UNIT - IV BACKTRACKING & BRANCH AND BOUND.

Backtracking :- General method, applications - 8-queen problem, sum of subsets, graph coloring, Hamiltonian cycles.

Branch and Bound :- General method, Applications - Travelling sales person, 0/1 Knapsack problem - LC Branch and Bound solution, FIFO Branch and Bound solutions.

BACKTRACKING

GENERAL METHOD :-

→ Backtracking is one of the most general technique.

→ In this technique, we search for the set of solutions (or) optimal solution which satisfies some constraints.

→ one way of solving a problem is by exhaustive search, we enumerate all possible solutions and see which one produces the optimum result.

For example : Knapsack problem.

→ We look at every possible subset objects and find out one which has the greatest profit value and at the same time not greater than the weight bound.

→ Backtracking is a variation of exhaustive search, where the search is refined by eliminating certain possibilities.

→ Backtracking is usually faster method than an exhaustive search.

→ In the backtracking method,

1. The desired solution is expressible as an 'n' tuple $(x_1, x_2, \ldots x_n)$ where x_i is chosen from some finite set S_i.
2. The solution maximizes (or) minimizes (or) satisfies a criterion from function $C(x_1, x_2, \ldots, x_n)$.

→ The problem can be categorized into three categories.

1. for Instance:-

→ for a problem p let C be the set of constraints for p. Let D be the set containing all solutions satisfying C then.

2. Finding whether there is any feasible solution? - is the decision problem.
3. What is the best solution? - is the optimization problem.

→ Listing of all the feasible solution - is the enumeration problem.

→ The basic idea of backtracking is to build up a vector, one component at a time & to test whether the vector being formed has any chance of success.

→ The major advantage of this algorithm is that we can realize the fact that the partial vector generated does not lead to an optimal solution. In such a situation that vector can be ignored.

→ Backtracking algorithm determines the solution by systematically searching the solution space. (i.e, set of all feasible solutions) for the given problem.

→ Backtracking is a depth first search with some bounding function. All solutions using backtracking are required to satisfy a complex set of all constraints.

→ The constraints may be explicit (or) implicit.

→ Explicit constraints are rules, which restrict each vector element to be chosen from the given set.

→ Implicit constraints are rules, which determine which tuples in the solution space, actually satisfy the criterion function.

Some terminologies used in Backtracking:-

→ Backtracking algorithms determine problem solutions by systematically searching for the solutions using tree structure.

1. Problem state:-
→ Each node in the tree is called a problem state.

2. State space:-
→ All paths from the root to other nodes define the state space of the problem.

3. Tuple:-
→ The solution states are those problem states 's' for which the path from root to 's', defines a tuple in the solution space.

4. Solution states:-
→ In some trees, the leaves define the solution states.

5. Answer states:-
→ These are the leaf nodes which correspond to an element in the set of solutions, these are the states which satisfy the implicit constraints.

6. Live node:-
→ A node which is been generated and all whose children have not yet been generated is called live node.

7. E-node:-
→ The live node whose children are curre-ntly being expanded is called E-node.

8. Dead node:-

→ A dead node is a generated node which is not be expanded further (or) all of whose children have been generated.

APPLICATIONS:-

1. N Queens Problem.
2. Sum of subsets problem.
3. Graph coloring.
4. Hamiltonian cycles.

8-QUEEN PROBLEM:-

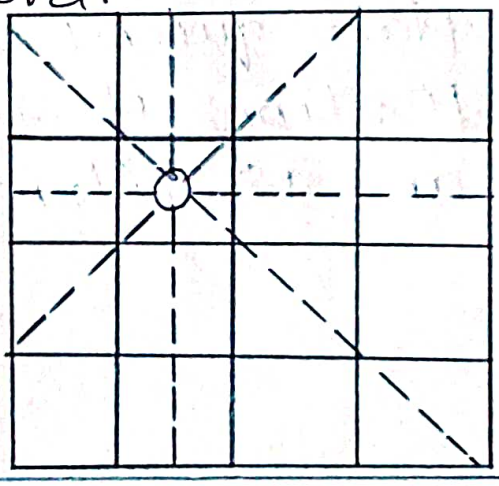
Problem Statement:-

→ The n queen's problem can be stated as follows.

→ Consider a nxn chessboard on which, we have to place 'n' queens. So that no two queens attack each other by being in the same row (or) in the same column (or) on the same diagonal.

For example:-

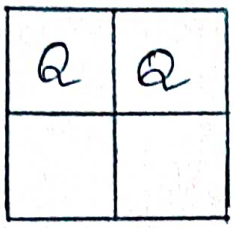
Consider 4x4 board:-



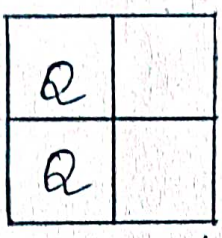
The next Queen - if is placed on the paths marked by dotted lines, then they can attack each other.

2 Queen's problem is not solvable:-

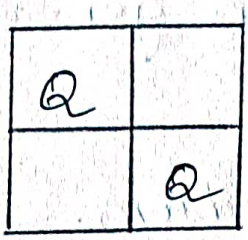
→ Because 2-queens can be placed on 2×2 chessboard as,



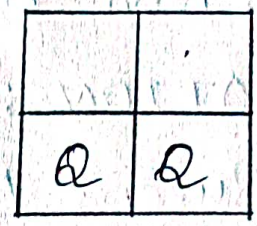
Illegal



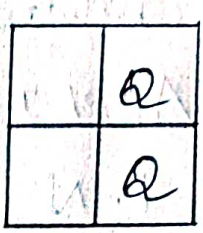
Illegal



Illegal

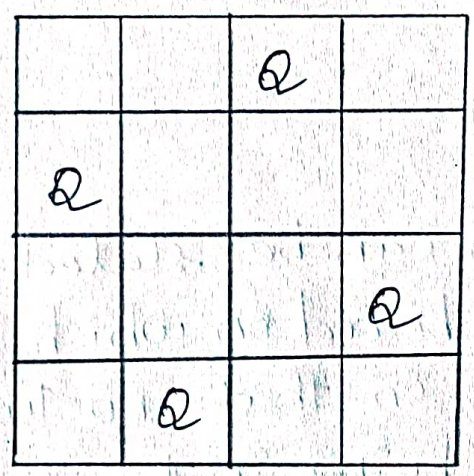


Illegal



Illegal

But 4-queen's problem is solvable:-



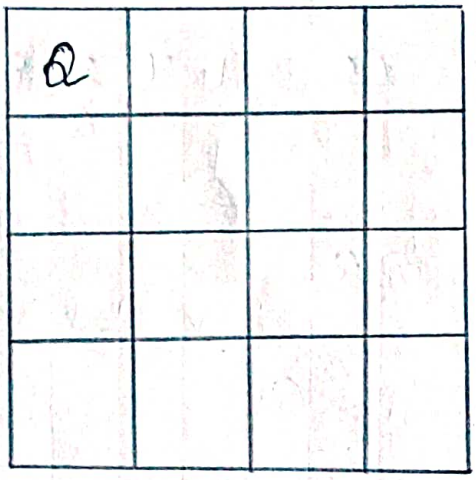
← No two queens can attack each other.

How to solve n-queens problem?

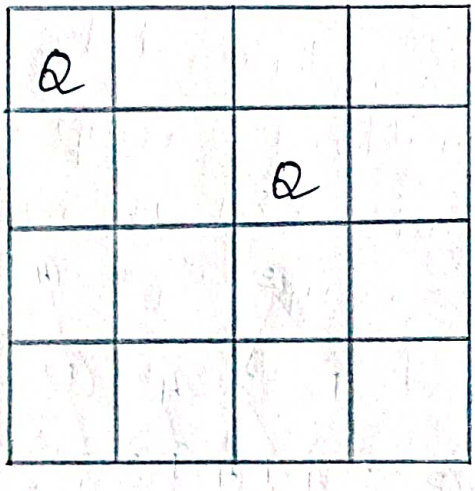
Let us take 4-queen's & 4×4 chessboard.

Step 1:- Now we start with empty chessboard.

Step 2:- Place queen 1 in the first possible position of its row i.e 1st row and 1st column.

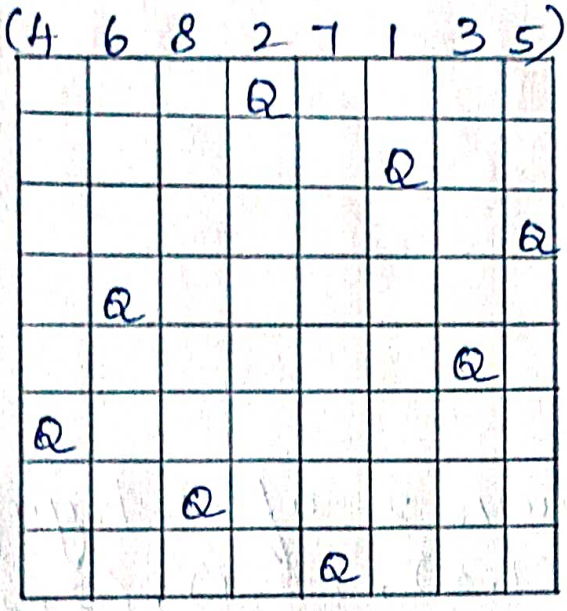


Step 3:- Then place queen 2 after trying unsuccessful place - 1 (1,2), (2,1), (2,2) at (2,3) we can place it i.e 2nd row & 3rd column.

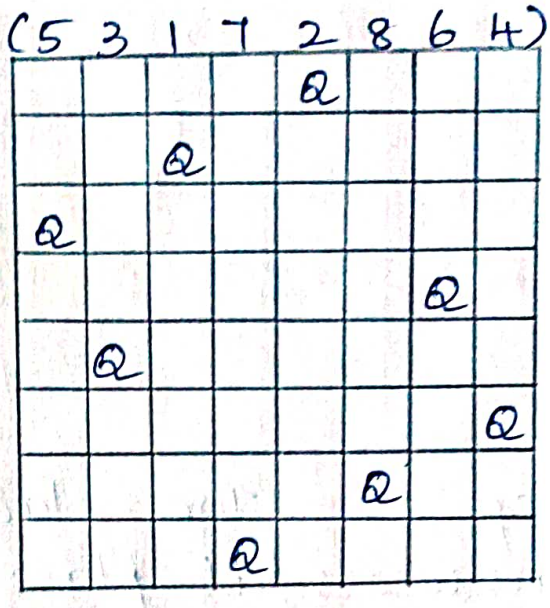


Step 4:- This is the dead end.
Because a 3rd queen cannot be placed in next column. As there is no acceptable position for queen 3. Hence algorithm backtracks and places the 2nd queen at (2,4) position.

8-queens problem:-



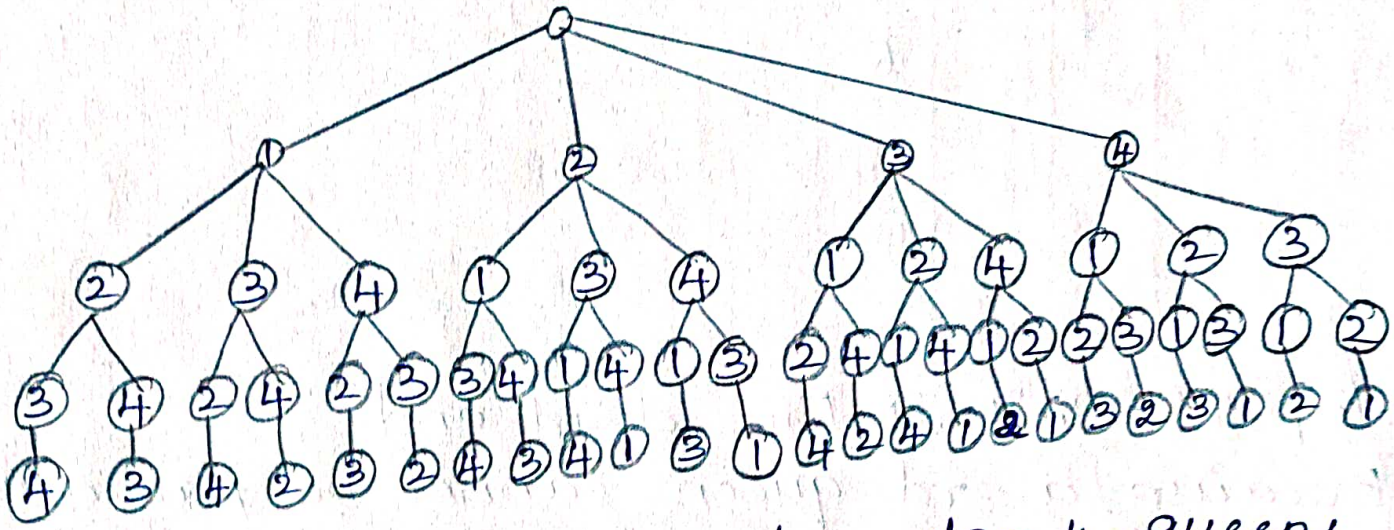
Solution 1



Solution 2

→ Formula to check

$$|i-k| = |j-l|$$



Sample state space tree for 4-queens problem.

→ solution representation using state space tree for 8-queens problem.

$$1 + \sum_{j=0}^{7} \left[\prod_{i=0}^{j} (8-i) \right] = 69821$$

(Total nodes)

(To find how many number of nodes in state space tree).

SUM OF SUBSETS:-

Problem statement:-

→ Let $S = \{s_1, \ldots s_n\}$ be a set of n positive integers, then we have to find the subset whose sum is equal to given positive integer d.

→ It is always convenient to sort the set's elements in ascending order. That is,

$$s_1 \leq s_2 \leq \ldots \leq s_n$$

→ Let us first write an general algorithm for sum of subset problem.

Algorithm:-

→ Let s be a set of elements and d is the expected sum of subsets. Then,

Step 1:- Start with a empty set.

Step 2:- Add to the subset, the next element from the list.

Step 3:- If the subset is having sum d then stop with that subset as solution.

Step 4:- If the subset is not feasible (or) if we have reached to end of the set then backtrack through the subset until we find the most suitable value.

Step 5 :- If the subset is feasible then repeat step 2.

Step 6 :- If we have visited all the elements without finding a suitable subset and if no backtracking is possible then stop without solution.

Example :-
consider a set $S = \{5, 10, 12, 13, 15, 18\}$ and $d = 30$. Solve it for obtaining sum of subset.

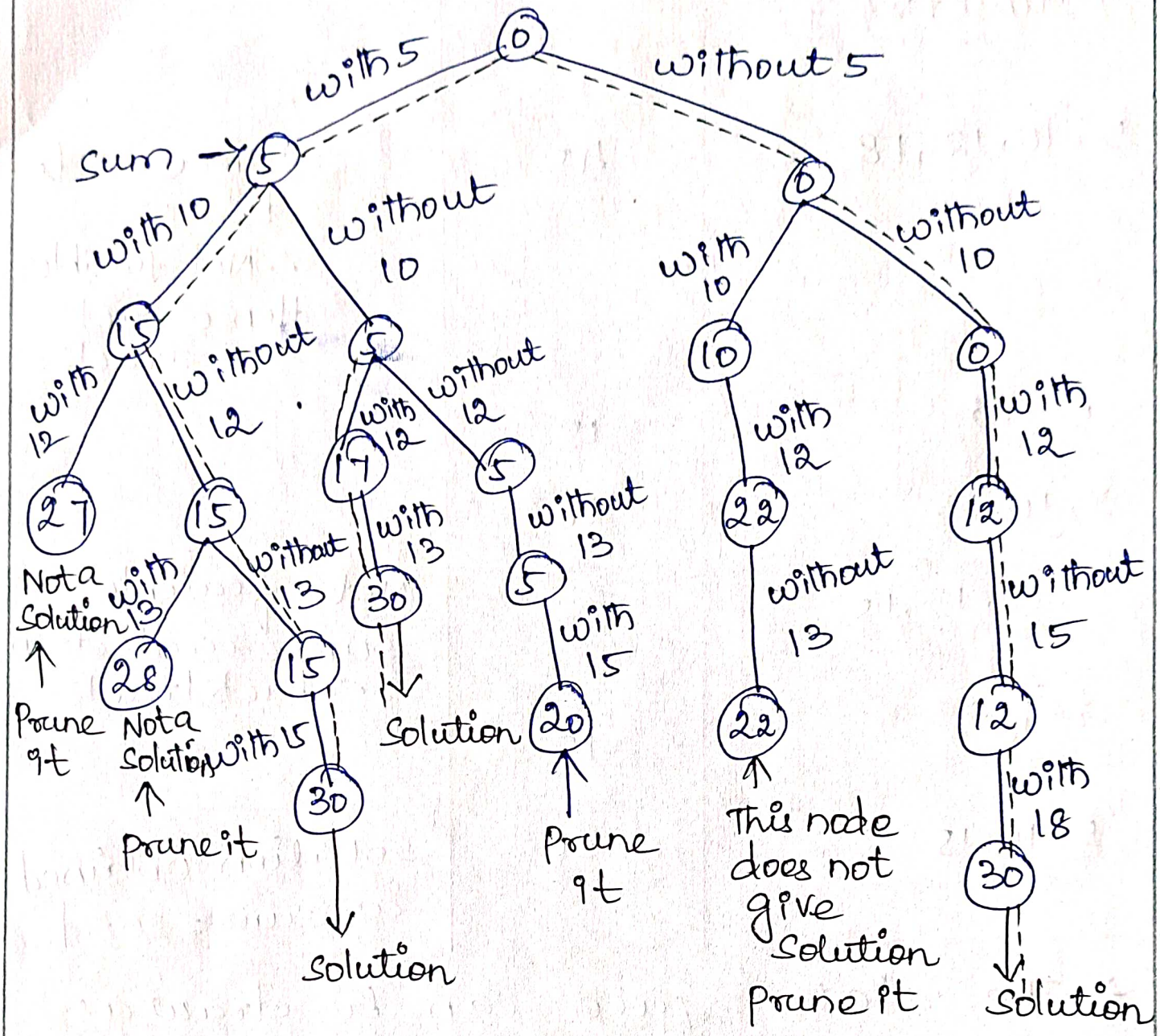
Initially subset $\{\}$	sum = 0	-
5	5	Then add next element
5, 10	15 $\therefore 15 < 30$	Add next element
5, 10, 12	27 $\therefore 27 < 30$	Add next element
5, 10, 12, 13	40 $\therefore 40 > 30$	Sum exceeds $d = 30$
5, 10, 12, 15	42	Hence backtrack Sum exceeds $d = 30$ $\therefore$ Backtrack applying

Initially subset={ }	sum=0	-
5, 10, 12, 18	45	sum exceeds d. ∴ Not feasible Hence back track.
5, 10	15	
5, 10, 13	28	
5, 10, 13, 15	33	Not feasible sum exceeds d=30 ∴ backtrack.
5, 10	15	
5, 10, 15	30	solution obtained as sum = 30 = d

→ The state space can be draw as follows.

{5, 10, 12, 13, 15, 18}

State space tree for sum of subset:-



GRAPH COLORING:-

→ Graph coloring is a problem of coloring each vertex in graph in such a way that no two adjacent vertices have same color and yet m-colors are used.

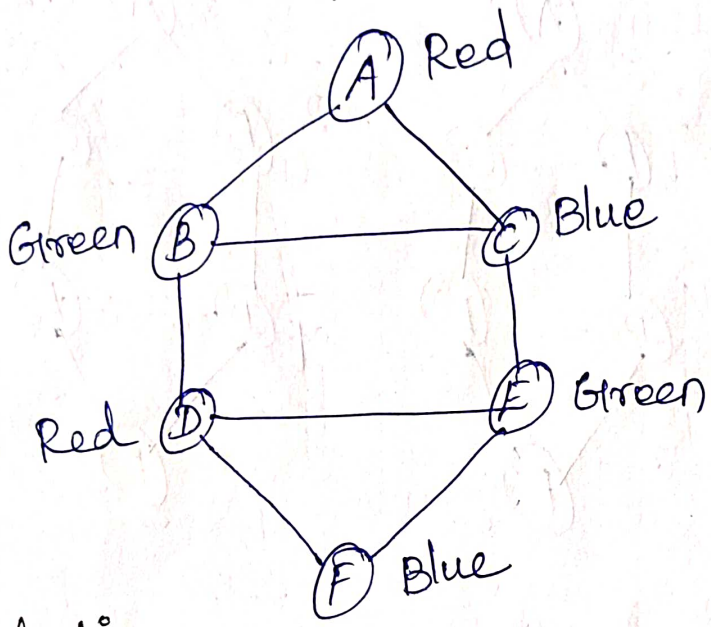
→ This problem is also called m-coloring problem.

→ If the degree of given graph is d then we can color it with d+1 colors.

→ The least number of colors needed to graph is called its chromatic number.

For example:-

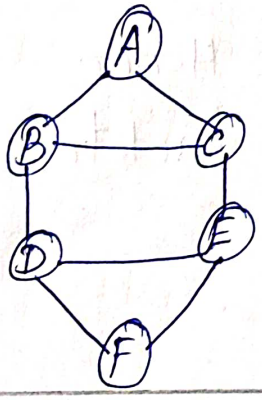
consider a graph given.



Solution:-

→ As given in figure, we require three colors to color the graph. Hence the chromatic number of given graph is '3'. We can use backtracking technique to solve the graph coloring problem as follows.

Step 01:-



→ A graph 'G' consists of vertices from A to f.

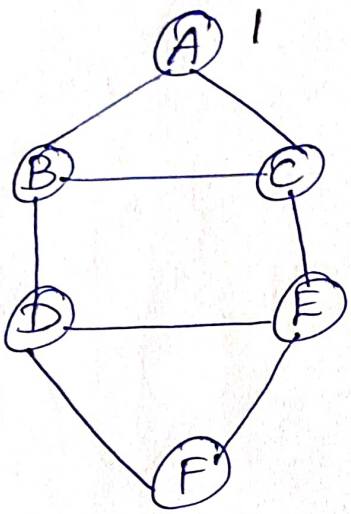
→ There are three colors used Red, Green & Blue.

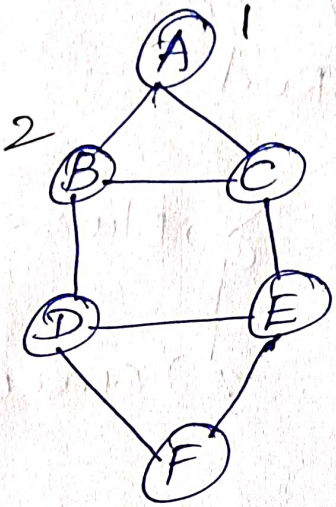
→ We will number them out.

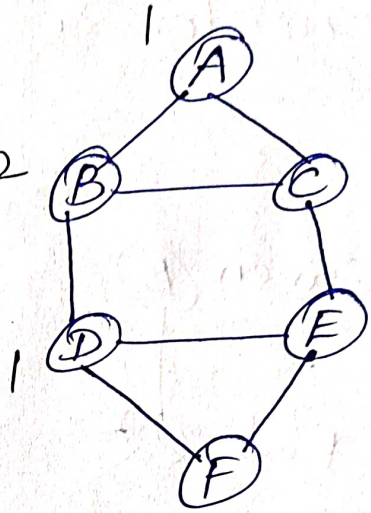
→ That means.

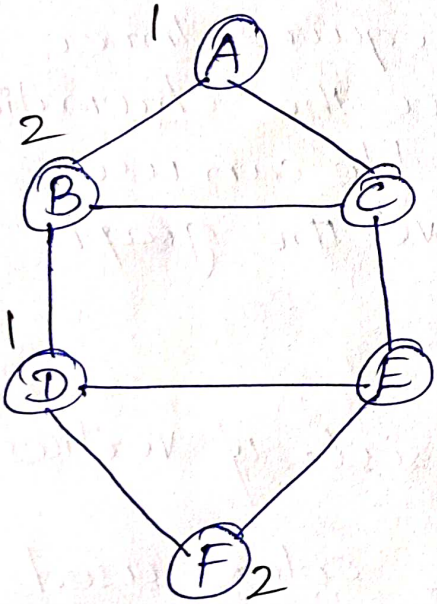
1 indicates → Red
2 indicates → Green
3 indicates → Blue color.

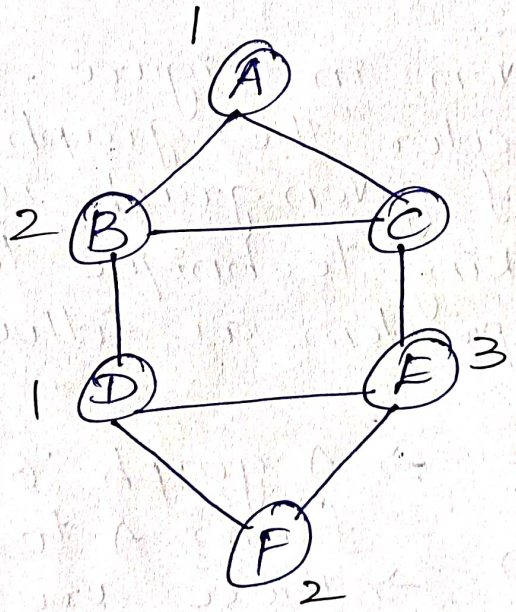
Step 02:-







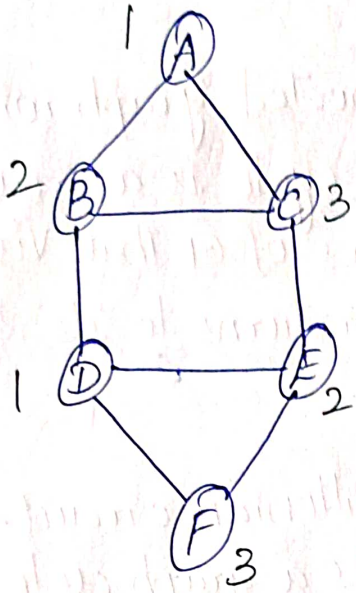




cannot assign
(1) (or)
(2) (or)
(3)

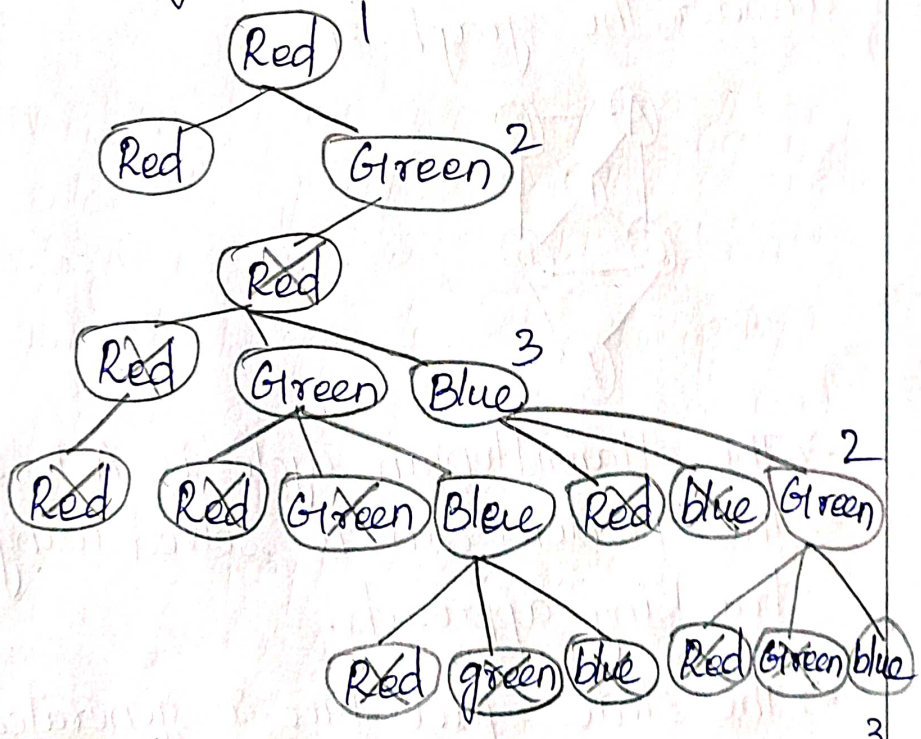
Hence backtrack

Step 03:-



→ Thus the graph coloring problem is solved.

→ The state space tree can be drawn for better understanding of graph coloring technique using backtrack approach.



∴ Here we have assumed, color index Red = 1, Green = 2, Blue = 3.

HAMILTONIAN CYCLES :-

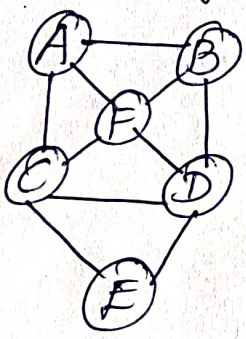
Definition :-

Let G=(V, E) be a connected graph with n vertices. A Hamiltonian cycle is a round trip path along n-edges of G that visits every vertex once and returns to its starting position.

→ It is called the Hamiltonian circuit.
→ Hamiltonian circuit is a graph cycle (i.e, closed loop) through a graph that visits each node exactly once.
→ A graph possessing a Hamiltonian cycle is said to be Hamiltonian graph.

For example :-

Consider the graph G1.

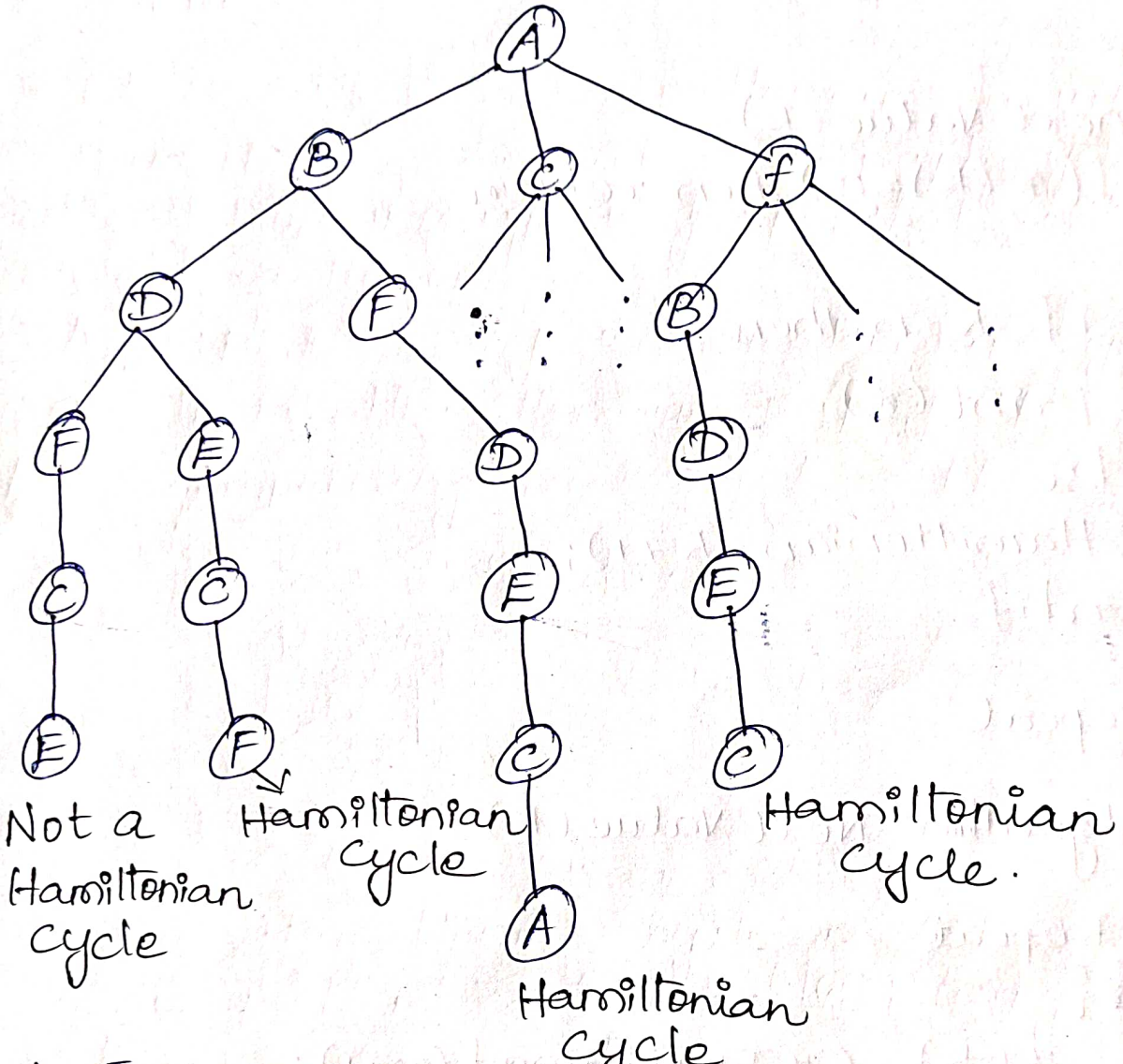


→ The Hamiltonian cycle is A-B-D-E-C-F-A.
→ This problem can be solved using back tracking approach.
→ The state space tree is generated in order to find all the Hamiltonian cycles in the graph.
→ Only distinct cycles are output of this algorithm.

→ Hamiltonian cycle can be identified as follows.

State space tree for finding Hamiltonian cycle:-

→ In below figure clearly the backtracking approach is adopted.



→ For instance A-B-D-F-C-E; here we get stuck.

→ For returning to A we have to revisit atleast one vertex.

→ Hence we backtrack and from 'D' node another path is choosen.

→ A-B-D-E-C-F-A which is Hamiltonian cycle.

Algorithm:-

```
Algorithm Hamiltonian(K)
{
  loop
    next value(K)
    if(x(K)=0) then return
    {
      if K=n then
      print(x)
    Else
      Hamiltonian(K+1);
    Endif
  }
  Repeat
}
  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) (k=n) and G1[x(n), x(1)] ≠0)
                                then return
}
until false
}
```

BRANCH AND BOUND

GENERAL METHOD:-

→ Branch and Bound (B & B) is general algorithm (or systematic method) for finding optimal solution of various optimization problems, especially in discrete and combinatorial optimization.

→ The Branch and Bound is very similar to backtracking in that a state space tree is used to solve a problem.

→ The differences are that the B & B method

1. Does not limit us to any particular way of traversing the tree.
2. It is used only for optimization problem.
3. It is applicable to a wide variety of discrete combinatorial problem.

→ Branch and Bound is rather general optimization technique that applies where the greedy method and dynamic programming fail.

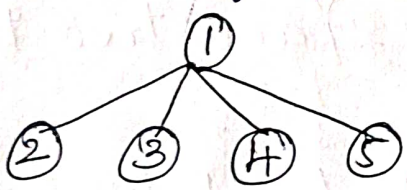
→ It is much slower, indeed, it often leads to exponential time complexities in the worst case.

→ In term B & B refers to all state space search methods in which all children of the "E-node" are generated before any other "live node" can become the "E-node".

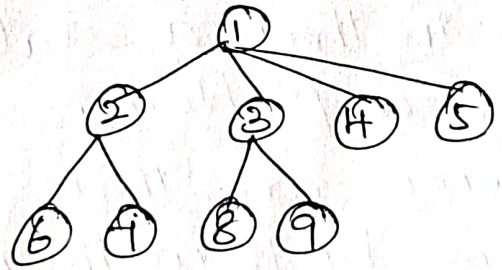
Live node:- Live node is a node that has been generated but whose children have not yet been generated.

E-node:- E-node is a live node whose children are currently being explored.

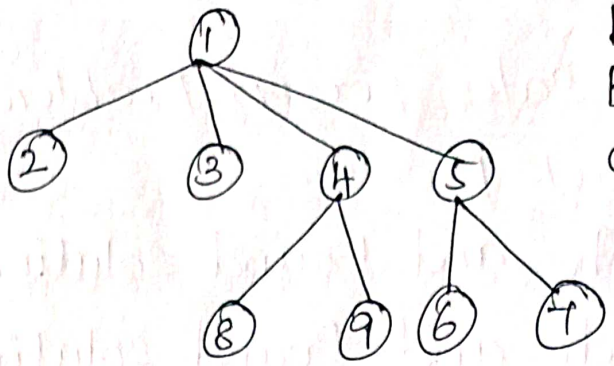
Dead node:- Dead node is a generated node that is not to be expanded or explored any further. All children of a dead node have already expanded.



Live node: 2, 3, 4 and 5



FIFO Branch & Bound (BFS) children of Enode are inserted in a queue.



LIFO Branch & Bound (D-Search) children of E-node are inserted in a stack.

→ Two graph search strategies, BFS and D-search (DFS) in which the exploration of a new node cannot begin until the node currently being explored is fully explored.

→ Both BFS & D-search (DFS) generalized to B & B strategies.

BFS:- Like state space search will be called FIFO (First In First Out) search as the list of live nodes is "First-in-first-out" list (or queue).

D-search (DFS):- Like state space search will be called LIFO (Last In First Out) search as the list of live nodes is a "last-in-first-out" list (or stack).

→ In backtracking, bounding function are used to help avoid the generation of sub-trees that do not contain an answer node.

→ We will use 3-types of search strategies in branch and bound.

1) FIFO (First In First Out) search.
2) LIFO (Last In First Out) search.
3) LC (Least count) search.

APPLICATIONS:-

1) Travelling Sales person problem.
2) 0/1 Knapsack problem.
3) LC Branch and Bound solution.
4) FIFO Branch and Bound solution.

TRAVELLING SALES PERSON PROBLEM:-

→ If there are 'n' cities and cost of travelling from one city to other city is given.

→ A salesman has to start from any one of the city and has to visit all the cities exactly once and has to return to the starting place with shortest distance (or) minimum cost.

→ Let $G1 = (V, E)$ be a directed graph defining an instance of the travelling sales person problem.

→ Let C_{ij} be the cost of the edge $(i,j) \in$ $C_{ij} = \infty$ if $(i,j) \notin (G1)$ & let $|V| = n$.

→ To use least cost branch and bound to search the travelling salesperson state space tree, we define a cost function $c(x)$ and two other function $\hat{c}(x)$ & $\hat{u}(x)$ such that $\hat{c}(x) \leq c(x) \leq \hat{u}(x)$.

Reduced cost matrix:-

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

→ A matrix is reduced iff every row & column is reduced.

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

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

With every node in state space tree, we associate a reduced cost matrix.

→ Let A be the reduced cost matrix for 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.

1) Change all entries in row i, column j of A to ∞.

2) Set A(j, 1) to ∞.

3) Apply row reduction and column reduction except for rows and columns containing ∞.

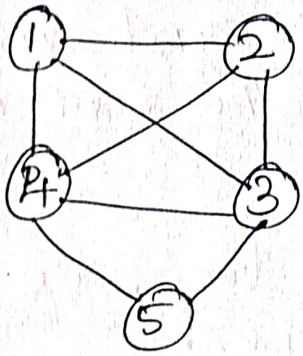
4) The total cost for node S can be calculated as,

$$\hat{c}(S) = \hat{c}(R) + A(i, j) + r$$

where r is the total amount subtracted in matrix.

Example:-

Solve the following instance of travelling sales person problem using LCBB.



Cost adjacency matrix

	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	∞

Step 01:-

Row Reduction:-

→ Note down the minimum value as per row wise and subtract it

	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

⇒

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

Column Reduction:-

→ Note down the minimum value as per column wise and subtract it.

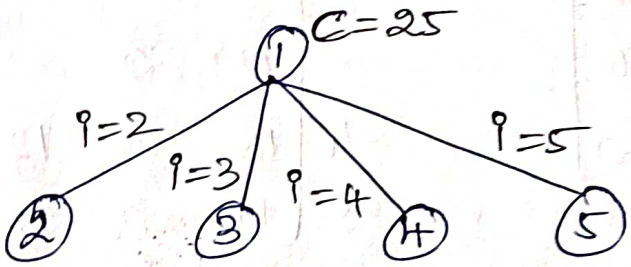
	1	2	3	4	5
1	∞	10	20	0	1
2	13	∞	14	2	0
3	1	3	∞	0	2
4	16	3	15	∞	0
5	12	0	3	12	∞
	1	0	3	0	0

⇒

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

∴ Total amount subtracted, r = 21+4 = 25

State space tree for the 1st node



Step 02:-

→ consider the path (1,2) : change all entries of first row and second column of reduced matrix to ∞ & set A(2,1) to ∞.

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

Row Reduction:-

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

⇒

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

Column Reduction:-

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

⇒

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

∴ Total amount reduced $\hat{r} = 0+0 = 0$

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

$= 10 + 25 + 0$

$= 35$

step 03:-

→ consider the path (1, 3):- change all entries of first row and third column of reduced matrix to ∞ and set A(3, 1) to ∞.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 15 & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & 12 & \infty \end{bmatrix} \end{array}$$

Row Reduction:-

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 15 & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & 12 & \infty \end{bmatrix} \end{array}\begin{array}{c} \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \Rightarrow \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 15 & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & 12 & \infty \end{bmatrix} \end{array}$$

Column Reduction:-

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ \\ \end{array}\begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 15 & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & 12 & \infty \end{bmatrix} \\ \begin{array}{ccccc} 11 & 0 & & 0 & 0 \end{array} \end{array} \Rightarrow \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix} \end{array}$$

$\therefore$ Total amount reduced $\hat{r} = 1$

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

$$= 25 + 17 + 11$$

$$= 53$$

Step 04 :-

→ consider the path (1,4) : change all entries of first row and fourth column to '∞' and set A(4,1) to ∞.

1st matrix Reduced :

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

Row Reduction :-

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

⇒

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

Column Reduction :-

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

⇒

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

∴ Total subtracted reduction $\hat{r} = 0$

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

$= 0 + 25 + 0$

$= 25$

Step 05:-
Consider the path (1,5): Change all entries of first row and 5th column to ∞ and set A(5,1) to ∞

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

Row Reduction:-

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

⇒

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

Column Reduction:-

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

⇒

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

∴ Total subtracted Reduction

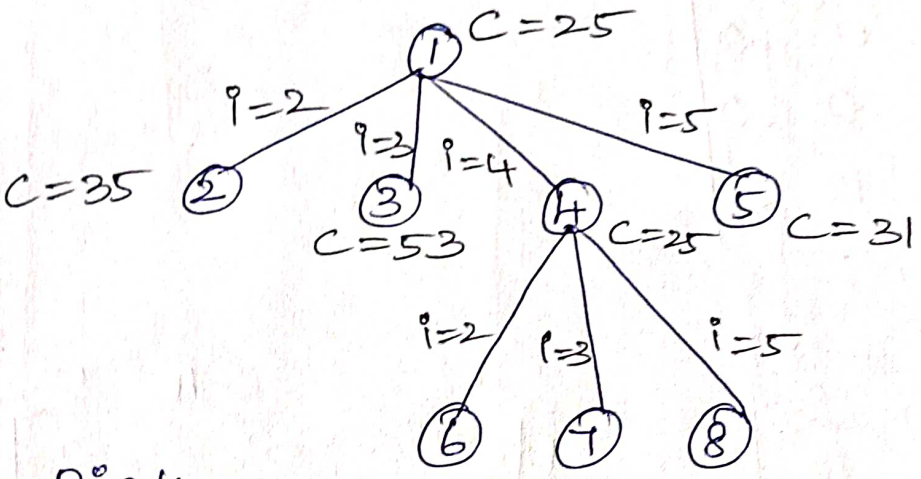
$$\hat{r} = 5$$

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

$$= 1 + 25 + 5$$

$$\hat{c}(5) = 31$$

State space Tree :-



→ Pick up the least cost from the state space tree and explore the child nodes for the node (4).

Reduced cost matrix for the node 4 is :-

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

Step 06 :-

→ consider the path (4,2): Change all entries of 4th row and 2nd column from reduced matrix as ∞ and set (2,1) to ∞.

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

Row Reduction :-

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

⇒

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

Column Reduction :-

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

⇒

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

∴ Total subtracted reduction $\hat{r} = 0$

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

$$= 3 + 25 + 0$$

$$\hat{C}(6) = 28$$

Step 07 :-

→ consider the path (4,3) : change all the entries of 4th row and 3rd column as ∞ and set (3,1) to ∞.

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

Row Reduction :-

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

⇒

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

Column Reduction :-

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

⇒

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

∴ Total subtracted reduction

$\hat{r} = 11 + 2 = 13$

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

$= 12 + 25 + 13$

$= 50$

Step 08:-

→ consider the path (4,5): change all entries of 4th row and 5th column to ∞ and set (5,1) to ∞.

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

Row Reduction:-

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

⇒

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

Column Reduction:-

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

⇒

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

∴ Total subtracted reduction

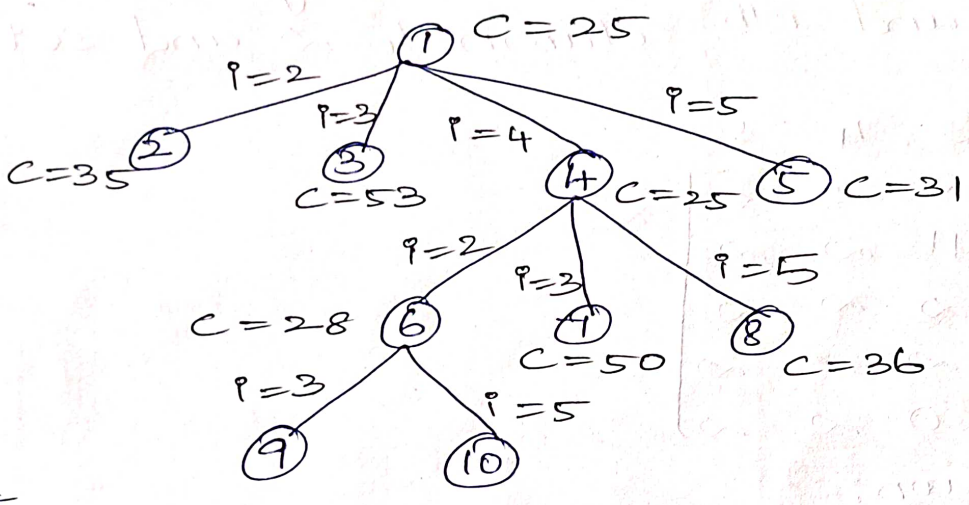
$$\hat{r} = 11 + 0 = 11$$

$$\hat{c}(8) = c(4,5) + \hat{c}(4) + \hat{r}$$

$$= 0 + 25 + 11$$

$$\hat{c}(8) = 36$$

state space tree:-



step 09:-

→ consider the path (2,3): change all entries of 2nd row and 3rd column to ∞ and set (3,1) to ∞

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

Row Reduction:-

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

$\Rightarrow$

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

Column Reduction:-

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

$\Rightarrow$

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

$\therefore$ Total subtracted reduction $\hat{r} = 13 + 0 = 13$

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

$= 11 + 28 + 13$

$= 52$

Step 10:-

→ Consider the path (2,5): Change all entries of 2nd row and 5th column to ∞ and set (5,1) to ∞.

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

Row Reduction:-

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

$\Rightarrow$

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

Column Reduction:-

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

$\Rightarrow$

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

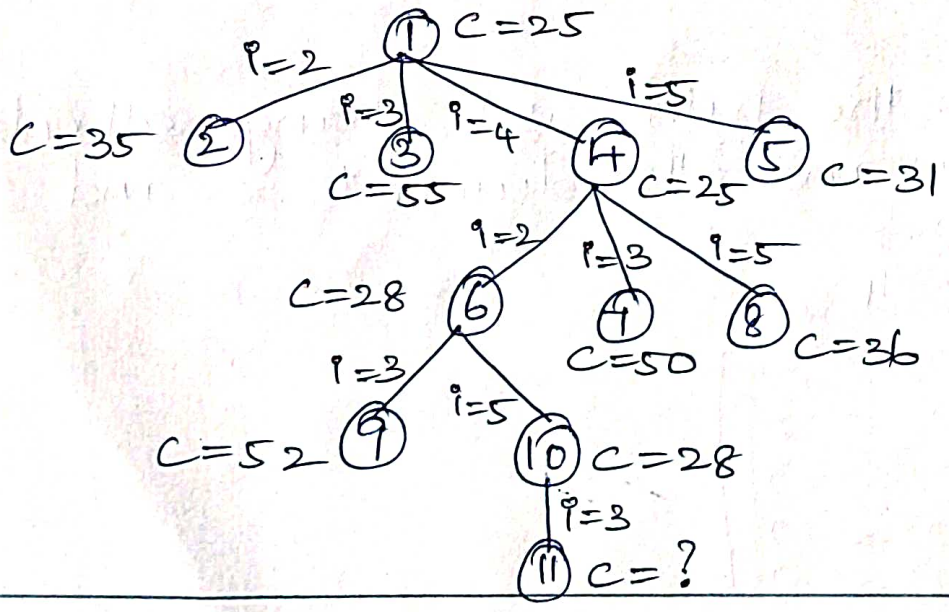
$\therefore$ Total subtracted reduction $\hat{r} = 0$

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

$= 0 + 28 + 0$

$= 28$

State space Tree:-



∴ The minimum cost in the state space tree is c = 28 for node 10.

10th matrix =

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

Step 11:-

→ consider the path (5,3): Change all entries of 5th row and 3rd column to ∞ and set (3,1) to ∞.

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

∴ Row Reduction : '0'

Column Reduction : '0'

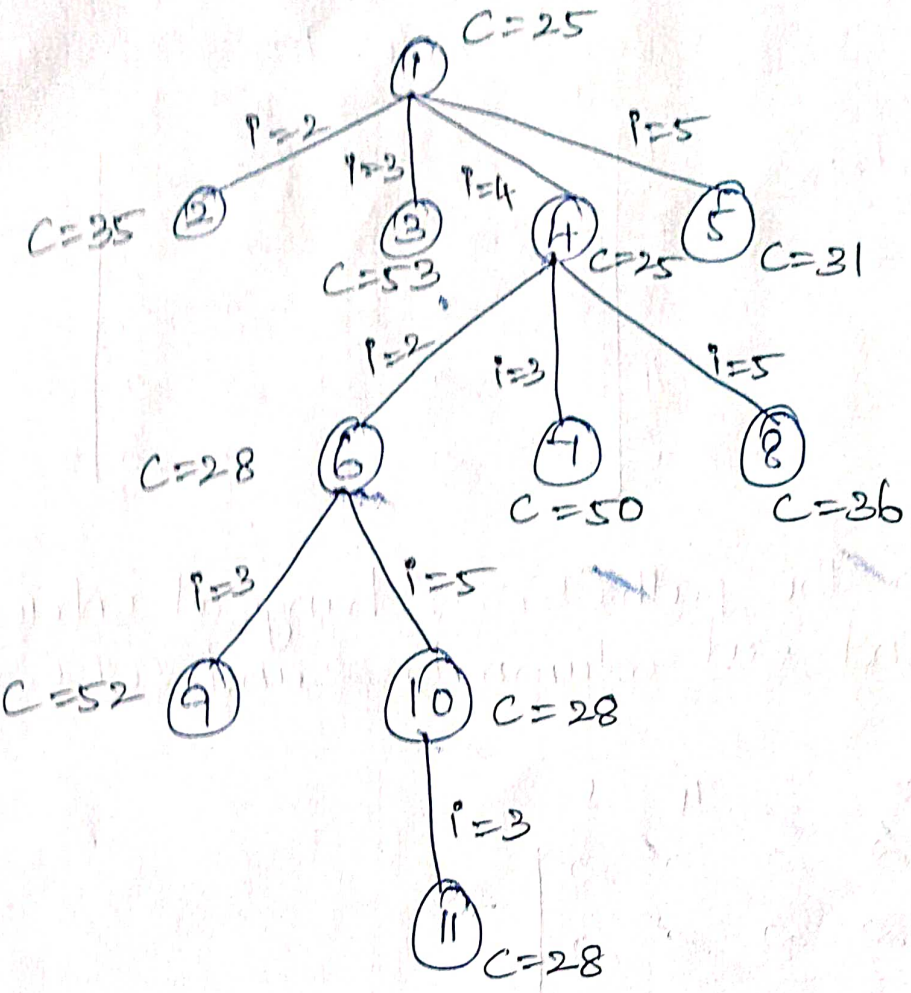
$\hat{r} = 0$

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

$= 0 + 28 + 0$

$\hat{c}(11) = 28$

Final state space Tree :-



∴ The path is 1 - 4 - 2 - 5 - 3 - 1

∴ The minimum tour cost is 28.

0/1 KNAPSACK PROBLEM :-

→ The 0/1 knapsack problem states that there are 'n' objects given & capacity of knapsack is 'm'.

→ Then select some objects to fill the knapsack in such a way that should not exceed the capacity of knapsack and maximum profit can be earned.

→ The 0/1 knapsack problem can be stated as

$$\text{Max } z = P_1x_1 + P_2x_2 + \cdots + P_nx_n.$$

for weights

$$w_1x_1 + w_2x_2 + \cdots + w_nx_n \leq m$$

$x_i = 0$ (or) 1

→ In a branch and bound technique is used to find solution to the knapsack problem. But we cannot directly apply the branch and bound technique to the knapsack problem.

→ Because, the branch and bound deals only the minimization problems.

→ We modify the knapsack problem maximi-zation to the minimization problem. The modified problem is,

Draw a portion of state space tree generated by LCBB by the following 0/1 knapsack problem. $n=4$, $(P_1, P_2, P_3, P_4) = (10, 10, 12, 18)$, $(w_1, w_2, w_3, w_4) = (2, 4, 6, 9)$ & $M = 15$.

items i	1	2	3	4
Profits	10	10	12	18
Weight	2	4	6	9

Step 1:- convert the profits to negative.

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

Step 2:-

→ place the first item in bag i.e W=2.

→ calculate the upperbound (u) and cost (c).

$u = -\sum p_i x_i$ (without fraction)

$c = -\sum p_i x_i$ (with fraction)

→ place the second item in bag w=4.

∴ 2+4=6

→ place the third item in bag i.e w=6

2+4+6 = 12

→ If we place 4th item the capacity exceeds

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

for weights

$W_1x_1 + W_2x_2 + \cdots + w_nx_n \leq m$

$x_i = 0$ (or) 1

→ Let two functions used c(x) & u(x)

c(x) = cost function

u(x) = upper bound function

$c(x) = -\sum p_i x_i$ & $u(x) = -\sum p_i x_i$

c(x) = calculate without fraction.

u(x) = calculate the cost without fraction.

→ Then select the node whose cost is minimum i.e,

$c(x) = \min\{c(l\text{child}(x)), c(r\text{child}(x))\}$

→ The problem can be solved by making a sequence of decisions on the variables $x_1, x_2, \ldots x_n$ level wise.

→ A decision on the variable x_i involves determining which of the values 0 (or) 1 is to be assigned, to it by defining c(x) recursi-vely.

→ The path from root to the leaf node whose height is maximum is selected & is the solution space for the 0/1 Knapsack problem.

∴ upper bound $u = -(10+10+12)$

$$\boxed{u = -32}$$

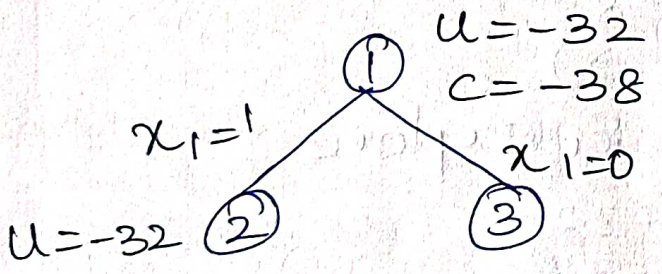
∴ cost $c = -(\leq p_i x_i)$ [with fraction]

$$= -(10+10+12+18/9 \times 3)$$

$$= -(10+10+12+6)$$

$$\boxed{c = -38}$$

State space tree:-



Step 3:-

For node 3, $(x_1 = 0)$

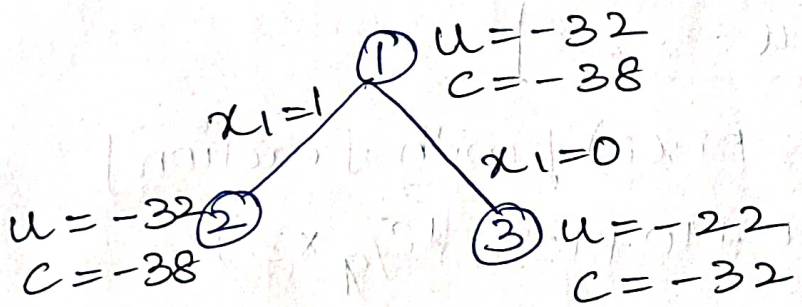
→ If 1st object is not included in the bag, then

weight (without fraction)	weight (with fraction)
4+6=10	4+6+5/9
	4+6+5=15

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

$c = -(10+12+18 \times 5/9)$
$= -(10+12+10)$
$= -32$

State space tree



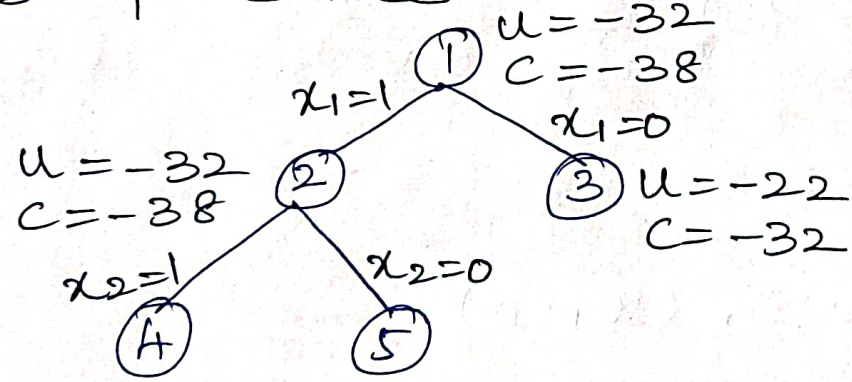
Step 4 :-

→ Select the minimum cost i.e,

$\min(\hat{c}(2), \hat{c}(3)) = \min\{-38, -32\}$
$= -38$
$= \hat{c}(2)$

∴ Choose the node 2 & explore

State space tree



Step 5 :-

For node 4 ($x_2 = 1$) :- Included 2nd object

weight (with fraction)

$w = 2+4+6+3/9$

$= 2+4+6+3$

$\boxed{w = 15}$

weight (without fraction)

$w = (2, 4, 6)$

$= 2+4+6$

$\boxed{w = 12}$

$c = -\sum p_i x_i$

$= -(10+10+12+\overset{2}{\cancel{18}} \times 3/9)$

$= -(10+10+12+6)$

$\boxed{c = -38}$

$u = -\sum p_i x_i$

$= -(10+10+12)$

$\boxed{u = -32}$

For node 4 ($x_2 = 0$) :- If 2nd object not included.

weight (with fraction)

$w = (2, 6, 7/9)$

$c = -\sum p_i x_i$

$= -(10+12+\overset{2}{\cancel{18}} \times 7/9)$

$= -(10+12+14)$

$\boxed{c = -36}$

weight (without fraction)

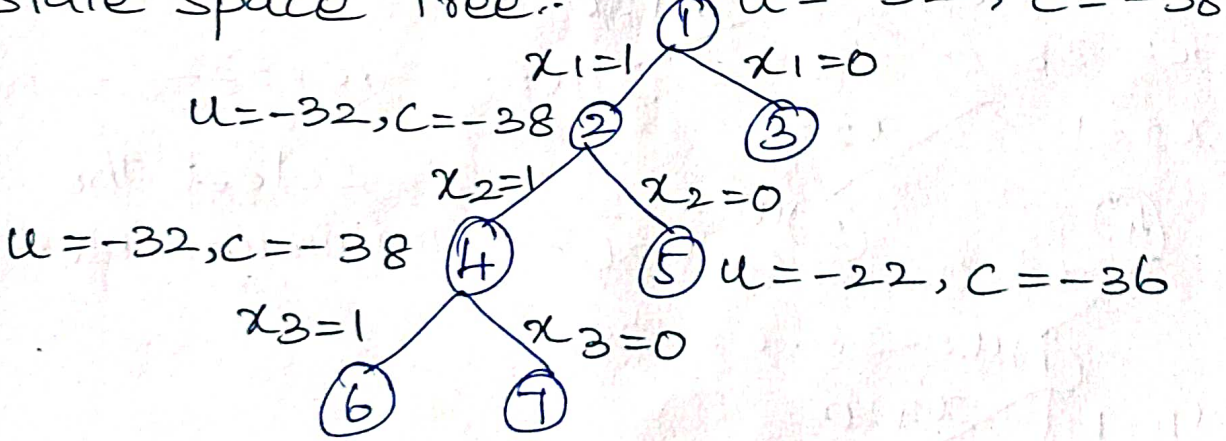
$w = (2, 6)$

$u = -\sum p_i x_i$

$= -(10+12)$

$\boxed{u = -22}$

State Space Tree:-



∴ Select the minimum cost i.e, $\hat{c}(4)$ and explore the nodes.

Step 6:-

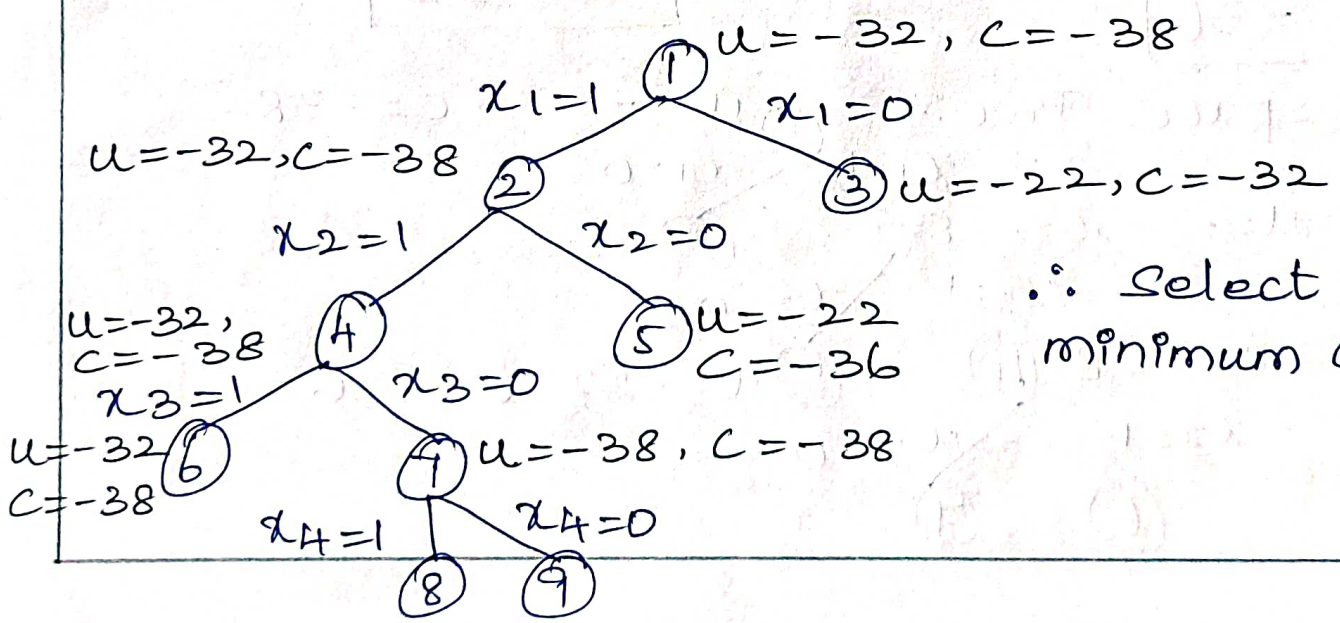
For node 6 ($x_3=1$)

weight (with fraction)	weight (without fraction)
$w=(2,4,6,3)$	$w=(2,4,6)$
$c=-\sum p_i x_i$	$u=-(10+10+12)$
$=-(10+10+12+\frac{2}{3}\times 18)$	$u=-32$
$c=-38$	

For node 7 ($x_3=0$):-

→ If 3rd object is not included.

weight (with fraction)	weight (without fraction)
$w=(2,4,9)$	$w=(2,4,9)$
$c=-\sum p_i x_i$	$u=-\sum p_i x_i$
$=-(10+10+18)$	$=-(10+10+18)$
$c=-38$	$u=-38$



∴ Select the minimum cost.

Step 7:-

For node 8 ($x_4 = 1$)

→ If 4th object is included in the bag

$w = (2, 4, 6, 3/8)$

$c = -\sum p_i x_i$

$= -(10 + 10 + 12 + 18 \times 3/9)$

$\boxed{c = -38}$

$w = (2, 4, 6)$

$u = -(10 + 10 + 12)$

$\boxed{u = -32}$

For node 8 ($x_4 = 0$)

→ If 4th object is not included in the bag.

weight (with fraction)

$w = (2, 4)$

$c = -\sum p_i x_i$

$= -(10 + 10)$

$\boxed{c = -20}$

weight (without fraction)

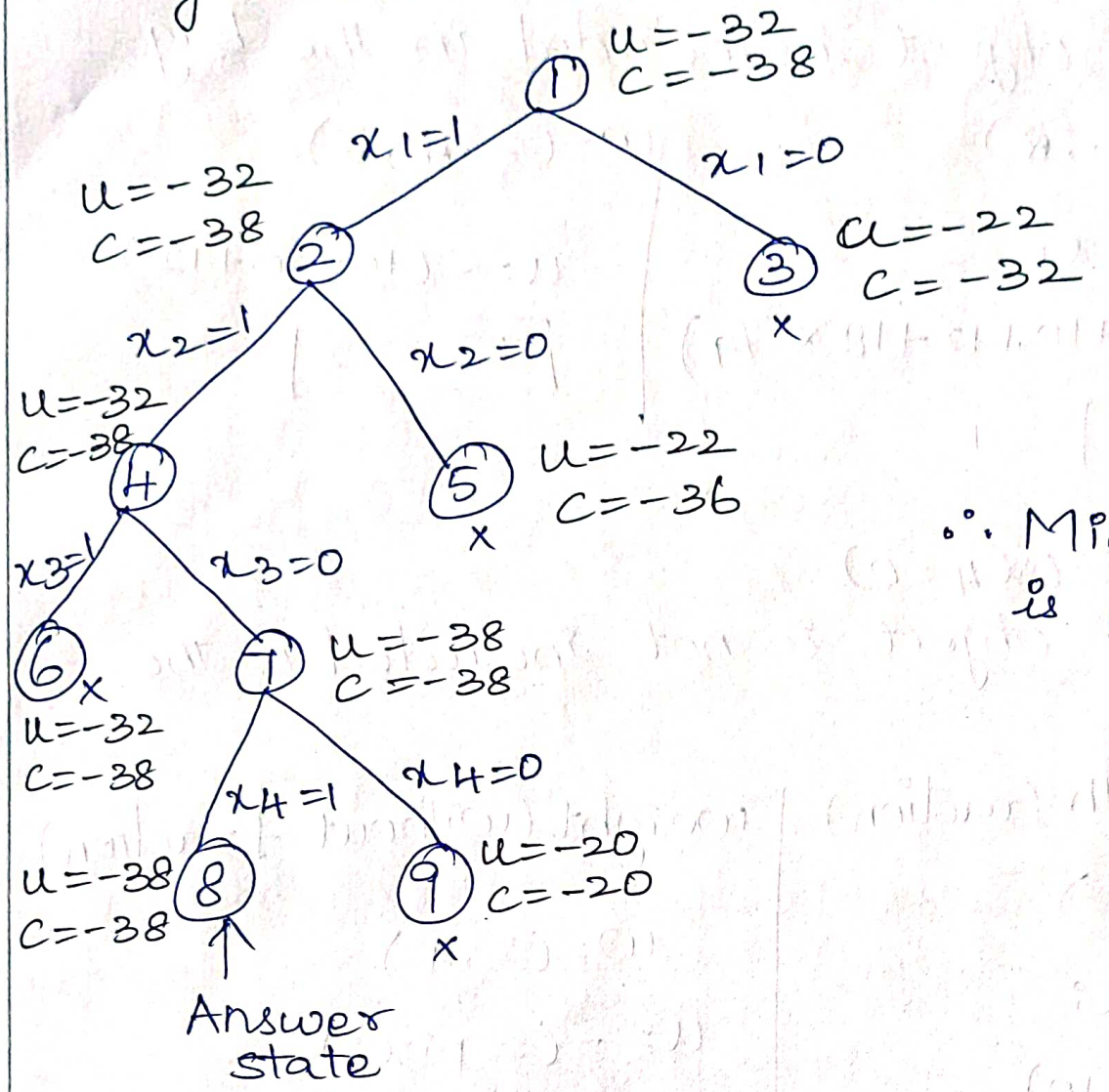
$w = (2, 4)$

$u = -(\sum p_i x_i)$

$= -(10 + 10)$

$\boxed{u = -20}$

Final state space tree for 0/1 knapsack using LCBB.



∴ Minimum node is $\hat{c}(8)$

∴ The path is 1 - 2 - 4 - 7 - 8

∴ The solution for 0/1 knapsack is

$x = \{1, 1, 0, 1\}$

∴ Maximum profit = 10 + 10 + 0 + 18

= 38

∴ weight = 2 + 4 + 9 = 15

LC BRANCH AND BOUND SOLUTION:-

→ The LC branch and bound solution can be obtained using fixed tuple size formulation.

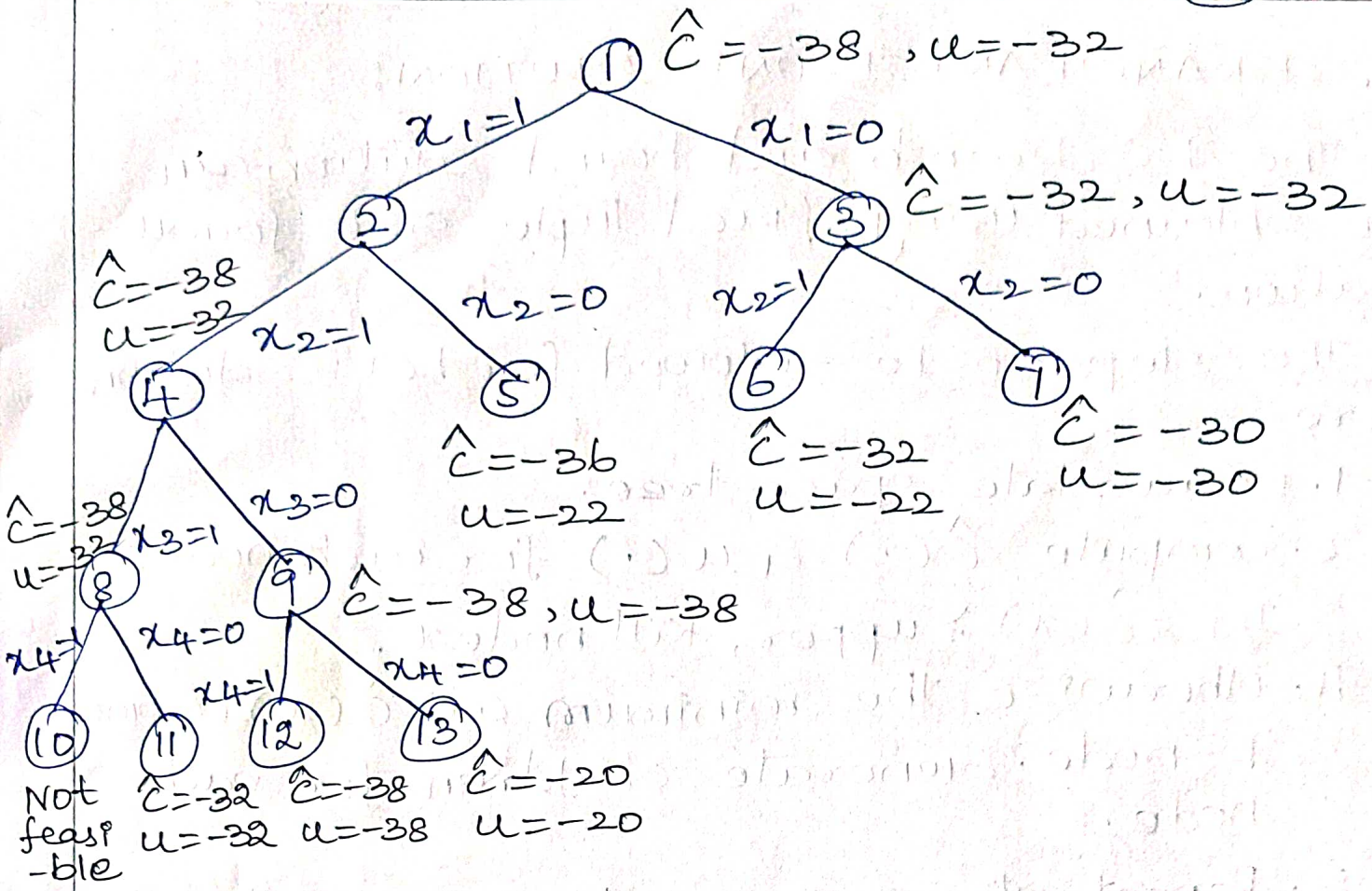
→ The steps to be followed for LCBB solution are

1. Draw state space tree.
2. Compute $\hat{c}(\cdot)$ & $u(\cdot)$ for each node.
3. If $\hat{c}(x) >$ upper, kill node x.
4. Otherwise the minimum cost $\hat{c}(x)$ becomes E-node. Generate children for each node.
5. Repeat step 3 & 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.

FIFO Branch and Bound solutions:-

→ TO understand FIFO branch and bound, we will consider the Variable tuple size formulation.

→ The state space tree can be drawn as below for knapsack problem.



→ Finally node 2 becomes an answer node.

→ Therefore solution is

$x_1 = 1$, $x_2 = 1$, $x_3 = 0$, $x_4 = 1$.

——— x ———