UNIT – IV

DESIGN AND ANALYSIS OF ALGORITHMS

BRANCH AND BOUND

BASIC METHOD

Branch and Bound

Branch-and-Bound refers to all state space search method in which all children of the E-node are generated before any other live node can become the E-node. In BFS and D, the exploration of new node cannot begin until the node currently begin explored is fully explored.

BFS like state space search will be called FIFO search as the list of live nodes is the First-in first-out list. A D-search like state space search will be called LIFO search as the list of live nodes is a Last-in-First-out (stack).

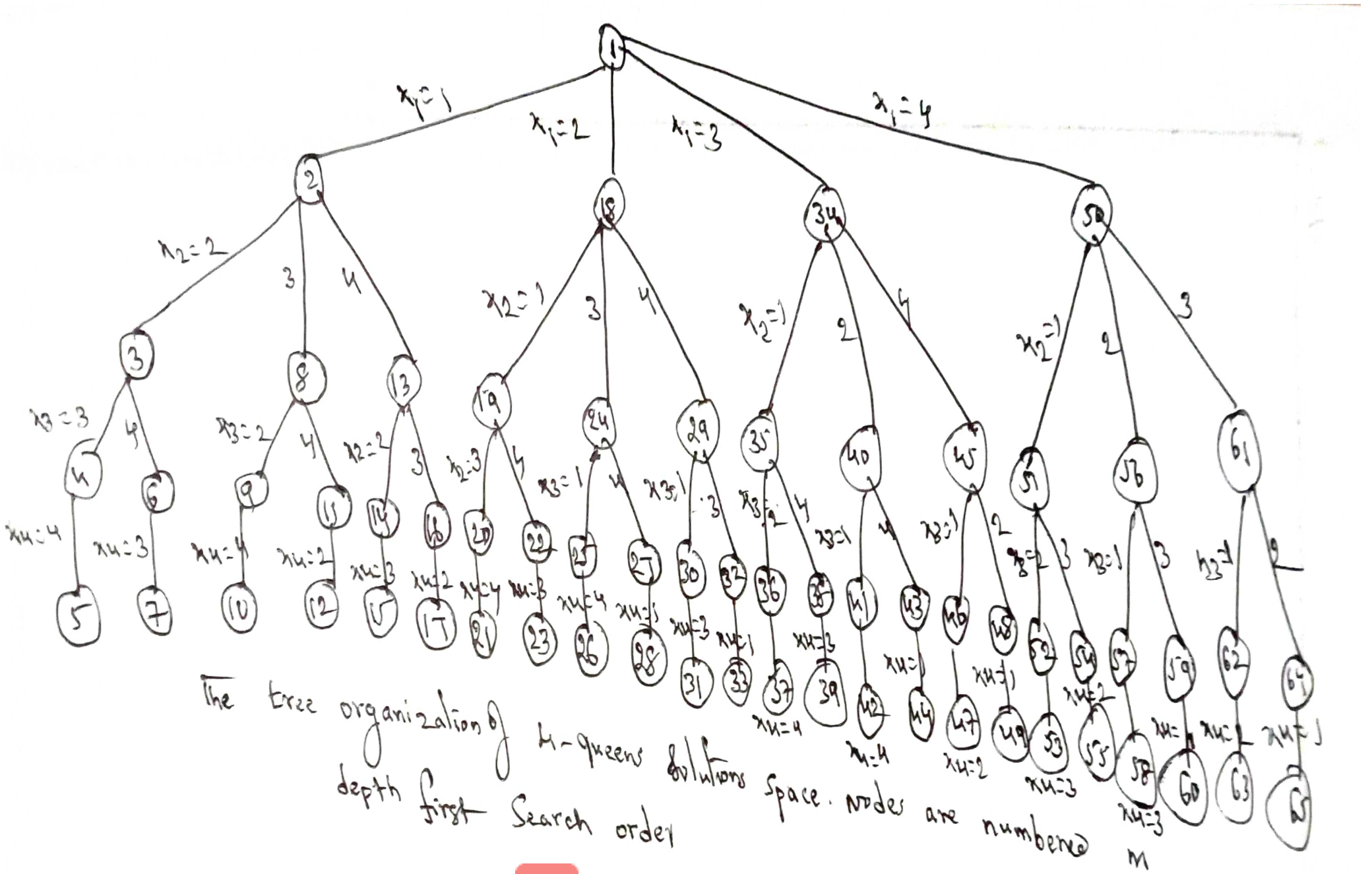
Ex: [4-queens]: we will see how FIFO Branch-and-Bound algo would search the state space tree of 4-queens problem. Initially there is only one live node 1. (no queen has been placed on the chessboard. This node become the E-node. It is expanded as the children "2, 18, 34 and 50" are generated. These nodes represents a chess board with queen 1 in row 1 and columns 1, 2, 3 and 4 respectively.

BASIC METHOD

The only live nodes are now 2, 18, 34 and 50. The next E-node is node 2. It is expanded as nodes 3, 8 and 13 are generated; node 3 is immediately killed using the bounding function. 8 and 13 are added to queue of live nodes. Node 18 becomes the next E-node. Nodes 19, 24 and 29 are generated. Nodes 19 and 24 are killed as a result of bounding function. Node 29 is added to the queue.

The next E-node is node 34. Below fig (1) shows the portion of the 4-queens state-space tree ~~that~~ is generated by FIFO Branch and Bound. Nodes that are killed as a result of bounding function have a "B" under them. Number inside the node corresponds to the state-space tree of 4-queens problem, given in Depth first Search order. Numbers outside the nodes given the order in which the nodes are generated by FIFO branch-and-bound. At the time the answer node 31 is reached, ~~the~~ only live node are 38 and 54.

4-Queens Problem



The tree organization of 4-queens solutions space. Nodes are numbered depth first search order

BASIC METHOD

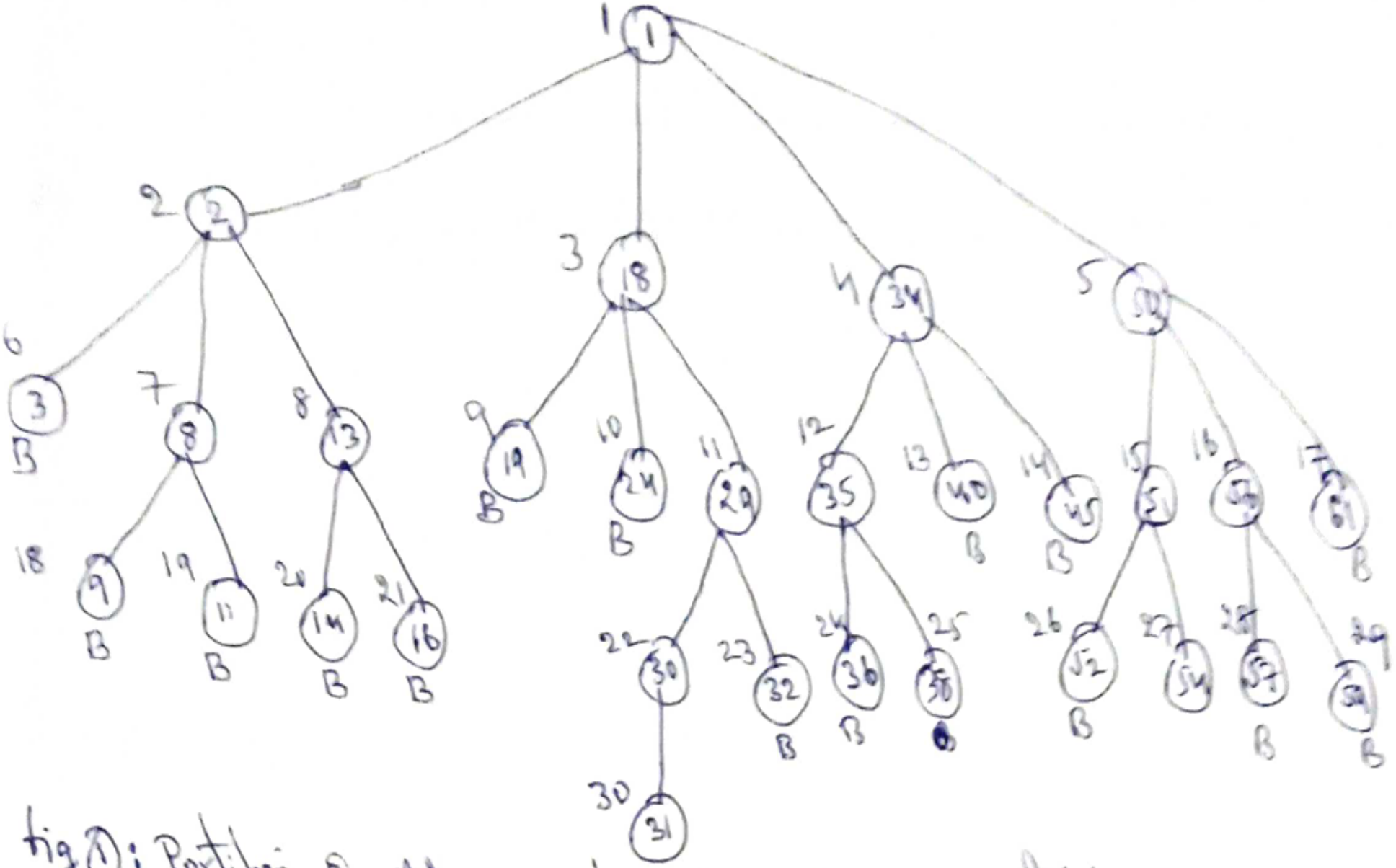


fig 1: Portion of state-space tree of 4-Queens generated by FIFO Branch-and-Bound

Least Cost (LC) Search

Least Cost (LC) Search :

In both FIFO and LIFO branch-and-bound the selection rule for the next E-node is rather rigit and in a sense blind. It doesn't give any preference to a node that has a very good chance of getting the search to an answer node quickly. When node 30 is generated, it should have become obvious to the search algorithm that this node will lead to an answer node in one move. However the rigid FIFO requires the expansion of all live nodes generated before node 30 was expanded.

The search for an answer node can often be speeded by using an "intelligent" ranking function $\hat{c}(\cdot)$ for live node. The next E-node is selected on the basis of ranking function. On the 4-queens ex, we use a ranking function that assign node 30 a better rank than all other live nodes that node 30 will become the E-node following 29. The remaining live nodes will never become the E-node as the expansion of node 30 results in generation of an answer node (node 31).

Least Cost (LC) Search

For any live node x, the computational effort could be

① The number of nodes in the subtree x that need to be generated before an answer node is generated (or)

② The number of levels of the nearest answer node is from x.

Using cost measure ②, the cost of the root of the tree in the above fig is 4 (node 31 is 4 levels away from 1). Using these costs as before to select next E-node, the E-nodes are nodes 1, 18, 29, and 30, the other nodes get generated as 2, 34, 50, 19, 24, 32 and 31.

If cost measure ① is used, then the search would always generate the minimum number of nodes on the path from the root to the nearest answer node. If the cost measure ② is used, then the only nodes to become E-node are the nodes on the path from root to the nearest answer node.

Least Cost (LC) Search

Search algorithm usually rank nodes only on the basis of an estimate $\hat{g}(\cdot)$ of their cost.

$\hat{g}(\cdot)$ is the estimate of the additional effort needed to reach the answer node from x. Node x is assigned a rank using a function $\hat{c}(\cdot)$. such that

$$\hat{c}(x) = f(h(x)) + \hat{g}(x)$$

where $h(x)$ is the cost of ranking x from the root and $f(\cdot)$ is any non-decreasing function

$$\hat{g}(y) \leq \hat{g}(x), \text{ } y \text{ is child of } x$$

A search strategy that uses a cost function $\hat{c}(x) = f(h(x)) + \hat{g}(x)$ to select next E-node would always choosen for its next E-node a live node with least $\hat{c}(\cdot)$. Hence such a strategy is called an LC-search.

BFS and D are special cases of LC-search

- If $\hat{g}(x) = 0$, $f(h(x)) =$ level of node x, LC-search generate nodes by level (BFS)
- If $f(h(x)) = 0$ and $\hat{g}(x) \geq \hat{g}(y)$ if y is child of x, D-search

Least Cost (LC) Search

Control abstraction for LC_search()

```
Algorithm LC_search(t)
{
If *t is an answer node then output *t and return;
    E := t   // E-node
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 is a new live node
        (x→parent := E)    // Points path to the root
    }
    if there are no live nodes then
    {
        write("no answer node");
        return;
    }
    E := least()
} until(false);
```

LC Branch and Bound

LC Branch-and-Bound

The LC Branch-and-Bound solution can be obtained using fixed tuple size formulation.

The steps to be followed for LC Branch & Bound solution are:

1. Draw the state space tree
2. Compute $\hat{c}(\cdot)$ and $u(\cdot)$ for each node
3. If $\hat{c}(x) >$ upper, kill the node x
4. Otherwise, the minimum cost $\hat{c}(x)$ becomes the next E-node. Generate children for E-node.
5. Repeat steps 3 and 4 until all the nodes get covered
6. The minimum cost $\hat{c}(x)$ becomes the answer node. Trace the path in backward direction from x to root for solution subset.

LC Branch and Bound

Ex: Consider the knapsack instance $n=4$ with capacity $m=15$ such that

object i	P_i	w_i
1	10	2
2	10	4
3	12	6
4	18	9

Sol: Let we design state space tree using fixed tuple size formula. Computation of $\hat{c}(x)$ and $u(x)$ for each node x is required.

$u(1)$ can be computed as $u(i) = -\sum P_i$

$-\sum P_i = -(10+10+12) = -32$ [we can select only first 3 items]

$u(1) = -32$

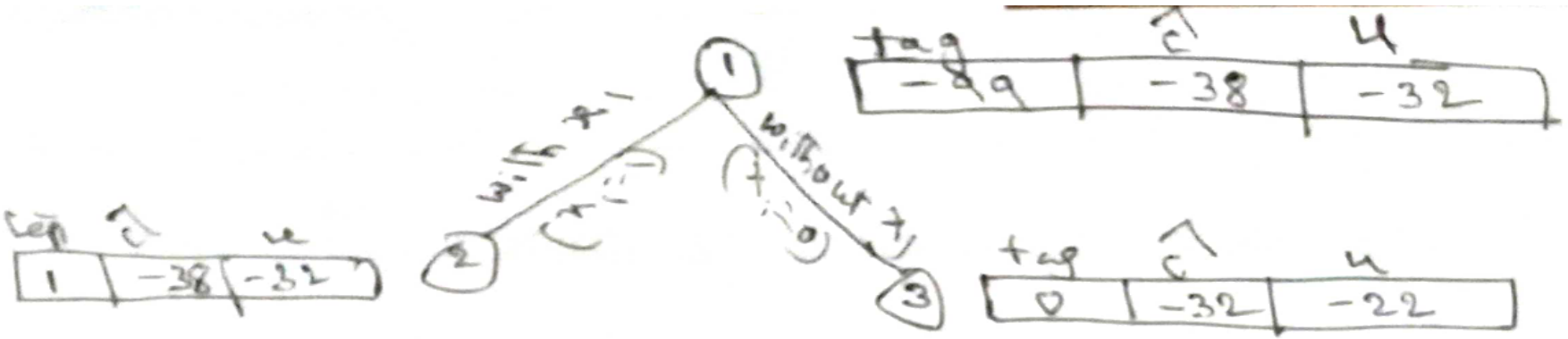
If we select 4th item, weight (21) exceeds capacity of knapsack.

$$\hat{c}(i) = u(i) - \left[\frac{m - \text{current totalweight}}{\text{Actual weight of remaining objects}}\right] * \left[\text{Actual profit of remaining object}\right]$$

$$\hat{c}(i) = -32 - \left[\frac{15-(2+4+6)}{9}\right] * 18 = -32 - \frac{3}{9} * 18^2$$

$$= -32 - 6 = -38$$

LC Branch and Bound



for node 2: we consider first item.

$u(2) = -\sum p_i = -32$

$\hat{c}(2)$: $-32 - \left[\frac{W - 12}{9}\right] * 18 = -32 - \left[\frac{3}{9}\right] * 18$

$= -32 - 6 = -38$

w_i
2 ✓
4 ✓
6 ✓
9

for node 3: we omitted first item

$u(3) = -\sum p_i = -(10+12) = -22$

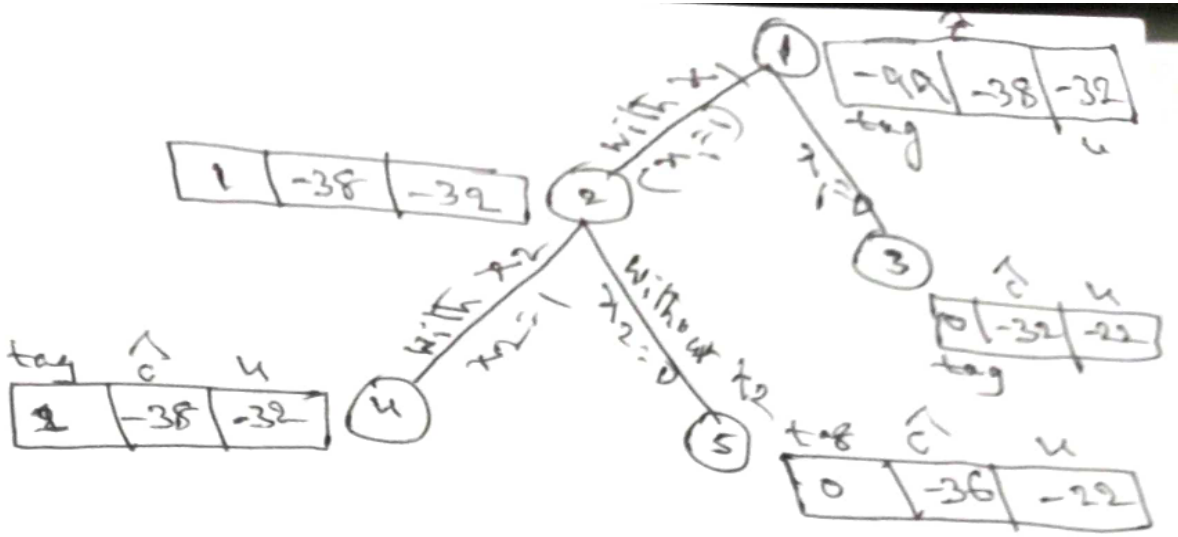
$\hat{c}(3) = -22 - \left[\frac{15 - 10}{9}\right] * 18$

$= -22 - [5] * 18 = -22 - 10 = -32$

w_i
~~2~~
4 ✓
6 ✓
9 ✗

LC Branch and Bound

w_i	p_i
2	10
4	10
6	12
9	18



for node 4 :

$$u[4] = -\sum p_i = -32$$

$$\hat{c}(4) = -32 - \left[\frac{15-12}{9}\right] * 18 = -32 - \frac{3}{9} * 18^2 = -38$$

for node 5 :-

$$u[5] = -\sum p_i = -22$$

$$\hat{c}[5] = -22 - \left[\frac{15-8}{9}\right] * 18$$

$$= -22 - \left[\frac{7}{9}\right] * 18^2 = -22 - 14 = -36$$

select node 4 as next E-node

LC Branch and Bound

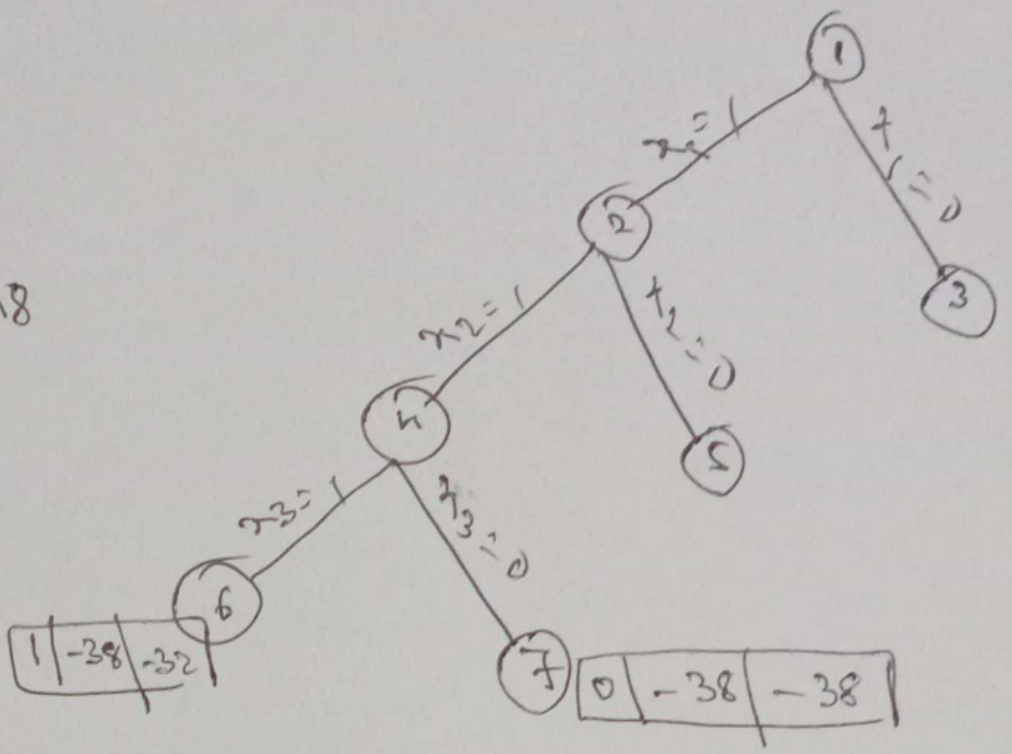
For node 6:

$u[6] = -32$

$\hat{c}[6] = -32 - \left[\frac{15-12}{9}\right] * 18$

$= -32 - \frac{3}{9} * 18^2$

$= -38$



For node 7 :

$u[7] = -38$

$c[7] = -38 - \left[\frac{15-15}{0}\right] * 0$

$= -38$

Costs of both the nodes are same. so Compare upper bounds node 7 upper bound is less than node 6. So E-node is 7.

LC Branch and Bound

For node 8:

$u[8] : -38$

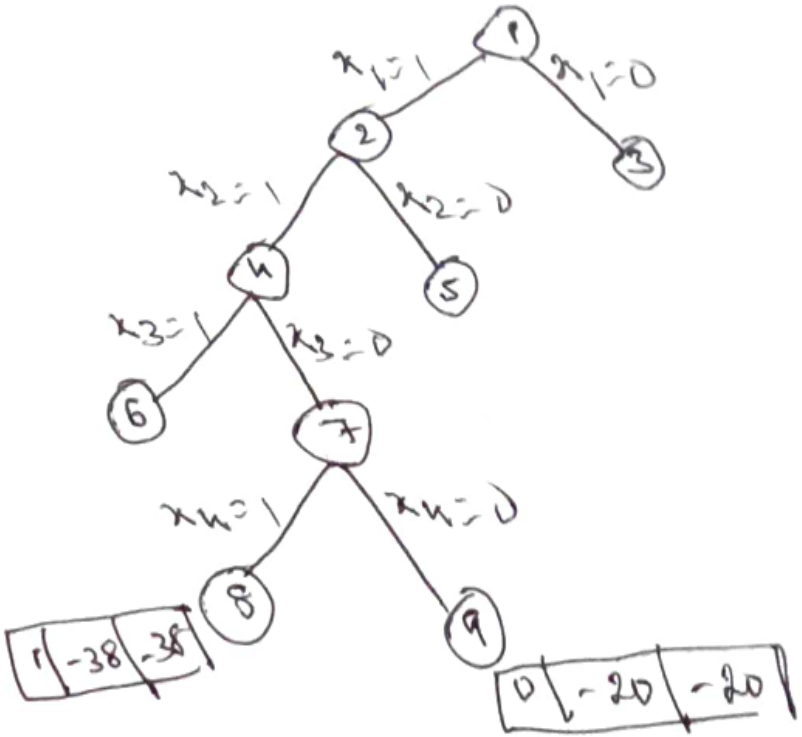
$\hat{c}[8] : -38 - \left[\frac{15-15}{0}\right] 0$

$= -38$

$u[4] : -20$

$\hat{c}[4] : -20 - \left[\frac{15-6}{0}\right] *0$

$= -20$



So, answer node is 8 and

solution is $x_1=1,\ x_2=1,\ x_3=0,\ x_4=1$

Profit obtained is : $10+10+0+18 = 38$

Total weight included is $= 2+4+0+9 = 15$

Travelling Salesperson Problem

Travelling Salesperson Problem (TSP)

It there are n-cities and cost of travelling from any city to any other city is given. Then we have to obtain the cheapest round-trip such that each city is visited exactly once and returning to the starting city, completes the tour.

TSP can be represented by weighted graph.

$$G = \begin{bmatrix} \infty & 20 & 30 & 10 \\ 15 & \infty & 16 & 4 \\ 3 & 5 & \infty & 2 \\ 19 & 6 & 18 & \infty \\ 16 & 4 & 7 & 16 \end{bmatrix} \begin{matrix} 10 \\ 2 \\ 2 \\ 3 \\ 4 \end{matrix}$$

Row wise minimum values

21

Red_Row[n] can be obtained as

$$\text{Red\_Row}[n] = \begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{bmatrix}$$

1 ignore 3 ignore ignore = 4

Travelling Salesperson

Red_col[m] can be obtained as

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

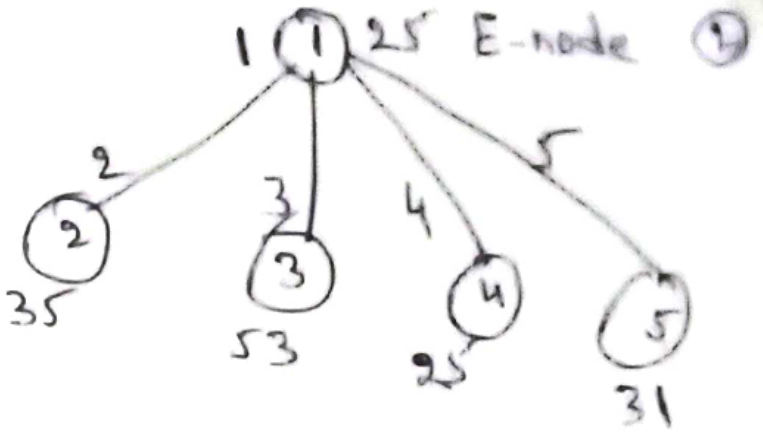
← fully reduced matrix

Total reduced cost will be

= Cost (Red_Row[m] + Cost (Red_col[m])

= 21 + 4 = 25

Travelling Salesperson



Consider Path 1,2 : make 1st row and 2nd column all entries ∞, and set M[2][1] = ∞

∞	∞	∞	∞	∞	ignore
∞	∞	11	2	0	ignore
0	∞	∞	0	2	ignore
15	∞	12	∞	0	ignore
11	∞	0	12	∞	ignore
	ignore	ignore	ignore	ignore	

Cost of node 2 is = Cost of its parent + row wise Reduction Sum + column wise Reduction Sum + M[1][2]

= 25 + 0 + 0 + 10

= 35

Travelling Salesperson

for path 1,3: Make 1st row and 3rd column entries are ∞ and also set m[3][1] as ∞.

∞	∞	∞	∞	∞	← ignore
12	∞	∞	2	0	← ignore
∞	3	∞	0	2	← ignore
15	3	∞	∞	0	← ignore
11	0	∞	12	∞	← ignore
↑	↑	↑	↑	↑	
11	ignore	ignore	ignore	ignore	

No row wise reduction

Reduced Matrix =

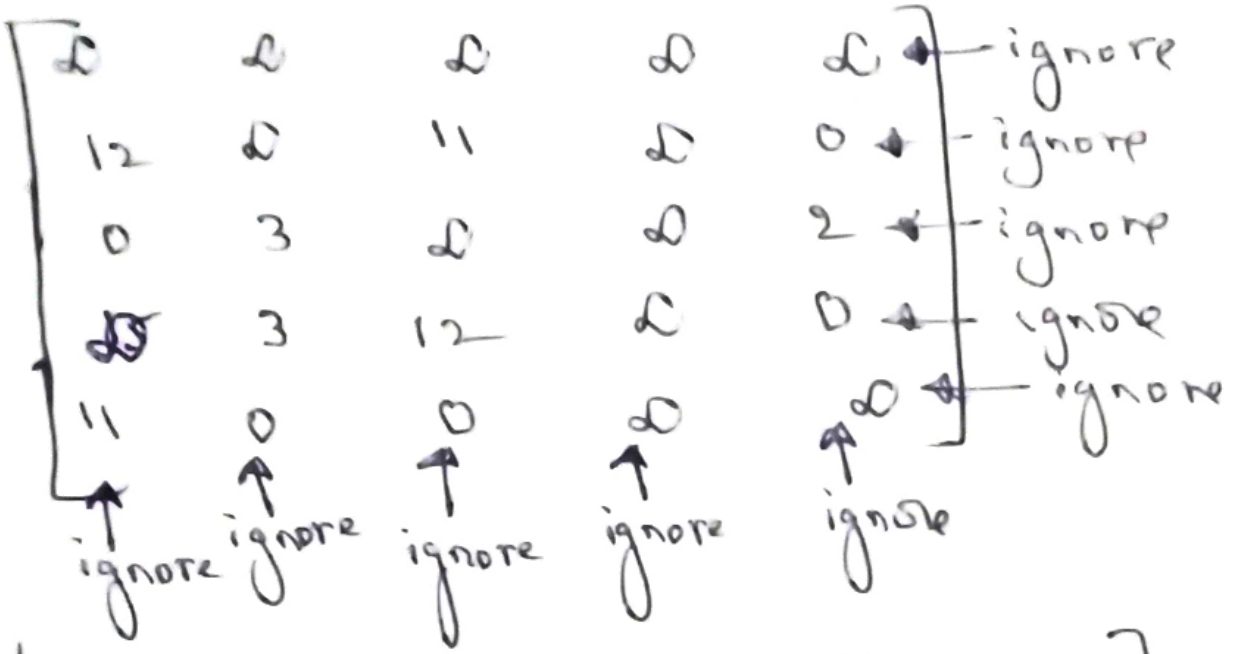
∞	∞	∞	∞	∞
1	∞	∞	2	0
∞	3	∞	0	2
4	3	∞	∞	0
0	0	∞	12	∞

Cost of node 3 = 25 + 0 + 11 + 17 (m[1][3])

= 53

Travelling Salesperson

For path 1, 4: make 1st row, and 4th columns are ∞ and also set m[4][1] as ∞.

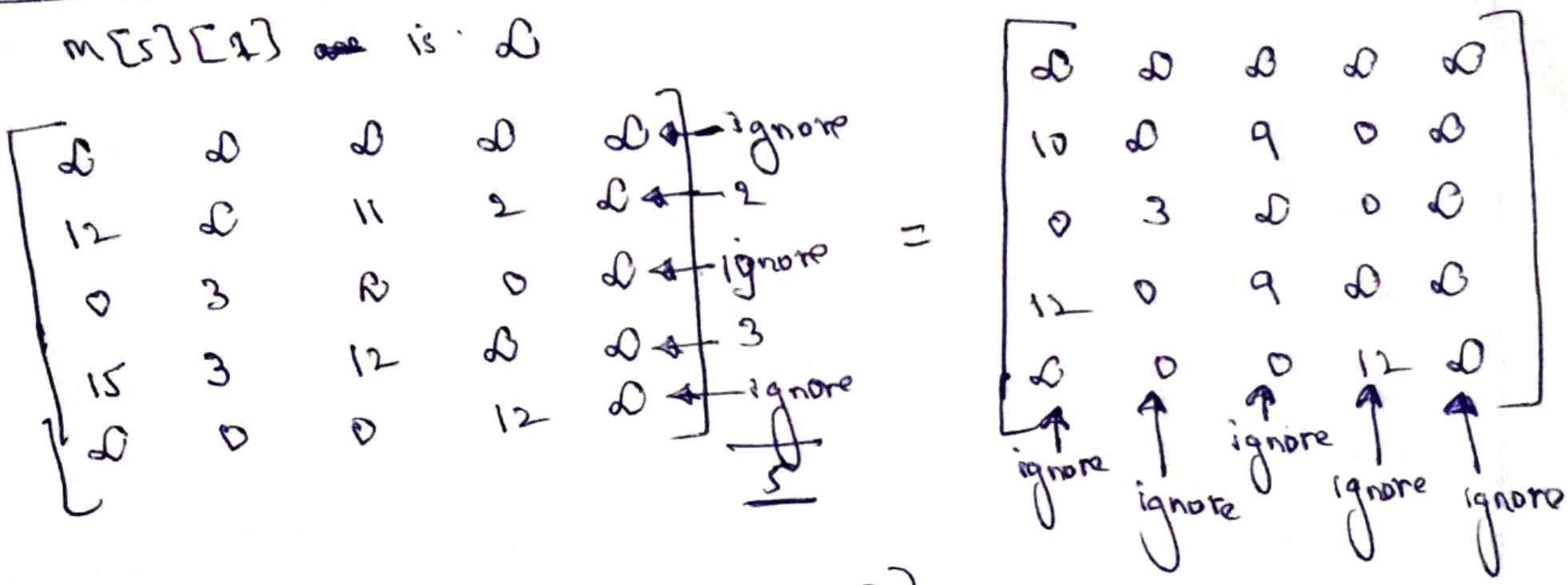


Cost of node 4 = 25 + 0 + 0 + 0 [m[1][4]]

= 25

Travelling Salesperson

For path 1,5: make 1st row, and 5th column are ∞ and also set m[5][1] is ∞



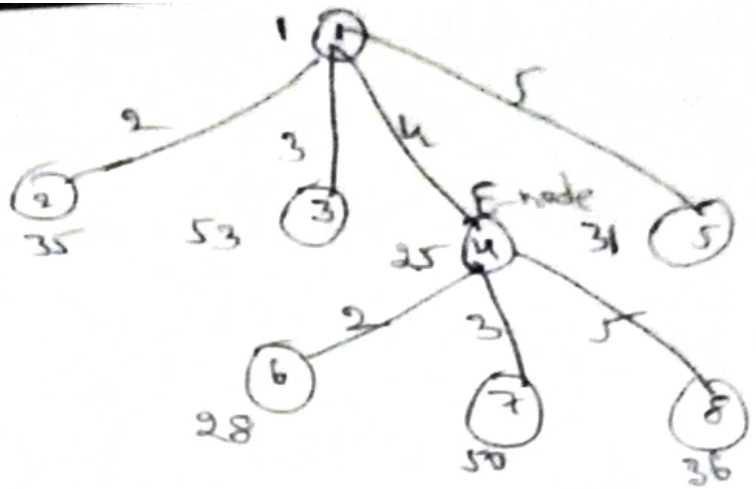
Cost of node 5 = 25 + 5 + 0 + 1 [m[1][5]]

= 31

Now, as cost of node 4 is minimum, we will select node 4 as next E-node and generate its children node 6, 7 and 8

Travelling Salesperson

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



* Consider Path 1, 4, 2 for node 6:

Set 1st row, 4th row and 2nd column are ∞ and also set $m[4][1]$ & $m[2][1]$ are ∞. also 4th column as ∞

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

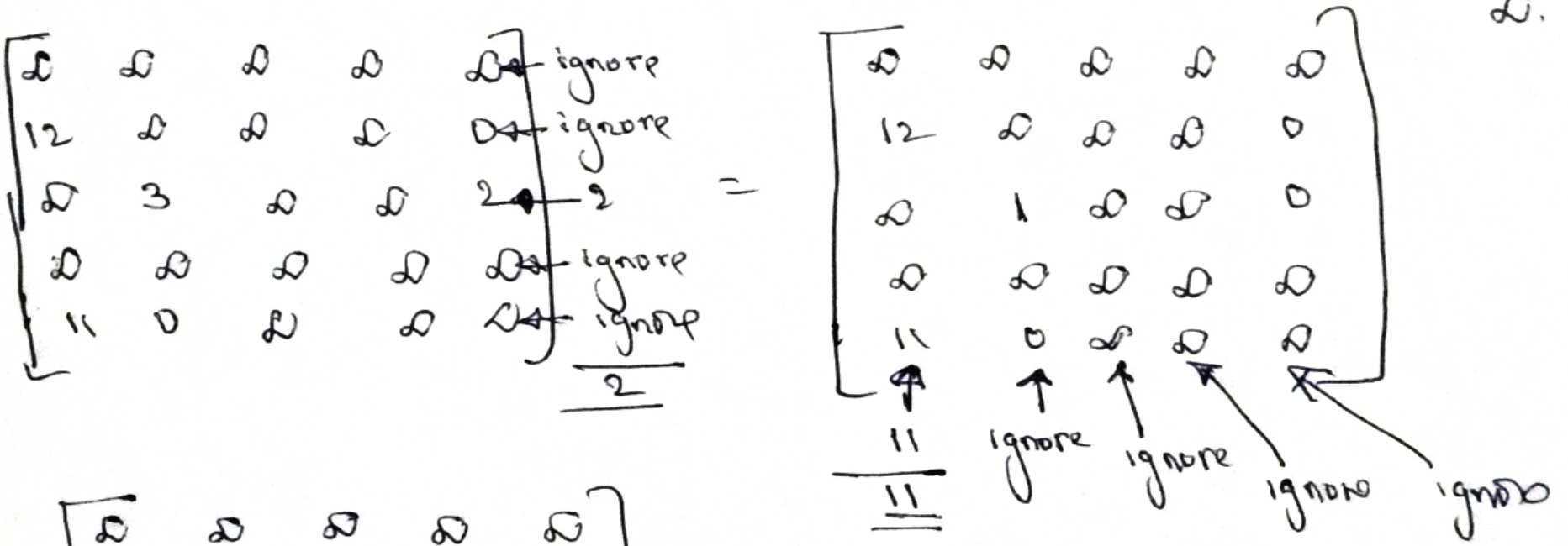
ignore, ignore, ignore, ignore, ignore, ignore, ignore, ignore, ignore, ignore.

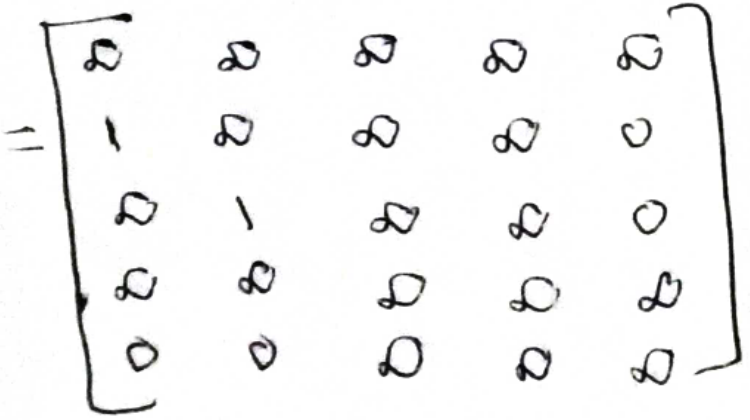
Cost of node 6 = $25 + 0 + 0 + 3$ [$m[4][2]$]

= 28.

Travelling Salesperson

*Consider path 1,4,3 for node 7: set 1st row, 4th row and 3rd column as ∞ and also make m[4][1] = ∞, m[3][1] = ∞ also 4th column as ∞.





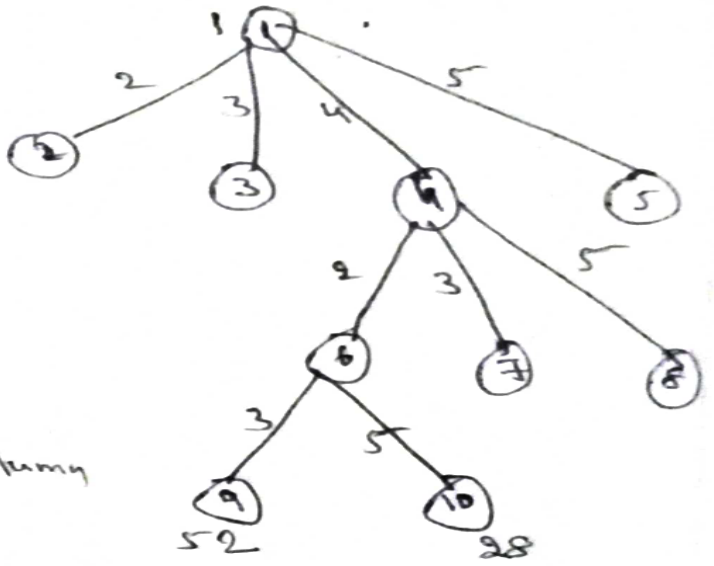
Cost of node 7 = 25 + 2 + 11 + 12 (m[4][3])

= 50

Travelling Salesperson

node 6 matrix

∞	∞	∞	∞	∞
∞	∞	11	∞	0
0	∞	∞	∞	∞
∞	∞	∞	∞	0
11	∞	0	∞	∞



Consider path 1, 4, 2, 3 for node 9:

Make 1st row, 4th row, 2nd row, 4th column, 2nd column and 3rd column are ∞ and also set m[4][1], m[2][1] and m[3][1] are ∞

∞	∞	∞	∞	∞	ignore
∞	∞	∞	∞	∞	ignore
∞	∞	∞	∞	2	2
∞	∞	∞	∞	∞	ignore
11	∞	∞	∞	∞	ignore
					2

=

∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	0
∞	∞	∞	∞	∞
11	∞	∞	∞	∞
↑	↑	↑	↑	↑
11	ignore			

=

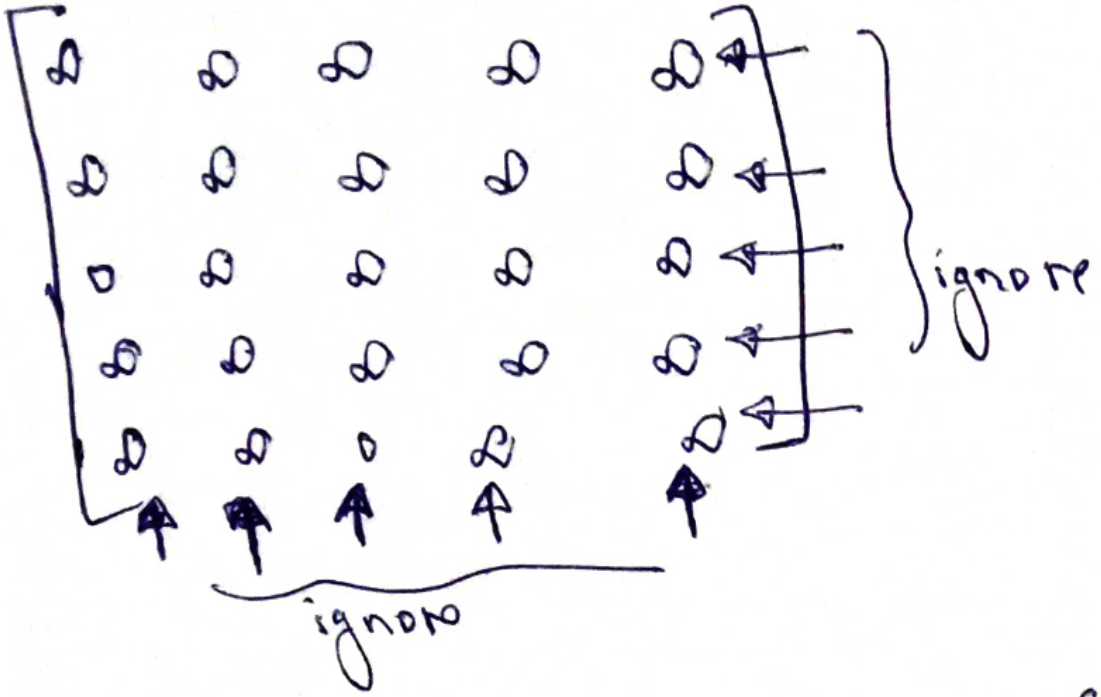
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞
∞	∞	∞	∞	0
∞	∞	∞	∞	∞
∞	∞	∞	∞	∞

Cost of node 9 = 28 + 2 + 11 + 11 (m[2][3])

= 52

Travelling Salesperson

Consider path 1, 4, 2, 5 for node 10. Make 1st row, 4th row, 2nd row and 4th column, 2nd column and 5th column are ∞ and also set m[4][1], m[2][1] and m[5][1] are ∞.



Cost of node 10 = 28 + 0 + 0 + 0 (m[2][5])

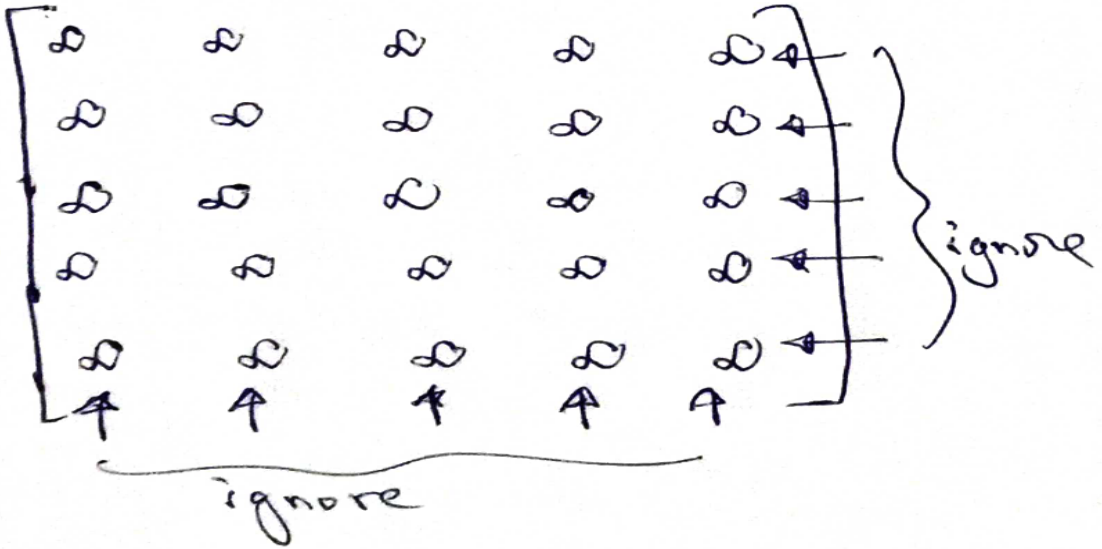
= 28

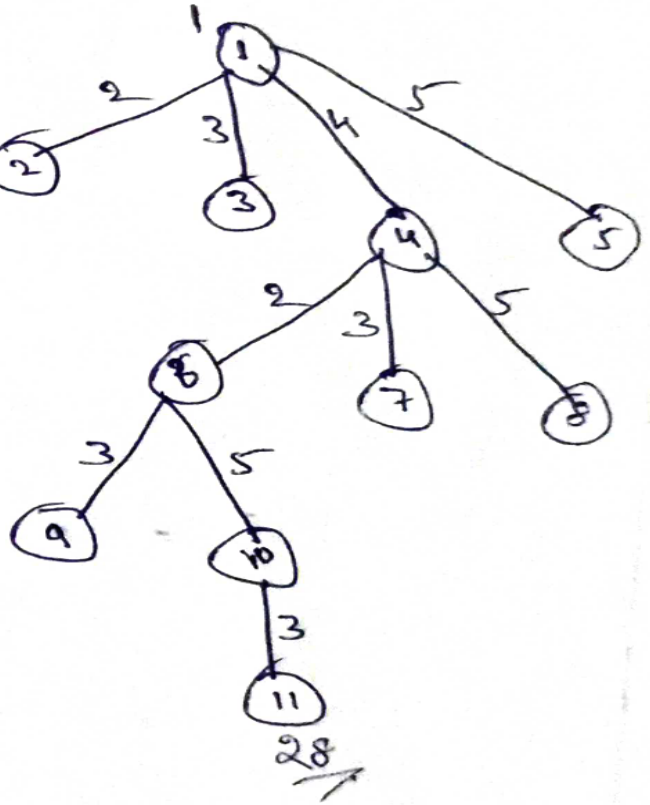
Now node 10 becomes next E-node

Travelling Salesperson

* Now Consider 1, 4, 2, 5, 3 Path for node 11:

Make 1st row, 4th row, 2nd row, 5th row and 4th column, 2nd column, 5th column, 3rd column are ∞ and also set m[4][1], m[2][1], m[5][1] and m[3][1] are ∞.





Cost of node 11 = 28 + 0 + 0 + 0 [m[5][3]]

= 28

Path is: 1 →(10) 4 →(6) 2 →(2) 5 →(7) 3 →(3) 1

28