UNIT-V

Back Tracking: the General method, applications - n queen problem, Sum of Subsets problem, graph coloring, Hamiltonian Cycles.

Branch and Bound: the General method; applications - Travelling Sales person problem, 0/1 Knapsack problem - LC Branch and Bound solution, FIFO Branch and Bound solution.

BackTracking - General method :-

→ BackTracking is one of the important algorithm design technique.

→ Many problems which deal with searching for a set of solutions (or) ask for an optimal solution satisfying some constraints can be solved using Backtracking.

→ All the solutions solved using Backtracking must satisfy two constraints.

(a) Explicit Constraints
(b) Implicit Constraints

→ Explicit constraints are the rules which restrict each x_i to take on values from a given set.

Ex:- ① $x_i \geq 0$ (or) $S_i = \{$ set of all non-negative real numbers $\}$

② $x_i = 0$ (or) $x_i = 1$ (or) $S_i = \{0, 1\}$

→ Explicit constraints depends on the particular instance I of the problem being solved. All the tuples that satisfy explicit constraints defines the possible solution space for "I"

→ The implicit constraints are the rules that determine which of the tuples in the solution space of I satisfies the criterion function.

→ If the solution is expressible as an n-tuple $(x_1, x_2, \ldots, x_n)$ where x_i are chosen from some finite set, then we can apply Backtracking

→ Generally Backtracking solves three problems.

① Enumeration problem: In Enumeration problem, we solve all the possible feasible solutions

② Decision problem: we find whether any feasible solution (or) not.

③ optimization problem:- In optimization problem, we find whether there exist any best solution.

→ state space tree: In backtracking, while solving a given problem, a tree is constructed based on the choices made. Such a tree with all possible solutions is called a state space tree.

→ Each state space tree contains some nodes.

① Live node:- A node which has been generated, But its children have not yet been generated is called a "live node".

② E-node:- A live node whose children are currently being generated is called a E-node.

③ dead node: A node which is already expanded and there is no use for future

→ Bounding functions are used to kill live nodes without generating its children.

⇒ Depth first node generation with Bounding functions is called Backtracking

⇒ there are two types of nodes

① promising node

② Non promising node.

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

In otherwords if we expand the node, if it gets the complete solution, then it is called a promising node.

⇒ A node in a state space tree is said to be non-promising, if it can not lead to a complete solution.

⇒ Each node in the state space tree defines a problem state. All paths from root node to leaf nodes (or) other nodes defines a solution space.

⇒ Solution states are the problem states S for which the path from the root to S defines a tuple in the solution space.

⇒ Answer states are those solution states S for which the path from root to S defines a tuple which is a member of the set of solutions of the given problem

Recursive Backtracking algorithm

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

```
Algorithm I Backtrack(k)
// This algorithm describes the backtracking process
// All solutions are generated in a[1:n] and printed
// as soon as they are determined.
{
    k = 1
    while (k ≠ 0) do
    {
        if (there remains an untried
            a[k] ∈ T(a[1], a[2], ..... , a[k-1]) and
            B_k(a[1], a[2], ..... , a[k]) is true)
        then
        {
            if (a[1], ..... , a[k] is a path to an
                answer node) then
                write (a[1:k])
            k = k + 1 ; // Consider the next set
        }
        else
            k = k - 1 ; // Back to the previous set
    }
}
```

Algorithm SumOfSub (s, k, r)

// Find all subsets of $w[1:n]$ that sum to m. The values
// of $x[j]$, $1 \le j < k$, have already been determined.
// $s = \sum_{j=1}^{k-1} w[j] * x[j]$ and $r = \sum_{j=k}^{n} w[j]$.
// The $w[i]$'s are in nondecreasing order. It is
// assumed that $w[1] \le m$ and $\sum_{i=1}^{n} w[i] \ge m$.

{

// Generate left child.

$x[k] = 1$;

if $(s + w[k] = m)$ then write $(x[1:k])$;

else if $(s + w[k] + w[k+1] \le m)$

then SumOfSub $(s + w[k], k+1, r - w[k])$;

// Generate right child and evaluate B_k

if $((s + r - w[k] \ge m)$ and $(s + w[k+1] \le m))$ then

{

$x[k] = 0$;

SumOfSub $(s, k+1, r - w[k])$;

}

}

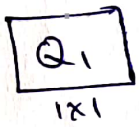
N - Queens problem in Backtracking

Consider n x n chess board. Now n queens are to be placed on an n x n chess board so that no two attack.

i.e: no two are on the same row
no two are on the same column.
no two are on the same diagonal.

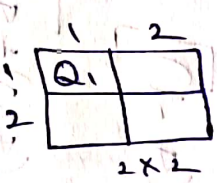
If n = 4 ; 4 - Queens problem
If n = 8 ; 8 - Queens problem

n = 1



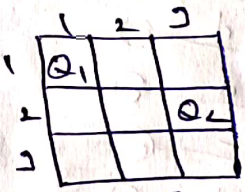
Trivial solution

n = 2



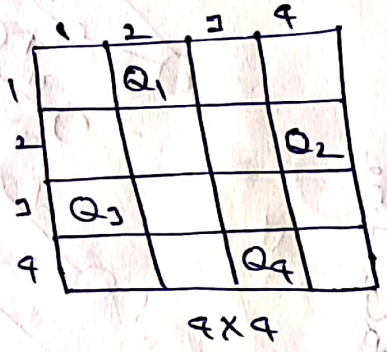
no solution

n = 3



no solution

n = 4



the solution vector for n = 4 is $(x_1, x_2, x_3, x_4) = (2, 4, 1, 3)$

(or)

$(x_1, x_2, x_3, x_4) = (3, 1, 4, 2)$

the number of ways 4 Queens can be placed on 4x4 chess board is $16C_4$

now by Applying explict Constraint the possible ways are $4! = 24$

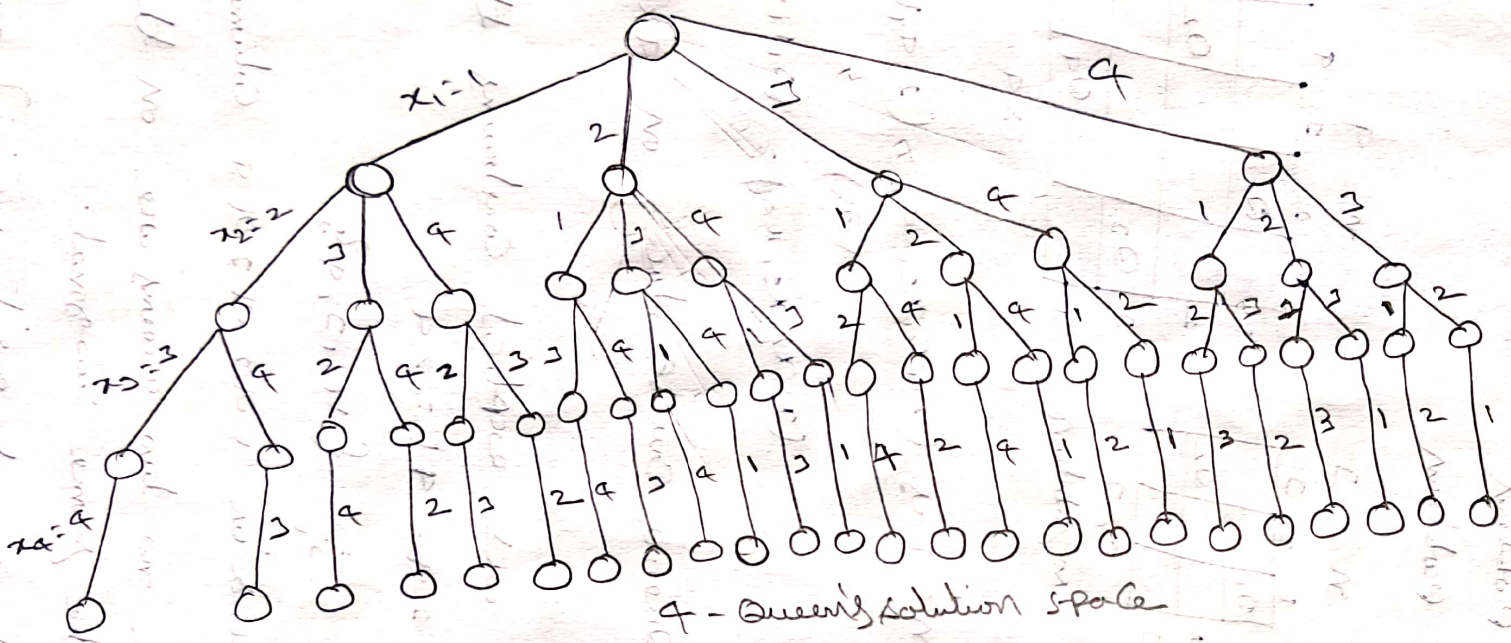
the explict Constraint are $S_i = \{1, 2, 3, 4\}$

x_i queen can be placed in 1 or 2 or 3 or 4 Columns

the implict Constraint are no two queens are on the same column and on the same diagonal.

Suppose two queens are placed at positions (i, j) and

state space tree



4-Queen's solution space

the total number of nodes in the 4-Queen's state space tree is $1 + \sum_{j=0}^{3} \left[\prod_{i=0}^{j} (4-i) \right]$

$= 1 + [(4\times3\times2\times1) + (4\times3\times2) + (4\times3) + 4]$

$= 1 + [4 + 12 + 24 + 24]$

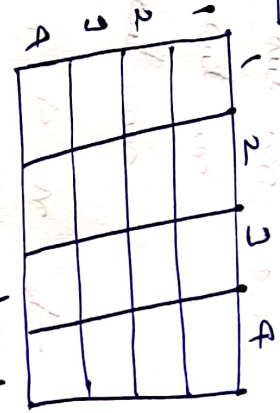
$= 1 + 64$

$= 65$

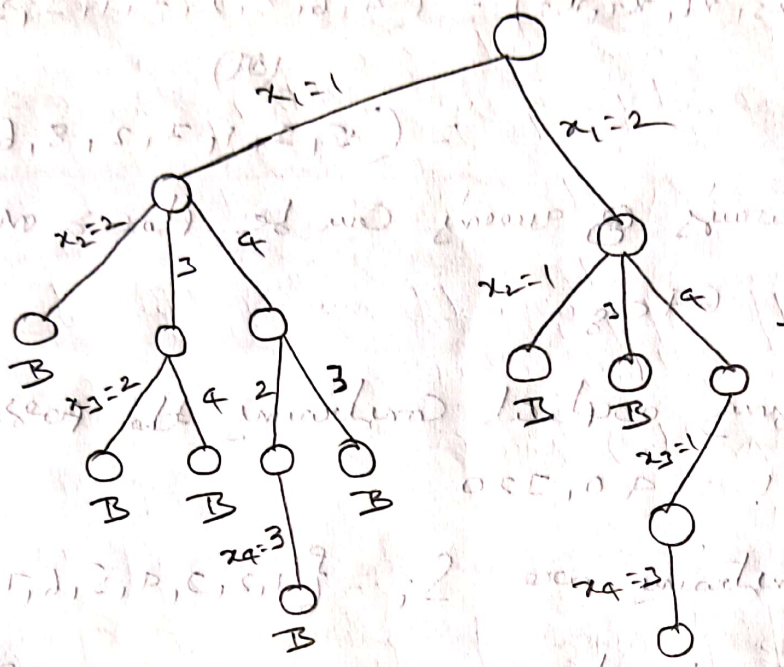
Hence the number of possible solutions are, by Applying explicit constraints are $4! = 24$ ways

and (k,l). then they are on the same diagonal if and only if $|j-k| = |i-l|$.

4-Queen's problem in Backtracking



Now by Applying Bounding function & backtracking



→ portion of the state space tree

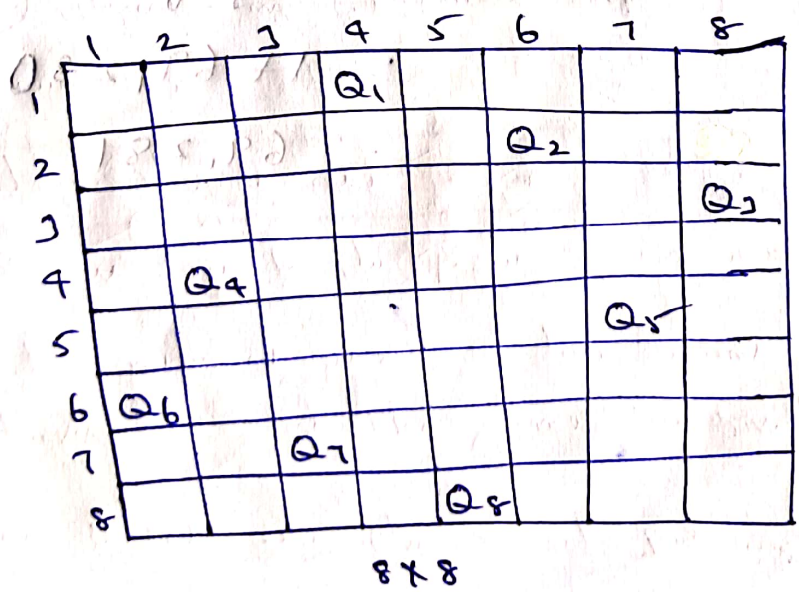
the solution vector is $(x_1, x_2, x_3, x_4) = (2, 4, 1, 3)$

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

8 - Queens problem in Backtracking :-

Consider 8 × 8 chess board. Now 8 queens are to be placed on an 8 × 8 chess board so that no two attack.

i.e., no two are on the same row
no two are on the same column.
no two are on the same diagonal



8 × 8

** By Applying Boundary function and Backtracking the solution vector for $n = 8$ is

$(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (4, 6, 8, 2, 7, 1, 3, 5)$

(or)

$= (5, 3, 1, 7, 2, 8, 6, 4)$

→ the number of ways 8 queens can be placed on 8×8 chess board is $64C_8$.

Now by applying explicit constraints the possible ways are $8! = 40,320$

the explicit constraints are $S_i = \{1, 2, 3, 4, 5, 6, 7, 8\}$

x_i queen can be placed in 1 2 3 4 5 6 7 8 columns.

the implicit constraints are no two queens are on the same column and on the same diagonal.

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

the total number of nodes in the 8-queens state space tree is $1 + \sum_{j=0}^{7} \left[\prod_{i=0}^{j} (8-i) \right]$

$= 1 + 69,280$

$= 69,281$ ← **

Sum of Subsets problem in Backtracking

→ Suppose we are given "n" distinct possible numbers and we desire to find all combinations of these numbers whose sum is "M". This is called as the "sum of subsets problem".

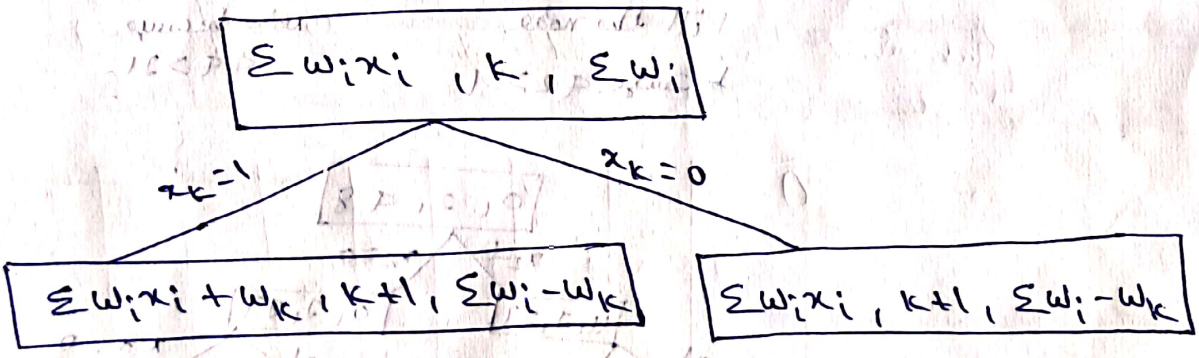
→ the element "x_i" of the solution vector is either 0 or 1 depending on whether the weight "w_i" is included or not.

→ for a node at level "i" be the left child corresponding to $x_i = 1$ and the right child corresponding to $x_i = 0$

→ the Bounding function we use $B_k(x_1, x_2, \ldots, x_k)$ = true if and only if $\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \geq M$

and $\sum_{i=1}^{k} w_i x_i + w_{k+1} \leq M$

→ the state space tree can be drawn using the following Technique.



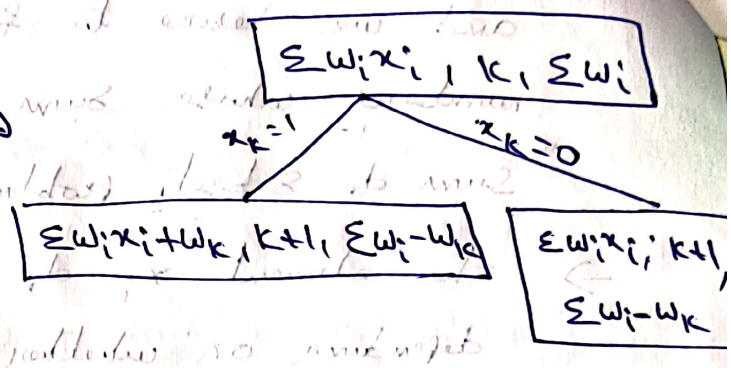
Ex:- Draw the state space tree for M = 31 and
$w[1:4] = \{7, 11, 13, 24\}$

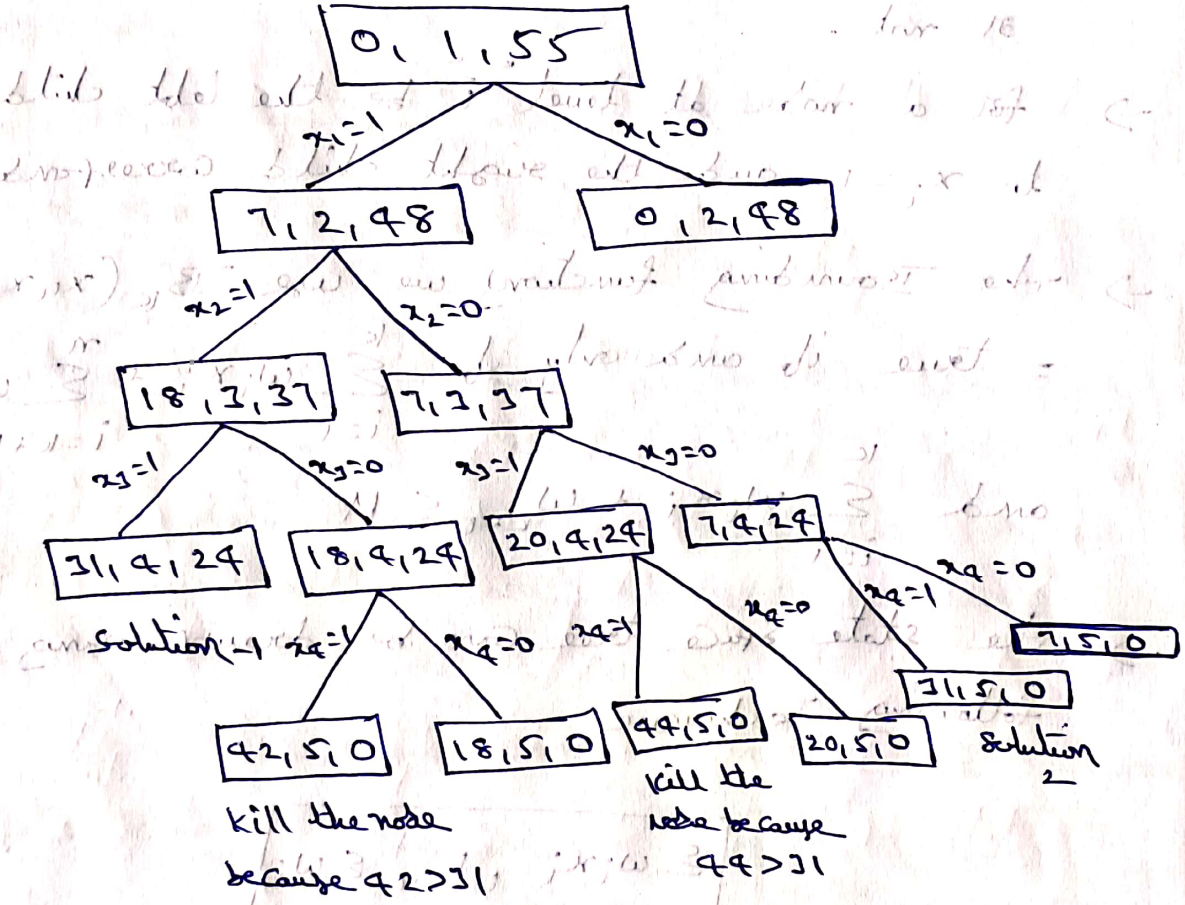
Soln

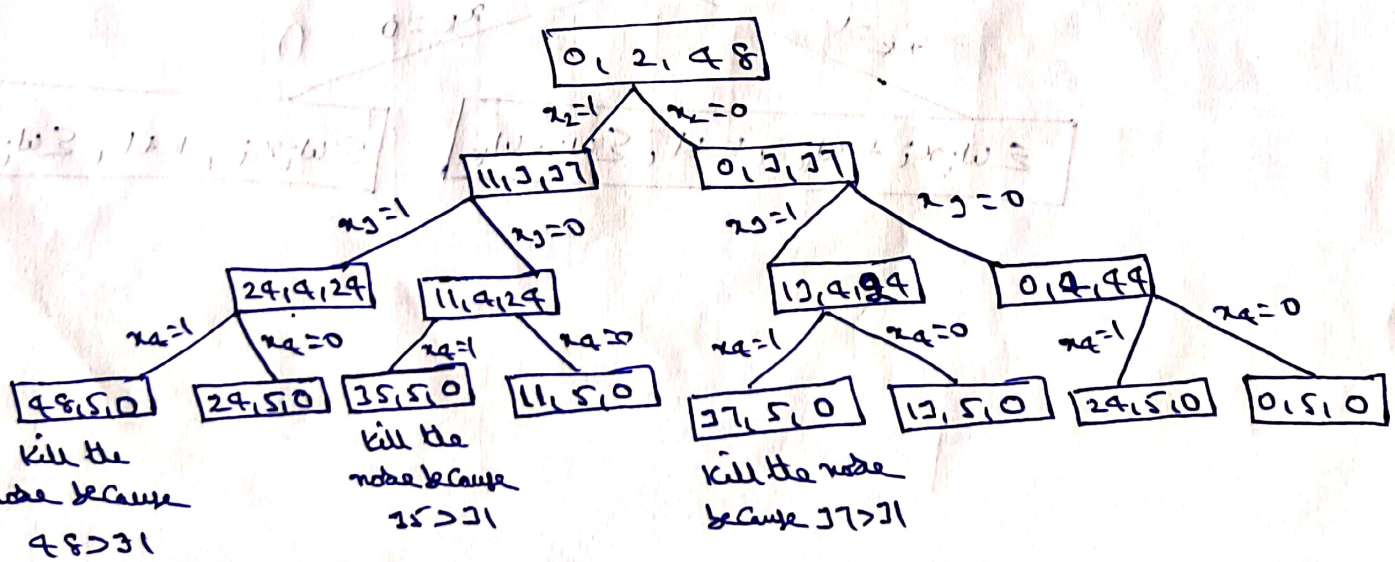
Hence

solution 1 = $\{x_1, x_2, x_3, x_4\} = \{1, 1, 1, 0\}$

solution 2 = $\{x_1, x_4\} = \{1, 0, 0, 1\}$





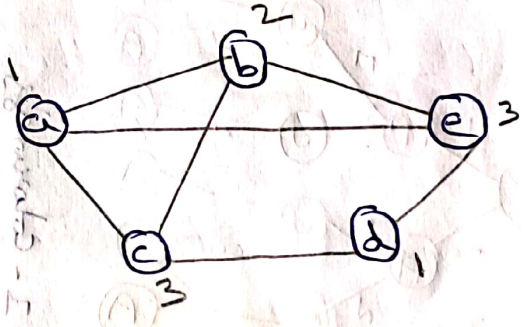


Draw the state space tree for $m = 30$ and $w[1:6] = \{5, 10, 12, 13, 15, 18\}$

Graph Coloring

→ Let $G = (V, E)$ be a graph, in graph coloring problem, we have to find out whether all the vertices of the given graph are colored or not, with the constraint that no two adjacent vertices have the same color.

Ex:-



m = 3 colored graph

Example graph and its coloring.

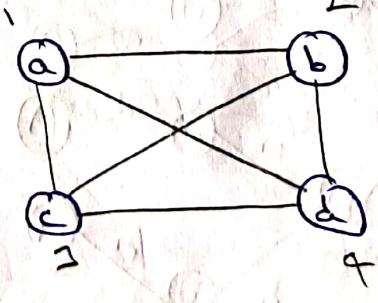
→ the problem has two versions

1. m - Colorability Decision problem.
2. m - colorability optimization problem.

→ chromatic number :-

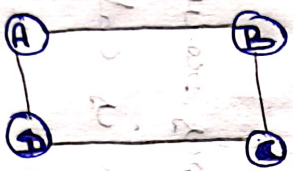
the minimum number of colors required to color all the vertices of the given graph is called chromatic number.

Ex:-

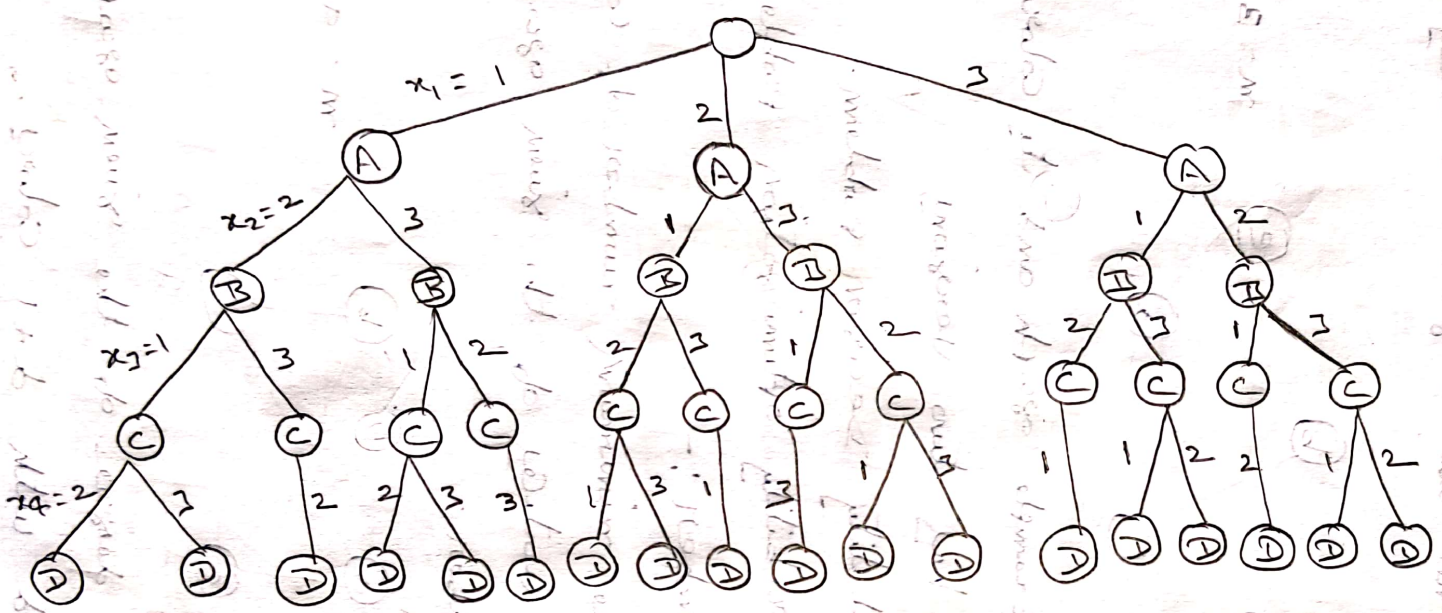


m = 4 - colored graph

→ If d is the degree of the given graph, then it can be colored with $d+1$ colors.



$n = 4$, $m = 3$

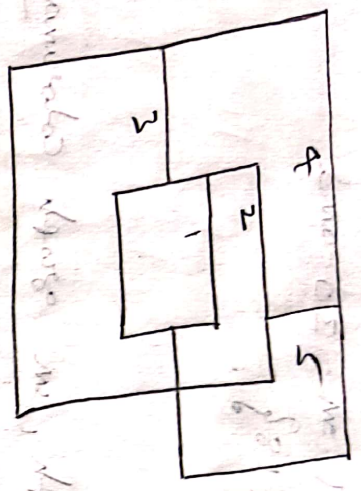


State space tree for a 4-Node graph and all possible 3-Colorings

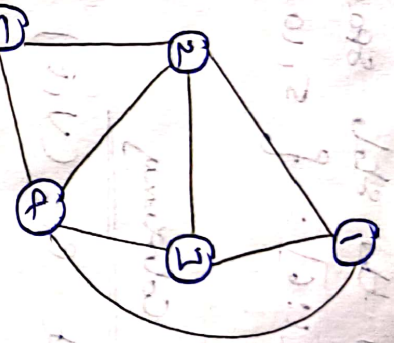
Time Complexity = $O(n \cdot m^n)$.

Ex:- Explain the graph-coloring problem. And draw the state space tree for $m = 3$ colors $n = 3$ vertices graph.

A MAP



Planar Graph representation.



Algo

// K

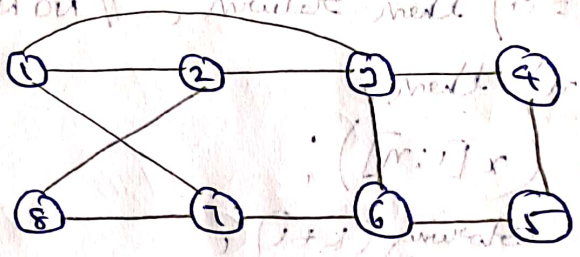
```
Algorithm m Coloring (k)
// k is the index of the next vertex to color.
{
    repeat
    {
        Next value (k);
        if (x[k] = 0) then return; // NO New Color Possible
        if (k = n) then
            write (x[1:n]);
        else m coloring (k+1);
    }
        until (false);
}

Algorithm Next value (k)
{
    repeat
    {
        x[k] = (x[k]+1) mod (m+1); // Next highest color
        if (x[k] = 0) then return; // All colors have been used.
        for j = 1 to n do
        {
            if ((G[k,j] ≠ 0) and (x[k] = x[j]))
            then break;
        }
        if (j = n+1) then return; // new color found
    }
    until(false); // otherwise try to find another color
}
```

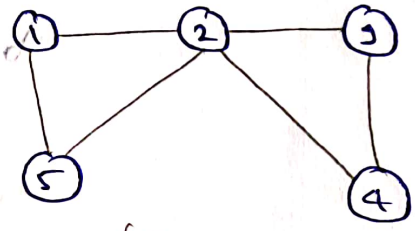
Hamiltonian Cycles

Let $G = (V, E)$ is a connected graph with "n" vertices. A Hamiltonian cycle is a round trip path along "n" edges of G that visits each and every vertex exactly once except the starting and ending vertex and returning to the starting vertex.

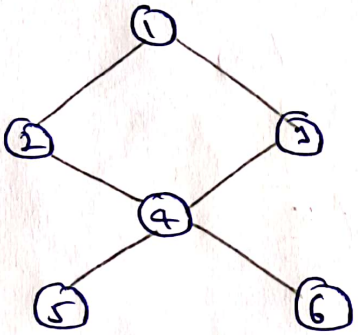
Ex:-



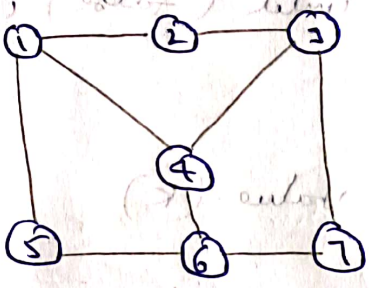
(G_1)



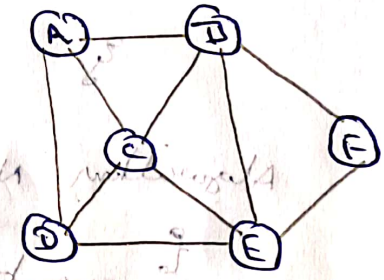
(G_2)



(G_3)



(G_4)



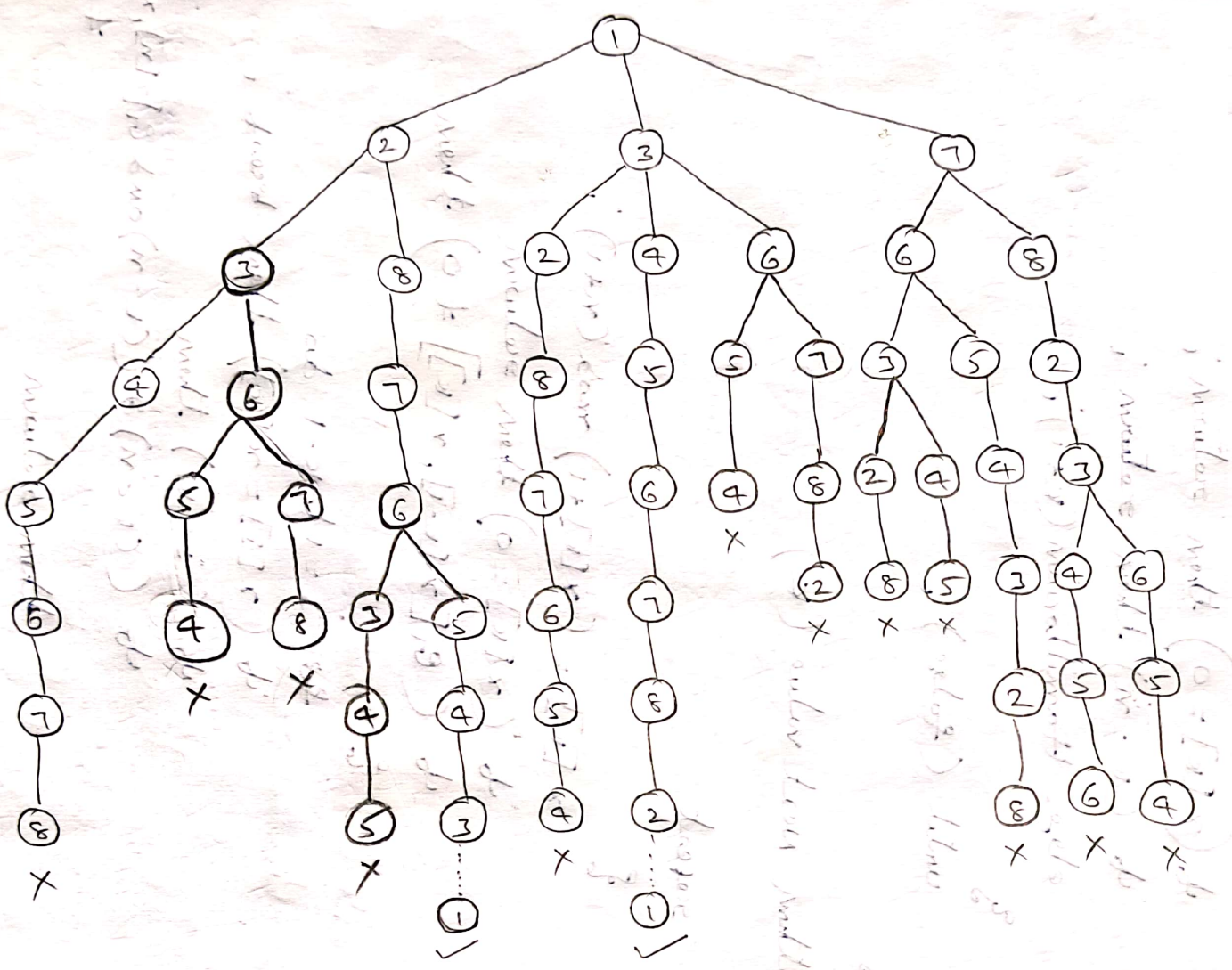
(G_5)

The graph G_1, G_5 Contain Hamiltonian Cycles

The graph G_2, G_3, G_4 does not Contain Hamiltonian Cycles.

```
Algorithm Hamiltonian(k)
{
    repeat
    {
        Next value(k);
        if(x[k]=0) then return;
        if (k=n) then
        else Hamiltonian(k+1);
    }
    until (false);
}

Algorithm Next value(k)
{
    repeat
    {
        x[k] = (x[k]+1) mod(n+1);
        if (x[k]=0) then return
        if (G[x[k-1], x[k]] ≠0) then
        {
            for j:=1 to k-1 do
            if (x[j]=x[k]) then break;
            if (j=k) then
            if ((k<n) or ((k=n) and G[x[n], x[1]]
                                    ≠0))
                then return
        }
    }
    until (false);
}
```



the portion of the state space tree for G_1

The Hamiltonian cycles are

① 1 – 2 – 8 – 7 – 6 – 5 – 4 – 3 – 1

② 1 – 3 – 4 – 5 – 6 – 7 – 8 – 2 – 1

Branch and Bound General Method

→ Branch and Bound is a systematic method for solving optimization problems.

→ Branch and Bound technique applied when the greedy method and dynamic programming method may fail.

→ Branch and Bound is much slower. Indeed it often leads to exponential time complexities in the worst case. On the other hand, if applied carefully it can lead to algorithms that run reasonably fast on average.

→ The general idea of Branch and Bound is a BFS like search for the optimal solution but not all the nodes get expanded. Rather, a carefully selected criterion determines which node to expand and when and another criterion tells the algorithm when an optimal solution has been found.

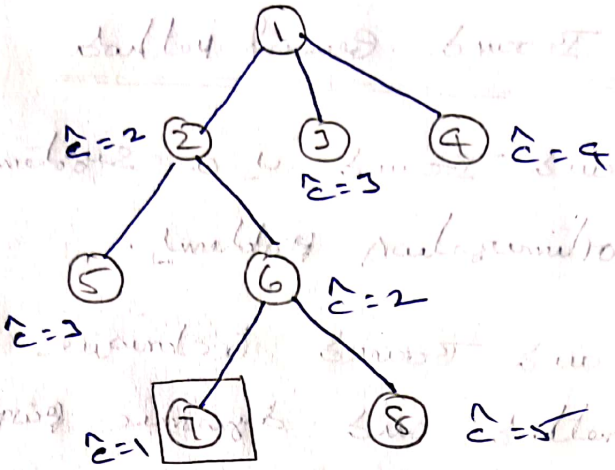
→ Both BFS and DFS generalized to Branch and Bound strategies.

1. BFS is an FIFO search in terms of live nodes. List of live nodes is a Queue.

2. DFS is a LIFO search in terms of live nodes. List of live nodes is a stack.

→ Branch and Bound refers to all the state space search methods in which all the children of an E-node are generated before any other live node can become the E-node.

Ex:-



Initially we will take node 1 as E-node. Next generate the children of the node 1. The children of node 1 are 2, 3 and 4. By using ranking function, we will calculate the cost of node 2, 3 and 4. $\hat{c}(2)=2$; $\hat{c}(3)=3$; $\hat{c}(4)=4$. respectively

Next we will select a node which has minimum cost. i.e node 2. Generate the children of node 2 i.e. node 5 and 6. Between node 5 and 6, we will select node 6, since the cost is minimum.

Generate the children of node 6 i.e nodes 7 and 8, we will select node 7, since the cost is minimum. node 7 is the answer node, now we have to terminate the searching process.

→ Branch and Bound methods of algorithm design involves:

1. Tree organization of solution space
2. use of Bounding function to limit the search. i.e. to avoid the generation of sub trees that do not contain an answer node.

→ Applications of Branch and Bound Technique:

1. 0/1 Knapsack problem
2. Travelling salesperson problem (TSP)

→ Search techniques used in Branch and Bound:

1. BFS 2. DFS 3. least cost search (LC)

→ Two types of Bounds used in LCBB

1. Lower Bound ($\hat{c}$)
2. upper Bound ($\hat{u}$)

→ while calculating $\hat{c}$ for a node in state space tree, fractions are allowed.

→ while calculating $\hat{u}$ for a node in state space tree, fractions are not allowed.

→ while node $\hat{u}$ is minimum cost that node is expanded, that node becomes the E-node.

→ Two types of Branch and Bound Techniques available.

1. least cost Branch and Bound (LCBB)
2. FIFO Branch and Bound (FIFOBB)

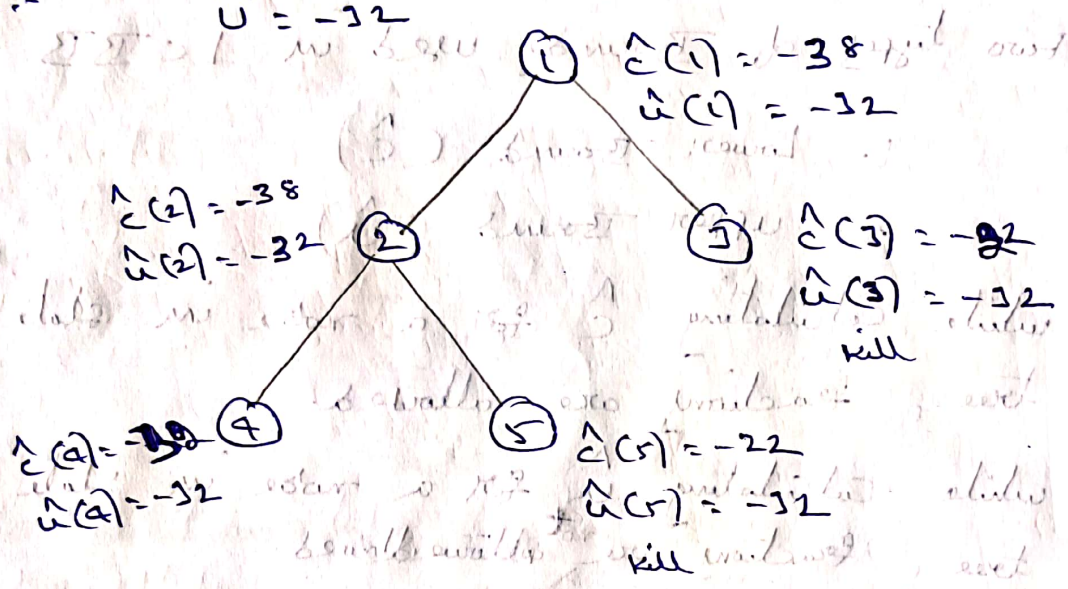
→ In case of FIFOBB, we have to calculate three Bounds.

1. lower Bound ($\hat{c}(x)$)
2. upper Bound ($\hat{u}(x)$) where x is any node.
3. Global upper Bound ($\hat{U}$)

→ To expand a node in FIFO BB, the following Constraint is used.

If lower bound of a node $\hat{c}(x)$ is greater than global upper bound $\hat{U}$, then we kill the node, otherwise we expand the node.

Ex:- $\hat{U} = -32$



0/1 knapsack problem using Branch and Bound

→ there are 'n' objects and the capacity of the knapsack is M. we have to select some objects from 'n' objects in such a way that it should not exceed the capacity of the knapsack 'M' and the maximum profit can be earned.

→ So, the knapsack is maximization problem. But Branch and Bound easily only with the minimization problem. So, we modify the knapsack problem to the minimization problem. the modified problem is

$$\min z = -p_1x_1 - p_2x_2 - \cdots - p_nx_n$$

the Constraints are $w_1x_1 + w_2x_2 + \cdots + w_nx_n \leq M$

and $x_i = 0$ or 1

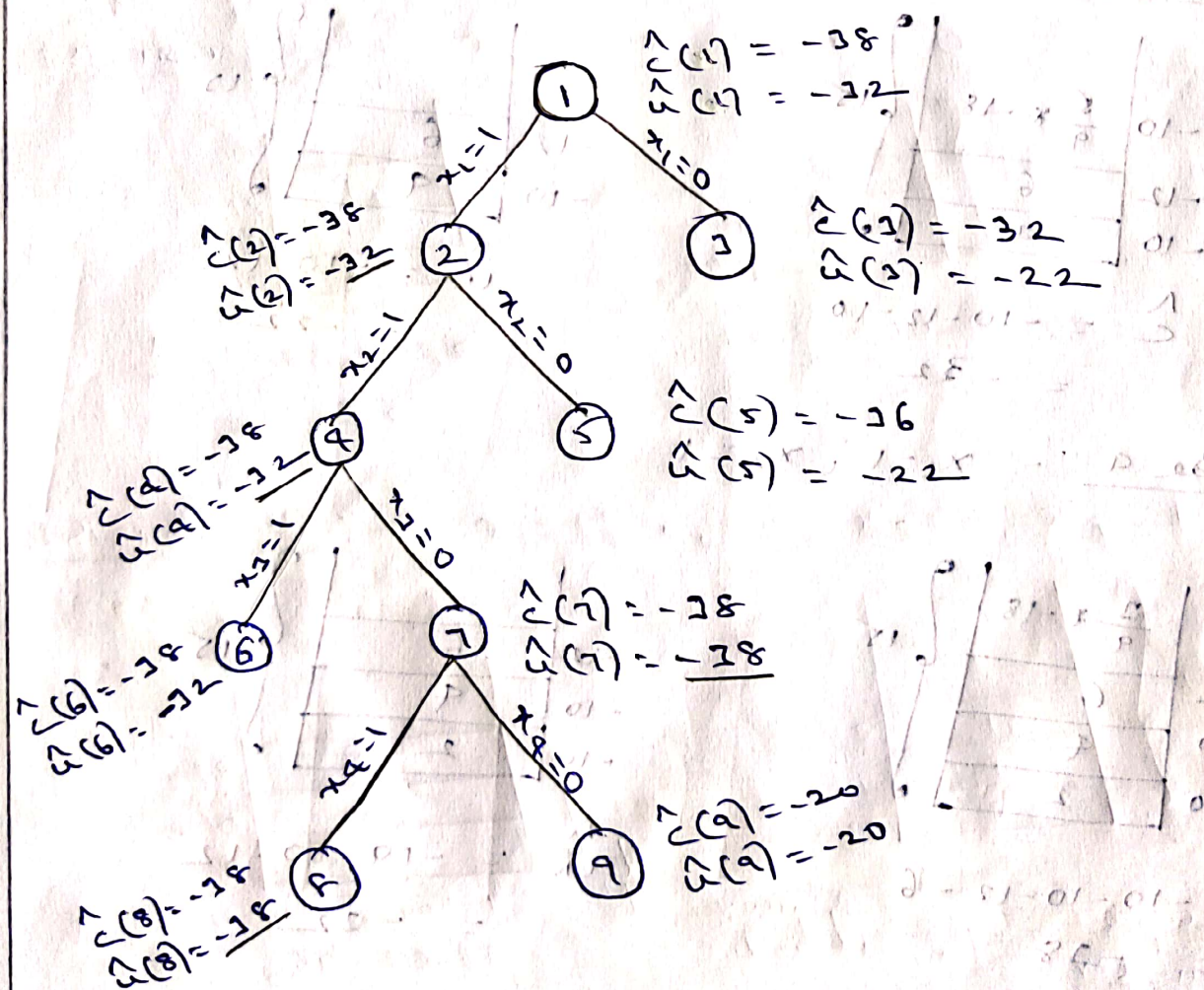
→ In Branch and Bound, we calculate the lower bound and upper bound for each and every node in state space tree.

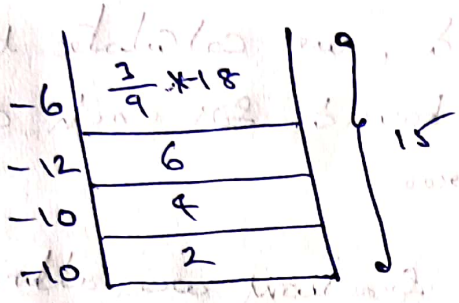
→ In lower bound ($\hat{c}$), fractions are allowed,
In upper bound ($\hat{u}$), fractions are not allowed.

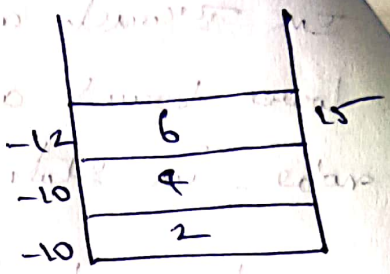
Ex:- Draw the portion of state space tree generated by LCBB for knapsack instance, $n = 4$, $M = 15$
$(P_1, P_2, P_3, P_4) = (10, 10, 12, 18)$, $(w_1, w_2, w_3, w_4) = (2, 4, 6, 9)$

Soln: we have to convert the given profits into negative profits by putting -ve sign to each and every profit
$\therefore (P_1, P_2, P_3, P_4) = (-10, -10, -12, -18)$.

Here $\hat{c}$ = lower bound ; $\hat{u}$ = upper bound.



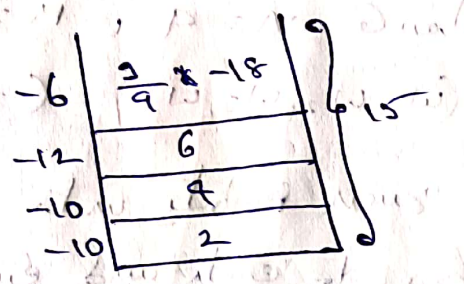


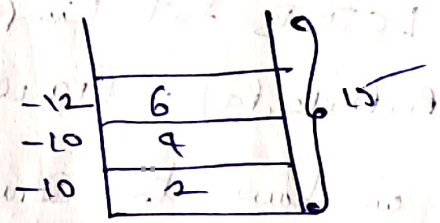


lower bound ($\hat{c}$) = -10-10-12-6
= -38

upper bound ($\hat{u}$)
= -12-10-10 = -32

For node 2 :- $x_1 = 1$

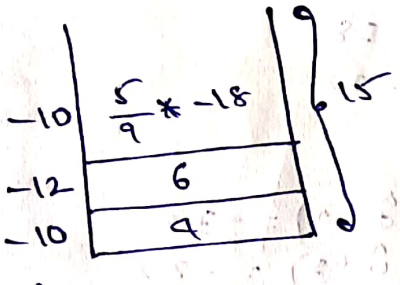




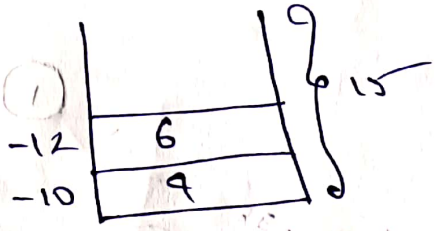
lower bound ($\hat{c}$) = -10-10-12-6
= -38

upper bound ($\hat{u}$) = -10-10-12
= -32

For node 3 :- $x_1 = 0$

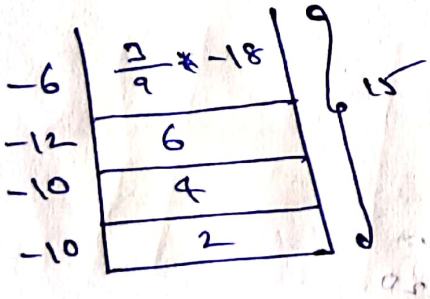


$\hat{c}$ = -10-12-10
= -32

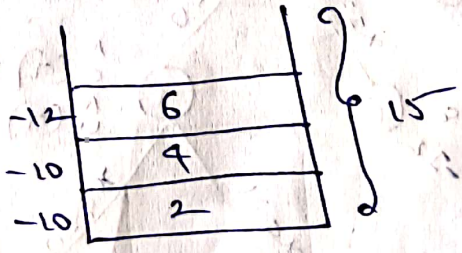


$\hat{u}$ = -10-12
= -22

For node 4 :- $x_1 = 1$; $x_2 = 1$

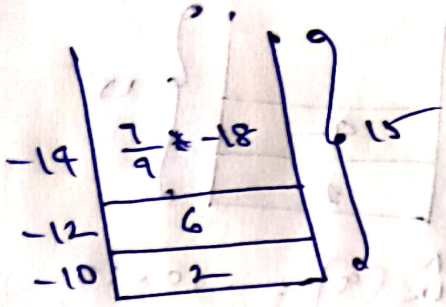


$\hat{c}$ = -10-10-12-6
= -38



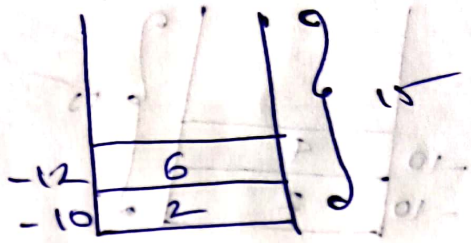
$\hat{u}$ = -10-10-12
= -32

For Node 5 :- $x_1 = 1$, $x_2 = 0$



$\therefore \hat{c} = -10-12-14$

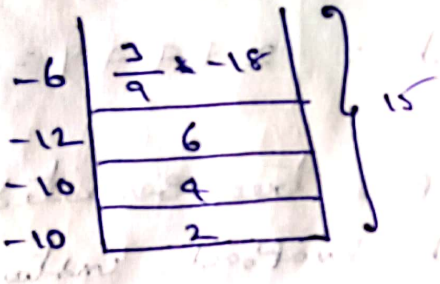
$= -36$



$\hat{u} = -10-12$

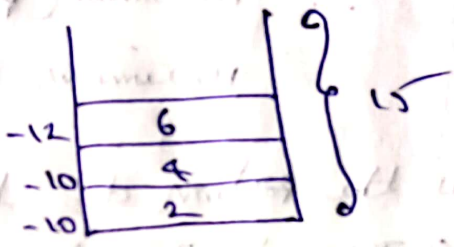
$= -22$

For Node 6 :- $x_1 = 1$, $x_2 = 1$, $x_3 = 1$



$\hat{c} = -10-10-12-6$

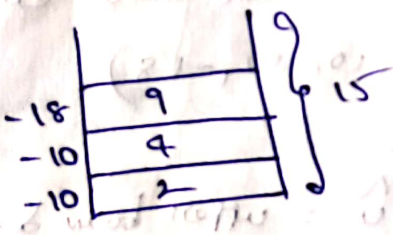
$= -38$



$\hat{u} = -10-10-12$

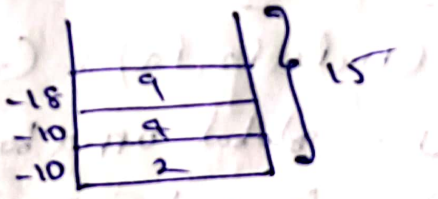
$= -32$

For Node 7 :- $x_1 = 1$, $x_2 = 1$, $x_3 = 0$



$\hat{c} = -10-10-18$

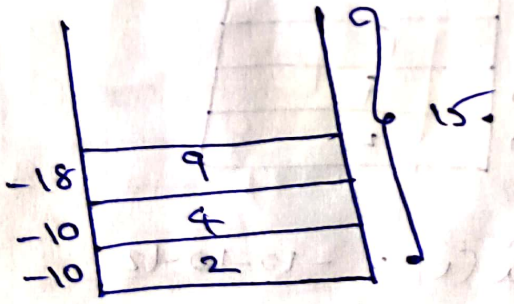
$= -38$



$\hat{u} = -10-10-18$

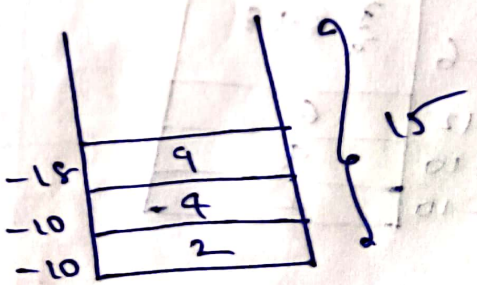
$= -38$

For Node 8 :- $x_1 = 1$, $x_2 = 1$, $x_3 = 0$, $x_4 = 1$



$\hat{c}(8) = -10-10-18$

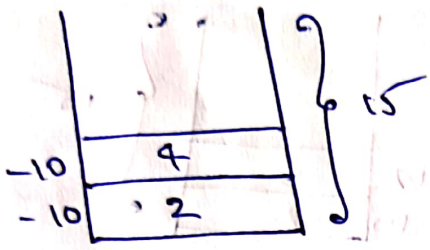
$= -38$



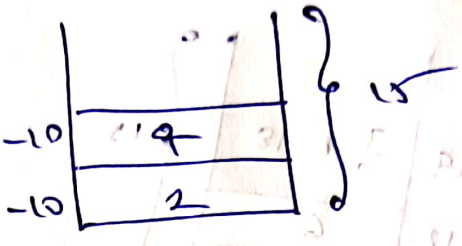
$\hat{u}(8) = -10-10-18$

$= -38$

For node 9 :- $x_1 = 1$; $x_2 = 1$; $x_3 = 0$; $x_4 = 0$



$\hat{c}(9) = -10 - 10$
$= -20$



$\hat{u}(9) = -10 - 10$
$= -20$

Hence the solution vector $(x_1, x_2, x_3, x_4) = (1, 1, 0, 1)$

∴ Maximum profit = $10 + 10 + 18 = 38$

Maximum weight = $2 + 4 + 9 = 15$

Ex: Draw the portion of state space tree generated by FIFO Branch Bound for the knapsack instance, $n = 4$, $(P_1, P_2, P_3, P_4) = (10, 10, 12, 18)$ and $(W_1, W_2, W_3, W_4) = (2, 4, 6, 9)$, $M = 15$

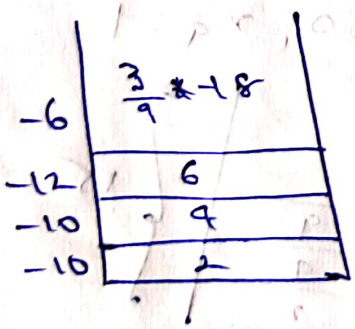
Solution: we have to convert the given profits into negative profits by putting -ve sign of each and every profit

∴ $(P_1, P_2, P_3, P_4) = (-10, -10, -12, -18)$

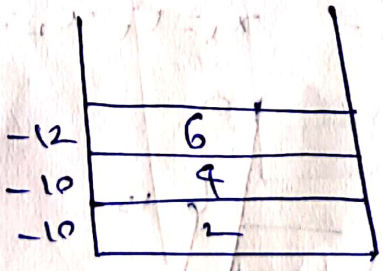
Here $\hat{U}$ = global upper bound

$\hat{c}$ = lower bound ; $\hat{u}$ = upper bound.

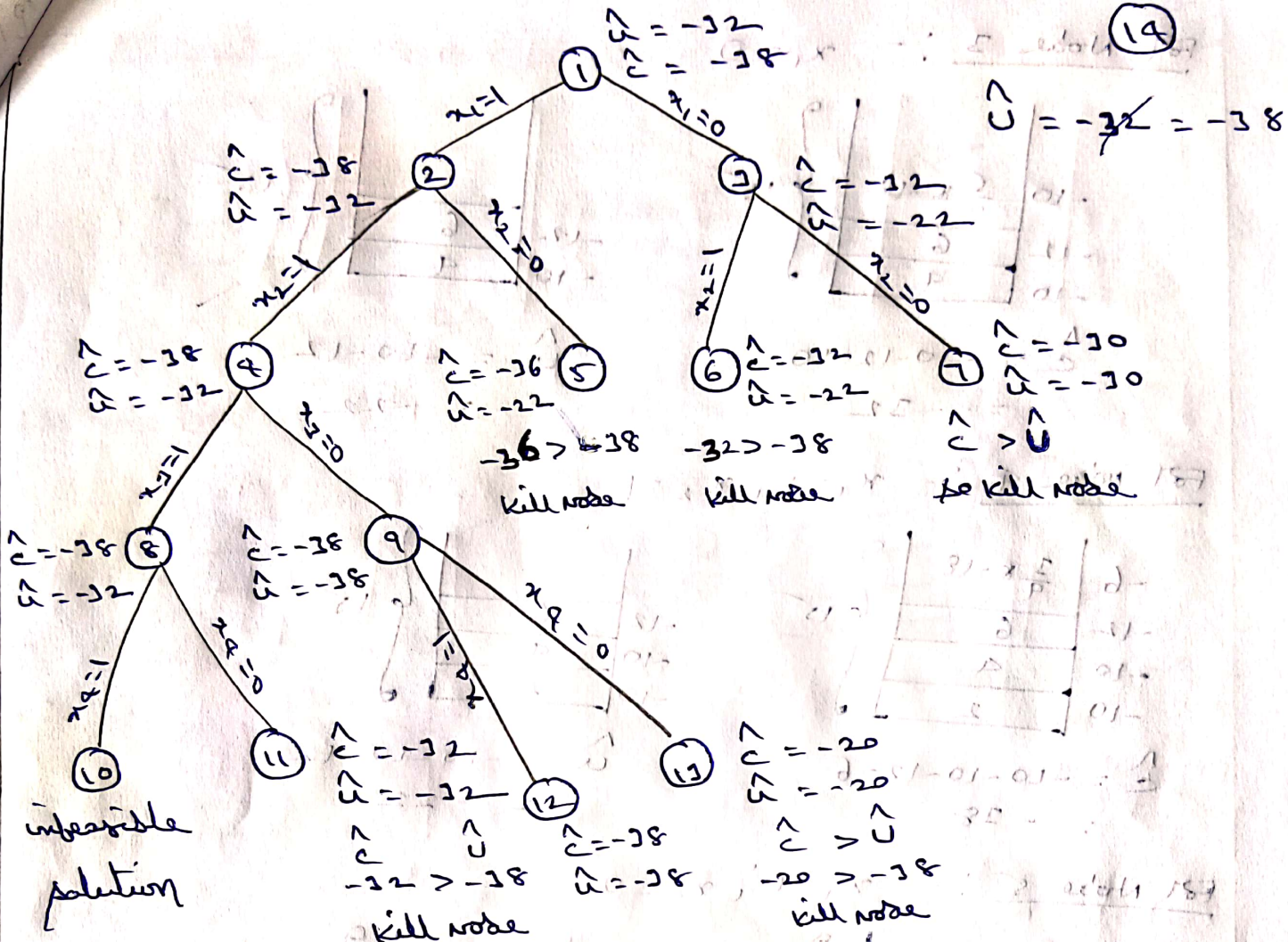
For node 1 :-



$\hat{c}(1) = -10 - 10 - 12 - 6$
$= -38$



$\hat{u}(1) = -10 - 10 - 12$
$= -32$



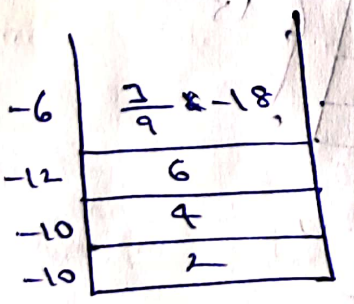
$\hat{u} = -32 = -38$

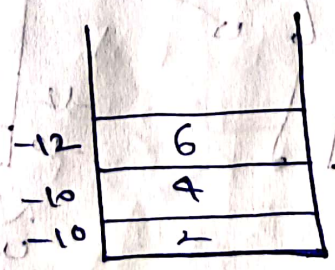
the solution vector $(x_1, x_2, x_3, x_4) = (1, 1, 0, 1)$

Maximum profit = $10 + 10 + 18 = 38$

Maximum weight = $2 + 4 + 9 = 15$

For Node 2 :- $x_1 = 1$





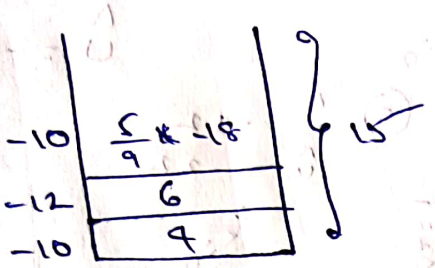
$\hat{c}(2) = -10 - 10 - 12 - 6$

$= -38$

$\hat{u}(2) = -10 - 10 - 12$

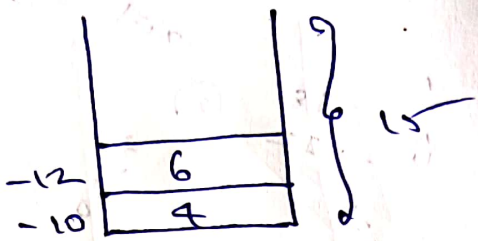
$= -32$

For Node 3 :- $x_1 = 0$



$\hat{c} = -10 - 12 - 10$

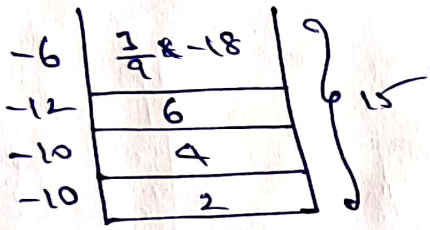
$= -32$



$\hat{u} = -10 - 12$

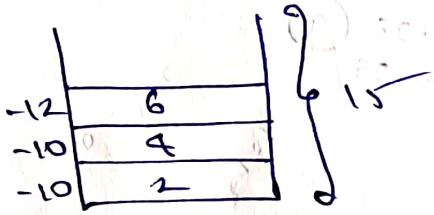
$= -22$

For Node 4 :- $x_1 = 1, x_2 = 1$



$\hat{c} = -10 - 10 - 12 - 6$

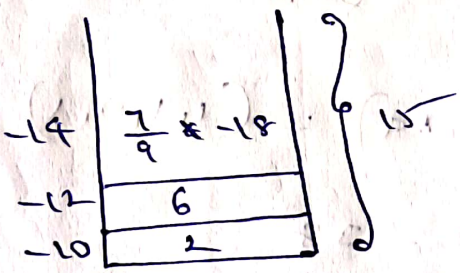
$= -38$



$\hat{u} = -10 - 10 - 12$

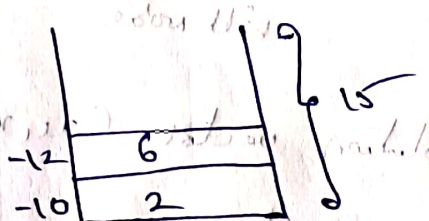
$= -22$

For Node 5 :- $x_1 = 1, x_2 = 0$



$\hat{c} = -10 - 12 - 14$

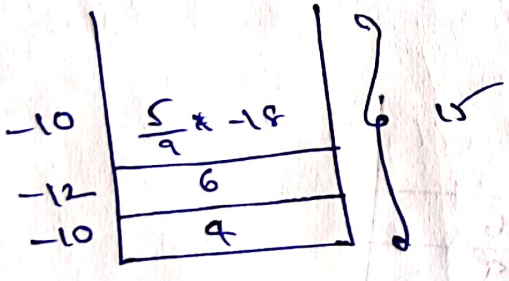
$= -36$



$\hat{u} = -10 - 12$

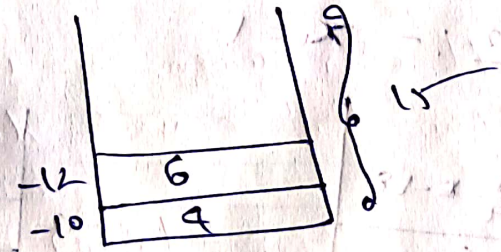
$= -22$

For Node 6 :- $x_1 = 0, x_2 = 1$



$\hat{c} = -10 - 12 - 10$

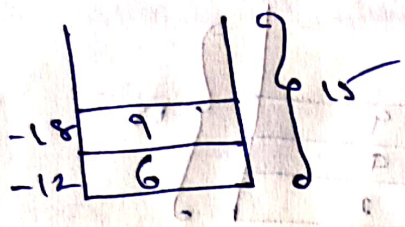
$= -32$



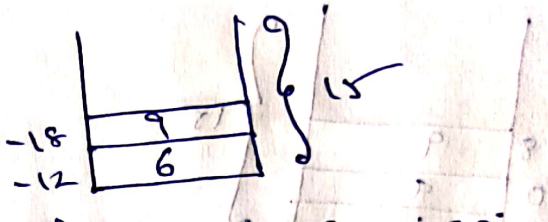
$\hat{u} = -10 - 12$

$= -22$

For Node 7 :- $x_1 = 0$; $x_2 = 0$

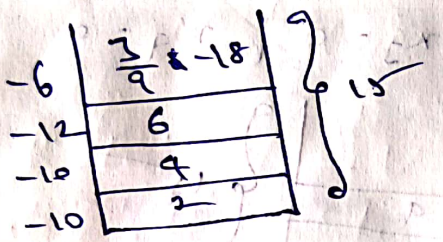


$\hat{c} = -12 - 18 = -30$

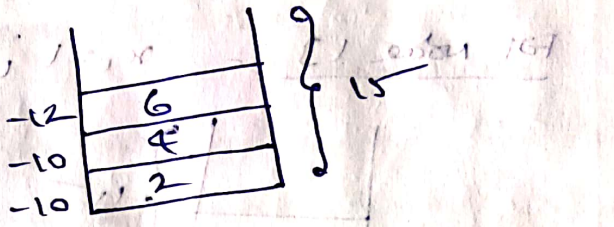


$\hat{u} = -12 - 18 = -30$

For Node 8 :- $x_1 = 1$; $x_2 = 1$; $x_3 = 1$

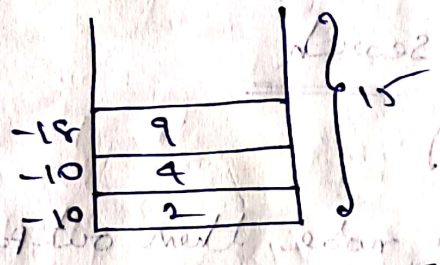


$\hat{c} = -10 - 10 - 12 - 6 = -38$

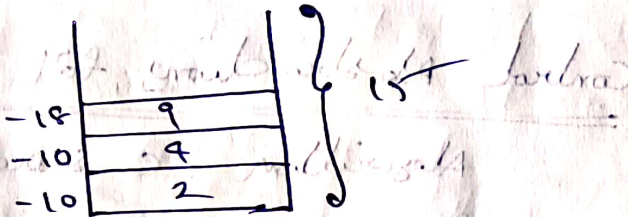


$\hat{u} = -10 - 10 - 12 = -32$

For Node 9 :- $x_1 = 1$; $x_2 = 1$; $x_3 = 0$



$\hat{c} = -10 - 10 - 18 = -38$



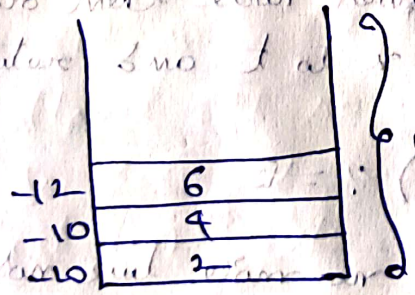
$\hat{u} = -10 - 10 - 18 = -38$

For Node 10 :- $x_1 = 1$; $x_2 = 1$; $x_3 = 1$; $x_4 = 0$

total weight $= 2 + 4 + 6 + 9 = 21 < 15 = M$ false

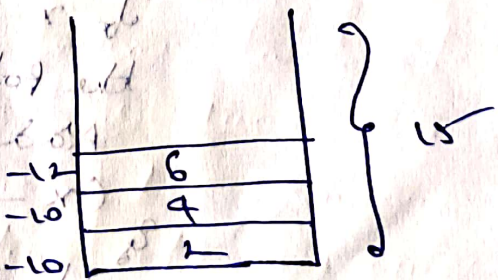
It is infeasible solution

For Node 11 :- $x_1 = 1$; $x_2 = 1$; $x_3 = 0$; $x_4 = 0$



$\hat{c} = -10 - 10 - 12$

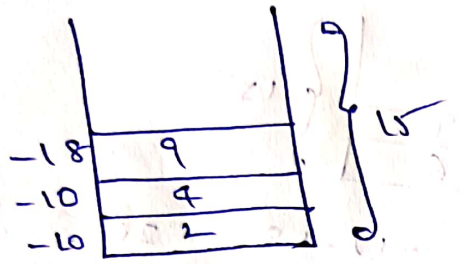
$= -32$



$\hat{u} = -10 - 10 - 12$

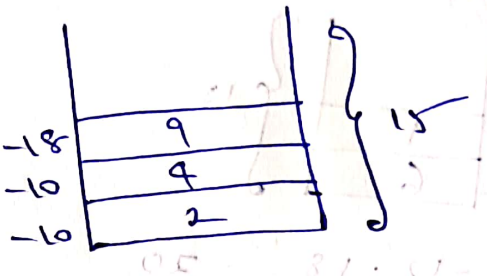
$= -32$

For node 12 :- $x_1=1$; $x_2=1$; $x_3=0$; $x_4=1$



$\hat{c} = -10-10-18$

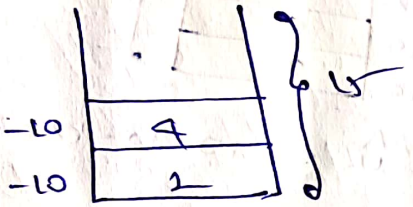
$= -38$



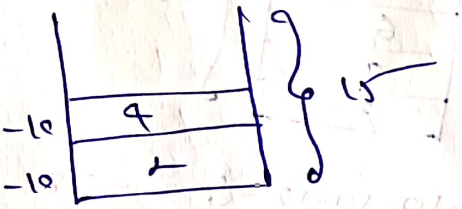
$\hat{u} = -10-10-18$

$= -38$

For node 13 :- $x_1=1$; $x_2=1$; $x_3=0$; $x_4=0$



$\hat{c} = -10-10 = -20$



$\hat{u} = -10-10 = -20$

Control Abstractions for LC-Search

```
Algorithm LC search (t)
{
    If t is an answer node then output t
    and return;
    E := t;
    Initialize the list of live nodes to be empty;
    repeat
    {
        for each child x of E do
        {
            if x is an answer node then output
            the path from x to t and return;
            Add (x);
            (x -> Parent) := E
        }
        if there are no more live nodes then
        { write ("No answer node"); return
        }
        E := least ();
    } until (false);
```

Distinguish between back tracking and branch and bound techniques.

Solution

	Back tracking	Branch and Bound
1	In back tracking, DFS technique is used for tracing the solution for the given problem.	In Branch and Bound, BFS technique is used for tracing the solution for the given problem.
2	Typically decision problems can be solved using Back tracking	Typically optimization problems can be solved using branch and bound.
3	while finding the solutions to the problem, bad choices can be made.	It proceeds only on optimal or better solutions
4	the state space tree is searched until the solution is obtained.	the state space tree needs to be searched completely as there may be chances of being an optimal solution any where in the state space tree.
5	Applications of Back tracking are: N-Queens problem. Graph Coloring problem. Hamiltonian Cycle problem.	Applications of branch and bound are: 0/1 knapsack problem. Travelling sales person problem.

Travelling salesperson problem using LCBB

If there are n cities and cost of travelling from one city to another city given. A salesperson has to start from one city and has to visit all the cities exactly once and has to return to the starting place with shortest distance or minimum cost.

→ A Tour of G is a directed simple cycle that includes every vertex in "V". the cost of the Tour is the sum of the costs of the edges on the Tour.

→ Let C_{ij} be the cost of the edge (i,j) and $C_{ij} = \infty$ if $(i,j) \notin E(G)$. and $|V| = n$

→ A row or column is said to be reduced, if it contains atleast one zero and all the remaining entries are non-negative

→ A Matrix is reduced, iff every row and column is reduced.

→ If a constant "t" is chosen to be minimum entry in row i or column j, then subtract it from all entries in row i or column j will introduce a zero into a row i or column j.

→ the total amount subtracted from the columns and rows is lower bound on the length of a minimum cost tour and can be used as $\hat{c}(x)$ value for the root of the state space tree.

→ Let A be the reduced cost matrix for the node R Let S be the child of R such that the edge (R,S) corresponds to including edge (i,j)

in the tour.

If 'S' is not a leaf node, then the reduced cost matrix for node S can be obtained as follows.

(i) change all the entries in row i and column j to ∞

(ii) set A(j,1) to ∞

(iii) Apply row reduction and column reduction except for rows and cols containing "∞"

(iv) the total cost of a node "S" can be calculated as $\hat{c}(S) = \hat{c}(R) + A(i,j) + r$

where "r" is the total amount subtracted in step 3

Q Apply the least cost Branch and Bound algorithm to solve the TSP for the following cost matrix

$$\begin{bmatrix} \infty & 11 & 10 & 9 & 6 \\ 8 & \infty & 7 & 3 & 4 \\ 8 & 4 & \infty & 4 & 8 \\ 11 & 10 & 5 & \infty & 5 \\ 6 & 9 & 5 & 5 & \infty \end{bmatrix}$$

Soln row reduction :-

$$\begin{bmatrix} \infty & 11 & 10 & 9 & 6 \\ 8 & \infty & 7 & 3 & 4 \\ 8 & 4 & \infty & 4 & 8 \\ 11 & 10 & 5 & \infty & 5 \\ 6 & 9 & 5 & 5 & \infty \end{bmatrix} \begin{matrix} 6 \\ 3 \\ 4 \\ 5 \\ 5 \end{matrix} = \begin{bmatrix} \infty & 5 & 4 & 3 & 0 \\ 5 & \infty & 4 & 0 & 1 \\ 4 & 0 & \infty & 0 & 4 \\ 6 & 5 & 0 & \infty & 0 \\ 1 & 4 & 0 & 0 & \infty \end{bmatrix}$$

$$\underline{23}$$

Column reduction :-

$$\begin{bmatrix} \infty & 5 & 4 & 3 & 0 \\ 5 & \infty & 4 & 0 & 1 \\ 4 & 0 & \infty & 0 & 4 \\ 6 & 5 & 0 & \infty & 0 \\ 1 & 4 & 0 & 0 & \infty \end{bmatrix} = \begin{bmatrix} \infty & 5 & 4 & 3 & 0 \\ 4 & \infty & 4 & 0 & 1 \\ 3 & 0 & \infty & 0 & 4 \\ 5 & 5 & 0 & \infty & 0 \\ 0 & 4 & 0 & 0 & \infty \end{bmatrix}$$

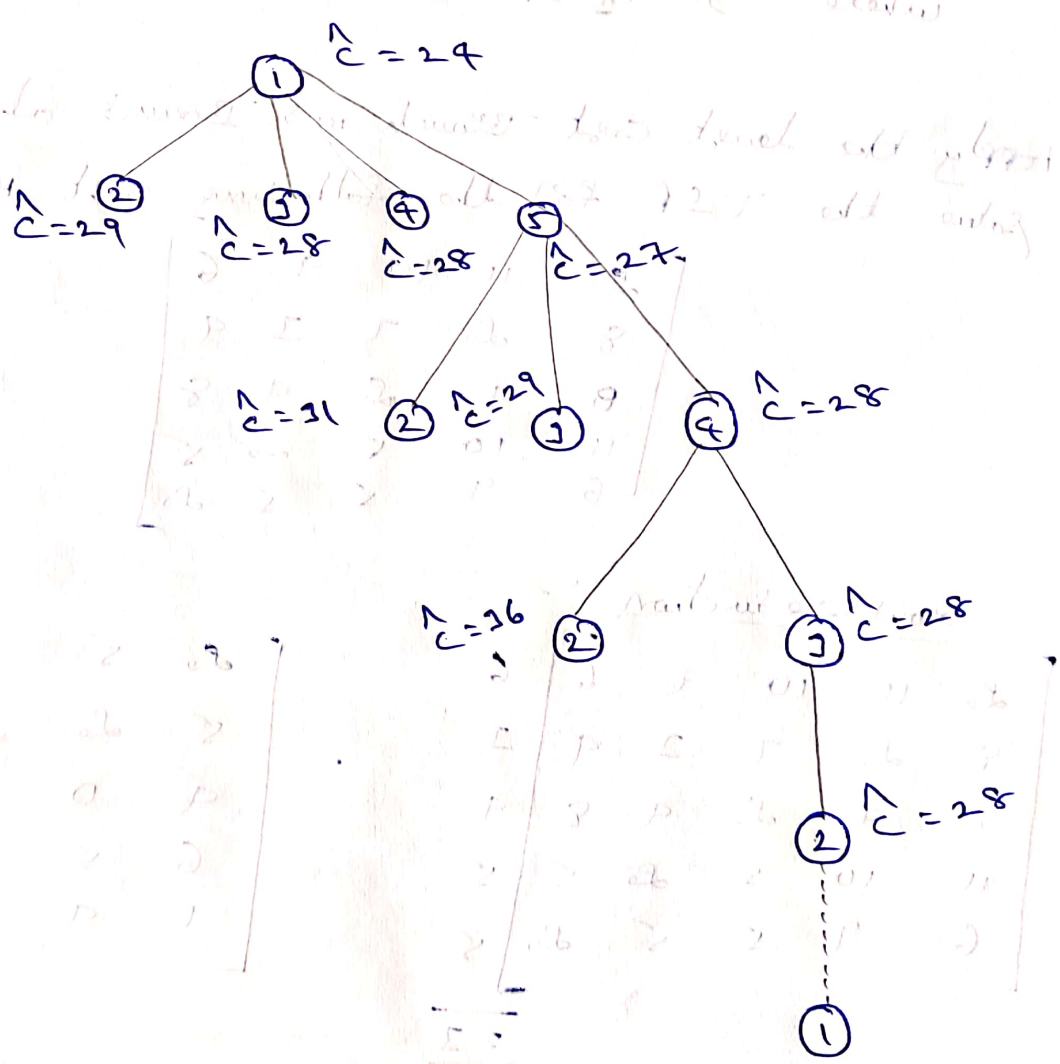
1 0 0 0 0 = 1

reduction cost = row reduction cost + column reduction cost

= 23 + 1

= 24

Cost of the root node or vertex 1 is $\hat{c}(1) = 24$



$$A = \begin{bmatrix} \infty & 5 & 4 & 3 & 0 \\ 4 & \infty & 4 & 0 & 1 \\ 3 & 0 & \infty & 0 & 4 \\ 5 & 5 & 0 & \infty & 0 \\ 0 & 4 & 0 & 0 & \infty \end{bmatrix}$$

A is the fully reduced matrix

Consider the path $i=2$:-

Make the 1st row and 2nd column of A to ∞ and set $A[2,1] = \infty$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 4 & 0 & 1 \\ 3 & \infty & \infty & 0 & 4 \\ 5 & \infty & 0 & \infty & 0 \\ 0 & \infty & 0 & 0 & \infty \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}$$

$$\begin{matrix} 0 & 0 & 0 & 0 \end{matrix}$$

$\therefore$ row reduction $= 0$ and column reduction $= 0$

$\therefore r = 0 + 0 = 0$

$$\hat{C}(2) = \hat{C}(1) + A[1,2] + r$$

$$= 24 + 5 + 0$$

$$= 29$$

Consider the path $i=3$:-

Make the 1st row and 3rd column of A to ∞ and set $[3,1] = \infty$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 4 & \infty & \infty & 0 & 1 \\ \infty & 0 & \infty & 0 & 4 \\ 5 & 5 & \infty & \infty & 0 \\ 0 & 4 & \infty & 0 & \infty \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix}$$

$$\begin{matrix} 0 & 0 & 0 & 0 \end{matrix}$$

$\therefore x = 0 + 0 = 0$

$\hat{c}(3) = \hat{c}(1) + A[1,3] + x$

$= 24 + 4 + 0$

$= 28$

Consider the path 1-4 :-

Make the 1st row, 4th column of A to ∞, set $A[4,1] = \infty$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 4 & \infty & 4 & \infty & 1 \\ 3 & 0 & \infty & \infty & 4 \\ \infty & 5 & 0 & \infty & 0 \\ 0 & 4 & 0 & \infty & \infty \end{bmatrix} \begin{matrix} 1 \\ 0 \\ 0 \\ 0 \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 3 & \infty & 3 & \infty & 0 \\ 3 & 0 & \infty & \infty & 4 \\ \infty & 5 & 0 & \infty & 0 \\ 0 & 4 & 0 & \infty & \infty \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$\therefore x = 1 + 0 = 1$

$\hat{c}(4) = \hat{c}(1) + A[1,4] + x$

$= 24 + 3 + 1$

$= 28$

Consider the path 1-5 :-

Make the 1st row, 5th column of A to ∞ and set $A[5,1] = \infty$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 4 & \infty & 4 & 0 & \infty \\ 3 & 0 & \infty & 0 & \infty \\ 5 & 5 & 0 & \infty & \infty \\ \infty & 4 & 0 & 0 & \infty \\ 3 & 0 & 0 & 0 & \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 4 & 0 & \infty \\ 0 & 0 & \infty & 0 & \infty \\ 2 & 5 & 0 & \infty & \infty \\ \infty & 4 & 0 & 0 & \infty \end{bmatrix}$$

$\therefore x = 0 + 3 = 3$

$\hat{c}(5) = \hat{c}(1) + A[1,5] + x = 24 + 0 + 3 = 27$

Now, as cost of node 5 is minimum, Hence we will select node 5 as E-node and generate its children 2, 3, 4

$$\therefore A = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 4 & 0 & \infty \\ 0 & 0 & \infty & 0 & \infty \\ 2 & 5 & 0 & \infty & \infty \\ \infty & 4 & 0 & 0 & \infty \end{bmatrix}$$

Consider the path 5-2 :-

Make the 5th row and 2nd column of A to ∞ and set $A[2,1] = \infty$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 4 & 0 & \infty \\ 0 & \infty & \infty & 0 & \infty \\ 2 & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & & 0 & 0 & \end{bmatrix} \begin{matrix} \\ 0 \\ 0 \\ 0 \\ \\ \end{matrix}$$

$\therefore r = 0 + 0 = 0$

$c(2) = c(5) + A[5,2] + r$

$= 27 + 4 + 0$

$= 31$

Consider the path 5-3 :-

Make the 5th row and 3rd column of A to ∞ and set $A[3,1] = \infty$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 0 & \infty \\ \infty & 0 & \infty & 0 & \infty \\ 2 & 5 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \begin{matrix} 0 \\ 0 \\ 2 \\ \\ \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 0 & \infty \\ \infty & 0 & \infty & 0 & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & & 0 & \end{bmatrix}$$

$$x = 2 + 0 = 2$$

$$\therefore \hat{c}(3) = \hat{c}(5) + A[5,7] + x$$
$$= 27 + 0 + 2$$
$$= 29$$

Consider the path 5 - 4 :-

Make the 5th row, 4th column of A to ∞ and set A[4,1] = ∞

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 4 & \infty & \infty \\ 0 & 0 & \infty & \infty & \infty \\ \infty & 5 & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \begin{matrix} 1 \\ 0 \\ 0 \\ \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & 7 & \infty & \infty \\ 0 & 0 & \infty & \infty & \infty \\ \infty & 5 & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

$$\therefore x = 1 + 0 = 1$$

$$\therefore \hat{c}(4) = \hat{c}(5) + A[5,4] + x$$
$$= 27 + 0 + 1$$
$$= 28$$

Now, As cost of node 4 is minimum, Hence we will select 4 as E-node and Generate its children 2, 7

$$\therefore A = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & 7 & \infty & \infty \\ 0 & 0 & \infty & \infty & \infty \\ \infty & 5 & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

Consider the path 4 - 2 :-

Make the 4th row and 2nd column of A to ∞ and set A[2,1] = ∞

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 3 & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \begin{matrix} 3 \\ 0 \end{matrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ 0 \end{matrix}$$

$\therefore x = 3 + 0 = 3$

$\hat{c}(2) = \hat{c}(4) + A[4,2] + x$

$= 28 + 5 + 3 = 36$

Consider the path 4 - 3 :-

Make the 4th row and 3rd column of A to ∞ and set $A[3,1] = \infty$

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & & & \end{bmatrix} \begin{matrix} \\ 0 \\ 0 \\ \\ \\ \end{matrix}$$

$\therefore x = 0 + 0 = 0$

$\hat{c}(3) = \hat{c}(4) + A[4,3] + x$

$= 28 + 0 + 0$

$= 28$

Now, As cost of node 3 is minimum, Hence we will select 3 as E-node and Generate its children 2

$$\therefore A = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

Consider the path 1-2 :-

Make the 1st row and 2nd column of A to ∞ and set A[2,1] = ∞

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

$\therefore x = 0 + 0 = 0$

$\hat{c}(2) = \hat{c}(1) + A[1,2] + x$

$= 28 + 0 + 0 = 28$

the minimum cost of the tour is = 28

the minimum cost path is 1-5-4-3-2-1 with cost 28.