UNIT V

BACKTRACKING – ALGORITHMS

Backtracking & I Backtracking:

N Queens:

Sum of Subsets:

Hamiltonian Cycles:

Graph Coloring:

```
1   Algorithm Place(k, i)
2   // Returns true if a queen can be placed in kth row and
3   // ith column. Otherwise it returns false. x[ ] is a
4   // global array whose first (k − 1) values have been set.
5   // Abs(r) returns the absolute value of r.
6   {
7       for j := 1 to k − 1 do
8           if ((x[j] = i) // Two in the same column
9               or (Abs(x[j] − i) = Abs(j − k)))
10                  // or in the same diagonal
11              then return false;
12      return true;
13  }
```

Algorithm 7.4 Can a new queen be placed?

```
1   Algorithm NQueens(k, n)
2   // Using backtracking, this procedure prints all
3   // possible placements of n queens on an n × n
4   // chessboard so that they are nonattacking.
5   {
6       for i := 1 to n do
7       {
8           if Place(k, i) then
9           {
10              x[k] := i;
11              if (k = n) then write (x[1 : n]);
12              else NQueens(k + 1, n);
13          }
14      }
15  }
```

lgorithm 7.5 All solutions to the n-queens problem

```
1   Algorithm SumOfSub(s, k, r)
2.  // Find all subsets of w[1 : n] that sum to m. The values of x[j],
3   // 1 ≤ j < k, have already been determined. s = Σ_{j=1}^{k-1} w[j] * x[j]
4   // and r = Σ_{j=k}^{n} w[j]. The w[j]'s are in nondecreasing order.
5   // It is assumed that w[1] ≤ m and Σ_{i=1}^{n} w[i] ≥ m.
6   {
7       // Generate left child. Note: s + w[k] ≤ m since B_{k-1} is true.
8       x[k] := 1;
9       if (s + w[k] = m) then write (x[1 : k]); // Subset found
10          // There is no recursive call here as w[j] > 0, 1 ≤ j ≤ n.
11      else if (s + w[k] + w[k + 1] ≤ m)
12          then SumOfSub(s + w[k], k + 1, r − w[k]);
13      // Generate right child and evaluate B_k.
14      if ((s + r − w[k] ≥ m) and (s + w[k + 1] ≤ m)) then
15      {
16          x[k] := 0;
17          SumOfSub(s, k + 1, r − w[k]);
18      }
19  }
```

```
1   Algorithm Hamiltonian(k)
2   // This algorithm uses the recursive formulation of
3   // backtracking to find all the Hamiltonian cycles
4   // of a graph. The graph is stored as an adjacency
5   // matrix G[1 : n, 1 : n]. All cycles begin at node 1.
6   {
7       repeat
8       { // Generate values for x[k].
9           NextValue(k); // Assign a legal next value to x[k]
10          if (x[k] = 0) then return;
11          if (k = n) then write (x[1 : n]);
12          else Hamiltonian(k + 1);
13      } until (false);
14  }
```

```
1   Algorithm NextValue(k)
2   // x[1 : k − 1] is a path of k − 1 distinct vertices. If x[k] = 0, then
3   // no vertex has as yet been assigned to x[k]. After execution,
4   // x[k] is assigned to the next highest numbered vertex which
5   // does not already appear in x[1 : k − 1] and is connected by
6   // an edge to x[k − 1]. Otherwise x[k] = 0. If k = n, then
7   // in addition x[k] is connected to x[1].
8   {
9       repeat
10      {
11          x[k] := (x[k] + 1) mod (n + 1); // Next vertex.
12          if (x[k] = 0) then return;
13          if (G[x[k − 1], x[k]] ≠ 0) then
14          { // Is there an edge?
15              for j := 1 to k − 1 do if (x[j] = x[k]) then break;
16                                  // Check for distinctness.
17              if (j = k) then // If true, then the vertex is distinct.
18                  if ((k < n) or ((k = n) and G[x[n], x[1]] ≠ 0))
19                      then return;
20          }
21      } until (false);
22  }
```

```
Algorithm Backtrack(k)
// This schema describes the backtracking process using
// recursion. On entering, the first k − 1 values
// x[1], x[2], . . . , x[k − 1] of the solution vector
// x[1 : n] have been assigned. x[ ] and n are global.
{
    for (each x[k] ∈ T(x[1], . . . , x[k − 1]) do
    {
        if (B_k(x[1], x[2], . . . , x[k]) ≠ 0) then
        {
            if (x[1], x[2], . . . , x[k] is a path to an answer node)
                then write (x[1 : k]);
            if (k < n) then Backtrack(k + 1);
        }
    }
}
```

```
1   Algorithm IBacktrack(n)
2   // This schema describes the backtracking process.
3   // All solutions are generated in x[1 : n] and printed
4   // as soon as they are determined.
5   {
6       k := 1;
7       while (k ≠ 0) do
8       {
9           if (there remains an untried x[k] ∈ T(x[1], x[2], ...,
10              x[k − 1]) and B_k(x[1], ..., x[k]) is true) then
11          {
12              if (x[1], ..., x[k] is a path to an answer node)
13                  then write (x[1 : k]);
14              k := k + 1; // Consider the next set.
15          }
16          else k := k − 1; // Backtrack to the previous set.
17      }
18  }
```

```
1   Algorithm mColoring(k)
2   // This algorithm was formed using the recursive backtracking
3   // schema. The graph is represented by its boolean adjacency
4   // matrix G[1 : n, 1 : n]. All assignments of 1, 2, ..., m to the
5   // vertices of the graph such that adjacent vertices are
6   // assigned distinct integers are printed. k is the index
7   // of the next vertex to color.
8   {
9       repeat
10      {// Generate all legal assignments for x[k].
11          NextValue(k); // Assign to x[k] a legal color.
12          if (x[k] = 0) then return; // No new color possible
13          if (k = n) then     // At most m colors have been
14                              // used to color the n vertices.
15              write (x[1 : n]);
16          else mColoring(k + 1);
17      } until (false);
18  }
```

20/2/19

UNIT-5

Back Tracking:-

Back Tracking is the one of the optimization technique

Back Tracking is one of the systematic searching technique

By using Back Tracking strategy we can obtain possible no. of feasible solutions.

The optimal solutions may be a maximum (or) minimum.

Back Tracking provides a dynamic applications like

1) N-Queen's problems.
   - → 4 X 4
   - → 8 X 8
2) Sum of subsets problem.
3) Graph coloring
4) Hamiltonious cycle

Sum of Subsets problem:-

Suppose we are given 'n' distinct positive numbers (weights) and we design to find all combinations of these numbers whose sums are 'm' is called sum of subsets problem.

Procedure:-

Step(1) If $S+w(k)=M$ then return subset

Step(2): If $S+w(k) \neq M$ then generate left child and Right child.

Step(3):- (Left child) If $S+w(k)+w(k+1) \leq M$ then write the solution $x(k)=1$ and the sum of sub sets are $S+w(k), k+1, r-w(k)$

Step(4):- (Right Child) If $S+r-w(k) \geq M$ and $S+w(k+1) \leq M$ then write the solution $x(k)=0$ and the sum of subsets are $S, k+1, r-w(k)$

Find possible no. of sum of subsets using sum of subsets algorithm $m=9$, $n=3$, $(w_1, w_2, w_3) = (2, 4, 5)$

Initially $s=0$

$k = w_1$

$r = 2+4+5 = 11$

$s + w(k) = m$ $(0, 2, 11)$

$0 + 2 \neq 9$

Now generate Left child and Right child

Left child

$s + w(k) + w(k+1) \leq m$

$0 + 2 + 4 \leq 9$

$6 \leq 9$ (T)

Sum of sub sets are

$s + w(k)$, $k+1$, $r - w(k)$

$0+2$, $1+1$, $11-2$

$(2, 2, 9)$

Right child

$s + r - w(k) \geq m$ and $s + w(k+1) \leq m$

$0 + 11 - 2 \geq 9$ and $0 + 4 \leq 9$

$9 \geq 9$ and $4 \leq 9$ (T)

Sum of subsets are

s, $k+1$, $r - w(k)$

0, 2, $11-2$

$(0, 2, 9)$

Consider

$s = 2$, $k = 2$, $r = 9$

$s + w(k) = m$

$2 + 4 = 9$

$6 \neq 9$

Left child

$s + w(k) + w(k+1) \leq m$

$2 + 4 + 5 \leq 9$

$11 \leq 9$ (F)

Right child

$s + r - w(k) \geq m$ and $s + w(k+1) \leq m$

$2 + 9 - 4 \geq 9$ and $2 + 5 \leq 9$

$7 \geq 9$ and $7 \leq 9$ (F)

S=0 , K=2, r=9

S+W(K)=m

0+4=9

4≠9

→ Left child

S+W(K)+W(K+1) ≤ m

0+4+5≤9

9≤9

(4, 3, 5)

consider

S=4 , K=3, ~~r=9~~ r=5

S+W(K)=m

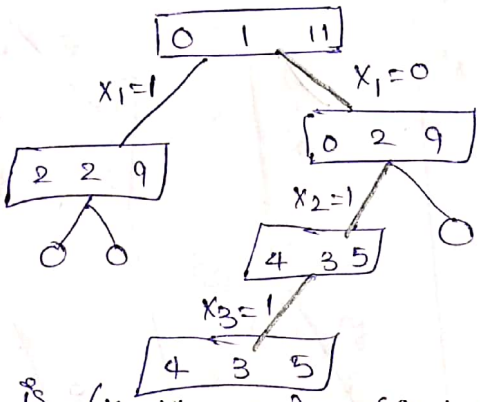
4+5=9

9=9 (T)

Right child

0+9-4≥9 and 0+5≤9

5≥9 and 5≤9 (F)



the feasible solution is $(x_1, x_2, x_3) = (0,1,1)$

$$\sum_{i=1}^{3} w_i x_i \le m$$

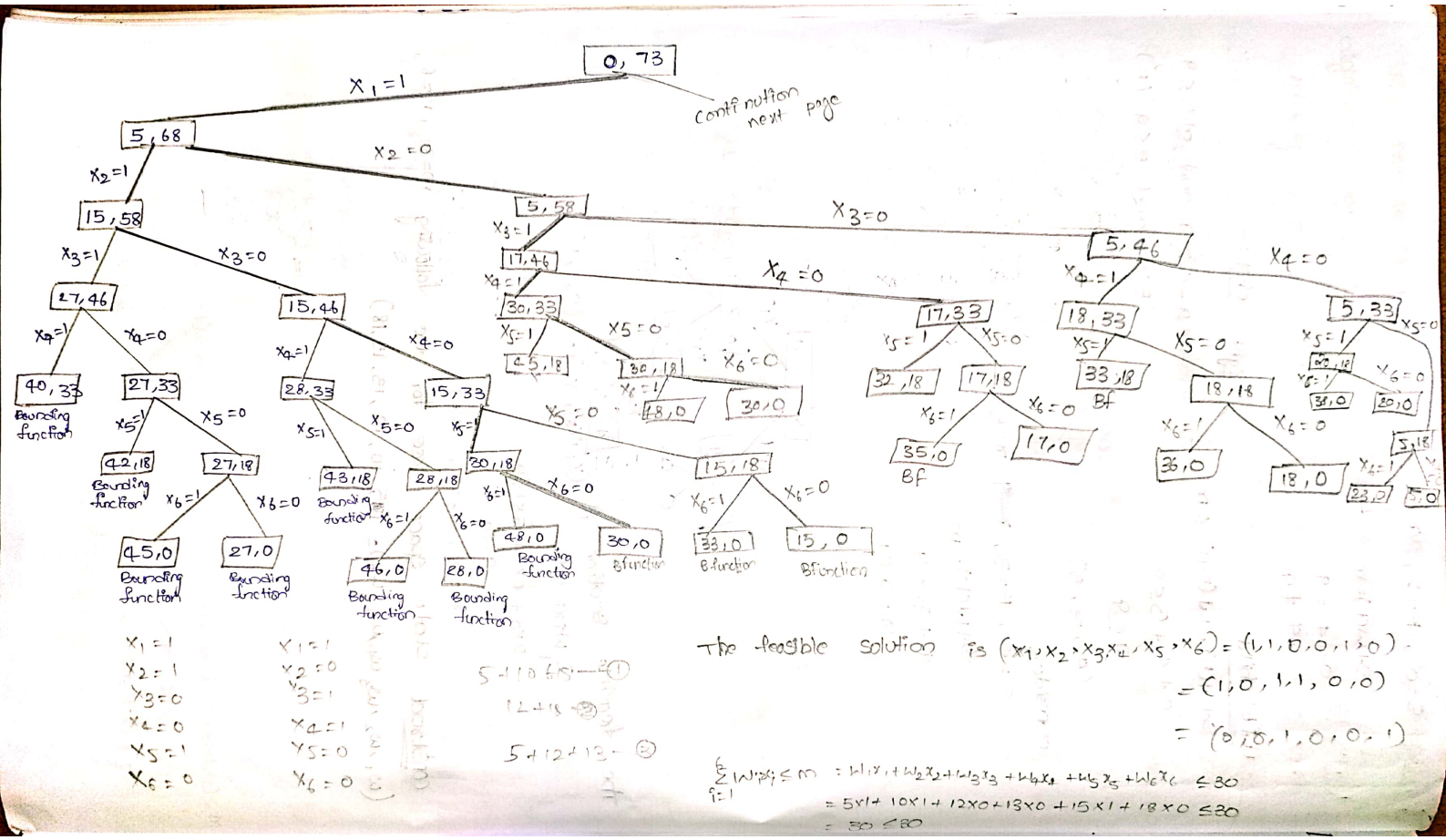
2x0 + 4x1 + 5x1 ≤ 9

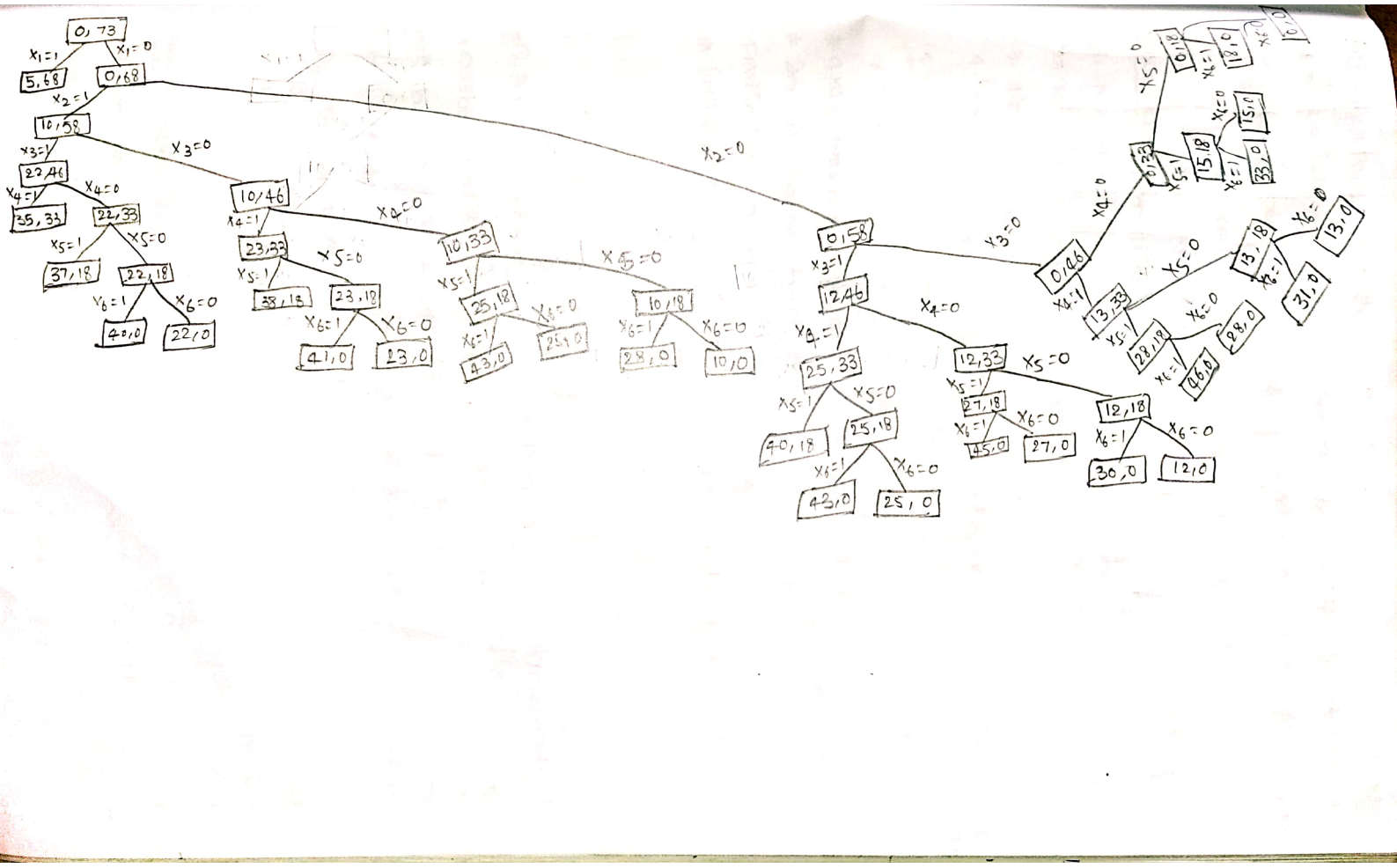
9 ≤ 9

$x_1=0$, $x_2=1$, $x_3=1$

Construct state space tree for the following m=30 , n=6,

$(w_1, w_2, w_3, w_4, w_5, w_6) = (5, 10, 12, 13, 15, 18)$


0, 73
$x_1=1$
Continuation next page
5, 68
$x_2=0$
$x_2=1$
15, 58
5, 58
$x_3=0$
$x_3=1$
27, 46
15, 46
17, 46
5, 46
$x_4=0$
$x_4=1$
40, 33
Bounding function
27, 33
28, 33
15, 33
30, 33
17, 33
18, 33
5, 33
$x_5=1$
$x_5=0$
42, 18
Bounding function
27, 18
43, 18
28, 18
30, 18
45, 18
30, 18
32, 18
17, 18
33, 18
Bf
18, 18
15, 18
$x_6=1$
$x_6=0$
45, 0
Bounding function
27, 0
46, 0
28, 0
48, 0
30, 0
48, 0
30, 0
35, 0
Bf
17, 0
36, 0
18, 0
33, 0
B function
15, 0
B function
$x_1=1$
$x_2=1$
$x_3=0$
$x_4=0$
$x_5=1$
$x_6=0$
$x_1=1$
$x_2=0$
$x_3=1$
$x_4=1$
$x_5=0$
$x_6=0$
$5+10 \leq 20$ — ①
$12+18$ — ②
$5+12+18$ — ③
The feasible solution is $(x_1, x_2, x_3, x_4, x_5, x_6) = (1,1,0,0,1,0)$
$= (1,0,1,1,0,0)$
$= (1,0,1,0,0,1)$
$\sum_{i=1}^{6} w_i x_i \leq m : w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6 \leq 30$
$= 5x1 + 10x1 + 12x0 + 13x0 + 15x1 + 18x0 \leq 30$
$= 30 \leq 30$


0,73
X1=1
X1=0
5,68
0,68
X2=1
X2=0
10,58
X3=1
X3=0
22,46
10,46
X4=1
X4=0
X4=1
X4=0
35,33
22,33
23,33
10,33
X5=1
X5=0
X5=1
X5=0
X5=1
X5=0
37,18
22,18
28,18
23,18
25,18
10,18
X6=1
X6=0
X6=1
X6=0
X6=1
X6=0
X6=1
X6=0
40,0
22,0
41,0
23,0
43,0
25,0
28,0
10,0
0,58
X3=1
X3=0
12,46
0,46
X4=1
X4=0
X4=1
X4=0
25,33
12,33
13,33
0,33
X5=1
X5=0
X5=1
X5=0
X5=1
X5=0
X5=1
X5=0
40,18
25,18
27,18
12,18
28,18
15,18
X6=1
X6=0
X6=1
X6=0
X6=1
X6=0
X6=1
X6=0
43,0
25,0
45,0
27,0
30,0
12,0
46,0
28,0
33,0
15,0
31,0
13,0

N - QUEEN'S PROBLEM:-

We can place N no. of Queen's (Q) in N×N chessboard

N-Queen problem is majorly divided into two types

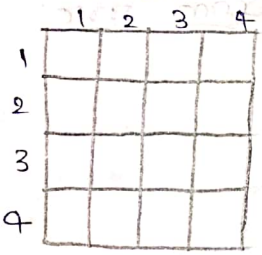
1. 4 Queen's problem
2. 8 Queen's problem

procedure (conditions):-

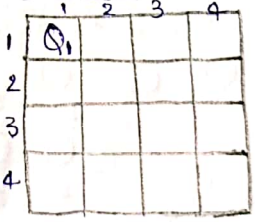
1. No two of them attack each other
2. No two Queen's placed in same row
3. No two Queen's placed in same column.
4. No two Queen's placed in diagonal of a chessboard.

4×4 Queen's problem:-

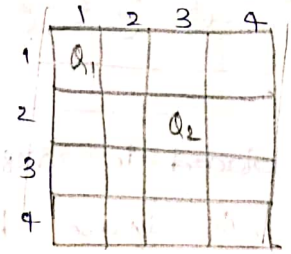
* 4 Number of Queen's can be placed in 4×4 chessboard.
* 4×4 chessboard contains 16 square boxes.
* In 16 square boxes we can place only 4 queen's at the particular positions.
* Initially 4×4 chessboard is



Step① Q_1 is placed in 1×1 position in the chessboard. The possible positions of Q_1 is 1×2, 1×3, 1×4



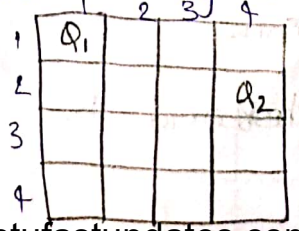
Step② Now Q_2 is placed in 2×3 position in the chessboard. The possible positions of Q_2 is 2×4 only.



not

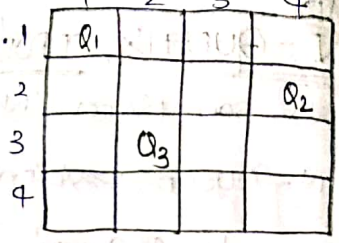
Step:③ Now Q_3 is * placed in any positions of 3×1, 3×2, 3×3, 3×4.

Hence we can apply back tracking on Q_2. Now Q_2 is placed in 2×4.

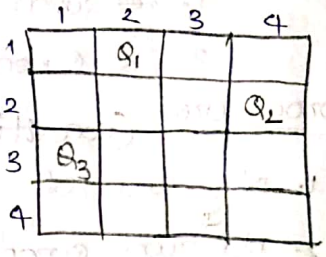


www.Jntufastupdates.com

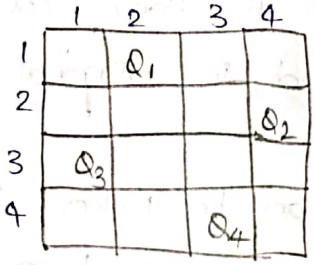
Step④ Now Q_3 is placed in 3x2 position. There is no possible positions.



Step⑤ Now Q_4 is not placed in any positions of 4x1, 4x2, 4x3, 4x4. Hence, apply Backtracking on Q_3, Q_2 & Q_1. Now Q_1 is placed in 1x2, Q_2 is placed in 2x4, Q_3 is placed in 3x1.



Step:6 Now Q_4 is placed in 4x3 position.

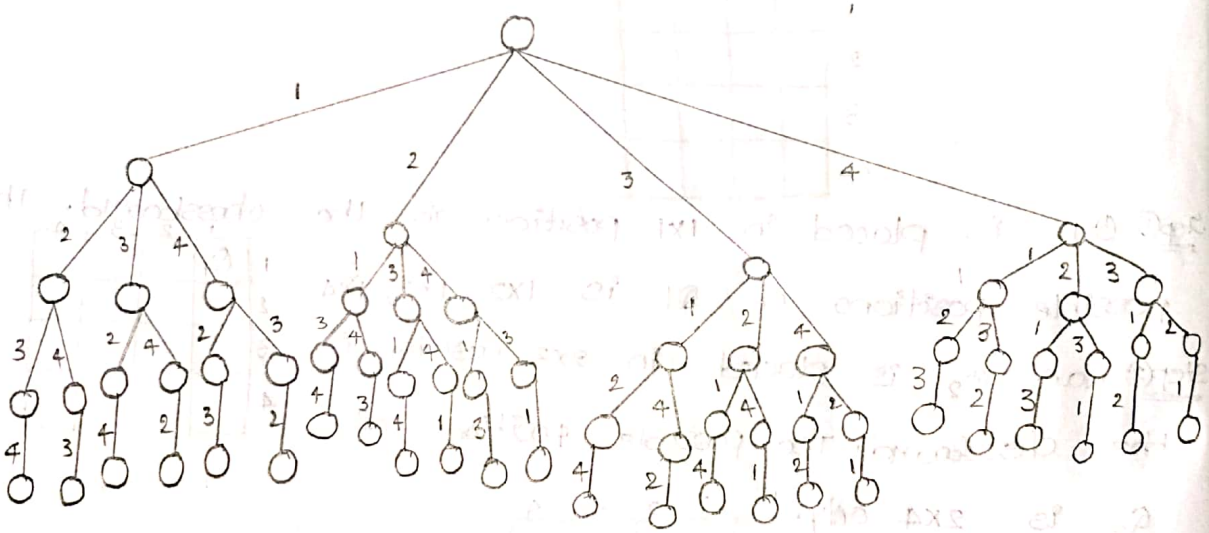


* Hence the possible solution is 1x2, 2x4, 3x1, 4x3. positions are

~~Hence~~

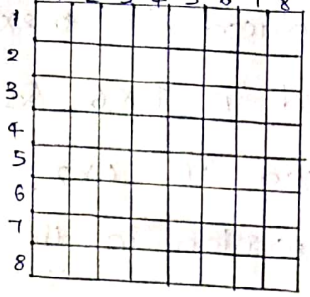
* Hence the feasible solution is (2, 4, 1, 3)

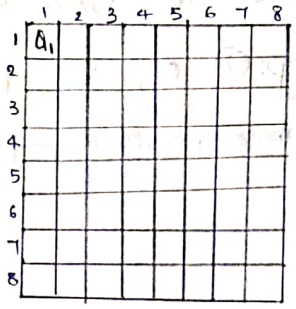
8x8 Queen's problem: State space Tree:-



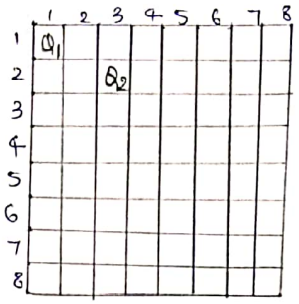
8x8 Queen's problem:-

* 8 No. of queen's can be placed in 8x8 chessboard
* 8x8 chessboard contain 64 square boxes
* In 8 64 square boxes we can place only 8 queen's at the particular positions 4, 6, 8, 2, 7, 1, 3, 5
* Initially 8x8 chessboard is



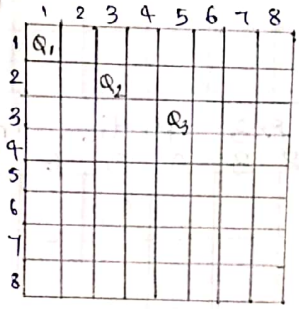


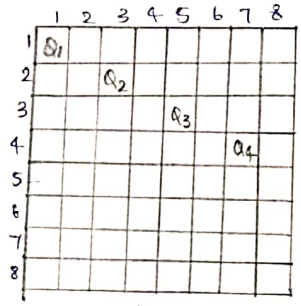
Step① Q_1 is placed in 1X1 position in the chessboard. The possible positions of Q_1 is 1X2, 1X3, 1X4, 1X5, 1X6, 1X7, 1X8.



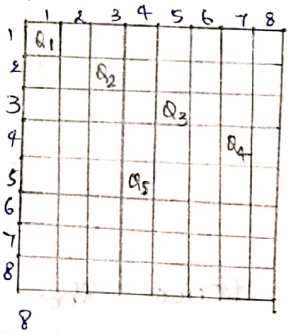
Step② Q_2 is placed in 2X3 position in the chessboard. The possible positions of Q_2 is 2X4, 2X5, 2X6, 2X7, 2X8

Step③ Q_3 is placed in 3X5 position in the chessboard. The possible positions of Q_3 is 3X6, 3X7, 3X8.



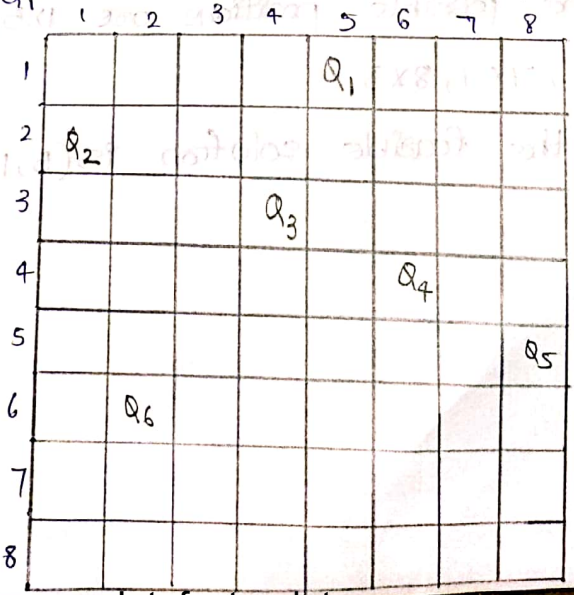


Step:④ Q_4 is placed in 4X7 position in the chessboard the possible positions of Q_4 is 4X8. only.



Step⑤ Q_5 is placed in 5X4 position in the chessboard

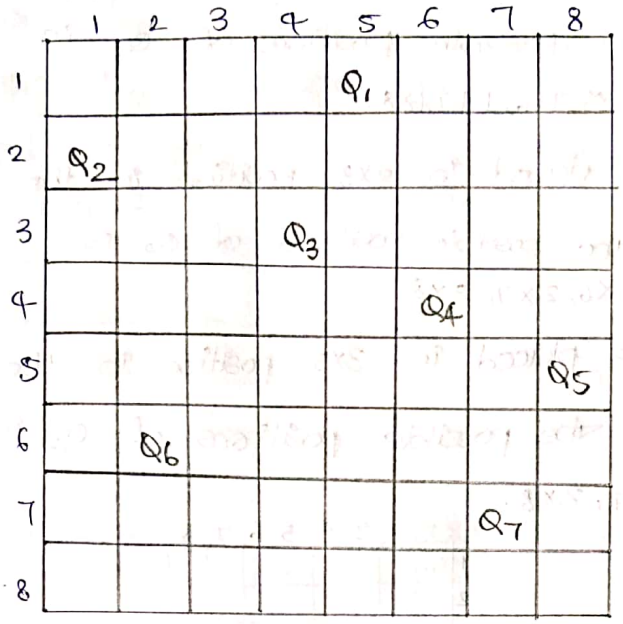
Step:6 Q_6 is not placed in any positions of 6X1, 6X2, 6X3, 6X4, 6X5, 6X6, 6X7, 6X8. Hence, apply Backtracking on Q_5, Q_4, Q_3, Q_2, Q_1



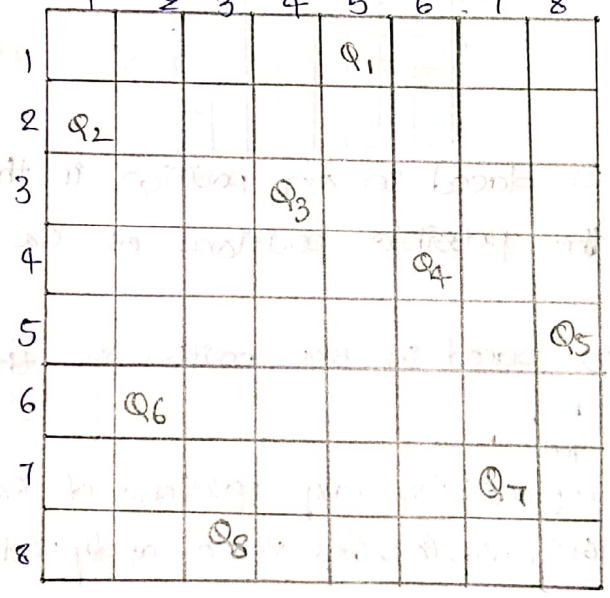
Now Q_1 is placed in 1x5, Q_2 is placed in 2x1, Q_3 is placed in 3x4 and Q_4 is placed in 4x6, Q_5 is placed in 5x8 and Q_6 is placed in 6x2.

Step:7 Q_7 is placed in 7x7 position in the chessboard.

~~there is no possible position~~



Step:8 Q_8 is placed in 8x3 position in the chessboard.



Hence the possible positions are 1x5, 2x1, 3x4, 4x6, 5x8, 6x2, 7x7, 8x3.

Hence the feasible solution is (5,1,4,6,8,2,7,3)

GRAPH COLORING:-

* Let G be a undirected and unweighted graph and m be a positive integer.

* It is to find whether the nodes of G can be color in such a way that no two adjacents nodes have the same color. Yet only m colors are used.

* Graph coloring problem having two general parts
  1. 'm' coloring decision problem
  2. 'm' colorability optimization.

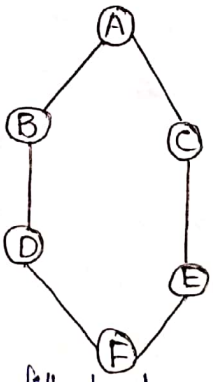
* here 'm' is called chromatic number (NUMBER OF COLORS). where m represents chromatic number.

* If 'd' is the degree of the given graph G. Then it can be colored with 'd+1.'

* Graph coloring problem is also known as m coloring problem If m=3 hence it is called 3 colorfull graph.

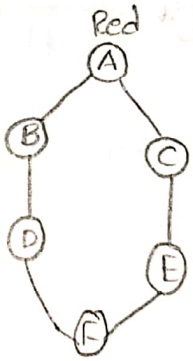
* Time complexity is $O(n \cdot m^n)$

Consider the Graph

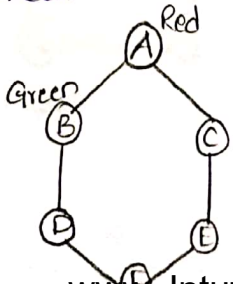


m=3 (Red, green, Blue)

Step (1) Select node A is filled by Red color the adjacent of node A is B & C. So we can not assign Red color at B & C. Hence we can fill node B with Green.



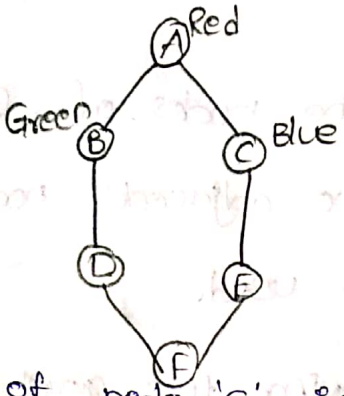
Step:2



www.Jntufastupdates.com

The adjacent of node B is D & A. So we can not assign Green color at A/D. The node A is already fill with Red. Hence we can fill C with Blue color.

Step: 3



The adjacent of node 'C' is A & E. So we can not assign Blue color at A/E. The node A is already fill with Red. Hence we can fill E with Green color.

Step: 4

The adjacent of node 'D' is B & F so we can not assign Blue at B, F. The node B is already fill with Green. Hence

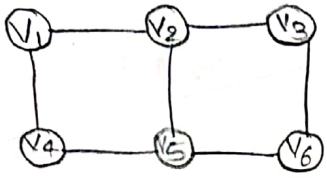
Step: 5



Step: 6

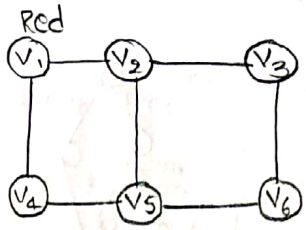


Using Back tracking algorithm for the m coloring problem. find all possible colors of the graph using 3 colors (Red, Green, white) so the actions step by step.



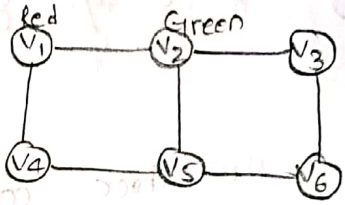
① select node V_1 is filled by Red color. The adjacent of node V_1 is V_2 & V_4. So we can not assign Red color at V_2 & V_4.

Hence we can fill node V_2 with Green.



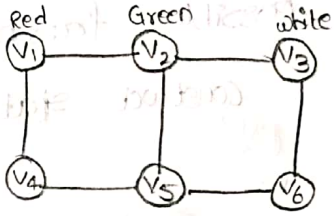
② The adjacent of node V_2 is V_1, V_3, V_5. So we can not assign Green at V_1, V_3, V_5. The node V_1 is already fill with Red.

Hence we can fill V_3 with white color.



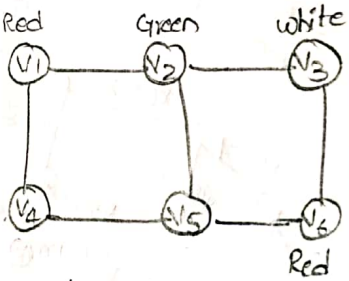
③ The adjacent of node V_3 is V_2 & V_6. So we can not assign white color at V_2 & V_6. The node V_2 is already fill with Green.

Hence we can fill V_6 with ~~Green~~. Red



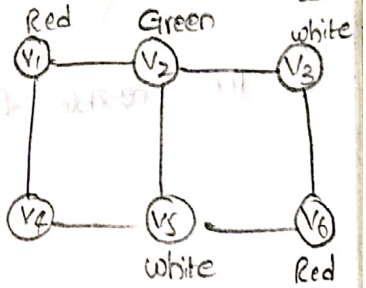
④ The adjacent of node V_6 is V_3 & V_5. So we can not assign ~~Green~~ Red color at V_3 & V_5. The node V_3 is already fill with ~~Red~~ white.

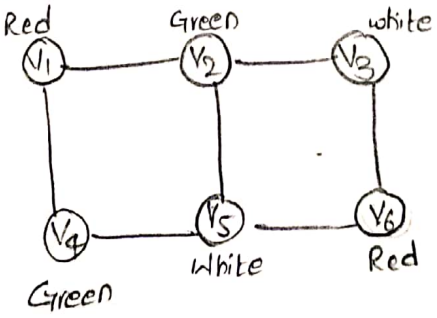
Hence we can fill V_5 with ~~Red~~. white.



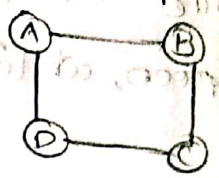
⑤ The adjacent of node V_5 is V_2, V_4 & V_6. So we can not assign ~~Green~~ white Color at V_2, V_4, V_6. The node V_2 is already fill with Green & also V_6 is already fill with Red.

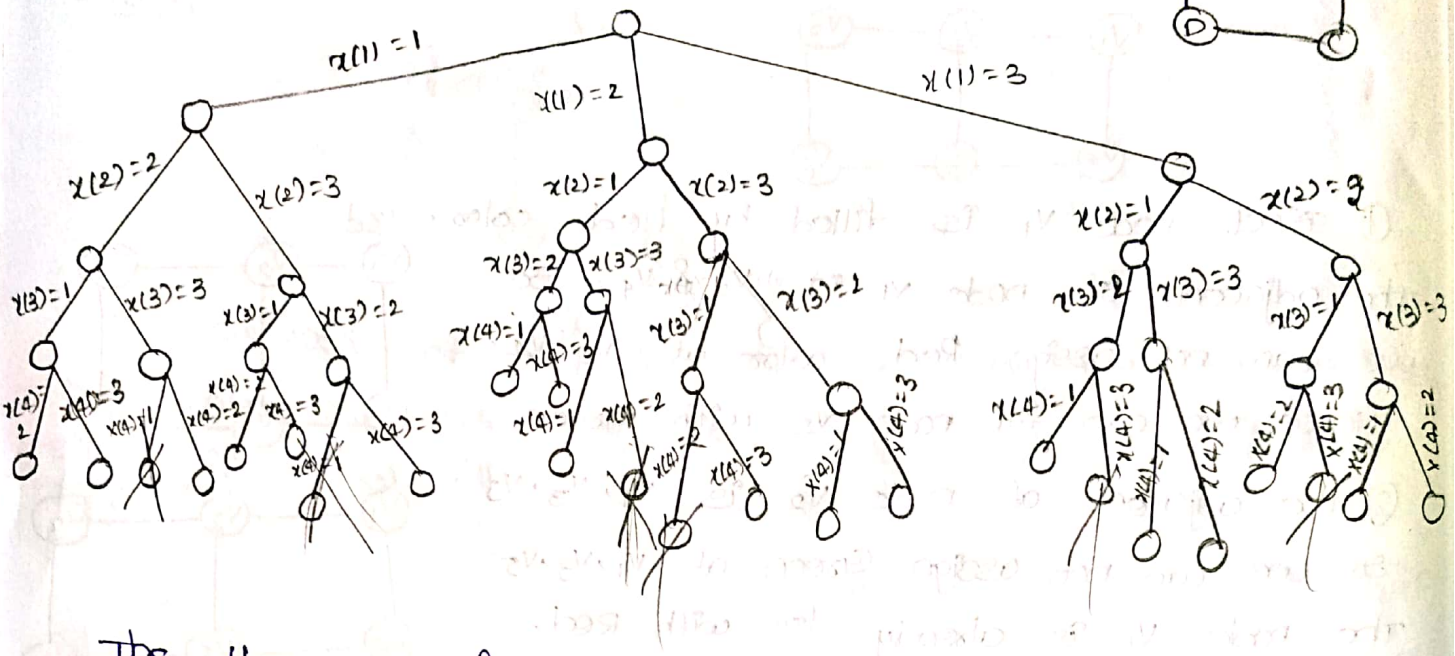
Hence we can fill V_4 with Green.





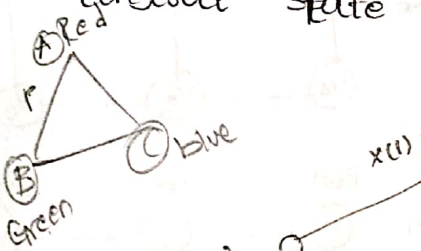
→ construct state space tree for the following graph. n=4 & m=3 (red, green, blue) Red is represented by 1. Green is represented by 2 & Blue is represented by 3.





The three colorfull graph (m = 3 & n = 4) produce 18 possible feasible solutions.

Construct state space tree for m=3 & n=3 (red, green, blue)



The possible feasible solutions are six.

Hamiltonian Cycles:-

* The great Mathematician William Hamiltonian, introduced the dynamic concept is called Hamiltonian cycle (or) Hamiltonian circuit

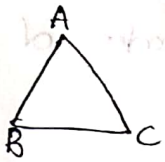
* formally a Hamiltonian cycle of G contain (V, E)

where E represents no. of edges.

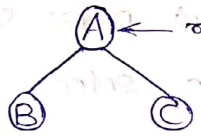
V represents no. of vertices

* In Hamiltonian cycles, except root node (root vertex) remaining vertex visit only once.

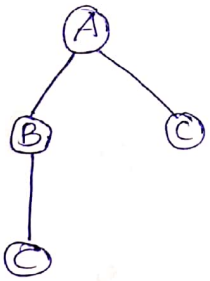
Construct state space tree for the following undirected graph.



Step① select root node as A. Now identify adjacent nodes of A.

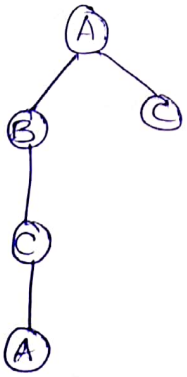


Step:2 Now consider node B and identify adjacent node of B. is A & C.



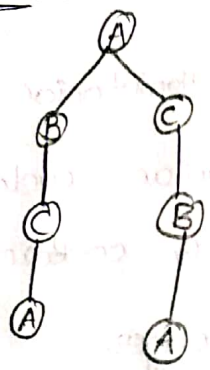
Node A is already visited hence select node C.

Step:3 Now consider node C and identify adjacent nodes of C is A, B. Now B is already visited. Hence select node A.



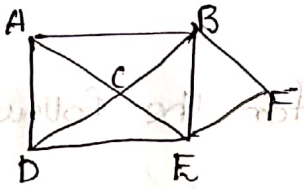
Hence, the hamiltonian cycle is A-B-C-A.

state space tree:-

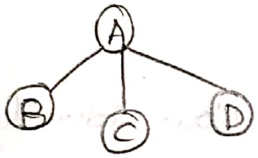


Hamiltonian cycles are A-B-C-A, A-C-B-A.

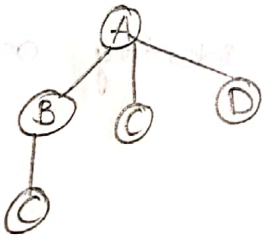
construct state space tree for the following graph using Back tracking

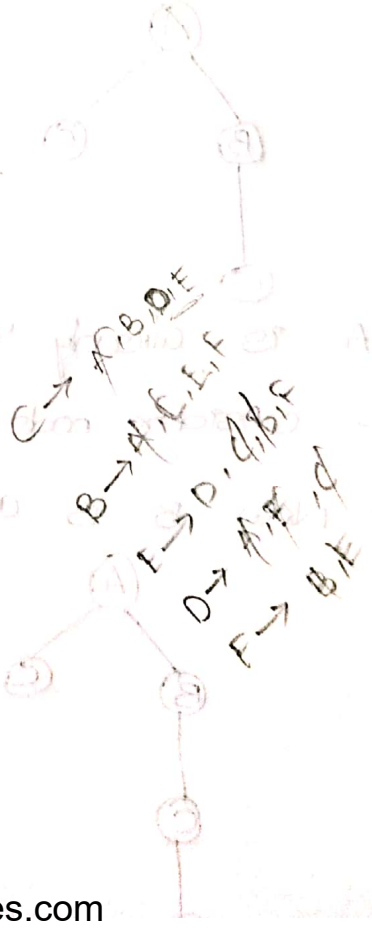


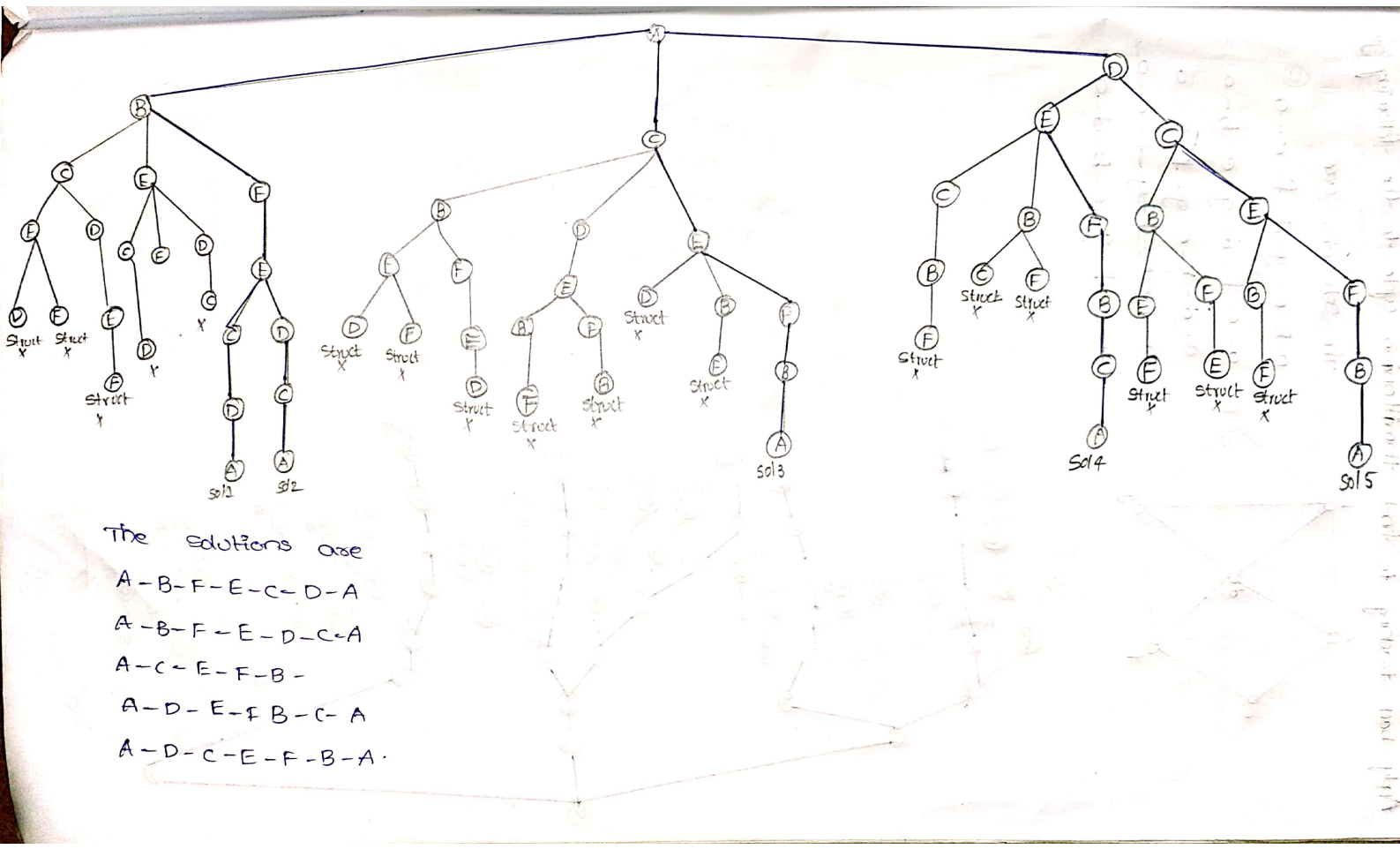
① Select root node as A. Now identify adjacent nodes of A is B, C, B, D.



② Now consider node B and identify adjacent nodes of B is A, C, E, F. Node A is already visited hence select node C







The solutions are

A - B - F - E - C - D - A

A - B - F - E - D - C - A

A - C - E - F - B -

A - D - E - F B - C - A

A - D - C - E - F - B - A.

Apply back tracking to find Hamiltonian cycle to the following graph

The solutions are

a-b-e-g-d-f-c-a

a-c-f-d-g-e-b-a

a-d-b-e-g-f-c-a

~~a-d-e-b-g-f-c-a~~

a-c-f-g-e-b-d-a

a → b, c, d

b → a, c, e

c → a, b, d, e

d → a, c, e

e → b, c, d

Draw state space tree for the following undirected graph.

The solutions are

a-b-c-e-d-a

a-b-e-c-d-a

a-b-e-d-c-a

a-c-b-e-d-a