ADA

INDEX

NAME: SUBJECT:

Std.: Div.: Roll No.:

School/collage:

S.No.	Date	Title	Page No.	Remarks
		Linear search : O(n)		
		Binary search : O(log n)		
		Bubble sort : $O(n^2)$		
		Merge sort : O(n log n)		

Algorithm [illegible] અને language માં [illegible] Add/Delete item [illegible] અને Need to write program

	Avg. case	Worst case	Best case
Binary search	O(log n)	O(log n)	~~O(log n)~~ O(1)
Quick sort	O(n log n)	$O(n^2)$	O(n log n)
Bubble / selection / insertion	$O(n^2)$	$O(n^2)$	$O(n^2)$.

Best case = Ω
Average case = Θ
Worst case = O

$O(1)$ means some constant, N might be 5, or 1 or 1000.

Strassen's Matrix Multiplication. $[8T(n/2) + n^2 \Rightarrow 7T(n/2) + n^2]$.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} P_4+P_5+P_6-P_2 & P_1+P_2 \\ P_3+P_4 & P_1-P_3+P_5-P_7 \end{bmatrix}$$

$P_1 = a(f-h)$
$P_2 = (a+b)h$
$P_3 = (c+d)e$
$P_4 = d(g-e)$
$P_5 = (a+d)(e+h)$
$P_6 = (b-d)(g+h)$
$P_7 = (a-c)(e+f)$

Rules for big O Notation.

Analyzing algorithms:

1. Rule of sum: $O(n^2), O(n^3), O(n \cdot \log n)$
   $\Rightarrow O(\max(n^2, n^3))$ gives $O(n^3)$
   $\Rightarrow O(n^3)$ vs $O(n \log n)$
   $\Rightarrow O(\max(n^3, n \cdot \log n))$
   $\Rightarrow O(n^3)$.

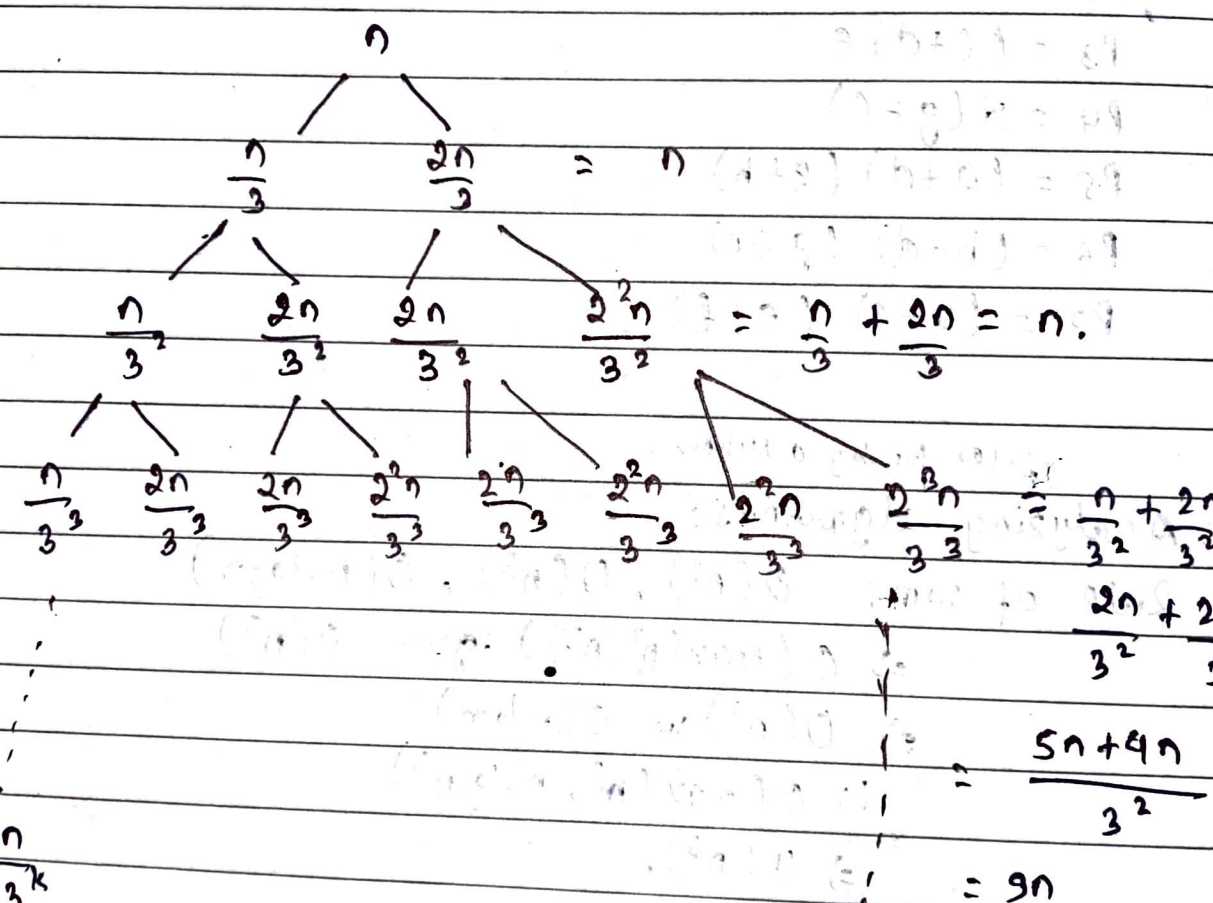
2. Rule of products:

→ Ignoring constant factors: $O(20n^3) = O(n^3)$.
→ Ignoring smaller terms: $O(n^2+n) = O(n^2)$.
→ $O(n(\log n + 1)) = O(n \log n)$.

$1 + r + r^2 + r^3 + \cdots = \frac{1}{1-r} \quad r < 1$

- Recurrance Tree method is a technique used to solve recurrance relations (which are equations that describes the relationship between values of a sequence)
- This method is useful for solving divide-and-conquer algorithm.

Ex: $T(n) = T(n/3) + T(2n/3) + n.$



So total cost is, $n \times \log_{3/2} n.$

$O(n \log_{3/2} n)$

Tree Vertex splitting: $T(V, E, W)$.

→ electric signal, or commodities such as oil are transmitted

Vertex are the receiving stations

Edges are transmission lines.

some loses are there during transmission. (loses means drop in voltage, pressure and so on). (δ)

whereas the loses shouldn't be below the limit; for that boosters are placed

So, TVS is used to determine an optimal placement for booster.

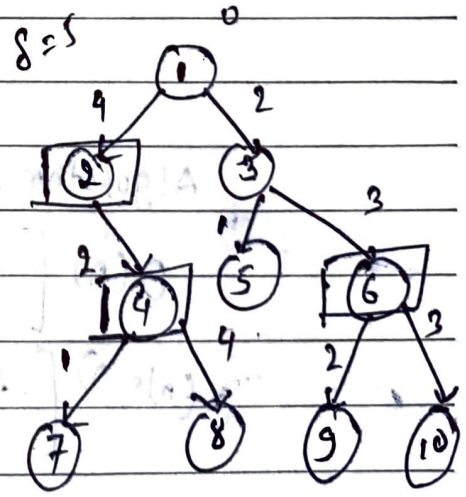
It is assumed that the boosters can only be placed in the nodes of the tree.

W_{ij} = weight bet^n vertex(i) and vertex(j).

{ Number of entering edges = indegree
Number of leaving edges = out degree }

for source, indegree = 0

for sink, outdegree = 0.



Algorithm TVS (T, δ) ← tolerance level
↳ root Node

```
TVS(T, δ)
{
  if (T ≠ 0) {
    d[T] = 0
    for each child v of T {
      TVS(v, δ);
      d[T] = max{d[T], d[v] + W(T, v)}
    }
  if ((T is not the root) and (d(T) + W(parent(T), T) > δ)) then
    write(T); d(T) = 0;
}}}
```

delay

→ Greedy about profit
" " weight
" " Both (P/W)

Knapsack problem (By Greedy method):
aim → maximize the value.

Example: Capacity W = 100

W	10	20	30	40	50
V	20	30	66	40	60
V/W	2	1.5	2.2	1	1.2

Arranging per decreasing order of V/W ratio.

W	30	10	20	50	40
V	66	20	30	60	40
V/W	2.2	2	1.5	1.2	1

Profit = 66 + 20 + 30 + 1.2 × 40
= 164

Algorithm: [knapsack capacity (M)]

O(n) [for i = 1 to n
calculate profit/weight

O(nlogn) [sort objects in decreasing order of P/W Ratio

for i = 1 to n
if M > 0 and Wi ≤ M
M = M - Wi;
P = P + Pi;
else break;
if M > 0
P = P + Pi (M/Wi);

Q.

Profit value(v)	30	20	100	90	160
Weight (w)	5	10	20	30	40
v/w	6	2	5	3	4

(Given W=60)

Arranging in descending order for v/w ratio.

v	30	100	160	90	20
w	5	20	40	30	10
v/w	6	5	4	3	2

$\therefore$ Profit = $30 + 100 + 160 \times \frac{35}{40}$

= 270.

Quick sort:

Worst case: $O(n^2)$ Average case: $O(n \log n)$.

- Chose any element as pivot and swap with the last of array for ease
- item from left larger than pivot & item from Right smaller than pivot swap item.
- Stop when index of itemfromLeft > index of itemfromRight. swap item of Left with pivot.

Ex: 2 6 5 3 8 7 1 0.

Ball → Substitution method 2064 faster (substitution method of a recurrence relation case 4 A solve 518 accurate and f2063)

Date
Page

Master Theorem Method:

The master method ~~gives~~ is general method for determining the asymptotic characterization of a wide variety of recurrance equations.. It is used for recurrance equations of the form :

Master Method (theorem):

$T(n) = aT(n/b) + f(n)$

where, $a \geq 1$, $b > 1$.

Solution is:

$$T(n) = n^{\log_b a}[U(n)]$$

→ $U(n)$ depends upon $h(n)$.

→ $h(n) = \dfrac{f(n)}{n^{\log_b a}}$.

Relation bet^n $h(n)$ and $U(n)$ is

if $h(n)$	$U(n)$
n^r, $r > 0$	$O(n^r)$
n^r, $r < 0$	$O(1)$
$(\log n)^i$ $i \geq 0$	$\dfrac{(\log_2 n)^{i+1}}{i+1}$

Q. $T(n) = 8T(n/2) + n^2$.

Ans. Here,

$a = 8$, $b = 2$, $f(n) = n^2$.

$\therefore T(n) = n^{\log_b a}[U(n)]$

$= n^{\log_2 8}[U(n)]$

$= n^3[U(n)]$

Now, $h(n) = \dfrac{f(n)}{n^{\log_b a}} = \dfrac{n^2}{n^3} = n^{-1}$

since, $r < 0$ $\therefore U(n) = O(1)$

$\therefore T(n) = n^3\, O(1)$

$= n^3 \cdot 1$

$= O(n^3)$

Q. $T(n) = T(n/2) + c$.

Ans. Here, $a = 1, b = 2, f(n) = c$

$\therefore T(n) = n^{\log_b a} [U(n)]$

$= n^{\log_2 1} [U(n)]$

$= n^0 . [U(n)]$

$= [U(n)]$.

Now, $h(n) = \frac{f(n)}{n^{\log_b a}} = c = (\log_2 n)^0 . c$

$\therefore U(n) = \frac{(\log_2 n)^{i+1}}{i+1}$

$= \frac{(\log_2 n)^{0+1}}{0+1}$

$= \log_2 n . c.$

$\therefore T(n) = O(\log_2 n)$

Q. $T(n) = \begin{cases} T(\sqrt{n}) + \log n & \text{if } n \geq 2 \\ O(1) & \text{else} \end{cases}$

Ans. Let $n = 2^m$.

$\therefore T(2^m) = T(2^{m/2}) + \log 2^m$

or $T(2^m) = T(2^{m/2}) + m$

let $T(2^m) = S(m)$

$\therefore S(m) = S(m/2) + m.$

so, $a = 1, b = 2$ and $f(n) = m.$

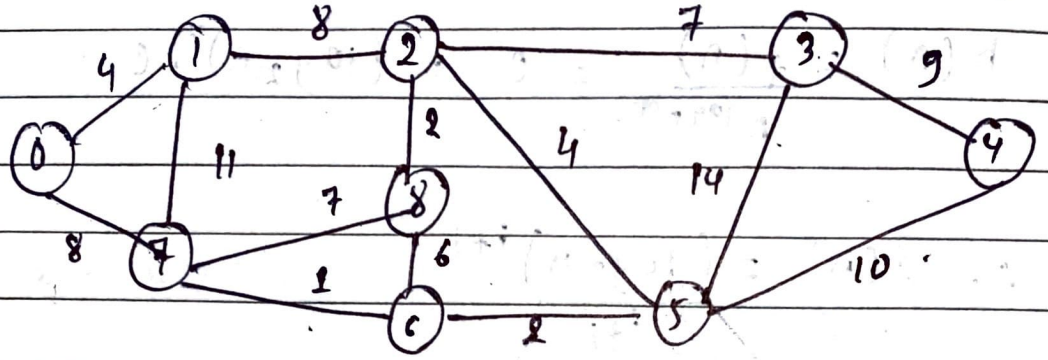
$V-1=E$.

spanning tree:
$v-1=E$
and connected graph.

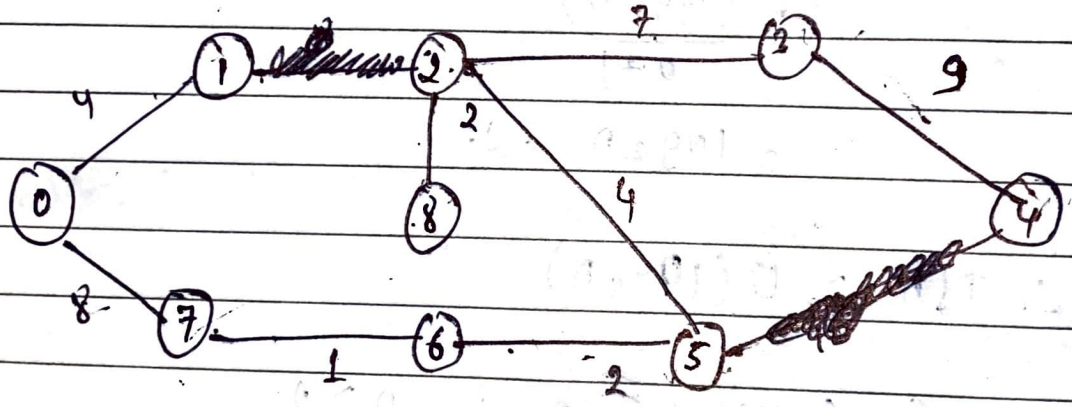
cost

- Minimum spanning Tree:
  1. Prim's Algorithm.
  2. Kruskal Algorithm.

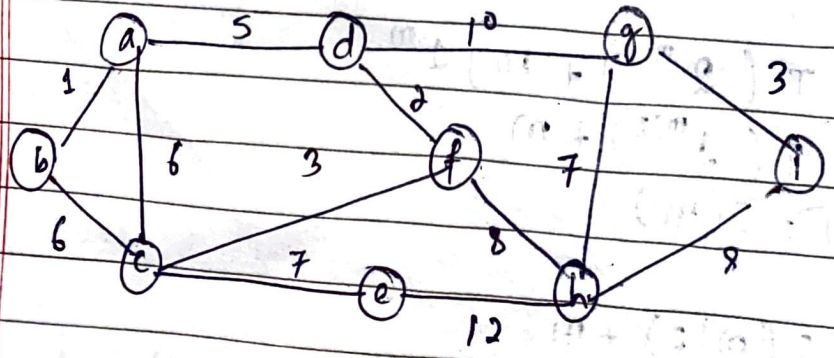
1. Prim's Algorithm:
   - pick any node. (Vertex)
   - select the min^m, ~~from~~ (connected from chosen vertex) compare all but donot form cycle, & check by previous weight also.



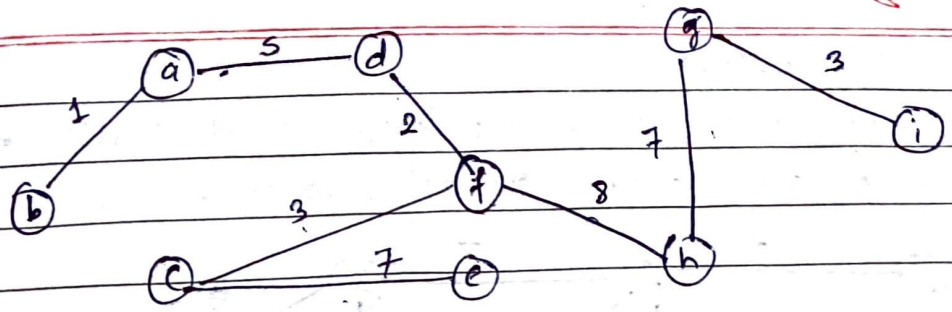
Ans.



$\text{cost} = 4+8+1+2+4+2+7+9 = 37$



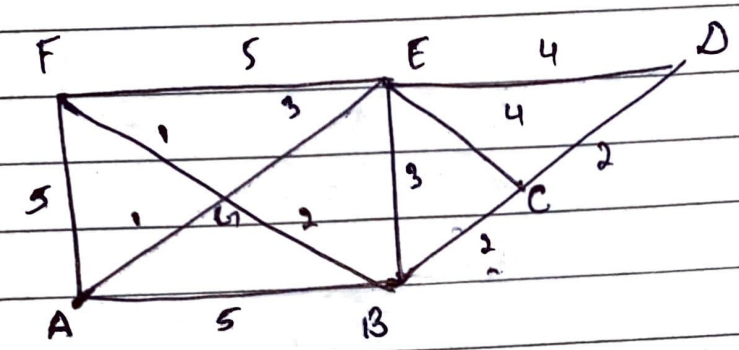
vertex = 9

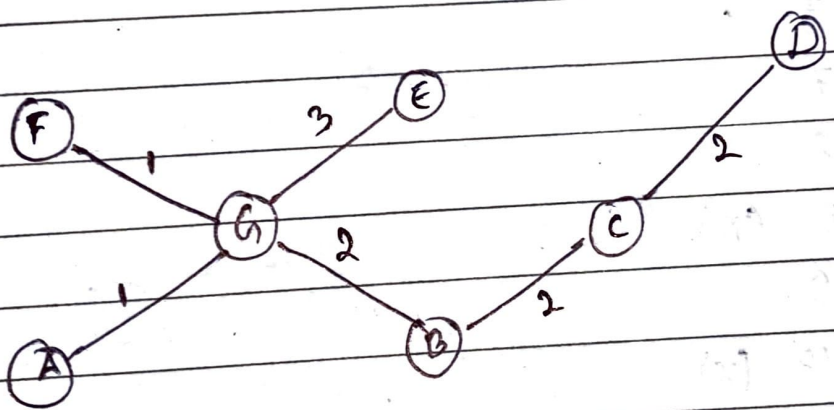


∴ Total Cost = 1+5+2+3+7+8+7+3 = 36

Here, Edges = V - 1 = 9-1 = 8 (which is true)

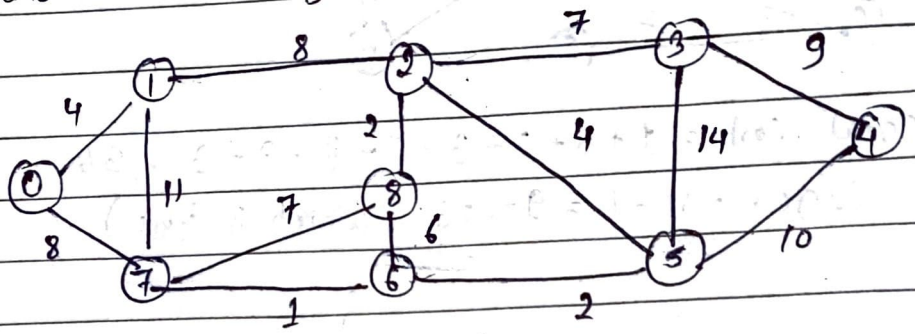
Q.



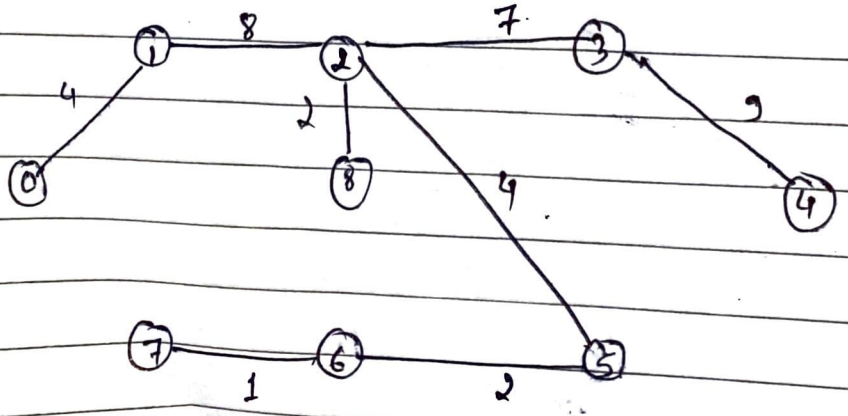


∴ Total Cost = 1+1+3+2+2+2 = 11.

2. Kruskal's algorithm:
Minm edge cost pairing and keeping in descending order. and then making graph without forming loop.

Q. 

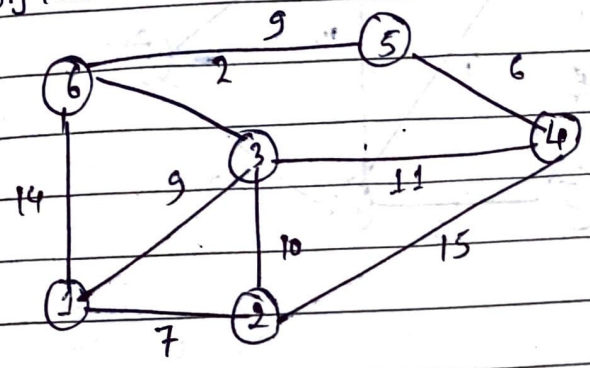
→ {7,6} = 1.
{6,5} = 2.
{2,8} = 2
{0,1} = 4
{2,5} = 4
{8,6} = 6 (x)
{7,8} = 7 (x)
{2,3} = 7
{1,2} = 8
{0,7} = 8 (x)
{3,4} = 9
{5,4} = 10 (x)
{1,7} = 11 (x)
{3,5} = 14. (x)



∴ Total cost = 4+8+2+4+2+1+7+9
= 37

Single Source Shortest Path Algorithm (SPA).

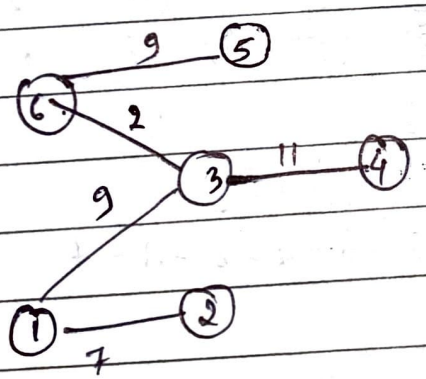
1. Dijkstra's Algorithm: (directed or undirected graph + weighted graph).

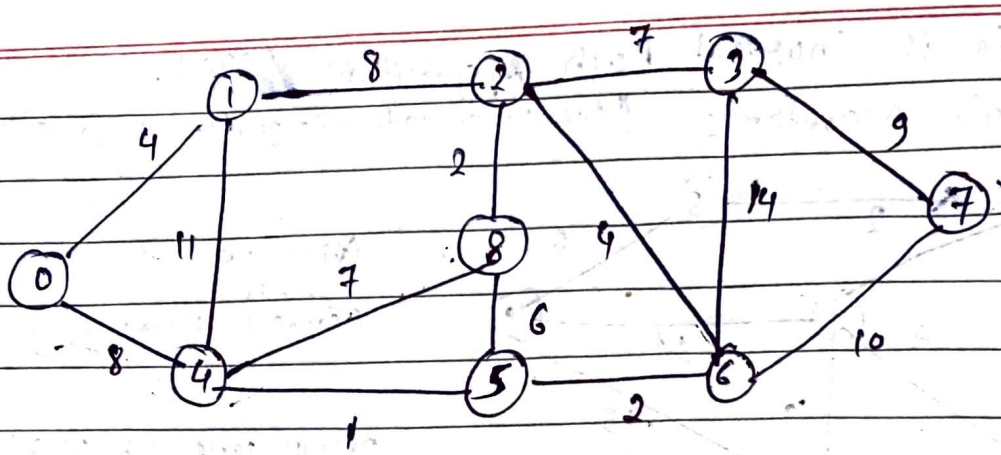


(Node via directly connect नगरेको ∞ हुन्छ)

यसमा some ans मिल्न or may differ.

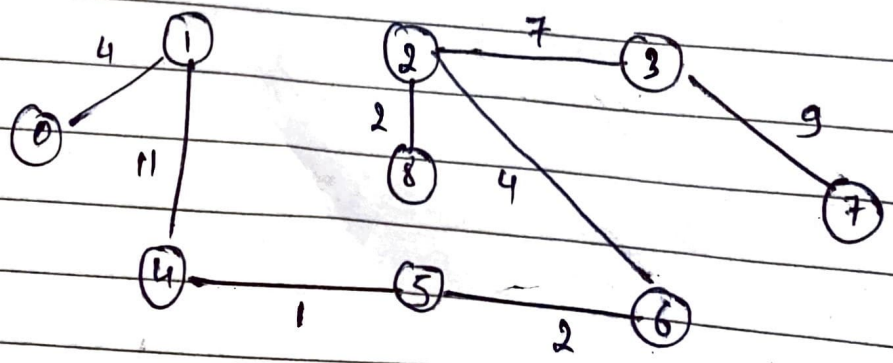
Source 1	Destination 2	3	4	5	6
	∞	∞	∞	∞	∞
	(7)	9	∞	∞	14
1, 2	(7)	(9)	22	∞	14
1, 2, 3	(7)	(9)	20	∞	(11)
1, 2, 3, 6	(7)	(9)	(20)	20	(11)
1, 2, 3, 6, 4, 5.	(7)	(9)	(20)	(20)	(11)





Source	Destination							
0	1	2	3	4	5	6	7	8
	∞	∞	∞	∞	∞	∞	∞	∞
0	(4)	∞	∞	8	∞	∞	∞	∞
0, 1	(4)	12	∞	(8)	∞	∞	∞	∞
0, 1, 4	(4)	12	∞	(8)	(9)	∞	∞	15
0, 1, 4, 5	(4)	12	∞	(8)	(9)	(11)	∞	15
0, 1, 4, 5, 6	(4)	(12)	25	(8)	(9)	(11)	21	15
0, 1, 4, 5, 6, 2	(4)	(12)	19	(8)	(9)	(11)	21	(14)
0, 1, 4, 5, 6, 2, 8	(4)	(12)	(19)	(8)	(9)	(11)	21	(14)
0, 1, 4, 5, 6, 2, 8, 3, 7	(4)	(12)	(19)	(8)	(9)	(11)	(21)	(14)

Shortest path : 0 - 1 - 4 - 5 - 6 - 2 - 8 - 3 - 7.



8 queen's problem:

In 8 queen's problem, we have to place 8 queens on an 8x8 chessboard such that none of them attack one another (ie. no two queens are in the same row, column or diagonal).

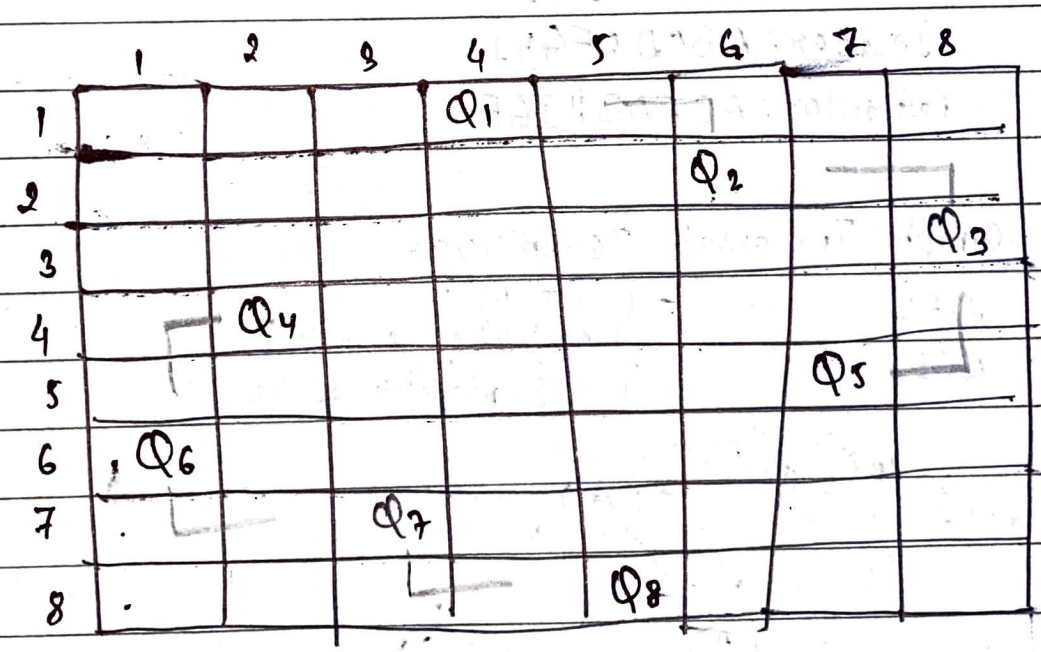
All solutions to the 8-queens problem can be represented as 8-tuples $(x_1, \ldots, x_8)$, where x_i is the column of the i^{th} row where the i^{th} queen is placed.

Explicit constraint : $1 \leq x_i \leq 8$.

Implict constraint : (same as 4 queen problem).

- No 2 x_i's can be same (i.e. all queen must be placed on different columns)
- No 2 queens can be on the same diagonal.

1 → 4 column
8 → 5 column.



8 x 8 chessboard.

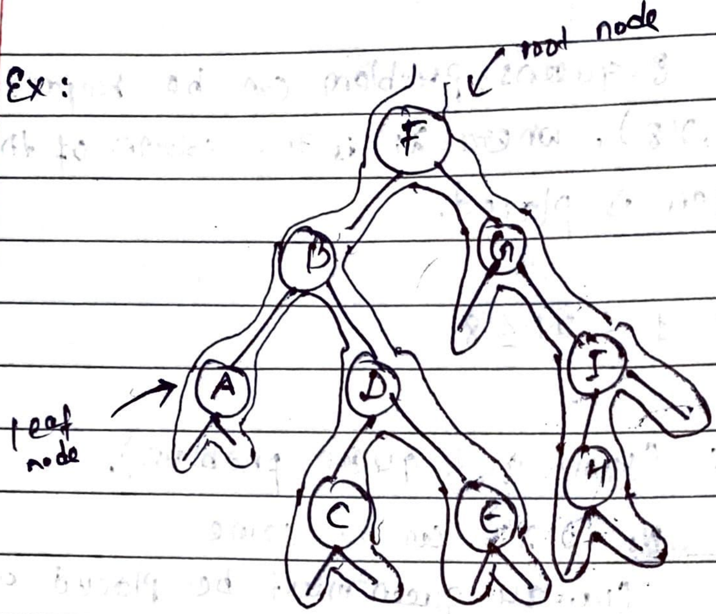
Solution vector:

	1	2	3	4	5	6	7	8
x_i	4	6	8	2	7	1	3	5

Tree Traversal Technique:

- Preorder : Root Left Right. (1)
- Inorder : Left Root Right. (2)
- Post order : Left Right Root. (3)

Ex:



Preorder : FBADCEGIH

Inorder : ABCDEFGHI

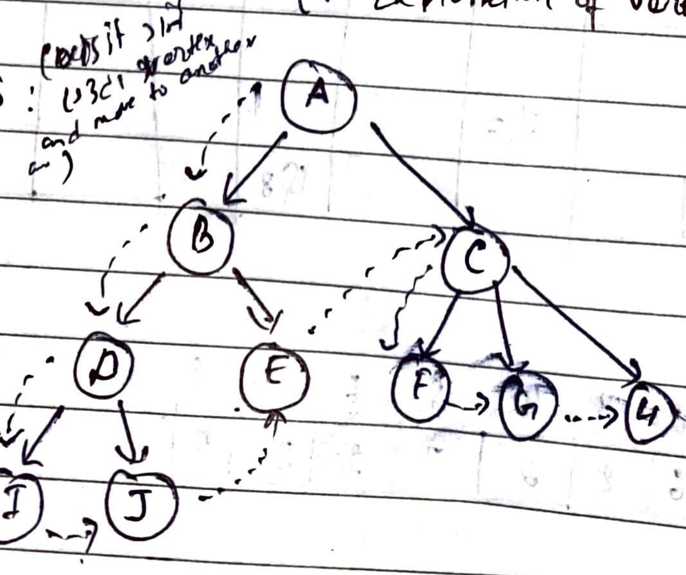
Post order : ACEDBHIGF

Graph Traversal Technique:

- DFS
- BFS.

two things:
- Visiting a vertex
- Exploration of vertex

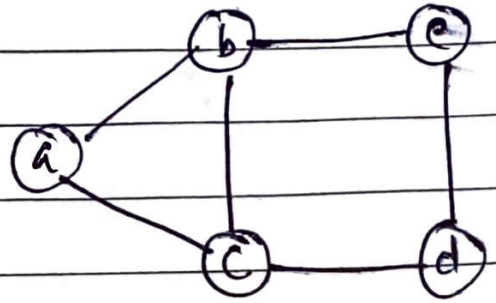
(1) DFS :



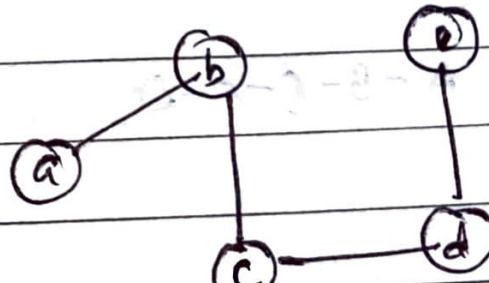
{ we can make spanning tree by doing traverse using BFS or DFS }

Date
Page

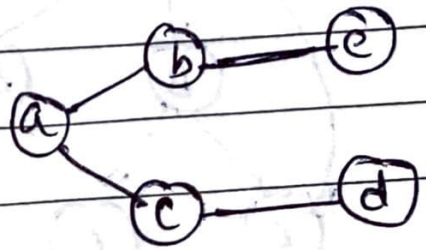
let G be a simple graph. A spanning tree of G is a subgraph of G that is a tree containing ~~vert~~ every vertex of G.



Spanning tree created by DFS.



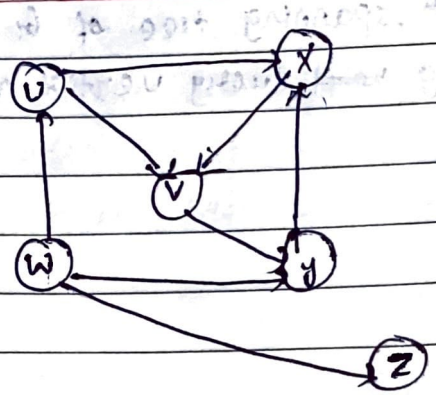
Spanning tree created by BFS.



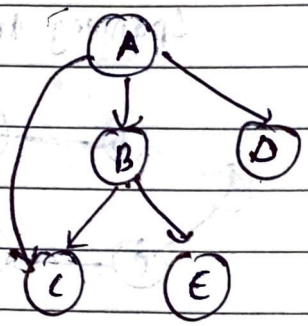
{ Tree is also Graph }

Date
Page

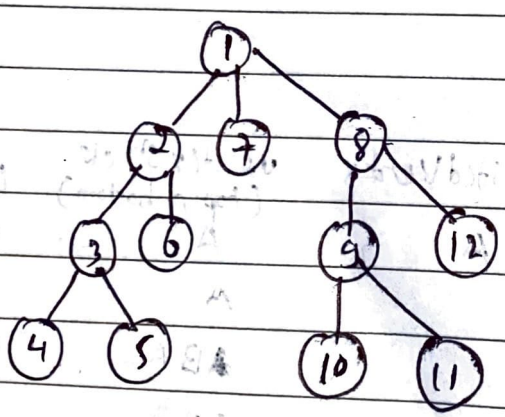
DFS Exercise:



Ans: W U V Y X Z

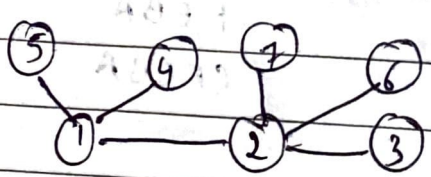


Ans: A-B-C-E-D



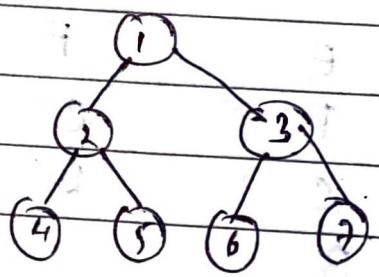
Ans: 1-2-3-4-5-6-7-8-9
-10-11-12

Q.



BFS: 1, 2, 4, 5, 7, 6, 3
DFS: 1, 2, 3, 6, 7, 4, 5

Q.

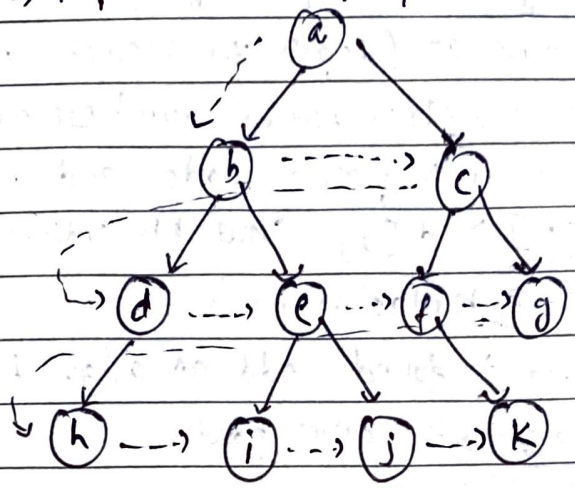


BFS: 1, 2, 3, 4, 5, 6, 7. (level order)
DFS: 1, 2, 4, 5, 3, 6, 7 (pre order)

BFS: (explore गरेर अर्को and move on to next)

↳ checks node of same level.

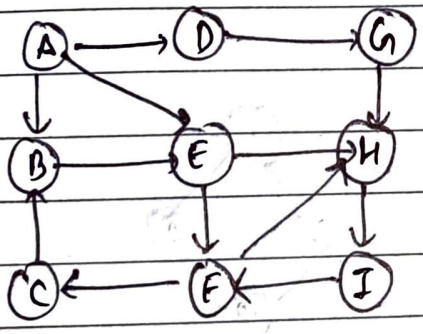
↳ implemented using queue.



from BFS: a b c d e f g h i j k

from DFS: a b d h e i j c f k g

Q.

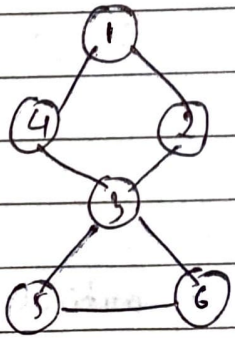


BFS: A B D E G F H C I

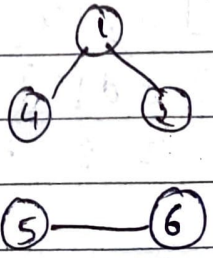
(cut point)

An articulation point, also known as a cut vertex, is a vertex in graph that, if removed along with its incident edges, would increase the number of connected components in the graph.

Ex:



Here, 3 is articulation point.

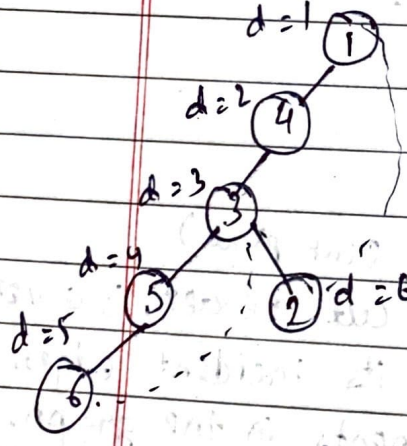


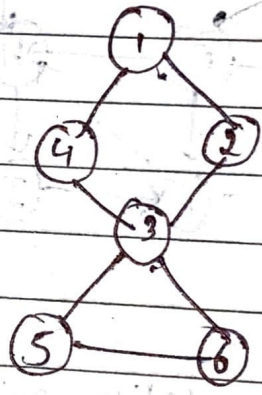
To remove AP, connect components with edge.

Algorithm for finding Articulation point.

1. Prepare DFS spanning tree for given graph.
2. Maintain discovery number (depth for search number); if any back edge maintain lowest number also.
3. Create array u, v where u is parent node and v is child node. and compare $L[v] \geq d[u]$ (this condition is for all vertices except root node) if true, then u is articulation point.
   If articulation point is found add an edge between the components. to remove articulation point.
4. To check if root is AP;
   if root is multiple children then root is articulation point. In our below example, root is having only one child so it is not A.P.

Ex:





	1	2	3	4	5	6
d =	1	6	3	2	4	5
L =	1	1	1	1	3	3.

$L[v] \geq d[u]$

$L[2] \geq d[3]$

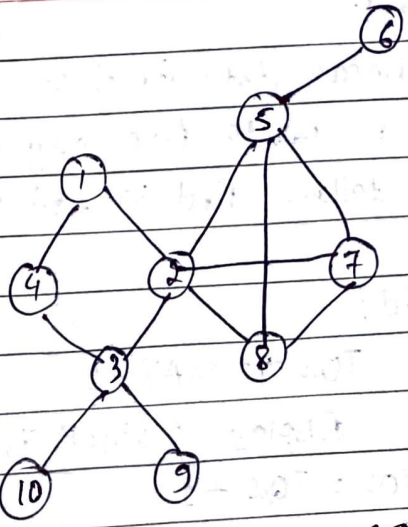
$\Rightarrow 1 \geq 3$ true $\therefore$ 3 is articulation point.

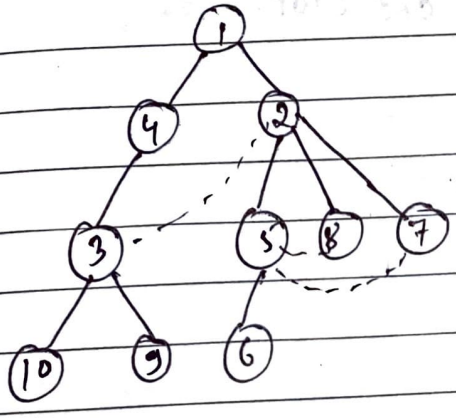
common in both:
(can start from any vertex, while visiting, any order you like)

BFS & DFS Spanning tree:

BFS: 1, 4, 2, 3, 5, 8, 7, 10, 9, 6

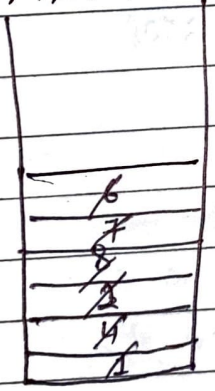
Q | ~~1~~ | ~~4~~ | ~~2~~ | ~~3~~ | ~~5~~ | ~~8~~ | ~~7~~ | ~~10~~ | ~~9~~ | ~~6~~ |

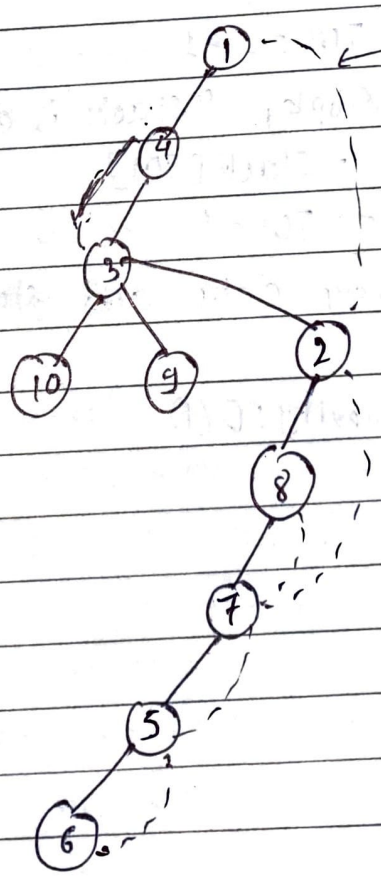




[... बाकी का ~~already~~ ~~visited~~ cross edges
dotted edges.]

DFS: 1, 4, 3, 10, 9, 2, 8, 7, 5, 6





DFS मा एउटा adjacent pickup गरेर Explore गर्ने
BFS मा सबै adjacent टिप्ने ।
(explore पछि गर्ने)

Stack:

- a linear data structure in which insertion and deletion can be made from only one end called as 'Top of Stack' (TOS).
- It follows First in Last out. / Last in First Out (LIFO).

Add:

1. If TOS == MAX − 1
   Display "Stack is full" and STOP.
2. TOS = TOS + 1
3. stack [TOS] = data.
4. STOP.

Complexity: O(1)

Deletion:

1. If TOS == −1
   Display "Stack is empty" and STOP.
2. data = Stack [TOS]
3. TOS = TOS − 1.
4. Display data and stop.

Complexity: O(1)

Binary Search Tree (BST):

- has binary nodes.
- each node to the left is "smaller" and to the right is "greater" and so-on till the end of the tree.
- faster as compared to linear searching.
- But for faster and optimized search, the tree must be balanced one.

Operation	Time Complexity
search	O(logn) best O(n) worst
Add	O(logn) best O(n) worst
remove	O(logn) best O(n) worst.

Delete an element from BST:

a. Start at the root of BST.
b. If the root is null, return null as the BST is empty.
c. If the key to be deleted is less than the value at the root node, then recursively call the delete function on the left subtree of the root.
d. If the key to be deleted is greater than the value at the root node, then ~~there are~~ recursively call the delete function on the right subtree of the root.
e. If the key to be deleted is equal to the value at the root node, then there are three cases to consider:
   a. If the node to be deleted is a leaf node (i.e, it has no children) simply delete it by setting the parent's pointer to null.
   b. If the node to be deleted has only one child, replace it with its child by setting the parent's pointer to the child.
   c. If the node to be deleted has two children, find the node with the minimum value in its right subtree (i.e, the leftmost node in the right subtree), replace the value of the node to be deleted with the value of minimum node, and delete the min node.

Queue:

Algorithm to insert an element in circular queue:

Step 1: IF (REAR+1) % MAX = FRONT
Display "OVERFLOW"
Goto Step 4
END IF.

Step 2: IF FRONT = -1 and REAR = -1
SET FRONT = REAR = 0.
ELSE IF REAR = MAX - 1 and FRONT != 0
SET REAR = 0
ELSE
SET REAR = (REAR + 1) % MAX
END IF.

Step 3: SET QUEUE [REAR] = VAL.
Step 4: EXIT.

Algorithm to remove an element in a circular queue.

step 1: IF FRONT = -1
Display "UNDERFLOW"
Goto step4
END IF.

Step 2: SET VAL = QUEUE [FRONT]

Step 3: IF FRONT = REAR
SET FRONT = REAR = -1

Binary tree vs Binary Search tree

↓ no restriction for value order

↓ left ko small right node ko Greater

Date
Page

```
ELSE
    IF FRONT = MAX - 1.
        SET FRONT = 0
    ELSE
        SET FRONT = FRONT + 1.
    END IF
END IF
```

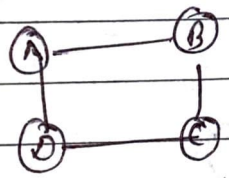
Step 4 : Exit.

Graph Coloring:

Let G = (V, E).

It means color each and every vertex of Graph such that no two adjacent vertex have same color.

Ex:



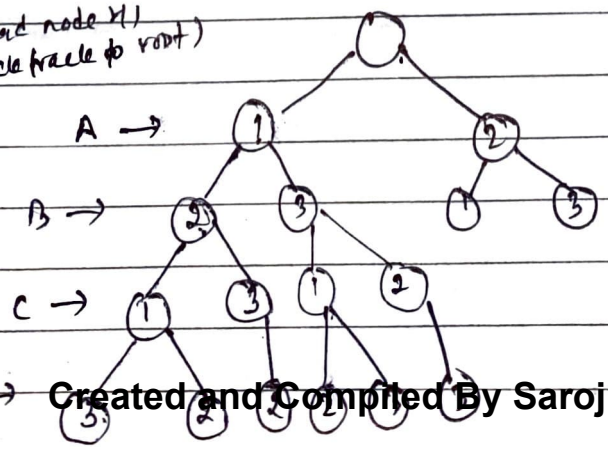
Color of A ≠ Color of B and Color of D
Color of B ≠ Color of A and Color of C
Color of C ≠ Color of B and Color of D.
Color of D ≠ Color of C and Color of A.

Chromatic Number: The min^{m} no. of colors required to color the graph is called chromatic number of graph.

let m = 3
1, 2, 3.

State space tree

(Dead node ko skip and backtrack to root)

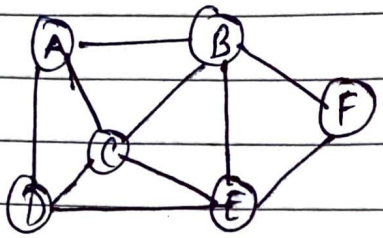


Hamiltonian Cycle:

$G = (V, E)$

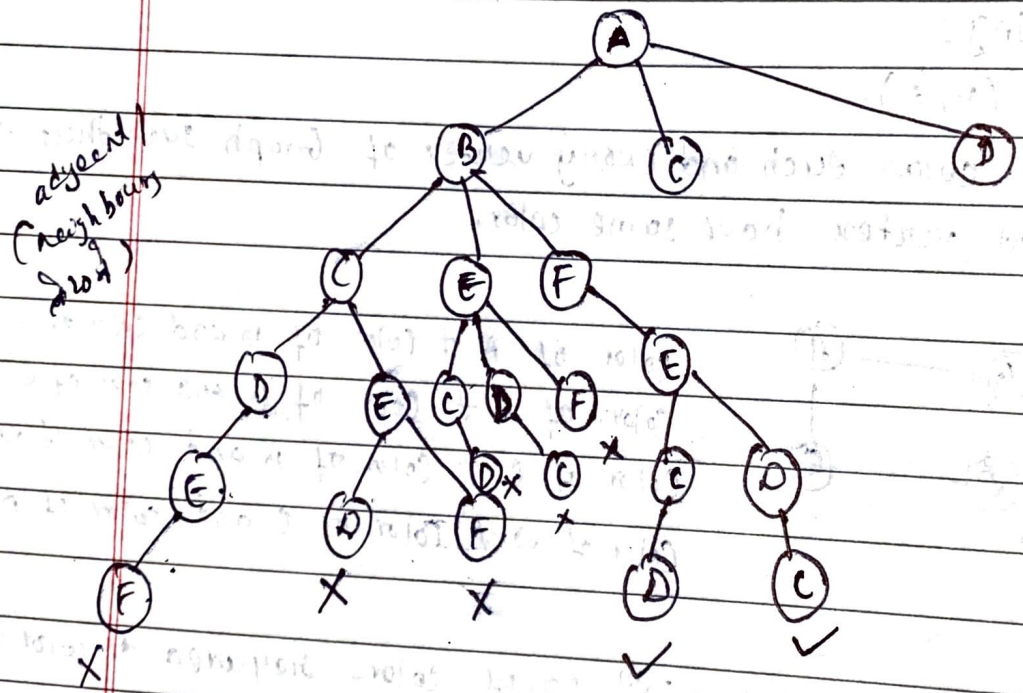
It is roundtrip in the edges of G such that each vertex is visited once and stops to the starting same vertex.

Ex:



=> ABFECD

=> ABFEDC.

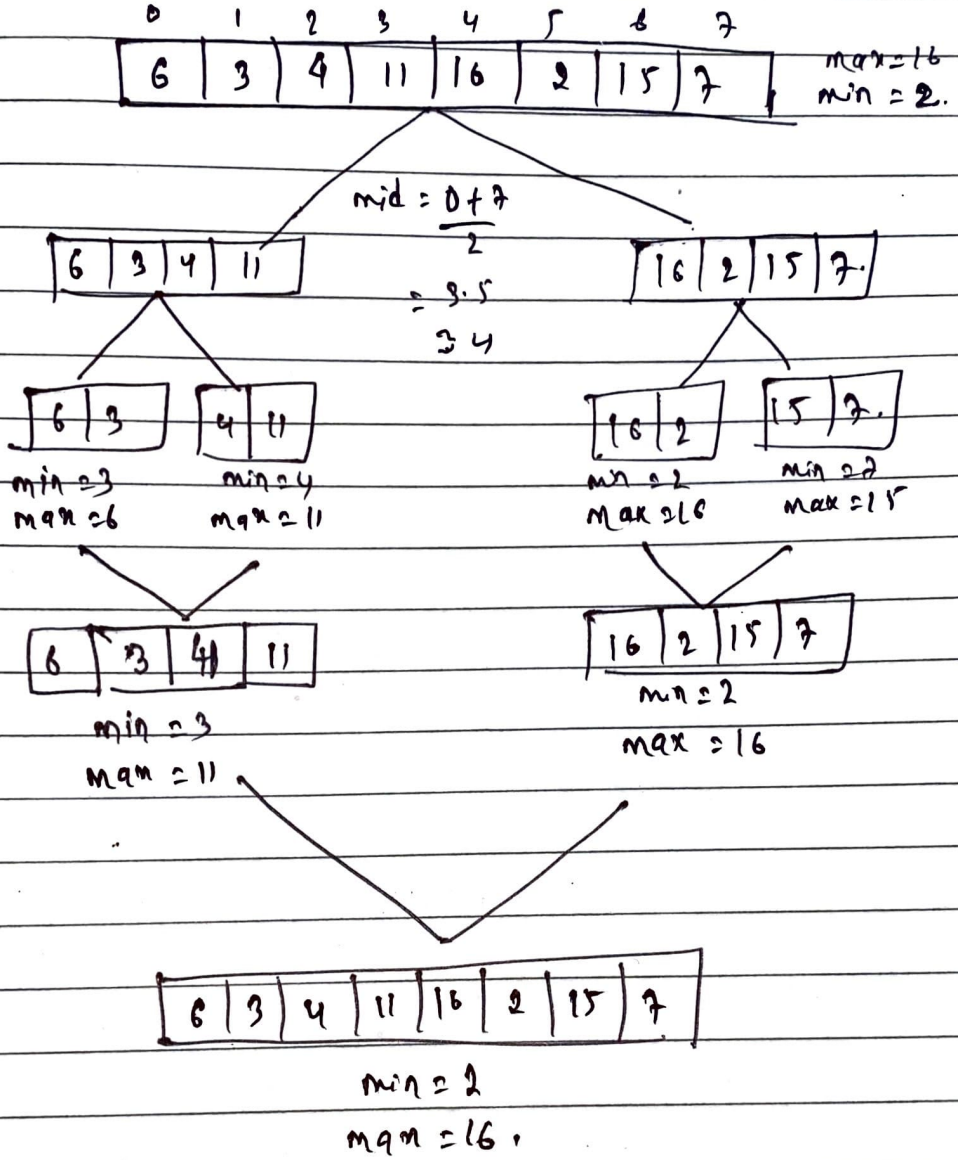


state space tree.

Min-max using Divide Conquer.

{ Normal min-max : O(n) }.

Ex:



Optimal storage on Tape:

$$T_j = \sum_{k=1}^{j} L_k$$

$$MRT = \frac{1}{n} \sum_{i=1}^{N} T_j = \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{i} L_k.$$