UNIT - IV

Backtracking

"Backtracking is a problem solving strategy used to find all possible solutions of a problem."

→ Backtracking uses Bruteforce technique to find the solutions

→ Backtracking follows Depth first Search technique to find the solutions.

When to use Backtracking ?

When we have multiple solutions for a problem and if we need all possible solutions then we use Backtracking approach.

Application:

Backtracking is used to solve

1) N-Queens Problem
2) Sum of Subsets problem
3) Graph coloring problem . . . .
4) Hamiltonian cycle .
5) Maze solving Problem .
6) The Knight's Tour Problem.

Time Complexity: Generally all backtracking algorithms takes $O(2^n)$ time complexity (or) $O(n^n)$.

Topic-1:- # N-Queens Problem

"N Queens problem is the classical Example of Backtracking. N-Queens problem is defined as, "given N×N chess board, arrange N queens in such that no two queens attack each other by being in same row, column or diagonal".

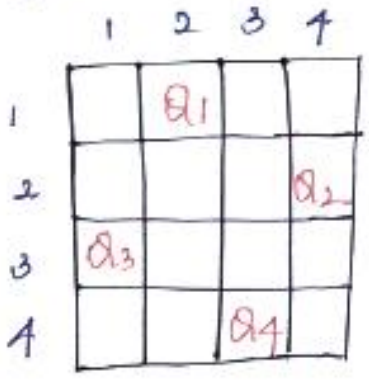
Note:- * For N=1, this is trial case: 1 [Q]

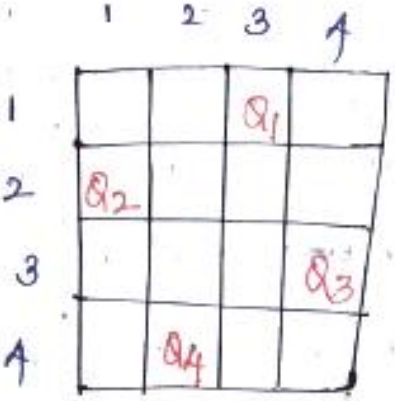
* For N=2 and N=3, solution is not possible, so we start with N=4 and if N≥4 we can obtain solutions.

Problem:- Let's solve 4-Queens Problem using Backtracking

Solution:- Given 4×4 chessboard, arrange four queens in a way, such that no two queens attack each other.

1. For 4 Queens there are two possible solutions they are

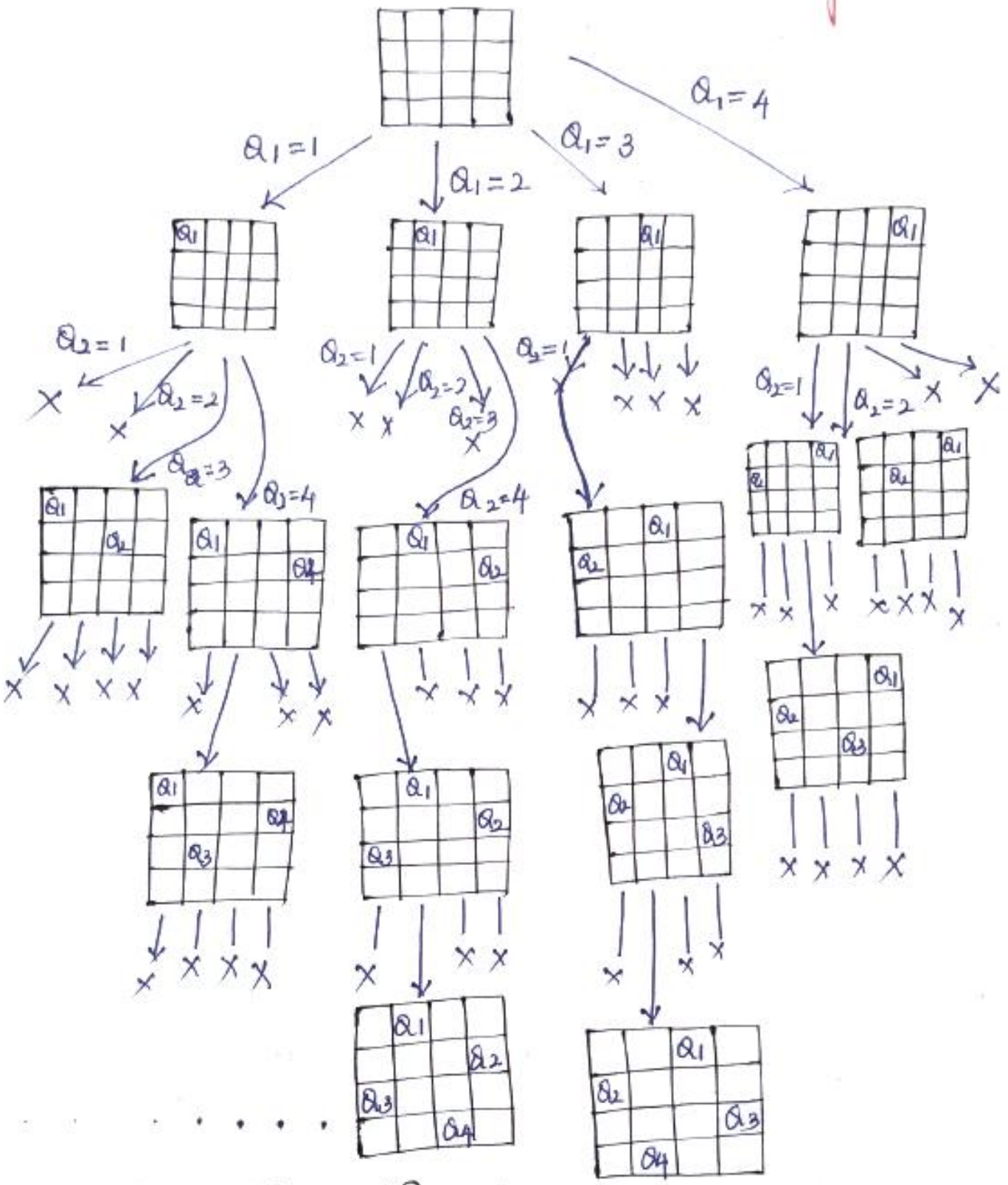




→ solution 1 = {2, 4, 1, 3}

solution 2 = {3, 1, 4, 2}

Lets find the above solutions using state-space tree where we can see the backtracking.



Since All Queens are placed Succesfully, the resultant arrays are

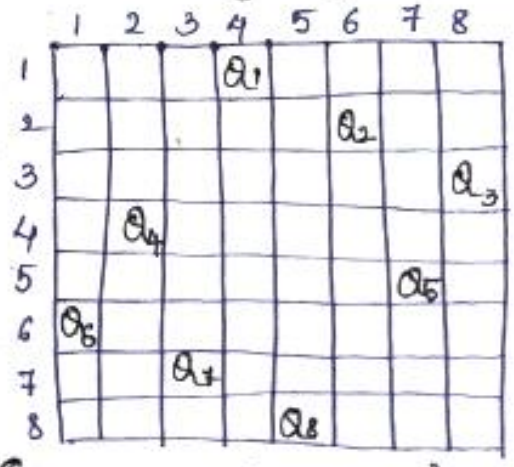
$S1 =$ | 2 | 4 | 1 | 3 |
$Q_1\ Q_2\ Q_3\ Q_4$

$S_2 =$ | 3 | 1 | 4 | 2 |
$Q_1\ Q_2\ Q_3\ Q_4$

Time Complexity:- The time complexity of N-Queens Problem is $O(2^n)$

Algorithm of N-Queens Problem

```
1.  Algorithm NQueens(k, n)
2.  { // This procedure prints all possible places of n
3.    //queens on an N×N chess board, so that they are
4.    // non-attacking.
5.    for i=1 to n do
6.    {
7.      if Place(k,i) then
8.      {
9.        x[k]=i;
10.       if (k==n) then
11.         print(x[1:n])
12.       else
13.         NQueens(k+1, n)
14.     }
15.   }
16. }
17. Algorithm Place(k, i)
18. {
19.   for j=1 to k-1 do
20.     if ((x[j]=i) or (abs[x[j]-i]= abs(j-k)))
21.       return False
22.     return True
23. }
```



$x1 = \{4, 6, 8, 2, 7, 1, 3, 5\}$

Topic-2 : Sum of Subsets Problem

Sum of subsets problem is to find subset of elements that are selected from a given set whose sum adds up to a given number K.

→ The set contains non-negative values.

→ It is assumed that the input set is unique (no duplicates are presented)

→ Backtracking can be used to make a systematic consideration of the elements to be selected.

Time Complexity: $O(2^n)$

Example Problems with direct solutions are given below.

1. set = {10, 5, 15, 12, 8} Sum = 30

   subset 1 = {10, 5, 15} = 30

   subset 2 = {10, 12, 8} = 30.

2. set = {2, 4, 6, 8, 10} Sum = 10

   subset 1 = {2, 8}

   subset 2 = {4, 6}

   subset 3 = {10}

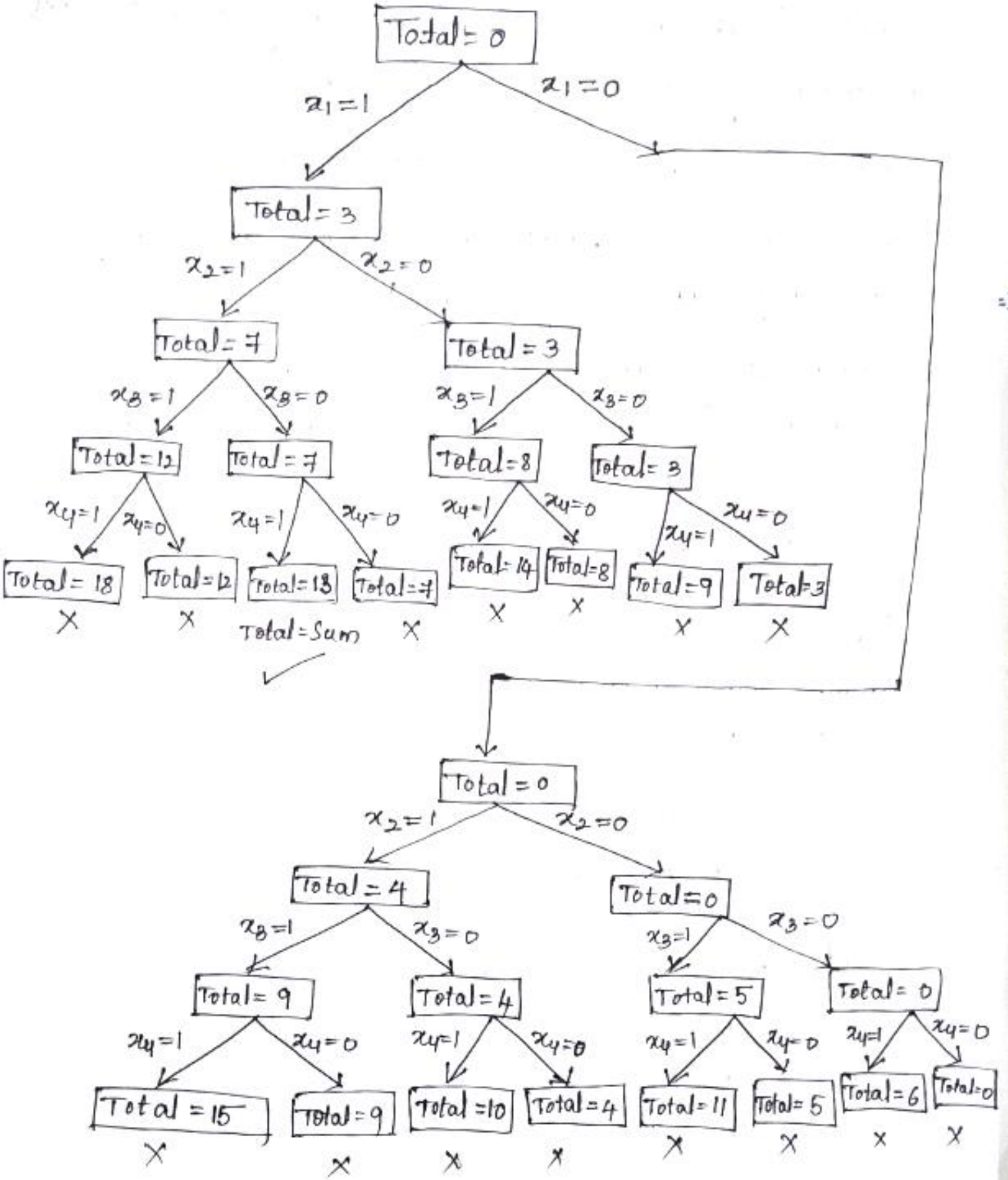
3. S = {2, 4, 6, 8, 10} sum = 15

   No subset gives the sum = 15

Example Problem on Sum of Subsets using Backtracking

$x_1\ x_2\ x_3\ x_4$

Set = {3, 4, 5, 6} Sum = 13.



→ The only solution we got is {1, 1, 0, 1} ⇒ {3, 4, 6}

Algorithm for Sum of Subsets using Backtracking

```
1. Algorithm SubsetSum(set, subset, n, subsize, total, node, sum)
2. {
3.  //Input: The given set and subset, size of set and subset
4.  //a total of the subset, no of elements in the subset
5.  //and the given sum.
6.  //Output: All possible subsets whose sum is the same
7.  //          as the given sum.
8.  {
9.   Begin
10.    if total = sum, then
11.        display the subset.
12.    //go for finding next subset
13.    SubsetSum(set, subset, subsize-1, total-set[node], node+1,
                 sum)
14.    return
15.  else
16.    for all element i in set, do
17.        subset[subsize]:=set[i]
18.        subsetsum(set, subset, n, subset+1, total+set[i],
                     i+1, sum)
19.    done
20.  End
21. }
22.}
```

Topic-3:- Graph Coloring

Graph coloring refers to the problems of coloring vertices of a graph in such a way that no two adjacent vertices have the same color. This is also called as a vertex coloring problem.

→ If coloring is done using at most k colors, it is called k-coloring

→ The smallest no. of colors required for coloring graph is called its Chromatic Number

→ The chromatic number is denoted by $\chi(G)$

→ Finding the chromatic no. of a graph is NP-complete problem.

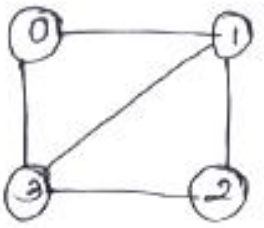
→ Graph coloring problem is both decision problem as well as optimization problem.

→ Decision problem is stated as, " with given M colors and graph G. whether such color scheme is possible (or) not ?

→ The optimization problem is stated as, " Given M colors and graph G, find the minimum no. of colors required for graph coloring.

Example Problem:- Solve the given graph coloring problems

$n = 4$, $m = 3$.

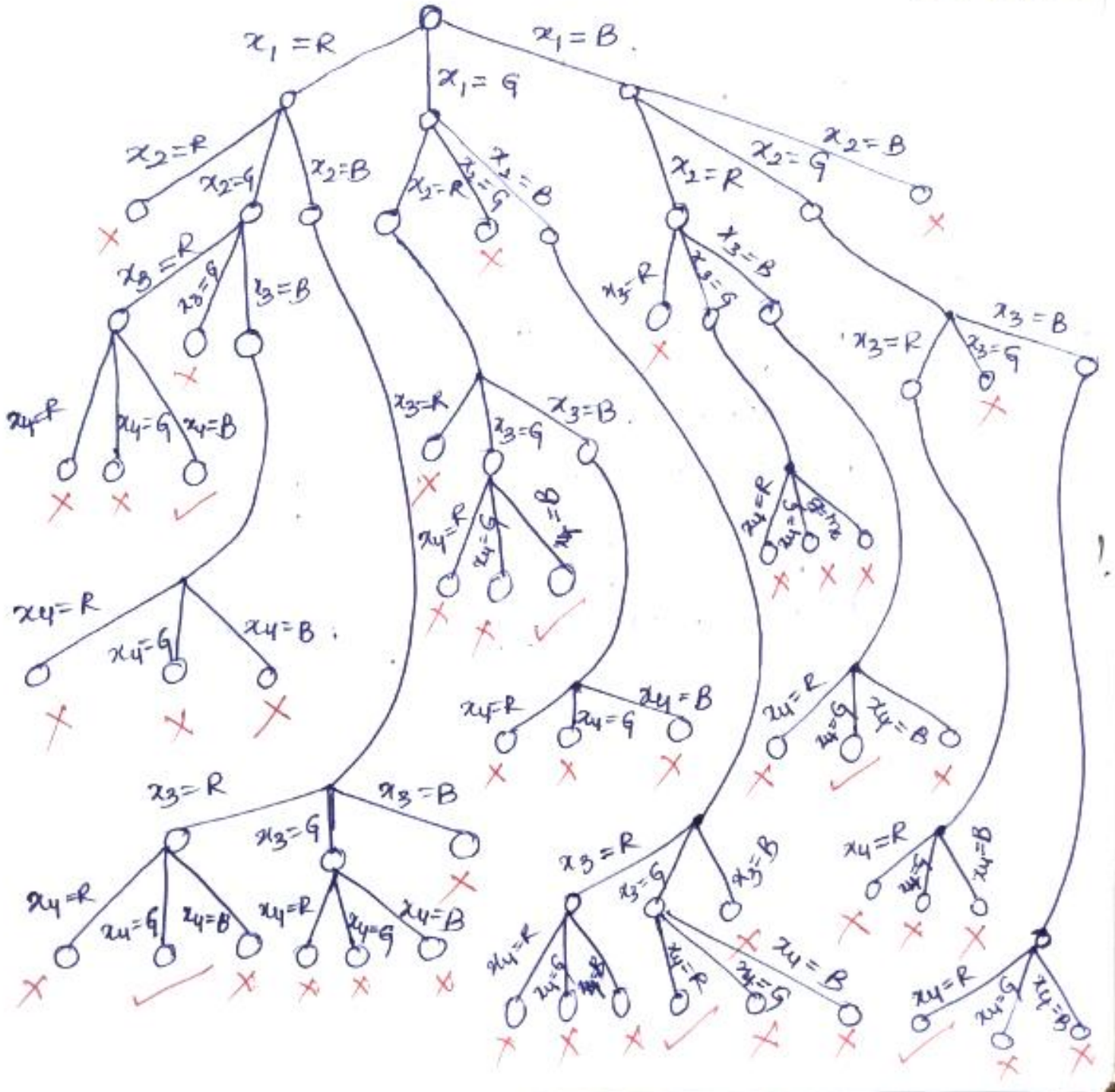


Here $n = 4$; i.e $\{0, 1, 2, 3\}$

$x = \{x_1, x_2, x_3, x_4\}$

$m = 3$, Lets say $m = \{R, G, B\}$

Solution:- From the above problem $m = 3$ means consider any 3 colors, lets say $m = \{R, G, B\}$ where R ← Red
G ← Green
B ← Blue



The possible solutions are.

$S_1 = \{R, G, R, B\}$

$S_2 = \{R, B, R, G\}$

$S_3 = \{G, R, G, B\}$

$S_4 = \{G, B, G, R\}$

$S_5 = \{B, R, B, G\}$

$S_6 = \{B, G, B, R\}$

Time Complexity:- $O(m^v)$ where $m \leftarrow$ colors
$v \leftarrow$ vertices.

Applications:-

1) Design a timetable
2) Sudoku
3) Register allocation in the compiler
4) Map coloring
5) Mobile Radio frequency assignments.

Algorithm of Graph Coloring

```
1. Algorithm mcoloring(k)
2. {
3.  Repeat
4.  {
5.    //Generate all legal assignment for x[k]
6.    nextvalue(k);
7.    if (x[k]=0) then return ; //No color possible
8.    if (k=n) then  //at most m color have been used.
9.        Write (x[1:n]);
10.   else
10.       mcoloring(k+1);
12.  } until(false);
13. }
14. Algorithm nextvalue(k);
15. {
16.  Repeat
17.  {
18.   x[k]=(x[k]+1) mod(m+1); //next highest color
19.   if (x[k]=0) then return ; //all colors have been used.
20.   for j=1 to n do
21.   {
22.     if ((G[k,j] ≠0) and (x[k]=x[j]))
23.      //adjacent vertex have the same color
24.      //then break.
25.   }
26.   if (j=n+1) then return;  //new color found
27.  } until(false);
28. }
```

Topic-4:- 0/1 Knapsack Problem using Backtracking

Knapsack Problem using Backtracking can be solved as follow.

1. The knapsack problem is useful in solving resource allocation problems.
2. Let $X = \langle x_1, x_2, x_3 \ldots x_n \rangle$ be the set of n items,
   $W = \langle w_1, w_2, \ldots w_n \rangle$ be the set of weight and value associated with each item in x, respectively.
3. Let M be the total capacity of the knapsack, i.e knapsack cannot hold items having a collective weight greater than M.
4. Select items from X such that it maximizes the profit and the collective weight of selected items does not exceed the knapsack capacity
5. The knapsack problem has two variants.
   0/1 knapsack does not allow breaking up the item, where as fractional knapsack does. 0/1 knapsack is also known as a Binary knapsack.
6. The knapsack problem can be formulated as.

Maximum → $\sum_{i=1}^{n} v_i x_i$

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

x_i in (0,1)

Algorithm of 0/1 Knapsack Problem using Backtracking

```
Algorithm  BK_knapsack(M, w, V, fw, fp, X)
{
  M: Knapsack capacity
  w(1..n): Set of weight of the items.
  V(1...n): Set of profits associated with items
  Fw: Final Knapsack weight
  Fp: Final earned profit
  X(1..n): Solution vector

  cw ← 0    // current weight
  cp ← 0    // current profit
  fp ← -1   // final profit
  k ← 1     // index of item being processed.

{ loop:
  { while: (k≤n) and (cw+w[k]≤M)
    { do:
        cw ← cw + w[k]
        cp ← cp + v[k]
        Y[k] ← 1
        k ← k+1

    if (k>n)
    {
        fp ← cp
        fw ← cw
        k ← n
        X ← Y
    }
```

```
    else
      {
        y[k] ← 0
      }
    end
    } while (Bound(cp, cw, k) ≤ fp
  }
  while (k ≠ 0 and y[k] ≠ 1)
    {
      k ← k-1
     if (k == 0)
        return
    }
    k = k+1
  }
}
```

Time Complexity:- $O(nW)$

where, $n \leftarrow$ no. of items

$W \leftarrow$ Capacity of knapsack

using Backtracking $\leftarrow$ $O(2^n)$

Example Problem:- items ← {1, 2, 3}

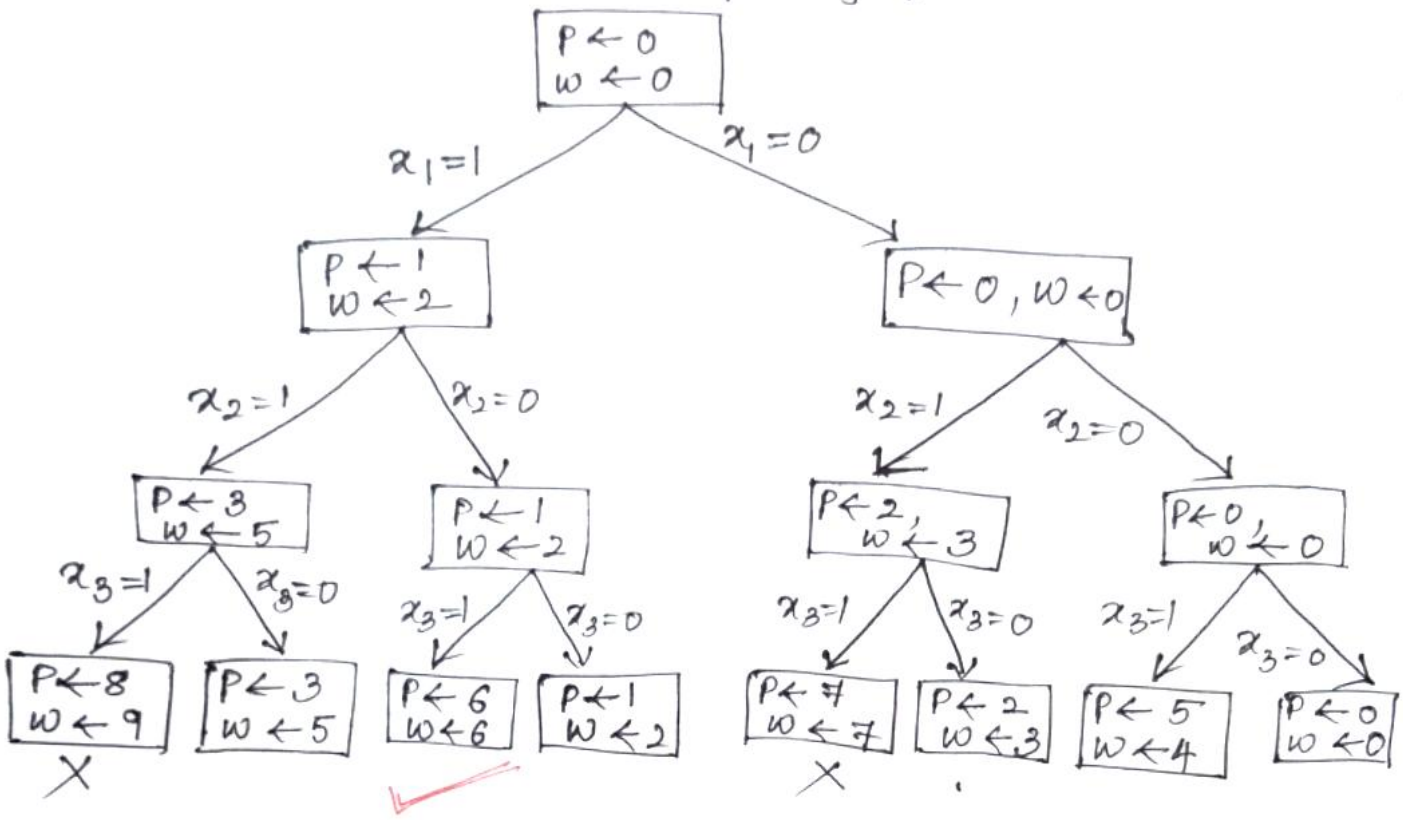
weight ← {2, 3, 4} W = 6

Profit ← {1, 2, 5}

x_1 x_2 x_3

P ← Profit

W ← capacity of knapsack.



→ There are multiple possible solutions, among those the solution with maximum profit is selected.

i.e. $x = \{1, 0, 1\}$ (x_1 x_2 x_3)

When n = 3, the possibilities are 8.

∴ Time complexity = $2^3 = 2^n$.

∴ $O(2^n)$

UNIT-IV

Branch and Bound.

"The Branch and Bound is a method used in combinational optimization problems to systematically search for the best solution. It works by dividing the problem into smaller subproblems (or) branches, and then eliminating certain branches based on bounds on the optimal solution".

Different search techniques in branch and bound.

1. LC Search
2. BFS
3. DFS.

1) LC Search (Least Cost Search)

→ It uses a heuristic cost function to compute the bound values at each node.
→ Nodes are added to the list of live nodes as soon as they get generated.
→ Node with the least cost function is selected as a next E-node.

2) BFS (Breadth First Search)

→ It is also known as a FIFO search
→ Maintains the list of live nodes in FIFO order.
→ FIFO node is expanded for next E nodes.

3) DFS (Depth First Search)

→ It is also known as a LIFO search.

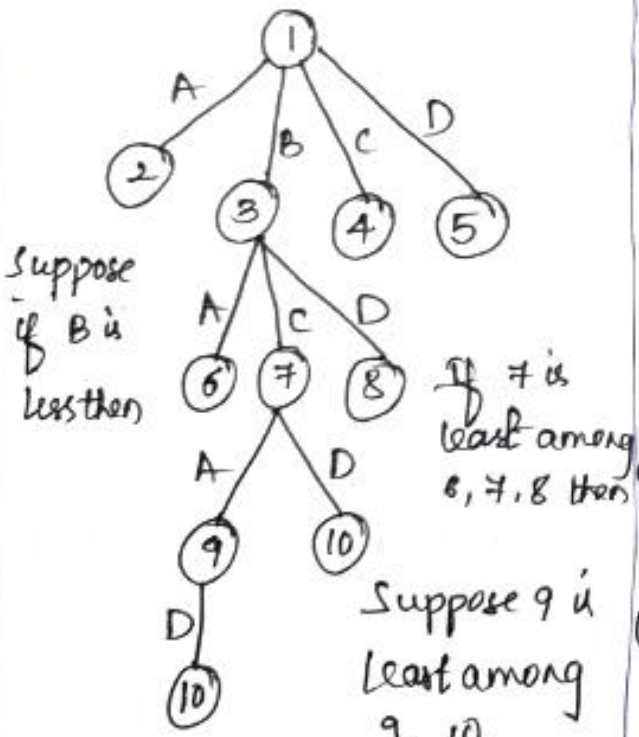
→ It maintains the list of live nodes in LIFO order

→ The live nodes are searched in the LIFO order to make next E-nodes.

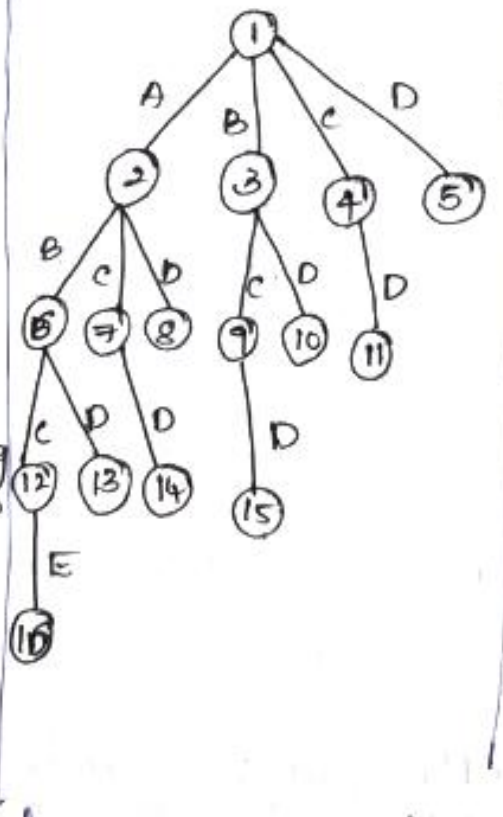
* Branch and Bound strategy is generally used to solve minimization problems.

Example;- Set ← {A, B, C, D}

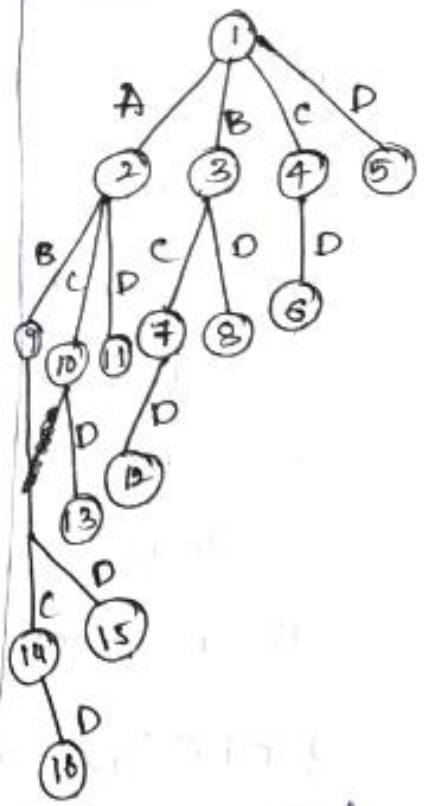
Applying LC Branch and Bound



Applying FIFO Branch & Bound



Applying LIFO Branch & Bound.



→ Branch and Bound algorithms takes exponential time complexity

→ Among LC BB, BFSBB and DFSBB, LCBB is best.

Topic-1:- 0/1 Knapsack Using Branch and Bound.

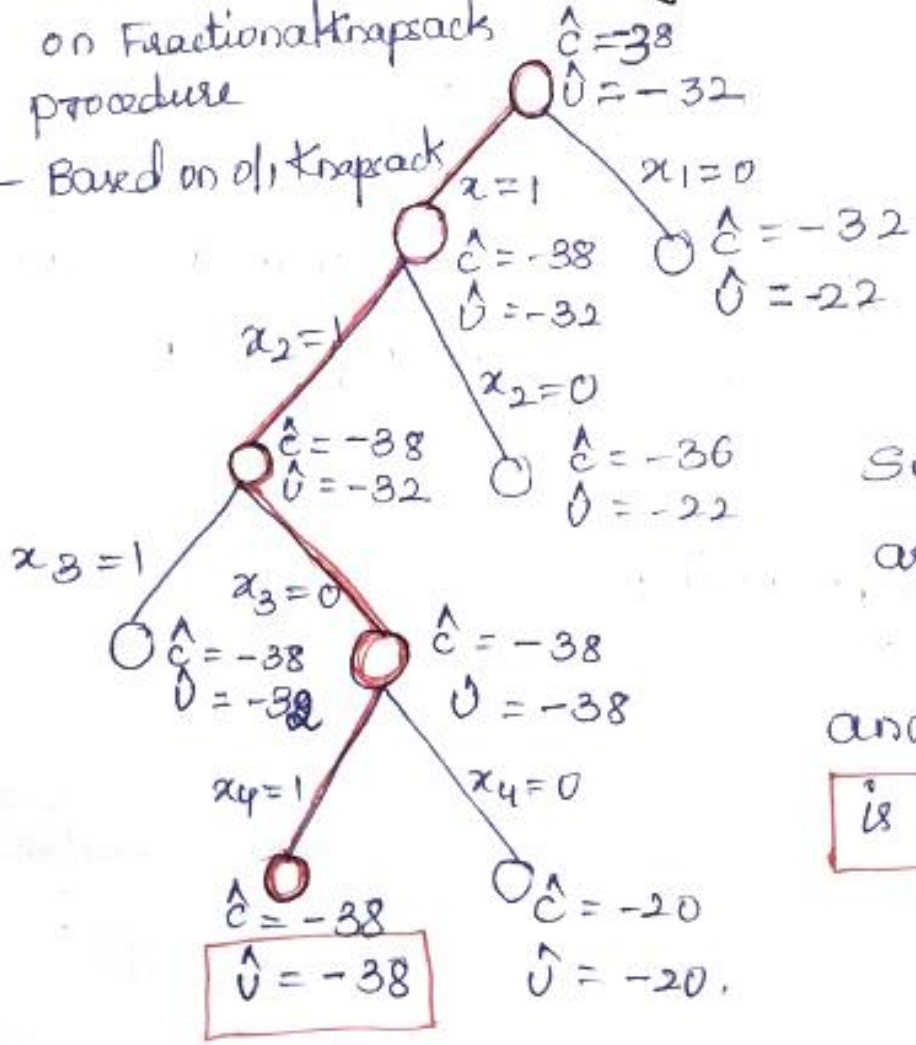
"Given N items with weights w[0...n-1], values v[0..n-1] and a knapsack with capacity C, select the items such that:

1. The sum of weights taken into the knapsack is less than or equal to C.
2. The sum of values of the items in the knapsack is maximum among all the possible combinations

Example N = 4 C = 15 values ← {10, 10, 12, 18}
Weights ← {2, 4, 6, 9}

$\hat{c}$ ← calculated based on Fractional knapsack procedure

$\hat{v}$ ← Based on 0/1 knapsack



Selected objects are x = {1, 1, 0, 1}
x1 x2 x3 x4

and the Profit is 38

```
Algorithm BB0/1Knapsack(int W, item arr[], int n)
{
    queue < Node > Q
    Node u, v
    Q.push(u)
    while (!Q.empty())
    {
        u = Q.front() & Q.pop()
        v.level = u.level + 1
        v.weight = u.weight + arr[v.level].weight
        v.profit = u.profit + arr[v.level].value
        if (v.weight <= W && v.profit > maxProfit)
            maxProfit = v.profit
        v.bound = bound(v, n, W, arr)
        if (v.bound > maxProfit)
            Q.push(v)
        v.weight = u.weight
        v.profit = u.profit
        v.bound = bound(v, n, W, arr)
        if (v.bound > maxProfit)
            Q.push(v)
    }
    return (maxProfit)
}
```

Topic-2 :- Travelling Salesman Problem using Branch & Bound.

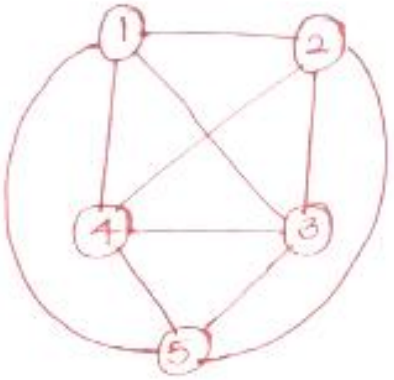
Given a set of cities and distances between every pair of cities, the problem is to find the shortest possible tour that visits every city exactly once and returns to the starting point.

Procedure to solve the problem (Algorithm)

1. Construct the cost matrix if only graph is provided.
2. Reduce the rows and columns of the matrix such that each row and column of cost matrix must have atleast one zero.
3. Now consider vertex 1 as a source vertex and assign the cost of reduction Matrix of step 2 to vertex 1
4. Now consider remaining vertices, and find the distances from 1 to each remaining vertex using following three steps. [To find C[i,j]]
   i) Make i^{th} row and j^{th} column as ∞
   ii) Make C[j,1] as ∞
   iii) find cost $c[i,j] = c[i,j] + \text{cost(parent)} + \text{Cost of Reduction Matrix}(i,j)$
5. Now find the least cost vertex, and consider that vertex as a root node and repeat step 4.
6. Repeat step 4 and 5 until all the nodes are visited

Time Complexity:- $O(n!)$

Example Problem: Travelling Salesperson Problem using Branch and Bound.



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \end{array}$$

Solution:-

Step 1:- Reduce the given cost matrix.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix} \end{array}\begin{array}{l} 10 \\ 2 \\ 2 \\ 3 \\ 4 \end{array}$$

Row Reduction →

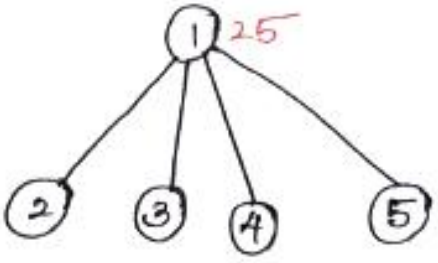
$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 15 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{bmatrix} \\ 1 \quad\quad 3 \end{array}$$

Column Reduction →

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}\begin{array}{l} 10 \\ 2 \\ 2 \\ 3 \\ 4 \end{array} \\ 1 \quad\quad 3 \quad\quad\quad 25 \end{array}$$

The cost of Reduction matrix is 25

Now choose vertex 1 as root node. and assign the cost to it and descendant nodes are remaining vertices.



→ Next step is to find the distances from 1 to 2, 3, 4 and 5.

→ To find the distances, consider the above reduction matrix as a base matrix.

Steps to find distance

1. Make i^{th} row and j^{th} columns as ∞
2. Make $c(j,1)$ as ∞
3. find $c(i,j) = c(i,j') + \text{parentcost}(j') + \text{Reduction cost}(i,j)$.

→ Now Lets find.

1) $c(1,2)$ →

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

[0]

$\therefore c(1,2) = c(1,2) + \text{parentcost}(2) + \text{ReductionCost}(1,2)$

$= 10 + 25 + 0 = 35$

2) $c(1,3)$ →

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

11

Colm Reduction →

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

4

11

$c(1,3) = c(1,3) + \text{parentcost}(3) + \text{ReductionCost}(1,3)$

$= 17 + 25 + 11 = 53$

3) $c(1,4)$ →

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

[0]

$c(1,4) = c(1,4) + \text{parentcost}(4) + \text{ReductionCost}(1,4)$

$= 0 + 25 + 0 = 25$

4) c(1,5)

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

RR →

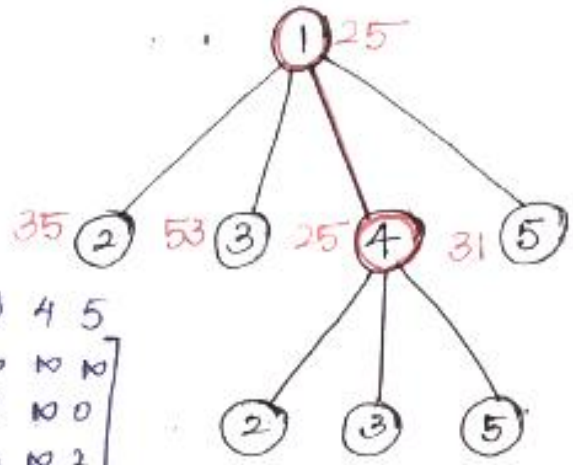
	1	2	3	4	5	
1	∞	∞	∞	∞	∞	
2	10	∞	9	0	∞	2
3	0	3	∞	0	∞	
4	12	0	9	∞	∞	3
5	∞	0	0	12	∞	

5

$c(1,5) = c(1,5) + parentCost(5) + ReductionCost(5,5)$

$= 1 + 25 + 5 = 31.$

∴



Minimum node is selected and visit remaining vertices from 4.

c(1,4)

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

→ Now find the distances from 4 to 2, 3, & 5 by having c(1,4) as a base matrix.

1) c(4,2)

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

0

$c(4,2) = c(4,2) + parentcost(2) + ReductionCost(4,2)$

$= 3 + 25 + 0 = 28.$

2) c(4,3) →

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

0

$$c(4,3) = c(4,3) + \text{parentcost}(3) + \text{ReductionCost}(4,3)$$

$$= 12 + 25 + 0 = 37$$

3) c(4,5) →

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

11

RR →

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

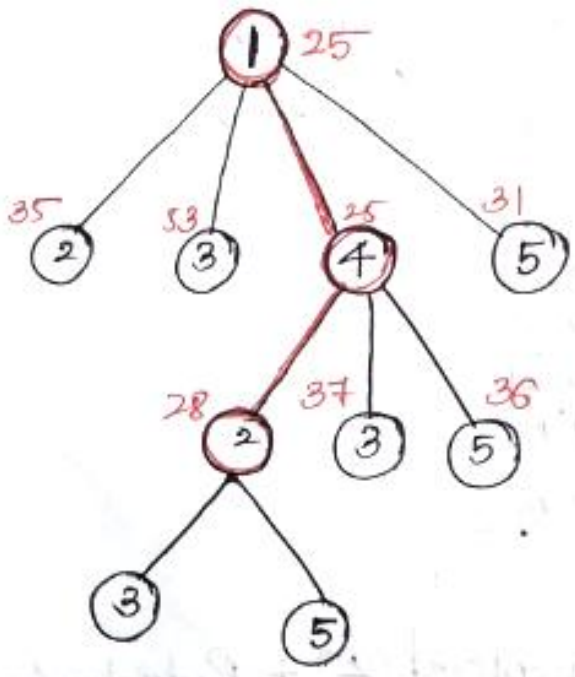
11

11

$$c(4,5) = c(4,5) + \text{parentcost}(5) + \text{ReductionCost}(4,5)$$

$$= 0 + 25 + 11 = 36$$

i.e.



c(4,2) →

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

→ Now, consider Node 2 as a root and consider C(4,2) as a Base Matrix

Now find

$C(2,3)$ →

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

RR →

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

[13]

$c(2,3) = c(2,3) + \text{parentcost}(3) + \text{ReductionCost}(2,3)$

$= 11 + 28 + 13$

$= 52$

$C(2,5)$ →

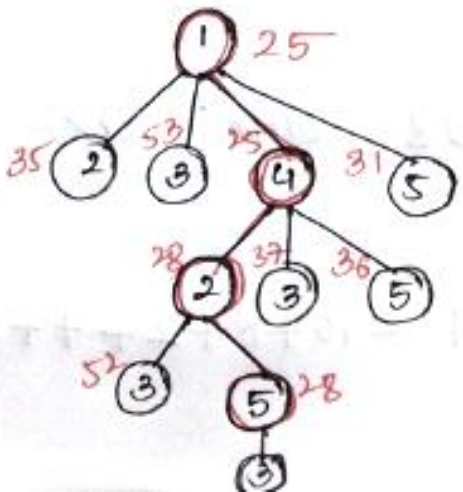
	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	0	∞	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	∞	0	∞	∞

[0]

$c(2,5) = c(2,5) + \text{parentcost}(5) + \text{ReductionCost}(2,5)$

$= 0 + 28 + 0 = 28$

∴



$C(2,5)$ →

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

Now find the distance from 5 to 3.

$\xrightarrow{C(5,3)}$

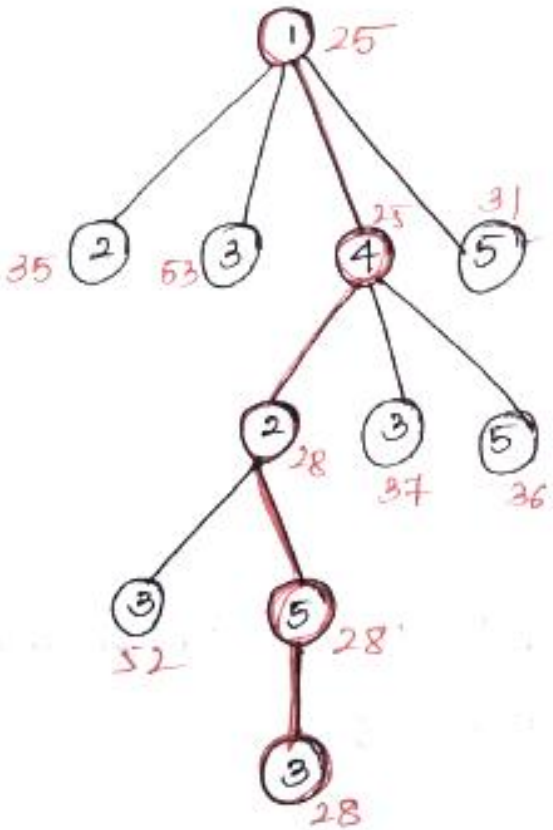
	1	2	3	4	5
1	∞	∞	∞	∞	∞
2	∞	∞	∞	∞	∞
3	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	∞

[0]

$\therefore C(5,3) = c(5,3) + \text{parent cost}(3) + \text{Reduction Cost}(5,3)$

$= 0 + 28 + 0 = 28$

∴ The final state space tree



The Minimum cost is 28 with the following tour.

$1-4-2-5-3-1 = 10+6+2+7+3 = 28.$