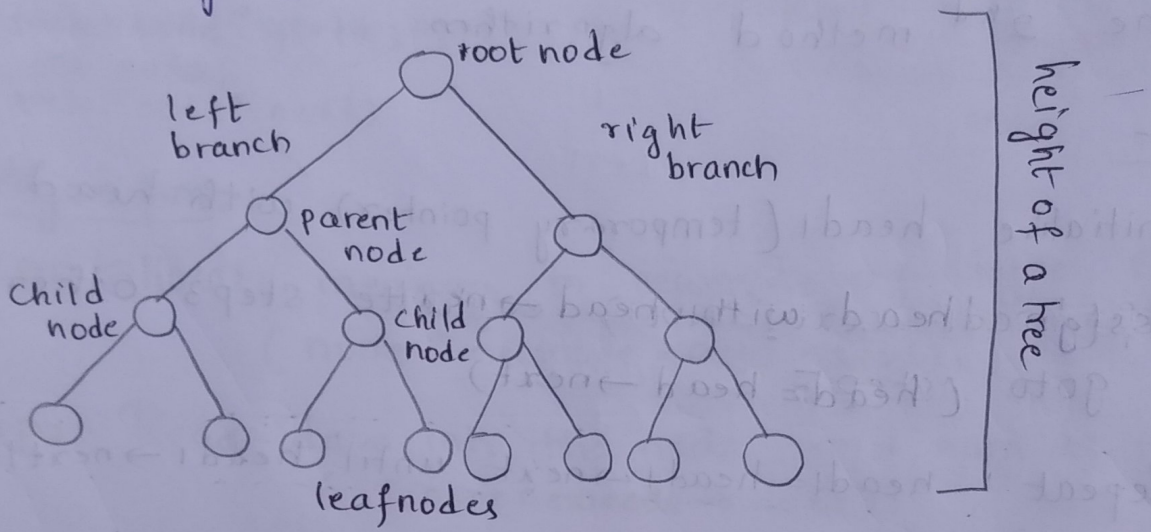
Step 5: head = NULL

Step 6: deallocate the memory of head1 (free(head1);)

UNIT-4

1) a) What is a binary tree? How to represent binary tree? Explain.

Ans:- A binary tree is a tree type non-linear data structure with a maximum of 2 children for each parent. Every node in a binary tree has a left and right reference along with the data element. The node at the top of the hierarchy of a tree is called the root node.



Terminologies associated with Binary trees & its types

Node: it represents a termination point in a tree

Root: A trees topmost node

Parent: Each node (apart from root) in a tree that has atleast one subnode of its own is called parent node

Child: A node that straightway came from a parent node when moving away from the root is child node.

Leaf nodes: These are external nodes, these are the nodes that have no child

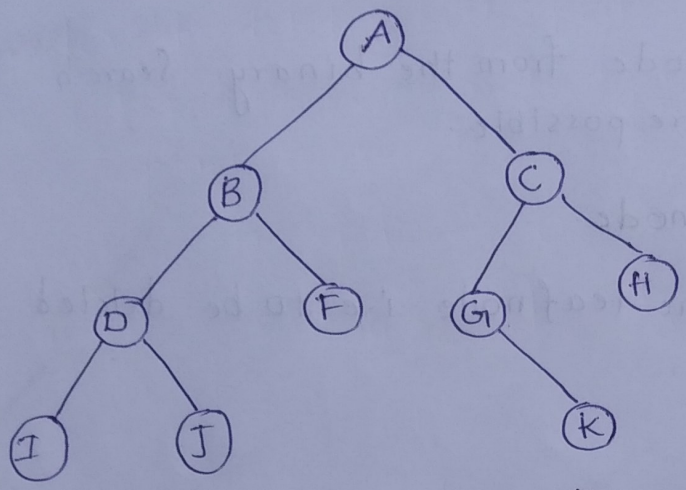
Internal node: These are internal nodes with atleast one child

Depth of a Tree: The number of edges from the trees node to root is

Height of a tree: This is the no. of edges from the node to the deepest leaf also known as root height

Binary tree representations:

1) Array representation
2) Linked list representation



1) Array representation of Binary Tree.

Here we use 1-D array, the above Binary tree is represented as

A	B	C	D	F	G	H	I	J	–	–	–	K	–	–

–	–	–	–	–	–

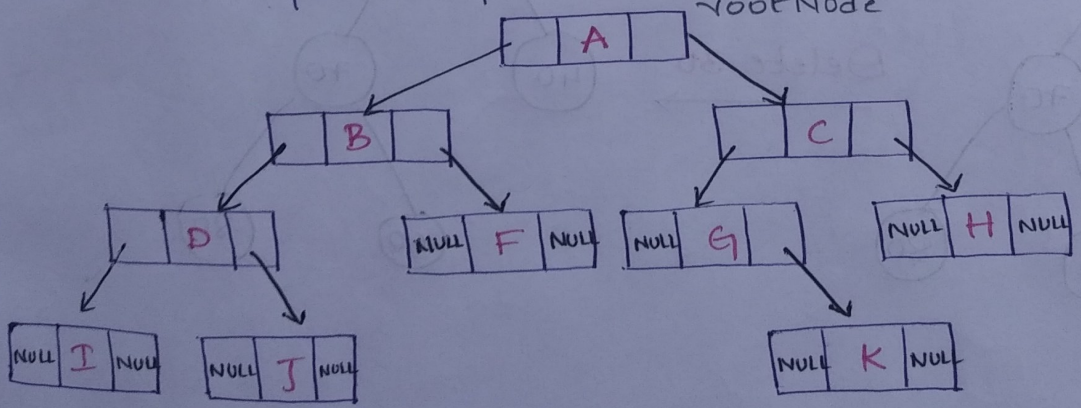
To represent a Binary tree of depth 'n' using array representation, we need 1-D array of max size 2^{n+1}

2) Linked list representation of a Binary tree.

we use a double linked list to represent a binary tree.

Left child address	Data	Right child address

above example is represented as



b) Explain the deletion operation on BST with an example

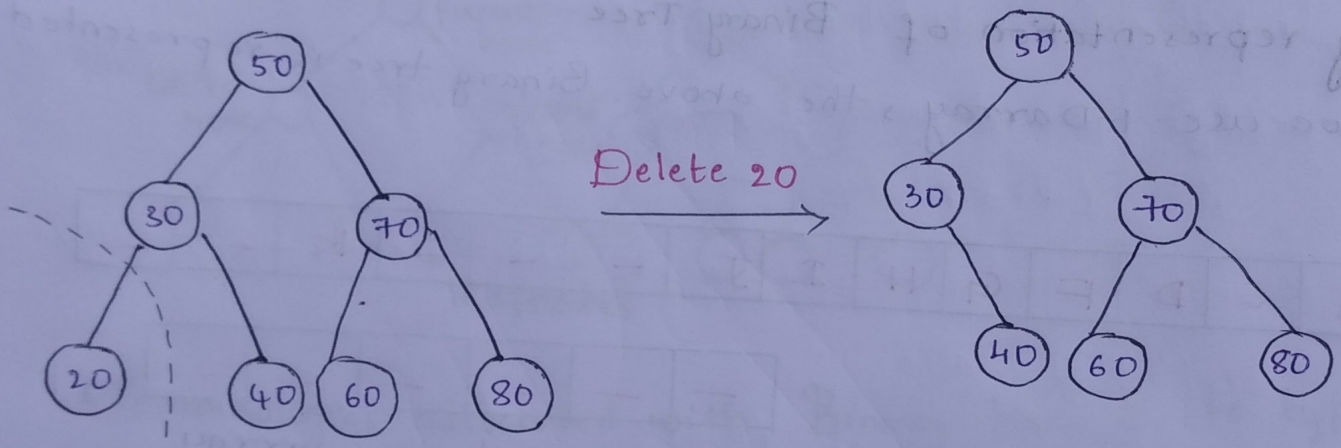
Deletion operation is performed to delete a particular element from the binary Search Tree.

when it comes to deleting a node from the binary Search tree, following three cases are possible.

case 1: Deleting a leaf node.

Just remove/disconnect the leaf node i.e to be deleted from the tree.

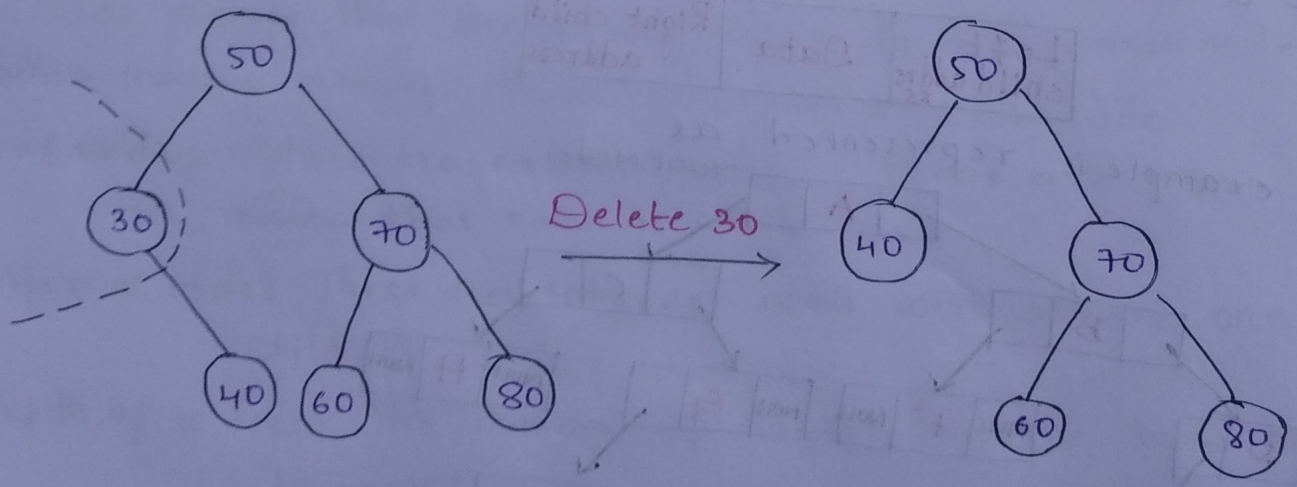
Ex:-



case 2: Deletion of a node having only one child.

Just make the child of the deleting node, the child of its grandparent.

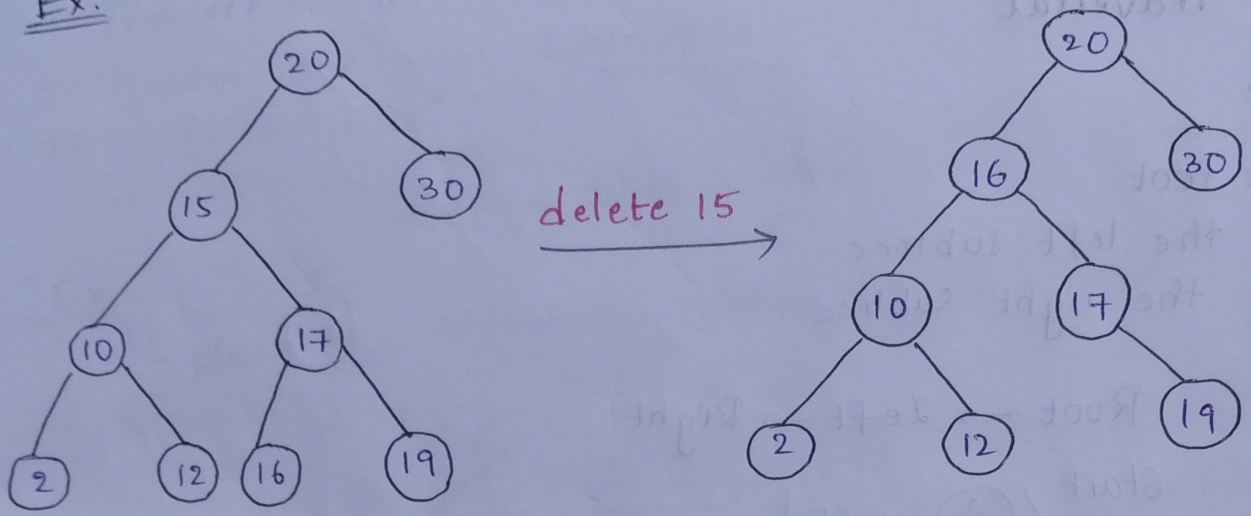
Ex:-



Case 3: Deleting a node with 2 children.

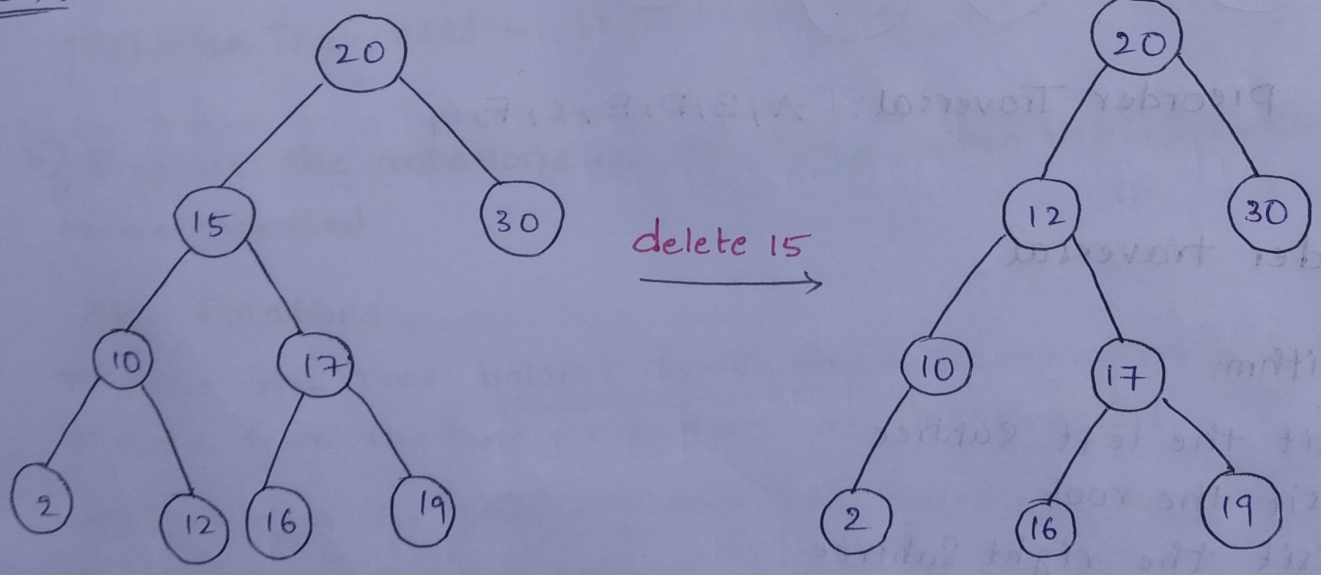
method 1: Go to right subtree of deleting node, pluck the least element called inorder Successor and replace with the deleting node.

Ex:-



method 2: Go to the left subtree of the deleting node, pluck the greatest element called inorder predecessor and replace with the deleting node.

Ex:-



2) a) Explain about binary tree traversing techniques.

1) Preorder Traversal
2) Inorder Traversal
3) Postorder Traversal

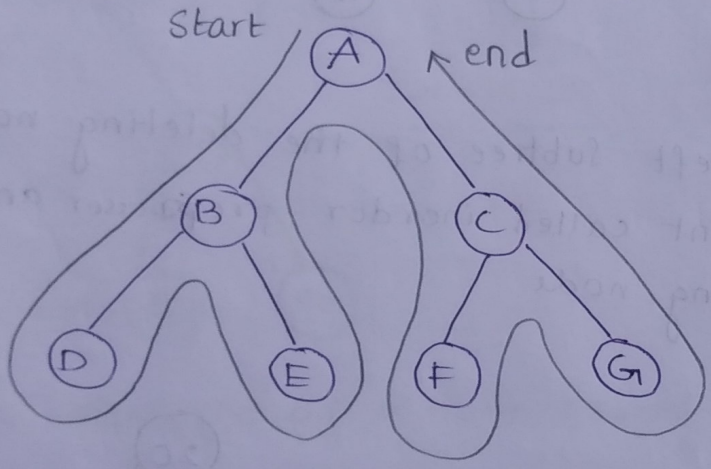
Preorder Traversal

Algorithm:

1) Visit the root
2) Traverse the left subtree
3) Traverse the right subtree

Root → Left → Right

Ex:-



Preorder Traversal: A, B, D, E, C, F, G

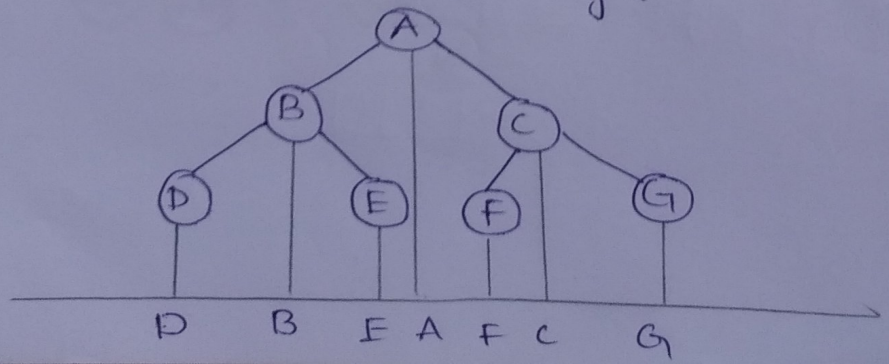
Inorder traversal

Algorithm:

1) visit the left subtree
2) visit the root
3) Visit the right subtree

Left → Root → Right

Ex:-



Inorder Traversal: D, B, E, A, F, C, G

post order traversal

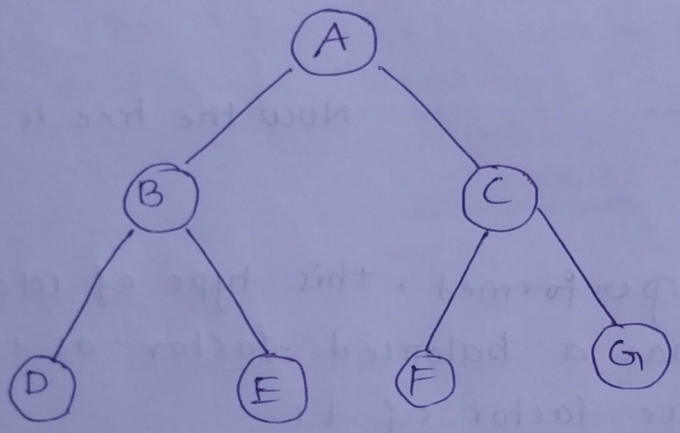
Algorithm

1. Traverse the left subtree
2. Traverse the right subtree
3. Visit the root

Left → Right → Root

Ex:-

pluck the leftmost leafnodes one by one



postorder Traversal:- D, E, B, F, G, C, A

b) Explain the rotations in AVL Tree when the element has to be inserted

AVL Rotations

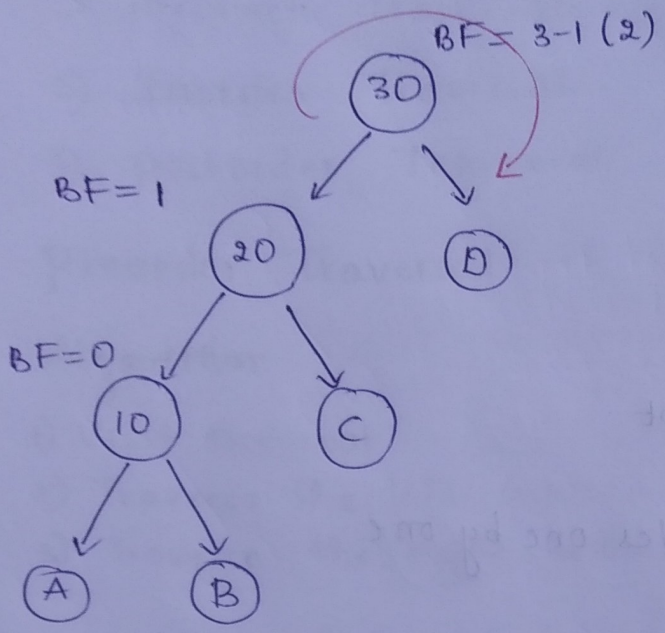
To make AVL Tree balance itself, when inserting or deleting a node from the tree, rotations are performed.

we perform the following rotations based on the Situations,

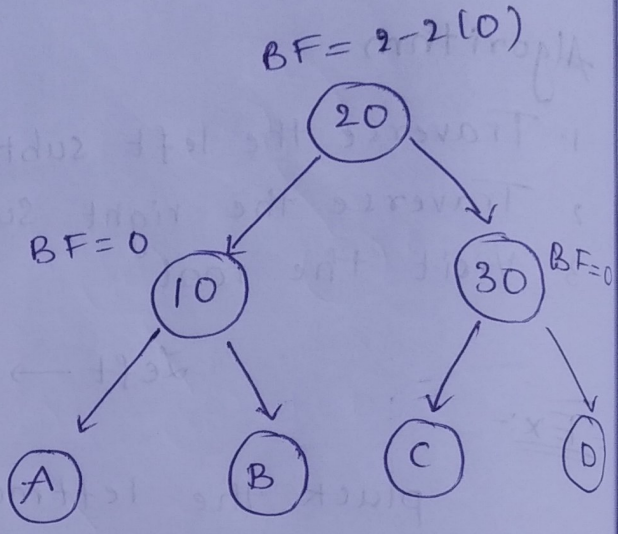
- Left - Left Rotation
- Right - Right Rotation
- Right - Left Rotation
- Left - Right Rotation

Left - Left Rotation.

This rotation is performed when a newnode is inserted at the leftchild of left subtree



Single right rotation



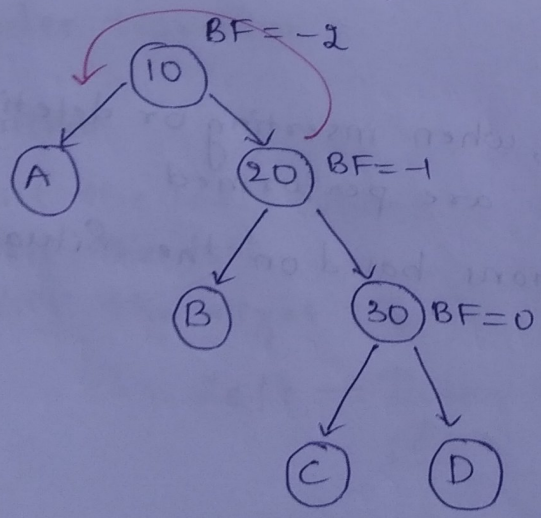
Tree is imbalanced as
BF(30) = +2

Now the tree is Balanced

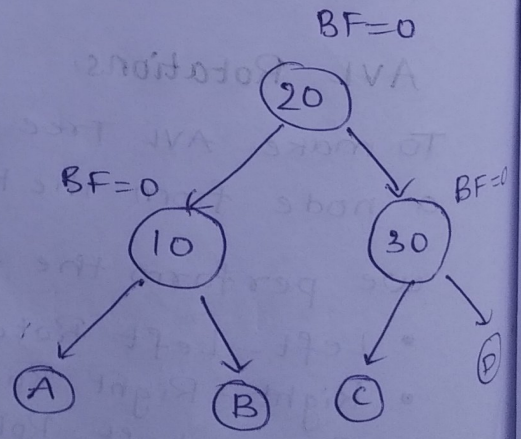
a Single right rotation is performed, this type of rotation is identified when a node has a balanced factor as +2, and its left child has balance factor of +1.

Right-Right rotations:

This rotation is performed when a newnode is inserted at the right child of the right subtree.



Single left rotation



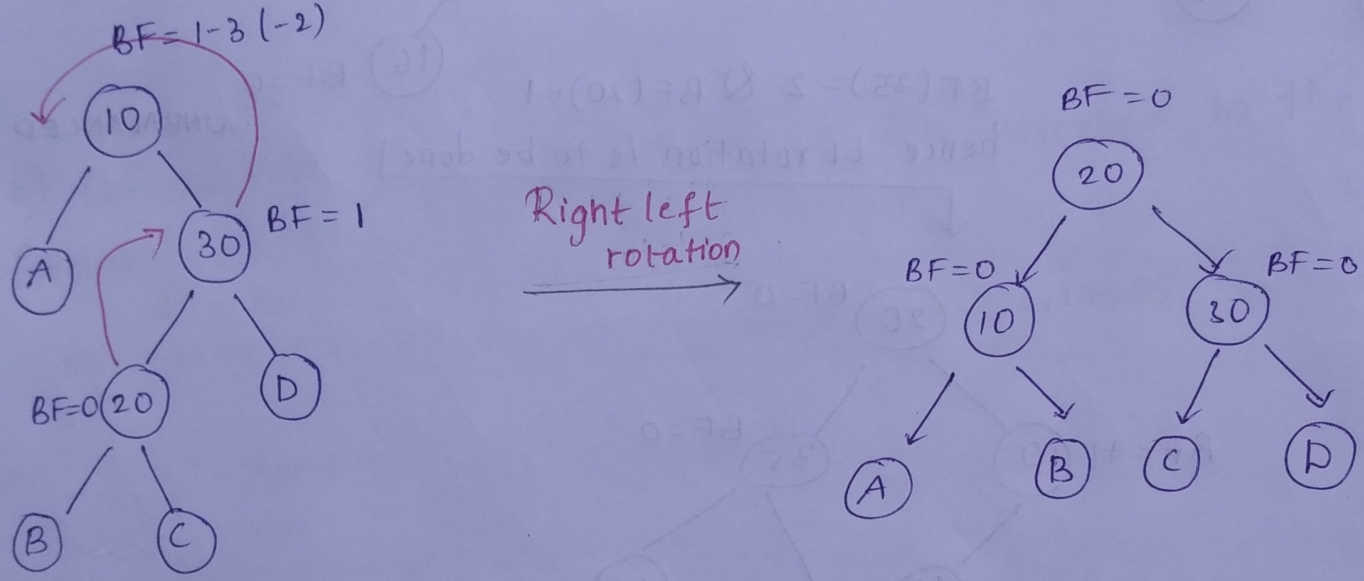
Tree is imbalance as BF(10)=-2

Now tree is balanced

A single left rotation is performed. This type of rotation is identified when a node has a balanced factor as -2, and its right child has a balanced factor of -1

Right- Left rotation

This rotation is performed when a newnode is inserted at the right child of the left subtree.



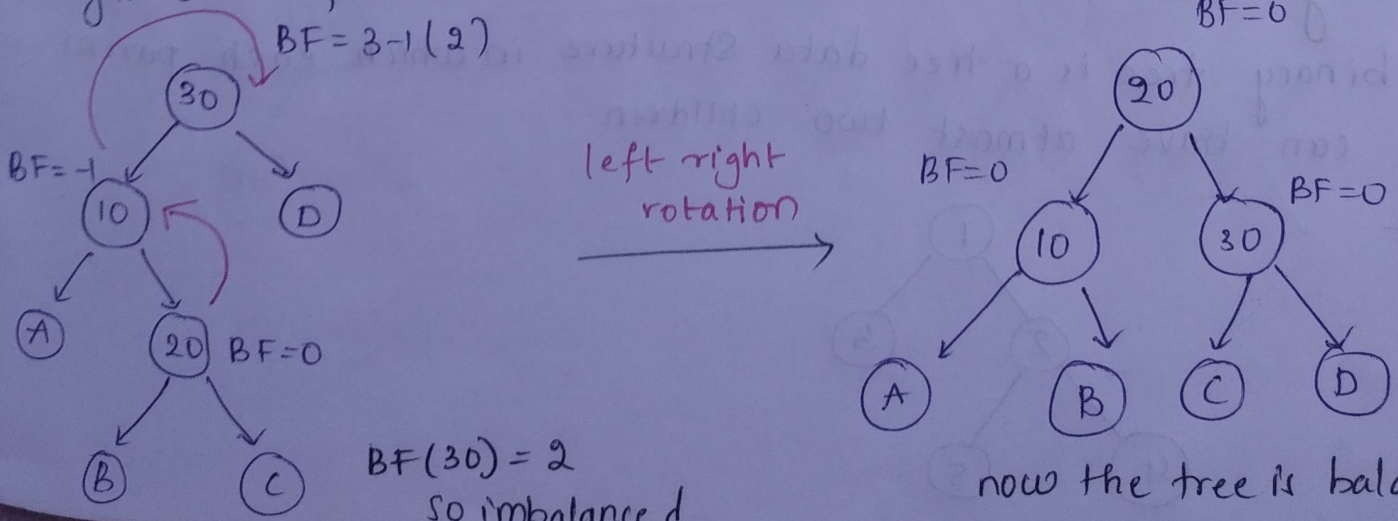
Tree is imbalance as BF(10) = -2

Now the tree is balanced

This rotation is performed when a node has a balance factor as -2 and its right-child has a balance factor as +1.

Left - Right rotation

This rotation is performed when a newnode is inserted at the right child of left subtree.



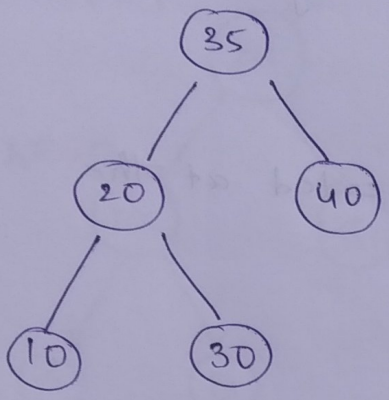
BF(30) = 2

So imbalanced

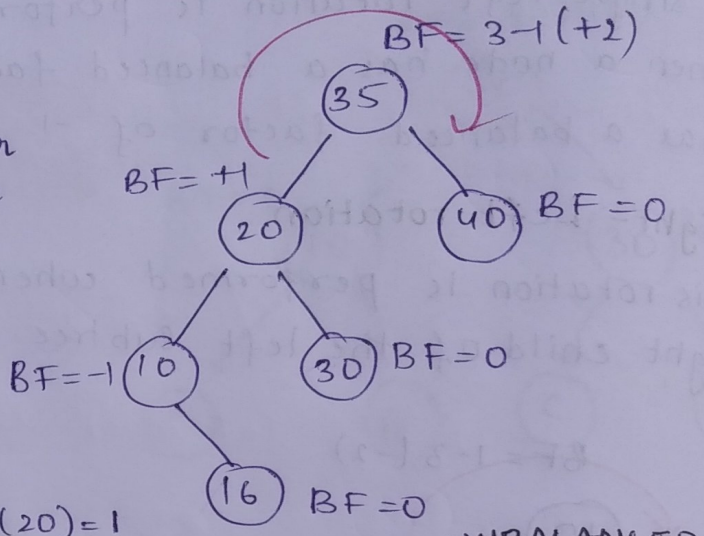
now the tree is balanced

This rotation is performed when a node has a balance factor as +2, and right-child has balance factor as -1.

Example:-



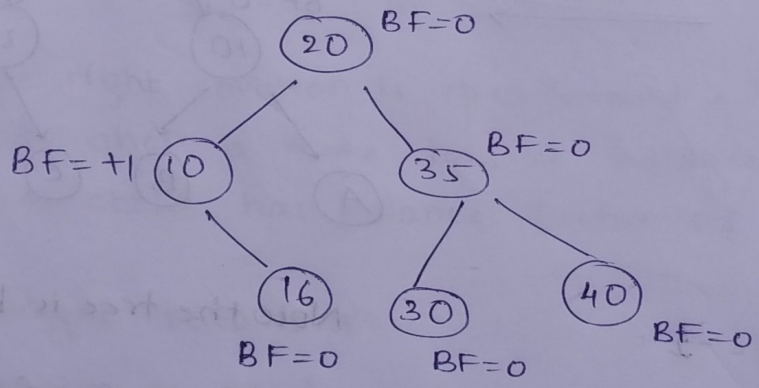
Insert 16 as per BST and update BF of each node



BF(35) = 2 & BF(20) = 1

UNBALANCED

hence LL rotation is to be done



NOW THE TREE GOT BALANCED

3)

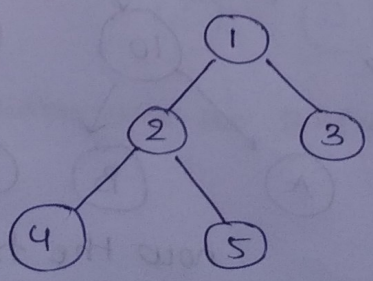
a) Explain the following with neat example

i) Binary Tree ii) Full Binary Tree.

Binary Tree:-

A binary tree is a tree data structure in which each parent node can have atmost two children.

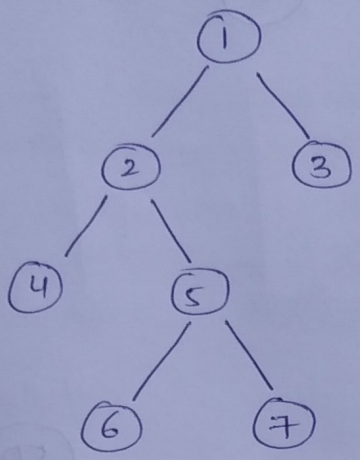
Ex:-



Full Binary Tree:-

A full binary tree is a special type of binary tree in which every parent node / internal node has either two or no children.

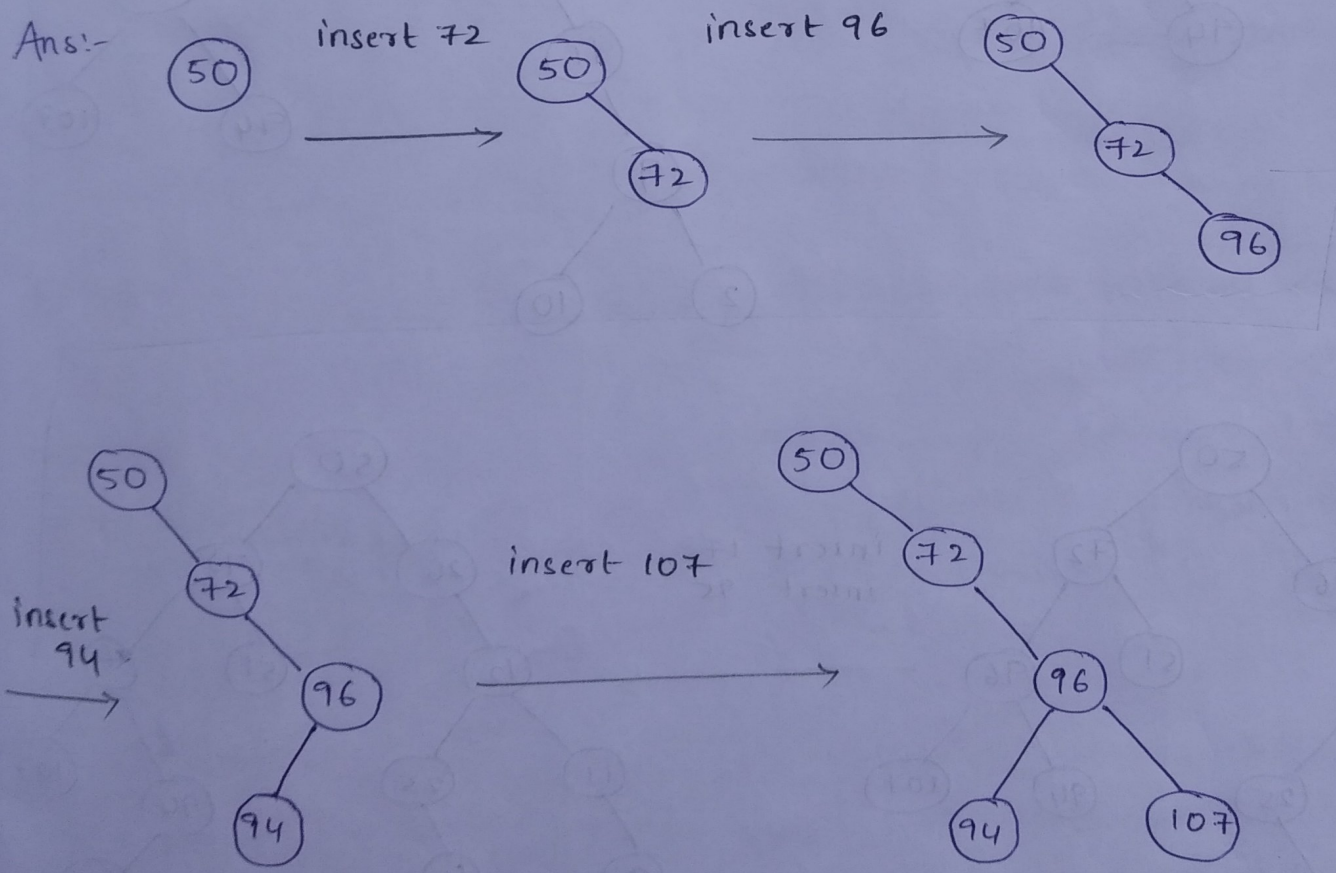
Ex:-

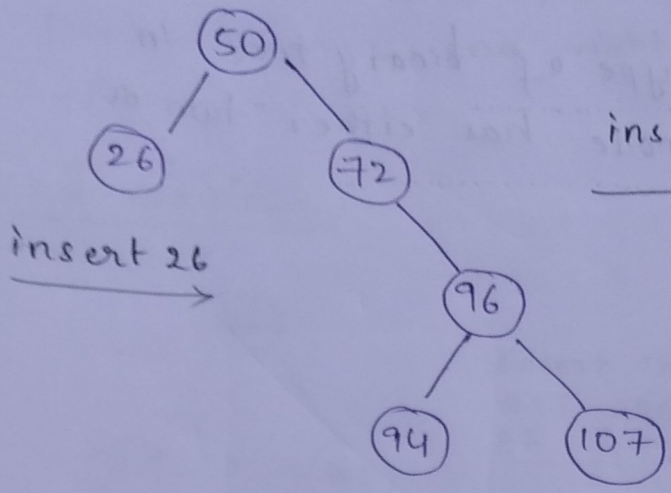


b) Construct a BST whose elements are inserted in the following order

50, 72, 96, 94, 107, 26, 12, 11, 9, 2, 20, 25, 51, 16, 17, 95.

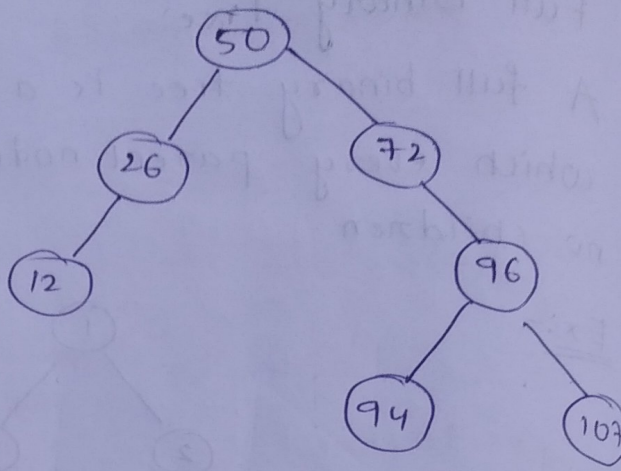
Ans:-



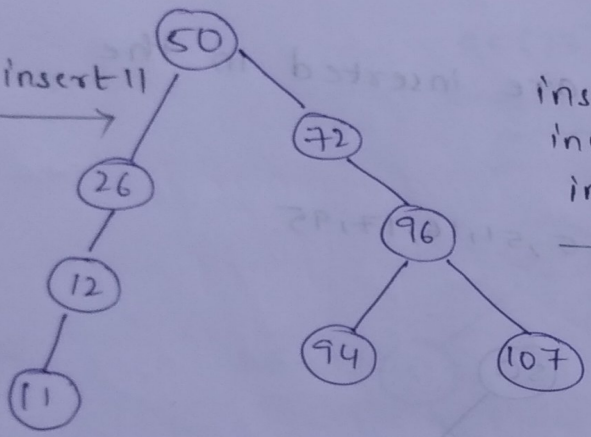


insert 26

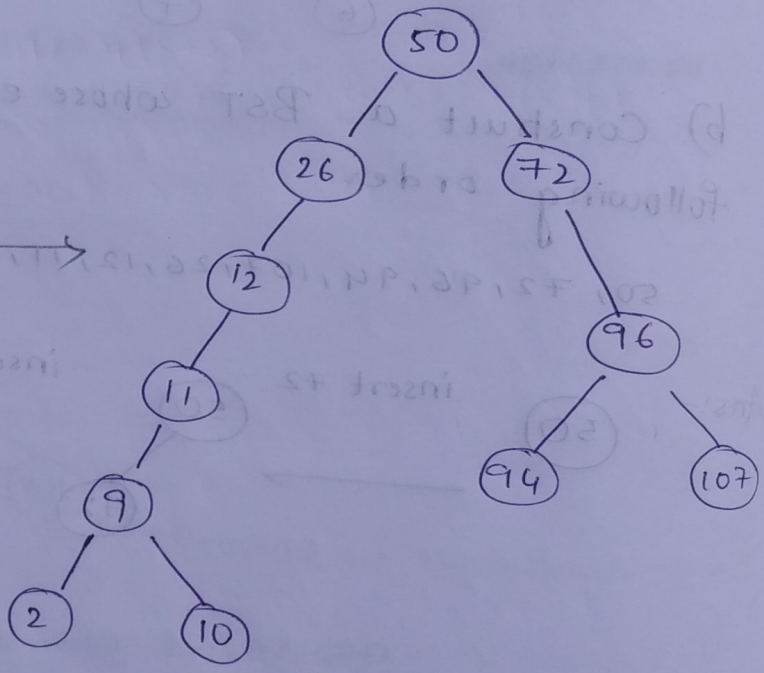
insert 12



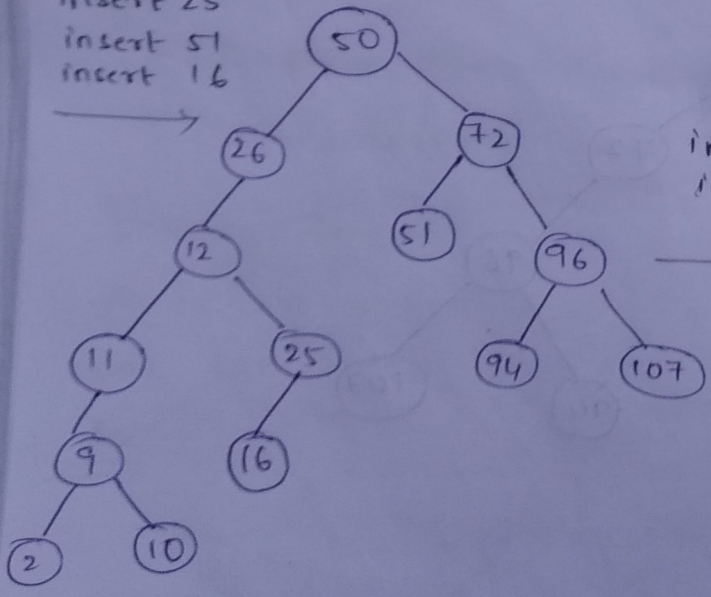
insert 11



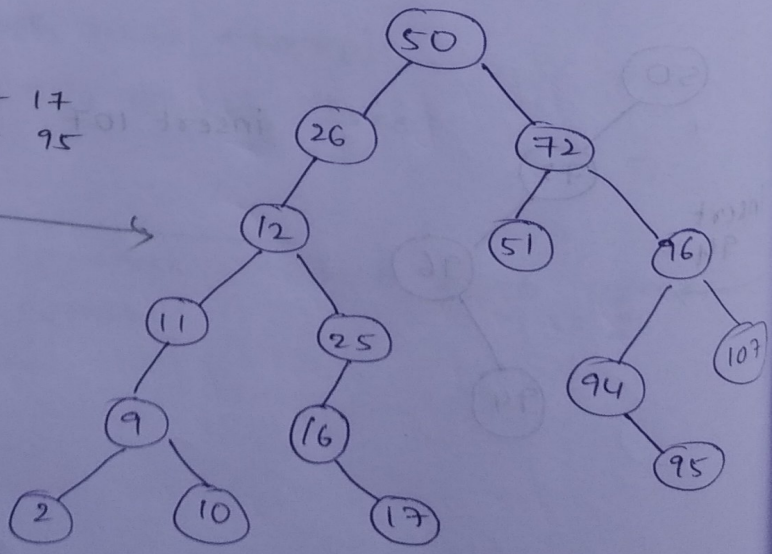
insert 9
insert 2
insert 10



insert 25
insert 51
insert 16



insert 17
insert 95



This is the final required Tree.

UNIT-5

1) a) Differentiate between DFS and BFS

Sr.NO	Key	BFS	DFS
1.	Definition	BFS, stands for Breadth First Search	DFS, stands for Depth First Search.
2.	Data Structure	BFS uses queue to find the shortest path	DFS uses stack to find the shortest path
3.	Source	BFS is better when target is closer to source	DFS is better when target is far from source.
4.	Suitability for descision tree	As BFS considers all neighbour so it is not suitable for descision tree using in puzzles	DFS is more suitable for descision tree. As with one descision, we need to traverse further to augment the deccion. if we reach the conclusion we won
5	Speed	BFS is slower than DFS	DFS is faster than BFS
6	Time Complexity	O(V+E) where V is Vertices, E is edges	Same as BFS

b) What is minimum cost Spanning tree? Discuss with an example

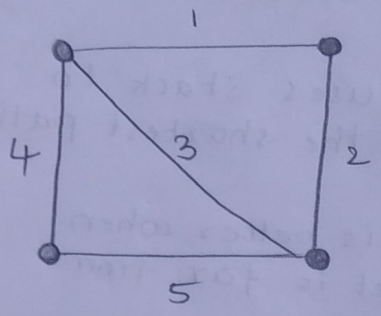
The cost of the Spanning tree is the sum of the weights of all the edges in the tree. There can be many Spanning trees. Minimum Spanning tree is a Spanning tree where the cost is minimum among all the trees. There also can be many minimum Spanning trees.

Minimum Spanning tree has direct application in the design of networks. It is used in the algorithms approximating the travelling Salesman problem, multi-terminal minimum cut problem and minimum-cost weighted perfect matching.

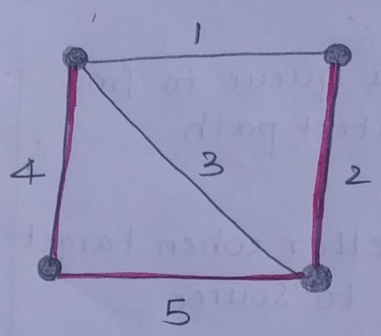
Other practical applications are

1) cluster analysis
2) Handwriting recognition
3) Image Segmentation

Ex:-

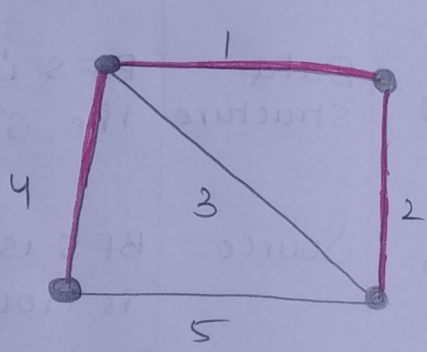


undirected graph



Spanning tree

cost = 11 (= 4+5+2)



Minimum Spanning Tree

cost = 7 (= 4+1+2)

c) Explain Prim's algorithm and trace with an Example.

Ans:- Prims algorithm is used for finding the minimum Spanning Tree (MST) of a given graph.

→ To Apply prims Algorithm, the given graph must be weighted, connected and undirected.

The implementation of prims Algorithm is explained in the following steps:-

Step 1:

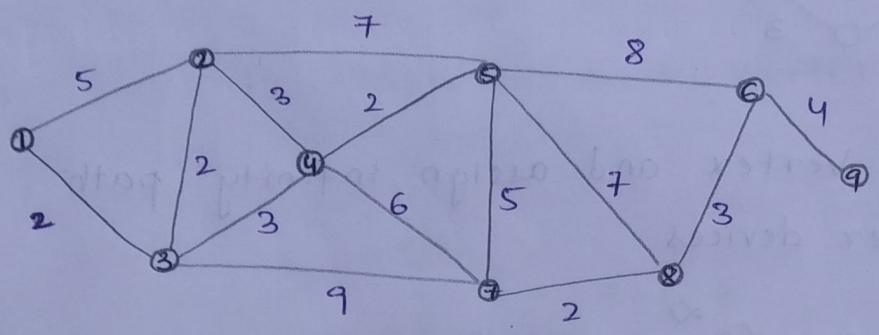
1) Randomly choose any Vertex
2) The vertex Connecting to the edge having least weight is usually Selected.

Step 2:

1) find all the edges that connect the tree to new Vertices
2) find the least weight edge among those edges & include it in the existing tree.
3) if including that edge creates a cycle, then reject that edge and look for the next least weight edge.

Step 3: Keep repeating Step-2 until all the Vertices are included and minimum spanning tree is obtained

Example:



→ Let us strat from vertex 1
→ Vertex 3 is connected to Vertex 1 with minimum edge cost hence edge (1,2) is added to the spinning tree.
→ Next, edge (2,3) is considered as this is the minimum among other edges which are remaining
→ In next step, we selected edge (3,4)
→ In similar way, edges (4,5), (5,7), (7,8), (8,6), (6,9) are Selected
→ As all vertices are visited, now the algorithm stops.
i.e, The cost of Spanning tree is

$(2+2+3+2+5+2+3+4) = 23$

There is no more Spanning tree in this graph without less than 23

2)
a) Explain in detail about Dijkstra's algorithm for finding Shortest path with an example.

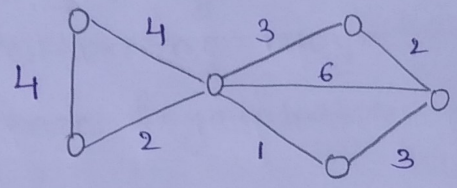
Ans:- i) Dijkstras algorithm allows us to find the shortest path between any Vertices of a graph.

2) It differs from the minimum Spanning tree because the Shortest distance b/w two vertices might not include all the Vertices of the graph

Example of Dijkstra's Algorithm

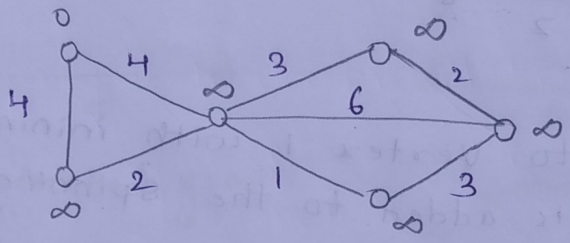
Step 1:

Start with a weighted graph:



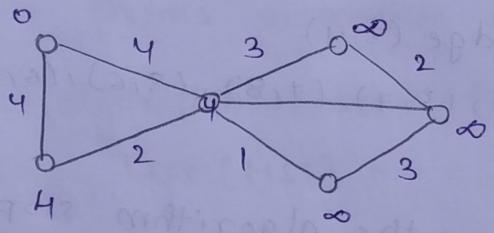
Step 2:

Choose a starting Vertex and assign infinity path values to all other devices.

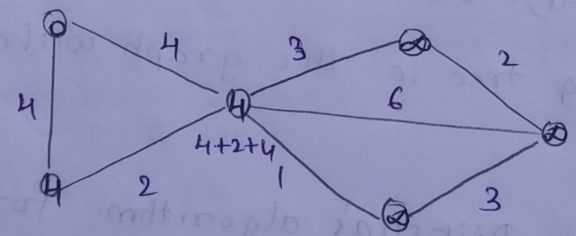


Step 3:

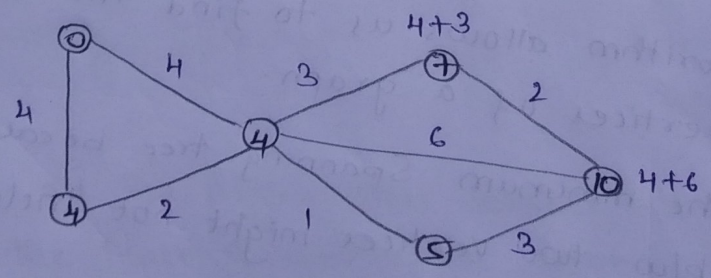
Go to each Vertex and update its path length



Step 4: If path length of the adjacent Vertex is lesser than new path length, dont update it

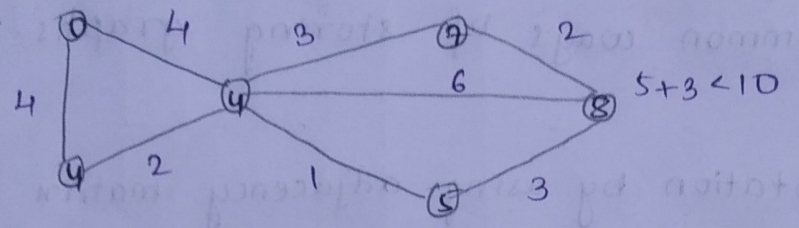


Step 5: avoid updating pathlengths of already visited Vertices

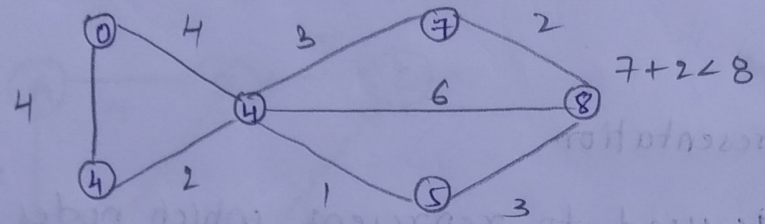


Step 6:

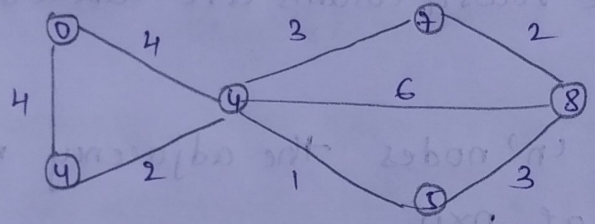
After each iteration, we pick the unvisited Vertex with the least path length. So we choose 5 before 7



Step 7: Notice how the right most Vertex has its path length update twice.



Step 8: Repeat until all the vertices have been visited.



The time complexity is $O(E \log V)$

$E \rightarrow$ no. of edges

$V \rightarrow$ no. of vertices

Space complexity is $O(V)$

b) Define graph and explain the representation of graphs with neat examples.

it is basically a collection of Vertices (also called nodes) and edges that connect these vertices

graph is viewed as a generalization of the tree

Definition:- A graph G is defined as an ordered set (V, E), where V(G) represents the set of vertices and E(G) represents the edges that connect those vertices.

* A graph can be directed / undirected graph.

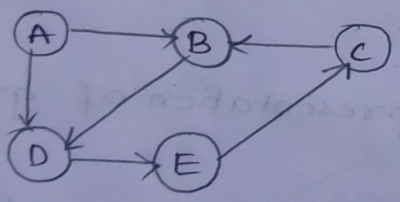
Representation of graphs:

There are 3 common ways of storing graphs in the Computers memory

1. Sequential representation by using adjacency matrix.
2. Linked representation by using adjacency list that stores the neighbors of a node using a linked list.
3. Adjacency multilist which is an extension of linked representation

1) Adjacency matrix representation.

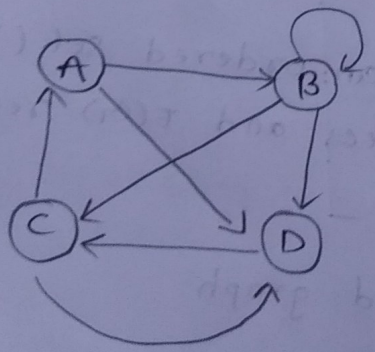
* An adjacency matrix is used to represent which nodes are adjacent to one another.
* In an adjacency matrix the rows & colums are labelled by graph vertices.
* for any graph 'G' having 'n' nodes the adjacency matrix will have the defnition of n×n.
* adjacency matrix contains only 0's and 1's. also known as Bit matrix (or) Boolean matrix.



(Directed graph)

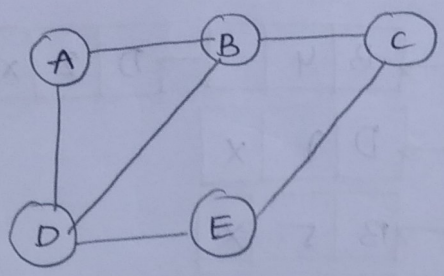
Matrix

	A	B	C	D	E
A	0	1	0	1	0
B	0	0	0	1	0
C	0	1	0	0	0
D	0	0	0	0	1
E	0	0	1	0	0



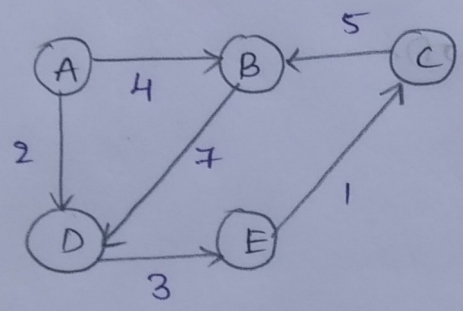
	A	B	C	D
A	0	1	0	1
B	0	1	1	1
C	0	0	0	1
D	0	0	1	0

Directed graph with loop



	A	B	C	D	E
A	0	1	0	1	0
B	1	0	1	1	0
C	0	1	0	0	1
D	1	1	0	0	1
E	0	0	1	1	0

undirected graph

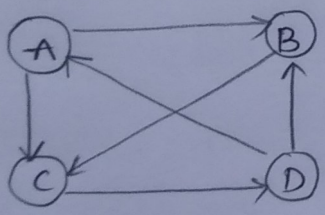


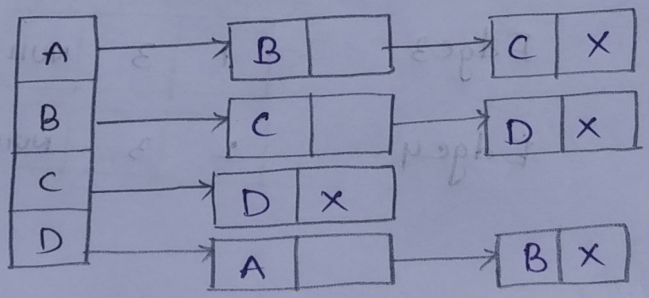
	A	B	C	D	E
A	0	4	0	2	0
B	0	0	0	7	0
C	0	5	0	0	0
D	0	0	0	0	3
E	0	0	1	0	0

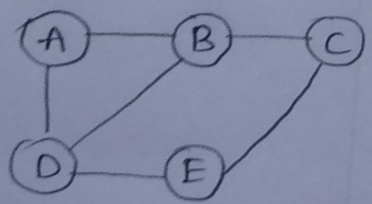
weighted graph

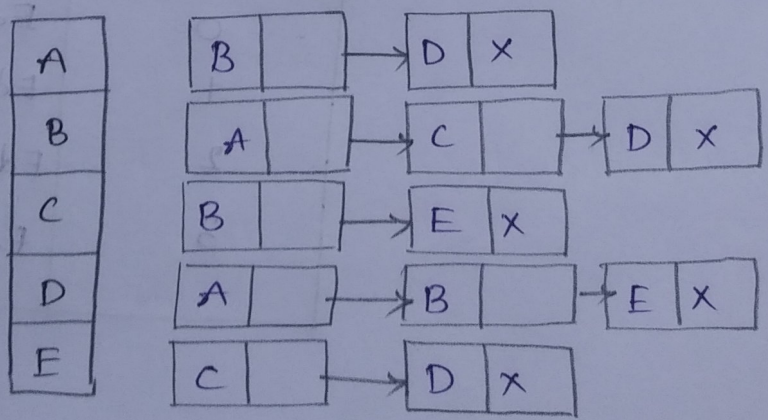
2) Adjacency list representation

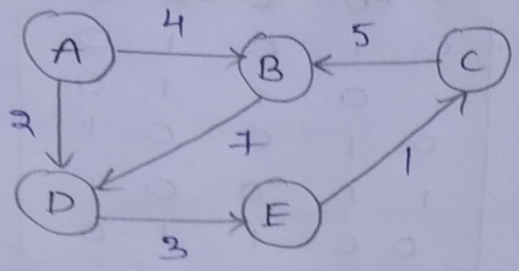
An adjacency list representation is another way in which graphs can be represented in the computers memory This structure consists of a list of all nodes in graph G.

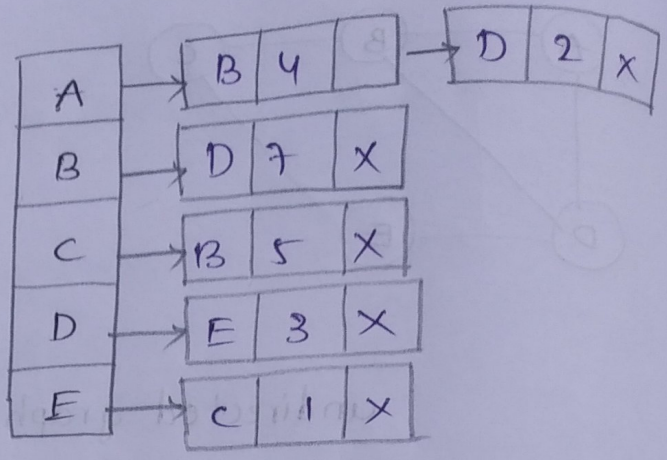






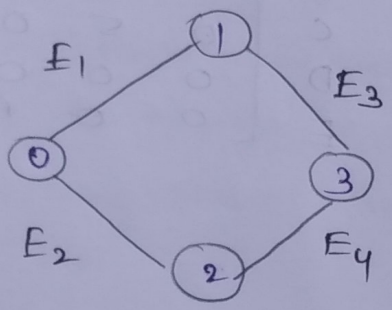






Adjacency multi list representation.

Ex:-



Edge1

0	1	E_2	E_3

Edge2

0	2	E_1	E_4

Edge3

1	3	NULL	NULL

Edge4

2	3	NULL	NULL

Vertex	list of Edges
0	Edge1, Edge2
1	Edge1, Edge3
2	Edge2, Edge4
3	Edge3, Edge4