Unit 4:

Backtracking, Branch & Bound

→ In case of greedy & dynamic programming techniques, we will use Brute Force approach

↓

→ It means we will evaluate all possible solutions, among which we select one solution as optimal either (maximum or minimum).

→ Backtracking also use Brute force approach but here we will get same optimal solution with a less no. of steps (hence it is efficient than greedy & dynamic).

→ In this we will use bounding function (Criterion function, implicit & explicit constraints. (conditions).

→ The major advantage of backtracking method is if a partial solution

$(x_1, x_2 \ldots x_i)$ we are sure that it cannot lead to optimal solution then $(x_{i+1}, \ldots x_n)$ solutions are ignored

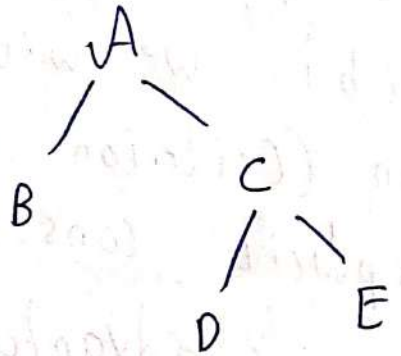
→ Here we follow Depth First Search method.

Terminology:-

i) Criterion function:- It is a function $P(x_1, x_2, \ldots x_n)$ which needs to be maximized or minimized for a given problem

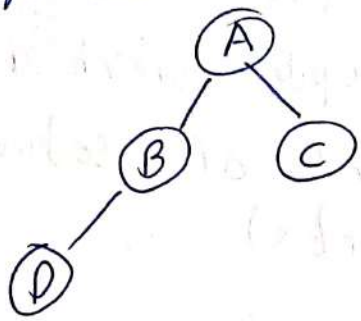
ii) Solution space:- All tuples that satisfy the explicit constrians (to be a solution)

Ex



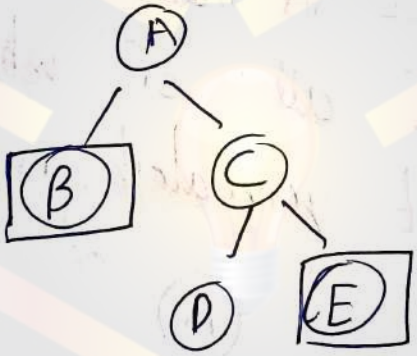
∴ AB, ACD, ACE are tuples in solution space.

iii) Problem state:- each node in the tree ~~organization~~ define a problem state



=> A, B, C, D are problem state.

iv) Solution state:- these are those problem state s for which path from root to s define a tuple in the solution space



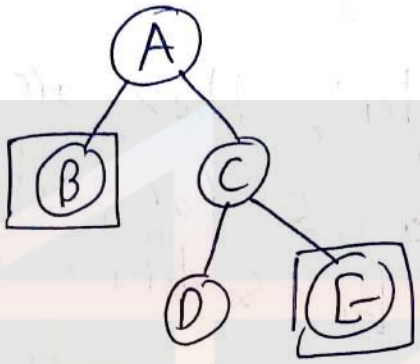
indicates solution

A, B ε.

A, C, E are solution states

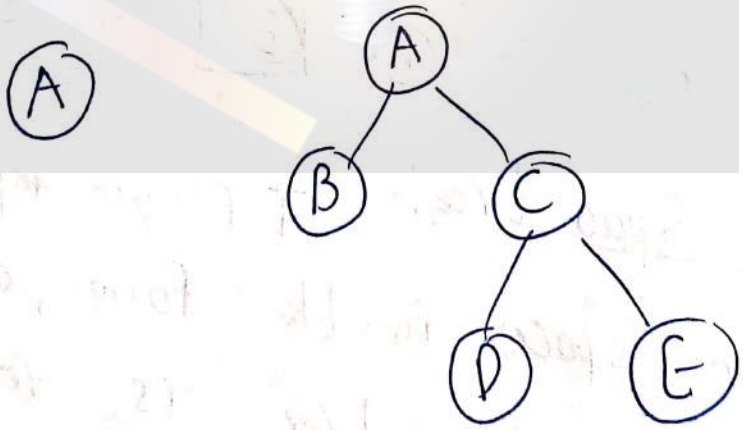
v) State Space tree:- If we represent solution space in the form of a tree then the tree is referred as the state space tree.

vi) Answer state: These solution states s for which the path from the root to s defines a tuple which is a member of the set of solutions (satisfies constraints).



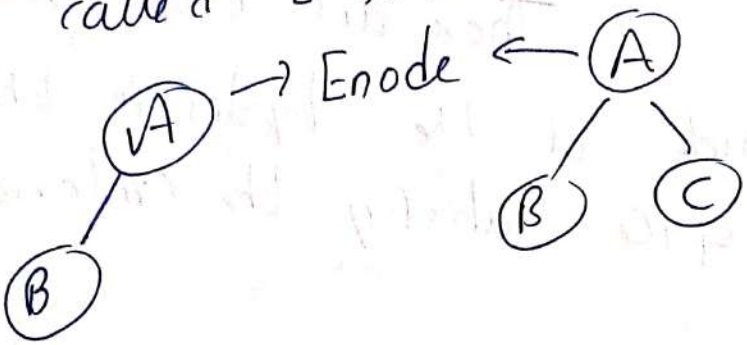
B, E are answer states.

vii) Live node: A node which has been generated & all of whose children have not yet generated is live node.

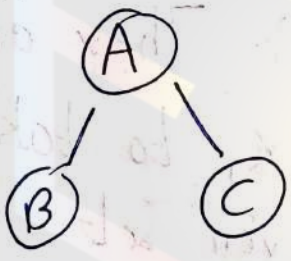


A as it could generated children | B, D, E as it could generate children.

viii) Enode: the live node whose children are currently being generated is called Enode



ix) Read node: It is a generated node that is either not to be expanded further or one for which all its children are generated



A, B, C are dead node
(if B, C does not expand).
(A generated all possible hence it is dead)

x) Bounding Function: is a function used to kill live nodes without generating all their children.

→ Constraints (rules / conditions)

i) Implicit constraints:

These are rules which determine which of the tuples in the solution space satisfy the criterion function.

ex: In 4 queens problem, the implicit constraints are no two queens can be on the same column, same row or same diagonal.

ii) Explicit Constraints: These are rules which restrict each x_i to take on values only from a given set.

ex: in knapsack (0/1)

(i) $x_i = 0$ or 1

(ii) $0 \leq x_i \leq 1$

→ N Queen Problem :-

Consider an n x n chessboard let there are n queens. These n queens are to be placed on the n x n chessboard so that no two queens are on the same column, same row or same diagonal.

```
Algorithm NQueens(k, n)        → row no.
{
    For i ← 1 to n do
    {
        if (place(k, i)) then
        {
            x[k] = i;
            if (k == n) // column no
            then write (x[1:n]);
            else NQueen (k+1, n);
        }
    }
}
```

```
Algorithm Place (k, i)
{
    For j ← 1 to k-1 do
    {
        if ((x[j] == i) or
        ABS(x[j] - i) == ABS(j-k)))
            then return false;
        else
            return true;
    }
}
```

2 in same col

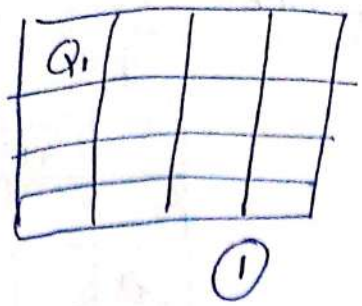
check diagonal

→ 4 Queens Problem :-

→ Consider a 4 × 4 chessboard let there are 4 queens.

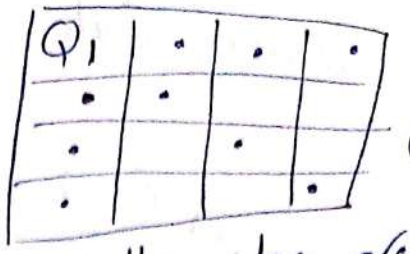
→ The objective is to place there 4 queens on 4 × 4 chessboard in such a way that no two queens queens should be placed in the same row, same column or diagonal position.

ex:



①

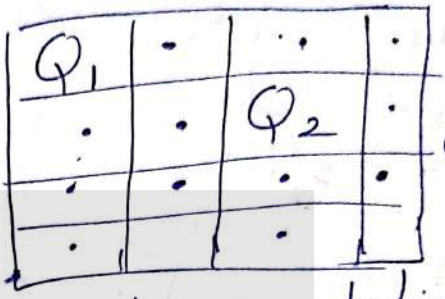
⇒



②

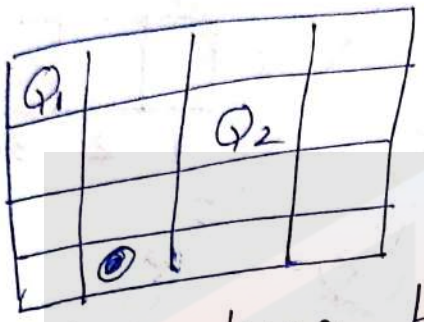
all . place are not allowed

⇓



③

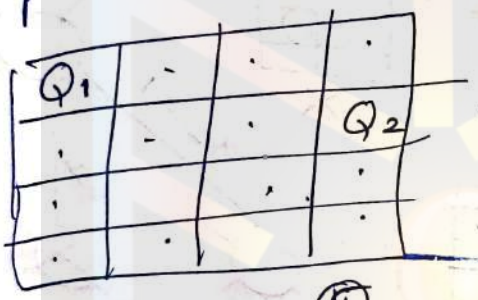
⇐



④

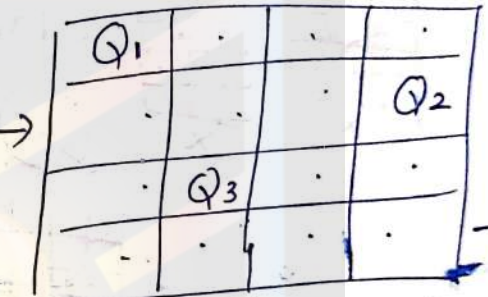
here we cannot place hence this is not a solution

2 ⟶



⑤

→

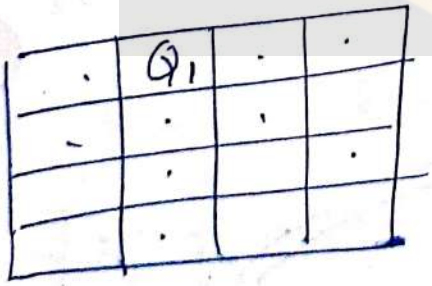


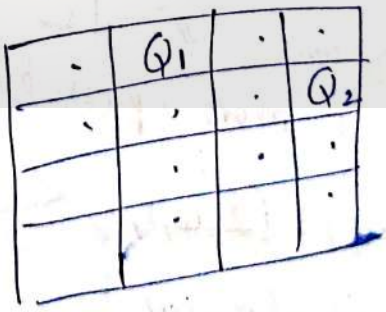
⑥ → block

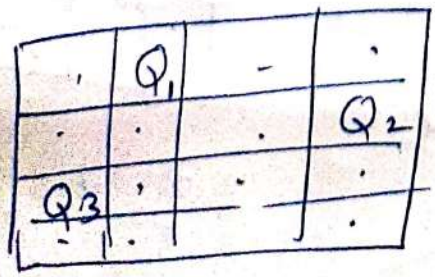
back

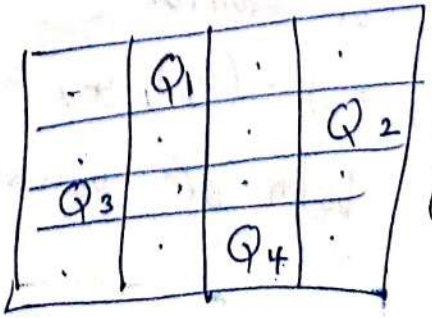
track back to 1

1







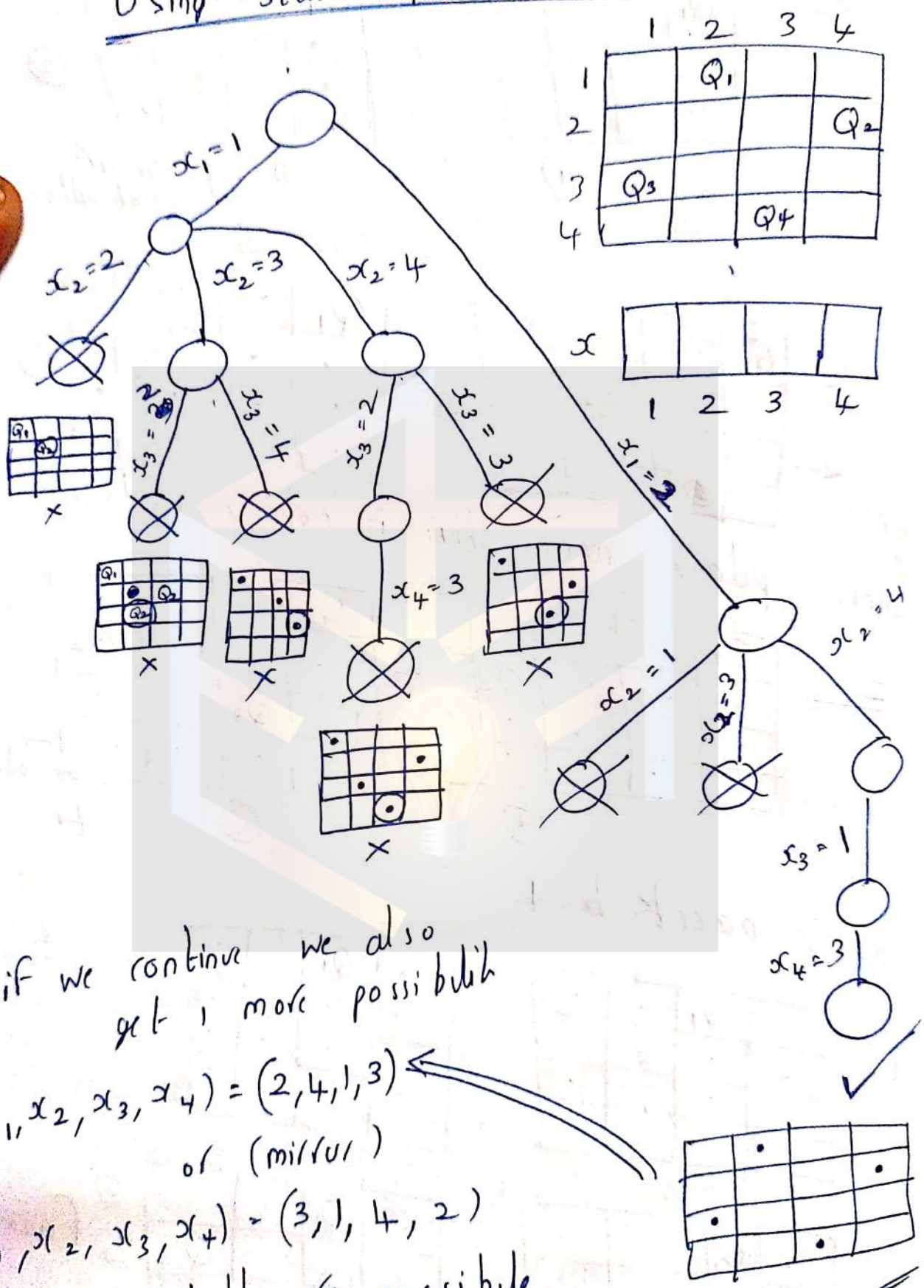


→ Found

(2, 4, 1, 3)

↑ is order

using state space tree.



if we continue we also get 1 more possibility

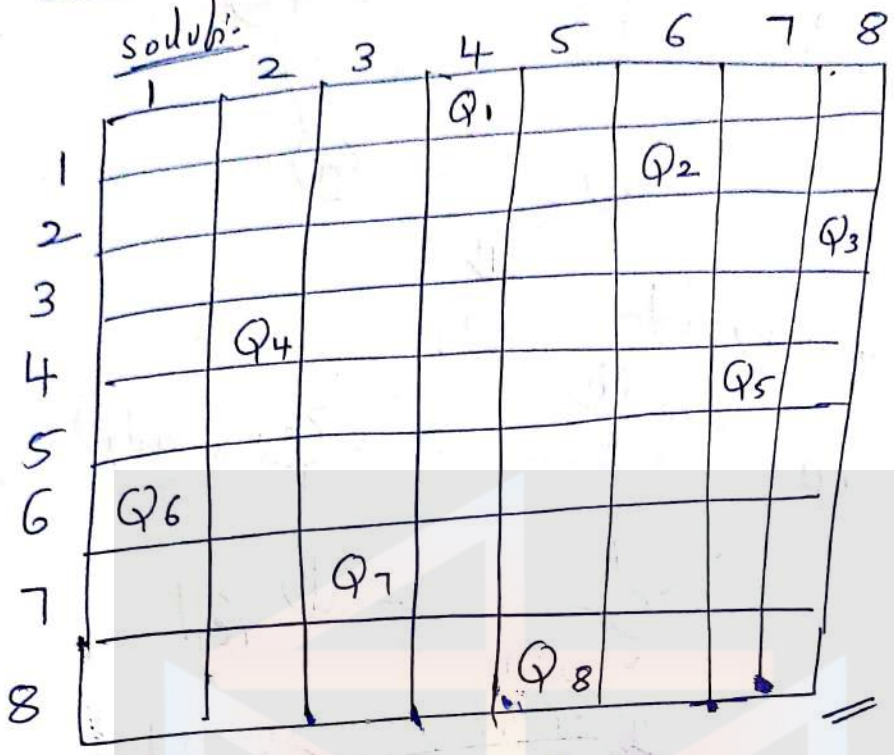
$(x_1, x_2, x_3, x_4) = (2, 4, 1, 3)$

or (mirror)

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

both are possible.

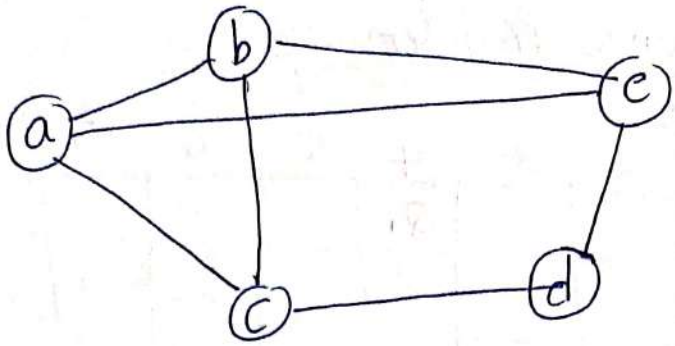
→ 8 Queens Problem :-



→ Graph Coloring problem:

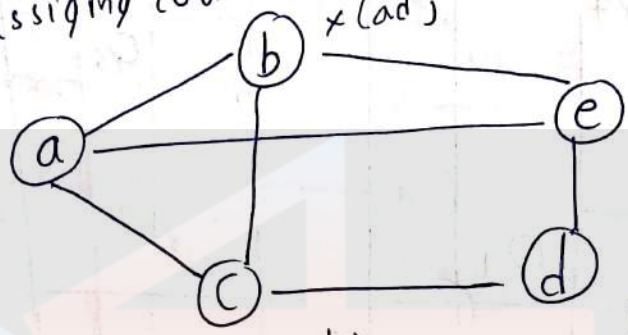
→ Let G = (V, E) be a graph, In graph coloring problem, we have to find out wheather all the vertices of the given graph are colored or not, with the constraint that no two adjacent vertices have the same color.

Sol:

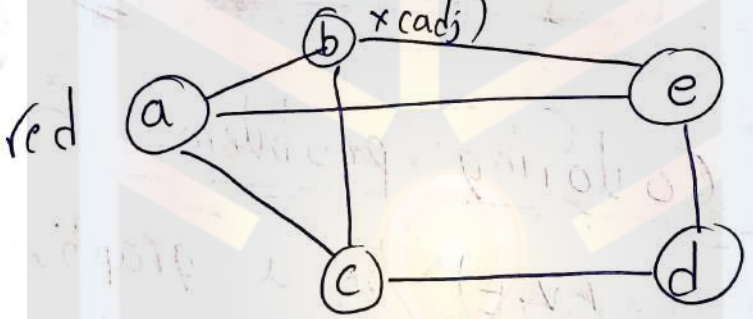


start assigning colors

i) red

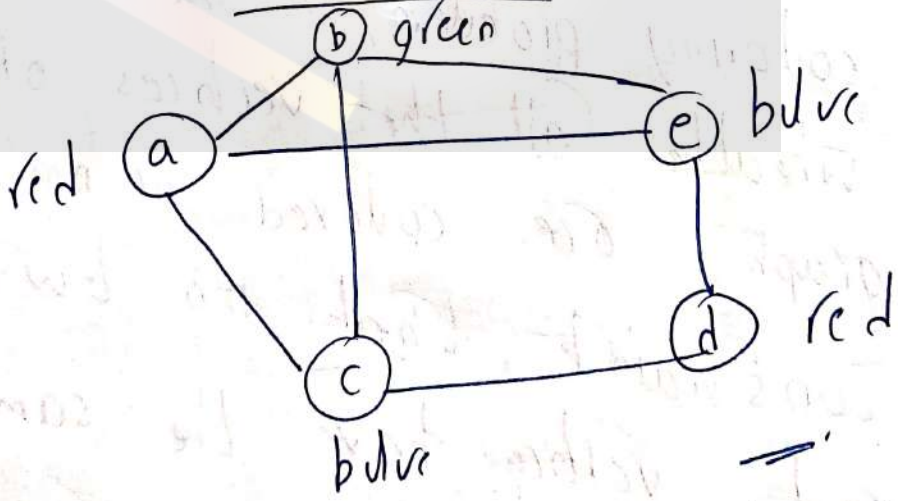


ii) blue



blue

iii) green



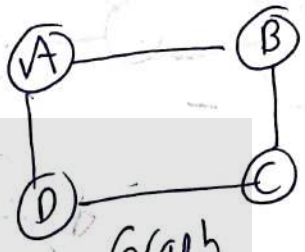
min no. of colors required are 3.

→ Chromatic number : (m)

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

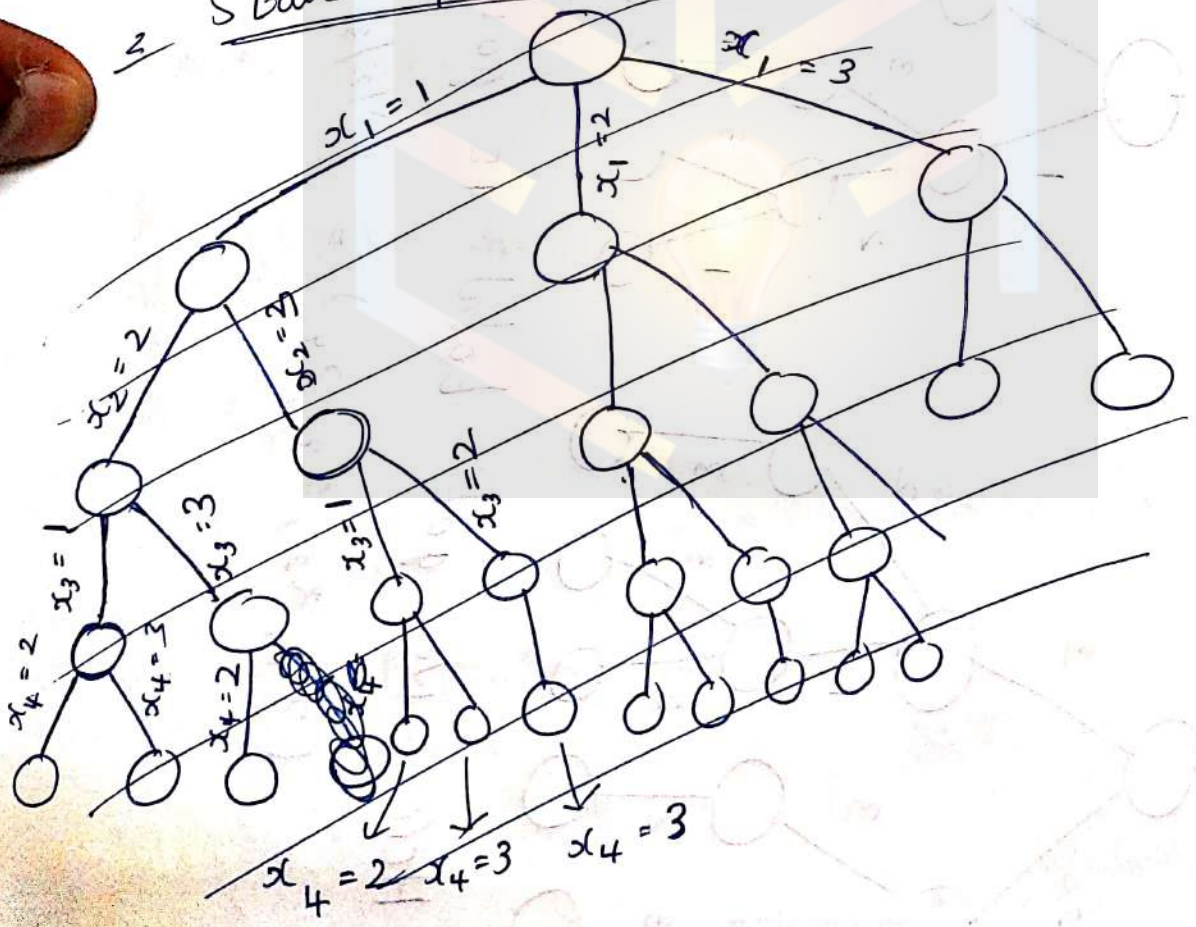
previous example (m = 3)

ex:- m = 3

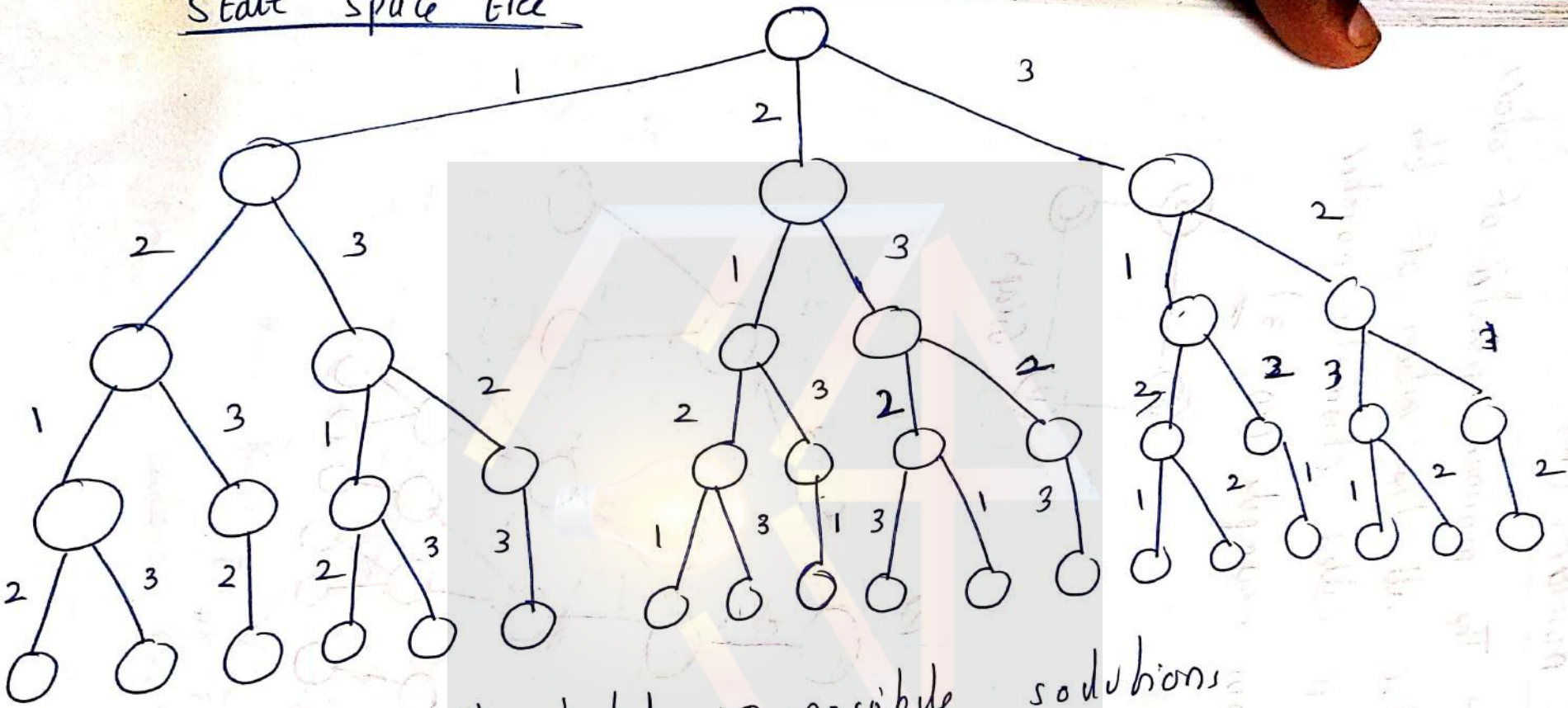


Graph

2. State space tree



State space tree



Here we got total 18 possible solutions

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

```
algorithm  m coloring (k)        → index of array
                                   Initially = 1
{
  repeat
  {
    Next Value (k);
    if (x[k] = 0) then
        return;
    if (k = n) then
        write (x[1:n]);
    else
        m Coloring (k+1);
  } until (false);
}
```

x | 0 | 0 | 0 | 0 ...
1 2 3 4

```
algorithm  Next Value (k)
{
  repeat
  {
    x[k] ← (x[k] + 1) mod (m+1);      → obtaining next vertex.
    if (x[k] = 0) then
      return;
    for j ← 1 to n do
    {
      if ((G[k,j] ≠ 0) & (x[k] = = x[j]))
          then break;
    }
```

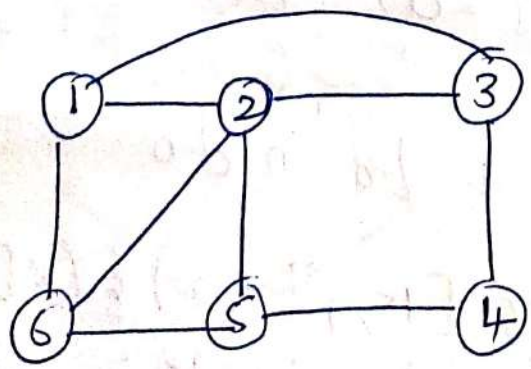
```
    if (j = n+1) then
        return;
    }
    unbid (false);
}
```

→ Hamiltonian Cycles:-

→ In a connected graph a path that visits every vertex of the graph the G exactly once except the starting/ending vertices. it is hamiltonian cycle. (start & end at same vertex.)

→ We could have multiple hamiltonian cycles (finding them is an NP hard problem as we need to consider all cases).

ex)



1, 2, 3, 4, 5, 6, 1 ✓
1, 2, 6, 5, 4, 3, 1 ✓
1, 6, 2, 5, 4, 3, 1 ✓
2, 3, 4, 5, 6, 1, 2 ✗

same

```
algorithm Hamiltonian (k)
{  do
   {  next Vertex (k);
      if (x[k] == 0) then
            return;
      if (k == n) then
            write (x[1:n]);
      else
         Hamiltonian(k+1);
   } while (true);
}
algorithm NextVertex (k)
{
   do
   {
      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)
          if (k<=n or(k == n) &&
                        G[x[n],x[1]] ≠ 0 )

            return;

        }
    } while (true);
}
```

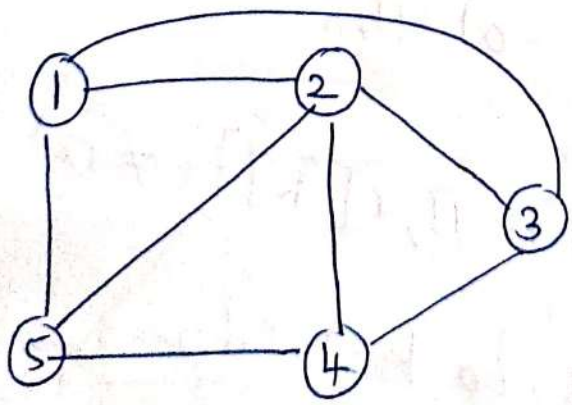
Bounding function

→ value taken in x array should no exist (start for 1 commonly).

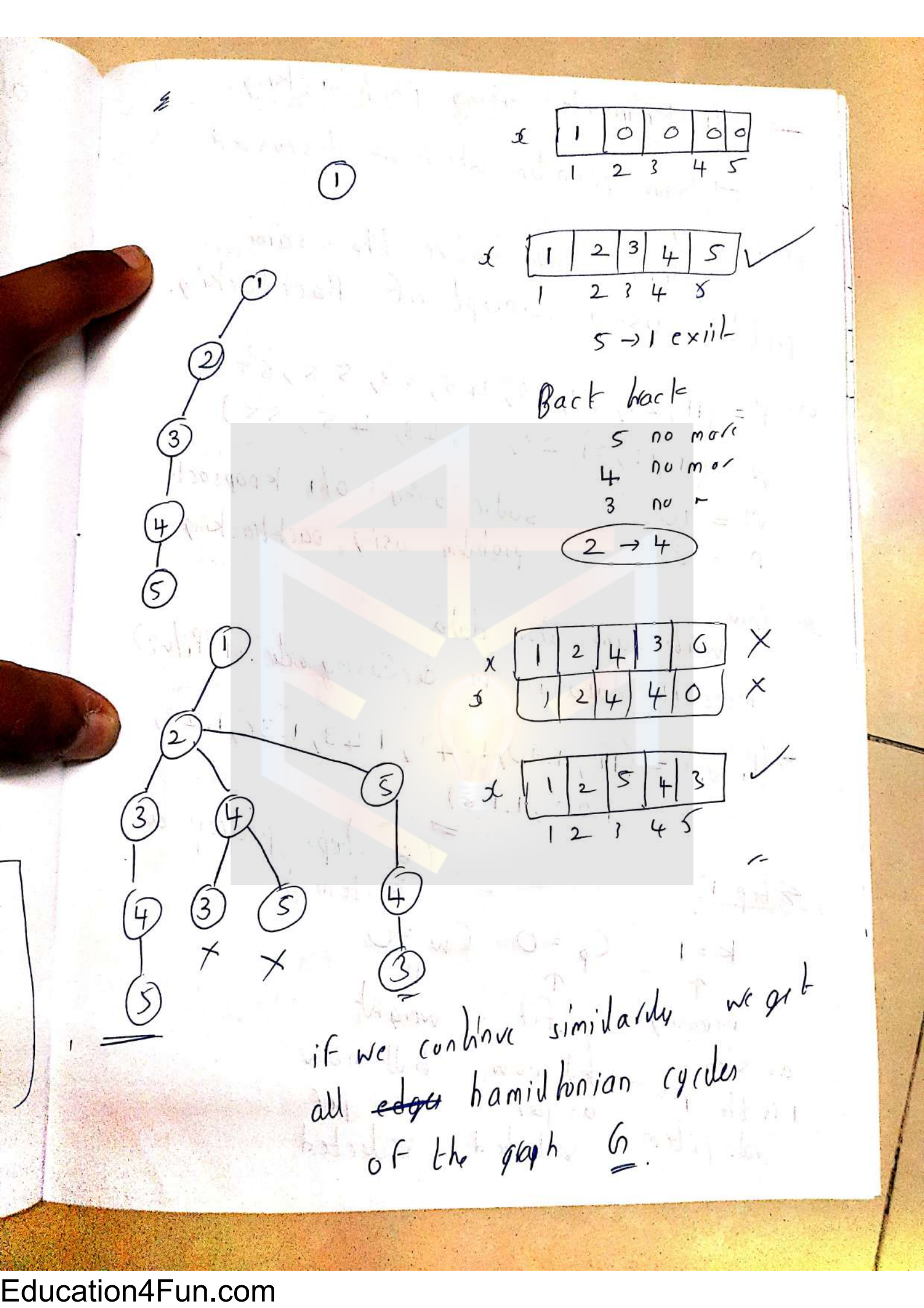
→ edge should exist to previous one

→ last x value should have edge to vertex 1 //

ex:



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



→ 0/1 Knapsack using Backtracking:

→ Same problem which we discussed previously

→ Here we solve the same problem using concept of Backtracking.

ex: P = (11, 21, 31, 33, 43, 53, 55, 65)
W = (1, 11, 21, 23, 33, 43, 45, 55)
M = 10
n = 8

solve using 0/1 knapsack problem using backtracking

⇒ Give write same above data make the data is in decreasing order of (P_i/w_i)

→ (P_i/w_i) = (11, 1.90, 1.47, 1.43, 1.30, 1.23, 1.22, 1.18)

⇒ (8 steps process as 8 items).

Step 1:

k = 1 C_p = 0 C_w = 0

↑ k: increasing as step 1 is the 1st value/item

↑ C_p: profit till now as per selected

↑ C_w: weight till now as per selected

Check $C_w + w[k] \leq m$ ← max weight.

$0 + 1 \leq 100$ (True) $[y[1] = 1]$

Check $(k <= n) \Rightarrow (1 < 8)$ ✓

continue to next step (2, 11, 1)

Step 2:

$k = 2$ $C_p = 11$ $C_w = 1$

Check $C_w + w[k] \leq m$

$(1 + 11 \leq 110)$ True. $[y[2] = 1]$

check $(k <= n) \Rightarrow (2 < 8)$ ✓

continue to next step (3, 32, 12)

Step 3:

$k = 3$ $C_p = 32$ $C_w = 12$

check $C_w + w[k] \leq m$

$(12 + 21 \leq 110)$ True $[y[3] = 1]$

check $(k <= n) \Rightarrow (3 < 8)$ ✓

continue to next step (4, 63, 33)

Step 4

k = 4 cp = 63 cw = 33

Check if ((w + w[k] ≤ m)

33 + 23 ≤ 110 (True) [y[4] = 1]

check (k < n) => 4 < 8 ✓

next step (5, 96, 56)

Step 5

k = 5 cp = 96 cw = 56

check if ((cw + w[k] ≤ m)

56 + 33 ≤ 10 (True) [y[5] = 1]

check (k < n) => 5 < 8 ✓

(5 < 8) next step (6, 139, 89)

Step 6

k = 6 cp = 139 cw = 89

Check if (cw + w[k] ≤ m)

89 + 43 ≤ 110 (False)

∴ Y[6] = 0

(6 < 8) next step (7, 139, 89)

Step 7

k = 7 $C_p = 139$ $C_w = 89$

check $(C_w + w[k] \leq m)$

$89 + 45 \leq 110$ (false)

$\therefore Y[7] = 0$

$(7 < 8)$ next step $(8, 139, 89)$

Step 8

$k = 8$ $C_p = 139$ $C_w = 89$

Check $(C_w + w[k] \leq m)$

$89 + 55 \leq 110$

$144 \leq 100$

Check $(k < n)$ $(8 < 8)$ false

$\therefore$ print $x[1:n]$

$x[1] = 1$

$x[2] = 1$

$x[3] = 1$

$x[4] = 1$

$x[5] = 1$

$x[6] = 0$

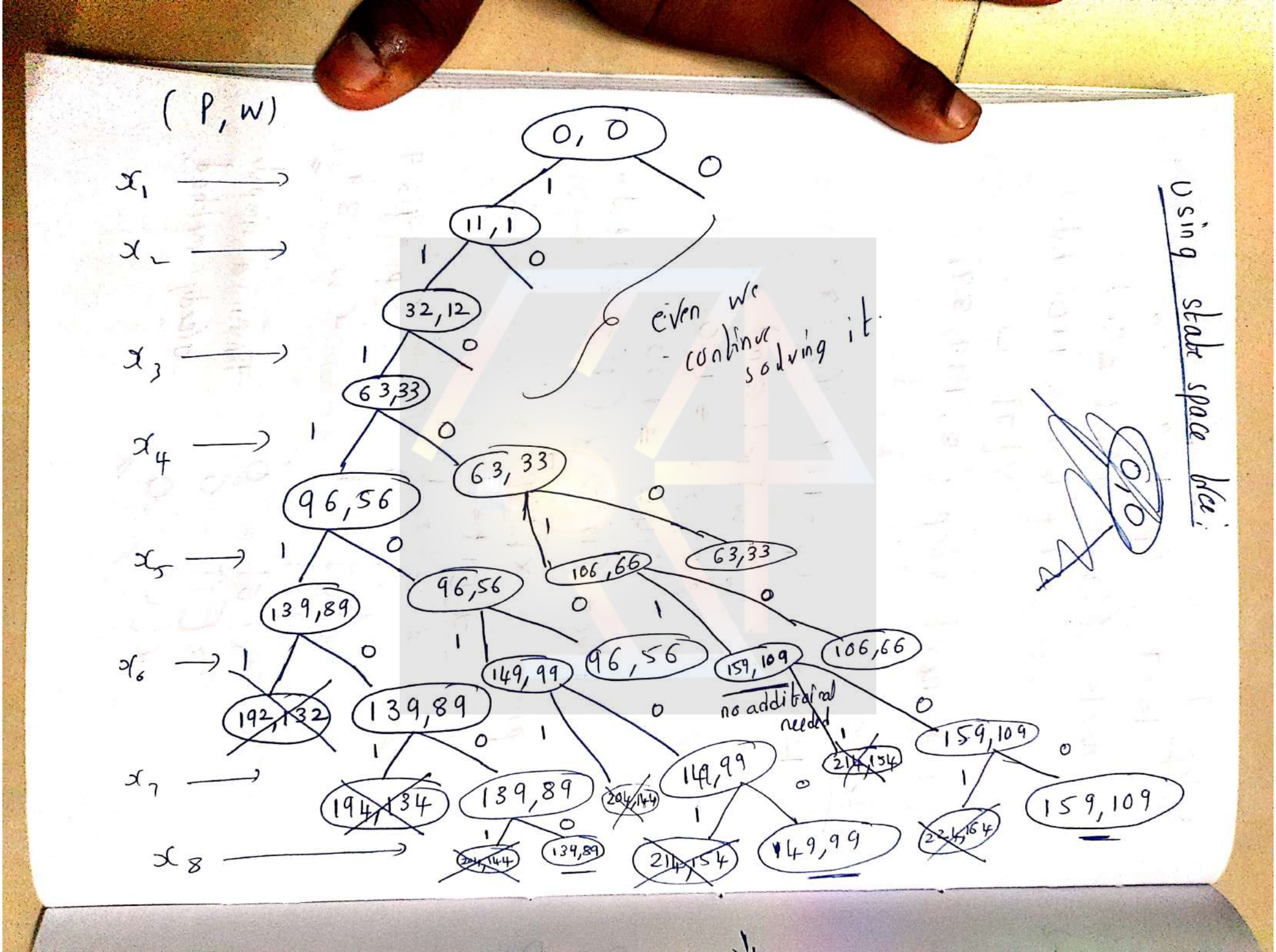
$x[7] = 0$

$x[8] = 0$ //

$C_p = 139$

$C_w = 89$

This is a simple greedy method.



The best optimal solution is

$(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (1, 1, 1, 0, 1, 1, 0, 0)$

$\Rightarrow C_p = 159$

$C_w = 109$ $(m = 110)$

as above problem has 8 items it is too complex to draw the state space tree.

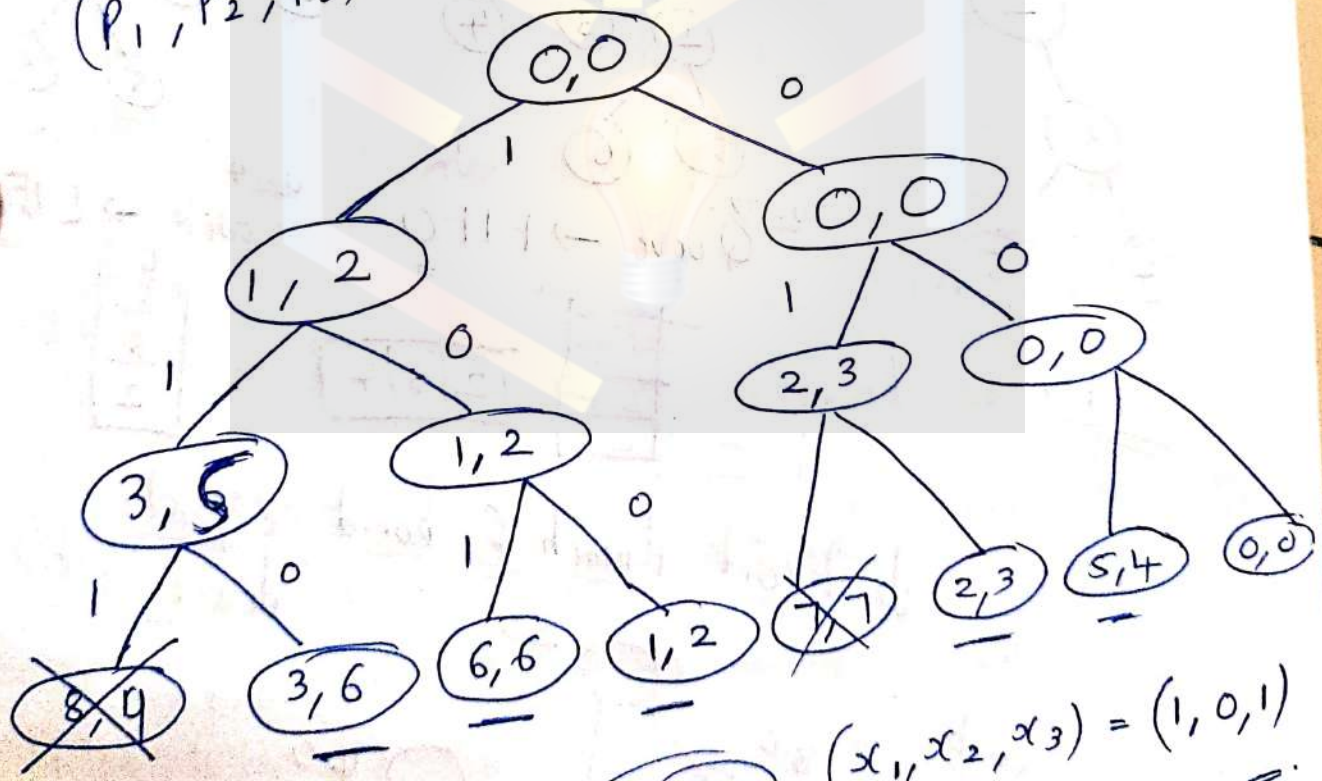
→ a small problem

$(w_1, w_2, w_3) = (2, 3, 4)$

$(p_1, p_2, p_3) = (1, 2, 5)$

$m = 6$

p, w



Best → 6,6

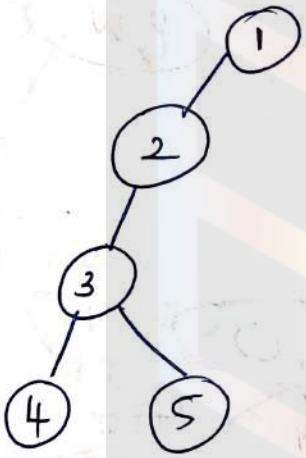
$(x_1, x_2, x_3) = (1, 0, 1)$

→ Branch & Bound :-

→ The branch & bound algorithm is similar to backtraking but is used For solving optimization problems (minimization)

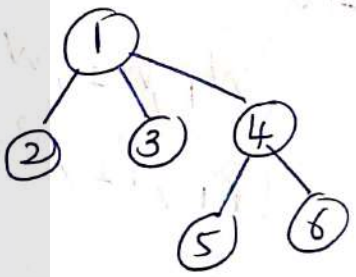
→ It performs a graph traversal on the state space tree using BFS instead of DFS → Backtracking.

eg Backtracking

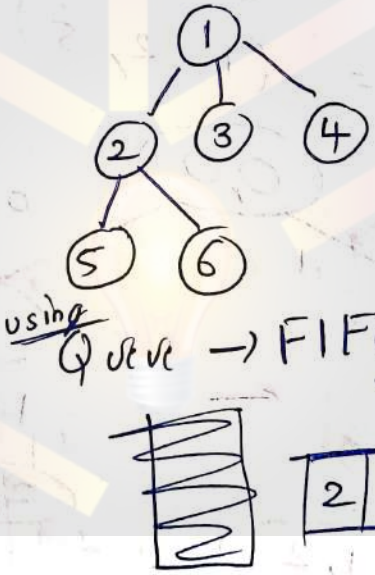


DFS

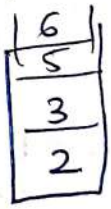
Branch & Bound



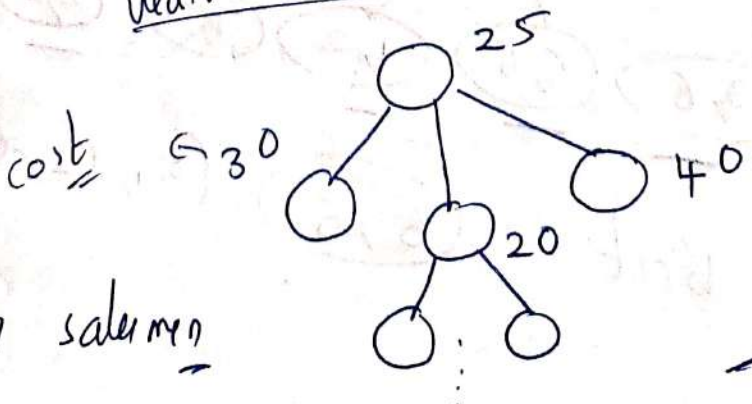
using Queue → FIFO



using stack → LIFO



least cost Branch & bound (expand least cost)



eg Travelling salesman

→ least cost Branch & Bound (LCBB)

→ In both LIFO & FIFO branch & bound the selection rule for next node is rigid & blind (a pattern exist)

→ Here in LCBB we take least cost only (expand them) avoiding all the others (constrain)

→ To calculate the cost of a node x

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

$\hat{c}(x)$ → It is the estimated min cost to reach the goal node.

$f(x)$ = no. of moves from initial state

$\hat{g}(x)$ = no. of non blank tiles that are not in their goal position.

ex: 15 puzzle problem, 0/1 knapsack problem, travelling salesmen problem.

→ 15 puzzle problem (least cost Branch & Bound (LCBB))

→ The 15-puzzle consist of 15 numbered tiles on a square frame with a capacity of 16 tiles

→ We are given initial arrangement of the tiles & the objective is to transform this arrangement into the goal arrangement.

ex

1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

given arrangement

→

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

goal arrangement

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

↳ is the estimated min cost to reach the goal node

$f(x)$ = No. of moves from initial state

$g(x)$ = no. of non blank tiles that are not in goal position

moving based on empty space

1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

$\hat{C}(1) = 0 + 3 = 3$

up: $\hat{C}(2) = 1 + 4 = 5$

1	2		4
5	6	3	8
9	10	7	11
13	14	15	12

X

down (min): $\hat{C}(3) = 1 + 2 = 3$

1	2	3	4
5	6	7	8
9	10		11
13	14	15	12

left: $\hat{C}(4) = 1 + 4 = 5$

1	2	3	4
5		6	8
9	10	7	11
13	14	15	12

X

right: $\hat{C}(5) = 1 + 4 = 5$

1	2	3	4
5	6	8	
9	10	7	11
13	14	15	12

X

down

1	2	3	4
5	6	7	8
9	10	15	11
13	14		12

X

$\hat{C}(6) = 2 + 3 = 5$

left

1	2	3	4
5	6	7	8
9		10	11
13	14	15	12

X

$\hat{C}[7] = 2 + 3 = 5$

right

1	2	3	4
5	6	7	8
9	10	11	
13	14	15	12

$\hat{C}[8] = 2 + 1 = 3$ (min)

up

1	2	3	4
5	6	7	
9	10	11	8
13	14	15	12

$C[9] = 3 + 2 = 5$

down

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

$C[10] = 3 + 0 = 3$

hence goal reached

0

→ 0/1 Knapsack Problem using Least Count Branch & Bound (LCBB)

→ The same concept of knapsack Problem which we discussed previous

→ Here we know in Branch & Bound we do solve for minimization hence here we consider the ~~post~~ profit as negative (5 ⇒ -5)

here for each node we calculate cost & upperbound

$$U = \sum_{i=1}^{n} p_i x_i \quad \Big| \quad \leq m$$

$$C = \sum_{i=1}^{n} p_i x_i \quad \Big| \quad \leq m$$

↙ only for computation we use this

↓ (with fraction)

ex: $n = 5$

P_1 to P_5 = (10, 15, 6, 8, 4)

w_1 to w_5 = (4, 6, 3, 4, 2)

$m = 12$ using LCBB

- Convert profits to negative

$(P_1, P_2, P_3, P_4, P_5) = (-10, -15, -6, -8, -4)$

$m = 12$

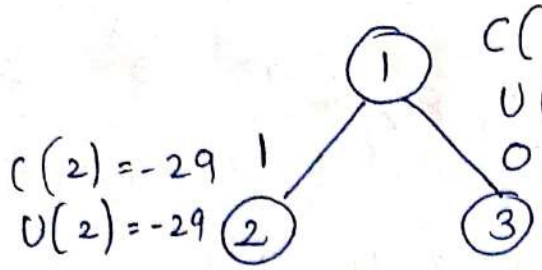
Placing items 1, 2, 5 ⇒ weight (in order) ↳ 4 + 6 + 2 = 12

⇒ Profit earned ⇒ $-10 - 15 - 4 = -29$ ↑ U(1) upper bound

lower bound ⇒ including (Fractional)

$C(1) \Rightarrow -10 - 15 - \frac{2}{3} \times 6 = -29$

$x_1 = 1$ (include 1st item)	$x_1 = 0$ (no 1st item)
$C(2) = -10 - 15 - \frac{2}{3} \times 6$	$C(3) = -15 - 6 - \frac{3}{4} \times 8$
$= -29$	$= -27$
$U(2) = -10 - 15 - 4$	$U(3) = -15 - 6 - 4$
$= -29$	$= -25$



$C(1) = -29$
$U(1) = -29$

$C(2) = -29$
$U(2) = -29$

$C(3) = -27$
$U(3) = -25$

$C(\min)$ ✓ ✗

$x_2 = 1$

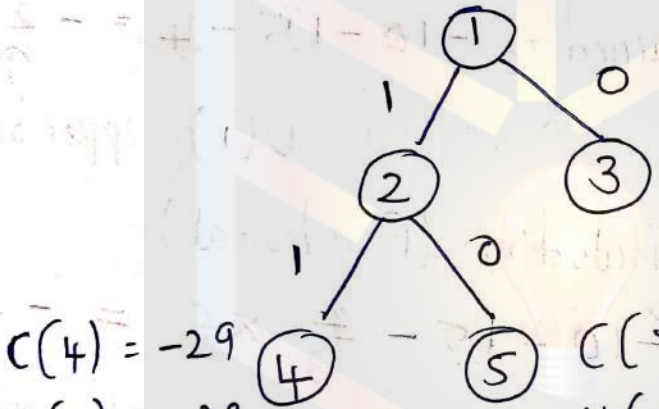
$C(4) = -10 - 15 - \frac{2}{3} * 6$
$= -29$

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

$x_2 = 0$

$C(5) = -10 - 6 - 8 - \frac{1}{2} \times 4$
$= -26$

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



$C(4) = -29$
$U(4) = -29$

$C(5) = -26$
$U(5) = -24$

✗

$x_3 = 1$

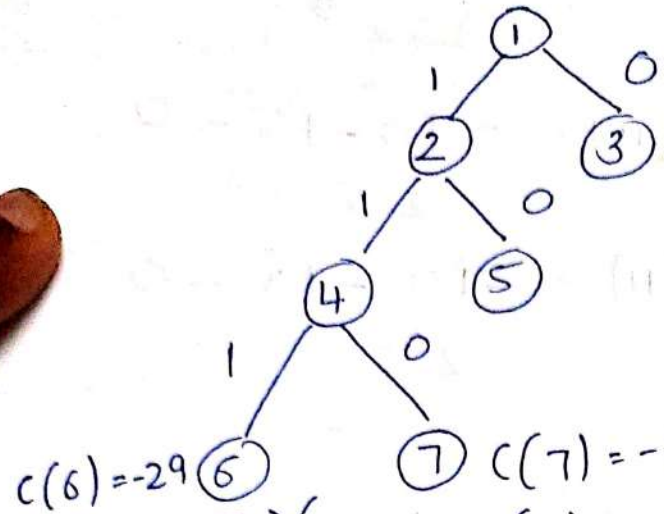
$C(6) = -10 - 15 - \frac{2}{3} * 6$
$= -29$

$U(6) = -10 - 15 - 9$
$= -29$

$x_3 = 0$

$C(7) = -10 - 15 - \frac{2}{3} \times 6$
$= -29$

$U(7) = -10 - 15 - 4$
$= -29$



$C(6) = -29$
$U(6) = -25$ ✗

$C(7) = -29$
$U(7) = -29$

$C(6) == C(7)$ ✓

$\min(U(6), U(7)) \Rightarrow U(7)$

$x_4 = 1$

$C(8) = -10 - 15 - \frac{2}{4} \times 8$

$= -29$

$U(8) = -10 - 15 - 0$

$= -29$

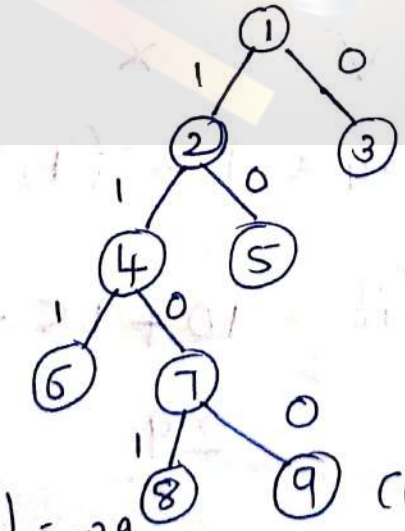
$x_4 = 0$

$C(9) = -10 - 15 - \frac{2}{3} \times 6$

$= -29$

$U(9) = -10 - 15 - 4$

$= -29$



$C(8) = -29$
$U(8) = -25$ ✗

$C(9) = -29$
$U(9) = -29$

$x_5 = 1$

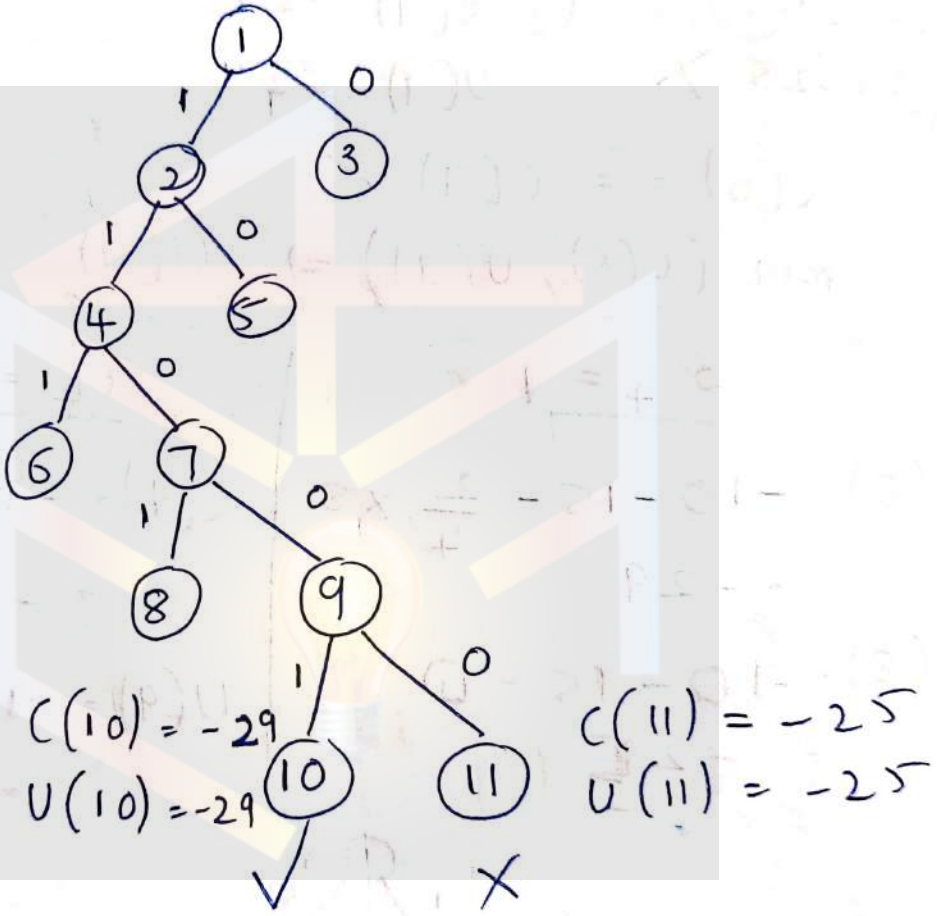
$c(10) = -10 - 15 - 4$
$= -29$

$u(10) = -10 - 15 - 4$
$= -29$

$x_5 = 0$

$c(11) = -10 - 15 - 0$
$= -25$

$u(11) = -10 - 15 - 0$
$= -25$



$\therefore (x_1 x_2 x_3 x_4 x_5) = (1, 1, 0, 0, 1)$

$\Rightarrow$ max profit $= 10 + 15 + 4$

$= 29$

→ 0/1 Knapsack Problem using FIFO Branch & Bound

→ The concept of knapsack problem which we discussed previously.

→ Here we know in Branch & Bound we do solve for minimization hence here we consider the profit as negative (5 ⇒ −5).

→ here we calculate lower bound (cost & upper bound for each node.

$$U = \sum_{i=1}^{n} p_i x_i$$

$$C = \sum_{i=1}^{n} p_i x_i \text{ (with fraction)}$$

Here we maintain a global upper bound to kill unwanted (non optimal) solution path

ex: $n = 4$

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

$m = 15$ $(W_1, W_2, W_3, W_4) = (2, 4, 6, 9)$

$\Rightarrow$ convert profits to negative

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

global $U = \cancel{0}\ \cancel{-32}\ -38$ $U(1) = ?$ (1)

$C(1) = ?$

$m = 15$

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

down or $C(1) = -10 - 10 - 12 - \frac{3}{9} * 18 = -38$

$U(1) = -32$ (1)

$C(1) = -38$

$x_1 = 1$

$U(2) = -10 - 10 - 12$

$= -32$

$C(2) = -10 - 10 - 12 - \frac{3}{9} * 18$

$= -38$

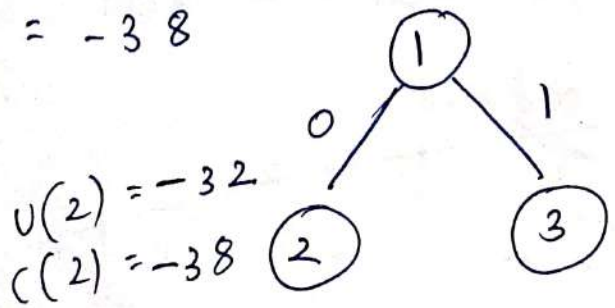
$x_1 = 0$

$U(3) = -10 - 12$

$= -22$

$C(3) = -10 - 12 - \frac{5}{9} * 18$

$= -32$



$U(2) = -32$

$C(2) = -38$

$U(3) = -22$

$C(3) = -32$

For $x_1 = 1$ $\quad x_2 = 1$

$U(4) = -10 - 10 - 12$

$= -32$

$C(4) = -10 - 10 - 12 - \frac{3}{9} * 18$

$= -38$

For $x_1 = 0$ $\quad x_2 = 1$

$U(6) = -10 - 12$

$= -22$

$C(6) = -10 - 12 - \frac{5}{9} \times 18$

$= -32$

$x_2 = 0$

$U(5) = -10 - 12$

$= -22$

$C(5) = -10 - 12 - \frac{7}{9} * 18$

$= -36$

$x_2 = 0$

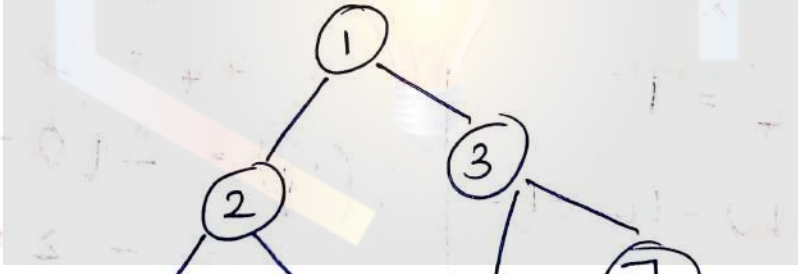
$U(7) = -12 - 18$

$= -30$

$C(7) = -12 - 18$

$= -30$

check condition ⇒ lower bound > global upper bound kill it



$U(4) = -32$
$C(4) = -38$

$U(5) = -22$
$C(5) = -36$ ✗

$U(6) = -22$
$C(5) = -32$ ✗

$U(7) = -30$
$C(7) = -30$ ✗

$x_3 = 1$

$U(8) = -10 - 10 - 12$

$= -32$

$C(8) = -10 - 10 - 12$

$\frac{3}{9} * -18$

$= -38$

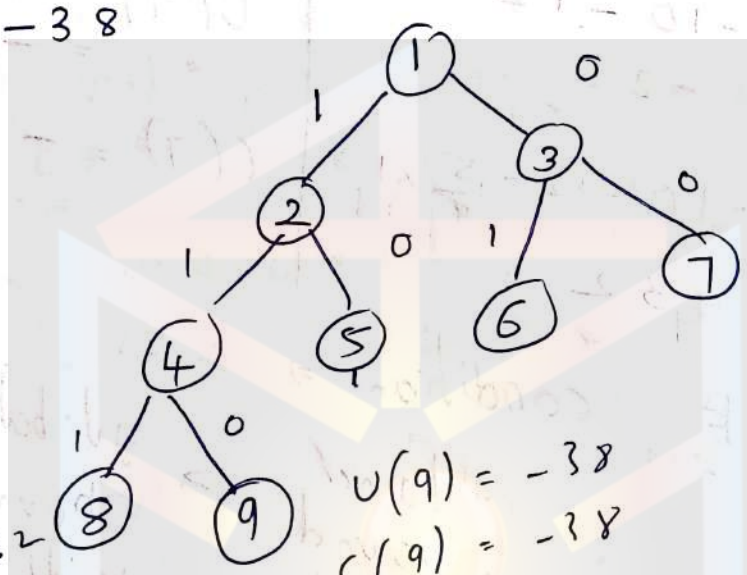
$x_3 = 0$

$U(9) = -10 - 10 - 18$

$= -38$

$C(9) = -10 - 10 * -18$

$= -38$



$U(8) = -32$

$C(8) = -38$ ✗

$U(9) = -38$

$C(9) = -38$

$x_4 = 1$

$U(10) = -10 - 10 - 18$

$= -38$

$C(10) = -10 - 10 - 18$

$= -38$

$x_4 = 0$

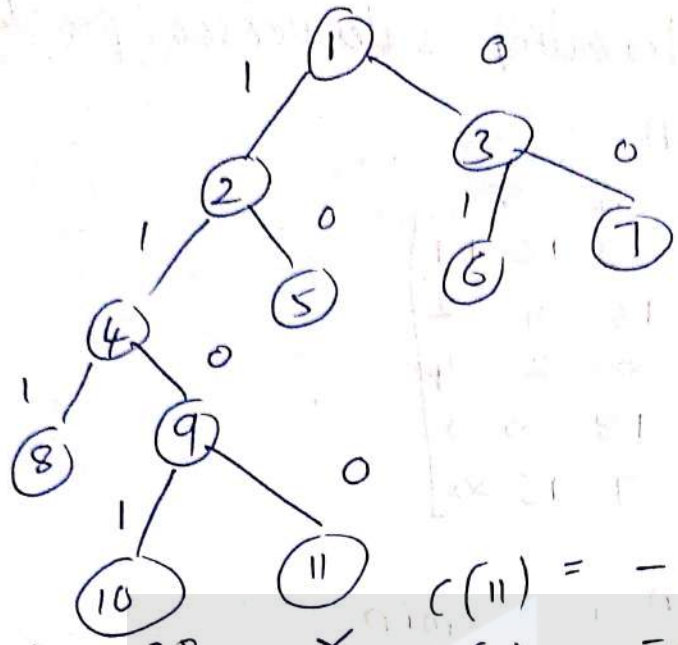
$U(10) = -10 - 10$

$= -20$

$C(10) = -10 - 10$

$= -20$

$=$



$c(10) = -38$
$u(10) = -38$

$\times$

$c(11) = -20$
$u(11) = -20$

soln $= (x_1, x_2, x_3, x_4) = (1, 1, 0, 1)$

$\therefore (x_1, x_2, x_3, x_4) = (1, 1, 0, 1)$

$\Rightarrow P = 10 + 10 + 18 = 38 \quad m = 15$

$w = 2 + 4 + 9 = 15$

→ Travelling sale person problem (LCBB)

→ the same concept of normal travelling salesperson problem.

→ Here we use reduction of matrices to solve the problem

→ & at the end we get the optimal (minimal) cost path (using state space diagram).

ex: solve travelling sale person problem using LCBB

	1	2	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	∞

sol: Row reduction

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

total 21

⇒

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

Column reduction

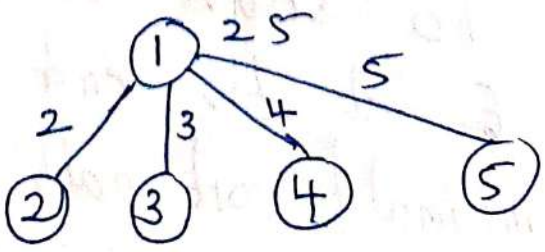
∞	10	20	0	1
13	∞	14	2	0
1	3	∞	0	2
16	3	15	∞	0
12	0	3	12	∞
1	–	3	–	–

4

⇒

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

cost = 25



Path (1,2) 1-row, 2-col ⇒ ∞

(1,2) = ∞

(2,1)

$$\begin{array}{|ccccc|c} \infty & \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & 11 & 2 & 0 & 0 \\ 0 & \infty & \infty & 0 & 2 & 0 \\ 15 & \infty & 12 & \infty & 0 & 0 \\ 11 & \infty & 0 & 12 & \infty & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

⇒ cost = C[1,2] + C(1) + r

= 10 + 25 + 0

= 35.

Path (1,3) 1row, 3-col ⇒ ∞

(1,3) (3,1) = ∞

$$\begin{array}{|ccccc|c} \infty & \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & 2 & 0 & 0 \\ \infty & 3 & \infty & 0 & 2 & 0 \\ 15 & 3 & \infty & \infty & 0 & 0 \\ 11 & 0 & \infty & 12 & \infty & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 11 \end{array} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix}$$

11

⇒ cost = C[1,3] + C[1] + r

= 17 + 25 + 11 = 53

=

Path (1,4) 1row, 4-col ⇒ ∞

(1,4) = ∞

(4,1)

$$\begin{array}{|ccccc|c} \infty & \infty & \infty & \infty & \infty & 0 \\ 12 & \infty & 11 & \infty & 0 & 0 \\ 0 & 3 & \infty & \infty & 2 & 0 \\ \infty & 3 & 12 & \infty & 0 & 0 \\ 11 & 0 & 0 & \infty & \infty & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

cost = C[1,4] + C[1] + r

= 15 + 25 + 0

= 25

Path (1,5) row - 1, col = 5 ⇒ ∞

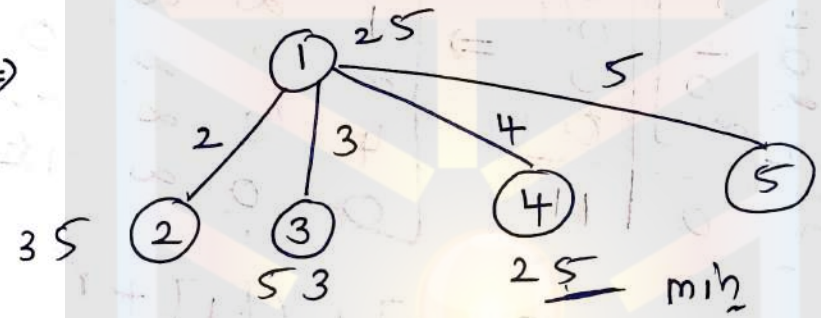
(5,1) = ∞

$$\left[\begin{array}{ccccc|c} \infty & \infty & \infty & \infty & \infty & 0 \\ 12 & \infty & 11 & 2 & \infty & 2 \\ 0 & 3 & \infty & 0 & \infty & 0 \\ 15 & 3 & 12 & \infty & \infty & 3 \\ \infty & 0 & 0 & 12 & \infty & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 5 \end{array}\right] \qquad \left[\begin{array}{ccccc} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 12 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{array}\right]$$

cost = C[1,5] + C[1] + ✓

= 1 + 25 + 5 = 31

⇒



Path (4,2) 4th row, 2nd col = ∞

(2,4) = ∞

use (1,4) matrix

$$\left[\begin{array}{ccccc|c} \infty & \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & 11 & \infty & 0 & 0 \\ 0 & \infty & \infty & \infty & 2 & 0 \\ \infty & \infty & \infty & \infty & \infty & 0 \\ 11 & \infty & 0 & \infty & \infty & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

cost =

C[4,2] +

C[4] +

= 3 + 25 + 0

= 28

Path (4,3)

4th row & 3rd col =1 ∞

(3,1) = ∞

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & \infty & 0 \\ \infty & 3 & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & 0 & \infty & \infty & \infty \end{bmatrix} \Bigg| \begin{matrix} 0 \\ 0 \\ 2 \\ 0 \\ 0 \end{matrix} \qquad \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & \infty & \infty & \infty \end{bmatrix}$$

$$\begin{matrix} 11 & 0 & 0 & 0 & 0 \end{matrix} \Big| \; 11+2 = 13$$

cost = C[4,3] + C[4] + r

= 12 + 25 + 13 = 50

Path (4,5)

4th row & 5th col =1 ∞

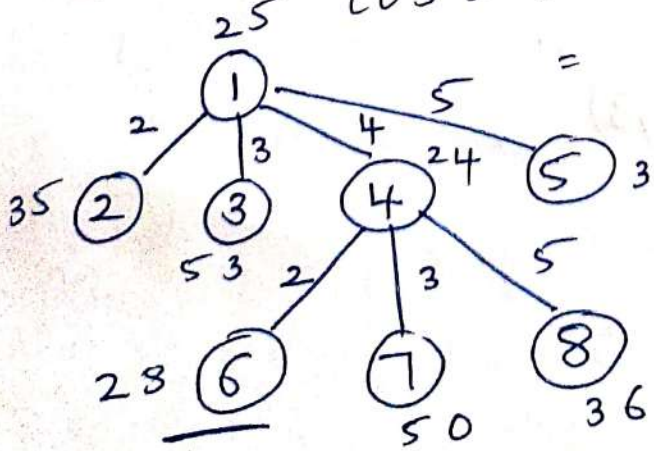
(5,1) = ∞

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty \end{bmatrix} \Bigg| \begin{matrix} 0 \\ 11 \\ 0 \\ 0 \\ 0 \end{matrix} \qquad \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty \end{bmatrix}$$

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \end{matrix} \Big| \; 11$$

cost ⇒ C[4,5] + C[4] + r

= 0 + 25 + 11 = 36



Path (2,3)

(using 4,2)

2nd row & 3rd col = ∞

(3,1) = ∞

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

11 0 0 0 0 | 11+2 = 13

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

cost = C[2,3] + C[2] + ✓

= ~~28~~ 11 + 28 + 13 = 52

Path (2,5)

2nd row & 5th col = ∞

(5,1) = ∞

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

cost = C[2,5] + C[2] + ✓

= 0 + 28 + 0

= 28

(2,5) min

dual path (5,3)

Path (5, 3) 5th row & 3rd col

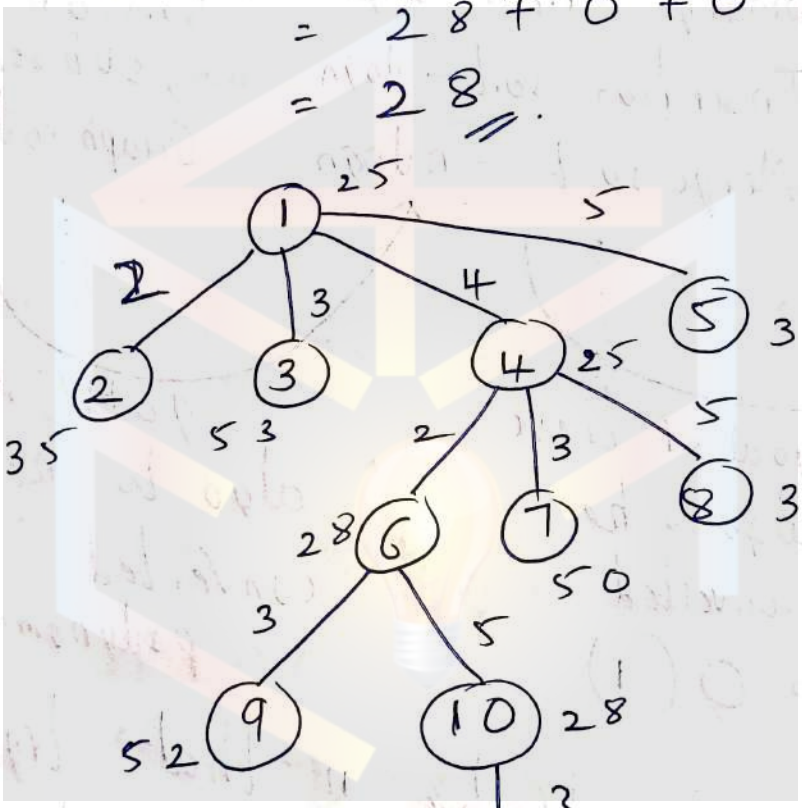
$(3,1) = \infty$

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

cost = C[5] + C[5,3] + r

= 28 + 0 + 0

= 28 //



28 min

path -> 1 → 4 → 2 → 5 → 3 → 1

cost = 10 + 6 + 2 + 7 + 3

= 28