UNIT - 4

Backtracking

The general method:

Backtracking is a systematic method for searching one or more solutions for a given problem. It is a refined brute force technique used for solving problems. Backtracking can effectively solve multi-decision problems, where the final solution is visualized as a set of decisions. The execution of a decision or choice leads to another set of decisions or choices.

A backtracking design paradigm can solve the following three types of problems:

1. Enumeration problems : All solutions are listed for a given problem.
2. Decision problems : A solution is given in terms of Yes & no
3. optimization problems: optimal solutions are required, which maximize or minimize the given objective function as per the constraints of the given problem.

-> Backtracking approach can solve most of the problems in polynomial ~~time~~ time.

-> A backtracking algorithm solves problems incrementally by adding the candidate solutions (called partial solutions) till the final solution is obtained.

→ If the partial solution is not leading to a solution, then it is rejected along with the other candidate solutions. this process is called <u>domino principle</u> or <u>pruning</u>.

<u>Basics:-</u>

Backtracking is a depth-first search, with some bounding functions. Bonding functions & Validity functions represent the constraints of the given problem. First, the backtracking process defines a solution vector as n-tuple vector $(x_1, x_2, \ldots, x_n)$ for the given problem. Here "n" is the no. of components of the solution vector and each x_i, where i ranges from 1 to n, represents a partial solution. These partial solution components x_i are generated based on constraints.

Constraints are rules that restrict the generation or processing of a tuple. There are two types of constraints: ① explicit and ② implicit constraints.

explicit: These are rules that restrict a component of the solution vector.

implicit: These are rules that limit the generation or processing of a solution vector.

The backtracking approach involves two stages:

① Generation of a State-space tree
② Exploring the state-space tree

Generation of State-space Trees:

A state-space tree is generated as part of the solution. A state-space tree is an arrangement of all possible solutions in a tree-like fashion. This trees can be a binary tree. Every node of the state-space tree represents a partial solution that illustrates the choices made from the root to the node, and the edges represent transition from states.

Searching state-space trees:

Backtracking uses the DFS approach, and hence, the backtracking can be called a refined DFS algorithm. The backtracking technique thus determines whether a node in the search space is 'promising' or 'non-promising'. A promising node can eventually lead to the final solution. A non-promising node can eventually lead to a situation where solutions cannot be expected. Hence, an alternative route is required to be explored.

```
Algorithm backtracking(u)
{
if (promising(u)) then
    if(u is a goal) then
        print the solution
  else
```

```
for each v, v ∈ child(u) do
  { backtracking(v)
  }
 }
}
}
```

Complexity:

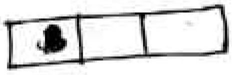
It is difficult to evaluate backtracking algorithm analytically. In Backtracking, a random path can be generated from the root to a leaf of a state-space tree and estimate the choices that are encountered in the path.

The Time Complexity $T(n) = 1 + C_1 + C_1C_2 + \dots + C_1C_2\dots C_n$.

Rough

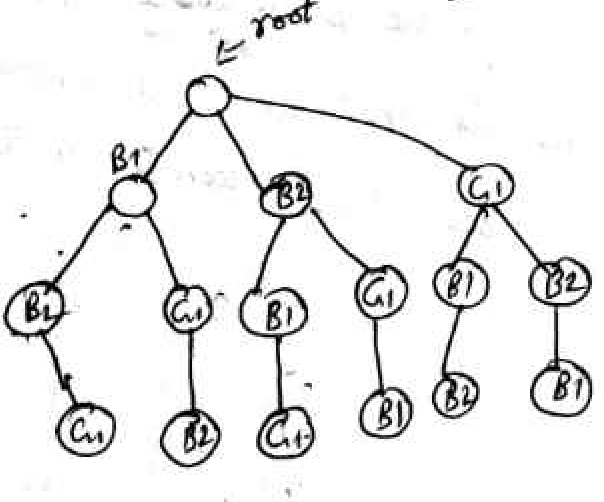
Ex:-

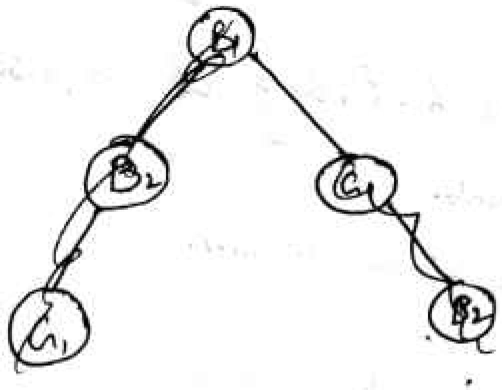
$B_1 B_2 G_1$



$n = 3$ $\quad 3!$ ways

state-space tree





killing node is Bounding function.

3!

N-Queen problem:-

The objective of this N-Queen problem is to place N-Queen on an N×N chessboard such that no two queens are in attacking position. It can be observed no solution exists for one-, two-, & three-queen problems for the boards.

To understand N-Queen problem lets consider and solve 4-Queen problem using backtracking approach.

4-Queen problem is also based on backtracking method. we are given 4-queens to be placed on an 4x4 chess board so that no two queens are on the same row, column or diagonal. Here the solution to the problem is 4-tuple where queens are placed with the constraints.

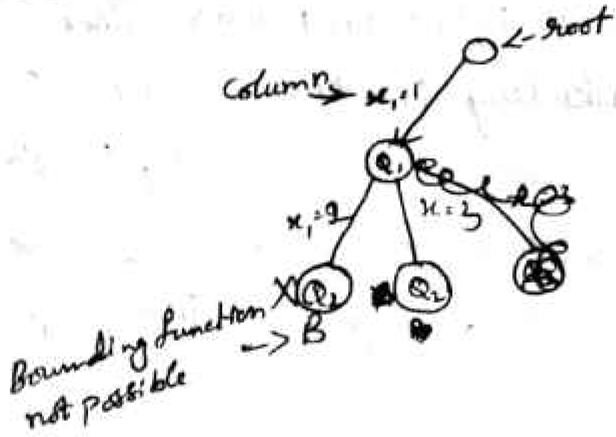
By applying the Brute force approach finding all the possible solutions for 4-Queens with constraints.

step-1: place first queen Q_1 in the first column.

	1	2	3	4
1	Q_1			
2				
3				
4				

step-2: After placing first queen in the first column we cannot place the Q2 in the first & second column. so, we place Q_2 in the third column.

	1	2	3	4
1	Q_1			
2			Q_2	
3				
4				



<u>Step-3</u>: After placing Q_1 & Q_2 there is no place to keep Q_3. Because of attacking position. So change the position of Q_2 & place it in 4^{th} column.

	1	2	3	4
1	Q_1			
2				Q_2
3				
4				

<u>Step-4</u> Place the Q_3 in column 2. Since there is no chance of attacking.

	1	2	3	4
1	Q_1			
2				Q_2
3		Q_3		
4				

Step-5 Q_4 cannot be placed in 1, 2, 3 (or) 4 rows. Since all the colum cells are in attacking position. So goback & change the position of Q_3. Since there is no possibility to place Q_3 in any column, so go back & change the position of Q_2. There is no way to place Q_2, so goback & change the position of Q_1.

Now place Q_1 in 2nd column

	1	2	3	4
1		Q_1		
2				
3				
4				

Step-6 Now place Q_2 in 4th column, since the remaining columns are in attacking position

	1	2	3	4
1		Q_1		
2				Q_2
3				
4				

Step-7 Now place the Q_3 in 1st column

	1	2	3	4
1		Q_1		
2				Q_2
3	Q_3			
4				

Step-8 place Q_4 in 3rd column. Since, Column 1 & 2 are in attacking position.

	1	2	3	4
1		Q_1		
2				Q_2
3	Q_3			
4			Q_4	

Step-9 Since all the Queens are placed & there is no attacking position for any Queen, the optimal solution form is $x_i = 2, 4, 1, 3$.

Note :- The other possible way to solve 4 Queen problem is

3, 1, 4, 2 [mirror image]

* Sum of Subsets :-

In this problem there is a given set with some integer elements. And another some value is also provided, we have to find a subset of the given set whose sum is the same as the given value.

Here we use backtracking approach for selecting a valid subset when an item is not valid, we will backtrack to get the previous subset & add another element to get the solution.

→ Algorithm:

Input :- The given set & subset, size of set & subset, a total of the subset, no. of elements in subsets & the given sum.

output :- All possible subsets whose sum as the given sum.

```
Begin
   if total = sum then
       display the subset
       // go for finding next subset
       subsetSum(set, subset, subsize-1, total-set[node],
                        node+1, sum)
       return
   else
       for all element i in the set, do
           subSet [subSize] = set[i]
           subsetsum(set, subset, n, subsize+1, total+set[i],
       done                              i+1, sum)
END
```

→ Example:

Given set = ~~[illegible]~~ [3, 4, 5, 2]

sum of subset = 9

Include :- It means that we are selecting the elements from the array.

Exclude :- It means that we are rejecting the elements.

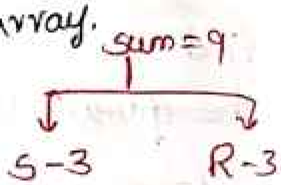
Step 1 :- Taking first element.

There are two scenarios:

1. Selecting 3.

   target sum − value of element = 9 − 3 = 6

   i.e. 3 gets stored in the result array.
   [3]



2. Rejecting 3.

   target sum = 9.

   i.e. 3 not stored in the result array

Step 2 :- Taking second element

There are two scenarios.

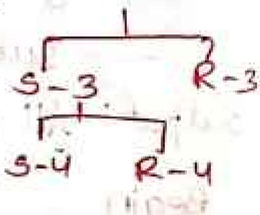
1. Selecting 4

   ∴ target sum − value of element = 6 − 4 = 2

   i.e. 4 gets stored in the result array
   [3, 4]

2. Rejecting 4.

   target sum = 6 i.e. 4 not stored.



Step 3 :- Taking Third element.

1. Selecting 5

   target sum − value of element = 2 − 5 = −3 False

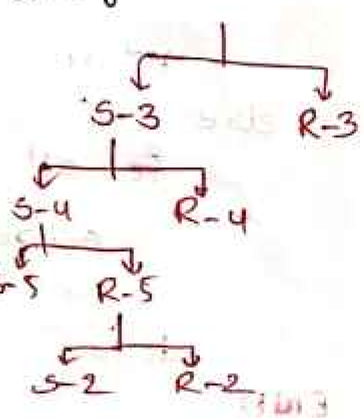
   ∴ i.e. 5 can't be stored in the array.

Step 4 :- Taking Fourth element

1. Selecting 2.

   2 − 2 = 0.

   ∴ i.e. 2 gets stored in result array



step-5 :- Backtrack to R-4.

1. select 5.

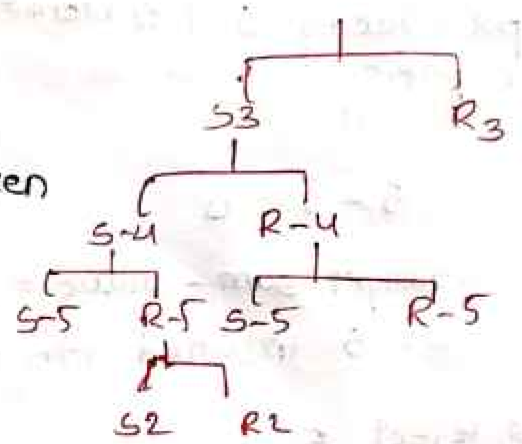
$9 - 3 = 6 - 5 = 1$. taken

∴ The subset [3,5]

2. Reject 5

$9 - 3 = 6$

The subset [3].



step-6 :- Backtrack to S-5

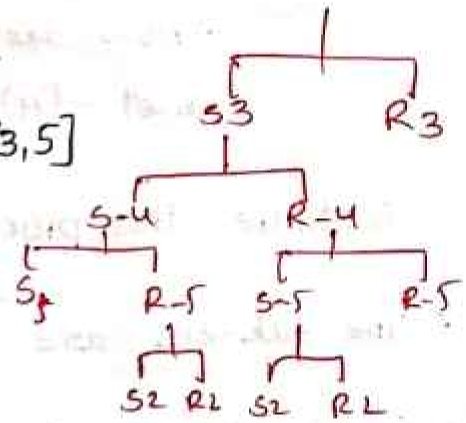
1. Select 2.

$1 - 2 = -1$ -ve value [3,5]

Not inserted.

2. ~~Bd~~ Reject 2.

∴ 2 is rejected subset [3,5]



step 7 :- Back track to R-5

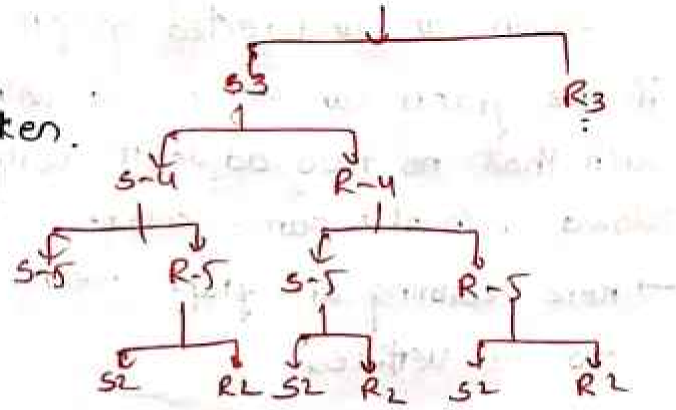
1. select 2

$9 - 3 - 2 = 4$ → taken.

The subset [3,2]

2. Reject 2.

subset [3]



Step 8 :- Back track to R-3.

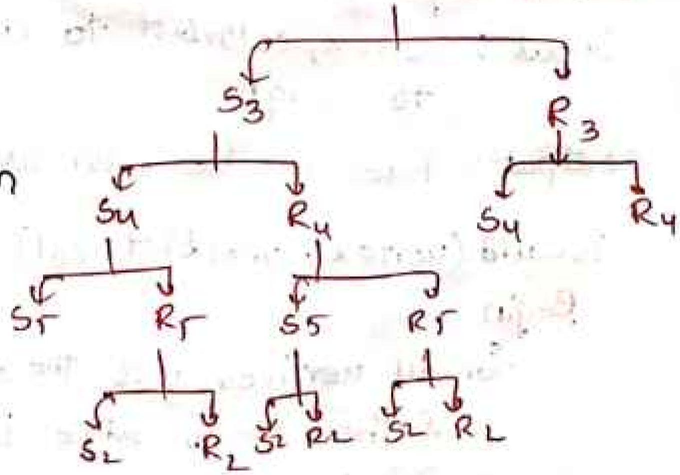
1. select 4.

$9 - 4 = 5$ → taken

∴ Subset [4]

2. Reject 4

target sum = 9.



Step 8:- Taking S-4 & element 5.

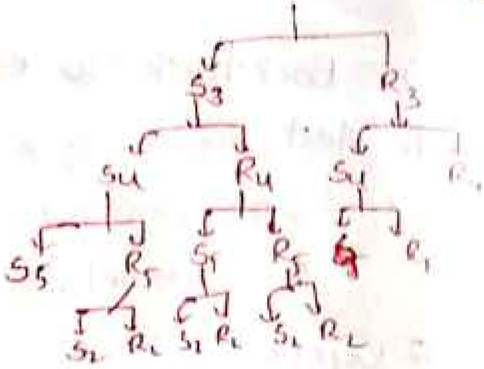
1. select

∴ Insert 5:

$$5 - 5 = 0$$

∴ target sum - value = 0

∴ 5 inserted into subset



2. reject 5.

Total target sum = 9

∴ Now target sum = 5

subset - {4}

continue this process until back-tracking all the nodes

∴ The subsets are {3, 4, 2} & {4, 5}

* Graph Colouring:

Given an undirected graph & a number n, determine if the graph can be colored with at most m colours such that no two adjacent vertices of the graph are colored with the same color.

→ Here coloring of graph means the assignment of colors to all vertices.

→ Formal algorithm.

Input : vertex, colorList to check, and color, which is trying to assign.

output :- True if the color assigning is valid, otherwise false

```
isValid (vertex, colorList, col)
Begin
    for all vertices u of the graph, do
        if there is an edge b/w u & i, & col = colorList[i], then
            return false
    done
    return true
END
```

Input :- most possible colors, the list for which vertices are colored with which color & the starting vertex.

output :- True, when colors are assigned, otherwise false

graphColoring (colors, colorList, vertex)

Begin

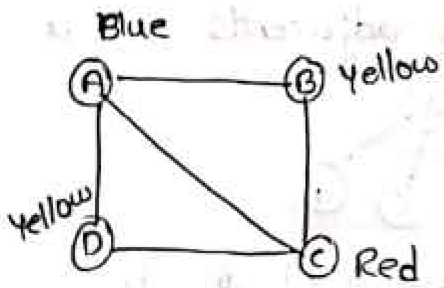
  if all vertices are checked, then
    return true

  for all colors col from available colors, do
    if isValid(vertex, color, col), then
      add col to the colorList for vertex
      if graphColoring (colors, colorList, vertex + 1) = true, then
        return true
      remove color for vertex

  done
  return false

END

Example:-



* Hamiltonian Cycle problem :-

→ Given a graph $G = (V, E)$, we have to find the Hamiltonian cycle using Backtracking approach.

→ we start our search from any arbitary vertex say 'a'

→ This vertex 'a' becomes the root of our implicit tree.

→ The first element of your partial solution is the first intermediate vertex of the Hamiltonian cycle that is to be constructed.

→ The next adjacent vertex is selected by alphabetical order.

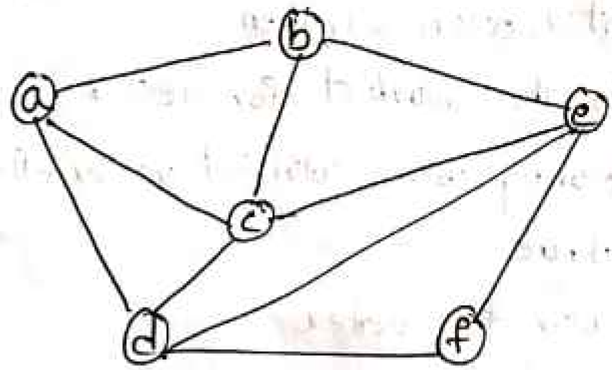
→ In at any ~~pase~~ stage any arbitary vertex makes a cycle with any vertex other than vertex 'a' then we say that dead end is reached.

→ In this case, we backtrack one step and again the search begins by selecting another vertex & back track the element from the partial solution must be removed.

→ The search using backtracking is successful if a Hamiltonian cycle is obtained.

Example:-

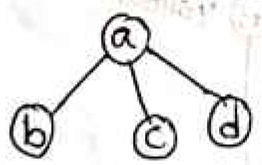
Consider a graph $G = (V, E)$ shown in fig.



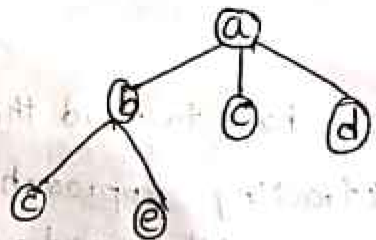
Solution:-

Step 1: First start from vertex 'a'. ⓐ ← root

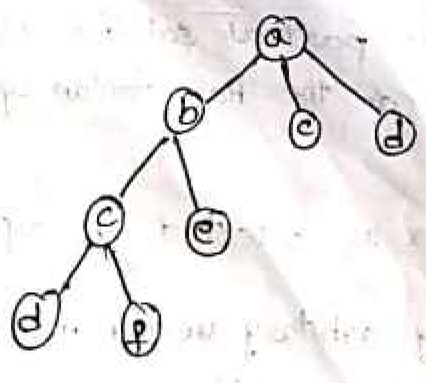
Step 2:- Find all the adjacents of a



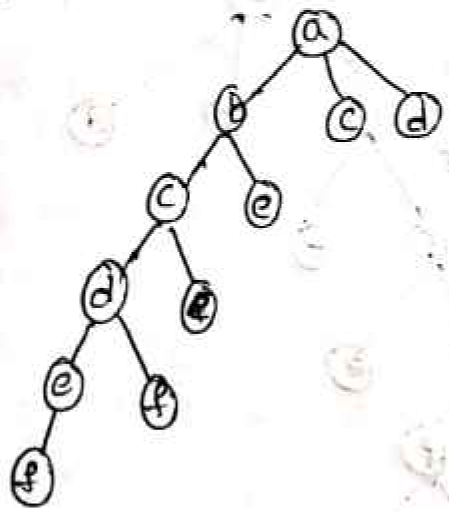
Step 3:- Now select adjacent of b.



Step 4:- Now find the adjacents of c which is adjacent to b

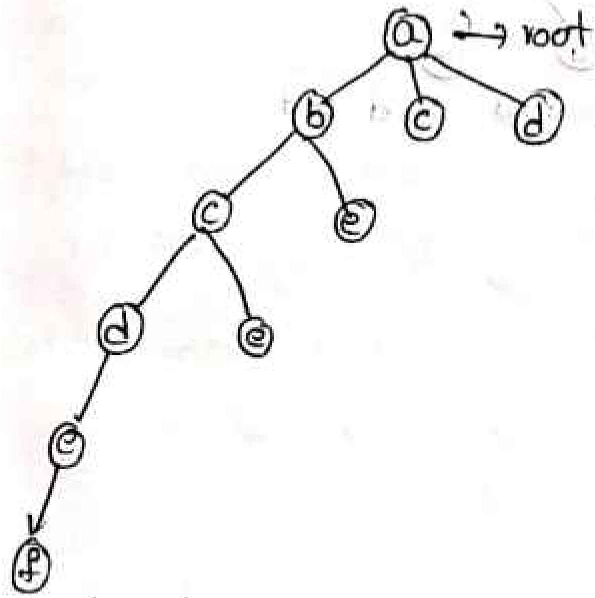


step 5:- Now find adjacents of d.



→ Here we can't find adjacents of f because. e, d are the adjacents of 'f' & they are all already visited.

→ 'f' dead end.



Dead end.

step 6:- From backtracking the vertex adjacent to 'e' is b, c, d and f. from which vertex 'f' has already been checked, and, b, c, d are already visited.

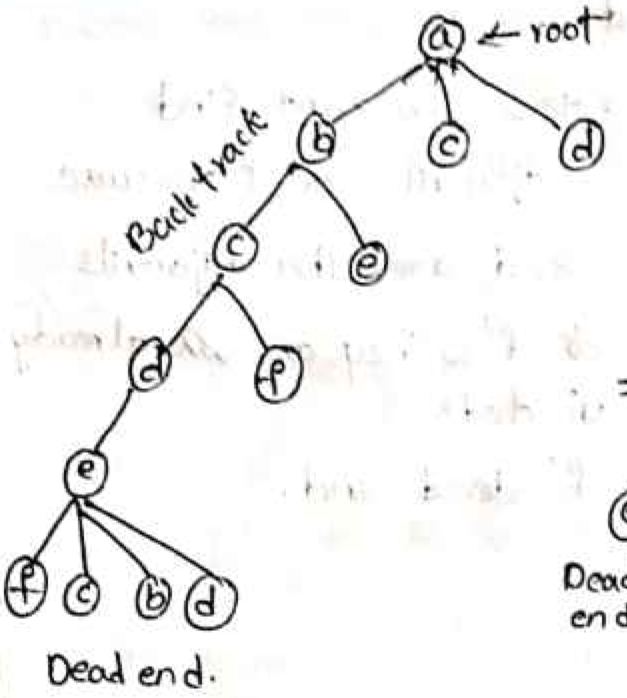
→ So again we back track on step

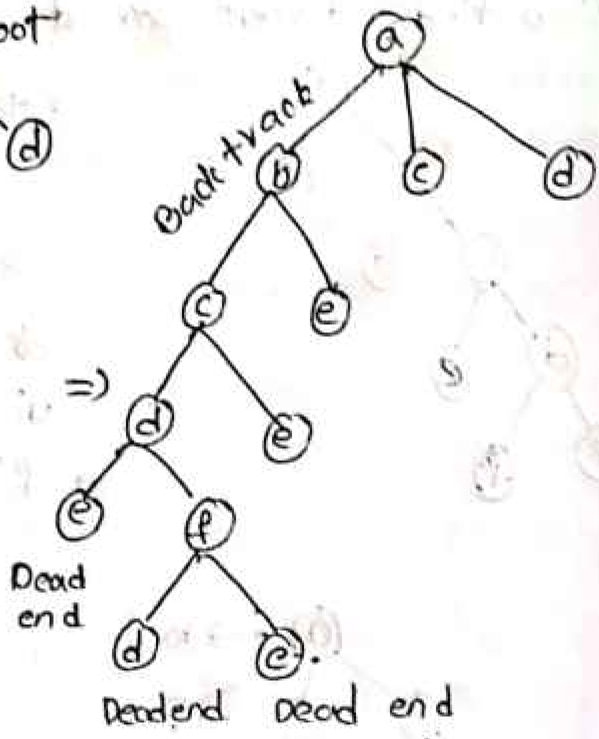
→ Now, the vertex adjacent to 'd' are e, f. from which e has already been checked and adjacent of 'f' are 'd' & e.

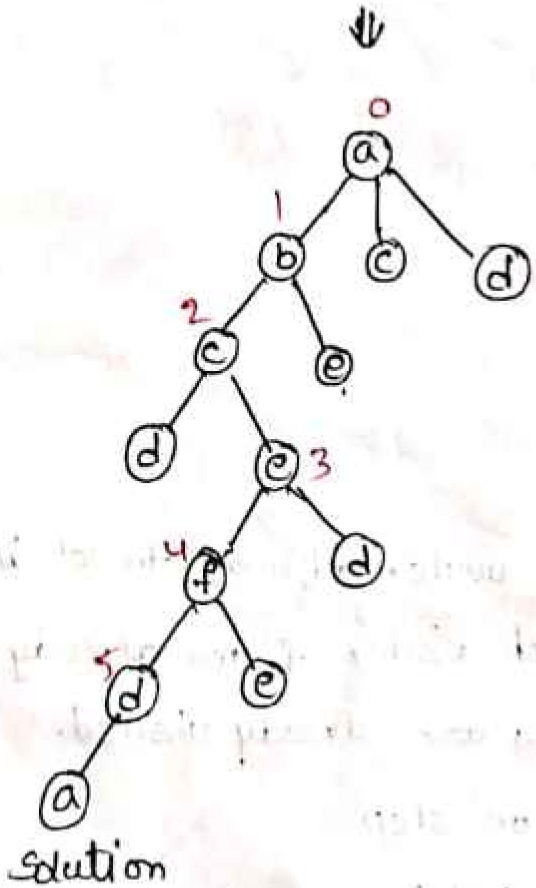
→ If 'e' vertex is revisited then we get a dead state

→ Now adjacent to c is 'e' & adjacent to 'e' is 'f' and adjacent to 'f' is 'd' & adjacent to 'd' is 'a'.

→ Here we get the Hamiltonian cycle as all the vertex other than the start vertex 'a' is visited only once.







. Here we have generated one Hamilton cycle but another Hamiltonian circuit can also be obtained by considering another vertex.

Branch and Bound

→ Branch & bound is one of the techniques used for problem solving.

→ It is similar to the backtracking since it also uses the state space tree.

→ It is used for solving the optimization problems & minimization problems.

→ If we have given a maximization problem then we can convert it using the BB technique by simply converting the problem into a maximization problem.

→ We solve BB problems by using three techniques

1. FIFO   2. LIFO   3. LC

Let's understand through an example.

Jobs = {j1, j2, j3, j4}

P = {10, 5, 8, 3}

d = {1, 2, 1, 2}

Assume than Solution1 = {j1, j4}

another representation

S2 = {1, 0, 0, 1}

First method :-

FIFO :- for solution1.

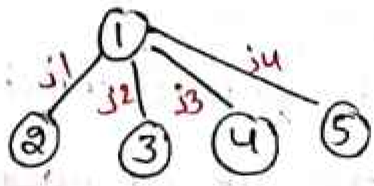
→ Here we move breadth wise for exploring the sum.

→ Here BFS is performed not DFS

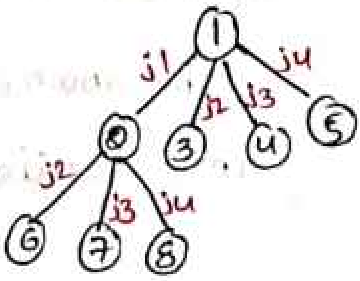
→ ~~od~~cel I

Step 1 :-

Initially



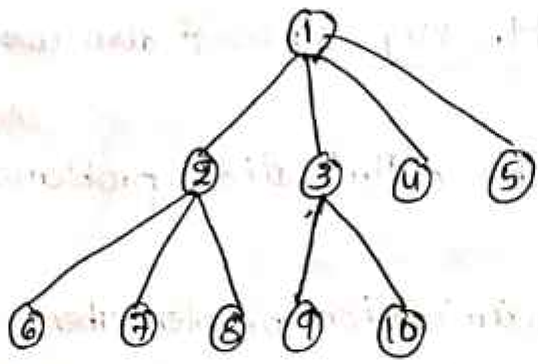
Step 2 :-



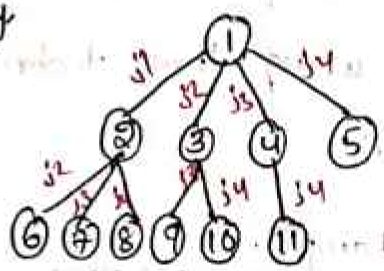
once I take J1 then we consider j2, j3 or j4.

→ If we follow the route then it says that we are doing Job 1 & job 4 & 2 & 3 step 3: Now we will consider the node 3.

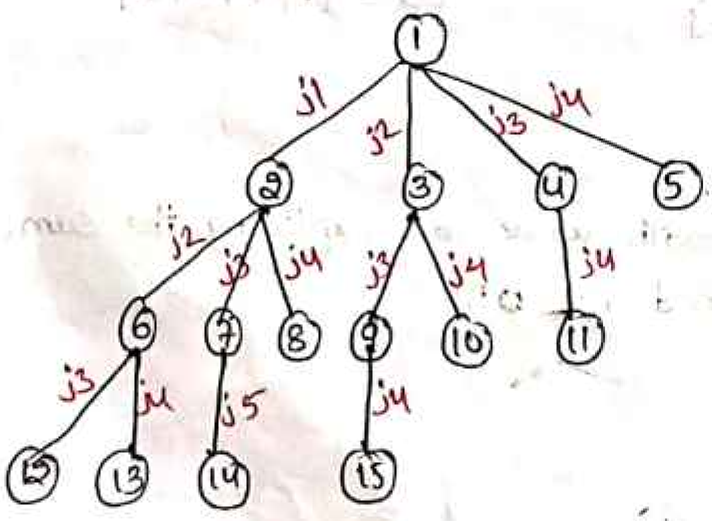
→ In this case we are doing j2 so we can consider either j3 or j4. Here we discard j1



step 4:- Now we will consider the node 3. We consider j4 only



step 5:- Now we will expand node node 6 & node 7 & node 9.

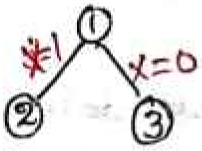


step 6:- The last node (node 12) which is left to be expanded. Here we consider job j4.

∴ The above is the state space tree for the solution S1 = {J1, J4}

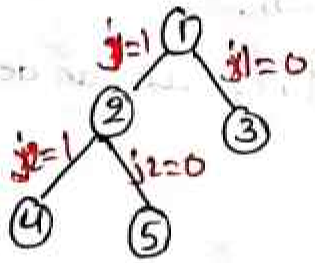
FIFO: for solution 2.   1 - taken

step 1: take job 1   0 - not taken



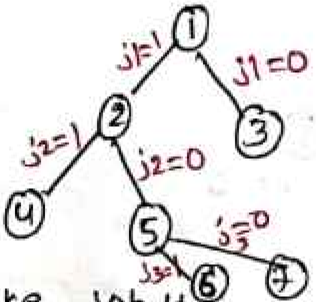
job 1 is taken

step 2: take job 2



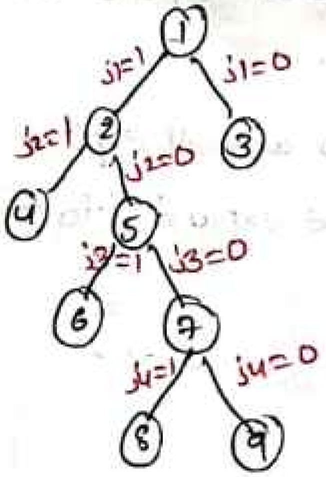
∴ job1 is taken so expand 2.

step3: take job 3



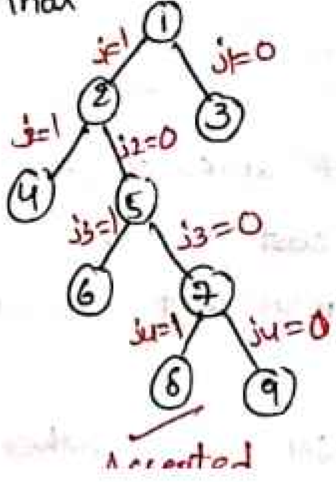
∴ job 2 is not taken so expand 5.

step4: take job 4



∴ job 3 is not taken so expand 7

step5: Final



∴ set S2 = {1, 0, 0, 1}

Second method: Least cost

→ In this technique, nodes are explored based on the cost of the node

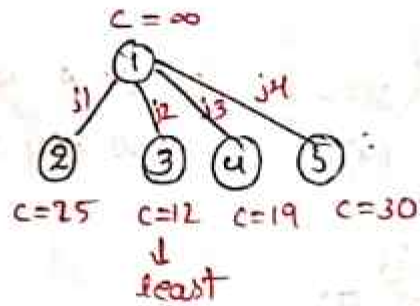
→ The cost of the node can be defined using the problem & with the help of the given pioblem, we can ~~cost~~ ~~definition~~ cost function.

→ once the cost function is defined, we can define cost of the node.

Example:

jobs = {j1, j2, j3, j4}   cost = [25, 12, 19, 30]

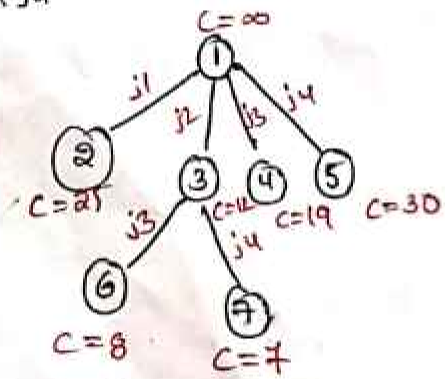
step 1:- Initially



step 2: → Since it is least cost BB so we will expand the node which is having the least cost.

→ node 3 has minimum cost. so we will explore it.

→ It works on j2 so we will be expand into two nodes j3 & j4



→ The node 6 works on j3 & node 7 works on j4.

→ Now we select the minimum cost

→ ~~so~~ The node 7 has the minimum cost so we will explore node 7

→ The node 7 already works on j4. so no further expansion

* Travelling Sales Person:-

Problem statement:-

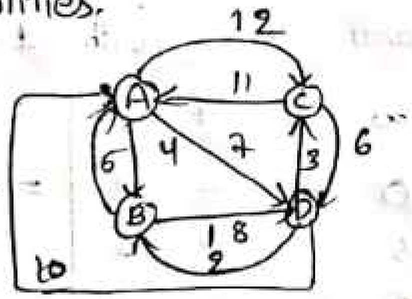
A traveler needs to visit all the cities from a list where distance b/w all the cities are known & each city should be visited just once.

Solution:-

→ It is the most notorious computational problem.

→ we can use brute-force approach to evaluate every possible tour & select the best one.

→ For n no. of vertices in a graph, there are $(n-1)!$ no. of possibilities.



step 1:-

Initially the cost matrix is

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} \infty & 4 & 12 & 7 \\ 5 & \infty & \infty & 18 \\ 11 & \infty & \infty & 6 \\ 10 & 2 & 3 & \infty \end{bmatrix} \end{array} = A_1$$

consider row reduction

→ If the row already contains an entry '0' then there is no need to reduce that row.

→ If the rows doesn't contain an entry '0' then

→ Reduce that particular row

→ select least value from that row

→ subtract that element form each element of that row.

→ This will create an empty '0' in that row, thus reducing that row

→ Subtract 1^{st} row with 4

→ 〃 2^{nd} row with 5

→ 〃 3^{rd} row with 6

→ 〃 4^{th} row with 2.

$$\begin{bmatrix} \infty & 4 & 12 & 7 \\ 5 & \infty & \infty & 18 \\ 11 & \infty & \infty & 6 \\ 10 & 2 & 3 & \infty \end{bmatrix} \begin{matrix} -4 \\ -5 \\ -6 \\ -2 \end{matrix} \Rightarrow \begin{bmatrix} \infty & 0 & 8 & 3 \\ 0 & \infty & \infty & 13 \\ 5 & \infty & \infty & 0 \\ 8 & 0 & 1 & \infty \end{bmatrix}$$

Apply column reduction.

Reduce or subtract column three with 1.

$$\begin{array}{cccc} & & -1 & \\ \end{array}$$
$$\begin{bmatrix} \infty & 0 & 8 & 13 \\ 0 & \infty & \infty & 13 \\ 5 & \infty & \infty & 0 \\ 8 & 0 & 1 & \infty \end{bmatrix} \Rightarrow \begin{bmatrix} \infty & 0 & 7 & 3 \\ 0 & \infty & \infty & 13 \\ 5 & \infty & \infty & 0 \\ 8 & 0 & 0 & \infty \end{bmatrix} = A_2.$$

step 2 :- cost(1) = 4 + 5 + 6 + 2 + 1 = 18.
= sum of reduction elements.

Now consider all other vertices

1. Node 2 (go to vertex-B)

m[A, B] = 0, set row A & column B to ∞

set M[B, A] = ∞.

$$\begin{bmatrix} \infty & 0 & 7 & 3 \\ 0 & \infty & \infty & 13 \\ 5 & \infty & \infty & 0 \\ 8 & 0 & 0 & \infty \end{bmatrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 13 \\ 5 & \infty & \infty & 0 \\ 8 & \infty & 0 & \infty \end{bmatrix}$$

$$\begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 13 \\ 5 & \infty & \infty & 0 \\ 8 & \infty & 0 & \infty \end{bmatrix} \begin{matrix} \\ -13 \\ \\ \\ \end{matrix} \xrightarrow[\text{reduction}]{\text{Row}} \begin{array}{c} \begin{matrix} -5 & & & \end{matrix} \\ \begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 \\ 5 & \infty & \infty & 0 \\ 8 & \infty & 0 & \infty \end{bmatrix} \end{array} \xrightarrow[\text{reduction}]{\text{Col}} \begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 \\ 0 & \infty & \infty & 0 \\ 3 & \infty & 0 & \infty \end{bmatrix}$$

$cost(2) = cost(1) +$ sum of reduction elements $+ m[A,B]$

$= 18 + 5 + 13 + 0$

$= 36$

2. Node 3 (go to vertex – C)

$m[A,C] = 7$, set row A & column C to ∞

set $m[C,A] = \infty$

$$\begin{bmatrix} \infty & 0 & 7 & 3 \\ 0 & \infty & \infty & 13 \\ 5 & \infty & \infty & 0 \\ 8 & 0 & 0 & \infty \end{bmatrix} \Longrightarrow \begin{bmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & 13 \\ \infty & \infty & \infty & 0 \\ 8 & 0 & \infty & \infty \end{bmatrix}$$

∴ Here no column reduction & Now row reduction.

∴ $cost(3) = cost(1) +$ sum of reduction elements $+ m[A,C]$

$= 18 + 0 + 7$

$= 25$

3. Node 4 (goto vertex – D)

→ $m[A,D] = 3$, set row A & column D to ∞

→ set $m[D,A] = \infty$

$$\begin{bmatrix} \infty & 0 & 7 & 3 \\ 0 & \infty & \infty & 13 \\ 5 & \infty & \infty & 0 \\ 8 & 0 & 0 & \infty \end{bmatrix} \Longrightarrow \begin{bmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \\ 5 & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty \end{bmatrix} \xrightarrow[-5 \text{ reduction}]{\text{row}} \begin{bmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty \end{bmatrix}$$

Now no column reduction here

$cost(4) = cost(1) +$ sum of reduction element $+ m[A,D]$

$= 18 + 3 + 5$

$= 26$

∴ In $cost(2)$, $cost(3)$, $cost(4)$

∴ $cost(3)$ is least.

∴ $cost(3) = 25$

$$A_3 = \begin{bmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & 13 \\ \infty & \infty & \infty & 0 \\ 8 & 0 & \infty & \infty \end{bmatrix}$$

step 3: Now consider all other vertices

1. Node 5 (goto vertex B)

$m[C,B] = \infty$

set row C & column B to ∞

set $[B,A] = \infty$

$$\begin{bmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & 13 \\ \infty & \infty & \infty & 0 \\ 8 & 0 & \infty & \infty \end{bmatrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 13 \\ \infty & \infty & \infty & \infty \\ 8 & \infty & \infty & \infty \end{bmatrix} \begin{matrix} \\ -13 \\ \\ -8 \end{matrix} \xrightarrow[\text{redu}]{\text{row}} \begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \end{bmatrix}$$

$$\text{cost}(5) = \text{cost}(3) + \text{sum of reduction elem} + m[C,B]$$

$$= 25 + 13 + 8 + \infty$$

$$= \infty$$

2. Node 6 (goto verte D)

$m[C,D] = 0$

set row C & column D to ∞

set $m[C,A] = \infty$

$$\begin{bmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & 13 \\ \infty & \infty & \infty & 0 \\ 8 & 0 & \infty & \infty \end{bmatrix} \Rightarrow \begin{bmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ 8 & 0 & \infty & \infty \end{bmatrix} \xrightarrow{\text{No}} \text{row \& column reduction}$$

$$\text{cost}(6) = 25 + 0 + 0$$

$$= 25.$$

In cost (5) & cost (6), cost(6) is least

route $A \rightarrow C \rightarrow D$

$$A_4 = \begin{bmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ 8 & 0 & \infty & \infty \end{bmatrix}$$

step 4:

Node 7 (goto vertex B)

$$m[D, B] = 0$$

set row D & column B to ∞ & set $[B, A] = \infty$

$$\begin{bmatrix} \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ 8 & 0 & \infty & \infty \end{bmatrix} \longrightarrow \begin{bmatrix} \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \end{bmatrix}$$

No row & column reduction

$\therefore$ Here all the element's are ∞

$$cost(7) = 25 + 0 + 0$$
$$= 25.$$

$\therefore$ The optimal path is $A \xrightarrow{12} C \xrightarrow{6} D \xrightarrow{2} B \xrightarrow{5} A$

$\therefore$ cost of optimal path = 25 units.