UNIT - IV

Backtracking: General Method, 8-Queens Problem, Sum of Subsets problem, Graph Coloring, 0/1 Knapsack Problem.

Branch and Bound: The General Method, 0/1 Knapsack Problem, Travelling Salesperson problem.

Backtracking General Method:

Backtracking was first introduced by H.J.Lehmer in 1950s. Backtracking is a general algorithmic technique that involves searching through all possible combinations to solve a problem by trying one option at a time. When a solution is found to be invalid or not optimal, the algorithm *backtracks* and tries the next option. This is particularly useful in solving constraint satisfaction problems, optimization problems, and decision problems.

Backtracking is used for solving problems where you need to explore possibilities one by one and decide whether to move forward or discard certain paths based on specific constraints. It is closely related to the idea of depth-first search (DFS) in tree or graph traversal.

Steps:

1. **Start from an empty solution**.
2. **Add a component** to the current solution and check:
    - If the solution is complete and valid, stop.
    - If not, try the next component recursively.
3. **If adding a component does not lead to a solution**, remove it (backtrack) and try the next component.
4. **Repeat** until you either find a valid solution or exhaust all possibilities.

Backtracking method requires that all the solutions satisfy a complex set of constrains. For any problem these constraints are divided into two categories. They are;

- **Explicit constraints** depend on the particular instance I of the problem being solved. All tuples that satisfy the explicit constraints define a possible solution space I.
- **Implicit constraints** are rules that determine which of the tuples in the solution space I satisfies the criterion function.

Applications:

- **N-Queens Problem**
- **Sum of Subsets Problem**
- **Graph Coloring Problem**
- **Hamiltonian Cycles**

n-Queens Problem:

The N queen problem is the problem of placing N chess queen on an NxN chessboard. So that no two queens are on the same row, same column or diagonal. The solution space consists of n! permutations of n-tuple.

Algorithm:

1. Place the queens column wise, start from the left most column
2. If all queens are placed.
    1. Return true and print the solution matrix.
3. Else
    1. Try all the rows in the current column.
    2. Check if queen can be placed here safely if yes mark the current cell in solution matrix as 1 and try to solve the rest of the problem recursively.
    3. If placing the queen in above step leads to the solution return true.
    4. If placing the queen in above step does not lead to the solution, BACKTRACK, mark the current cell in solution matrix as 0 and return false.
4. If all the rows are tried and nothing worked, return false and print NO SOLUTION.

4 Queens Problem:

It consists of 4 queens that should be placed on a 4 x 4 chess board. So that no two queens attack i.e. no two queens are on same row or same column or diagonal.

Algorithm: n-Queen problem Algorithm

➔ The First queen is placed in the first row and first column. So that the second queen should not by on 1st row or 1s column or diagonal.

Q1	*	*	*
*	*		
*			
*			

➔ Q2 should not be placed in 1st row or 1st column or 2nd column of the 2nd row. To it will be placed in 3rd column of the 2nd row.

Q1	*	*	*
*	*	Q2	*
*	*	*	*
*		*	

➔ We are unable to place Q3 in 3rd row so go back to Q2 and place it in 4th column.

Q1	*	*	*
*	*	*	Q2
*		*	*
*			*

Q1	*	*	*
*	*	*	Q2
*	Q3	*	*
*	*	*	*

➔ We are unable to place Q4 in 4th row because of diagonal attack from Q3, so go back and remove Q3 and place it in the next column. But it is not possible, so move back to Q2 and remove it to next column but it is not possible. So go back to Q1 and move it to next column.

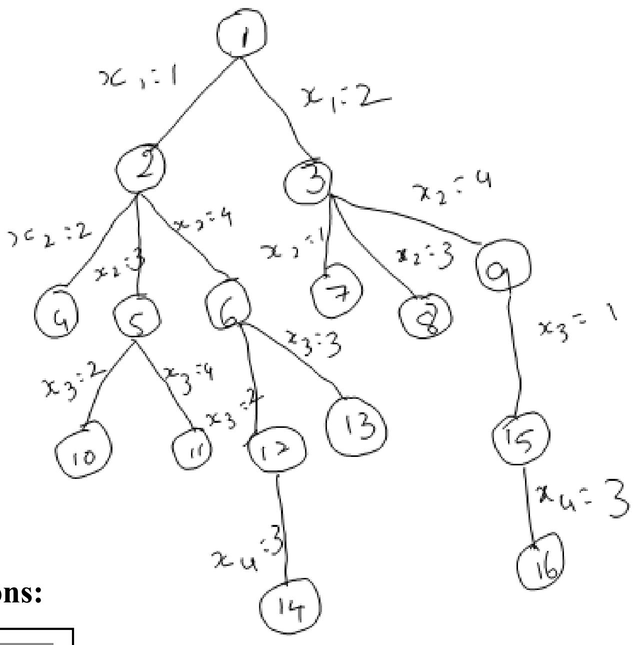
	Q1		

	Q1		
			Q2

	Q1		
			Q2
Q3			

	Q1		
			Q2
Q3			
		Q4	

The solution to the 4- Queen's problem is $x_1 = 2$, $x_2 = 4$, $x_3 = 1$ and $x_4 = 3$. The state space tree for the above problem is;



Other possible Solutions:

✖	✖	♛	✖
♛	✖	✖	✖
✖	✖	✖	♛
✖	♛	✖	✖

$x_1 = 3$, $x_2 = 1$, $x_3 = 4$ and $x_4 = 2$

8 Queen Problem:

It consists of 8 queens that should be placed on a 8 x 8 chess board. So that no two queens attack i.e. no two queens are on same row or same column or diagonal.

Place (k, i) algorithm returns a Boolean value that is true if the k^{th} queen is in i^{th} column. Let A[1:8, 1:8] chess board is represented as a 8x8 matrix. The queens are numbered 1 through 8.

Explicit Constraint: The explicit constraint specifies that 8 queens are to be placed in 8 x 8 chess board in 8^8 ways. The solution space is reduced by applying implicit constraints.

Implicit Constraint: Implicit constraints specify that no two queens are in the same row, or column or diagonal.

Suppose two queens are placed at positions (i, j) and (k, l) then they are on the same diagonal if and only if $|j - l| = | i - k|$.

Time Complexity: O(8!)

Algorithm:

```
Algorithm Place (k,i)
{
        for j :=1 to k-1 do
        if (x[i] – i)  or abs(x[j]-i) = abs(k-k)) then
                return false;
}

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

Possible Solutions:

		Q1					
					Q2		
	Q3						
						Q4	
Q5							
			Q6				
							Q7
				Q8			

1. $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (3, 6, 2, 7, 1, 4, 8, 5)$
2. $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (4, 6, 8, 2, 7, 1, 3, 5)$
3. $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (4, 7, 3, 8, 2, 5, 1, 6)$
4. $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (5, 2, 4, 7, 3, 8, 6, 1)$
5. $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (4, 2, 7, 3, 6, 8, 5, 1)$
6. $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (4, 6, 8, 3, 1, 7, 5, 2)$
7. $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (3, 6, 8, 1, 4, 7, 5, 2)$
8. $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (5, 3, 8, 4, 7, 1, 6, 2)$
9. $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (5, 7, 7, 1, 3, 8, 6, 2)$
10. $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (4, 1, 5, 8, 6, 3, 7, 2)$
11. $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (3, 6, 4, 1, 8, 5, 7, 2)$
12. $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (6, 2, 7, 1, 4, 8, 5, 3)$
13. $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (4, 7, 1, 8, 5, 2, 6, 3)$
14. $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) =(6, 4, 7, 1, 8, 2, 5, 3)$
15. $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (3, 5, 2, 8, 1, 7, 4, 6)$

Sum of Subsets Problem:

The sum of subsets problem is to find a subset of given set S= {S1, S2, ….., Sn} of 'n' positive integers whose sum of elements is equal to a given positive integer 'm'.

Recursive algorithm for sum of subsets:

```
Algorithm SumOfSub(s, k, r)
{
        x[k]:=1;
        if (s+w[k] = m) then eite (x[1:k]);
        if((s+r-w[k]>=m) and (s+w[k+1]<=m) then
        {
                x[k] := 0;
                SumOfSub(s, k+1, r-w[k]);
        }
}
```

Ex:

S={1, 2, 5, 6, 8}

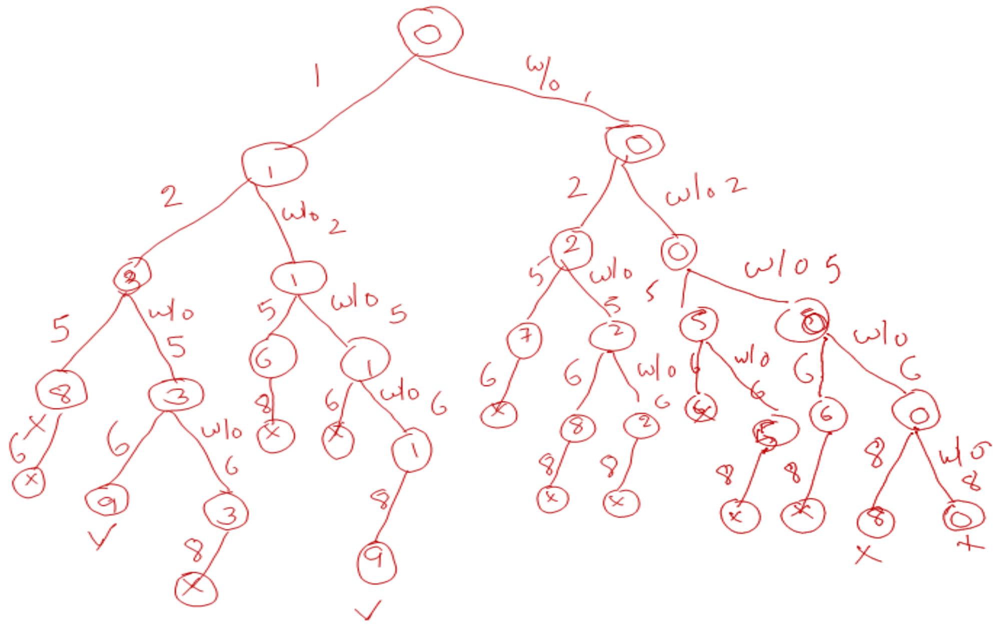
Find a subset of set S where sum of elements is 9.

Subset	**-**	**Sum**	
{1}	-	1	
{1, 2}	-	3	
{1, 2, 5}	-	8	
{1, 2, 5, 6}	-	14	**Sum exceeds 9 hence back track**
{1, 2, 6}	-	9	

State Space Tree: It will be drawn just like a binary tree.

1. The root node is set to 0. Because initially no elements are added to the subset.
2. Arrange the elements in increasing order.

   1, 2, 5, 6, 8
3. Check the sum of sub set at each time by including the element or excluding the element. If the element is included then go to left. Otherwise go to right.
4. If the sum > given sum then stop adding and backtrack. Continue this process until the solution is found.



Exercise:

1. **Consider a set S= {5, 10, 12, 13, 15, 18} and d= 30. Solve it for obtaining sum of subset.**
2. **{5, 7, 10, 12, 15, 18, 20} and m= 35.**
3. **{3, 2, 7, 1} and m=6.**
4. **{2, 4, 6, 8, 10} and m=20.**
5. **{7, 11, 13, 24} and m = 31**

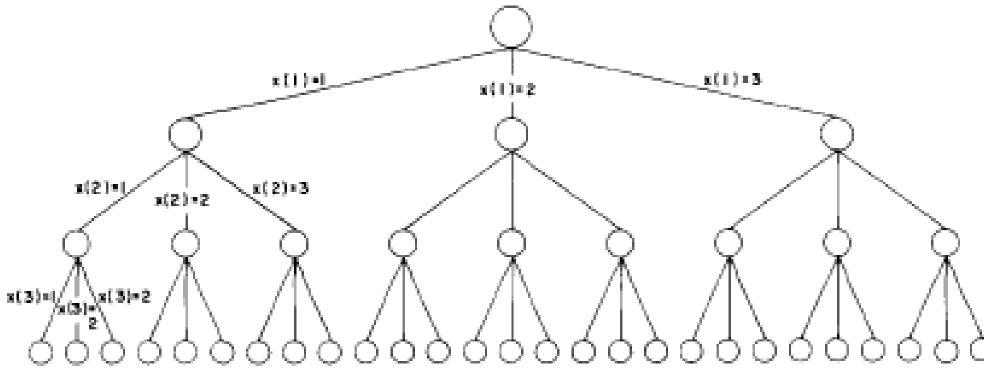
Graph Coloring:

Let G be a graph and m be a given positive integer. Graph coloring is a problem of coloring the nodes of graph G in such a way no two adjacent nodes have the same color yet only m colors are used. 'd' is the degree of the graph then d+1 colors are used to color a graph. According to m-colorability optimization the smallest integer 'm' can be used for coloring a graph G.

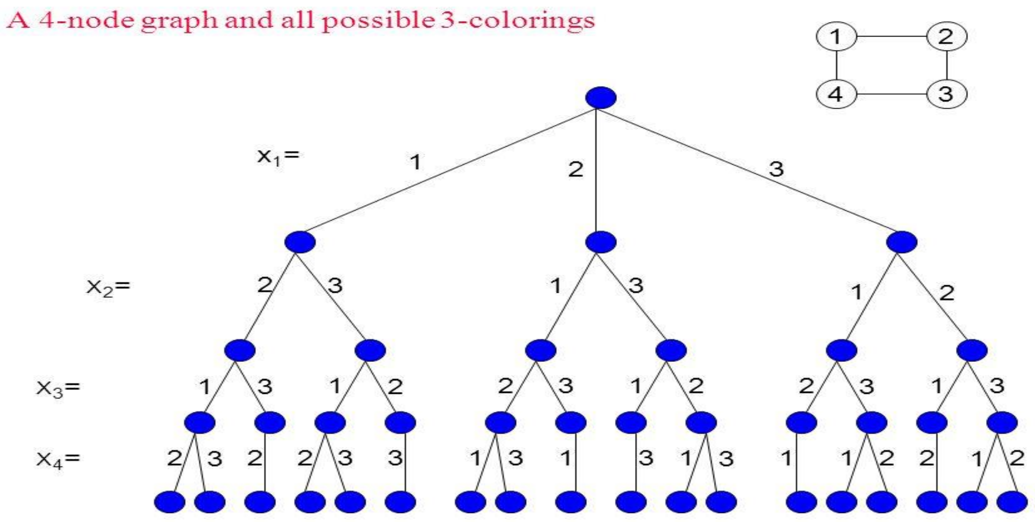
Chromatic Number of a graph defines the number of colors that are used in coloring a graph.

Time Complexity : O(n.m^n)

State Space Tree for Graph-Coloring when n=3 and m=3



Ex:



There are 6 solutions to the 3-Coloing problem.

Node	Solutions					
	I	II	III	IV	V	VI
1	1	1	2	2	3	3
2	2	3	1	3	2	1
3	3	2	3	1	1	2
4	2	3	1	3	2	1

Exercise:

Draw the search tree to color the graph with the three colors: red, blue, green.

0/1 Knapsack Problem:

The **0/1 Knapsack Problem** is a classic problem in optimization where you are given a set of items, each with a weight and value, and a knapsack that can carry a limited weight. The goal is to determine the maximum profit that can be obtained by selecting a subset of items such that the total weight of the selected items does not exceed the knapsack's capacity.

In the **0/1 Knapsack Problem**, each item can either be included in the knapsack or excluded.

$$\Sigma_{i = 1 \text{ to } k} w_i x_i \leq M \text{ and}$$
$$\Sigma_{i = 1 \text{ to } k} p_i x_i \text{ is maximizd}$$

Ex:

P1, P2, P3= 1, 2, 5

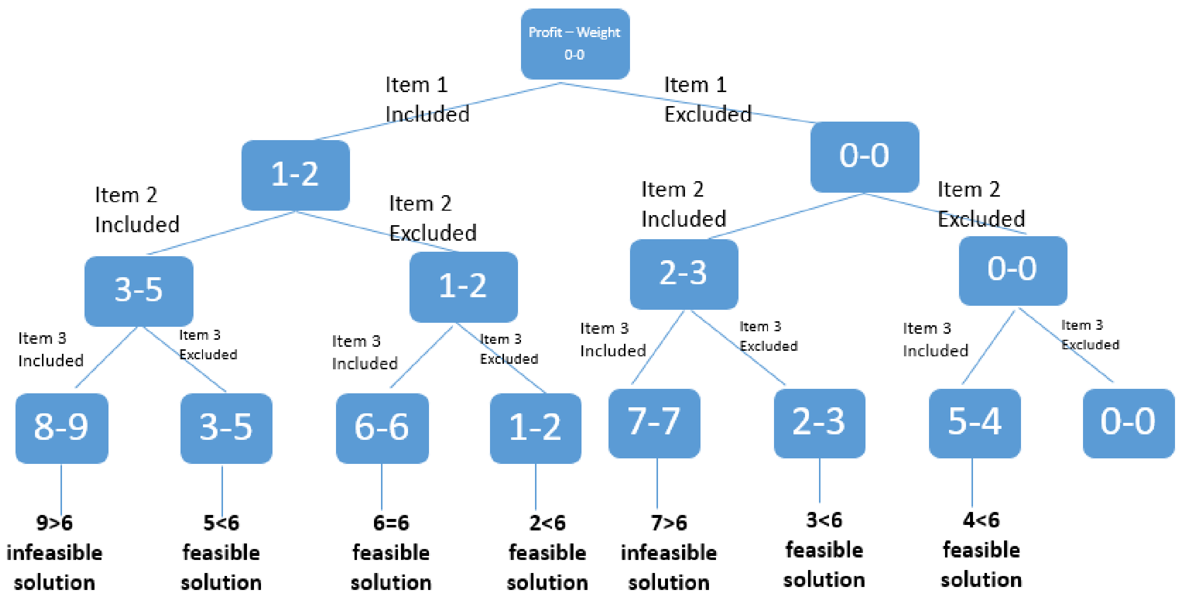
W1, W2, W3= 2, 3, 4

M=6

Profit – Weight

Initially we are not added any item so, profit 0 and weight 0.

State Space Tree:



The optimum solution is:

X1=1 X2 = 0 and X3 = 0

X1 Profit = 1 and Weight = 2

X3 Profit = 5 and Weight = 4

Total Profit = 6

Branch and Bound:

Live node:

A node that has not been expanded is called as live node.

E-node:

E-node is a live node currently being explored and its children are being generated.

Dead Node:

A dead node is either not to be explored further or all of whose children have already been explored.

General Method:

Brach and Bound is one of the algorithmic techniques used to solve optimization problems. Branch and Bound systematically explores the solution space by breaking the problem into smaller subproblems and uses a bounding function for finding optimal solution.

Branch and bound is generally used for optimization problems whereas backtracking is used for decision problems. Branch and bound is applicable for minimization problems.

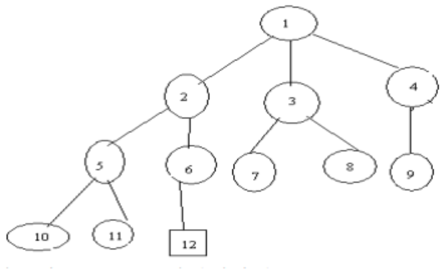
A branch and bound method searches a state space tree using any search mechanism in which all the children of the E-node are generated before another node becomes the E-node. Three common search strategies are used in branch and bound. They are;

1. FIFO (First In First Out Search) Branch and Bound Search
2. LIFO (Last In First Out Search) Branch and Bound Search
3. LC (Least Cost) Branch and Bound Search

FIFO (First In First Out Search) Branch and Bound Search:

It uses queue data structure and follows BFS Technique.

Example:



Assume that the answer node is 12.

In FIFO, search starts from root node i.e. from 1 as E-node. Next generate the children of node 1 as live nodes in the queue as follows;

2	3	4		

Now we will delete the first element from the queue and generate children of node 2 and place in the queue.

	3	4	5	6

Next, delete an element from queue and take it as E-node, generate the children of node 3, 7, 8 are children of 3 and these live nodes are killed by bounding functions. So we will not include in the queue.

		4	5	6

Next, delete an element from queue and take it as E-node, generate the children of node 4, 9 is the children of 4 and these live nodes are killed by bounding functions. So we will not include in the queue

			5	6

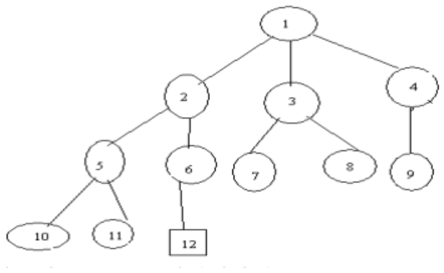
Next, delete an element from queue and take it as E-node, generate the children of node 5, 10 and 11 are the children of 5 and these live nodes are killed by bounding functions. So we will not include in the queue

				6

The child of node 6 is 12 and it satisfies the conditions of the problem, which is the answer node, so search terminates.

LIFO (Last In First Out Search) Branch and Bound Search:

It uses the stack data structure and follows DFS Technique.



Assume that the answer node is 12.

In LIFO, search starts from root node i.e. from 1 as E-node. Next generate the children of node 1 as live nodes in the stack as follows;

2
3
4

Remove element from stack and generate the children of it, place those nodes into stack. 2 is removed from stack. The children of 2 are 5, 6. The content of stack is;

5
6
3
4

Again remove an element from stack, i.,e node 5 is removed and nodes generated by 5 are 10, 11 which are killed by bounded function, so we will not place 10, 11 into stack.

6
3
4

Delete an element from stack, i.,e node 6. Generate child of node 6, i.,e 12, which is the answer node, so search process terminates.

LC (Least Cost) Branch and Bound Search:

LC algorithm is an intelligent search algorithm. This is also known as the best-first search technique. The selection rule for the next E-node in FIFO or LIFO branch and bound is sometimes "blind". i.e., the selection rule does not give any preference to a node that has a very good chance of getting the search to an answer node quickly. Search time in LC algorithm is reduced when compared to FIFO or LIFO. This algorithm uses a cost function, which varies from problem to problem. This method uses a rank or priority to all the nodes based on the cost function. Using this technique, the rank of a node is calculated as;

$$\mathbf{g\hat{}=h(X)+g(X)}$$

Here, h(x) – cost of the reaching node x from the root

g(x) – cost of moving from node x to the solution state

g^(x) – estimate of the real rank

Because of the exact cost cannot be determined, the minimum cost of this estimate is said to be least cost. The search technique uses a ranking function is called LC search with bounded function. The algorithm is as follows;

Step 1:

Generate a child for E-node.

Step 2:

Select the node with least cost function as the next E-node. For this it requires priority queue instead of normal queue.

0/1 Knapsack Problem:

The branch and bound technique can be applied to the knapsack problem. The knapsack problem is about filling a knapsack with ‘n’ items. Each item ‘i’ is associated with a profit Pi and weight Wi. The objective function is to achieve the maximization profit subjected to the constraint of the capacity of the knapsack. Branch and bound technique will not be directly applied to the knapsack problem because it deals only with the minimization problem. For this the problem will be modified as

$$\text{Minimization of } Z = -P_1X_1 - P_2X_2 - P_3X_3 - \ldots.. - P_nX_n$$

$$\text{Subject to } W_1X_1 + W_2X_2 + \ldots.. + W_nX_n <= m$$

$$X_i = 0 \text{ or } 1$$

C^(X) and U^(X) are two cost functions such that C^(x) <= C(X) <= U^(X) and satisfying requirements. C(X) = - $\sum \boldsymbol{PiXi}$

Where, C(X) is the cost function for the answer node x, and which lies between two functions called lower and upper bounds for the cost function C(X).

The search begins at the root node. Initially we compute lower and upper bounds at the root node. These are C^(1) and U^(1).

C(X) = min { (C(lchild(X)), (C(Rchild(X)) }

C(1) = min { C^(2), C^(3) }

Ex:

N = 5

(P1, P2, P3, P4 P5) = (10, 15, 6, 8, 4)

(W1, W2, W3, W4, W5) =(4, 6, 3, 4, 2)

M = 12

Ans:

1. Arrange all the items as per their profit/weight ratio.
2. Convert the profits into negative values i.e. -10, -15, -6, -8, -4.
3. Calculate lower and upper bounds for each node.
4. Place the first item in the bag, remaining weight is 12 – 4 = 8.
5. Place the second item in the bag, remaining weight is 8 – 6 = 2.
6. Since fractions are not allowed we cannot place third and fourth items in the bag.
7. Place the fifth item in the bag, remaining weight is 2 – 2 = 0.

Upper bound (Profit Earned) = -10-15-4 = -29.

8. To calculate the lower bound, place third item in the bag, since the fractions are allowed.

   Lower bound = -10-15-(2/3 * 6) = -10-15-4 = -29.

   So U^(1) = -29 and C^(1) = -29

For Node 2: X1 = 1 i.e. we should place first item in the bag;

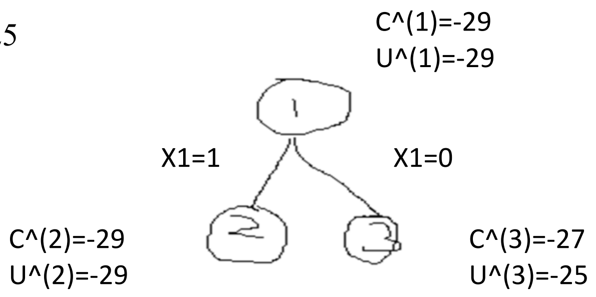
C^(2) = -10-15-(2/3 * 6) = - 29

U^(2) = -10-15-4 = -29

For Node 3: X1 = 0 i.e. we should not place first item in the bag;

C^(3) = -15-6-(3/4 * 8) = - 27

U^(3) = -15-6-4 = -25



Select the minimum lower bound;

C(x1) = min { (C^(2), C^(3) }

= min { -29, -27}

= -29

So, Node 2 is selected. i.e X1 =1

For Node 4: X2 = 1 i.e. we should place second item in the bag;

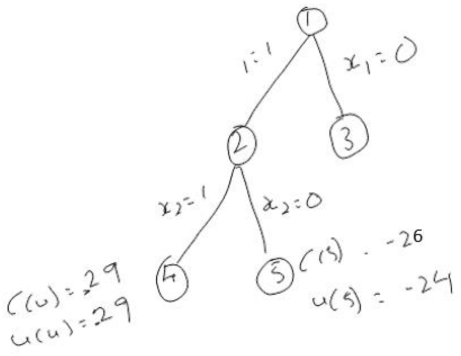
C^(4) = -10-15-(2/3 * 6) = - 29

U^(4) = -10-15-4 = -29

For Node 5: X2 = 0 i.e. we should not place second item in the bag;

C^(5) = -10-6-(1/2 * 4) = - 26

U^(5) = -10-6-8 = -24



Select the minimum lower bound;

C(x2) = min { (C^(4), C^(5) }

= min { -29, -26}

= -29

So, Node 4 is selected. i.e X2 =1

For Node 5: X3 = 1 i.e. we should place third item in the bag;

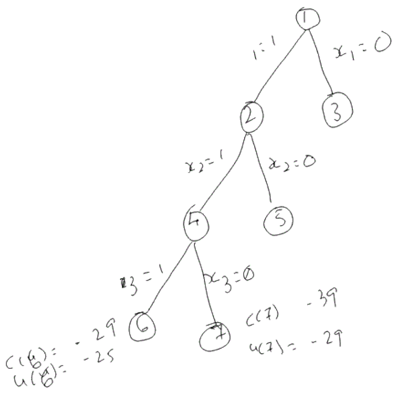
C^(6) = -10-15-(2/3 * 6) = - 29

U^(6) = -10-15 = -25

For Node 6: X3 = 0 i.e. we should not place third item in the bag;

C^(7) = -10-15-(2/4 * 8) = - 29

U^(7) = -10-15-4 = -20



Since lower bounds are same take minimum upper bound.

C(x3) = min { (U^(6), U^(7) }

= min { -25, -29}

= -29

So, Node 7 is selected. i.e X3 =0

For Node 8: X4 = 1 i.e. we should place fourth item in the bag;

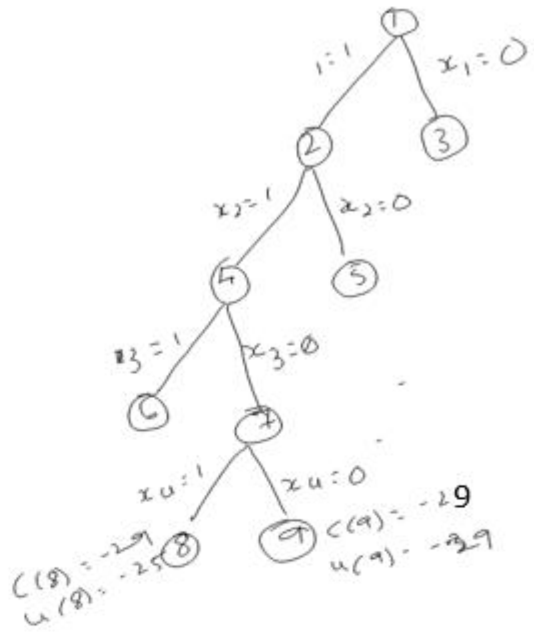
C^(8) = -10-15-(2/4 * 8) = - 29

U^(8) = -10-15 = -25

For Node 9: X1 = 0 i.e. we should not place fourth item in the bag;

C^(9) = -10-15-4 = - 29

U^(9) = -10-15-4 = -29



Select the minimum lower bound;

C(x4) = min { (U^(8), U^(9) }

= min { -25, -29}

= -29

So, Node 9 is selected. i.e X4 =0

For Node 10: X5 = 1 i.e. we should place fifth item in the bag;

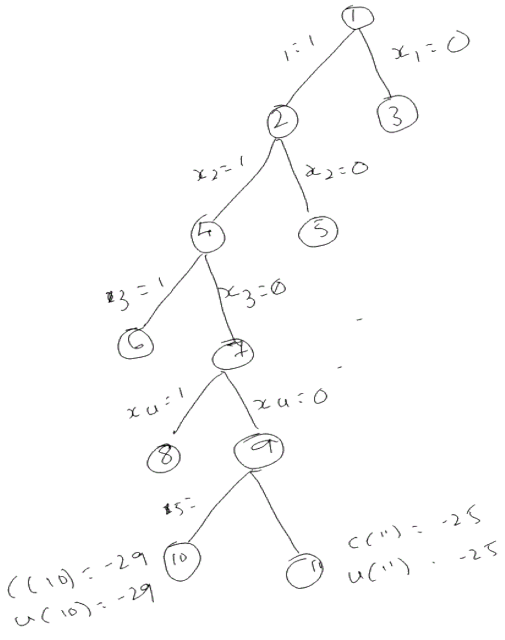
C^(10) = -10-15-4 = -29

U^(10) = -10-15-4 = -29

For Node 11: X5 = 0 i.e. we should not place fifth item in the bag;

C^(11) = -10-15 = - 25

U^(12) = -10-15 = -25



Select the minimum lower bound;

C(x5) = min { C^(10), C^(11) }

= min { -29, -25}

= -29

So, Node 10 is selected. i.e X5 =1

Path is 1-2-4-7-9-10

(x1, x2, x3, x4, x5) = (1, 1, 0 , 0, 1)

Maximum Profit = 10 + 15 + 4 = 29

Exercise:

1. **N = 4, (P1, P2, P3, P4) = (10, 10, 12, 18), (W1, W2, W3, W4) = (2, 4, 6, 9) AND M = 15**

Knapsack Problem Using FIFO Branch And Bound Solution:

Ex:

N=4, (P1, P2, P3, P4) = (10, 10, 12, 18)

(W1, W2, W3, w4)= (2, 4, 6, 9), W= 15.

The FIFOBB algorithm proceeds with node 1 as the root node and makes it E-node.

U(1)= -10-10-12 = -32

C(1) = -10-10-12-(3/9*18) = -38

The child nodes are generated only if upper bound of the E-node <= upper bound (1) otherwise the E-node becomes dead node.

FIFOBB initializes upper bound to -32 and hence nodes 2 and 3 are produced by node 1 and 2 and 3 are sent to the queue.

2	**3**			

Next '2' becomes E-node and it produces node 4 and 5 are as children. Therefore node 4 and 5 are queued and hence the queue is as follows;

3	**4**	**5**		

C(2)= -10-10-12-(3/9*18) = - 38

U(2)=-10-10-12 = -32 (Which is <=U1))

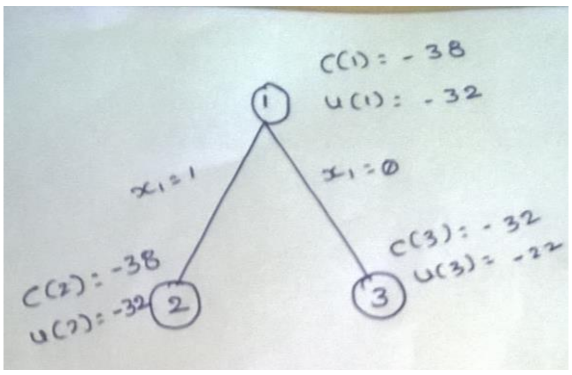
And node 3 is the next E-node and then node 6 and 7 are generated.

C(3)= -10-12-(5/9*18) = - 32

U(3)=-10-12 = -22 (Which is > U(1))

Node 3 cannot be expanded because U(3) > U(1).

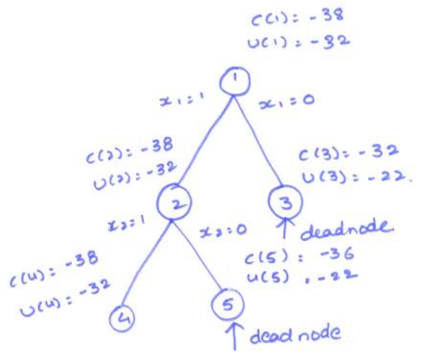
4	**5**			



C(4) = -10-10-12-(3/9*18) = -38

U(4) = -10-10-12= -32 (Which is <= U(1))

5	**6**	**7**		



C(5)= -10-12-(7/9*18)=-36

U(5)=-10-12=-22 (Which is > U(1))

6	**7**			

Node 5 cannot be expanded because U(5) > U(1).

C(6) =-10-10-12-(3/9*18) =-38

U(6) = -10-12 = 32 (Which is <= U(1))

7	**8**	**9**		

C(7)=-10-10-18=-38

U(7)=-10-10-18=-38 (Which is < U(1)

8	**9**	**10**	**11**	

C(8) = -10-10-12-(3/9*18) = - 38

U(8) = -10-10-12=-32 (Can't include item 4 so it becomes infeasible node)

9	**10**	**11**		

C(9) = -10-10-12=- 32

U(9) = -10-10-12=-32 (Which is < U(1))

10	**11**			

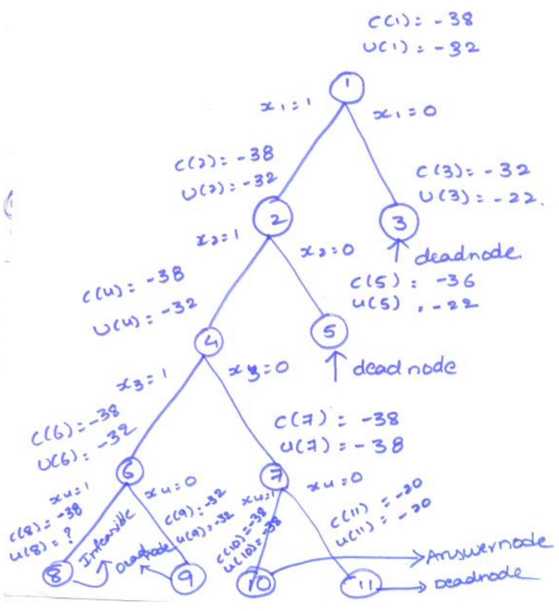
C(10)= -10-10-18) = - 38

U(10) = -10-10-18=-38 (Which is < U(1))

11				

C(11)= -10-10 = -20

U(11) = -10-10 = -20 (Which is > U(1))



Node 10 and node 11 satisfies the condition but the maximum profit is obtained by taking node 10.

(x1, x2, x3, x4, x5) = (1, 1, 0, 1)

Maximum Profit = 10+10+18 = 38

Exercise:

1. **N = 4, (P1, P2, P3, P4) = (10, 10, 12, 18), (W1, W2, W3, W4) = (2, 4, 6, 9) AND M = 15**

Travelling Salesman Problem:

The idea behind in TSP is find minimum weight route which starts from a vertex and then travels through every other vertex exactly once and reaches the starting vertex. Let G = (V, E) be a directed graph defining an instance of the travelling salesman problem. Let C_{ij} be the cost of the edge (i, j) and $C_{ij} = \alpha$ if $(i,j) ℂ\ E(G)$ and let $| V | = n$.

The following three bounds are associated with the problem. They are;

1. The bound c(i) represents the length of the path. If it is a leaf node, c(i) represents the length from the root to the given node i. If it is a non-leaf node then it is the minimum cost in the state space tree.
2. The bound l(i) represents the lower bound that indicates the length of the path. It is obtained from the reduced cost matrix. For finding the lower bound of i the distance matrix should be reduced. A matrix is said to be reduced if every row and column of the matrix is reduced.

   To reduce the distance matrix, subtract a constant 'p' from every entry. Choose the minimum value of every row and column. Repeat the process as required. The total amount subtracted from the rows and columns is a lower bond l for the root.
3. The upper bound u(i) represents the upper bound for the node i. The upper bound for all nodes is d. i.e. it is the maximum possible effort required for the tour.

Algorithm:

1. Let 'C' be the reduced cost matrix, and let us assume that the edge (i, j) is include in the tour. Change all the entries of ith row and jth column of the distance matrix to ∞. This is to include all the paths leaving the vertex i and reaching vertex j.
2. Set $C(i, l) = \infty$, this is to exclude all (i, l) paths.
3. Reduce the distance matrix except for the rows and columns that contains ∞.
4. Compute the lower bound as follows;

   $L(B) = l(A) + C(i, j) + T$

   Where,

   A → parent of the node B in the state space matrix.

   T → Sum of all elements involved in row and column reduction.

If the node is leaf node, then the lower bound l is the cost itself.

Ex:

$$\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & \infty & 4 & 2 \\ 2 & 3 & \infty & 4 \\ 3 & 1 & 8 & \infty \end{array}$$

First find the lower bound of the given matrix. For this first find the row minimum i.e. 2, 3 and 1. Subtract the row minimum from the respective rows.

After Row Reduction:

$$\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & \infty & 2 & 0 \\ 2 & 0 & \infty & 1 \\ 3 & 0 & 7 & \infty \end{array}$$

Next subtract the column minimum i.e. 0, 2 and 0 from the respective columns.

After Column Reduction:

$$\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & \infty & 0 & 0 \\ 2 & 0 & \infty & 1 \\ 3 & 0 & 5 & \infty \end{array}$$

Total Reduction = 2 + 3 + 1 + 0 + 2 + 0 = 8.

So the lower bound of 1 is 8.

It is the minimum cost required for any tour. Next we need to compute the reduced cost matrix for the node (1, 2).

Path (1, 2):

Set all entries of row 1 and column 2 to ∞. And set A(2, 1) to ∞.

$$\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & \infty & \infty & \infty \\ 2 & \infty & \infty & 1 \\ 3 & 0 & \infty & \infty \end{array}$$

After Row Reduction

$$\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & \infty & \infty & \infty \\ 2 & \infty & \infty & 0 \\ 3 & 0 & \infty & \infty \end{array}$$

After Column Reduction :

$$\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & \infty & \infty & \infty \\ 2 & \infty & \infty & 0 \\ 3 & 0 & \infty & \infty \end{array}$$

Total Reduction = 1 + 0 = 1

C(2) = l(1) + A(1, 2) + r

= 8 + 0 + 1 = 9

So lower bound of 2 is 9.

Path (1, 3):

Set all entries of row 1 and column 3 to ∞. And set A(3, 1) to ∞.

$$\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & \infty & \infty & \infty \\ 2 & 0 & \infty & \infty \\ 3 & \infty & 5 & \infty \end{array}$$

After Row Reduction

$$\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & \infty & \infty & \infty \\ 2 & 0 & \infty & \infty \\ 3 & \infty & 0 & \infty \end{array}$$

After Column Reduction

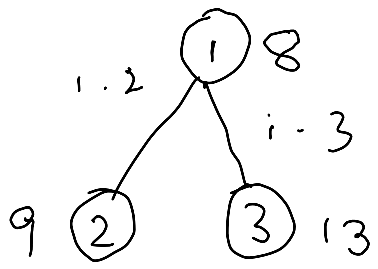
$$\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & \infty & \infty & \infty \\ 2 & 0 & \infty & \infty \\ 3 & \infty & 0 & \infty \end{array}$$

Total Reduction = 5 + 0 = 5

C(3) = l(1) + A(1, 3) + r

= 8 + 0 + 5 = 13

So lower bound of 3 is 13.



The lower bound of 2 is minimum than lower bound of 3. So the path is 1-2. Next find the path from 2 to 3. For finding path from 2 to 3 take the reduced cost matrix of the node (1, 2). Change 2nd row and 3rd column to ∞ and change (3, 2) to ∞.

	1	2	3
1	∞	∞	∞
2	∞	∞	0
3	0	∞	∞

After Row Reduction

	1	2	3
1	∞	∞	∞
2	∞	∞	0
3	0	∞	∞

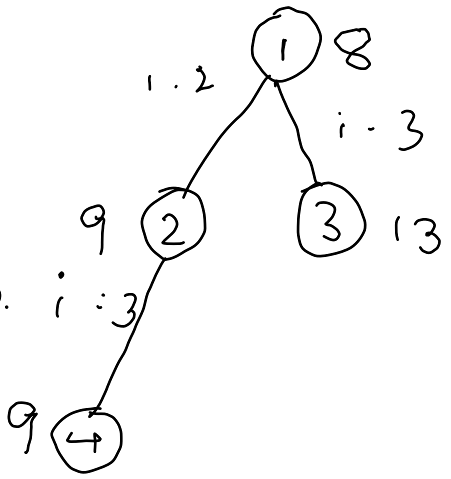
After Column Reduction

	1	2	3
1	∞	∞	∞
2	∞	∞	0
3	0	∞	∞

C(3) = l(2) + A(2, 3) + r

= 9 + 0 + 0 = 9.

So the optimal path = 1-2-3-1 and the cost is 4+4+1 = 9.



Exercise:

	1	2	3	4	5
1	∞	20	30	10	11
2	15	∞	16	4	2
3	3	5	∞	2	4
4	19	6	18	∞	3
5	16	4	7	16	∞

Ans: Path = 1-4-2-5-3-1 and Minimum Cost is 28.

	1	2	3	4	5
1	∞	7	3	12	8
2	3	∞	6	14	9
3	5	8	∞	6	18
4	9	3	5	∞	11
5	18	14	9	8	∞

Ans: Path = 1-5-3-4-2-1 and Minimum Cost is 29.

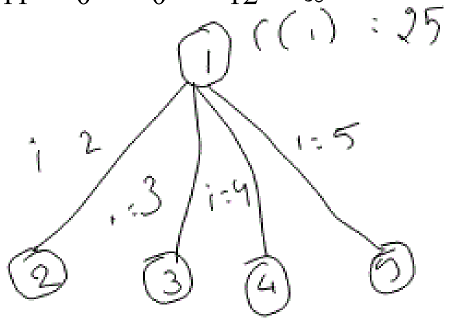
	1	2	3	4	5
1	∞	20	30	10	11
2	15	∞	16	4	2
3	3	5	∞	2	4
4	19	6	18	∞	3
5	16	4	7	16	∞

After Row Reduction:

	1	2	3	4	5
1	∞	10	20	0	1
2	13	∞	14	2	0
3	1	3	∞	0	2
4	16	3	15	∞	0
5	12	0	3	12	∞

After Column Reduction:

	1	2	3	4	5
1	∞	10	17	0	1
2	12	∞	11	2	0
3	0	3	∞	0	2
4	15	3	12	∞	0
5	11	0	0	12	∞



Total Reduction C(1) =

10+2+2+3+4+1+0+3+0+0 = 25

Path (1,2): 1st row - ∞, 2nd Column - ∞ and (2,1) - ∞

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	2	0
3	0	∞	∞	0	2
4	15	∞	12	∞	0
5	11	∞	0	12	∞

After Row Reduction:

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	2	0
3	0	∞	∞	0	2
4	15	∞	12	∞	0
5	11	∞	0	12	∞

After Column Reduction:

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	2	0
3	0	∞	∞	0	2
4	15	∞	12	∞	0
5	11	∞	0	12	∞

R= 0

C(2) = C(1) +A(1,2) + R

= 25+10+0

= 35

Path (1,3): 1st row - ∞, 3rd Column - ∞ and (3,1) - ∞

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	∞	2	0
3	∞	3	∞	0	2
4	15	3	∞	∞	0
5	11	0	∞	12	∞

After Row Reduction:

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	∞	2	0
3	∞	3	∞	0	2
4	15	3	∞	∞	0
5	11	0	∞	12	∞

After Column Reduction:

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	1	∞	∞	2	0
3	∞	3	∞	0	2
4	4	3	∞	∞	0
5	0	0	∞	12	∞

R= 11

C(3) = C(1) +A(1,3) + R

= 25+17+11

= 53

Path (1,4): 1st row - ∞, 4th Column - ∞ and (4,1) - ∞

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	11	∞	0
3	0	3	∞	∞	2
4	∞	3	12	∞	0
5	11	0	0	∞	∞

After Row Reduction:

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	11	∞	0
3	0	3	∞	∞	2
4	∞	3	12	∞	0
5	11	0	0	∞	∞

After Column Reduction:

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	11	∞	0
3	0	3	∞	∞	2
4	∞	3	12	∞	0
5	11	0	0	∞	∞

R= 0

C(4) = C(1) +A(1,4) + R

= 25+0+0

= 25

Path (1,5): 1st row - ∞, 5th Column - ∞ and (5,1) - ∞

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	11	2	∞
3	0	3	∞	0	∞
4	15	3	12	∞	∞
5	∞	0	0	12	∞

After Row Reduction:

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	10	∞	9	0	∞
3	0	3	∞	0	∞
4	12	0	9	∞	∞
5	∞	0	0	12	∞

After Column Reduction:

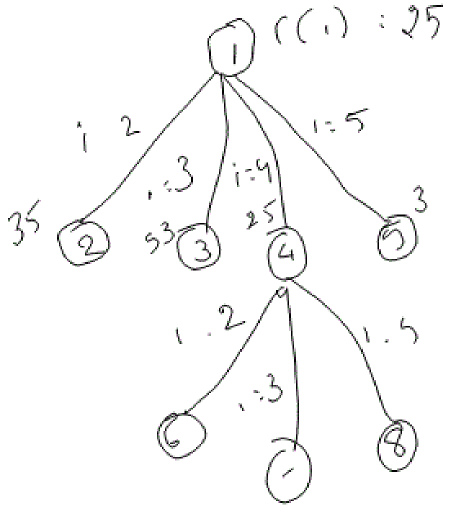
	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	10	∞	9	0	∞
3	0	3	∞	0	∞
4	12	0	9	∞	∞
5	∞	0	0	12	∞

R= 5

C(5) = C(1) +A(1,5) + R

= 25+1+5

= 31



Reduced cost matrix of (1,4)=

1	∞	∞	∞	∞	∞
2	12	∞	11	∞	0
3	0	3	∞	∞	2
4	∞	3	12	∞	0
5	11	0	0	∞	∞

Path (4,2) : 4th row - ∞, 2nd Column - ∞ and (2,1) - ∞

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	∞	0
3	0	∞	∞	∞	2
4	∞	∞	∞	∞	∞
5	11	∞	0	∞	∞

After Row Reduction:

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	∞	0
3	0	∞	∞	∞	2
4	∞	∞	∞	∞	∞
5	11	∞	0	∞	∞

After Column Reduction:

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	∞	0
3	0	∞	∞	∞	2
4	∞	∞	∞	∞	∞
5	11	∞	0	∞	∞

R= 0

C(2) = C(4) +A(4,2) + R

= 25+3+0

= 28

Path (4,3) : 4th row - ∞, 3rd Column - ∞ and (3,1) - ∞

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	∞	∞	0
3	∞	3	∞	∞	2
4	∞	∞	∞	∞	∞
5	11	0	∞	∞	∞

After Row Reduction:

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	∞	∞	0
3	∞	1	∞	∞	0
4	∞	∞	∞	∞	∞
5	11	0	∞	∞	∞

After Column Reduction:

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	1	∞	∞	∞	0
3	∞	3	∞	∞	2
4	∞	∞	∞	∞	∞
5	0	0	∞	∞	∞

R= 13

C(3) = C(4) +A(4,3) + R

= 25+12+13

= 50

Path (4,5) : 4th row - ∞, 5th Column - ∞ and (5,1) - ∞.

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	12	∞	11	∞	∞
3	0	3	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	0	0	∞	∞

After Row Reduction:

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	1	∞	0	∞	∞
3	0	3	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	0	0	∞	∞

After Column Reduction:

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	1	∞	0	∞	∞
3	0	3	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	0	0	∞	∞

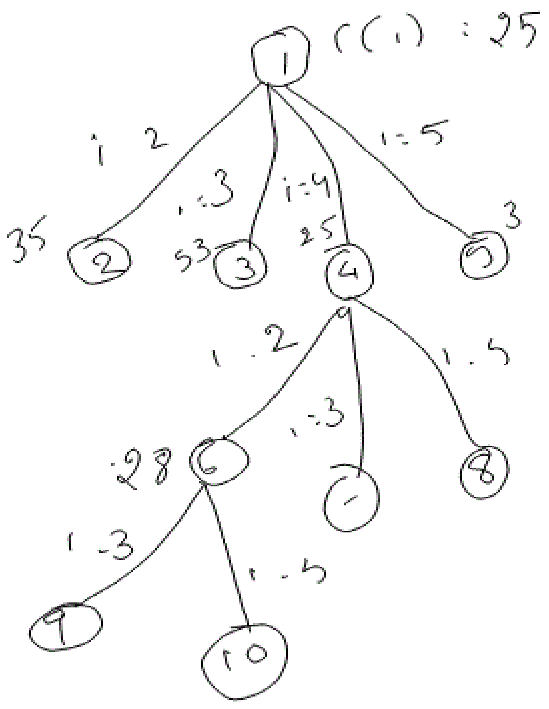
R=11

C(5) = C(4) +A(4,5) + R

= 25+0+11

= 36

Since minimum cost is 28. Select node 2.



Reduced Cost Matrix of (2,3) is :

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	11	∞	0
3	0	∞	∞	∞	2
4	∞	∞	∞	∞	∞
5	11	∞	0	∞	∞

Path (2,3) : 2nd row - ∞, 3rd Column - ∞ and (3,1) - ∞.

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	2
4	∞	∞	∞	∞	∞
5	11	∞	∞	∞	∞

After Row Reduction:

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	0
4	∞	∞	∞	∞	∞
5	0	∞	∞	∞	∞

After Column Reduction:

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	0
4	∞	∞	∞	∞	∞
5	0	∞	∞	∞	∞

R=13

C(3) = C(2) +A(2,3) + R

= 28+11+13

= 52

Path (2,5) : 2nd row - ∞, 5th Column - ∞ and (5,1) - ∞.

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	0	∞	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	∞	0	∞	∞

After Row Reduction:

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	0	∞	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	∞	0	∞	∞

After Column Reduction:

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	0	∞	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	∞	0	∞	∞

R=0

C(5) = C(2) +A(2,5) + R

= 28+0+0

= 28

Since the cost of path (2,5) will be taken as reduced cost matrix.

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	0	∞	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	∞	0	∞	∞

Path (5, 3) : 5th row - ∞, 3rd Column - ∞ and (3,1) - ∞.

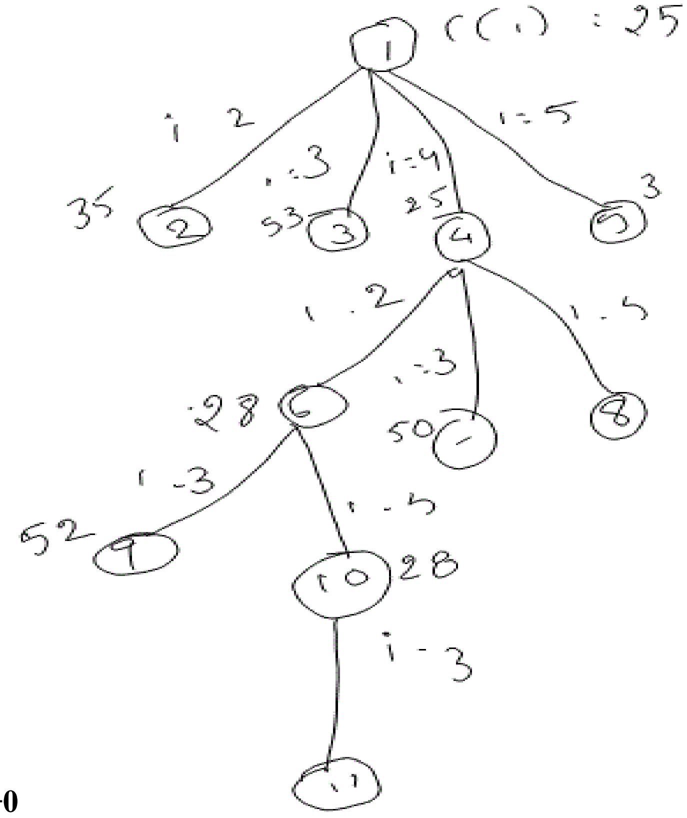
	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	∞

After Row Reduction:

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	∞

After Column Reduction:

	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	∞



C(3) **= C(5) +A(5,3)+R=28+0+0**

=28

PATH = 1-4-2-5-3-1

Cost = 10+6+2+7+3

= 28

Short Answer Questions

1. What is the main principle behind backtracking approach?
2. How does backtracking differ from brute force approach?
3. What is the objective of 8 queen problem.
4. What is the time complexity of 8queen problem?
5. What is the sum of subsets problem?
6. What is the difference between optimization problem and decision problem in view of graph coloring?
7. What is the objective of graph coloring? And also write its time complexity.
8. What is the main principle behind branch and bound technique?
9. What is the significance of bounding function in brach and bound?
10. What is the objective of TSP?
11. What are various common strategies used in branch and bound.

Essay Questions

1. Explain about General Method for backtracking.
2. Write an algorithm for how Eight Queen's problem can be solved using back tracking and explain with an example.
3. Solve 0/1 knapsack problem using backtracking technique. P={11,21,31,33,43,53,55,65}, w={1,11,21,23,33,43,45,55}, m=110. Generate the nodes using static tree and dynamic tree.
4. Explain about sum of subsets problem. The set S={1, 2, 5, 6, 8}. Find a subset of set S where sum of elements is 9. and also draw the state space tree.
5. (A) Describe Backtracking technique to m-coloring graph. Explain with example.
   (B) Draw the state space tree for graph coloring when n=3 and m=3.
6. Generate FIFO branch and bound solution for the given knapsack problem, m=15, n=3, (P1, P2, P3) = (10, 6, 8) and (w1, w2, w3) = (10, 12, 13).
7. Explain FIFO BB 0/1 knapsack problem procedure with the knapsack instance for n=4, (P1, P2, P3, P4) = (10, 10, 12, 18), (w1, w2, w3, w4) = (2, 4, 6, 9), and m = 15.
8. Draw the portion of the state space tree generated by LC branch and bound of knapsack problem for an instance n=4, (P1, P2, P3, P4) = (10, 10, 12, 18), (w1, w2, w3, w4) = (2, 4, 6, 9), and m = 15.
9. Describe the TSP and discuss how to solve it using branch and bound?