Techniques for Binary Trees:

Traversing a tree means visiting a node in tree only for once.

Their are three ways of traversing:

1) Preorder
2) Inorder
3) Postorder

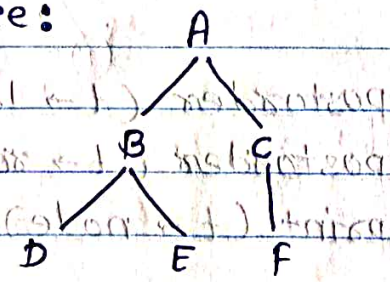
① Preorder:

a) Visit to root.
b) Traverse leftsubtree.
c) Traverse rightsubtree.

- Algorithm for preorder:

```
preorder (tree t)
{
    if (t != 0)
    {
        print (T→ data);
        preorder (T→ Lchild);
        preorder (T→ Rchild);
    }
}
```

- Traverse tree:



Preorder : ABDECF

② Inorder :

a) Traverse the left subtree.
b) Visit root Node.
c) Traverse right subtree.

• Algorithm for Inorder :

```
Inorder (treet)
{
    if (t != 0)
    {
        Inorder (t → lchild);
        print (t → data);
        Inorder (t → rchild);
    }
}
```

• Traverse tree : DBEACF

③ PostOrder :

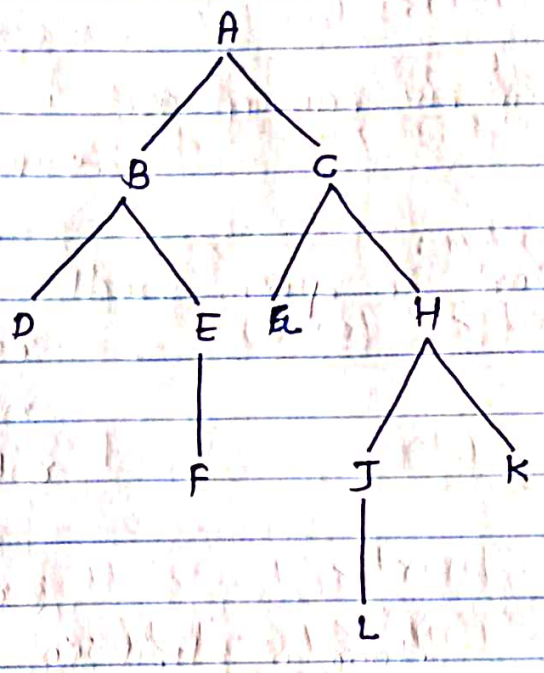
a) Traverse left subtree.
b) Traverse right subtree.
c) Visit root node.

• Algorithm for Postorder :

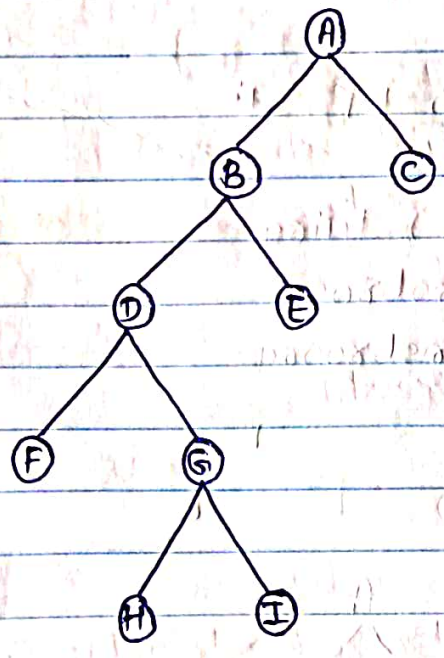
```
postorder (treet)
{
    if (t != 0)
    {
        postorder (t → lchild);
        postorder (t → rchild);
        print (t → node);
    }
}
```

• Traverse tree : DEBFCA

* find preorder, Inorder & postorder :



a) Preorder = ABDEFCGHJLK
b) Inorder = DBEFAGCLJHK
c) Postorder = DFEBGLJKHCA



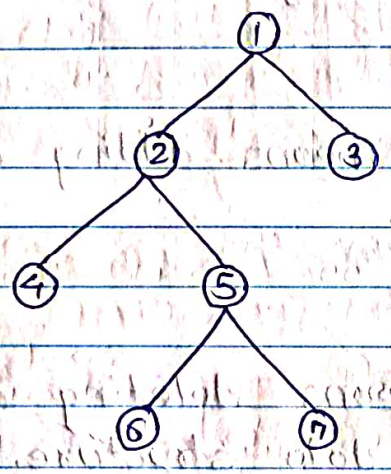
a) Preorder =
b) Inorder =
c) Postorder =

Binary Tree:

Binary tree is a set of nodes which is empty & divided into three disjoint sets namely, root, Right subtree, left subtree. provided that any node have maximum two children.

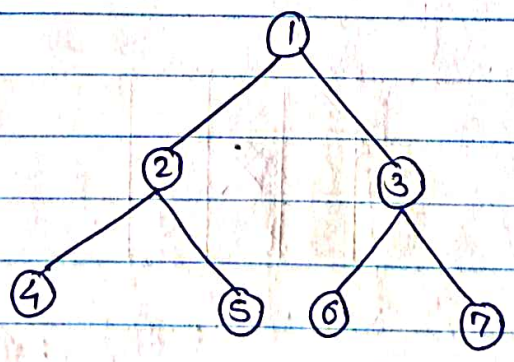
① Strictly Binary Tree:

It is a binary tree where all non-leaf nodes have two branches.



② Complete Binary Tree:

In the complete binary tree, it is a strictly binary tree where all non-leaf nodes are at same level.



1.24

Operations of Binary Tree:

There are three types of operations which is required to maintain a binary search tree.

a) Searching
b) Insertion
c) Deletion

a) Searching:

Searching means finding the location of data element from the given list.

b) Insertion:

Insertion means adding a new element in the data structure.

c) Deletion:

Deletion means deleting or removing an element from given data structure.

Game Tree:

Game tree is a directed graph whose nodes are position in games and whose edges are moves.

It is graphical representation of sequential game. It provides information about,

1) players
2) playoffs
3) strategies
4) order of moves.

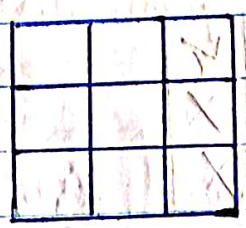
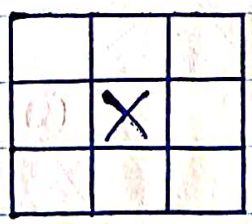


figure: Game tree or Tic-Tac-Toe

In Tic-Tac-Toe game tree structure have initial or first level of the game. a player have freedom or player faced any one of the nine moves.

In this game tree structure two players are play for winning with maximum.

Player I having maximum probability to win the game.



player 1 = X

player 2 = ✓

1.22

In second

		✓
	X	

3)

		✓
	X	
X		

4)

	✓	✓
	X	
X		

5)

X	✓	✓
	X	X

6)

X	✓	✓
✓	X	X

7)

X	✓	✓
✓	X	X
		X

Ie: first player will win the game (probability of winning is high with player first).

Graph:

A graph is a collection of two sets V & E where V is collection of vertices & E is collection of ~~ve~~ edges.

A graph can have 2 types.

1) Undirected graph.
2) Directed graph.

① Undirected graph:

It is a graph which has un-ordered pair of vertices.

② Directed graph:

It is graph which has ordered pair of vertices $<V_1, V_2>$ where V_1 is a tail & V_2 is head.

Techniques for Graphs:

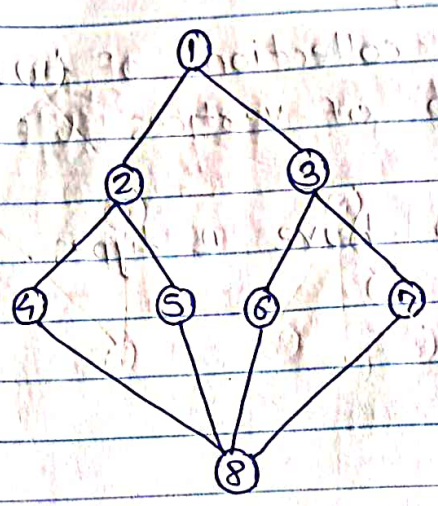
Their are two main techniques of graph

1) Breadth First Search (BFS)
2) Depth first Search (DFS)

In traversal of graph the graph should be undirected so every node be reached from each other.

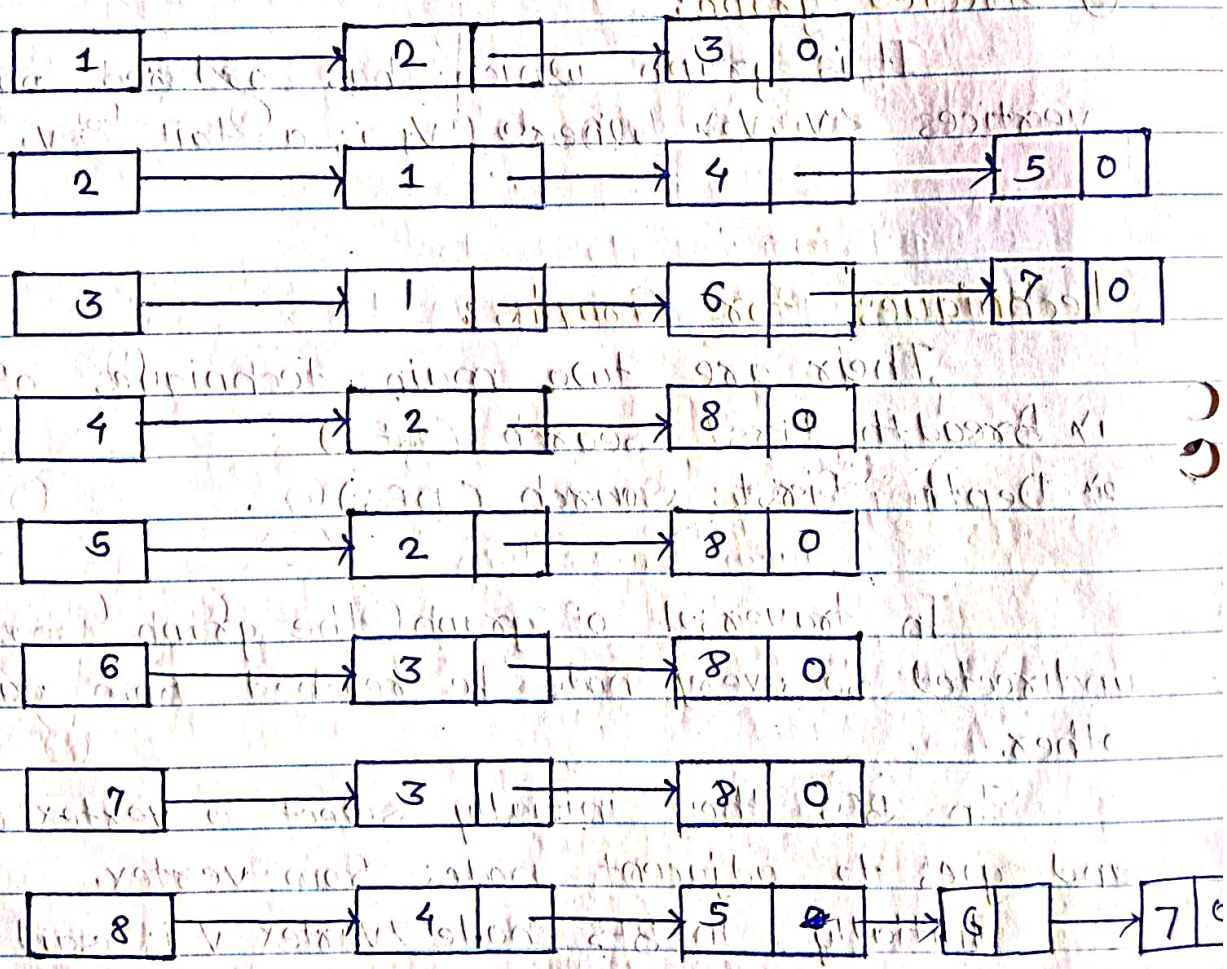
In BFS they initially select a vertex (V) and goes to adjacent nodes from vertex.

Initially in BFS node / vertex V is said to be unexplored when all the adjacent nodes from vertex V are visited then V is said to explored.



(Ka) undirected graph G

Node



Adjacent list for Graph G

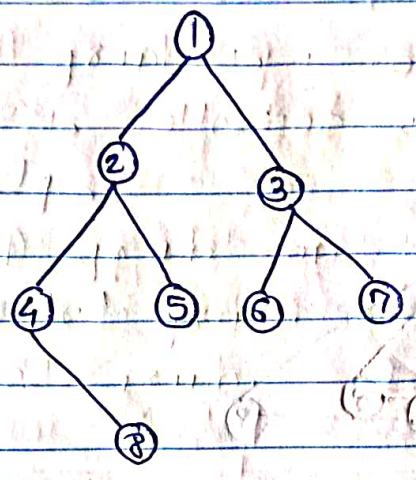


fig: BFS spanning Tree (BFS)

Algorithm BFT (G,n)

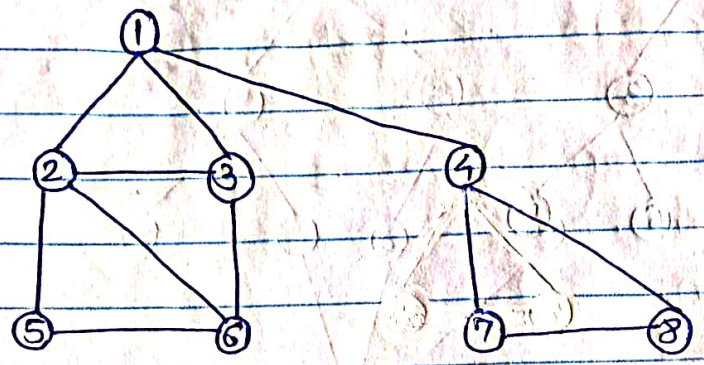
```
{
    for i=1 to n do
        visited[i]=0;
        for i=1 to n do
        if (visited[i]=0) then BFS (i);
}
```

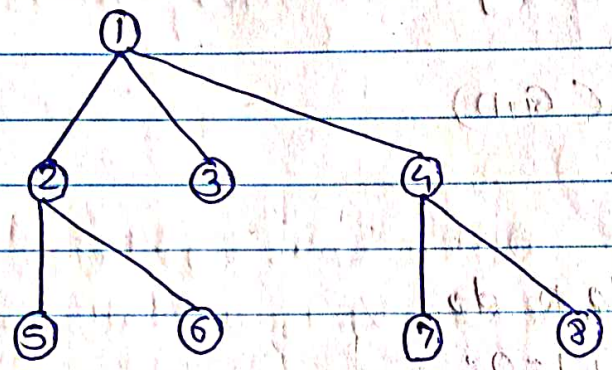
Time complexity: $\Theta(n)$

queen operation

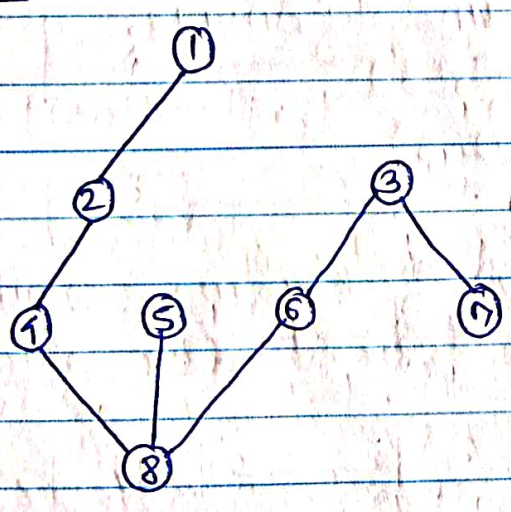
find out BFS spanning tree for following graph:

BFS





DFS



Q1) Explain indetail different techniques of graph. Traversal techniques in detail

Their are two main techniques of graph.
① Breadth First search of traversal.
② Depth first search traversal.

1) Breadth first search of traversal:

In BFS we start at the vertex v of mark it has having been visited the vertex v is at this time said to be unexplored. A vertex is said to hear been explored by an algorithm, when the algorithm has visited all adjucent from v are visited next. There are new unexpected vertices. Vertex v has be explored & are part onto end of a list of unexplored vertices havent been explored vertices is left the list at unexplored vertices operates as a queue of can be represented using any of the standerd queue representation.

Algorithm BFS (v):

```
{
    u:=v;
    visited [v]=1;
    repeat
    {
        Let all vertices adjent
        from ado
        {
            if (visited [w]=0) then
```

explore the vertex v then

once we start exploring
then visited a new vertex
we suspend the vertex
and then start the
vertex

for
exploration

```
{
    Add w to q;
    visited [w] = 1;
  }
}
if q is empty then return;
  Delete v from q
}
until (false);
}
Algorithm BFT (G,n)
{
  for i=1 to n do
    listed [i] = 0
    for i=1 to n do
      if [visited [i] = 0] then BFS(i);
  }
```

② Depth first search of traversal:

A DFS of a graph delivers from BFS in that the expression of the vertex v is suspended as soon as the new vertex is reached. At this time the exploration of the new vertex is begines when their new vertex has been explored the exploration of v continious.

The search terminates when all reached vertices, have been fully explored. This search process ie best described recursive in following algorithm.

Algorithm DFS(V):-

```
{
    visited [V]=1;
    for each vertex w adjacent from V
    do
    {
        if (visited [w]=0) then DFS [w];
    }
}
```

Q2) Explain connected component & spanning tree:

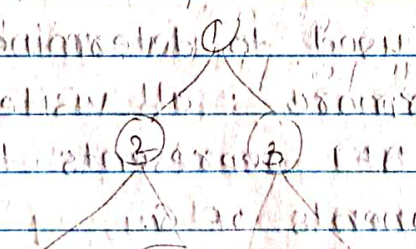
All the vertices of graph will get visited on the 1'st call to BFS. If it is not connected then atleast two calls to BFS will be needed, hence BFS can be used to determine where G is connected furthermore ; all visited vertices on a call to BFS from BFT represents the vertices in a connected components of G.

Hence the connected components of can be obtained using BFT for this BFS can be modified so that newly visited vertices are put onto a list. Then the subgraph formed by the vertices on their list make up a connected components hence the adjacency list one used. breadth 1'st traversal hence the adjacency list are used.

The breadth first traversal will obtain the connected components in O(n+e) time. Breadth first traversal can also be used to obtain the reflexive, transmitive closer matrix of an undirected graph G It at this matrix ~~of an undirected~~ then matrix A(i,j)=1 it's either i=j or i=j of i & j

i.e $i=j[i,j]$ or $i \neq j$ one in same connected components we can set up in $o(n+e)$ time an array connect such that connect[i] is the index set, connect[i] is the index do the connected components connecting the vertex i, $1 \leq i \leq n$ hence we can determine wheather $a(i,j)$ $i \neq j$ is 1 or 0 by simply seeing wheather connect[i] = connect[j] The reference transmitive, closer matrix of an undirected graph.

With 'n' vertices of e edges can therefore be computed in $o(n^2)$ here $o(n)$ space using either adjency lists or machines.



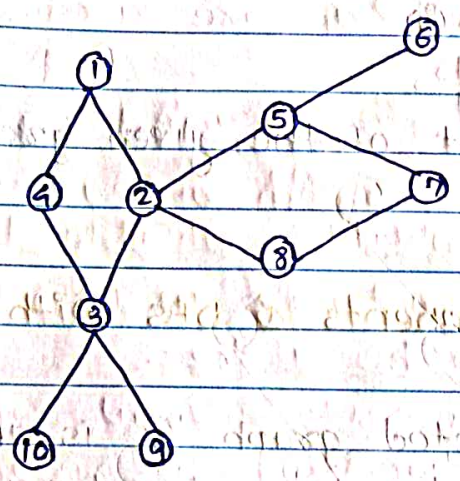
BFS is just like level order traversal of binary tree.

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

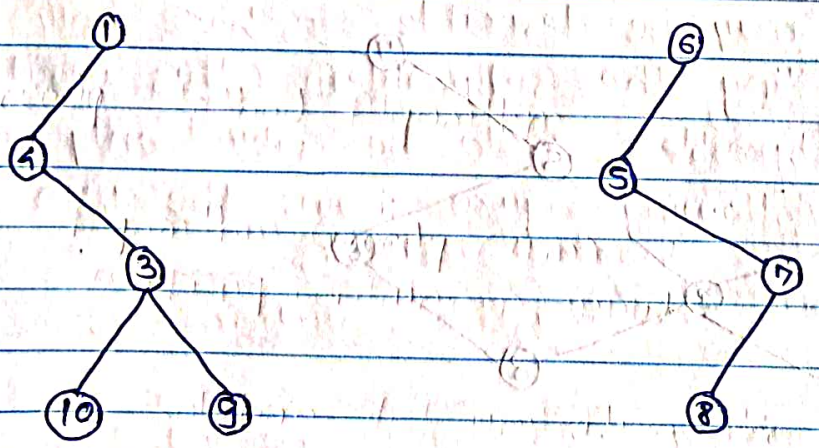
Q3) Define Artifilation point with example.

Artifilation Point:

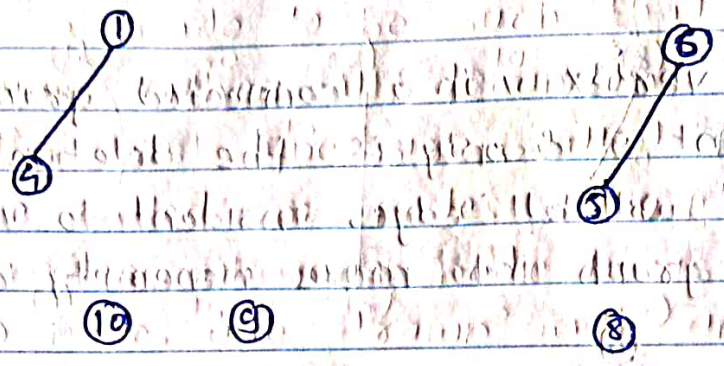
A vertex v in a connected graph G is an artifilation point if & only if the deletion of vertex v together will all edges insident to v disconnected the graph into more nonempty components



In above graph there are three artificial points 2, 3, & 5 so suppose we delete the node ② we get two nonempty components of the above graph.



again by using artifilation points

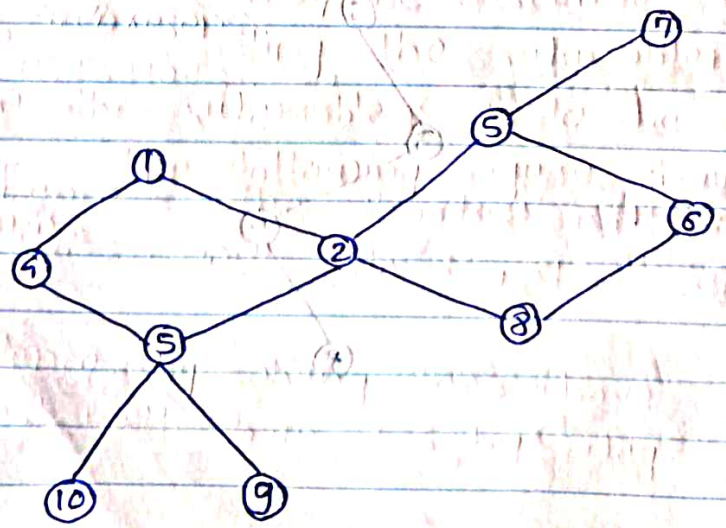


Hence it is the result of the given graph.

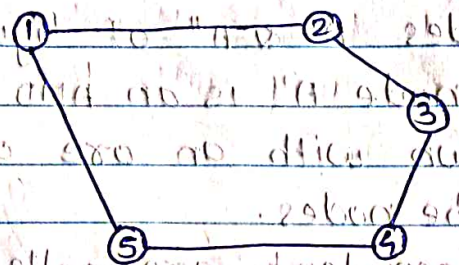
Q4) Explain biconnected components & DFS with example.

A vertex v in a connected graph 'G' is an artifilation point iff the detection of vertex v together with all edges insident to 'v' disconnected the graph into two or more nonempty components.

A graph G is the biconnected iff it contains no artifilation point.

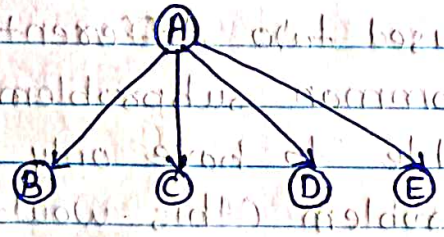


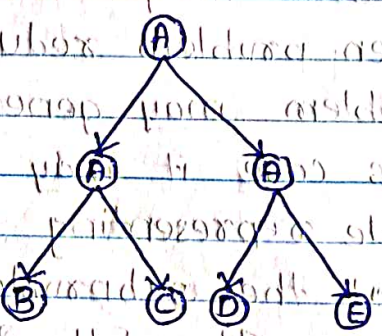
Above graph is not biconnected as it contains three artification points vertex ② vertex ③ & vertex ⑤ are three artifilation points in above graph.



The above graph has only one biconnected components (ie the entire graph) so the above graph is a biconnected graph.

Q5) Write a note on AND / OR Graph





The graph of fig (a) represents problem A that can be solved by solving either both subproblem B & C or subproblem D or E.

Graph of subproblem that must be solved in order to imply a solution to parent node are joined together by an are going access the respective edges (as they are access the edge <A,B> & <A,C>

By introducing dummy nodes in fig (b) all nodes can be made to be such that their solution requires either all decandent to be solved nodes of the first type are called AND nodes of the latice called or nodes A' & A'' of figure (b) are or nodes whereas node A'' is an AND node. The AND nodes are drawn with an are accross an edge reaching the nodes.

Nodes with no decendent are called terminal node represents premitive problem and are marke either solvable or not solvable. solvable terminal nodes are represented by rectangle an AND/OR graph need not always to be a tree.

Breaking down problem into several subproblem is known as problem reduction. problem reduction has been used on such problem as theorem providing symbolic integration & analysis of industrial shedules when problem reduction is used two different problem may generate a common subproblem in this case it may be desirable to have only one node representing the subproblem (this would imply that the subproblem is to be solved only once)

The following figure shows the AND/OR graphs.

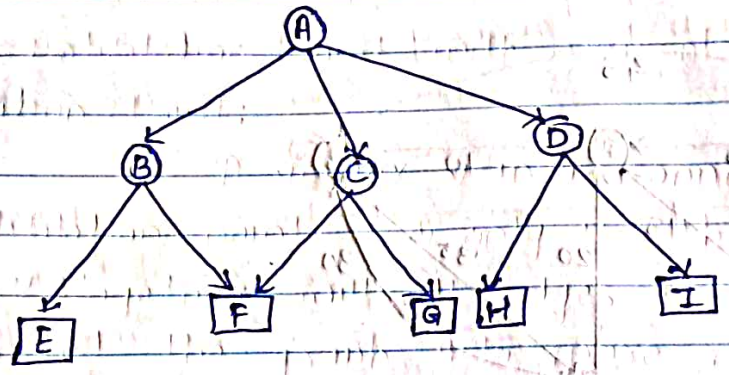


fig ⓐ

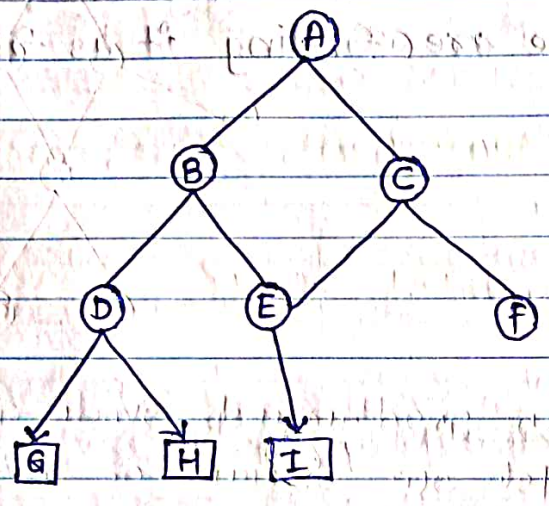
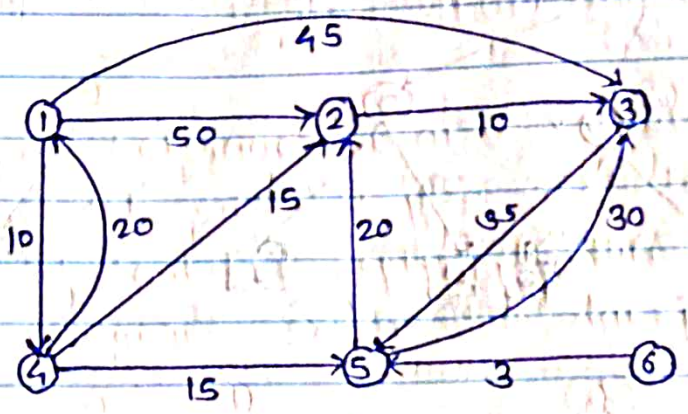


fig Ⓑ

Greedy Method Algorithm

```
Algorithm Greedy (a,n)
// a[1:n] contains n inputs
{
    solution = ϕ // initialize the solution
    for i=1 to n do
    {
        x=select (a);
        if feasible (solution : x) then
        solulution := union (solution, x);
    }
    return solution;
}
```

single source shortest path



Now we have to find the shortest path from 1 to 5 hence we are finding it as follow:

① 1 - 2 - 3 - 5 ⇒
50 + 10 + 35 = 95

② 1 - 3 - 5 ⇒
45 + 35 = 80

③ 1 - 4 - 5 ⇒
10 + 15 = 25

④ 1 - 4 - 2 - 3 - 5 ⇒
10 + 15 + 10 + 35 = 70

hence ⇒ The shortest path is
1-4-5 = with the feasible solution = 25

Hence, by solving the above, we get the shortest path from 1-5 by way 1-4-5

* Hamiltonian Cycle:

Let $G = \langle V, E \rangle$ be a connected graph with v vertices.

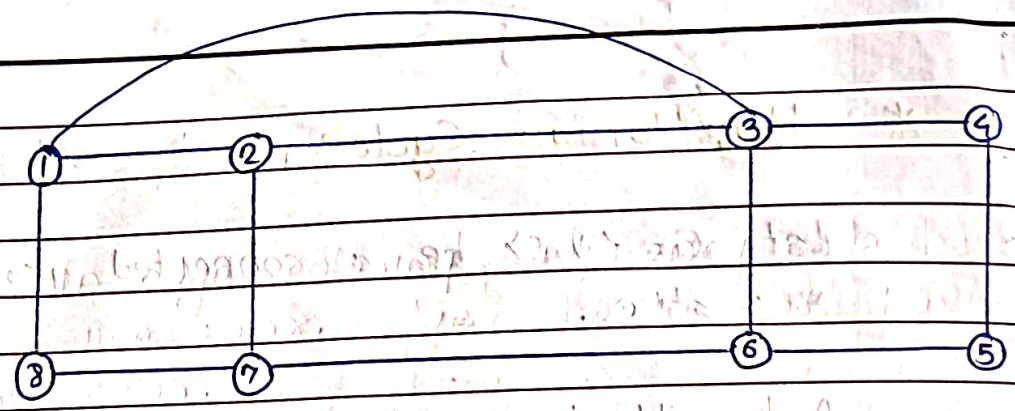
A hamiltonian cycle is a round-trip path along n edges of G that visits every vertex once & return to it's starting position.

In other words, if a hamiltonian cycle begines some vertex $V_1 \in G$ & the vertices of G are visited in the order $V_1, V_2, V_3 \ldots V_{n+1}$ then the edges (V_i, V_{i+1}) are in E, $1 \leq i \leq n$.

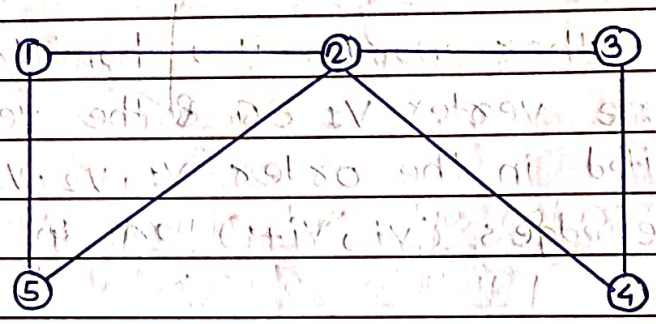
following are the algorithm for finding all hamiltonian cycle.

```
Algorithm Hamiltonian (k)
{
  repeat
  {
    // Generate values for x[k].
    NextValue (k); // assign a legal next
                      value to x[t].
    if (x [k]=0) then return;
    if (k=n) then written (x [1:n]);
    else Hamiltonian(k+1);
  }
  until( false);
}
```

G1:



G2:



The graph G1 of above fig contains the hamiltonian cycle 1, 2, 8, 7, 6, 5, 4, 3, 2, 1

The graph G2 of above fig contain no hamiltonian cycle. There is no known easy way to determine whether a given graph contains a hamitonian cycle.

Above backtracking algorithm that finds all the hamitonian cycles in a graph. The graph may be directed or undirected.

* Graph Coloring Problem:

Let G be a graph & m be a given positive integer. We want to discover wheather the node of G can be colored in such a way that no two adjacent nodes have the same color yet only m colors are used.

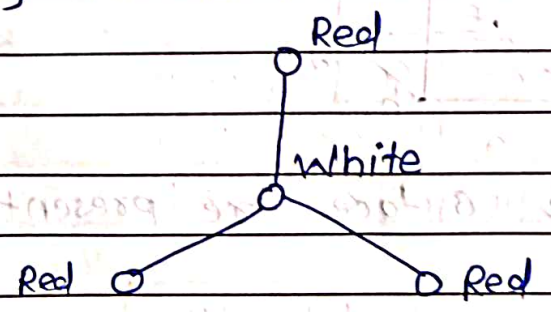
If d is the degree of the given graph, then it can be colored with d+1 m for which the graph G can be colored.

In the graph coloring there are three types, which is following:

1) Node Coloring
2) Edge Coloring
3) face coloring

① Node Coloring:

In node coloring adjacency node having not same color.

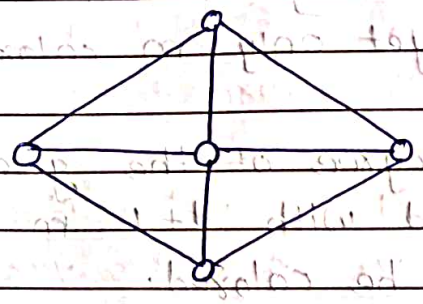


(G_1)

In above graph there are 2 colors aneeded.

② Edge Coloring:

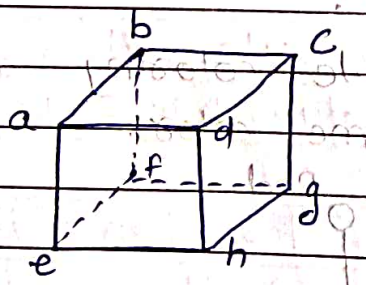
In edge coloring, adjacency ~~node having not same color~~ edge should not having same color.



In above graph for edge coloring 5 color are needed & for node colouring 3 colour are needed.

③ face Coloring:

In face colouring the adjacencency face not having same colour.



Their are 6 face are present & 3 colors are used.

Explain backtracking solution to n-queen problem.

Backtracking:

Backtracking represents one of the most general techniques, many problem which deals with searching for a set of solution or which ask for an optimal solution satisfied some constraints can be solved using the backtracking formulation.

In many application of the backtracking method the desired solution is expressible as an n-tuples $(x_1 - - - - x_n)$ where x_i are choosen from some finite set s_i.

A collection of problem whose solution are best done using backtracking.

N- queens problem:

The n-queens problem is a generalization of the 8-queen problem for example.

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

Now n queens are to be placed on an nxn chessboard so that no two attack; that is, no two queens are on the same row column or diagonal.

Generalizing our earlier discussion, the solution space consist of all n! permutations of the n-tuples (1, 2, 3, 4 - - - - n)

Following are the algorithm for the ~~N-tuples~~ N-queens problem.
All solution to the N-queen problem;

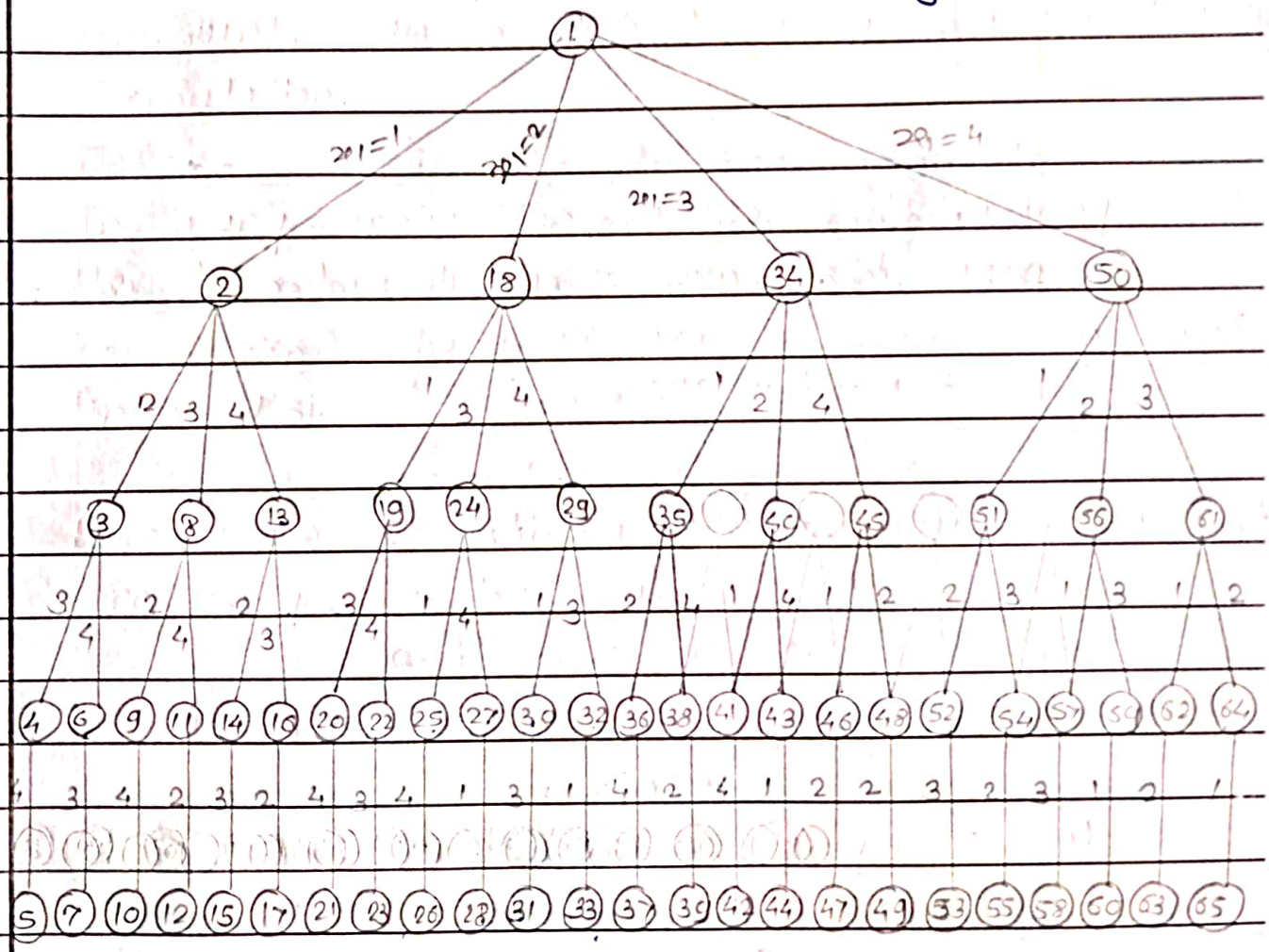
```
Algorithm NQueens(k,n)
{
  for i=1 to n do
  {
    if place(k,i) then
    {
      x[k]=i;
      if (k=n) then write (x[1:n]);
      else NQueens (k+1,n);
    }
  }
}
```

Now shows a possible tree organization for the case n=4. A tree such as this is called a permutation tree.

The edges are labeled by possible values of x_1 edges from level 1 to level 2 node specify the values for x_1.

Thus, the leftmost subtree contain all solution with values of $x_1 = 1$; its leftmost subtree contain all solution with $x_1 = 1$ & $x_2 = 2$ so on.

Edge from level 1 to defined by all paths from the root node to a leaf node. There are $4! = 24$ leaf nodes in the tree of below fig.



Explain Live node, E-node, dead node & bounding function in backtracking.

Backtracking:

Backtracking represents one of the most general tech. many problem which deals with searching for a set of fisible or which ask for an optimal solution.

In many application of the backtracking method, the desired solution is expressible an n-tuples $(x_1, \ldots x_n)$ Where the x_i are choosen from some finite set S_i.

Collection of problems whose solution are best done using backtracking.

Bounding function:

The backtrack algorithm has its virtue the ability to yeild the same answer with for fewer than m trails. It's basic idea is to build up the solution vector one component at the time & to use modified criterion function $P_i(x_1, \ldots x_i)$ Sometimes called bounding function.

live Node:

A node which has been generated & all of whose children have not yet been generated is called as a live node.

E- Node:

The live node whose children are currently being generated is called the E-node. (node being expanded)

Dead Node:

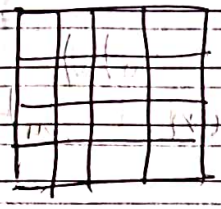
A dead node is a generated node which is not be expanded further or all of whose children have been generated. In both method of generating problem states.

8 Queen Problem

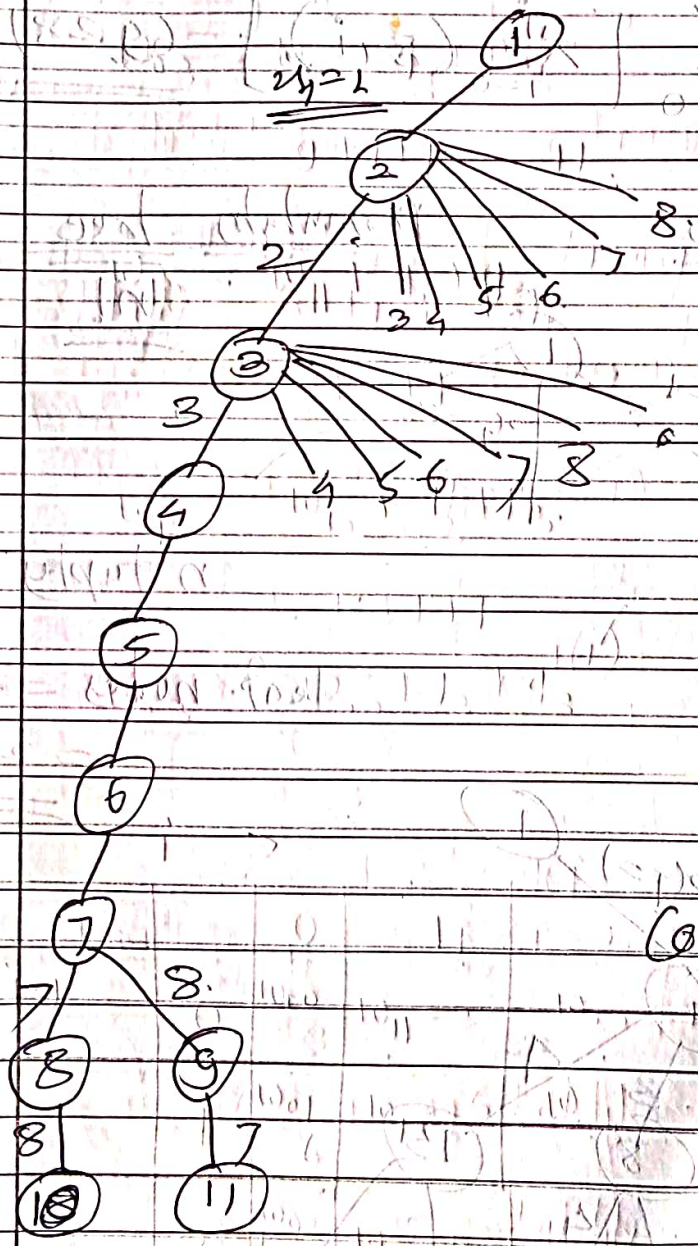
4, 6, 8, 2, 7, 1, 3, 5.

Coloumn, row, diagnal.

Possible Solution



8×8



Computing time

$O(n-1)$

For 8×8 chess board there are $\binom{64}{8}$ possible ways

But by allowing distinct row & columns it requires only 8! or only 40320 in 8 tuples

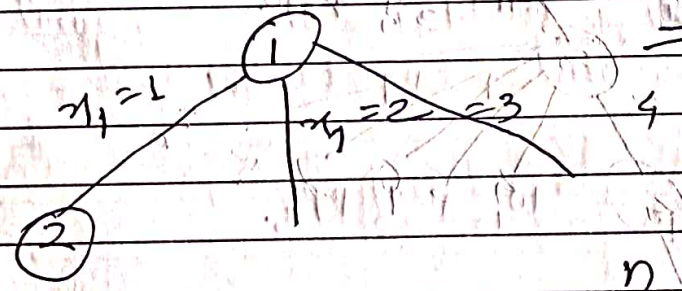
Total no of nodes in 8 queen state space tree is

$$1 + \sum_{j=0}^{7} \left[\prod_{i=0}^{j} (8-i) \right] = 69281$$

4 queen

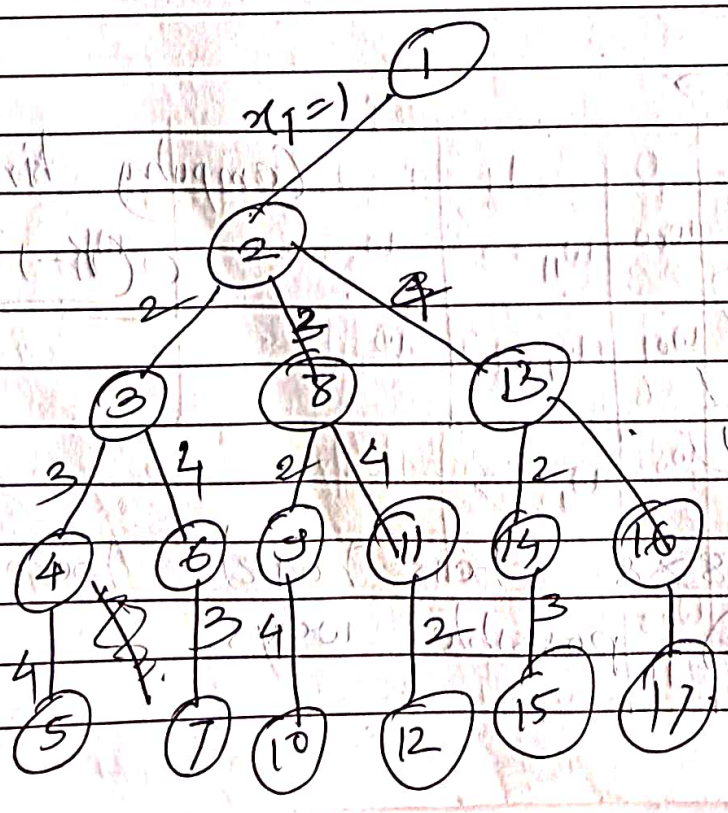
Permutation tree

n!



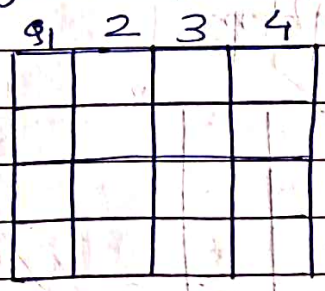
n tuples

leaf nodes = 4! = 24



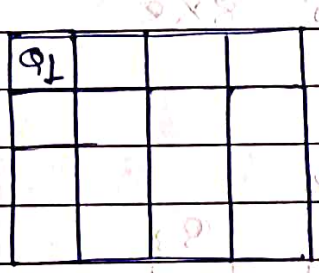
* **n×n Queen Problem**

- Backtracking Approach.
- Eg 8×8, 4×4



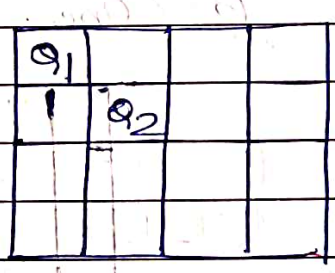
- No Rows, coloums or diagonals having more than one queen
- Possiblily of placing queen is $^{16}C_4$ (It will have repeation)

(A) (I)



✓

(II)

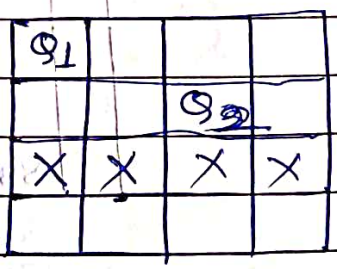


✗

(III)

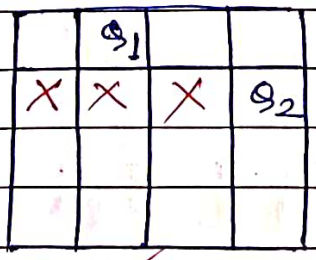
✓

(IV)



✗

(B) (I)



✓

(II)

(2, 4, 1, 3)

The problem can be solve by other way ie mirror of above

		Q_1	
Q_2			
			Q_3
	Q_4		

★ **8 X 8 Queen Problem**

①

Q_1	X						
X	X	Q_2					
	X		X	Q_3			
	X					Q_4	X
	Q_5		X				X
	X		Q_6	X			
	X		X	X	Q_7		
	X		X	X	X	X	

Backtracking

② One Solution for 8X8

		Q_1					
					Q_2		
							Q_3
Q_4							
			Q_5				
						Q_6	
				Q_7			
	Q_8						

One solution (3, 5, 8, 1, 4, 7, 5, 2)

* Knapsack Problem

$\sum_{1 \le i \le n} w_i x_i \le m$ & $\sum_{1 \le i \le n} p_i x_i$ is max

Eg:-

i	1	2	3	4
P	3	4	6	12
W	3	3	4	6

M/C = 10

At initial stage $S^0 = \{(0,0)\}$

<u>step 1</u> :- $S_1^0 = \{(3,3)\}$

<u>step 2</u> :- $S^1 = S^0 \cup S_1^0$

$= \{(0,0), (3,3)\}$

<u>step 3</u> :- $S_2^1 = \{(0,0)+(4,3), (3,3)+(4,3)\}$

$= \{(4,3), (7,6)\}$

<u>step 4</u> :- $S^2 = \{(4,3), (7,6)\} \cup \{\{0,0\}, (3,3)\}$

$= \{(0,0), (3,3), (4,3), (7,6)\}$

<u>step 5</u> :- $S_1^2 = \{(0,0), (3,3), (4,3) (7,6)\} + (6,4)$

$= \{(6,4), (9,7), (10,7), (13,10)\}$

<u>step 6</u> :- $S^3 = \{(0,0), (3,3), (4,3), (7,6), (6,4), (9,7), (10,7) (13,10)\}$

$S^3 + (10,6)$

$= \{(10,6), (13,9) (14,9), (17,12), (16,10), (19,13), (20,13), (23,16)\}$

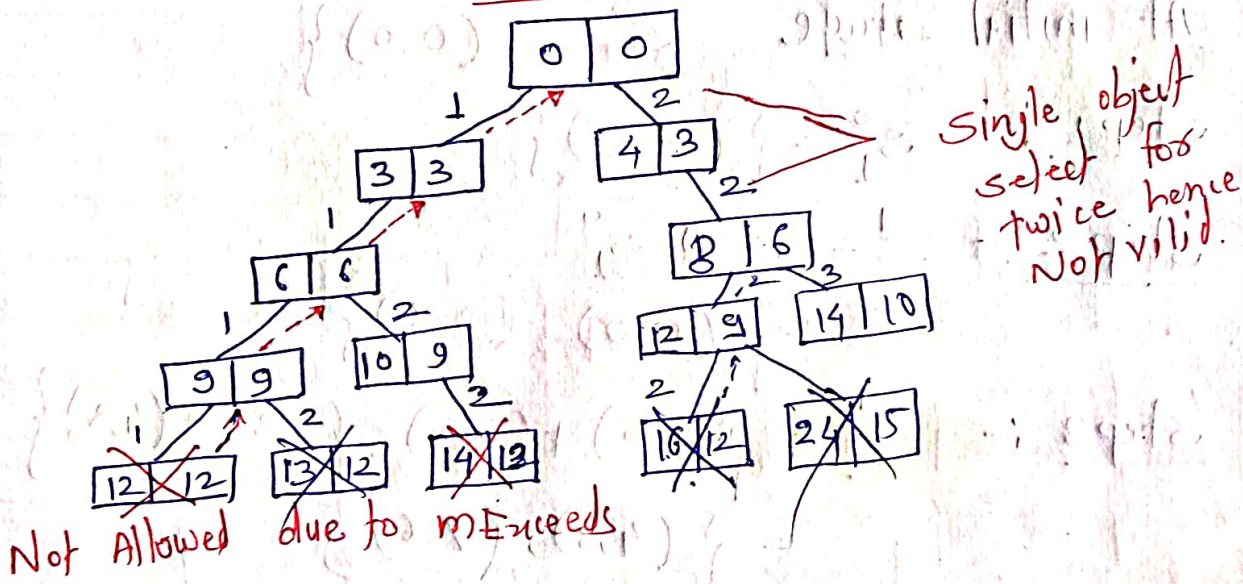
$S^4 = \{(0,0), (3,3), (4,3), (7,6), (6,4), (9,7), (10,7), (13,10), (10,6), (13,9), (14,9), (17,12), \boxed{(16,10)}, (19,13), (20,13), (23,16)\}$

Finally $\boxed{S^5 = S^4 + (12,6)}$

(0,0), (3,3), (4,3), (7,6), (6,4), (9,7), (10,7), (13,10)
(10,6), (13,9), (14,9), (17,12), (16,10), (19,12), (20,12)
(23,12), (12,6), (15,9), (16,9), (19,12) $\boxed{(18, 10)}$
(21,13), (22,12) (25,14) (22,12) (25,15) (26,15)
(29,18) (28,16) (31,19) (32,19) (35,22)

Max Profit for $M \leq 10$ ie 18 hence the answer is $(0, 0, 1, 1) = 0+0+6+12 = 18$

State Space Tree

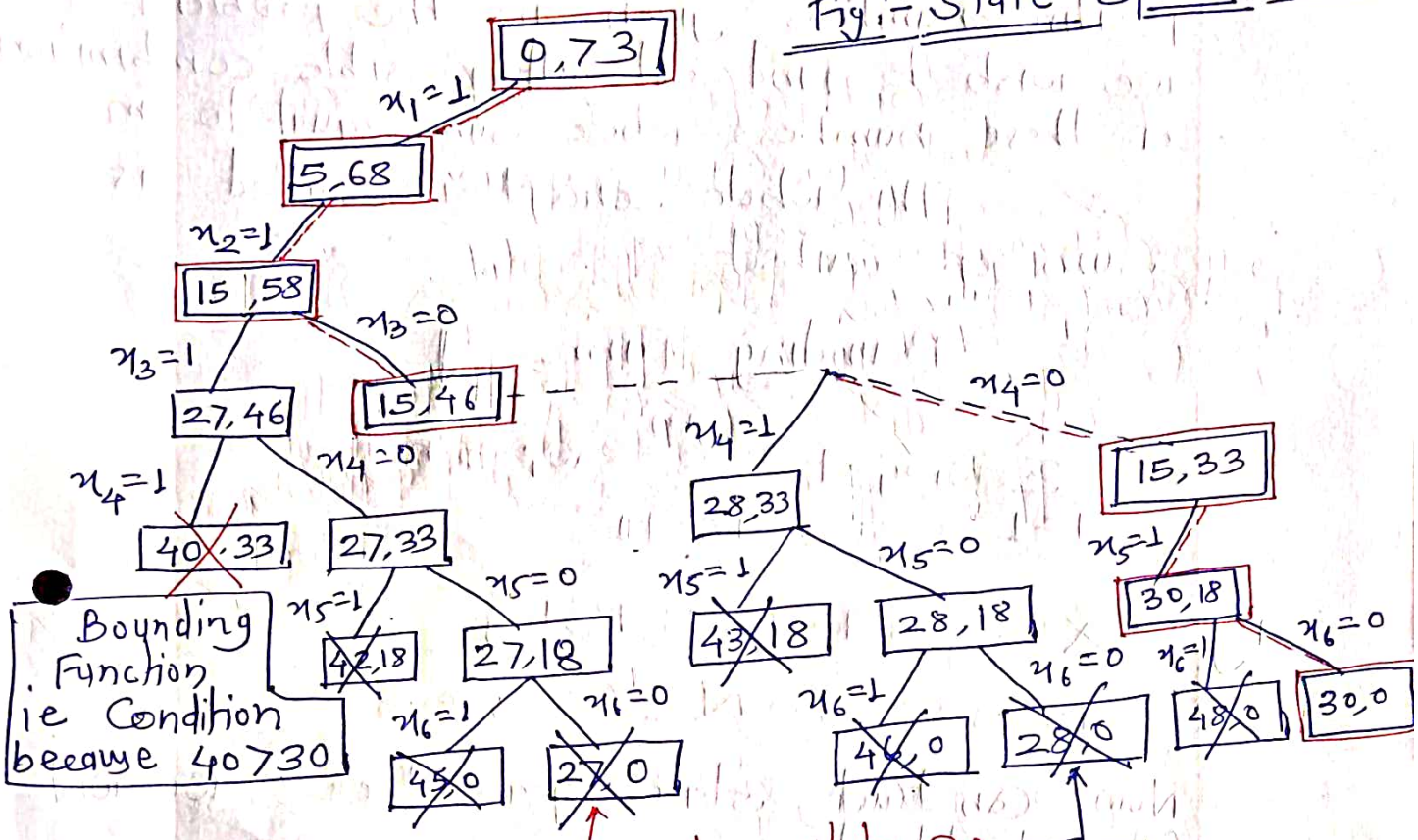


We have given $w = \{5, 10, 12, 13, 15, 18\}$

$\Sigma w_i = 5 + 10 + 12 + 13 + 15 + 18$

$= 73$

Fig:- State Space Tree



[As it is not equal to 30 hence it should be discarded]

- Now we got exactly solution equals to 30 by taking

$(x_1 = 1,\ x_2 = 1,\ x_3 = 0,\ x_4 = 0,\ x_5 = 1\ \&\ x_6 = 0)$

- Hence one of the optimal solution is

$(1, 1, 0, 0, 1, 0)$

- If we go for traditional method it will require $(2)^n = 2^n$ comparisions hence complexity will be $O(2^n) = 2^6$
- By using backtracking it is reduced

Some other sol'ns are $(1, 0, 1, 1, 0, 0)$
$(0, 0, 1, 0, 0, 1)$

Eg:- $M = 35$, $w = \{5, 7, 10, 12, 15, 18, 20\}$

* Sum of Subsets *

Suppose there are n distinct positive numbers given in the problem & we wish to find out all possible combinition of these numbers whose sum equal to m.

This whole concept is called as sum of subset.

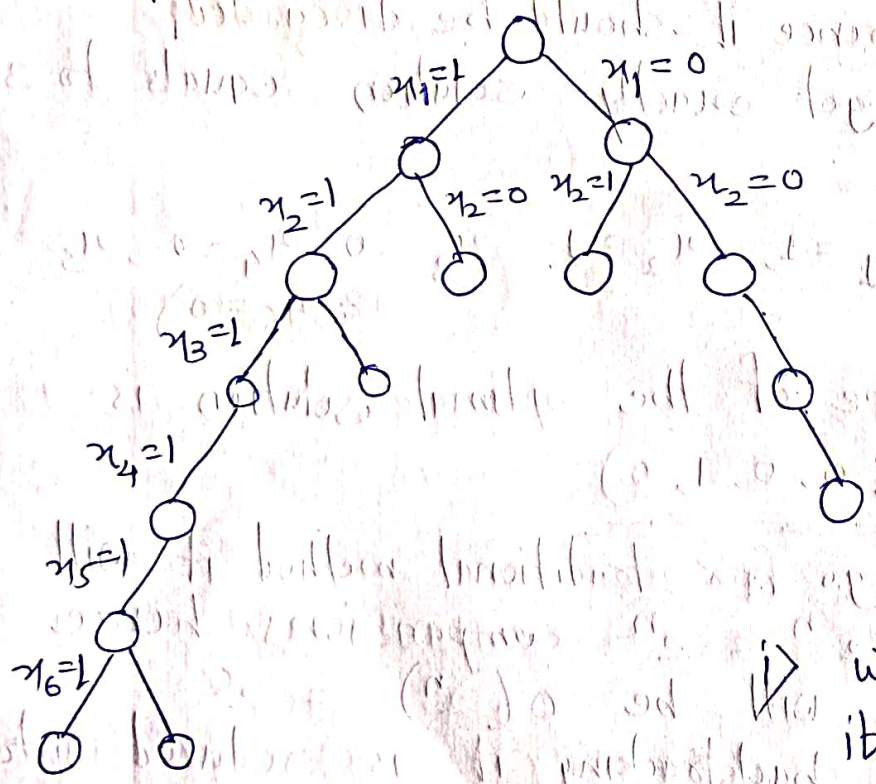
Bounding Funn ↓

$$\sum_{i=1}^{K} w_i x_i + \sum_{i=k+1}^{m} w_i \geq m$$

Eg:- $X = \{5, 10, 12, 13, 15, 18\}$

$M = 30$, $N = 6$

Now construct state space tree; we will follow backtracking



1) when $x_i = 1$ it is included

2) when $x_i = 0$ it is not included.