UNIT - 05

BACK TRACKING

* It is a problem solving strategy

* Using Back Tracking we can solve constraint satisfaction problems.

* (In Back Tracking) The problem solved using backtracking has many solutions.

* All these solutions are represented in the form of a tree known as the state space tree.

* Here the problem is solved by using Recursion.

* Back tracking follows DFS.

* Applications of Back Tracking are:
  - n queens problem
  - sum of subsets
  - Graph coloring
  - Hamiltonian cycle.

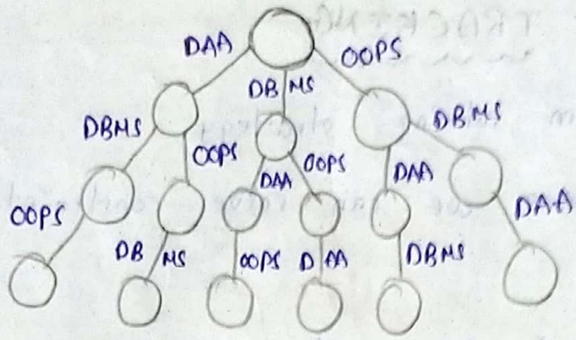
Note :

In Back Tracking, in each step, we check the condition. if the step satisfy the condition we continue generating subsequent solutions, if not we go to 1 step backward to check for another part.

Example :

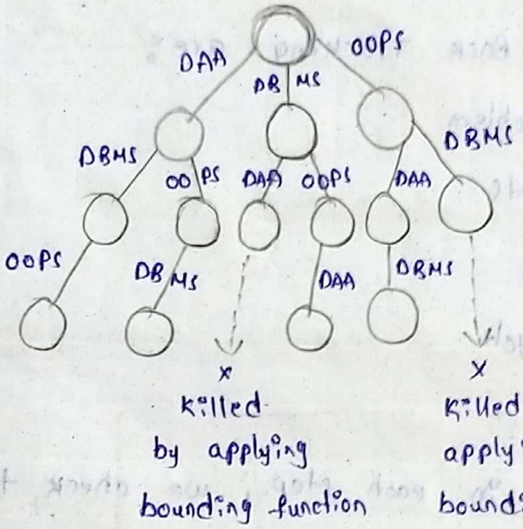
we have only 3 ~~cycles~~ classes and 3 subjects per a day, and you are asked to prepare the time table, then there are 3 factorial (3!) ways to prepare the time table for a day. and the three subjects are DAA, DBMS, OOPS.

The 6 possibilities are represented in a tree as follows:



Let us impose a constraint on the above problem then we can solve the problem using backtracking. the constraint is DAA should not be in second hour

* The state space tree for above problem using backtracking is:


DAA
DB MS
OOPS
DBMS
OO PS
DAA
OOPS
DAA
DBMS
OOPS
DB MS
DAA
DBMS
x
killed by applying bounding function
x
killed by applying bounding function.

Brute force approach vs Back Tracking:

1) In Brute force method we have many solutions for a problem and we pickup one solution as the best solution.

2) In Back Tracking we have many solutions for the problem and we list all the solutions which satisfy the constraint.

n-queens problem:

Here, n queens are given and nxn chess board is given, we have to place the queens on the chess board in such a way that no two queens are on the same row, are on the column, or on the same diagonal

- For 1 queen problem there is a trivial solution.
- For 2 queens and 3 queens problem there is no solution.
- Let us solve the 4 queens problem.

NOTE:

we have / must place first queen on the first row and second queen on the second row ---- n^{th} queen on the n^{th} row.

Here, we have to find out the column number to place the queen.

1	q_1			
2	•	•	q_2	
3	•	•	•	•
4				

Here we can't place the 3rd queen. So go back to one step and try to change the position of queen 2.

1	q_1			
2	•	•	•	q_2
3	•	q_3		
4	•	•	•	•

Here we can't place the 4th queen. So go back to one step and try to change queen 3 position but we can't change it. so again go back to previous step, but there are no more positions to check for queen2. so try to change position of queen 1.

*

1	•	q_1		
2	•	•	•	q_2
3	q_3	✗		
4	•	•	q_4	

All queens are placed.

Solution vector:

*x

2	4	1	3

*

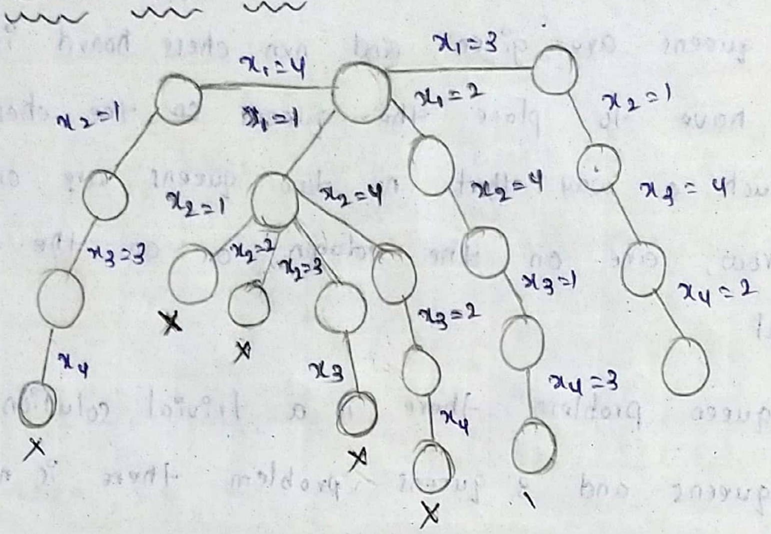
1	•	•	q_1	
2	q_2			
3	•	•	•	q_3
4	•	q_4	•	

Solution vector:

x =

3	1	4	2

State space tree:



5 queens problem:

q_1				
		q_2		
				q_3
	q_4			
			q_5	

1 3 5 2 4

q_1				
			q_2	
	q_3			
				q_4
		q_5		

1 4 2 5 3

	q_1			
			q_2	
q_3				
		q_4		
				q_5

2 4 1 3 5

	q_1			
				q_2
		q_3		
q_4				
			q_5	

2 4 3 1 5

		q_1		
q_2				
			q_3	
	q_4			
				q_5

3 1 4 2 5

		q_1		
				q_2
	q_3			
			q_4	
q_5				

3 5 2 4 1

			q_1	
q_2				
		q_3		
				q_4
	q_5			

4 1 3 5 2

			q_1	
	q_2			
				q_3
		q_4		
q_5				

4 2 5 3 1

				q_1
	q_2			
			q_3	
q_4				
		q_5		

5 2 4 1 3

				q_1
		q_2		
q_3				
			q_4	
	q_5			

5 3 1 4 2

Scanned with CamScanner

8×8 queens solutions

5					q1			
8								q2
4				q3				
1	q4							
3			q5					
6						q6		
2		q7						
7							q8	

4				q1				
1	q2							
5					q3			
8								q4
6						q5		
3			q6					
7							q7	
2		q8						

7					~~q4~~		q1	
2		q2						
6						q3		
3			q4					
1	q5							
4				q6				
8								q7
5					q8			

1	q1							
5					q2			
8								q3
6						q4		[illegible]
3			q5			~~q4~~		
7			[illegible]				q6	
2		q7						
4				q8				

5					q1			
7							q2	
2		q3						
6						q4		
3			q5					
1	q6							
4				q7				
8								q8

2		q1						
7							q2	
3			q3					
6					~~q4~~	q4		
8								q5
5				~~q6~~	q6			
1	q7							
4				q8				

8								q1
4				q2				
1	q3							
3			q4					
6						q5		
2		q6						
7							q7	
5					q8			

4				q1				
2		q2						
7							q3	
3			q4					
6						q5		
8								q6
5					q7			
1	q8							

For the following sequence solve the 8 queens problem the sequence is 6 4 7 1.

Ans: * Given the positions of first four queens

* Now we have to find the positions of next four queens.

* To find out the positions of next four queens, write the numbers that are not there in the sequence seperately.

* pick up a number from the sequence that we write seperately and check whether that queen can be placed in the picked up position, if can't be placed check for the other number, repeat this process until a correct position is found for the 5th queen

* Repeat the process for all the queens.

* Repeat the above step for remaining processes.

(6, 4, 7, 1, $\underline{3}$, $\underline{5}$, $\underline{2}$, $\underline{8}$)

					q_1		
			q_2				
						q_3	
q_4							
		q_5					
				q_6			
	q_7						
							q_8

(6, 4, 7, 1, $\underline{8}$, $\underline{2}$, $\underline{5}$, $\underline{3}$)

					q_1		
			q_2				
						q_3	
q_4							
							q_5
	q_6			q.			
				q_7			
		q_8					

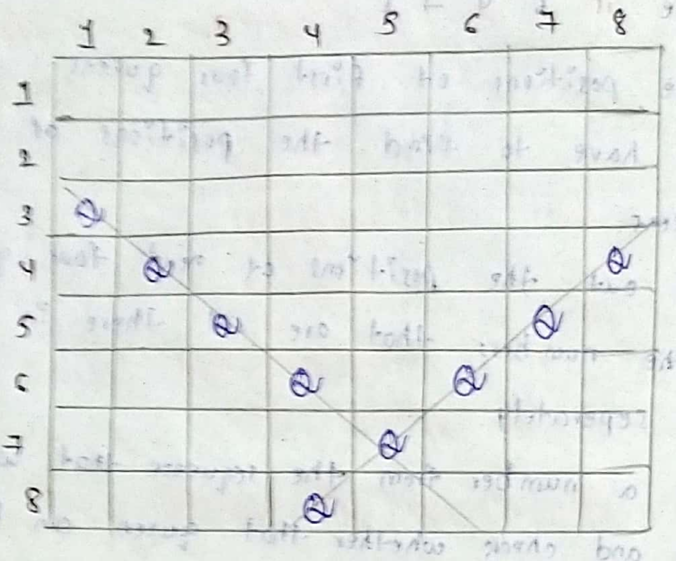
(7, 5, 3, 1, $\underline{4}$, $\underline{2}$, $\underline{8}$, $\underline{6}$)

						q_1	
				q_2			
		q_3					
q_4							
			q_5				
	q_6						
							q_7
					q_8		

(7, 5, 3, 1, $\underline{6}$, $\underline{8}$, $\underline{2}$, $\underline{4}$)

						q_1	
				q_2			
		q_3					
q_4							
					q_5		
							q_6
	q_7						
			q_8				

Checking for Diagonal :



Two queens (i, j) and (k, l) are said to be in the same diagonal if $i-j = k-l$ —— ①

Ex: The queens (3,1) (4,2) (5,3) (6,4) are in the same diagonal let $(i,j) = (3,1)$ and $(k,l) = (4,2)$

then $3-1 = 4-2$

There is another formula to said that the two queens are in the same diagonal.

i.e., $i+j = k+l$ —— ②

Ex: The queens (4,8) (5,7) (6,6) (7,5) (8,4) are in same diagonal let $(i,j) = (7,5)$, $(k,l) = (8,4)$

then $7+5 = 8+4$

From eqn ① we can write as $j-l = i-k$ —— ③

From eqn ② we can write as $j-l = k-i$ —— ④

From eqn ③ and eqn ④ we can write as

$$\boxed{|j-l| = |k-i|}$$

Note :

The queens are said to be in same diagonal (i, j) and (k, l)

$$\boxed{\text{if } |j-l| = |k-i|}$$

```
Algorithm NQueens (K, n)
{
// using backtracking, this procedure prints all possible
// placements of n queens on an nxn chessboard
// so that they are non-attacking.

  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);
     }
  }
}

Algorithm place (K, i)
// Returns true if a queen can be placed in a chessboard.
// Kth row and ith column. otherwise it returns false.
// x [ ] is a global array whose first (K-1) values
// have been set
{
   for j:=1 to K-1 do
   {
      if ((x [j] = = i) // Two in the same column
            || (Abs (x[j] - i) = = Abs (j - K))) then
                              // or on the same diagonal
         return false;
      return true;
   }
}
```

Time complexity :

The time complexity of al'gorithm is $O(n^n)$

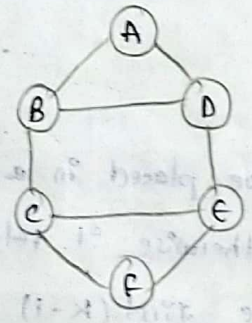
Graph coloring:

The problem here is, given a graph and an integer 'm' indicates the no.of colors. we have to assign a color to each and every node in the graph in such a way that, no two adjacent nodes of the graph receives the same color.

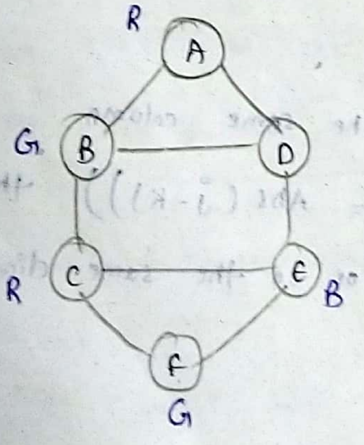
Graph coloring is also called as 'm' coloring as 'm' colors are given to color the graph.

* Back tracking is used to color the nodes in a graph. The minimum no.of colors required to color the graph is called as the "chromatic number".

Ex: Consider the following graph 'G' and color the graph with m=3.

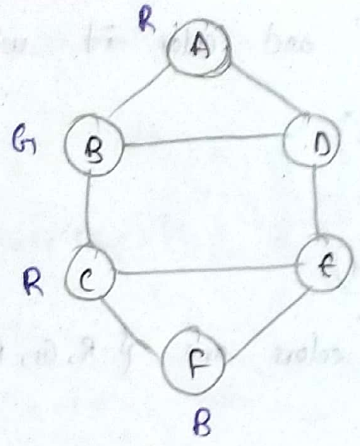


Here m=3 say the colors are {R, G, B}. Now start coloring the nodes of the graph.

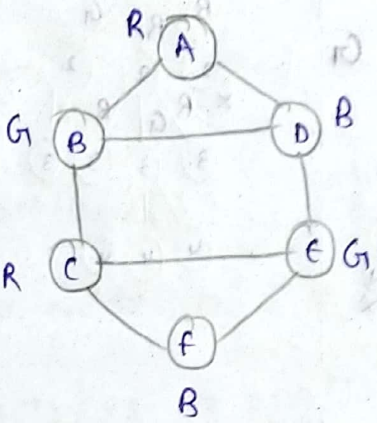


* stucks here as, we can't assign any color node 'D' because the adjacent nodes of 'D' receives all the three colors.

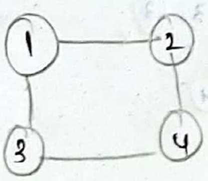
* so Back track from E to F and change the color of node 'F' to B as follows.



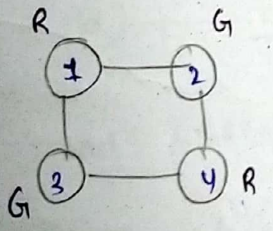
As we changed the color of node 'F' continue the process and color remaining two nodes.

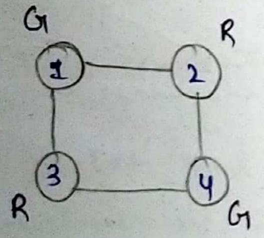


Ex2: Consider the graph 'G' and color the graph with m=2.

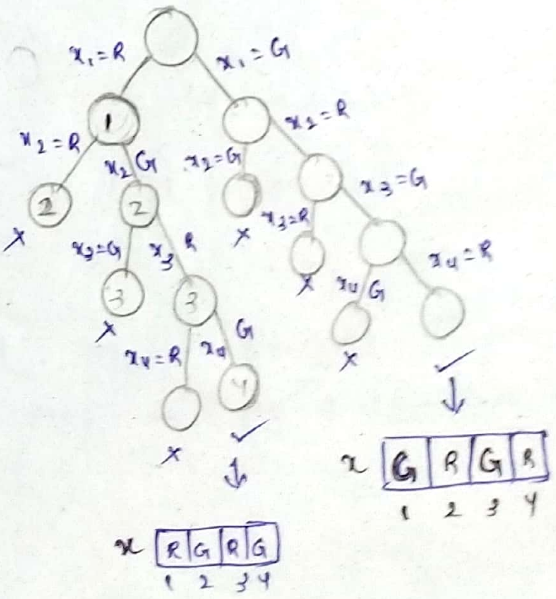


Ans: Here m=2 and say the colors are {R, G}

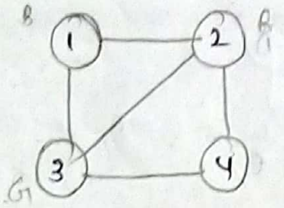




state space tree:

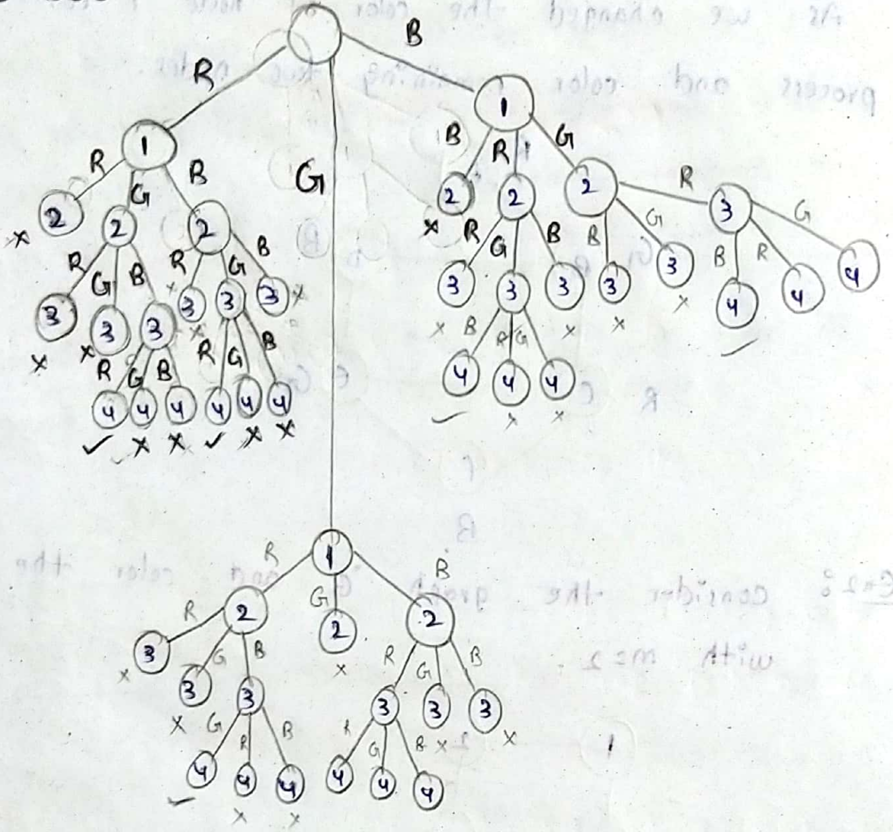


Ex3 Consider the graph 'G' and color it with m=3.



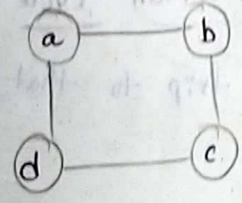
Ans: Here m=3 say the colors are {R, G, B}

state space tree:



24/10/19

```
Algorithm graphcoloring (K)
{
  for c=1 to m do
  {
    if ( issafe (K,c) )          // K is node or vertex and 'm'
    {                            // is the no.of colors
      x[K] = c;                  // G[K][i] = 1 if node 'K' is
      if (K+1 < n) then          // adjacent node 'i' otherwise '0'.
        graphcoloring (K+1);     // c is the color.
      else
        print x[];
      return ;
    }
  }
}
Algorithm issafe (K,c)
{
  for i=0 to n-1 do
  {
    if (G[K][i] == 1 && c==x[i])
      return false;
  }
  return true;
}
```



Adjacency matrix :

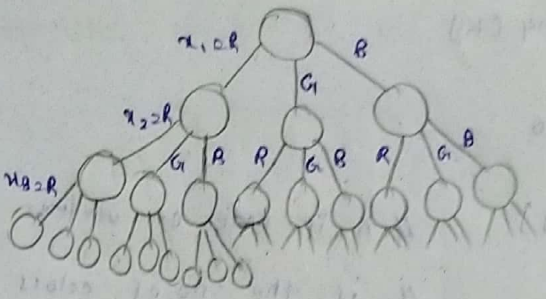
$$\begin{array}{c|cccc} & a & b & c & d \\ \hline a & 1 & 1 & 0 & 1 \\ b & 1 & 1 & 1 & 0 \\ c & 0 & 1 & 1 & 1 \\ d & 1 & 0 & 1 & 1 \end{array}$$

and solution vector :

x				

Time complexity :

Let m=3 and n=4 , say m= {R, G, B} if we construct the state space tree for the given data it looks like as follows :



The no.of possibilities here are

$$1+3+(3\times3)+(3\times3\times3)+(3\times3\times3\times3)$$

$$= 3^0+3^1+3^2+3^3+3^4 = m^n$$

Hence the time complexity is "$\underline{O(m^n)}$"

Hamiltonian Cycle / circuit:

Here a graph is given, we have to start at one node and then visit all the remaining nodes in the graph exactly once and return back to the node where we have started.

→ Here a cycle is formed with the starting node.

→ The problem is similar to the TSP problem in Dynamic programming.

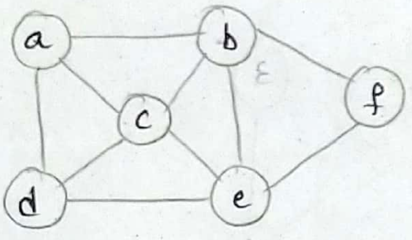
The difference between TSP and Hamiltonian cycle is:

→* In TSP, we find the round ~~off~~ trip to that gives the minimum cost.

i.e., In TSP, we have only one path that gives the minimum cost.

→* In Hamiltonian cycle we find the set of all ~~values~~ solutions that starts at one node and then visiting all the remaining nodes exactly once and return back to the same starting node.

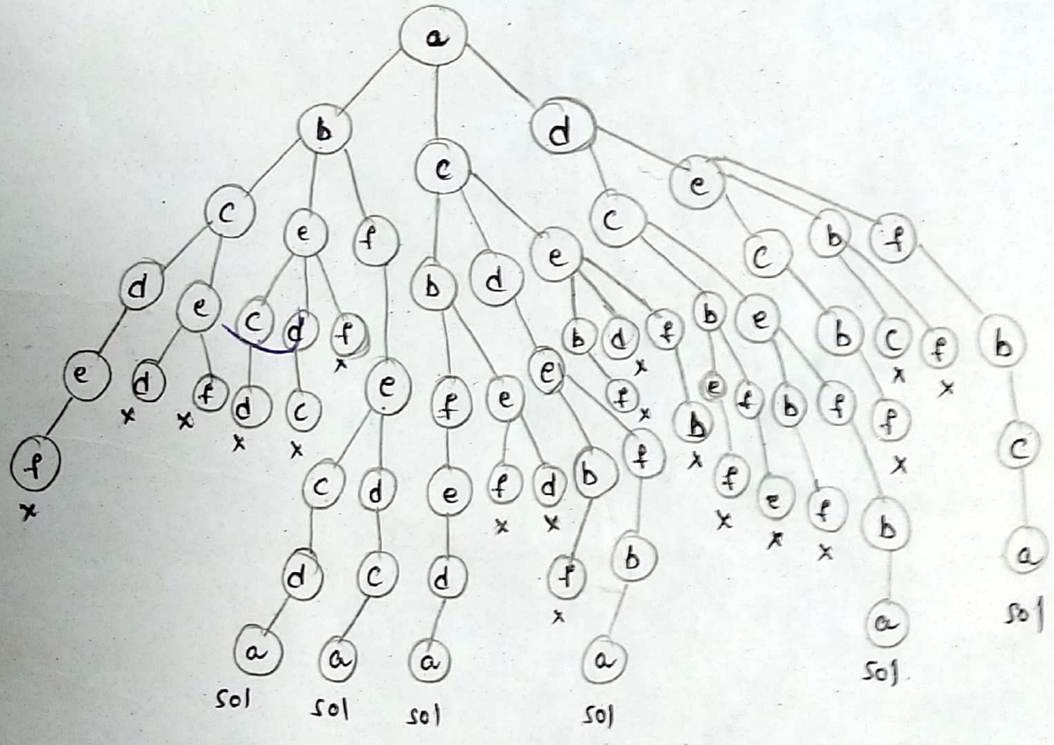
Ex1: find all the Hamiltonian cycles for the following graph.



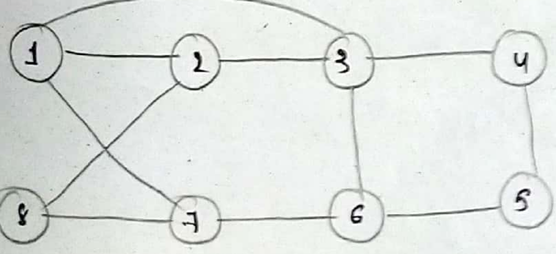
Ans: say that the source node/ starting node is 'a', the Hamiltonian cycles formed with 'a' are as follows.

* a d e f b c a
* a b f e c d a
* a c b f e d a
* a c d e f b a
* a d c e f b a
* a b f e d c a

- The state space tree for the above graph is:



Ex2:

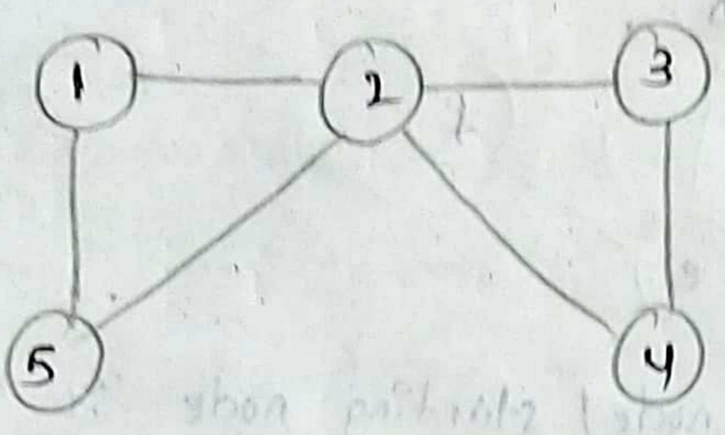


There are two solutions. They are

* 1 3 4 5 6 7 8 2 1

→* 1 2 8 7 6 5 4 3 1

Ex3:



In the above graph there are no hamiltonian cycles Because it has an articulation point. i.e,, node '2'.

→* If there is an articulation point, then the graph should not contain any hamiltonian cycles.