DSA UNIT - 3

Topics =
- Graphs
  - Prim's Algorithm
  - Kruskal's Algorithm
  - Dijkstra Algorithm
- Trees
  - Binary Search Tree
  - Left Binary Tree
  - Almost Complete Binary Search Tree (ACBT)
  - Complete Binary Search Tree
  - (Inorder, Preorder, Postorder)
- BFS (Breadth first search)
- DFS (Depth first search)
- AVL Trees
- Heap (Max Heap And Min Heap)
- B - Tree
- Hashing
  - Chaining (Open hashing)
  - Open Addressing
    - Linear Probing
    - Quadratic Probing
    - Double Probing

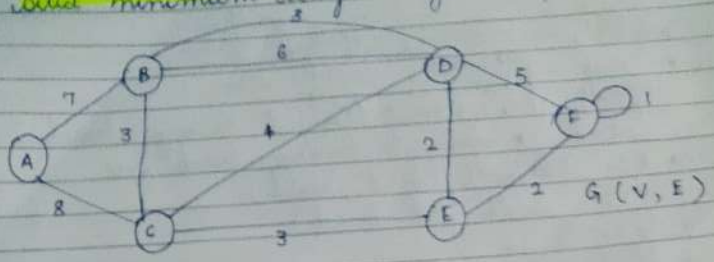
* Graphs = In Graphs each ~~edge~~ (Vertices / Nodes) are connected to each other by the help of adjacent ~~vertices~~ Edges
- Non-linear data structure
- To find Minimum Spanning Tree, there are three methods =
  1. Prim's Algorithm
  2. Kruskal's Algorithm
  3. Dijkstra Algorithm

(i) PRIM'S ALGORITHM = Time complexity = $O(V^2)$ where, V is the vertex (greedy Algo / Min Spanning Tree)
Steps = because vertex is inserted in priority queue only once
1. Remove the looping node.

Insertion in the vertex priority queue takes logarithmic time.

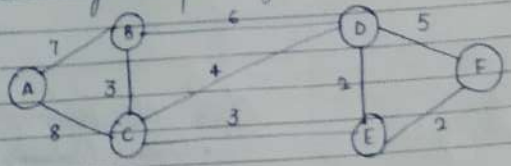
→ We choose any arbitrary node.
→ Used in dense graphs.
→ Condition = No, closed loop should be made.
2. Select any arbitrary node as root node.
3. Select minimum weight edge comparable.

Q.

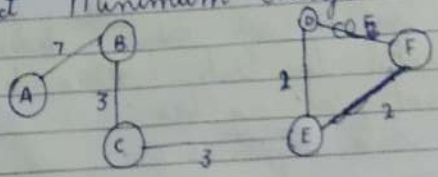


G(V, E)

→ Removing loop edge



Let's assume A as arbitrary edge.
Select Minimum weight.



G(V', E')

$V = V'$

$E' = |V| - 1$

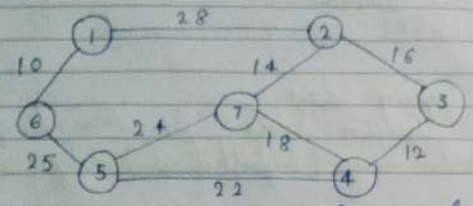
$(7 + 3 + 3 + 2 + 2) = 17$

$\because V = 6, \quad E' = (6 - 1) = 5$

Note = $E' = |V| - 1$

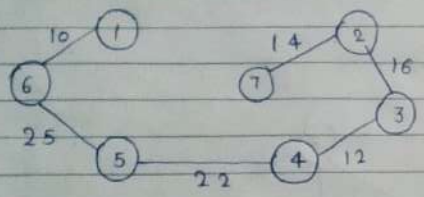
$V = V'$

Q.



G (V, E)

⇒ Let's say, 6 as the Arbitrary node



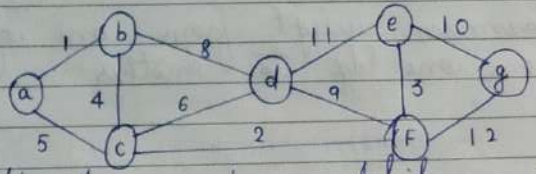
G' (V', E')

Minimum Spanning Tree = $V = V'$

$(10 + 25 + 22 + 12 + 16 + 14)$ $E' = |V| - 1$ ⇒ $(7-1) = E'$

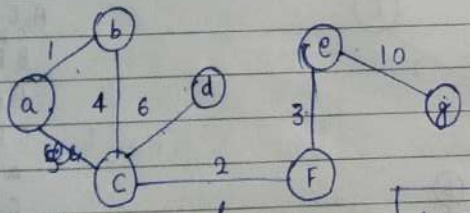
⇒ 99 ⇒ $E' = 6$

Q.



G (V, E)

Let's Assume b as Arbitrary Node



G' (V', E')

⇒ Minimum Spanning Tree = $(1+4+6+2+3+10)$ $V = V'$ $E' = |V| - 1$ ⇒ $(7-1) = E'$ ⇒ $E' = 6$

26 Units

a.

→ Heavy (unit) weighted node (vertices).
→ Used in sparse Graphs.

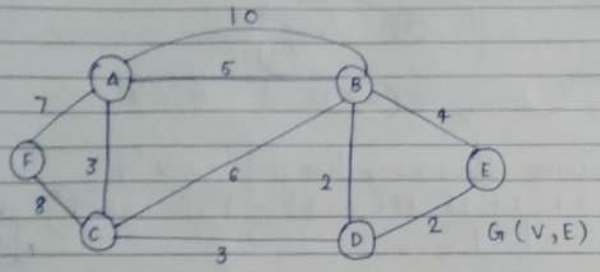
(ii) KRUSKAL'S ALGORITHM = $O(E \log V)$

Steps =

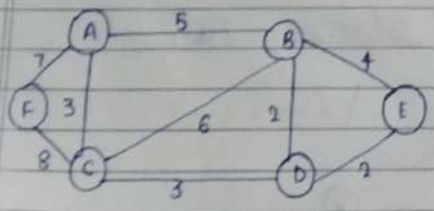
→ Remove the loopings edges.
→ Take slight sufficient edges. (In Ascending Order)
→ Take one arbitrary (hypothetical) edg and compare.

Condition = The edges should never make closed loop.

Q =

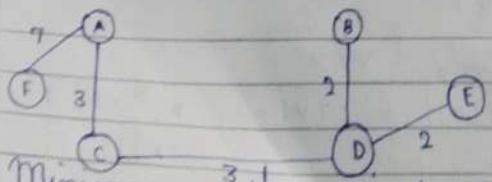


Remove Maximum weight from one of two weight from one edg to another.



~~AB = 5~~
~~AF = 7~~

BD = 2
DE = 2
AC = 3
~~CD = 3~~
~~BE = 4~~
AB = 5
CB = 6
AF = 7
CE = 8



G(V', E')

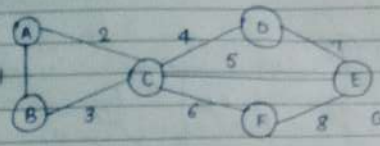
V' = V

E' = |V| - 1 = (6 - 1) = 5

Minimum spanning Tree = (7 + 3 + 3 + 2 + 2) = 17

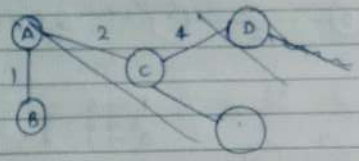
B+

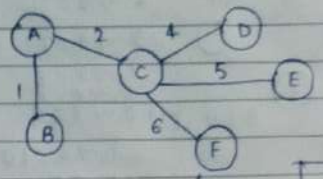
Q.



G(V, E)

AB = 1
AC = 2
BC = 3
CD = 4
CE = 5
CF = 6
DE = 7
FE = 8





G(V', E')

⇒ Minimum spanning Tree

$V' = V$

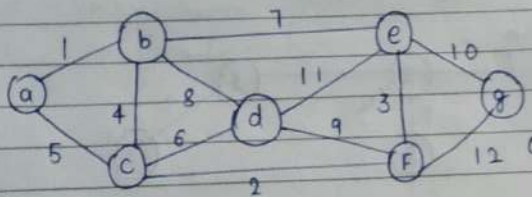
$E' = |V| - 1 = (6 - 1) = E'$

$E' = 5$

(1 + 2 + 4 + 5 + 6)

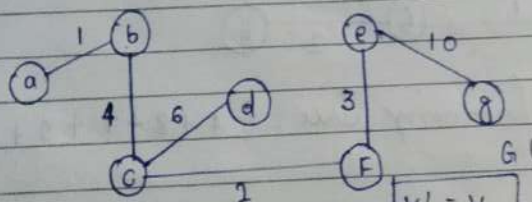
18

Q.



G(V, E)

ab = 1
cf = 2
ef = 3
~~bc = 4~~
ac = 5
cd = 6
be = 7
bd = 8
df = 9
eg = 10
de = 11
fg = 12



G(V', E')

$V' = V$

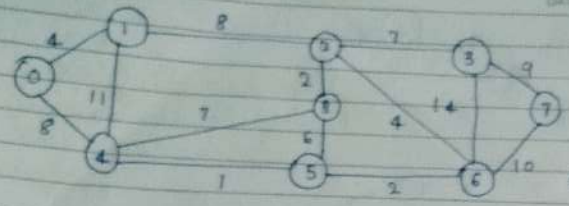
$E' = |V| - 1 = (7 - 1) = 6 = E$

⇒ Minimum spanning Tree

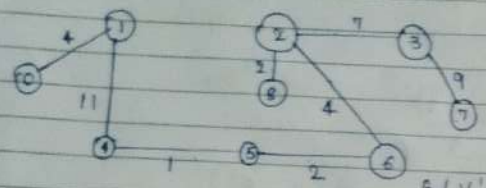
⇒ (1 + 4 + 6 + 2 + 3 + 10)

⇒ 26

Q =



G(V, E)



G(V', E')

$4 \to 5 = 1$
$5 \to 6 = 2$, $2 \to 8 = 2$
$0 \to 1 = 4$
$2 \to 6 = 4$
$5 \to 8 = 6$
$4 \to 8 = 7$
$2 \to 3 = 7$
$1 \to 2 = 8$
$8 \to 4 = 8$
$3 \to 7 = 9$
$7 \to 6 = 10$
$1 \to 4 = 11$
$3 \to 6 = 14$

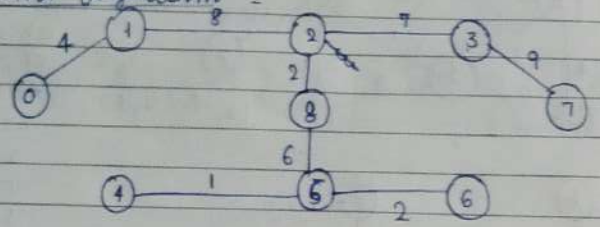
$V' = V$

$E' = |V| - 1$

$E' = 9 - 1 = 8$

⇒ Minimum Spanning tree = $(4 + 11 + 1 + 2 + 2 + 4 + 7 + 9)$
$= 40$

Prim's Algorithm =



⇒ Minimum Spanning Tree = $(4 + 8 + 7 + 2 + 6 + 2 + 14 + 9)$

⇒ 39

Applications = Used to resolve many real-life problems as =

i) Networking (telephone networks) in which edges channels will act as Edge and the nodes will be considered as Nodes (Vertices)
ii) Social media networks (Communication) like linkedin, Facebook, Google etc.

Breadth First Search :=
→ Works on concept of Queue.

Queue = linear data structure with the concept of first In first Out

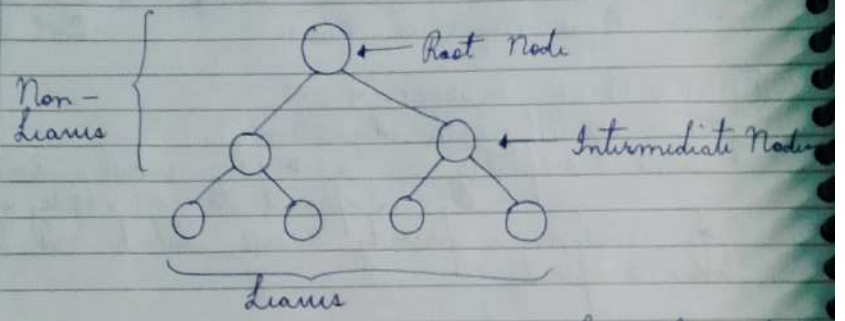
Applications :=
→ In movie Queue ticket taking (First In First Out)
→ In Mess for food Queue

(iii) DIJKSTRA ALGORITHM :=
→ The shortest path Algorithm.
→ Time complexity = $O(V^2)$ which can be improved to $O(V+E \log V)$ because of minimum priority Queue which has logarithmic time complexity.

DATE / /

* Trees = Trees are non-linear data structures which are not having connected circuit edges.
Or
Non-linear data structures which does not have connected circuit.

→ A node of tree has three parts =
(i) Data
(ii) Points to left Node. (Left Childs)
(iii) Pointer to Right Node (Right Childs)

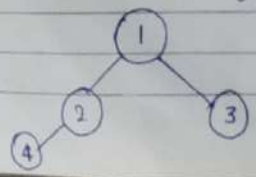


Let's Say there are n number of nodes then, the number of vertices will be

→ Number of Vertices = (n - 1)

(i) Binary Tree =
→ It must be having 0, 1 or 2 child nodes.

Eg =



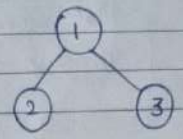
1 → 2 Child Nodes
2 → 1 Child Node
3 → 0 Child Node
Hence, Binary Tree.

DATE ___/___/___

(ii) Full Binary Tree =

→ Must be having 0 or 2 child nodes.

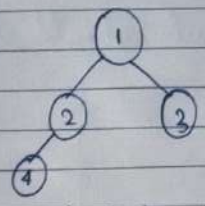
Eg = (a)



1 → 2 Child Nodes
2 → 0 Child Nodes
3 → 0 Child Nodes

Hence, full binary tree because having 0 or 2 child nodes.

(b)



1 → 2 Child Nodes
2 → 1 Child Node
3 → 0 Child Node
4 → 0 Child Node

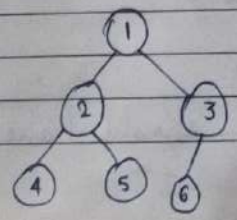
Hence, not full binary tree because 2 is having only one child node.

Note = A full binary tree is always a binary tree but binary tree is not always i.e; (that is) may or may not be full binary tree.

(iii) Almost Complete Binary Tree =

→ Their is scanning from left to right.
If the same level node is having 0 child, 1, 2 child then almost complete Binary tree.
Else it would be not almost Complete Binary Tree.

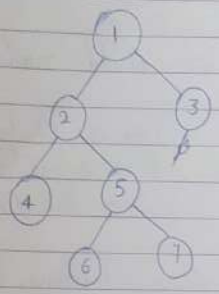
Eg = (a)



Almost Complete Binary Tree.

DATE / /

b)

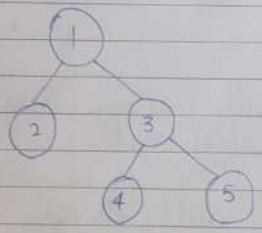


Not an Almost Complete Binary Tree.

iv) Complete Binary Tree =

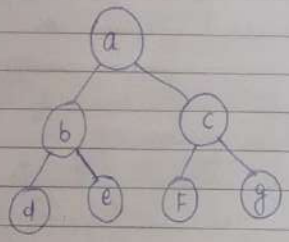
→ The nodes at same level should be having Equal 02 child nodes.

Eg = a)



Not a complete Binary Tree

b)



Complete Binary Tree.

Binary Search Tree =

Properties =

i) The left subtree should have smaller elements

DATE / /

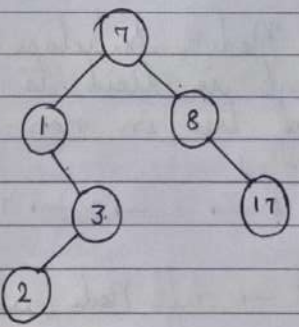
with respect to the root node.
ii) The right subtree should have greater elements as compared to the root node.

⇒ Note = Binary Tree = 0, 1, 2 childs
Full Binary Tree = 0, 2 childs
Almost Complete Binary Tree = Left to right scan, having 0 or 2 childs at same level.
Complete Binary Tree = Having same number of child at same level.

* BINARY SEARCH TREE =
→ The binary search tree is binary tree which is having left subtree having smaller elements than root node.
→ Right subtree is having greatest subtree.
→ Number of edges = (Vertices - 1)

Q

7	8	1	3	2	17



Binary Search tree

Edges = (Vertices - 1)

DATE / /

(i) According and with respect to breadth first search would be $O(h)$ where h is the height of the tree.

(ii) According to Breadth First Search the time complexity for Insertion in Binary Search Tree would be $O(n)$.

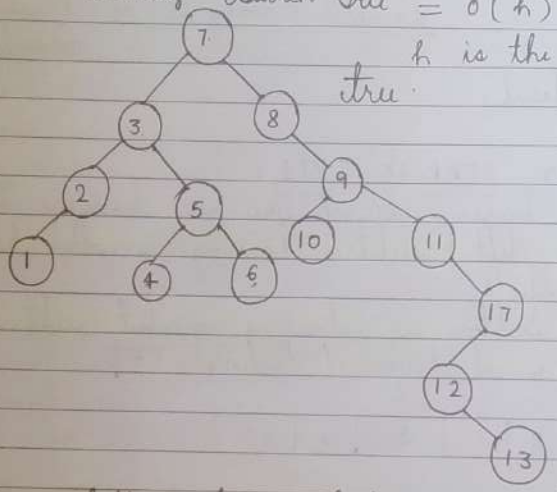
(iii) For deletion $O(n)$, time complexity is $O(h)$.

Q.

7	3	2	8	9	5	1	6	4	11	17	10	12	13

Binary Search Tree = $O(h)$

h is the height of the tree.



(i) Inorder = Left Node (Subclass) → Root → Right Node (Subclass)

(Inorder Traversal is used to get Element of tree in non-decreasing order)

1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 10 → 9 → 11 → 12 → 13 → 17

(ii) Preorder = Root → Left Node (Subclass) → Right Node (Subclass)

→ Used to arrange the elements and is also used to find prefix expression of the given tree

3+

DATE / /

Preorder = Root → Left Node (Subclass) → Right Node (Subclass)

7 → 3 → 2 → 1 → 5 → 4 → 6 → 8 → 9 → 10 → 11 → 17 → 12 → 13

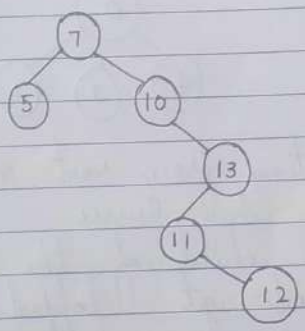
(iii) Postorder = Left Node (Subclass) → Right Node (Subclass) → Root

→ The Postorder traversal is used to delete the tree.

[Postorder = Left → Right → Root]

1 → 2 → 3 → 4 → 5 → 6 → 7 → 10 → 17 → 13 → 12 → 11 → 9 → 8

Q.



→ Inorder = Non-decreasing Order

Left Node → Root → Right Node

(Subclass) (Superclass)

5 → 7 → 10 → 13 → 11 → 12

Preorder = Used to find prefix Expression.

Root → Left Node → Right Node

(Subclass) (Subclass)

7 → 5 → 10 → 13 → 11 → 12

Postorder = Used to delete the tree.

Left Node → Right Node → Root

(Subclass) (Subclass)

→ 5 → 12 → 11 → 13 → 10 → 7

DATE __/__/__

* Breadth First Search =

→ Works on Queue.

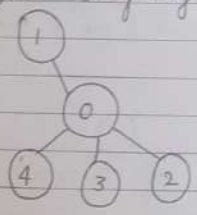
Queue is a linear data structure which works on concept of First In and First Out.

→ There are two ends front and Rear.
Insertion takes place from Rear end and deletion from Front.

→ Eg = A line of movie tickets.

→ Used to represent Adjancy list

Q.



Steps = (i) Starting from root node we have to push in Queue.

(ii) As the child of the root node comes the root node get popped.

(iii) If there are more than one child then concept of Queue is used.

S = X → Ø → X → 8 → X

BFS = 1 → 0 → 4 → 3 → 2

→ Time Complexity =

•) O(E + V) for Adjancy List where E is Edges and V is vertices

•) O(V²) for Adjancy Matrix, where V is i.e; nodes. vertices.

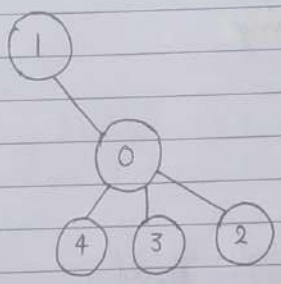
Since, we are dealing with Adjancy matrix, hence the time complexity becomes O(V²).

DATE ___/___/___

* Depth First Search =
→ Works on Stack.
On the concept of Last In First Out
→ It is a vertex based Technique.

Steps = (i) The root element is pushed in the stack
(ii) As the child element arrives, the root element should be popped from the stack and dropped
(iii) If there are more than two child elements then DFS is used.

Q.



S → 1 → 0 → 4 → 3 → 2
Depth First Search → 1 → 0 → 2 → 3 → 4

Note = The time complexities are as follo
(i) O(E+V) = Adjancy List
(ii) O(V2) = Adjancy Matrix

Since, we are dealing in terms of adjancy Li
Matrix, hence, time complexity would be
$O(v^2)$ where v is the vertex.