UNIT-5

Backtracking: Backtracking is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time.

Backtracking algorithm determines the solution by systematically searching the solution space for the given problem. Backtracking is a **depth-first search** with any bounding function. All solution using backtracking is needed to satisfy a complex set of constraints. The constraints may be explicit or implicit.

Explicit Constraint is ruled, which restrict each vector element to be chosen from the given set.

Implicit Constraint is ruled, which determine which each of the tuple in the solution space, actually satisfy the criterion function.

Algorithm:

```
if  r = n+1

   print Q[1…n]  //  we represents the positions of the queens using an array Q[1…n]

else

   for  j ← 1 to n

       legal ← true

       for i← to r --- 1

       if(Q[i] = j ) or (Q[i] = j + r-i) or (Q[i] = j – r + i) // Q[i] indicates which square in row I
contains a queen

      legal ← false

if legal

  Q[r] ← j

PLACE QUEENS(Q[1…n], r + 1)  // when PLACE QUEENS is called, the input parameters
r is the index of the first empty row and the prefix Q[1…r-1] contains the positions of the
first r-1 queens.
```

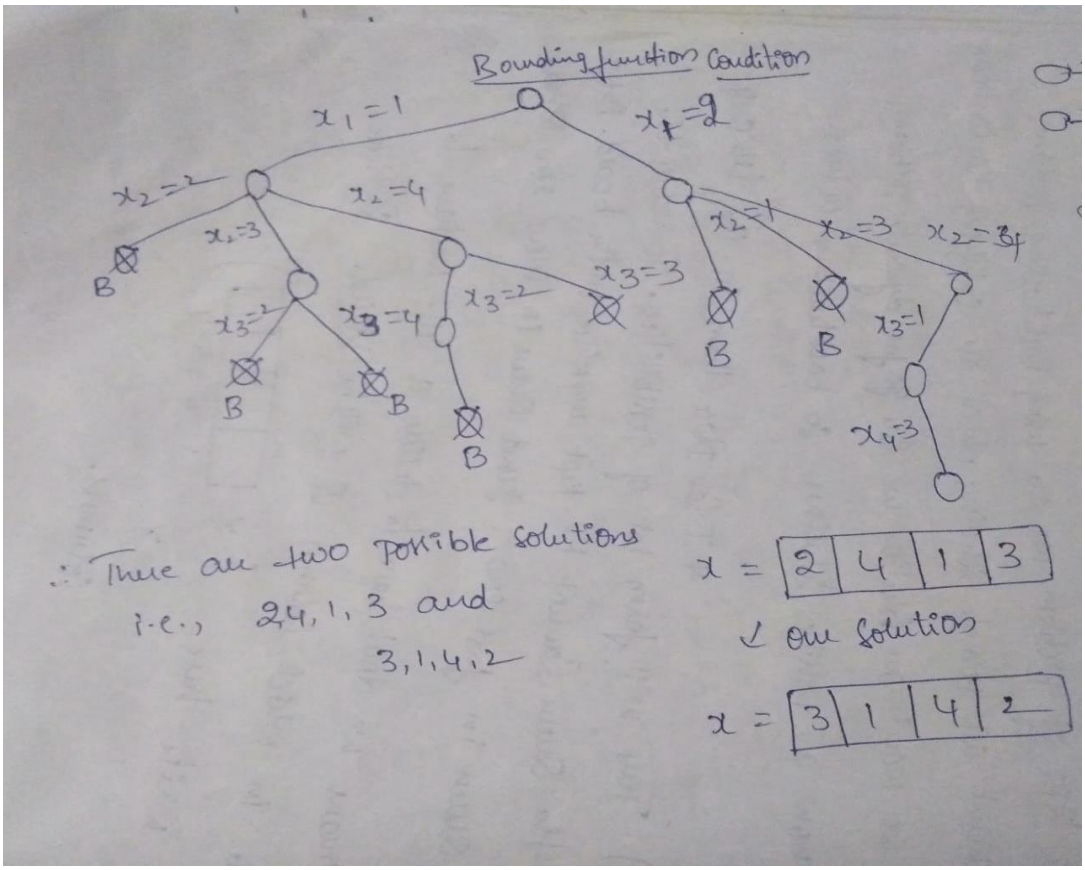
N Queens Problem: Let us understand what it means by "N-Queens Problem".

- The chessboard is given, of 4*4 means 16 cells. A standard chessboard will be 8*8 but for reducing a size of problem we are using 4*4. Here 4 queens are given (Q1Q2Q3Q4).
- So, in a game of chess, Queen will be there which will have its move like Horizontal, row or column, diagonal moves. We have to place that queen on this chessboard.
- So, all 4 queens we have to place such that, no two queens are under attack. When do you say they are under attack, if they are in same row or same column or they are in same diagonal.
- So, we have to avoid these and arrange themselves that, they are not under attack. It is possible to arrange? Yes, it is possible.
- We want all possible solutions which are satisfying this condition.

I can place them in 16c4 ways. This will form lot of possibilities. To reduce the size the queen cannot be kept anywhere on the board. But, the first queen in first row, second queen in the second row likewise.

It means we don't have to decide a cell. We have to decide in which column Q will be placed.

So, now checking all these conditions and prepares the solution in the form of "State space tree".



	1	2	3	4
1		Q1		
2				Q2
3	Q3			
4			Q4	

	1	2	3	4
1			Q1	
2	Q2			
3				Q3
4		Q4		

X =

2	4	1	3

3	1	4	2

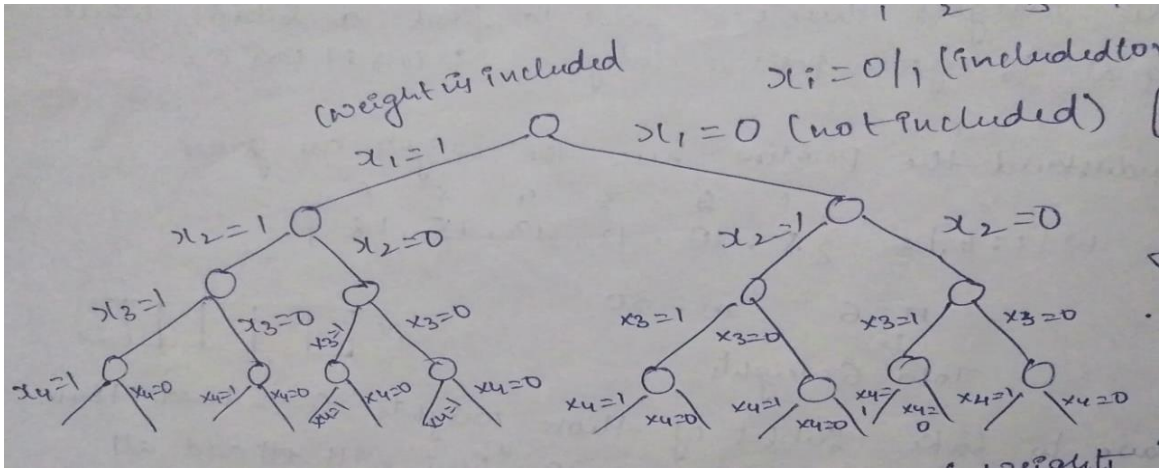
Sum of Subsets:

Let S = { S1, S2, S3,….., Sn } be a set of 'n' positive integers then we have to find a subset where sum is equals to given positive integers D or m or C.

Let us understand the problem here the weights are given

W[1:6] = { 5, 10, 12, 13, 15, 18 } n = 6(total 6 weights) m= 30

- We have to take subset of those weights such that there sum(total) is exactly m=30. If I will add all of the weights it doesn't be 30 more than 30.
- Whichever the weights that I am include I write them as solution and this solution will be either in 0/1 form. (Whether it includes or not).
- Now, how to solve the problem. So, the solution of those weights can be done in various ways.
- So, let us try all those possible ways of their selection and find out which of those selection is giving us this exact weight **M = 30.**



Like there are 6 weights I will be getting 7 levels including the root and the height will be '6'.

So total how many paths are making $2^6 = 2^n$ it's exponential and time consuming. So, the amount of time taken for solving this one is 2^n because that many paths are generating.

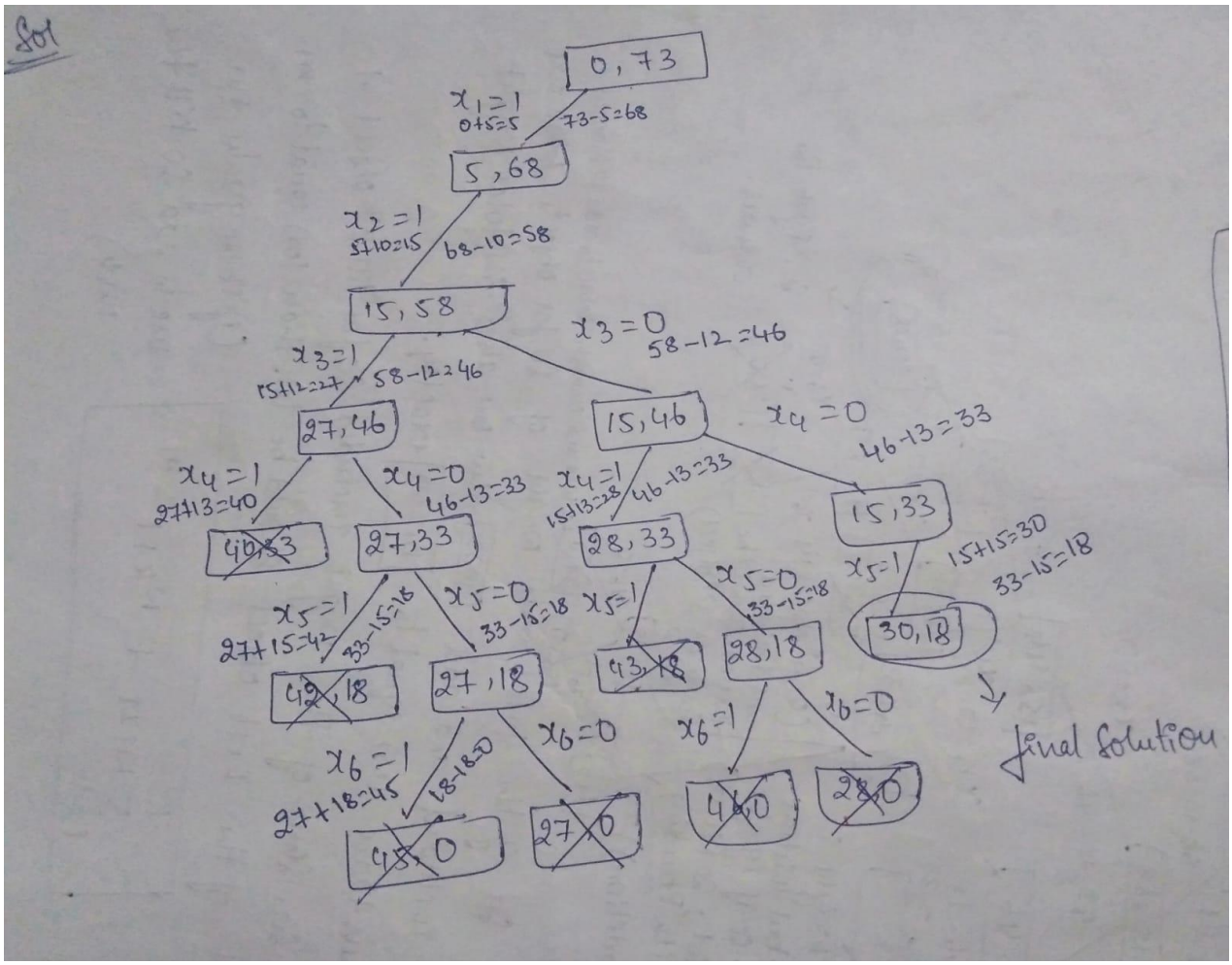
When we apply Backtracking we try to kill the nodes if they are not satisfying the Bounding Function.

Conditions: i) $\sum_{i=1}^{k} \text{wixi} + \text{wk} + 1 \leq \text{m}$

So, sum of the weights included so far till k^{th} object + weight of the next object should be less than or equal to m. If it is greater no use to add so, kill the node.

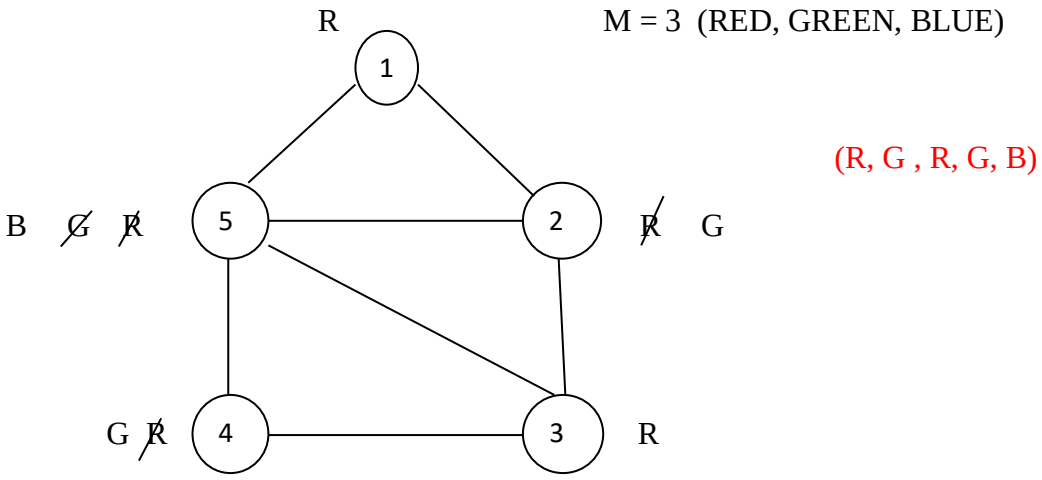
ii) $\sum_{i=1}^{k} \text{wixi} + \sum_{i=k+1}^{n} \text{wi} > m$

So far what all you have included and remaining weights i takes values from k+1 to n should be greater than m.



Graph Coloring: This problem is solved by using Backtracking. First let us know what the problem is. See graph is given and colors are given i.e., (red, green, blue) m=3.

The problem is we have to color the vertices of a graph such that, no two neighboring vertices or adjacent vertices have same color.



Let us try and see what colors we can give possible or not possible.

So, I will start from vertex 1 R now I pick up vertex 2. Who are neighboring to 2 {1,3} already red color is there so, this 2 vertex cannot placed by red. Vertex 2 either placed by green or blue.

Now go to vertex 3 can I give Red color yes, it is not adjacent to vertex 1. Vertex 4 can I give red color no, because it is adjacent to vertex 3 can I give green color yes. Vertex 5 directly I am assigning Blue because vertex 1 and 4 having color red and green it is not possible to assign the same color.

Is it the only solution or other possible solutions? Yes, I can color it in another way also. Here we want all possible solutions what are the ways they can be colored. So it is like permutation and trying out all possibilities and then picking up one which satisfying the condition.

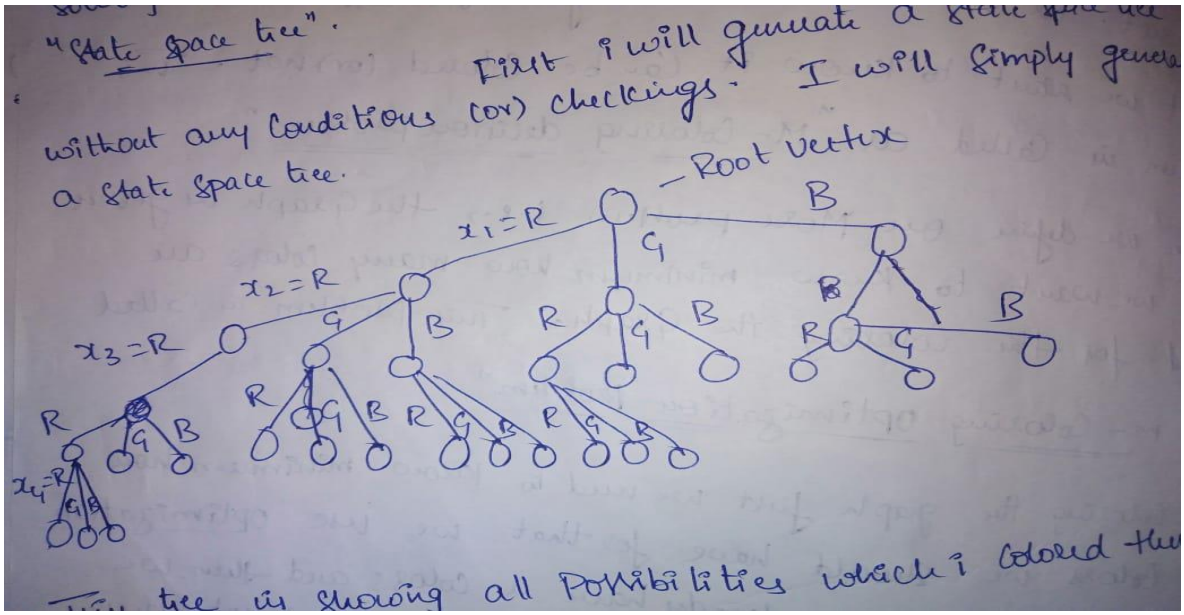
Condition: Neighboring vertices should not have a same color. This type of problem can be solved by using backtracking.

- See, here 3 colors are given if I give just 2 colors can the graph will be colored in this one. No it cannot be colored.
- If I give 4 colors can the graph colored yes, it can be colored. So, we will define one type of problem here.
- If a graph is given and colored too just we want to know graph can be colored by using those colors or not.
- Just we want to know it can be colored or not. This problem is called **"M-Coloring Decision Problem" or "Chromatic Number Problem".**

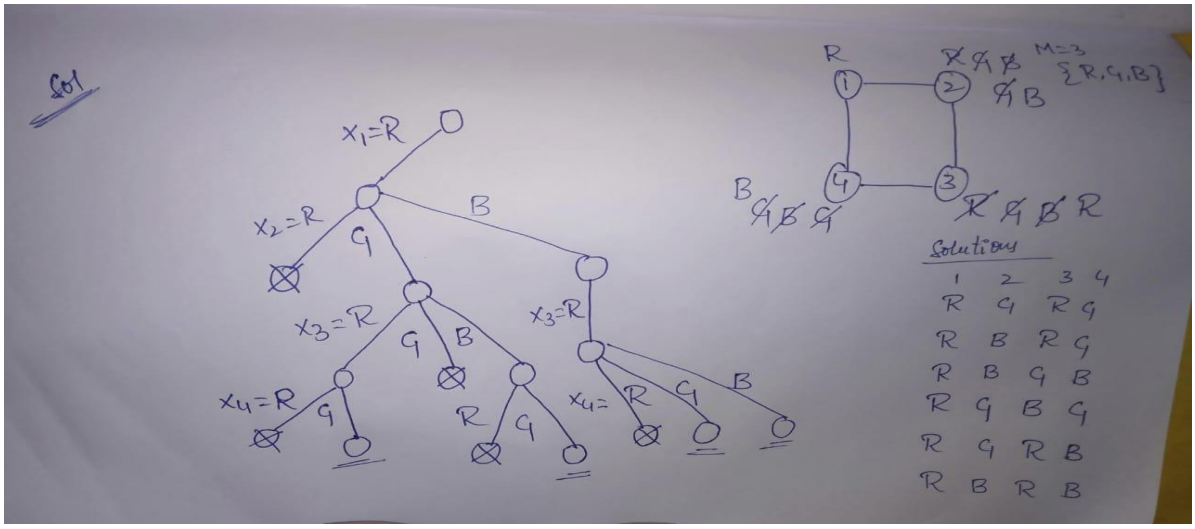
- One more problem we are going to define i.e., the graph is given and we want to know minimum how many colors are required for coloring the graph. This problem is called **"M-Coloring Optimization Problem".**

Now, I am taking a simple graph and show you how this can be solved by using Backtracking. Method of solving a problem in Backtracking is while generating a "State Space Tree".

First I will generate a state space tree without any conditions.



This tree is showing all possibilities which I colored them without checking the adjacency condition. Without using a condition it gives a big solution but, in backtracking we impose the conditions and solve the problem.



Hamiltonian cycle: Let us know that what Hamiltonian cycle is, see if a graph is given then we have to start from starting vertex and visit all the vertices exactly once and return back to the starting vertex.

That forms a cycle we have to check that is there any Hamiltonian cycle is possible in a given graph. If it is possible then what is a cycle and if there are multiple cycles we have to find out all those cycles. So, the problem is to find out if there is any Hamiltonian cycle in a graph or not.

- First of all the graph given may be directed or non directed but it must be connected. If it is not connected the cycle is not possible.
- This is a NP-Hard problem means exponential taking problem. So, there is no easy way to find out the Hamiltonian cycle presented in a graph or not.

Algorithm:

```
Algorithm Hamiltonian (K)
{
 do
{
Next vertex (k);
If(x[k]==0)
Return;
If(k==n)
Print (x[1:n]);
Else
Hamiltonian (k+1);
} while (true);
}
Algorithm Next vertex(k)
{
do
{
x[k] = (x[k]+1)  mod (n+1);
```

if(x[k]==0) return;

if(G[x[k-1], x[k]] ≠ 0)

{

For j=1 to k-1 do if (x[j]==x[k]) break;

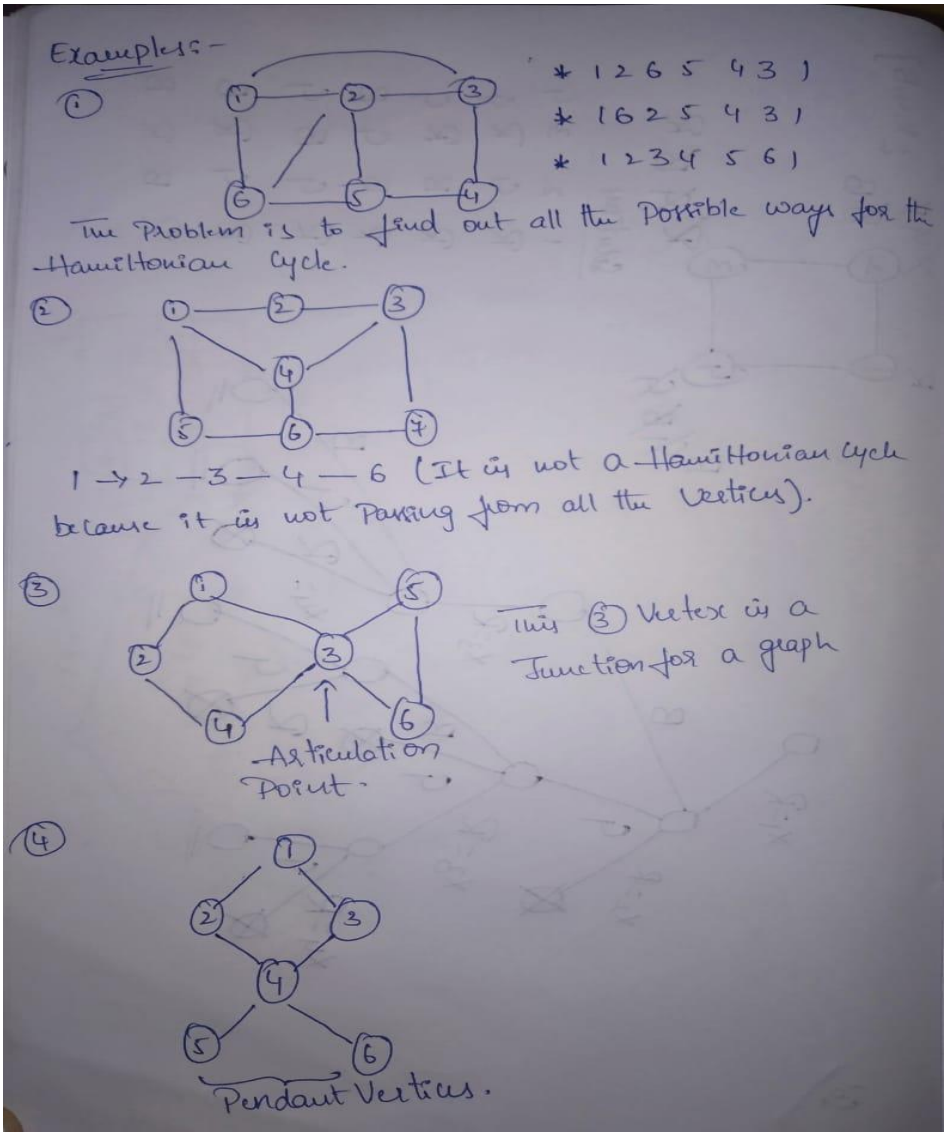
If(j==k)

If(k<n or (k==n) && G[x[n], x[1]]) ≠ 0

Return;

} while (true);

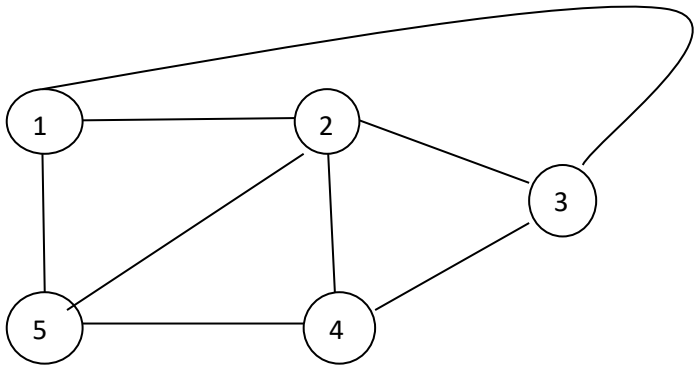
}



Bounding Function:

- Should not take duplicate function.
- Whenever you take any vertex there should be an edge from previous vertex.
- If you are in the last vertex there should be an edge to the starting vertex.

Example:



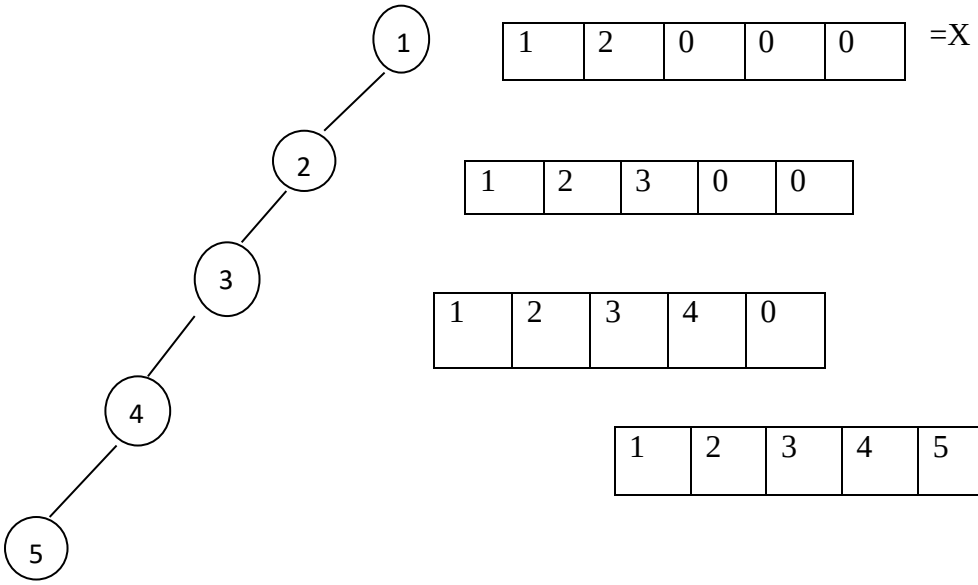
$$G = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Initially all these values are zero X=

0	0	0	0	0

Starting vertex is 1 X=

1	0	0	0	0



1	2	0	0	0

=X

1	2	3	0	0

1	2	3	4	0

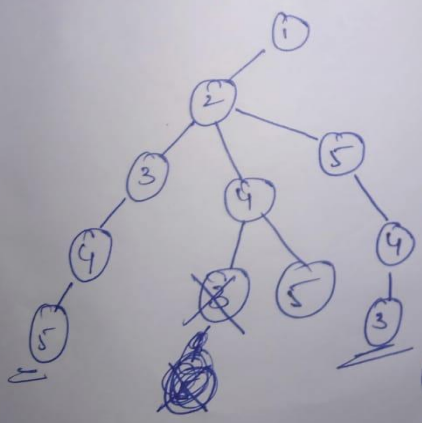
1	2	3	4	5

x | 1 | 2 | 3 | 4 | 0 | → 0 Means stopped (go back to next one)

x | 1 | 2 | 3 | 5 | 0 | → 3-5 edge is not there so it is invalid. After 5 it comes what zero

x | 1 | 2 | 3 | 0 | 0 |

→ It likes 0 1 2 3 4 5 0 then 1 2 3 4 5 0.



① | 1 | 2 | 4 | 0 | 0 |
2-4 edge

② | 1 | 2 | 4 | 3 | 0 |
4-3 edge

③ | 1 | 2 | 4 | 3 | 5 |
3-5 edge is not there. so again zero

④ | 1 | 2 | 4 | 3 | 0 |

⑤ | 1 | 2 | 4 | 5 | 0 |
4-5 edge is there

⑥ | 1 | 2 | 4 | 5 | 3 |
5-3 edge is not there. again zero

⑦ x | 1 | 2 | 5 | 0 | 0 |
2-5 edge

⑧ x | 1 | 2 | 5 | 4 | 0 |
5-4 edge is there

⑨ x | 1 | 2 | 5 | 4 | 3 |

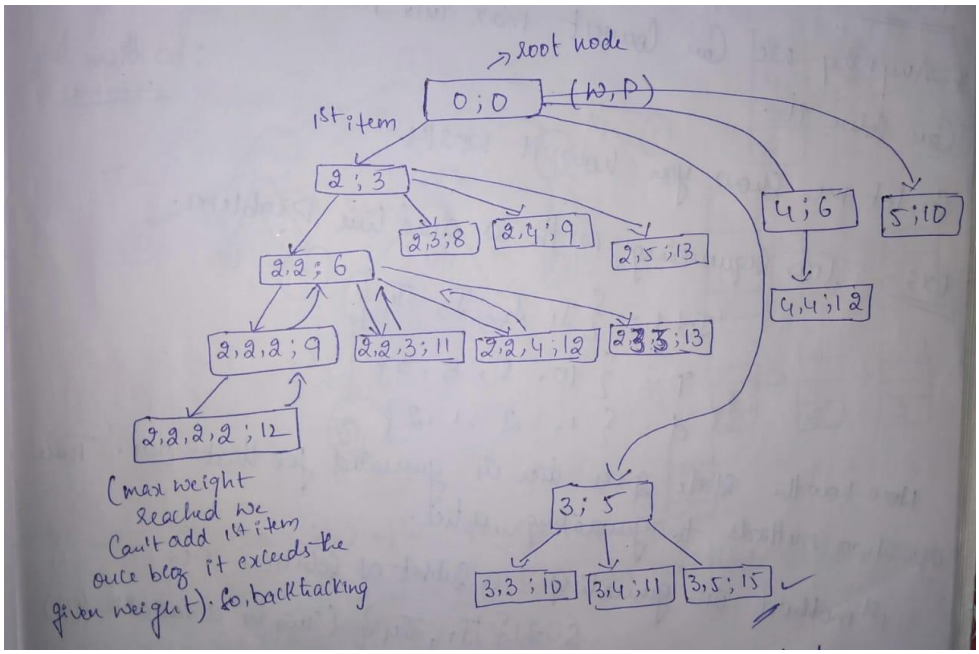
Solution 1, 2, 3, 4, 5 ;)
1, 2, 5, 4, 3 ;)

Knapsack problem:

The knapsack problem can carry weight not exceeding W. Our aim is to fill the knapsack in a way that maximizes the value of the included the objects, while respecting the capacity constraint. Here we can't take fractions.

Example: M = 8 kgs I = 4 items

W	2	3	4	5
P	3	5	6	10



By applying constraints and by arranging the recomputation we can create one small tree(partial tree).

So final solution is (3,5) = 8kgs and maximum profit is 15