1. A What is Dynamic Programming? Compute the transitive closure of the graph given below using Warshall's Algorithm

Dynamic programming.

Dynamic programming is an algorithm design method that can be used when the solution to a problem can be viewed as a result of sequence of decisions.

→ used when problem breaks down into recurring smaller subproblems.

→ It is used when solution can be recursively described in terms of solution to subproblem.

→ Another important feature of dynamic programming approach is that optimal solutions of the subproblems are retained or stored so that they can be used later for solving other instances of same problem.

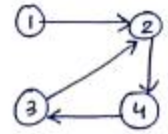
Basic Idea:

Optimal substructure: optimal solution to problem consists of optimal solution to sub problems

Overlapping subproblems: Few subproblems in total, many recurring instances of each.

It is solved by building table of solved subproblems that are used to solve larger ones.

Given graph is.



Adjacency Matrix is.

	1	2	3	4
1	0	1	0	0
2	0	0	0	1
3	0	1	0	0
4	0	0	1	0

$\therefore R^{(0)} =$

	1	2	3	4
1	0	1	0	0
2	0	0	0	1
3	0	1	0	0
4	0	0	1	0

One reflects the existence of path. Boxed row and column are used to get $R^{(1)}$.

$R^{(1)} =$

	1	2	3	4
1	0	1	0	0
2	0	0	0	1
3	0	1	0	0
4	0	0	1	0

one's reflect the existence of path with intermediate vertices numbered not higher than 1.
Boxed row & column are used to get $R^{(2)}$.

$R^{(2)} =$

	1	2	3	4
1	0	1	0	1
2	0	0	0	1
3	0	1	0	1
4	0	0	1	0

Since $(1,2)=1$ and $(2,4)=1 \therefore (1,4)=1$.
Since $(3,2)=1$ and $(2,4)=1 \therefore (3,4)=1$.
Boxed row and column are used to get $R^{(3)}$.

$R^3 =$

	1	2	3	4
1	0	1	0	1
2	0	0	0	1
3	0	1	0	1
4	0	1	1	1

Since $(4,3)=1$ & $(3,2)=1 \therefore (4,2)=1$
Since $(4,3)=1$ & $(3,4)=1 \therefore (4,4)=1$.
Boxed row and column are used to get $R^{(4)}$.

$R^{(4)} =$

	1	2	3	4
1	0	1	1	1
2	0	1	1	1
3	0	1	1	1
4	0	1	1	1

Since $(1,4)=1$ & $(4,3)=1 \therefore (1,3)=1$
Since $(2,4)=1$ & $(4,2)=1 \therefore (2,2)=1$
Since $(2,4)=1$ & $(4,3)=1 \therefore (2,3)=1$
Since $(3,4)=1$ & $(4,3)=1 \therefore (3,3)=1$
→ Transitive closure.

2. A) Solve the following instance of Knapsack problem using Dynamic Programming. Knapsack capacity is 5.

Knapsack problem using Dynamic programming.

Item.	Weight	Profit.
1	2	\$12
2	1	\$10
3	3	\$20
4.	2.	\$15.

W = Knapsack capacity = 5.

Initial conditions.

$V[0, j] = 0$ for $j \geq 0$ and

$V[i, 0] = 0$ for $i \geq 0$

V =

weight $\downarrow$	$i \downarrow$ \ w $j \rightarrow$	0	1	2	3	4	5
	0	0	0	0	0	0	0.
2	1	0	0	12.	12.	12	12.
1	2	0	10	12	22	22	22
3	3	0	10	12	22	30	32.
2	4	0	10	15	25	30	37

if $j < w_i$ then $V[i, j] = V[i-1, j]$.

$j \geq w_i$ then $V[i, j] = \max.\{V[i-1, j], v_i + V[i-1, j-w_i]\}$.

$V[1,1] = V[0,1] = 0$ since $j < w_1$

$V[1,2] = \max.\{V[0,2], v_1 + V[0,0]\}$

$= \max\{0, 12\} = 12$

$V[1,3] = \max.\{V[0,3], v_1 + V[0,1]\}$

$= \max\{0, 12\} = 12.$

$V[1,4] = \max\{V[0,4], v_1 + V[0,2]\}$

$= \max\{0, 12 + 0\} = 12.$

$V[2,1] = \max.\{V[1,1],\ V_2 + V[1,0]\}$

$= \max\{0,\ 10+0\}$

$= 10$

$V[2,2] = V[1,2] = 12.$

$V[2,3] = 22.$

$V[2,4] = 22$

$V[2,5] = 22.$

$V[3,1] = 10$

$V[3,2] = 12.$

$V[3,3] = 22$

$V[3,4] = 30$

$V[3,5] = 32.$

$V[4,1] = V[3,1] = 10$

$V[4,2] = \max\{V[3,2],\ V_4 + V[3,0]\}$

$= \max\{12,\ 15+0\}$

$= 15$

$V[4,3] = 25$

$V[4,4] = 30$

$V[4,5] = 37.$

2. B) Write the algorithm for Knapsack problem with Dynamic Programming

Algorithm: knapsack (W, n).
// compute the maximum value or profit using dynamic programming.
// Input: Capacity W, no of items n, weight & profit of n items.
// output: Maximum profit i.e V[n, W].

for j ← 0 to W do
  V[0, j] = 0
for i ← 1 to n do
  V[i, 0] = 0
for i ← 1 to n. do
  for j ← 1 to W do.
    if w_i <= j
      if v_i + V[i-1, j - w_i] > V[i-1, j]
        V[i, j] ← v_i + V[i-1, j - w_i]
      else.
        V[i, j] ← V[i-1, j]
    else
      V[i, j] ← V[i-1, j]

3) Solve and obtain the optimal Binary Search Tree from data given below for four keys

Key	A	B	C	D
Probability	0.1	0.2	0.4	0.3

Assumption No 1: fill $c[i, i]$ as P_i for $1 \le i \le n$

Assumption No 2: fill $c[i, i-1] = 0$ for. $1 \le i \le n+1$

Hence we have two tables.

i.e. $C[n+1][n+1]$ and. $R[n+1][n+1]$.

C =

	0	1	2	3	4
1	0	0.1			
2		0	0.2		
3			0	0.4	
4				0	0.3
5					0

diagonal 3.

diagonal 2

diagonal 1

R =

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

even $R[i, i] = i$

Now we should calculate the values for each diagonal.

When we calculate the values for diagonal 1 we consider ~~at~~ two keys at a time.

When we calulate the values for diagonal 2 we consider three keys

When we ~~consider~~ compute the values for diagonal 3. we consider all 4 keys. Hence.

$c[1][n]$ i.e $c[1][4]$ will give the optimal solution.

Calculating values for diagonal 1.

i.e $C[1,2] = \min \begin{cases} k=1: & C[1,0] + C[2,2] \\ k=2: & C[1,1] + C[3,2] \end{cases} + \sum_{s=1}^{2} P_s$

 i ↗ ↑ j

$= \min \begin{cases} 0 + 0.2 \\ 0.1 + 0 \end{cases} + \{0.1 + 0.2\}$

$= \min \{0.2, \underline{0.1}\} + \{0.3\}$.

$= 0.4$.

↳ This minimum value is obtained by $k=2$. Hence update $R[1,2] = 2$

$C[2,3] = \min \begin{cases} k=2: & C[2,1] + C[3,3] \\ k=3: & C[2,2] + C[4,3] \end{cases} + \sum_{s=2}^{3} P_s$.

 i ↗ ↑ j

$= \min \begin{cases} 0 + 0.4 \\ 0.2 + 0 \end{cases} + \{0.2 + 0.4\}$

$= \min. \{0.4, 0.2\} + 0.6$

$= 0.8$.

↳ This minimum value is obtained by $k=3$. Hence update $R[2,3] = 3$.

$C[3,4] = \min \begin{cases} k=3 & C[3,2] + C[4,4] \\ k=4 & C[3,3] + C[5,4] \end{cases} + \sum_{s=3}^{4} P_s$

 i ↗ ↑ j

$= \min \begin{cases} 0 + 0.3 \\ 0.4 + 0 \end{cases} + 0.7$

$= \min \begin{cases} 0.3 \\ 0.4 \end{cases} + 0.7$

Also $R[3,4] = 3$.

$= 1.0$

∴ Hence table looks as given below.

$C =$

	0	1	2	3	4	5
1	0	0.1	0.4	1.1	1.7	
2		0	0.2	0.8	1.4	
3			0	0.4	1.0	
4				0	0.3	
5					0	

$R =$

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

calculating values for diagonal · 2.

$$\underset{i\ \ j}{C[1,3]} = \min \begin{Bmatrix} k=1 & C[1,0]+C[2,3] \\ \cancel{k=2} & C[1,1]+C[3,3] \\ k=3 & C[1,2]+C[4,3] \end{Bmatrix} + \sum_{s=1}^{3} p_s$$

$$= \min \begin{Bmatrix} 0+0.8 \\ 0.1+0.4 \\ 0.4+0 \end{Bmatrix} + 0.1+0.2+0.4$$

$$= \min \{0.8,\ 0.5,\ 0.4\} + 0.7$$

$$= 0.4+0.7$$

$$= 1.1.$$

This minimum value is obtained by $k=3$. Hence update $R[1,3]=3$

Similarly $C[2,4] = 1.4$ (solve it).

Calculating value for diagonal 3. i.e $C[1,4]$ ← optimal Solution.

$$C[1,4] = \min \begin{Bmatrix} k=1 & C[1,0] + C[2,4] \\ k=2 & C[1,1] + C[3,4] \\ k=3 & C[1,2] + C[4,4] \\ k=4. & C[1,3] + C[5,4] \end{Bmatrix} + \sum_{s=1}^{4} p_s$$

(i j)

$$= \min \begin{Bmatrix} 0 + 1.4 \\ 0.1 + 1.0 \\ 0.4 + 0.3 \\ 1.1 + 0 \end{Bmatrix} + 0.1 + 0.2 + 0.4 + 0.3$$

$$= \min \begin{Bmatrix} 1.4 \\ 1.1 \\ 0.7 \ \cancel{0.8} \\ 1.1 \end{Bmatrix} + 1.0$$

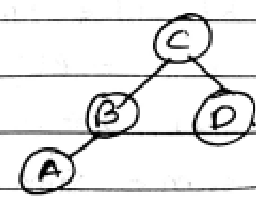
min value is obtained for $k=3$. Hence $R[1,4] = 3$.

$= 1.7.$

→ since $R[1,4] = 3$. The Root of binary search tree is 3rd key i.e. C.

→ since $R[1,2] = 2$. The Root of 1st key is 2nd key i.e Root of A is B.

Optimal Binary search tree is.



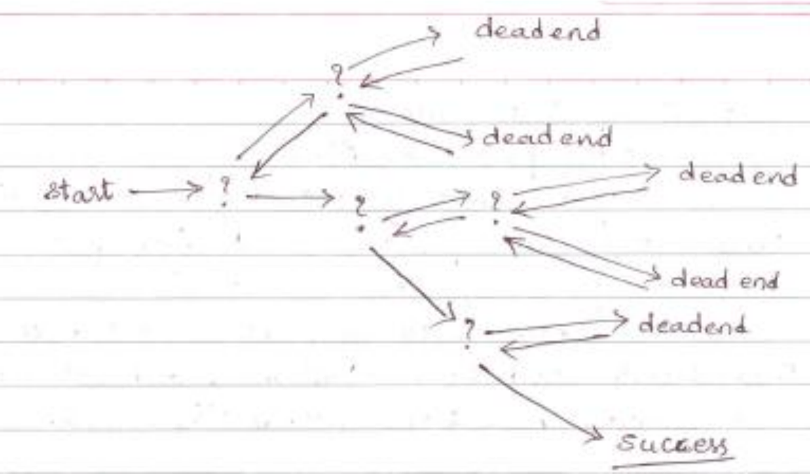
4. A Explain the concept of Backtracking and Branch and Bound

Backtracking :-

Suppose you have to make a series of decisions, among various choices, where

- You don't have enough information to know what to choose
- Each decision leads to a new set of choices
- Some sequence of choices (possibly more than one) may be a solution to your problem

Backtracking is a methodical way of trying out various sequences of decisions, until you find one that works.



Backtracking algorithm

ALGORITHM Backtrack(x[1⋯i])

// Gives a template of a generic backtracking algorithm
// Input : x[1⋯i] specifies first i promising components of solution
// Output : All the tuples representing the problems solutions

if x[1⋯i] is a solution
    write x[1⋯i]
else
    for each element $x \in S_{i+1}$ consistent with x[1⋯i] and the constraints do
        x[i+1] ← x
        Backtrack(x[1⋯i+1])

Branch and Bound :-

Feasible Solution is a point in the problems search space that satisfies all the problems constraints

Optimal Solution is a feasible solution with the best value of the objective function

Optimization Problem < Minimization Problem Eg:- TSP ⇒ Lowerbound
Maximization Problem Eg:- knapsack Prob ↳ Upper Bound

Branch and Bound strategy can be used to solve optimization problems without an exhaustive search in the average case

2 Mechanisms:

- A mechanism to generate branches when searching the solution space
- A mechanism to generate a bound so that many branches can be terminated.

It is efficient in the average case because many branches can be terminated very easily but a large state space tree is generated in a worst case.

Many NP-hard Problems can be solved by Branch and Bound efficiently in the average case, however the worst case time complexity is still exponential.

Compared to backtracking, branch-and-bound requires two additional items

- a way to provide, for every node of a state-space tree, a bound on the best value of the objective function on any solution that can be obtained by adding further components to the partially constructed solution represented by the node
- the value of the best solution so far.

4. B What is N-Queen's Problem? Illustrate 4-Queens problem using Backtracking and obtain the solution.

<u>n-Queens Problem</u>

Place n queens on an n-by-n chess board so that no two of them are in the same row, column or diagonal.

For $n=1$, 1×1 board

Q

— Trivial solution

For $n=2$, 2×2 board

Q1	

— No solution.

Q_2 ?

For $n=3$, 3×3 board

Q1		
		Q2

No solution

Q_3 ?

For $n=4$, 4×4 board

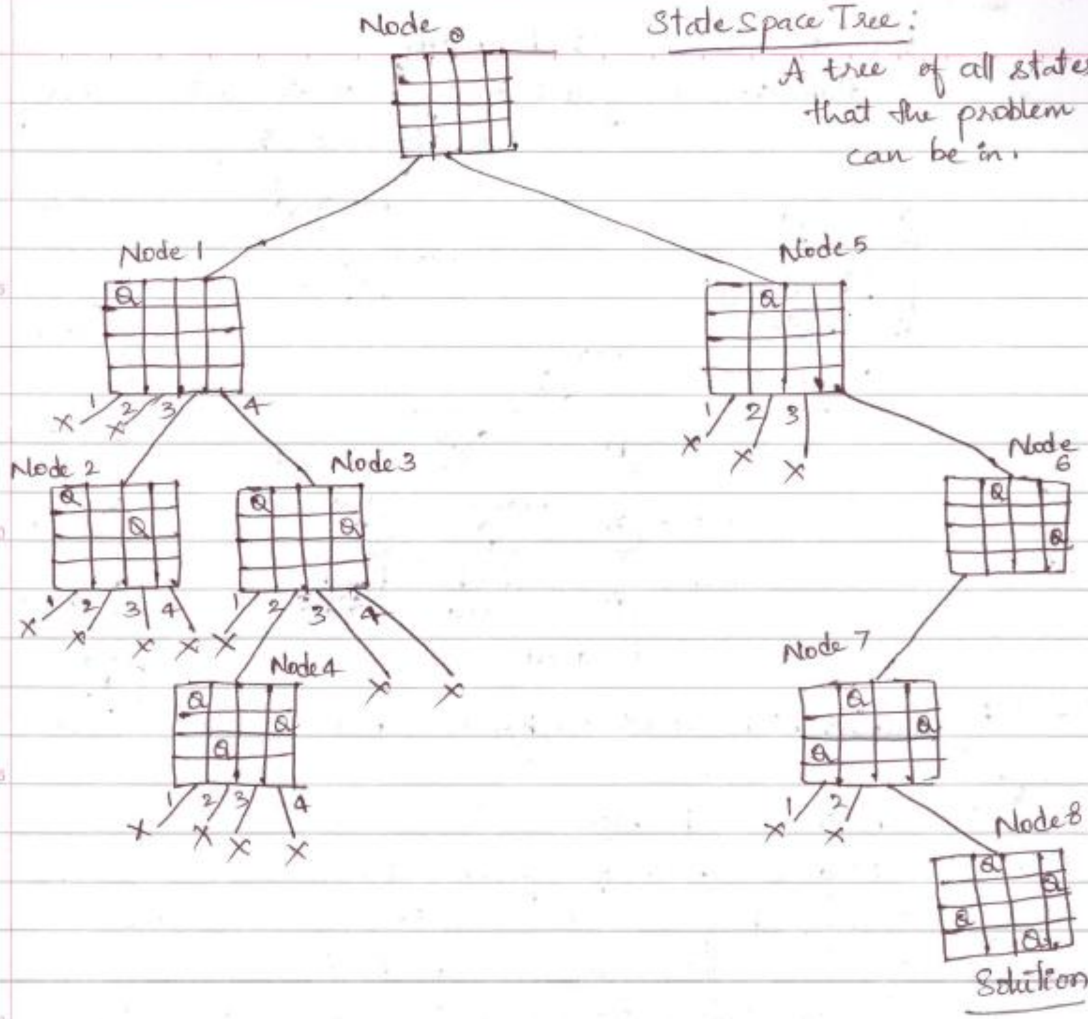
	1	2	3	4	
1					← queen 1
2					← queen 2
3					← queen 3
4					← queen 4

Attack positions →

Q	×	×	×
×	×		
×		×	
×			×

State Space Tree :

A tree of all states that the problem can be in.



5. A Solve subset sum problem for the following example S= {3, 5, 6, 7} and d=15. Construct a state space tree.

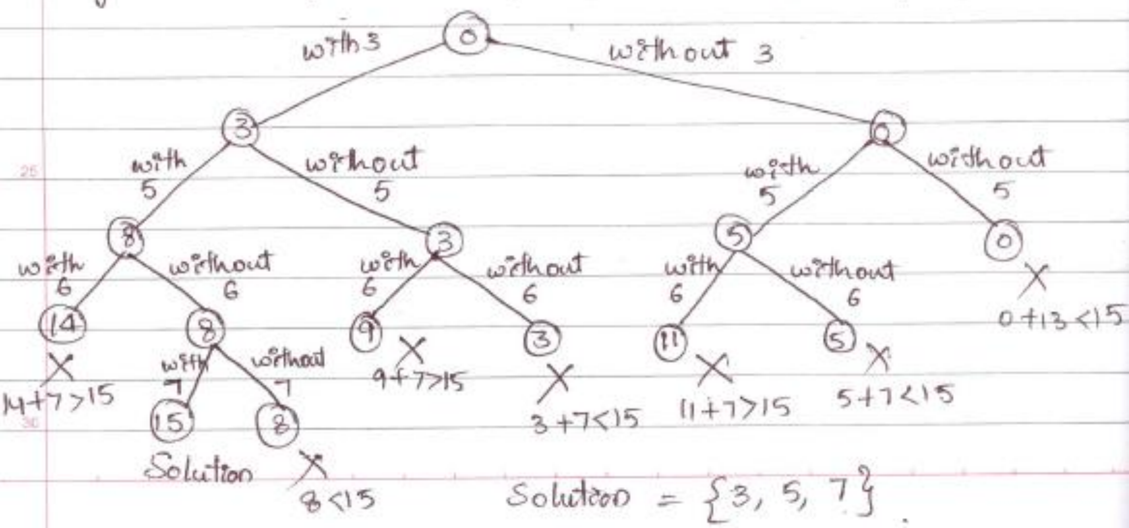
Subset - Sum Problem :-

Find a subset of a given set $S = \{s_1, \cdots s_n\}$ of n positive integers whose sum is equal to a given positive integer d.

Eg:- $S = \{1, 2, 5, 6, 8\}$ and $d = 9$

The two solutions are $\{1, 2, 6\}$ and $\{1, 8\}$

Eg:- State space tree for instance $S = \{3, 5, 6, 7\}$ and $d = 15$



Solution = $\{3, 5, 7\}$.

5. B Write algorithm for Graph Coloring problem

```
Algorithm mcoloring(int k)
{
    do {
        x[k] = getNodeColor(k);
        if (x[k] == 0) return; // No new color possible
        if (k == n) write(x[1...n]); // All vertices are coloured
        else mcoloring(k+1); // next node
    } while(true);
}
Algorithm getNodeColor(k)
{
    do {
        x[k] = (x[k]+1) mod (m+1); // Next highest color
        if (x[k] == 0) return; // All colors have been used
        for( j=1; j<=n; j++)
        {   if ( G[k][j] ≠ 0 && x[k] == x[j]) // same color
                break;
        }
        if (j == n+1) return; // new color found
    } while(1);
}
```

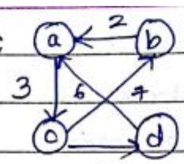
6. Solve and find the shortest path between each pair of vertices for the graph given below. Write the algorithm for Floyd's.

Algorithm: Floyd (W[1..n 1..n]).
// Input: weight matrix W of graph with no negative length cycle.
// Output: The distance matrix D of shortest paths lengths.

```
D(0) ← W
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            D[i,j] = min{ D[i,j], (D[i,k] + D[k,j]) }
return D
```

Time complexity = $\Theta(n^3)$.

Example:



Weight matrix $W = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & \infty & 3 & \infty \\ b & 2 & 0 & \infty & \infty \\ c & \infty & 7 & 0 & 1 \\ d & 6 & \infty & \infty & 0 \end{array}$

Stage 1

$D^{(0)} = W = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & \boxed{0} & \infty & 3 & \infty \\ b & 2 & 0 & \infty & \infty \\ c & \infty & 7 & 0 & 1 \\ d & 6 & \infty & \infty & 0 \end{array}$

→ $D^{(0)}$ is simply the weight matrix.

→ length of shortest path with no intermediate vertices.

→ Boxed row and column are used to generate $D^{(1)}$.

$$D^{(1)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & \infty & 3 & \infty \\ b & 2 & 0 & 5 & \infty \\ c & \infty & 7 & 0 & 1 \\ d & 6 & \infty & 9 & 0 \end{array}$$

→ lengths of shortest path by considering intermediate vertices not higher than 1. i.e just $\boxed{a}$.

Stage 2.

Two new shortest path from. b to c and d to c.

Boxed row and column are used for generating $D^{(2)}$.

$D^{(1)}[b,c] = \min\{D^{(0)}[b,c], (D^{(0)}[b,a] + D^{(0)}[a,c])\}$

$= \min\{\infty, 2+3\} = \min\{\infty, 5\}$

$= 5.$

Similarly. $D^{(1)}[d,c] = \min\{D^{(0)}[d,c], D^{(0)}[d,a] + D^{(0)}[a,c]\}$

$= \min\{\infty, 6+3\} = \min\{\infty, 9\}$

$= 9.$

$$D^{(2)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & \infty & 3 & \infty \\ b & 2 & 0 & 5 & \infty \\ c & 9 & 7 & 0 & 1 \\ d & 6 & \infty & 9 & 0 \end{array}$$

→ lengths of shortest path by considering intermediate vertices not higher than 2 i.e a and $\boxed{b}$

→ One new shortest path from. C to a.

Stage 3.

Boxed row and column are used for generating $D^{(3)}$.

$D^{(2)}[c,a] = \min\{D^{(1)}[c,a], D^{(1)}[c,b] + D^{(1)}[b,a]\}$

$= \min\{\infty, 7+2\} = 9.$

stage 4.

$$D^{(3)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 10 & 3 & \boxed{4} \\ b & 2 & 0 & 5 & \boxed{6} \\ c & 9 & 7 & 0 & 1 \\ d & 6 & 16 & 9 & 0 \end{array}$$

→ length of shortest path by considering intermediate vertices a, b, c

→ four new shortest path from a to b, a to d, b to d and d to b.

Boxed row and column are used for generating $D^{(4)}$.

$$D^{(3)}[a,b] = \min\{D^{(2)}[a,b],\ D^{(2)}[a,c] + D^{(2)}[c,b]\}$$
$$= \min\{\infty,\ 3+7\} = 10$$

$$D^{(3)}[a,d] = \min\{D^{(2)}[a,d],\ D^{(2)}[a,c] + D^{(2)}[c,d]\}$$
$$= \min\{\infty,\ 3+1\} = 4.$$

$$D^{(3)}[b,d] = \min\{D^{(2)}[b,d],\ D^{(2)}[b,c] + D^{(2)}[c,d]\}$$
$$= \min\{\infty,\ 5+1\} = 6.$$

$$D^{(3)}[d,b] = \min\{D^{(2)}[d,b],\ D^{(2)}[d,c] + D^{(2)}[c,b]\}$$
$$= \min\{\infty,\ 9+7\} = 16.$$

Stage 5. $D^{(4)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 10 & 3 & 4 \\ b & 2 & 0 & 5 & 6 \\ c & 7 & 7 & 0 & 1 \\ d & 6 & 16 & 9 & 0 \end{array}$

→ length of shortest path with intermediate vertices a, b, c, $\boxed{d}$

→ new shortest path from c to a

Final Distance matrix

$D^{(4)}[c,a] = \min\{D^{(3)}[c,a], D^{(3)}[c,d] + D^{(3)}[d,a]\}$

$= \min\{9, 1+6\}$

$= \min\{9, 7\}$

$= 7$

7. Solve the job assignment problem for the following and obtain the optimal solution

Assignment Problem :- (It is a Minimization Problem)

Assigning n people to n jobs so that the total cost of the assignment is as small as possible. An $n \times n$ cost matrix C is given

$$
\begin{array}{cc}
 & \begin{array}{cccc} \text{job1} & \text{job2} & \text{job3} & \text{job4} \end{array} \\
C = & \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix}
\end{array}
\begin{array}{l} \text{person a} \\ \text{person b} \\ \text{person c} \\ \text{person d} \end{array}
$$

Lower bount lb = sum of smallest elements in each row

$= 2 + 3 + 1 + 4 = 10$

smallest ele in row4

smallest ele in row1

Assigning job1 to person a, i.e. $a \to 1$, then cost $= 9$

then $lb = 9 + 3 + 1 + 4 = 17$

smallest ele in row 4

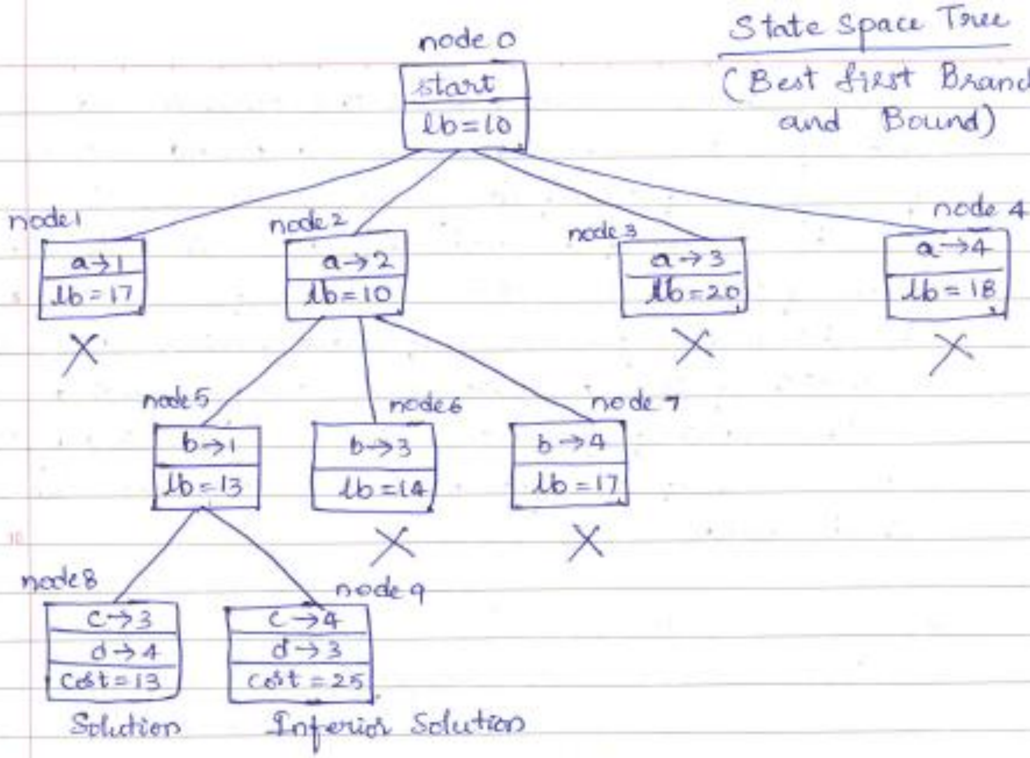
smallest ele in row 3

" " " row2

Assigning job 2 to person a, i.e $a \to 2$, then cost $= 2$

then $lb = 2 + 3 + 1 + 4 = 10$

State Space Tree
(Best first Branch and Bound)



Assignment

person	job	cost
a	2	2
b	1	6
c	3	1
d	4	4

Assignment cost = 13