UNIT - IV

Backtracking (PART-1)

→ **General Method**: Backtracking is used to solve problem in which a sequence of solutions is choosen from a specified set so that each one satisfies some criterion. The desired solution is expressed as an n-tuple $(x_1, \ldots, x_n)$ where $x_i \in S$, S being a finite set.

The solution is based on finding one (or) more vectors that maximizes, minimizes (or) satisfy a criterion function $P(x_1, \ldots x_n)$. Check at every step if this has any change of success. If the solution at any point seems not promising ignore it. All the solutions requires a set of constraints divided into 2 categories: Explicit & Implicit.

- **Explicit** – these are rules that restrict each x_i to take on values only from a given set. All the tuples that satisfy the explicit constraints define a possible solution space 'S'.
- **Implicit** – these are rules that determine which of the tuples in the solution space of 'S' satisfy the criterion function.

- for 8-Queens problem: Explicit constraints using 8-tuple solution for this problem are: $S = \{1, 2, 3, 4, 5, 6, 7, 8\}$.

The Implicit constraints are that no two queens should occur in same row (or) column (or) diagonal.

Backtracking is a modified Depth First Search (DFS) of a tree. Backtracking algorithms determine problem solutions by systematically searching the solution space for the given problem.

Backtracking is the procedure of finding solution as a child node but if it is dead end, we go back (backtrack) to the nodes parent & proceed with the search on the next child. Like this we will construct state space trees.

→ **4 - Queens Problem**: In this problem we have to place 4 queens in 4x4 chess board with constraints like no any two queens should occur in same row (or) column (or) diagonal.

By using backtracking we are going to find what all the possible solutions are there which satisfies the constraints. Now we can solve this problem by constructing a state space tree. If any node is not satisfying the "Bounding Function" that is no any 2 queens should occur in same row, col & diagonal we will kill that node & perform backtracking to select other node & expand that node.

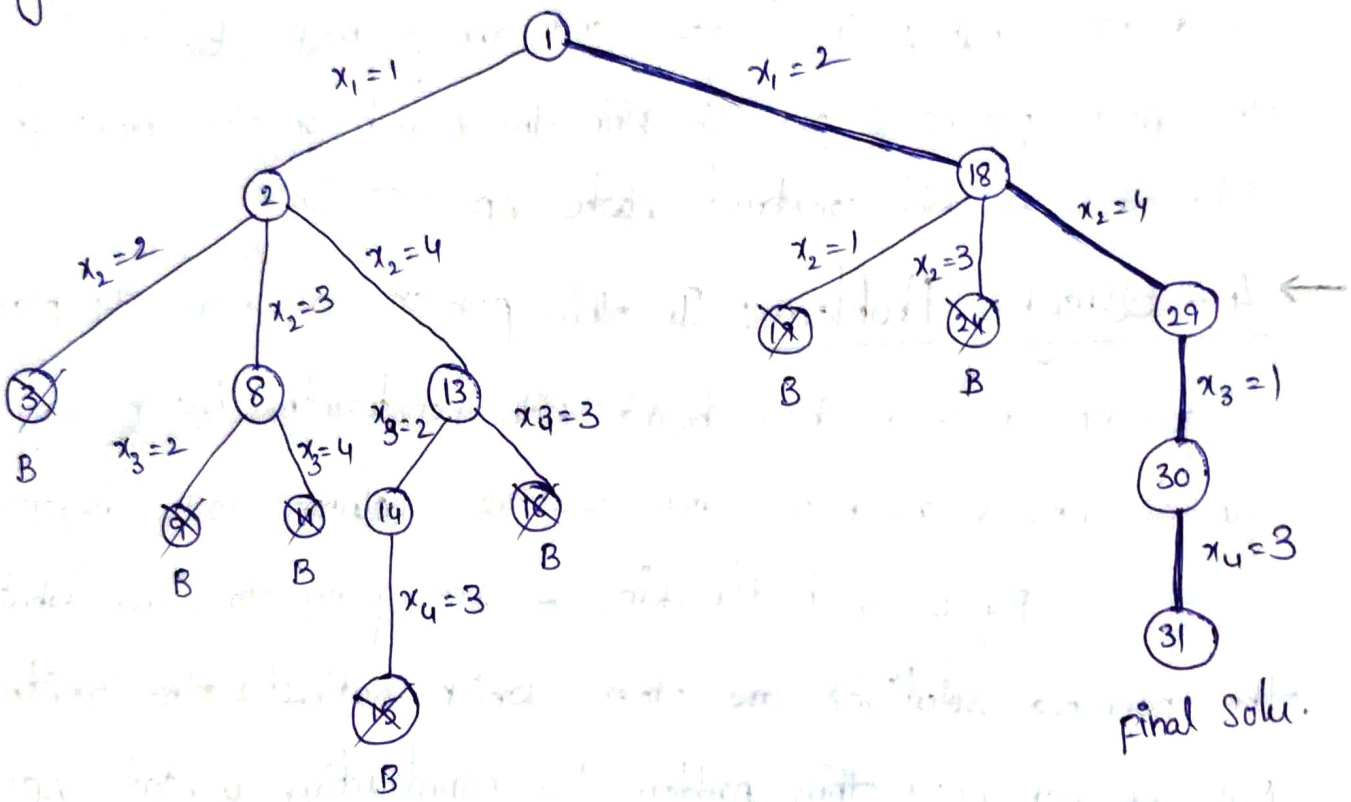
∴ Maximum no of nodes to be generate in a tree are:

$$1 + \sum_{i=0}^{3} \left[\prod_{j=0}^{i} (4-j) \right] = 65 \quad \text{(or)} \quad 1 + \sum_{i=0}^{k} \left[\prod_{j=0}^{i} (N-j) \right]$$

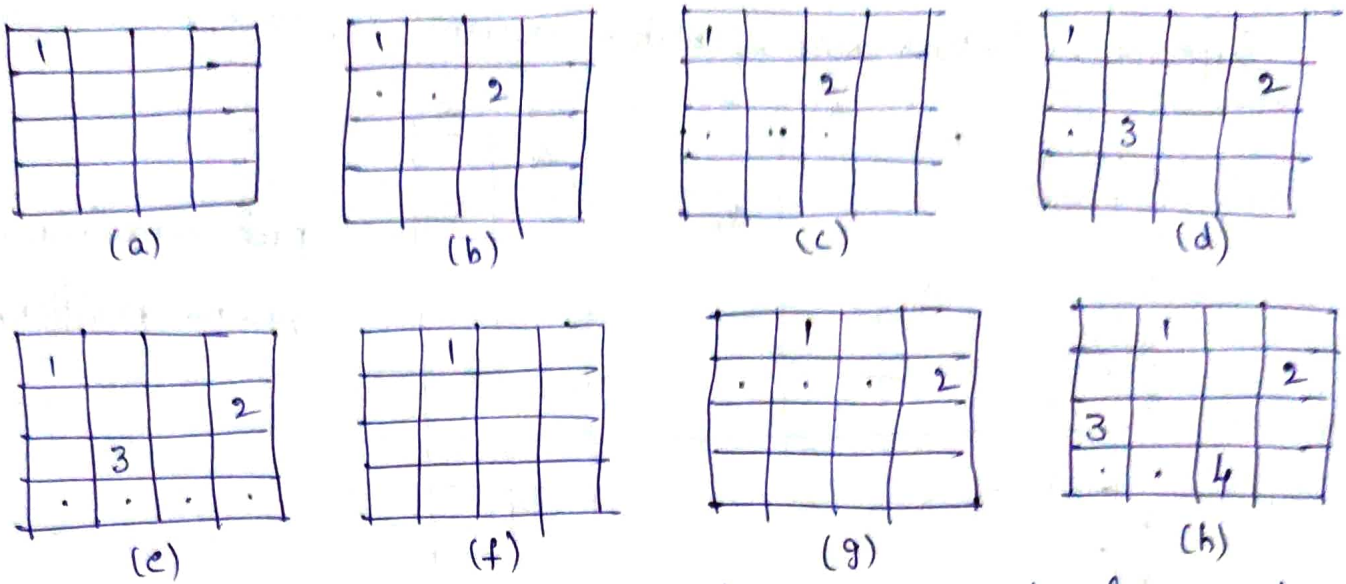
↓ for 4-Queens ↓ for N-Queens

Here we are not generating all 65 nodes only possible nodes we are generating.

Let us see how backtracking works on 4-queen problem. We start with the root node as the only live node. this becomes the E-node. We generate one node. children are generated in depth order.



From root node 1 node number 2 is generated & the path is now (1). this corresponds to placing queen '1' on column 1. Node 2 becomes E-node. Now node 3 is generated & immediately killed. the next generated node 8 & the path becomes (1,3). Node 8 becomes the E-node. However, it gets killed as all it's children cannot lead to an answer node. We backtrack to node 2 and generate another child, node 13. The path is now (1,4). Backtracking proceed is as follows:



In figure (b) the 2nd queen is placed in column 1 & 2 and finally settles on column 3. In fig (c) algorithm tries all four columns & is unable to place the next queen on a square. Backtracking now takes place. Remaining steps goes through until a solution is found.

The final Solution Vector is: x

1	2	3	4
2	4	1	3

column Numbers

→ **8-Queens Problem:** 8-Queens problem is to place eight queens on an 8×8 chessboard. So that no two attack on same row (or) column (or) diagonal. All solutions to the 8-Queens problem can be represented as 8-tuples $(x_1, \ldots, x_8)$.

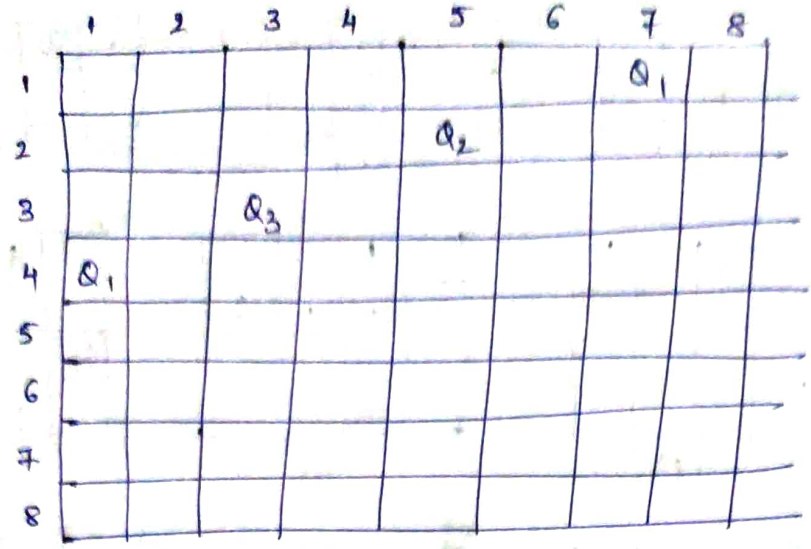
The explicit constraints are: $S_i = \{1,2,3,4,5,6,7,8\}$, $1 \le i \le 8$

The implicit constraints are: No two x_i's can be the same diagonal. (or) row (or) column.

Suppose 2 queens are placed at positions (i,j) & (k,l) then:

$|i-j| = |k-l|$ (or) $|i+j| = |k+l|$ represents two queens are in same diagonal. Conflict is occuring

Ex: Suppose we start with feasible sequence : 7, 5, 3, 1



find remaining queens positions.

Sol: i ⟶ Occupied positions

Queens :	1	2	3	4	5	6	7	8	
↓	7	5	3	1					No Conflicts
i	7	5	3	1	4				Conflict with (1,7)
	7	5	3	1	4	2			Conflict with (3,3) ←
	7	5	3	1	4	6			No conflicts
	7	5	3	1	4	8			Conflict with (5,4)
	7	5	3	1	4	8	2		Conflict with (5,4)
	7	5	3	1	4	8	6		Backtracking 5
	7	5	3	1	6				No Conflicts.
	7	5	3	1	6	2			conflict with (1,7)
	7	5	3	1	6	4			No Conflicts
	7	5	3	1	6	4	2		No Conflicts
	7	5	3	1	6	4	2	8	Conflict with (3,3)
	7	5	3	1	6	4	8		Backtracking 7
									Conflict with (5,5)
									Backtracking 6

7 5 3 1 6 8 — No conflicts ←

7 5 3 1 6 8 2 — No conflicts

7 5 3 1 6 8 2 4 — No conflicts

∴ Final Solution is: **7 5 3 1 6 8 2 4.**

- In this solution while placing a queen at a position we will compare with all the other queen's positions to verify are there any conflicts occuring at diagonals by checking both conditions: $|i-k| = |j-l|$ & $|i+j| = |k+l|$.

- If there are no any other chances for positions which are possible then we have to change the previous Queens locations using backtracking approach.

- In our example 4 times we are doing backtracking of Queens '5', '7' & '6'.

→ **Sum of Subsets**: Suppose we are given 'n' distinct +ve numbers and we want to find all combinations of these numbers whose sum is 'M'. this is called as "Sum of subsets" problem. Here +ve numbers we are taking as weights which are denoted as weight vector.

The elements x_i of the solution vector is either 0 (or) 1 depending on whether the weight w_i is included (or) not.

In the state space tree left subtree $x_i = 1$ (including object)

right subtree $x_i = 0$ (Not including object).

- Here Bounding functions are: Total weight of already included objects should be always less than or equal to M.

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

$\sum_{i=1}^{k} w_i x_i$ → already included objects

w_{k+1} → current object

- Second bounding condition is, we need enough weights to get exact value 'M'. that is so far what we have included & remaining weights should be greater than M.

$$\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i > M.$$

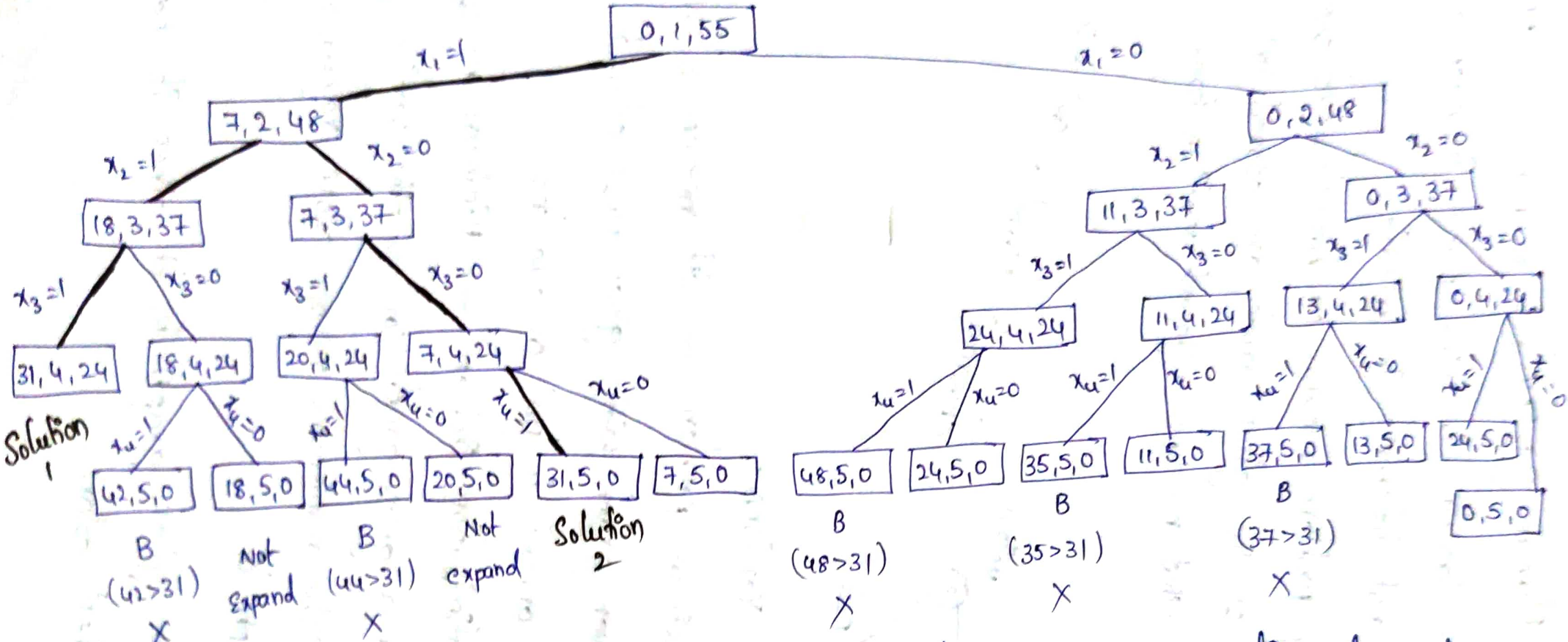
In tree each node contains 3 values : $\sum w_i x_i$, k, $\sum w_i$

$\sum w_i x_i$ → Sum of all weights

k → weight Number

$\sum w_i$ → Remaining weight

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



Solution 1 = {7, 11, 13}

= | 1 | 1 | 1 |

x_1 x_2 x_3

Solution 2 = {7, 24}

= | 1 | 0 | 0 | 1 |

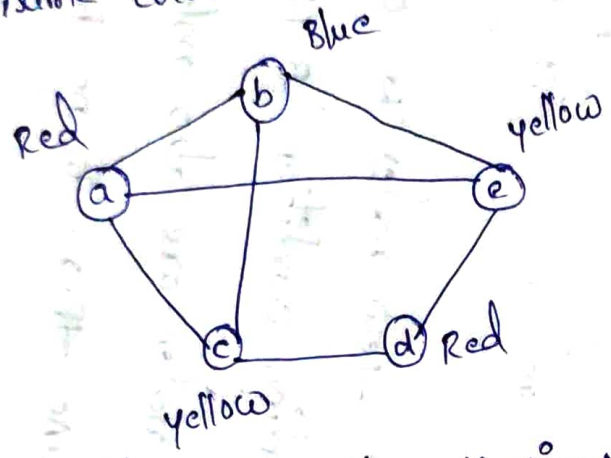
x_1 x_2 x_3 x_4

- Here, few nodes we are expanding due to bounding condition B (i.e). Sum of weights > M.
- few nodes are also not possible to expand why because there is no balance weight sum to get total weight M = 31.

→ **Graph Coloring (for Planar graphs)** : A planar graph is a graph that can be drawn in the plane in such a way that its edges will not intersect in middle.

Graph coloring problem means 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.

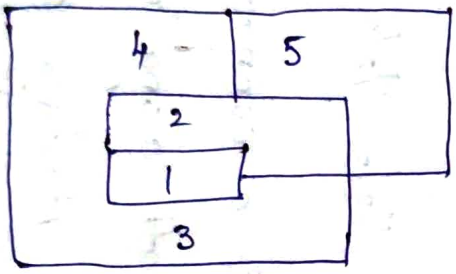
Ex:



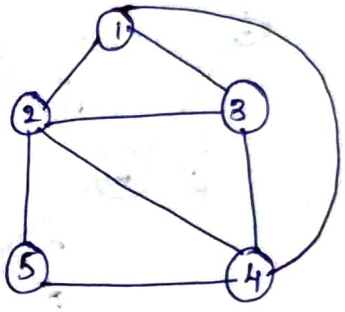
Chromatic Number (m) = 3.

- This problem has two versions:
  - i) m-colorability Decision problem — It means is it possible to color a given graph with given no. of colors 'm'. If it is possible then answer is "yes" or else "no".
  - ii) m-colorability Optimization Problem — Minimum no. of colors required to color a given graph (or) all the vertices.
- **Chromatic Number** : The minimum no. of colors required to color a given graph 'G'.
- If the degree of the given graph is 'd', then it can be colored with 'd+1' colors.

- If 'n' no.of colors are required to color all the vertices of a given graph (i.e) 'n' no.of vertices are there then that graph is called as : Kn graph.
- If a map contains some regions (or) states how can we color that map. first, convert that map into planar graph.



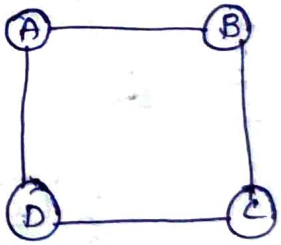
A map



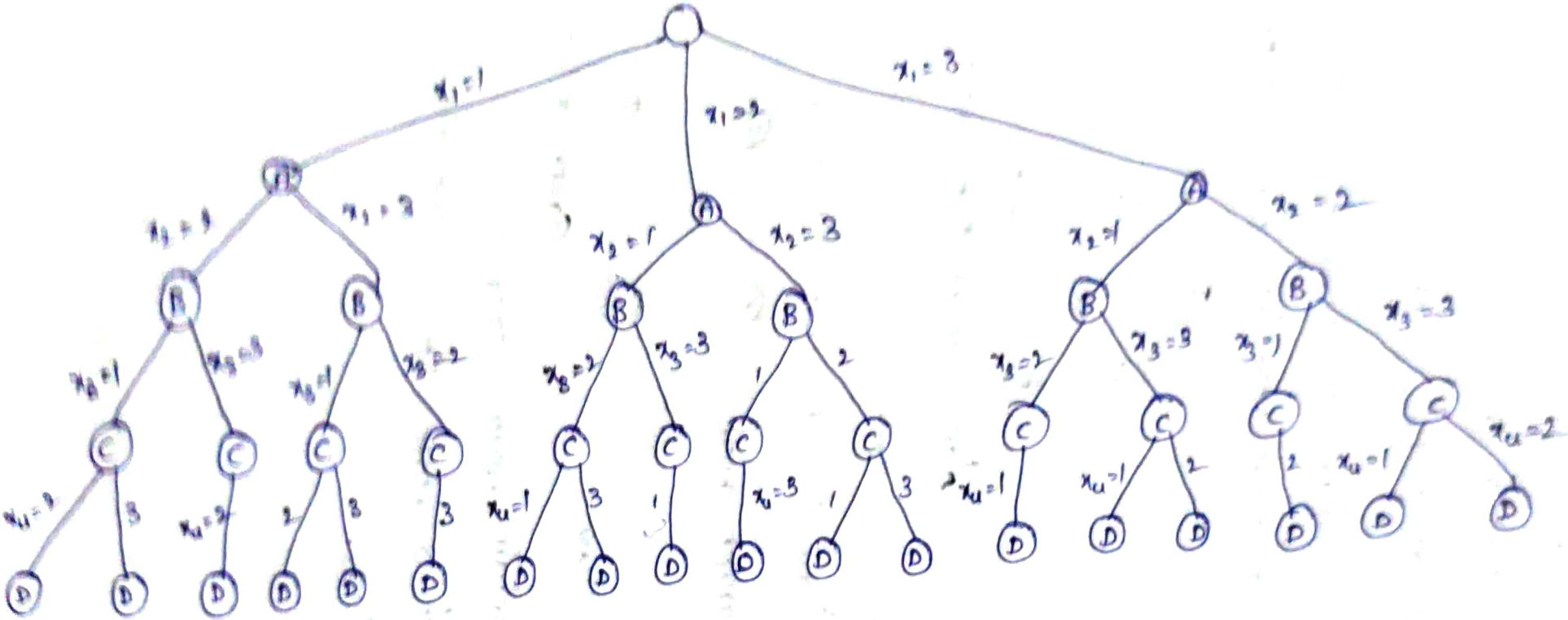
planar Graph Representation.

Ex: Given graph 'G' is having 4-vertices & no.of colors available are m = 3. find all the possible Solutions.

Sol:
- We have to color this graph with given no.of colors m = 3.
- Here constraint is no any two adjacent vertices should contain same color.
- Now we have to construct a state space tree to find all the possible solutions...



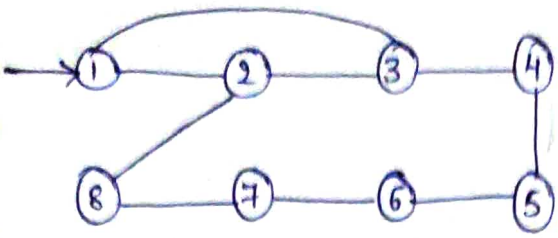
Graph, m = 3.



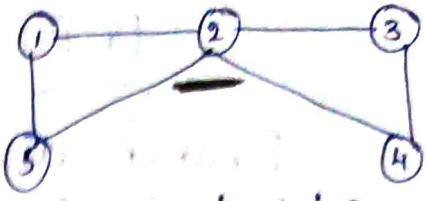
Here total possible solutions are : 18

Solution Vector is : $(x_1, x_2, x_3, x_4) = (1/2/3, 1/2/3, 1/2/3, 1/2/3)$

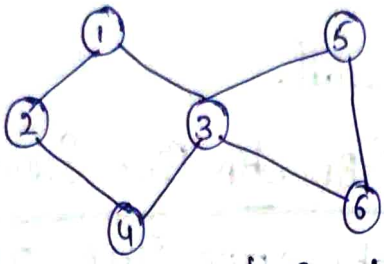
→ **Hamiltonian Cycle Problem** : Let $G(V, E)$ is a connected graph with 'n' vertices. A Hamiltonian cycle is a round trip path along 'n' edges of 'G' that visits each & every vertex exactly once except the starting & ending vertex and returns to the starting vertex.



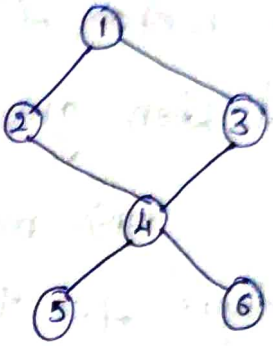
H. Cycle : 1-2-8-7-6-5-4-3-1
4-3-1-2-8-7-6-5-4



If any graph contains junction point there is no any H cycle



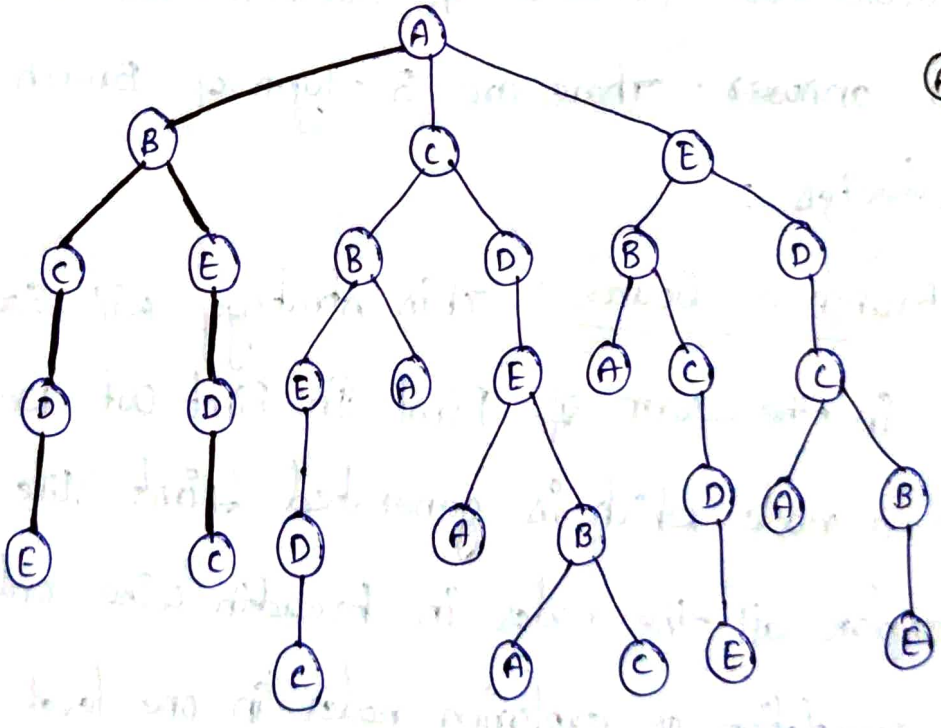
If any graph is having an Articulation point there is no any H. cycle

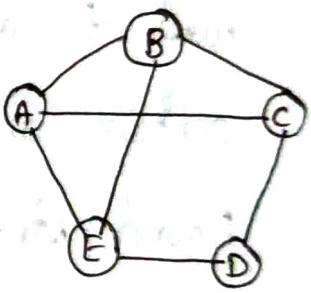


If any graph is having pendant vertices then there is no any Hamiltonian cycle

Ex: find out all possible Hamiltonian cycles of graph G.

Sol:





Hamiltonian cycles are : A B C D E A.
A B E D C A.

PART-2

Branch and Bound

→ General Method : Branch & Bound is a general algorithmic method for finding optimal solutions. this Method is useful when greedy & dynamic methods fail.

this method will follow BFS (Breadth First Search) technique for the construction of a state space Tree. then partitioning is done at each node of the tree. We compute lower bound & upper bound at each node. this computation leads to selection of answer node Bounding functions are used to avoid the generation of subtrees that do not contain an answer. there are 3 types of Branch and Bound strategies :

i) FIFO Branch & Bound : this strategy will Explore the nodes in the form of First In First Out. We have to explore a node which is generated first. like that we have to explore all the nodes in breadth wise order. After the completion of exploring nodes in one level next we will explore nodes at second level. Whereever the solution is not possible that node we will kill & stop the exploration. We will explore nodes where solution is possible

ii) LIFO Branch & Bound : In this strategy we have to expand last node in first level, next i have to explore last but one node like this we have to explore in backward direction. To store all the nodes Stack data structure is used. In FIFO strategy Queue data structure is useful.

iii) Least Cost (LC) Branch & Bound : In this we have to calculate least & upper bounds of each node. Next a node which is having least (or) minimum cost we should explore that node.

→ Applications of Branch & Bound :

i) 0/1 Knapsack problem using FIFO & LC branch & bound.

ii) Travelling Salesperson problem using LC Branch & Bound

→ 0/1 Knapsack Problem using FIFO strategy :

Ex: Draw the state space tree using FIFO Branch & Bound strategy for the knapsack problem, $n=4$, $M=15$, $(P_1, P_2, P_3, P_4) = (10, 10, 12, 18)$ & $(w_1, w_2, w_3, w_4) = (2, 4, 6, 9)$.

Sol: Branch & Bound will generate solution to Minimal optimization problem. So, we have to convert all profit values into -ve values.

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

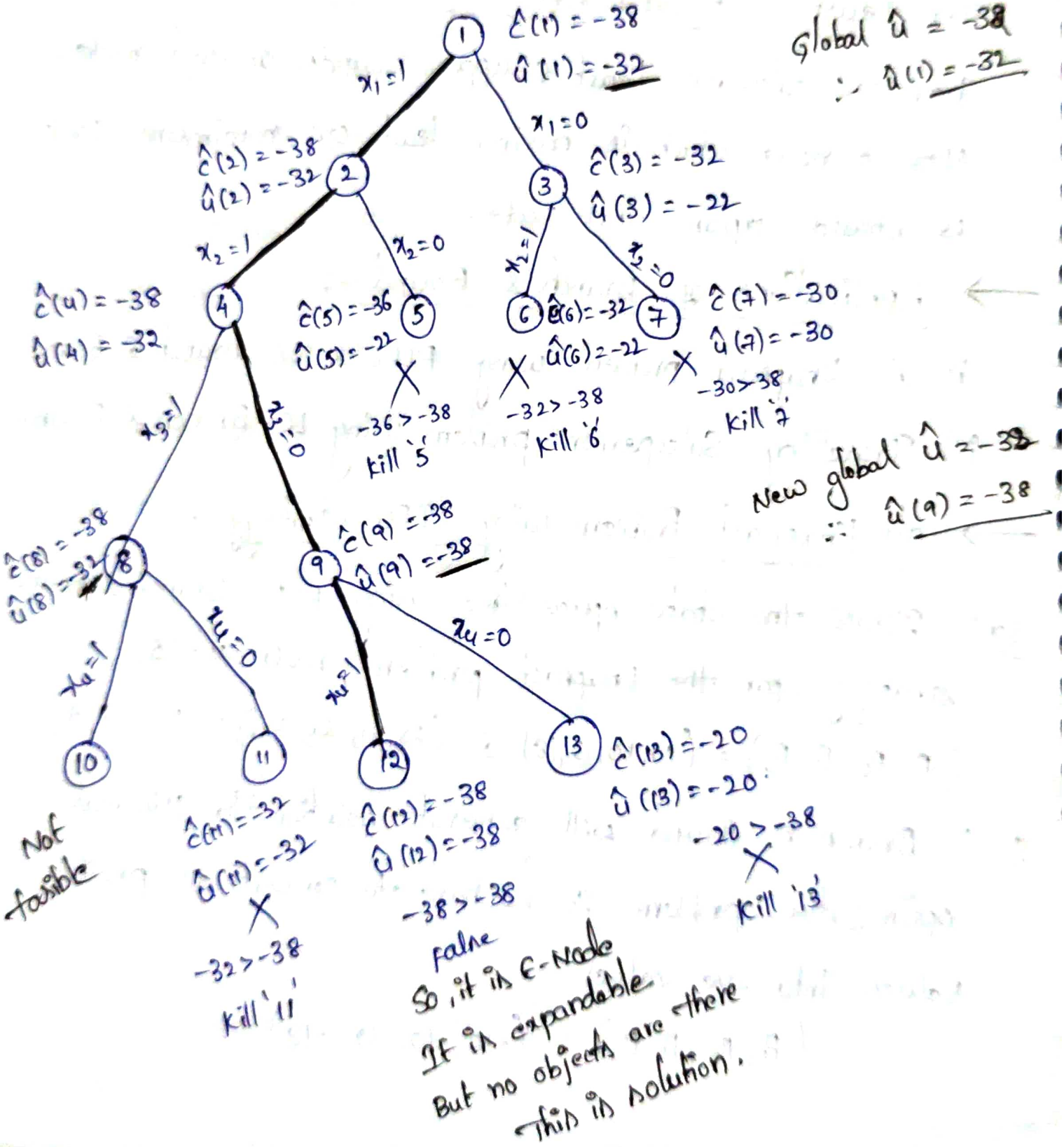
Subject to $\sum_{i=1}^{n} w_i x_i \leq M$

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

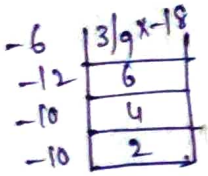
We will denote solution in form of 0 (or) 1, $x_i = 0$ (or) 1.

At each node we have to calculate $\hat{c}$ - lower bound & $\hat{u}$ - upper bound values. $\hat{c}$: fractions are allowed

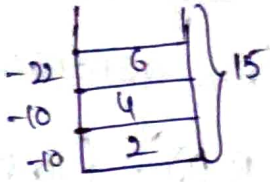
$\hat{u}$: fractions are not allowed.



for Node '1':

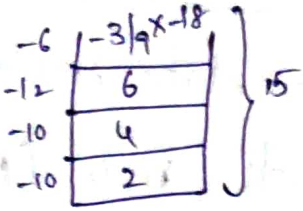


$\hat{c}(1) = -10-10-12-6 = -38$

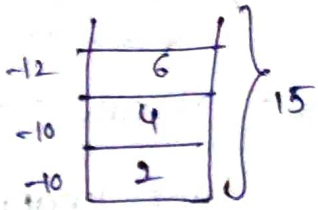


$\hat{u}(1) = -32$

for Node '2': $x_1 = 1$.

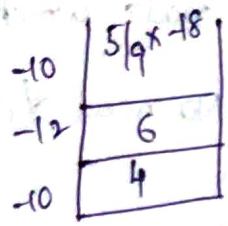


$\hat{c}(2) = -38$

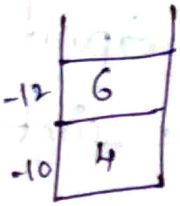


$\hat{u}(2) = -32$

for Node '3': $x_1 = 0$

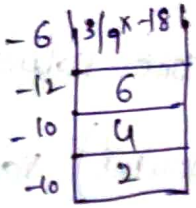


$\hat{c}(3) = -32$



$\hat{u}(3) = -22$

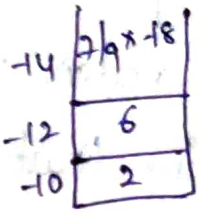
for Node '4': $x_1 = 1, x_2 = 1$



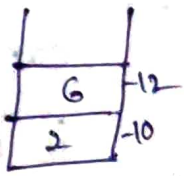
$\hat{c}(4) = -38$

$\hat{u}(4) = -32$

Node '5': $x_1 = 1$, $x_2 = 0$.



$\hat{c}(5) = -36$



$\hat{u}(5) = -22$

- Like this we have to calculate at each node. $\hat{u}(1)$ is global upper bound. At each node compare $\hat{c}$ value with global upper bound. If $\hat{c}(n) < \hat{u}$ then explore that node otherwise $\hat{c}(n) > \hat{u}$ then kill that node. construction of state space tree in FIFO order.
- At node '9', $\hat{u}(9) = -38$ which is less than global $\hat{u}(1)$. So, change global $\hat{u}$ as -38. compare $\hat{c}(n)$ with -38. Killing nodes '5', '6', '7'. Node '10' is infeasible solution why because, $x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1$. while inserting all objects weight of all the objects became '21'.

at node '12' {

∴ Final Solution is: $(x_1, x_2, x_3, x_4) = (1, 1, 0, 1)$.

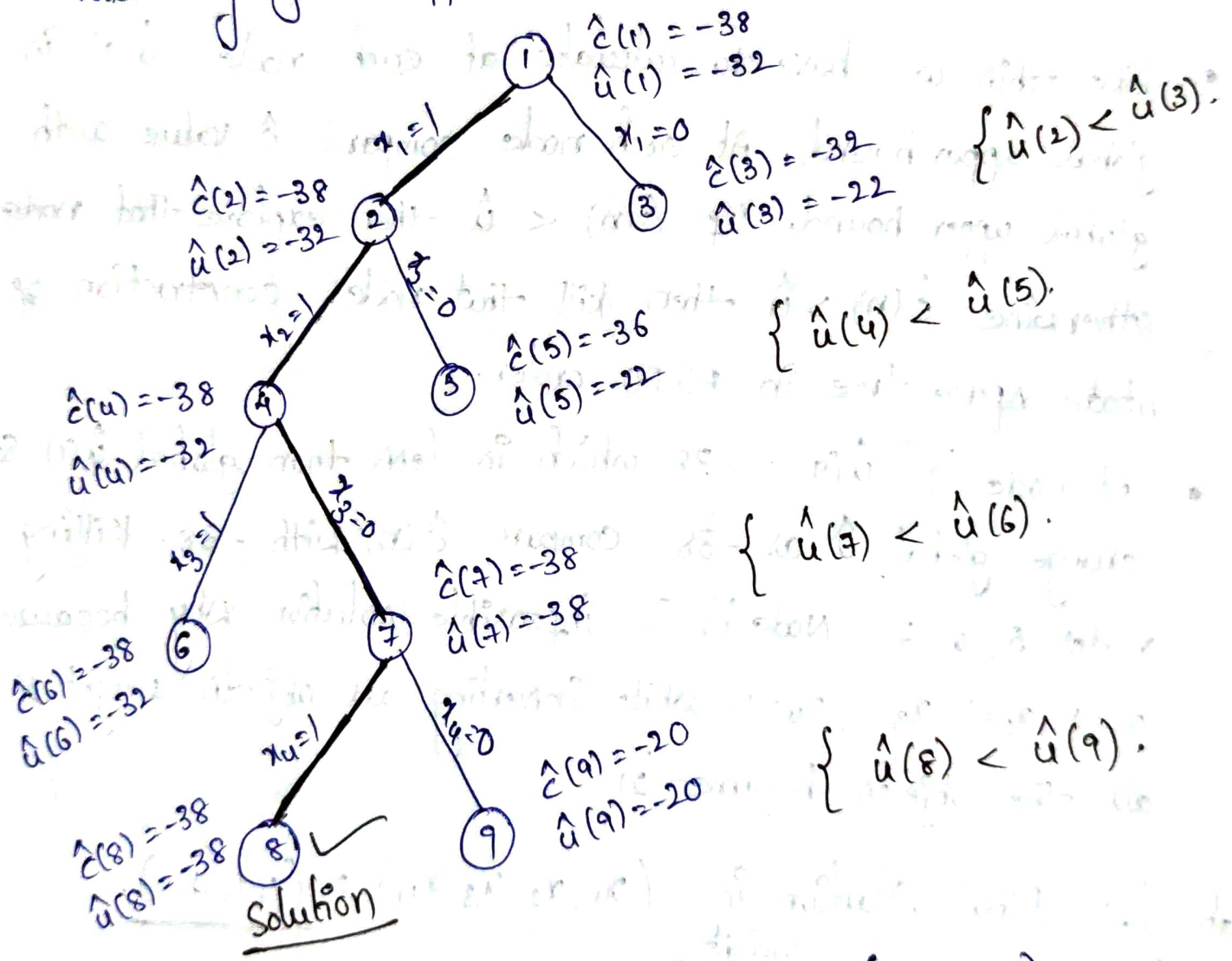
Maximum ~~Profit~~ weight is: $2+4+9 = 15$

Maximum Profit is: $10+10+18 = 38$

→ 0/1 Knapsack Problem Using LC Branch & Bound :

Ex: Draw the state space tree generated by LC B&B for knapsack, $n=4$; $(P_1, P_2, P_3, P_4) = (10, 10, 12, 18)$, $M=15$; $(W_1, W_2, W_3, W_4) = (2, 4, 6, 9)$.

Sol: This process is also same as before. But we have to explore nodes using Least cost value. Expand a node which is having least $\hat{u}$ (upper bound) value first. No need to take any global upper bound value



∴ Final Solution : $(x_1, x_2, x_3, x_4) = (1, 1, 0, 1)$.

Maximum Weight : $2 + 4 + 9 = 15$.

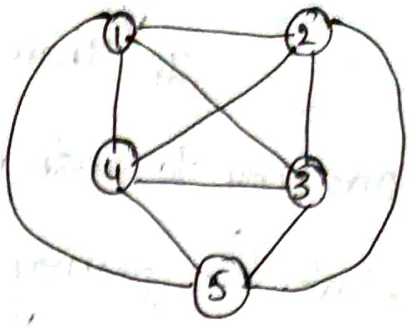
Maximum Profit : $10 + 10 + 18 = 38$.

→ Traveling Salesman Problem Using LC B&B :

If there are 'n' cities & cost of travelling from one city to another city are given. A salesperson has to start his journey from one city & has to visit all the cities exactly once & has to return to the starting place with shortest distance (or) minimum cost. For the given graph first we have to construct adjacency matrix.

- A row (or) column is said to be reduced, if it contains atleast one zero & all the remaining entries are non -ve.
- A Matrix is reduced iff every row & column are reduced.
- To reduce a matrix first have to perform "row minimization" (i.e) subtract minimum value of each each row from all elements in that row until we will get single '0' in each row.
- Next have to perform Column minimization "column minimization" (i.e) subtract select minimum value from each column & subtract that from all elements in that column. Until we have to get single '0' in each column.
- finally add that subtracted row value & column value we will get lower bound value (i.e) expected minimum cost of tour.

Ex: Construct state space tree for the given TSP using LC Branch & Bound.



Sol: Adjacency Matrix for 'G' is:

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

. Reduced cost = 25 ⇒ c(1).

Row Minimization : Subtract Mini. value from each row.

	1	2	3	4	5	
1	∞	20	30	10	11	−10
2	15	∞	16	4	2	−2
3	3	5	∞	2	4	−2
4	19	6	18	∞	3	−3
5	16	4	7	16	∞	−4
						21

⇒

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

Column Minimization : Subtract Mini. value from each col.

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

= 4.

⇒

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

Reduced Matrix with cost 25.

Add all Reduced row & column values : 21 + 4 = 25

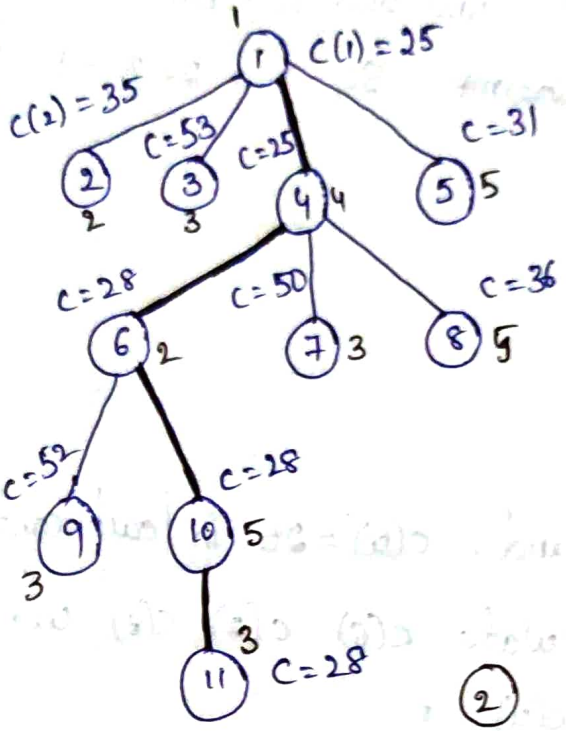
Atleast cost of the tour may be 25 (or) nearer to 25.

∵ c(1) = 25.

• Now by using Reduced matrix we will construct tree.

Global upper value is: α & lower cost 'c': 25 (c(1)).

1, 2, 3, 4, 5 – Node numbers in G.



c(2) – person can reach from 1 to 2. So make '1' row & '2' column as 'α'. person can't come back from 2 to 1. So, make 2 to 1 as 'α' in the Reduced Matrix.

c(3) – person travel from 1 to 3 and can't come back from 3 to 1. So, make 1 row 3 column & 3 to 1 as α in Reduced Matrix.

(2)

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

Matrix is already reduced.

$c(2) = c(1,2) + r + \hat{r}$

$= 10 + 25 + 0$ (Reduction in current matrix)

$= 35$.

(3)

	1	2	3	4	5	
1	α	α	α	α	α	
2	1	α	α	2	0	0
3	α	3	α	0	2	0
4	4	3	α	α	0	0
5	0	0	α	12	0	0
	0	0		0	0	

$c(3) = c(1,3) + r + \hat{r} = 17 + 25 + 11 = 53$

	1	2	3	4	5	
1	α	α	α	α	α	0
2	12	α	2	2	0	0
3	α	3	α	0	2	0
4	15	3	α	α	0	0
5	11	0	0	12	α	0
	-11	0	0	0	0	

Here 1st column don't have '0'. Reduce matrix by subtracting '11' from 1st column for c(3).

(4)

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

Already Reduced.

$c(4) = c(1,4) + r + \hat{r}$

$= 0 + 25 + 0 = 25$.

(5)

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

To get reduced Matrix here we are subtracting 2 from 2nd row and 3 from 3rd row column. So, $\hat{r} = 2+3 = 5$.

Already Reduced with cost '5'.

$c(5) = c(1,5) + r + \hat{r}$

$= 1 + 25 + 5$

$= 31.$

- This is least cost Branch & Bound. $c(4) = 25$ is least cost. expand '4'. $4 \rightarrow 2,3,5$. To calculate $c(6)$, $c(7)$, $c(8)$ use 4th matrix as base matrix. (i.e). $c(4)$.

(4)

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

⇒

(6)

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

Make row '4' & column '2' as '∞' and 2 to 1 as '∞'.

Already Reduced

$c(6) = c(4,2) + r + \hat{r} = c(4,2) + c(4) + \hat{r}.$

$= 3 + 25 + 0 = 28$

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

Not Reduced.

(7)

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

Made row '4' & column '3' as '∞' and 3 to 1 as ∞.

Reduced with cost '13'.

$c(7) = c(4,3) + c(4) + \hat{r}$

$= 12 + 25 + 13 = 50.$

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

Not Reduced

⑧

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

Made row '4' & column '5' as '∞' and 5 to 1 as ∞.

Reduced with cost = 11.

$c(8) = c(4,5) + c(4) + \hat{r} = 0 + 25 + 11 = 36$.

- Among c(6), c(7), c(8) cost at node '6' is least $c(6) = 28$. Expand '6'. $2 \rightarrow 3,5$. To calculate c(9), c(10) use 6th matrix as base matrix (i.e). c(6).

⑥

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

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

Made 2nd row & 3rd column as '∞' and 3 to 1 as '∞'.

Not Reduced.

⑨

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

$c(9) = c(2,3) + c(6) + \hat{r}$

$= c(2,3) + c(6) + 13 = 52$

Reduced with cost '13'.

⑩

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

Already Reduced

$c(10) = c(2,5) + c(6) + \hat{r}$

$= 0 + 28 + 0 = 28$

⑪

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

Already reduced

$c(11) = c(5,3) + c(10) + \hat{r}$

$= 0 + 28 + 0$

$= 28$

- Among c(9), c(10) node '10' is least. Expand Node '10'.
- finally the tour is: 1-4-2-5-3-1. with cost = 28.