Unit - IV: STATE SPACE SEARCH ALGORITHMS

PART – A

1. Differentiate backtracking and exhaustive search.

S.No	Backtracking	Exhaustive search
1.	Backtracking is to build up the solution vector one component at a time and to use modified criterion Function Pi(x1,..,xi) (sometimes called bounding function) to test whether the vector being formed has any chance of success.	Exhaustive search is simply a “brute – force” approach to combinatorial problems. It suggests generating each and every element of the problem’s domain, selecting those of them that satisfy the problem’s constraints, and then finding a desired element.
2.	Backtracking makes it possible to solve many large instances of NP-hard problems in an acceptable amount of time.	Exhaustive search is impractical for large instances, however applicable for small instances of problems.

2. What are the factors that influence the efficiency of the backtracking algorithm?

The efficiency of the backtracking algorithm depends on the following four factors. They are:

i. The time needed to generate the next xk

ii.The number of xk satisfying the explicit constraints.

iii.The time for the bounding functions Bk

iv.The number of xk satisfying the Bk.

3. What is the n-queens problem?

The problem is to place ‘n’ queens on an n-by-n chessboard so that no two queens attack each other by being in the same row or in the column or in the same diagonal.

4. Define the Hamiltonian cycle.

The Hamiltonian is defined as a cycle that passes through all the vertices of the graph exactly once. It is named after the Irish mathematician Sir William Rowan Hamilton (1805- 1865). It is a sequence of n+1 adjacentverticesvi0, vi1… vin-1, vi0 where the first vertex of the sequence is same as the last one while all the other n-1 vertices are distinct.

5. When can a node be terminated in the subset-sum problem?

The sum of the numbers included are added and given as the value for the root as s’. The node can be terminated as a non-promising node if either of the two equalities holds: s’+si+1>d (the sum s’ is too large)

$\sum_{j=i\_1}^{n} sj < d$ (the sum s’ is too small)

6. How can the output of a backtracking algorithm be thought of?

The output of a backtracking algorithm can be thought of as an n-tuple (x1, …xn) where each coordinate xi is an element of some finite linearly ordered set Si. If such a tuple (x1, …xi) is not a solution, the algorithm finds the next element in Si+1 that is consistent with the values of (x1, …xi) and the problem's constraints and adds it to the tuple as its (I+1)st coordinate. If such an element does not exist, the algorithm backtracks to consider the next value of xi, and so on.

7. Give a template for a generic backtracking algorithm.

```
ALGORITHM Backtrack(X [1..i])
//Gives a template of a generic backtracking algorithm
//Input X[1..i] specifies the first I promising components of a solution
//Output All the tuples representing the problem's solution
if X[1..i] is a solution write X[1..i]
else for each element x Si+1 consistent with X[1..i] and the constraints do
X[i+1] = x
Backtrack(X[1..i+1])
```

8. State m color-ability decision problem.

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

9. Define implicit constraint.

They are rules that determine which of the tuples in the solution space of I satisfy the criteria function. It describes the way in which the xi must relate to each other.

10. What is a promising node in the state-space tree?

A node in a state-space tree is said to be promising if it corresponds toa partially constructed solution that may still lead to a complete solution.

11. Define a state space tree.

The tree organization of the solution space is referred to as state space tree.

12. Compare backtracking and branch and bound.

Backtracking	Branch and bound
State-space tree is constructed using depth-first search	State-space tree is constructed using best-first search
Finds solutions for combinatorial non optimization problems	Finds solutions for combinatorial optimization problems
No bounds are associated with the nodes in the state-space tree	Bounds are associated with the each and every node in the state-space tree

13.Give an example for a sum-of-subset problem.

Subset sum problem is to find subset of elements that are selected from a given set whose sum adds up to a given number K. We are considering the set contains non-negative values. It is assumed that the input set is unique (no duplicates are presented).

14.Differentiate feasible solution and optimal solution.

A solution (set of values for the decision variables) for which all of the constraints in the Solver model are satisfied is called a feasible solution. An optimal solution is a feasible solution where the objective function reaches its maximum (or minimum) value.

15.Write the formula for the decision tree for searching a sorted array?

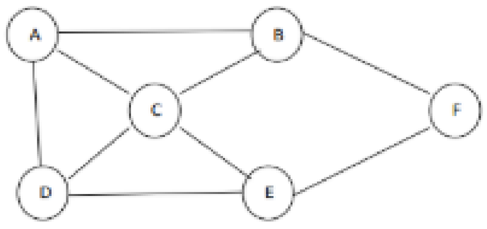
$$C_{worst}(n) \ge \lceil \log_2 n! \rceil.$$

Using Stirling's formula for $n!$, we get

$$\lceil \log_2 n! \rceil \approx \log_2 \sqrt{2\pi n}(n/e)^n = n \log_2 n - n \log_2 e + \frac{\log_2 n}{2} + \frac{\log_2 2\pi}{2} \approx n \log_2 n.$$

Part – B

1. Write an algorithm to determine the Hamiltonian cycle in a given graph using backtracking. For the following graph determine the Hamiltonian cycle. (13 marks)



ANS: Here's an algorithm to determine the Hamiltonian cycle in a given graph using backtracking:

1. Initialize an empty list to store the Hamiltonian cycle.
2. Start from any vertex in the graph.
3. Call the recursive function Hamiltonian Util (vertex, cycle).

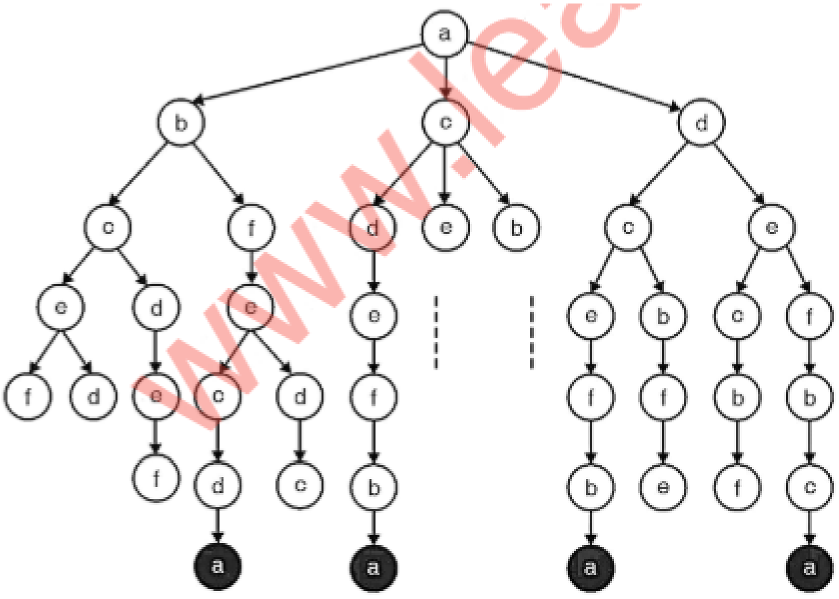
```
Function Hamiltonian Util (vertex, cycle):
  a. Add the current vertex to the cycle.
  b. If the cycle contains all vertices:
     i. If there is an edge from the last vertex in the cycle to the starting vertex:
        - Return the cycle as the Hamiltonian cycle.
  c. Else:
        i. For each neighbour of the current vertex:
            - If the neighbour is not already in the cycle:
                - Call Hamiltonian Util recursively with the neighbour and the updated cycle.
    d. Remove the current vertex from the cycle.

4. Start from each vertex in the graph and call Hamiltonian Cycle.
5. Return the first Hamiltonian cycle found or indicate if no cycle exists.
```

Description: This algorithm uses backtracking to explore all possible paths in the graph to find a Hamiltonian cycle.

Input: The input consists of an undirected, connected graph G = <V, E>, where V represents the set of vertices and E represents the set of edges. An initial vertex is also provided to start the search.

Output: The output is the Hamiltonian cycle found in the graph, if it exists. If no Hamiltonian cycle is found, the algorithm indicates that no cycle exists.

Time Complexity: The time complexity of this algorithm is O (2^N * N^2), where N is the number of vertices in the graph. This is because the algorithm explores all possible paths in the worst case, and at each step, it checks if a vertex is adjacent to another vertex, which takes O(N) time. The number of paths to explore is exponential due to the nature of backtracking.



2. Explain the 4 - queen's problem using backtracking. Write the algorithms. Give the estimated cost for all possible solutions of the 4 - queen's problem. Specify the implicit and explicit constraints. (13 marks)

ANS: The 4-queen problem is a classic puzzle in which you aim to place four queens on a 4x4 chessboard so that no two queens threaten each other. This means no two queens share the same row, column, or diagonal. Backtracking is commonly used to solve this problem.

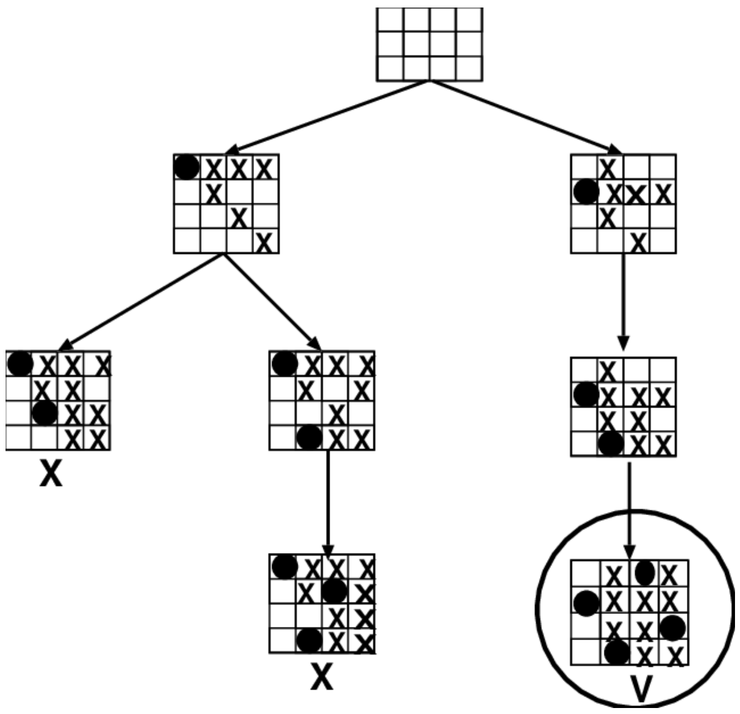
Algorithm for solving the 4-queen problem using backtracking:
1. Start with an empty 4x4 chessboard.
2. Place a queen in the first row, first column.
3. Move to the next column and place the next queen in a safe position.
4. Repeat step 3 until all queens are placed or until it's not possible to place the next queen safely.
5. If all queens are placed, return the solution.
6. If it's not possible to place the next queen safely, backtrack to the previous queen and try a different position.
7. Repeat steps 3-6 until all possible solutions are found.

Estimated cost for all possible solutions:
The estimated cost depends on the efficiency of the backtracking algorithm. Since the 4-queen problem has a small search space, the cost should be relatively low, especially with efficient pruning techniques.

Implicit constraints:
1. Each queen must be placed on a different row and column.
2. No two queens can share the same row, column, or diagonal.

Explicit constraints:
1. The chessboard is 4x4, limiting the number of possible positions for the queens.
2. Backtracking is used to explore all possible configurations efficiently.

This problem is small enough that the cost for all possible solutions can be calculated by exhaustively searching through the solution space, making it feasible to determine the exact number of solutions.



Pseudo code for n – queen’s problem

```
function n _ queens(n):
    solutions = []
    solve_ queens([None] * n, 0, n, solutions)
    return solutions

function solve _ queens (board, row, n, solutions):
    if row == n:
        solutions. Append (copy(board)) // Found a solution, copy board state
        return

    for col from 0 to n-1:
        if is _ safe (board, row, col):
            board[row] = col
            solve _queens (board, row + 1, n, solutions)

function is _ safe (board, row, col):
    for i from 0 to row - 1:
        if board[i] == col or abs(board[i] - col) == row - i:
            return false
    return true
```

2. Explain the n-Queens problem and trace it for n=6.

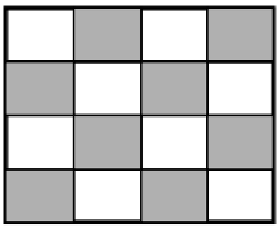
ANS:

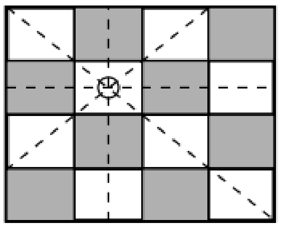
The n-queen's problem can be stated as follows.

Consider a n × n chessboard on which we have to place n queens so that no two queens attack each other by being in the same row or in the same column or on the same diagonal.

For example -

Consider 4 × 4 board.





The next queen - if is placed on the paths marked by dotted lines then they can attack each other

- **2-Queen's problem is not solvable -** Because 2-queens can be placed on 2 × 2 chessboard as

Q	Q

Illegal

Q	
Q	

Illegal

Q	
	Q

Illegal

Q	Q

Illegal

	Q
	Q

Illegal

- But 4-queen's problem is solvable.

		Q	
Q			
			Q
	Q		

⇐ Note that no two queens can attack each other.

For n = 6 the solution would be

	1	2	3	4	5	6
1		Q				
2				Q		
3						Q
4	Q					
5			Q			
6					Q	

```
Algorithm  Queen(n)
//Problem description : This algorithm is for implementing n
//queen's problem
//Input : total number of queen's n.
          for column ←1 to n do                        This function checks if
          {                                            two queens are on the
               if(place(row,column))then               same diagonal or not.
               {
                board[row]column//no conflict so place queen
               if(row=n)then//dead end
           print_board(n)
               //printing the board configuration
               else//try next queen with next position
                    Queen(row+1,n)
               }                                Row by row each queen
          }                                     is placed by satisfying
                                                constraints.
Algorithm place(row,column)
//Problem Description : This algorithm is for placing the
//queen at appropriate position
//Input : row and column of the chessboard
//Output : returns 0 for the conflicting row and column
//position and 1 for no conflict.
          for i ← 1 to row–1 do
          { //checking for column and diagonal conflicts
           if(board[i] = column)then

           return 0           Same column by 2 queen's

          else if(abs(board[i]– column) = abs(i – row))then
               return 0
          }                                   This formula gives that 2 queens
          //no conflicts hence Queen can be placed      are on same diagonal
     return 1
```

4.How does backtracking work on the 8-Queens problem with suitable examples? (13 marks)

- In the backtracking method,
  1. The desired solution is expressible as an n tuple (x_1, x_2,...x_n) where x_i is chosen from some finite set S_i.
  2. The solution maximizes or minimizes or satisfies a criterion function C (x_1, x_2,...x_n).
- The problem can be categorized into three categories.

  For instance - For a problem P let C be the set of constraints for P. Let D be the set containing all solutions satisfying C then,

  Finding whether there is any feasible solution ? - is the decision problem.

  What is the best solution ? - is the optimization problem.

  Listing of all the feasible solution - is the enumeration problem.
- The **basic idea of backtracking** is to build up a vector, one component at a time and to test whether the vector being formed has any chance of success.
- The **major advantage** of this algorithm is that we can realize the fact that the partial vector generated does not lead to an optimal solution. In such a situation that vector can be ignored.
- Backtracking algorithm determines the solution by systematically searching the solution space (i.e. set of all feasible solutions) for the given problem.
- Backtracking is a depth first search with some bounding function. All solutions using backtracking are required to satisfy a complex set of constraints. The constraints may be explicit or implicit.
- **Explicit constraints** are rules, which restrict each vector element to be chosen from the given set. **Implicit constraints** are rules, which determine which of the tuples in the solution space, actually satisfy the criterion function.

or example

xample

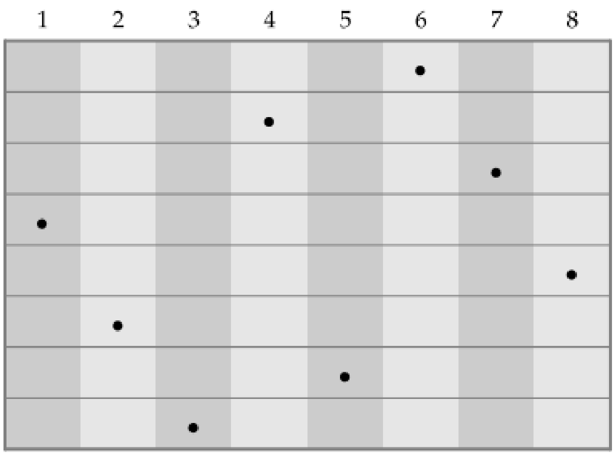
- **8-Queen's problem** - The 8-queen's problem can be stated as follows. Consider a chessboard of order 8 × 8. The problem is to place 8 queens on this board such that no two queens can attack each other. That means no two queens can be placed on the same row, column or diagonal.

The solution to 8-queens problem can be obtained using backtracking method.

The solution can be given as below -

The solution to 8-queens problem can be obtained using backtracking method.

The solution can be given as below -



Example : Sum of subsets -

There are n positive numbers given in a set. The desire is to find all possible subsets of this set, the contents of which add onto a predefined value M.

In other words,

Let there be n elements given by the set w = (w_1, w_2, w_3,..., w_n) then find out all the subsets from whose sum is M.

For example -

Consider n = 6 and (w_1, w_2, w_3, w_4, w_5, w_6) = (25, 8, 16, 32, 26, 52) and M = 59 then we will get desired sum of subset as (25, 8, 26).

We can also represent the sum of subset as (1, 1, 0, 0, 1, 0). If solution subset is represented by an n-tuple (x_1, x_2,...x_n) such that x_i could be either 0 or 1. The x_i = 1 means the weight w_i is to be chosen and x_i = 0 means that weight w_i is not to be chosen.

5. State the subset-sum problem and Complete state-space tree of the backtracking algorithm applied to the instance A= {3, 5, 6, 7} and d=15 of the subset-sum problem. (13 marks)

ANS: Subset-Sum Problem is finding a subset of a given set S = {s1,s2….sn} of n positive integers whos**e** sum is equal to a given positive integer d.

For example, for S = {1, 2, 5, 6, 8) and d = 9, there are two solutions: {1, 2, 6} and {1, 8}. Of course, some instances of this problem may have no solutions.

It is convenient to sort the set's elements in increasing order. So we will assume that **s1 ≤ s2 ≤ ……. ≤ sn**

The state-space tree can be constructed as a binary tree as that in the following figure for the instances S = (3, 5, 6, 7) and d = 15.

The root of the tree represents the starting point, with no decisions about the given elements made as yet.

Its left and right children represent, respectively, inclusion and exclusion ofs1 in a set being sought.

Similarly, going to the left from a node of the first level corresponds to inclusion of s2, while going to the right corresponds to its exclusion, and soon.

Thus, a path from the root to a node on the I' th level of the tree indicates which of the first i numbers have been included in the subsets represented by that node.

We record the value of s'the sum of these numbers, in the node, Ifs is equal to d. we have a solution to the problem.

We can either, report this result and stop or, if all the solutions need to he found, continue by backtracking to the node's parent.

If s' is not equal to d, we can terminate the node as non-promising if either of the two inequalities holds:

$$s' + s_{i+1} > d \text{ (the sum } s' \text{ is too large)}$$

$$s' + \sum_{j=i+1}^{n} s_j < d \text{ (the sum } s' \text{ is too small).}$$

1. Pseudocode For Backtrack Algorithms

```
ALGORITHM   Backtrack(X[1..i])
    //Gives a template of a generic backtracking algorithm
    //Input: X[1..i] specifies first i promising components of a solution
    //Output: All the tuples representing the problem's solutions
    if X[1..i] is a solution write X[1..i]
    else    //see Problem 8
        for each element $x \in S_{i+1}$ consistent with X[1..i] and the constraints do
            X[i + 1] ← x
            Backtrack(X[1..i + 1])
```

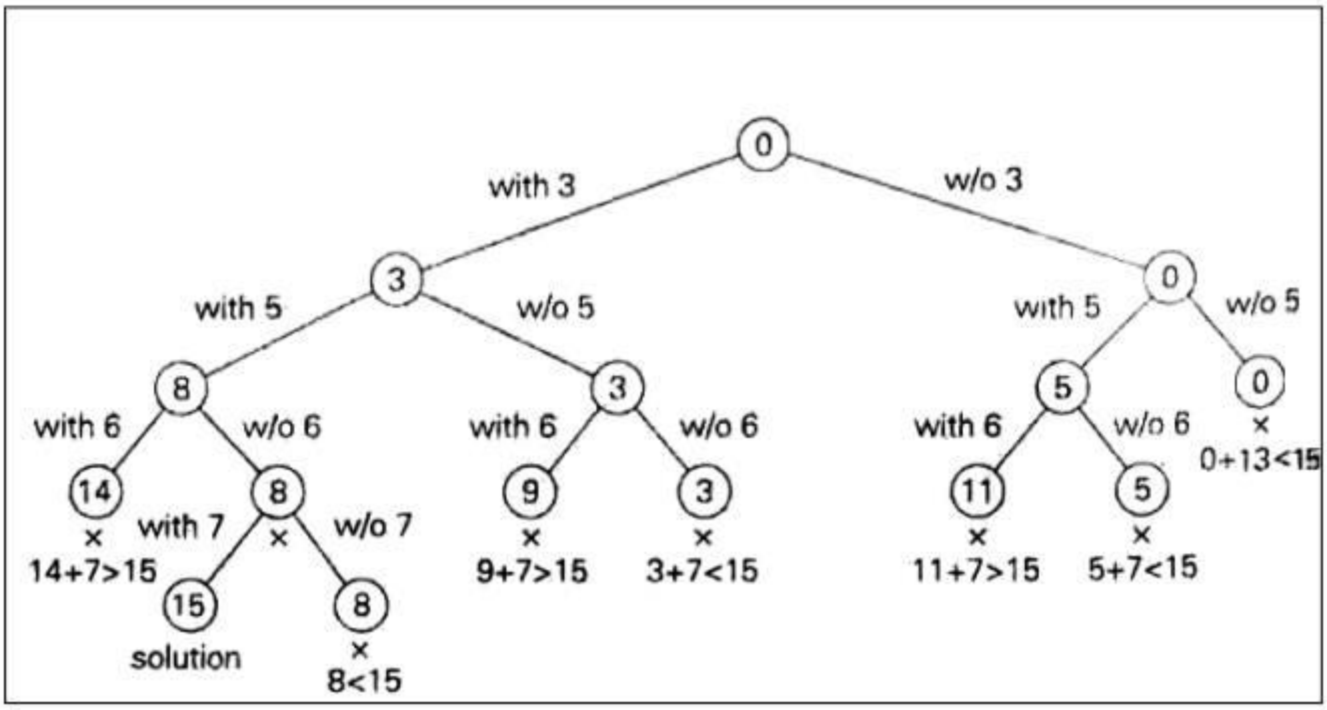


Fig: Complete state-space tree of the backtracking algorithm

(applied to the instance S = (3, 5, 6, 7) and d = 15 of the subset-sum problem. The number inside a node is the sum of the elements already included in subsets represented by the node. The inequality below a leaf indicates the reason for its termination)

6. Explain in detail about knapsack problem with example.

(13 marks)

ANS:

The Knapsack problem is a classic optimization problem where given a set of items, each with a weight and a value, the goal is to determine the maximum value that can be obtained by selecting a subset of the items such that the total weight of the selected items does not exceed a given capacity.

Formally, given a set of nn items, where item ii has a weight $wiwi$ and a value $vivi$, and a knapsack capacity WW, the Knapsack problem can be stated as:

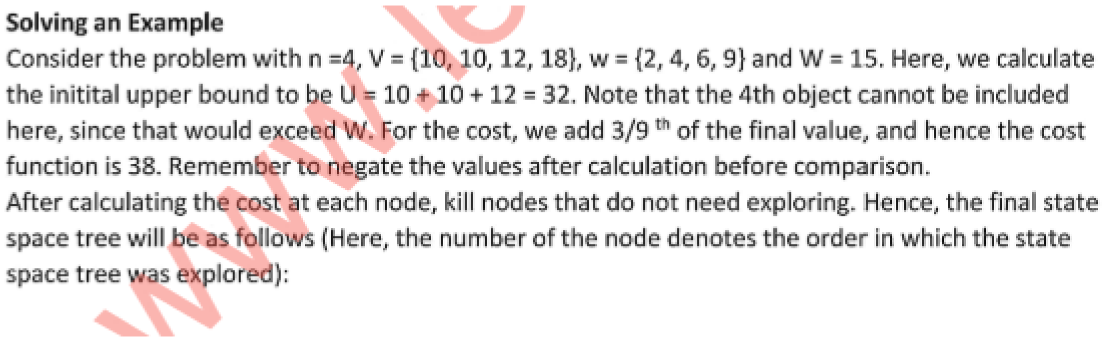
Maximize $\sum i=1nvi\cdot xi\sum i=1nvi\cdot xi$ subject to $\sum i=1nwi\cdot xi\leq W\sum i=1nwi\cdot xi\leq W$ where $xixi$ is a binary decision variable indicating whether item ii is selected (1) or not (0).

The branch and bound algorithm is a design paradigm used to solve combinatorial optimization problems like the Knapsack problem. It explores the solution space by systematically considering different combinations of items and uses lower bounds to prune branches of the search tree, reducing the computational effort required.

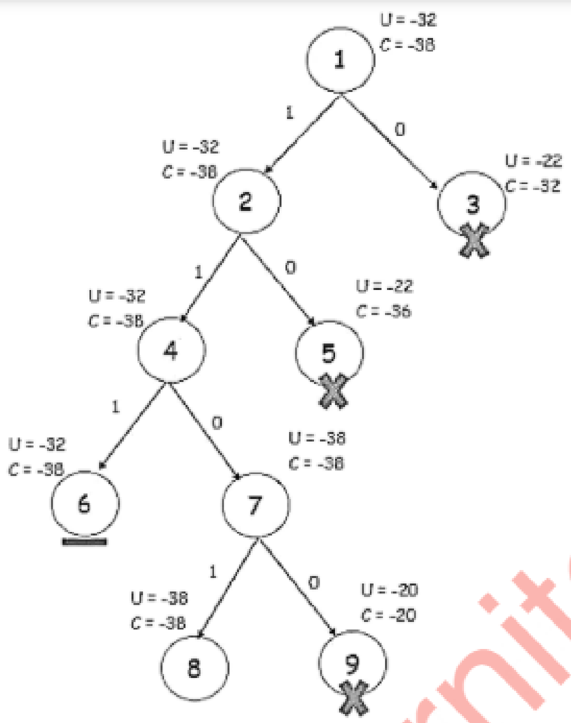
Here's an outline of how the Knapsack problem can be solved using the branch and bound algorithm:

1. **Initialization**: Start with an empty knapsack and set the initial upper bound to zero.
2. **Branching**: Choose an item to include or exclude in the knapsack. Branch into two sub problems:
    - Include the item: Reduce the knapsack capacity and update the current value.
    - Exclude the item: Proceed with the original capacity and value.
3. **Bounding**: At each node, compute a lower bound on the maximum possible value that can be achieved by considering the remaining items. This bound helps determine whether to explore a subtree further or prune it.
4. **Pruning**: If the lower bound of a node exceeds the current upper bound, prune the subtree rooted at that node.
5. **Backtracking**: Recursively explore the remaining branches until all possibilities are exhausted.
6. **Optimal Solution**: The optimal solution is the highest value encountered during the exploration.

TimeComplexity:$O(2^N)$
Auxiliary Space:O(N)



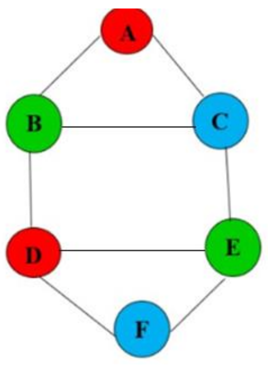
Solving an Example
Consider the problem with n =4, V = {10, 10, 12, 18}, w = {2, 4, 6, 9} and W = 15. Here, we calculate the initital upper bound to be U = 10 + 10 + 12 = 32. Note that the 4th object cannot be included here, since that would exceed W. For the cost, we add 3/9 th of the final value, and hence the cost function is 38. Remember to negate the values after calculation before comparison.
After calculating the cost at each node, kill nodes that do not need exploring. Hence, the final state space tree will be as follows (Here, the number of the node denotes the order in which the state space tree was explored):



Note here that node 3 and node 5 have been killed after updating U at node 7. Also, node 6 is not explored further, since adding any more weight exceeds the threshold. At the end, only nodes 6 and 8 remain. Since the value of U is less for node 8, we select this node. Hence the solution is {1, 1, 0, 1}, and we can see here that the total weight is exactly equal to the threshold value in this case.

7.With an example, apply and explain Graph Coloring Algorithm. (15 marks)

Graph coloring is a problem of coloring each vertex in graph in such a way that no two adjacent vertices have same color and yet m-colors are used. This problem is also called as **m-coloring problem**. If the degree of given graph is d then we can color it with d+1 colors. The least number of colors needed to color the graph is called its chromatic number. For example: Consider a graph,

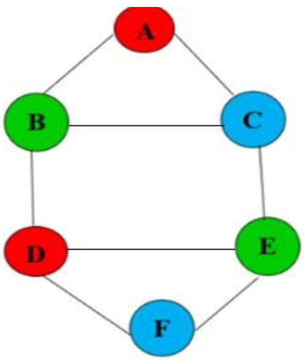


As given in the above graph we require three colors to color the graph. Hence the chromatic number of given graph is 3. We can use backtracking technique to solve the graph coloring problem as follows-

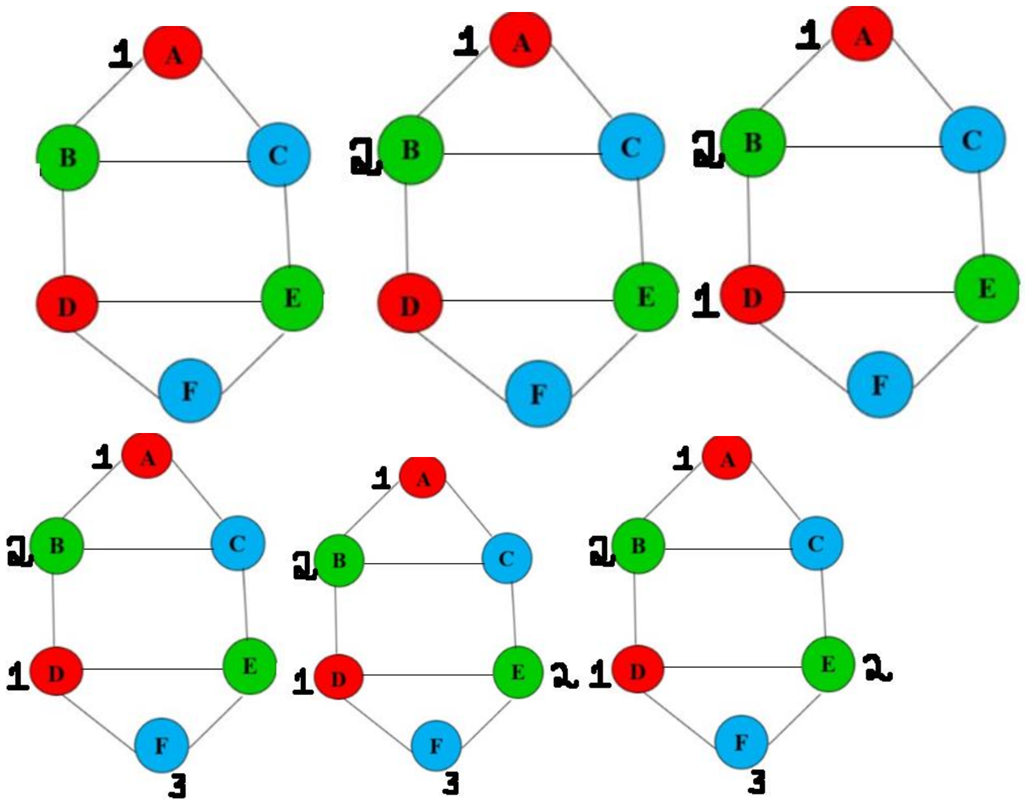
Step 1:

A Graph G consists of vertices from A to F.

There are three colors used Red, Green, Blue.
We will number them out. That means 1 indicates Red,
2 indicates Green and 3 indicates Blue color.



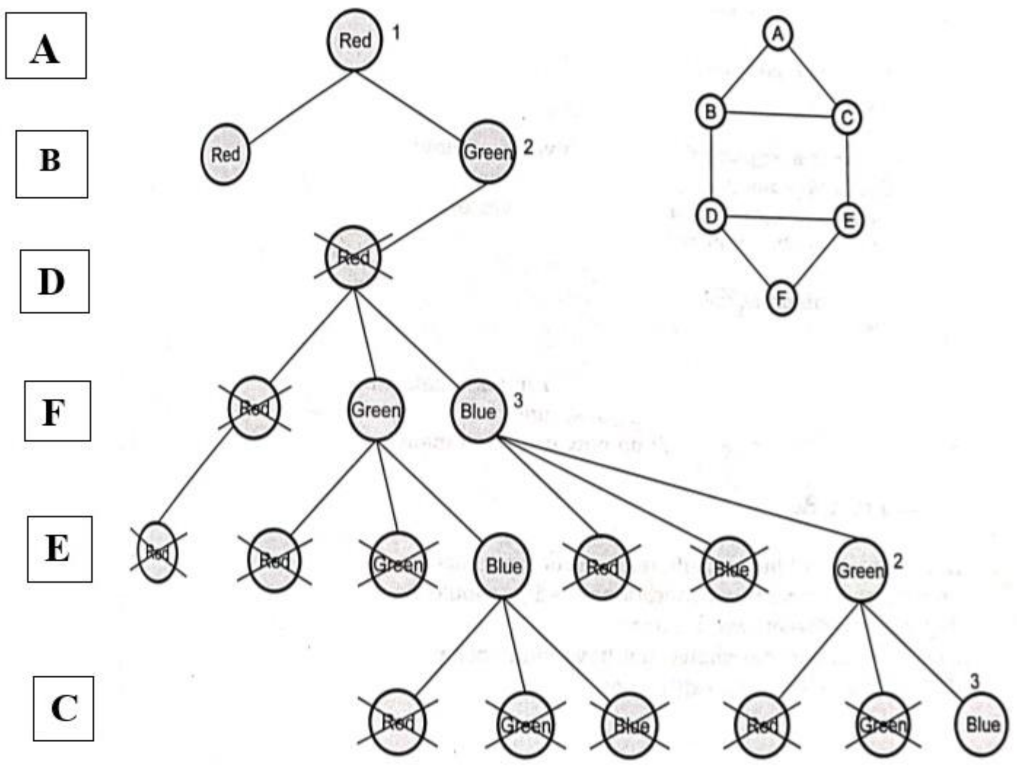
Step 2:



Stuck here!! Cannot assign 1 or 2 or 3.
Hence backtrack

Step 3:

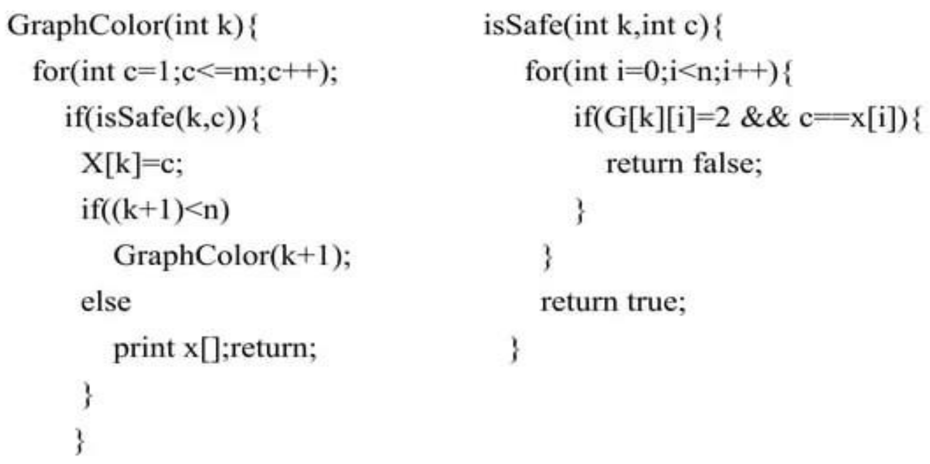
Thus the graph coloring problem is solved. The state-space tree can be drawn for better understanding of graph technique using backtracking approach.



Here we have assumed, color index Red=1, Green=2, and Blue=3.

Algorithm:

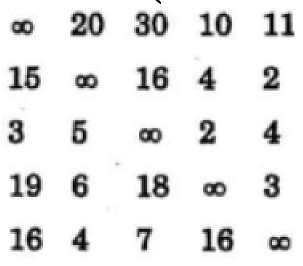
1. Create a recursive function that takes current index, number of vertices and output color array.
2. If the current index is equal to number of vertices. Check if the output color configuration is safe, i.e check if the adjacent vertices do not have same color. If the conditions are met, print the configuration and break.
3. Assign a color to a vertex (1 to m).
4. For every assigned color recursively call the function with next index and number of vertices
5. If any recursive function returns true break the loop and returns true.



```
GraphColor(int k){
  for(int c=1;c<=m;c++);
     if(isSafe(k,c)){
      X[k]=c;
      if((k+1)<n)
         GraphColor(k+1);
      else
         print x[];return;
      }
     }
```

```
isSafe(int k,int c){
     for(int i=0;i<n;i++){
         if(G[k][i]=2 && c==x[i]){
            return false;
         }
      }
      return true;
   }
```

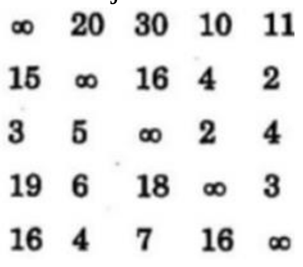
8.Consider the travelling salesperson instances defined by the following cost matrix. (15 marks)



∞	20	30	10	11
15	∞	16	4	2
3	5	∞	2	4
19	6	18	∞	3
16	4	7	16	∞

Apply the state space and show the reduced matrices corresponding to each of the node.

The adjacent matrix of the given problem is given by,



∞	20	30	10	11
15	∞	16	4	2
3	5	∞	2	4
19	6	18	∞	3
16	4	7	16	∞

As we can observe in the above adjacent matrix that 10 is the minimum value in the first row, 2 is the minimum value in the second row, 2 is the minimum value in the third row, 3 is the minimum value in the third row, 3 is the minimum value in the fourth row, and 4 is the minimum value in the fifth row.

Now, we will reduce the matrix. We will subtract the minimum value with all the elements of a row. First, we evaluate the first row. Let's assume two variables, i.e., i and j, where 'i' represents the rows, and 'j' represents the columns.

When i = 0, j =0

M[0][0] = ∞-10= ∞

When i = 0, j = 1

M[0][1] = 20 - 10 = 10

When i = 0, j = 2

M[0][2] = 30 - 10 = 20

When i = 0, j = 3

M[0][3] = 10 - 10 = 0

When i = 0, j = 4

M[0][4] = 11 - 10 = 1

The matrix is shown below after the evaluation of the first row:

$$\begin{pmatrix} \infty & 10 & 20 & 0 & 1 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{pmatrix}$$

Consider the second row.

When i = 1, j =0

M[1][0] = 15-2= 13

When i = 1, j = 1

M[1][1] = ∞ - 2= ∞

When i = 1, j = 2

M[1][2] = 16 - 2 = 14

When i = 1, j = 3

M[1][3] = 4 - 2 = 2

When i = 1, j = 4

M[1][4] = 2 - 2 = 0

The matrix is shown below after the evaluation of the second row:

$$\begin{pmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{pmatrix}$$

Consider the third row:

When i = 2, j =0

M[2][0] = 3-2= 1

When i = 2, j = 1

M[2][1] = 5 - 2= 3

When i = 2, j = 2

M[2][2] = ∞ - 2 = ∞

When i = 2, j = 3

M[2][3] = 2 - 2 = 0

When i = 2, j = 4

M[2][4] = 4 - 2 = 2

The matrix is shown below after the evaluation of the third row:

$$\begin{pmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{pmatrix}$$

Consider the fourth row:

When i = 3, j =0

M[3][0] = 19-3= 16

When i = 3, j = 1

M[3][1] = 6 - 3= 3

When i = 3, j = 2

M[3][2] = 18 - 3 = 15

When i = 3, j = 3

M[3][3] = ∞ - 3 = ∞

When i = 3, j = 4

M[3][4] = 3 - 3 = 0

The matrix is shown below after the evaluation of the fourth row:

$$\begin{pmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 16 & 4 & 7 & 16 & \infty \end{pmatrix}$$

Consider the fifth row:

When i = 4, j =0

M[4][0] = 16-4= 12

When i = 4, j = 1

M[4][1] = 4 - 4= 0

When i = 4, j = 2

M[4][2] = 7 - 4 = 3

When i = 4, j = 3

M[4][3] = 16 - 4 = 12

When i = 4, j = 4

M[4][4] = ∞ - 4 = ∞

The matrix is shown below after the evaluation of the fifth row:

$$\begin{pmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{pmatrix}$$

The above matrix is the reduced matrix with respect to the rows.

Now we reduce the matrix with respect to the columns. Before reducing the matrix, we first find the minimum value of all the columns. The minimum value of first column is 1, the minimum value of the second column is 0, the minimum value of the third column is 3, the minimum value of the fourth column is 0, and the minimum value of the fifth column is 0, as shown in the below matrix:

Now we reduce the matrix.

Consider the first column.

When i = 0, j =0

M[0][0] = ∞-1= ∞

When i = 1, j = 0

M[1][0] = 13 - 1= 12

When i = 2, j = 0

M[2][0] = 1 - 1 = 0

When i = 3, j = 0

M[3][0] = 16 - 1 = 15

When i = 4, j = 0

M[4][0] = 12 - 1 = 11

The matrix is shown below after the evaluation of the first column:

$$\begin{pmatrix} \infty & 10 & 20 & 0 & 1 \\ 12 & \infty & 14 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 15 & \infty & 0 \\ 11 & 0 & 3 & 12 & \infty \end{pmatrix}$$

Since the minimum value of the first and the third columns is non-zero, we will evaluate only first and third columns. We have evaluated the first column. Now we will evaluate the third column.

Consider the third column.

When i = 0, j =2

M[0][2] = 20-3= 17

When i = 1, j = 2

M[1][2] = 14 - 3= 11

When i = 2, j = 2

M[2][2] = ∞- 3 = ∞

When i = 3, j = 2

M[3][2] = 15 – 3 = 12

When i = 4, j = 2

M[4][2] = 3 - 3 = 0

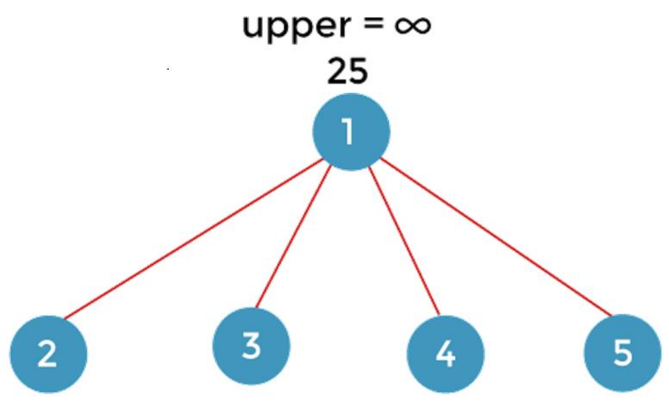
The matrix is shown below after the evaluation of the third column:

$$\begin{pmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{pmatrix}$$

The above is the reduced matrix. The minimum value of rows is (10+2+2+3+4=21), and the columns is (1+3=4). Therefore, the total minimum value is (21 + 4) equals to 25.

Let's understand that how to solve this problem using branch and bound with the help of a state-space tree.

To make a state-space tree, first, we consider node A. From node 1, we can go either to nodes 2, 3, 4, or 5 as shown in the below image. The cost of node 1 would be the cost which we achieved in the above-reduced matrix, i.e.., 25. Here, we will also maintain the upper bound. Initially, the upper bound would-be infinity.

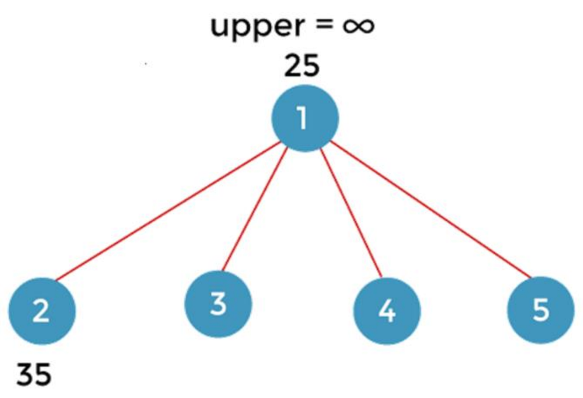


Now, consider node 2. It means that we are moving from node 1 to node 2. Make the first row and second column as infinity shown in the below matrix:

Once we move from node 1 to node 2, we cannot move back to node 1. Therefore, we have to make 2 to 1 as infinity shown in the below matrix:

$$\begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{pmatrix}$$

Since each row and column contains atleast one zero value; therefore, we can say that above matrix has been reduced. The cost of reduction of node 2 is c(1, 2) + cost of parent node + edge cost = 0 + 25 + 10 = 35.



Now we will find the minimum value of each column of the new reduced matrix. The minimum value of the first column is 11 and the minimum value of other three columns is 0.

Now, consider the node 3. It means that we are moving from the node 1 to node 3. Make the first row and third column as infinity shown in the below matrix:

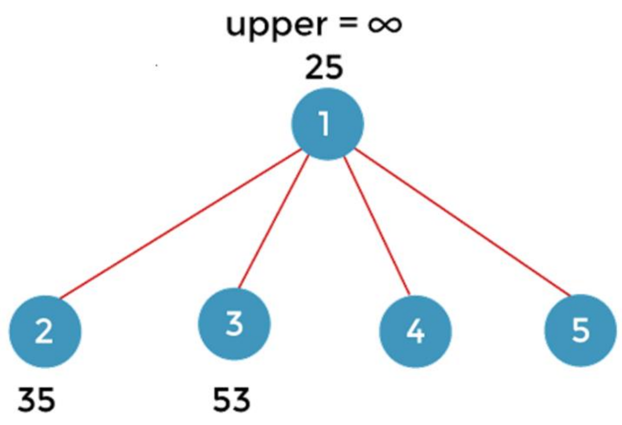
$$\begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 15 & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & 12 & \infty \end{pmatrix}$$

Since each row and column contains one non zero value; therefore, we can say that above matrix has not been reduced.

So the minimum value in the first column is 11. Now we have to subtract each element in the first column with 11.

$$\begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{pmatrix}$$

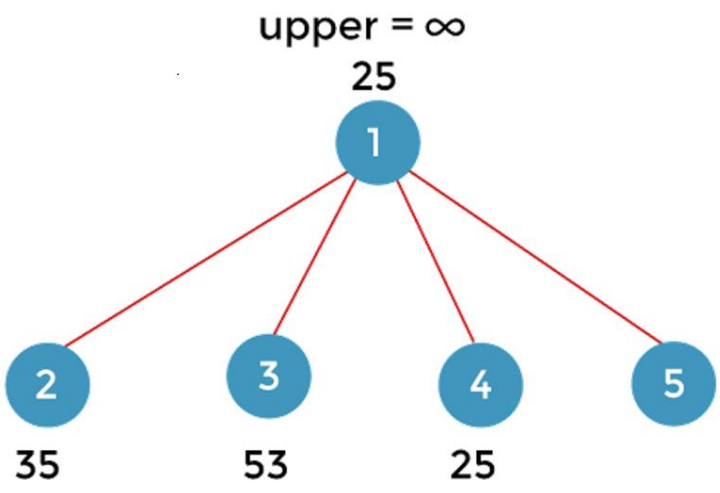
Now each row and column contains atleast one zero value; therefore, we can say that above matrix has been reduced. The cost of reduction of node 3 is c(1, 3) + cost of parent node + edge cost = 11 + 25 + 17 = 53.



Now, consider the node 4. It means that we are moving from the node 1 to node 4. Make the first row and forth column as infinity shown in the below matrix:

$$\begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{pmatrix}$$

Since each row and column contains atleast one zero value; therefore, we can say that above matrix has been reduced. The cost of reduction of node 4 is c(1, 4) + cost of parent node + edge cost = 25 + 0 + 0 = 25.



Now, consider the node 5. It means that we are moving from the node 1 to node 5. Make the first row and fifth column as infinity shown in the below matrix:

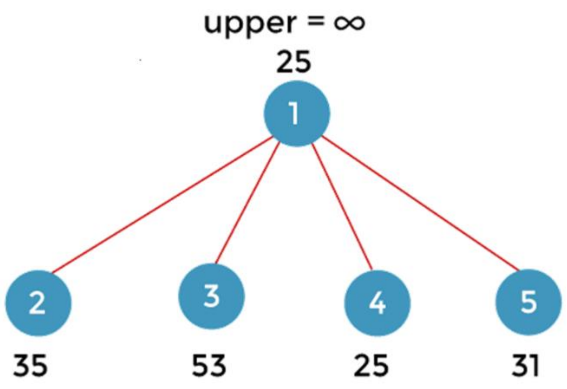
$$\begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & 2 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 15 & 3 & 12 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{pmatrix}$$

Since each row and column contains one non zero value; therefore, we can say that above matrix has not been reduced.

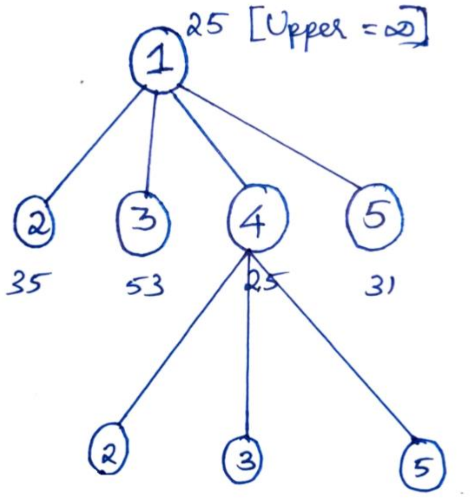
So the minimum value in the second row is 2 and minimum value in fourth row is 3. Now we have to subtract each element in the second row with 2 and fourth row with 3.

$$\begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{pmatrix}$$

Now each row and column contains atleast one zero value; therefore, we can say that above matrix has been reduced. The cost of reduction of node 5 is c(1, 5) + cost of parent node + edge cost = 5 + 25 + 1 = 31.



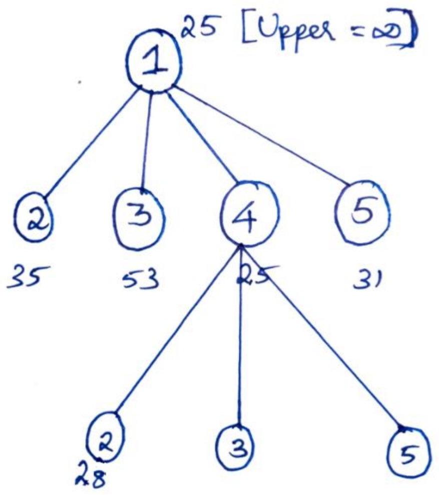
Since the node 4 has the minimum cost, i.e., 25. So we will explore the node 4 first. From the vertex 4, we can go either to the vertex 2, 3 or 5 as shown in the below image:



First, we consider the path from the vertex 4 to the vertex 2. We make fourth row as ∞ and second column as ∞. Since we cannot move back from 2 to 1, so 1 to 2 is also infinity as shown in the below matrix:

$$\begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & 0 & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{pmatrix}$$

Since all the rows and columns have atleast one zero value. Therefore, we can say that this matrix is already reduced. So, there would be no reduction cost. The cost of reduction of node 2 is c(4, 2) + cost of parent node + edge cost = 0 + 25 + 3 = 28



Now we have to calculate the cost of the path from the vertex 4 to the vertex 3. We make fourth row and third column as infinity as shown in the below matrix. Since we cannot move from the vertex 3 to 1, so we make 3 to 1 as infinity shown in the below matrix:

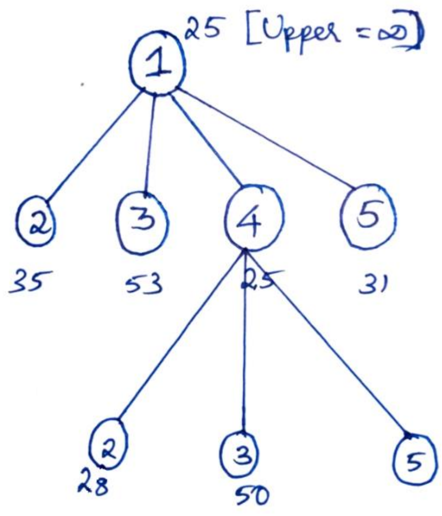
$$\begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & \infty & 0 \\ \infty & 3 & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & 0 & \infty & \infty & \infty \end{pmatrix}$$

Since each row and column contains one non zero value; therefore, we can say that above matrix has not been reduced.

So the minimum value in the third row is 2 and minimum value in first column is 11. Now we have to subtract each element in the third row with 2 and first column with 11.

$$\begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & \infty & \infty & \infty \end{pmatrix}$$

Now each row and column contains atleast one zero value; therefore, we can say that above matrix has been reduced. The cost of reduction of node 3 is c(4, 3) + cost of parent node + edge cost = 13 + 25 + 12 = 50.



Now we will calculate the cost of the path from the vertex 4 to 5. We make fourth row and fifth column as infinity. Since we cannot move back from the node 5 to 1, so we also make 1 to 5 as infinity shown in the below matrix:

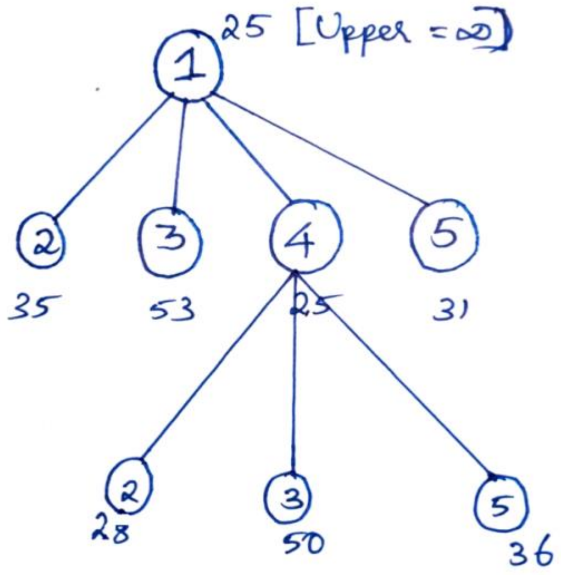
$$\begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty \end{pmatrix}$$

Since each row and column contains one non zero value; therefore, we can say that above matrix has not been reduced.

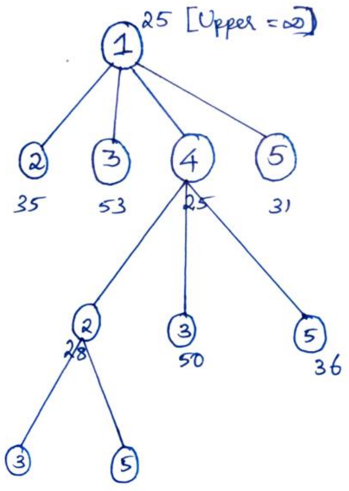
So the minimum value in the second row is 11. Now we have to subtract each element in the second row with 11.

$$\begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty \end{pmatrix}$$

Now each row and column contains atleast one zero value; therefore, we can say that above matrix has been reduced. The cost of reduction of node 5 is c(4, 5) + cost of parent node + edge cost = 11 + 25 + 0 = 36.



Now we will compare the cost of all the leaf nodes. The node with a cost 28 is minimum so we will explore this node. The node with a cost 28 can be further expanded to the nodes 3 and 5 as shown in the below figure:



Now we have to calculate the cost of both the nodes, i.e., 3 and 5. First we consider the path from node 2 to node 3.

We make second row and third column as infinity. Also, we cannot move back from the node 3 to node 1 so we make 3 to 1 as infinity as shown in the below matrix:

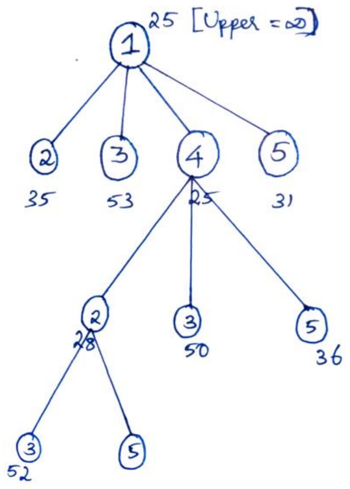
$$\begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & \infty & \infty & \infty \end{pmatrix}$$

Since each row and column contains one non zero value; therefore, we can say that above matrix has not been reduced.

So the minimum value in the third row is 2 and fifth row is 11. Now we have to subtract each element in the second row with 2 and fifth row with 11.

$$\begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \end{pmatrix}$$

Now each row and column contains atleast one zero value; therefore, we can say that above matrix has been reduced. The cost of reduction of node 3 is c(2, 3) + cost of parent node + edge cost = 13 + 28 + 11 = 52.

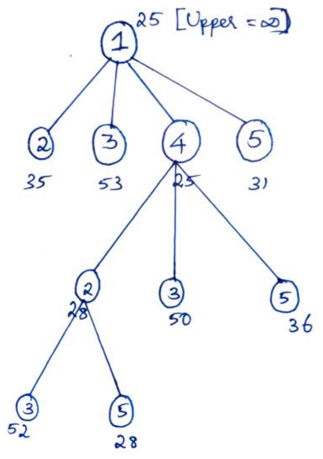


Since we cannot move back from the node 5 to node 1 so make 1 to 5 also as infinity as shown in the below matrix Consider the path from node 2 to node 5. Make the fourth row and third column as infinity.

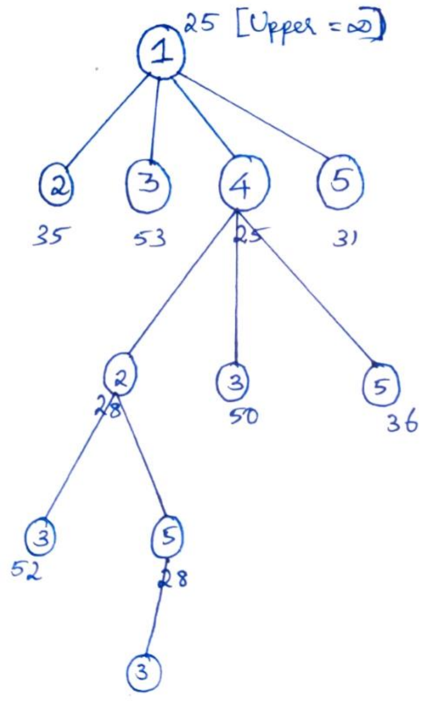
$$\begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{pmatrix}$$

Since every row and column contains a zero value; therefore, the above matrix is the reduced matrix.

The cost of reduction of node 5 is c(2, 5) + cost of parent node + edge cost = 0 + 28 + 0 = 28.



Now we will find out the leaf node with a minimum cost. The node 5 with a cost 28 is minimum so we select node 5 for the further exploration. The node 5 can be further expanded to the node 3 as shown in the below figure:

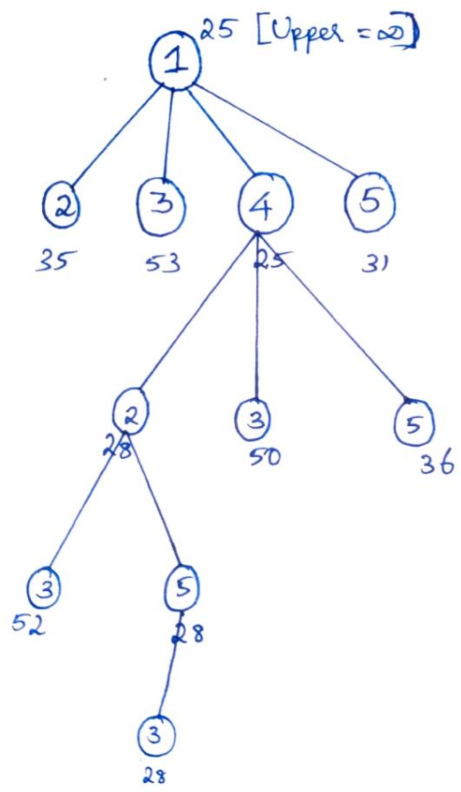


Consider the path from node 5 to node 3. Make the fifth row and third column as infinity. Since we cannot move back from the node 3 to node 1 so make 1 to 5 also as infinity as shown in the below matrix:

$$\begin{pmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{pmatrix}$$

Since every row and column contains a zero value; therefore, the above matrix is the reduced matrix.

The cost of reduction of node 3 is c(5, 3) + cost of parent node + edge cost = 0 + 28 + 0 = 28.



Finally, we traverse all the nodes. The upper value is updated from infinity to 28. We will check whether any leaf node has a value less than 28. Since no leaf node contains the value less than 28.

The path of the tour would be 1->4->2->5->3.