→ All pairs shortest path problem (8) FLOYD WARSHAL
→ Single Source shortest paths general weights
→ String Edition.
→ 0/1 Knapsack problem.
→ Reliability Design.

INTRODUCTION:

Dynamic programming is typically applied to optimization problem. This technique is invented by a U.S. Mathematician Richard Bellman in 1950.

→ Dynamic programming is technique for solving problems with overlapping subproblems.

→ In this method each subproblem is solved only once. The result of each subproblem is recorded in a table which we obtain a solution to the original problem.

DIFFERENCE BETWEEN DIVID AND CONQUER AND DYNAMIC PROGRAMMING

S.NO	DIVIDE AND CONQUER	DYNAMIC PROGRAMMING
1	The problem is divided into small sub problems. These sub problems are solved Independently. Finally all the solutions of sub problems are collected together to get	In dynamic programming Many decision sequences are generated and all the overlapping subinstances are considered.

	the solution to the given problem.	
2	In this Method duplications in Sub-solutions are Neglected i.e, duplicate Subsolutions May be obtained.	In Dynamic Computing duplications in solutions is avoided totally.
3	Divide and Conquer is less efficient because of rework on solution	Dynamic programming is efficient than divide and Conquer strategy.
4	The divide and Conquer uses top down approach of problem solving (recursive method)	Dynamic programming uses bottom up approach of problem Solving (Iterative)
5	Divide and Conquer splits its Input at specific deterministic points usually in the middle	Dynamic programming splits its input at every possible split points rather than at a particular point. After trying all split points it determines which split point is optimal.

DIFFERENCE BETWEEN GREEDY ALGORITHM AND DYNAMIC PROGRAMMING

	GREEDY METHOD	DYNAMIC PROGRAMMING
1	Greedy Method is used for obtaining optimum Solution	Dynamic programming is also for obtaining optimum Solution.

2.	In Greedy Method a set a feasible Solutions and the picks up the optimum Solution	There is no special set of feasible Solutions in this Method.
3.	In Greedy Method the optimum Selected is without revising previously generated Solutions.	Dynamic programming Considers all possible sequences in order to obtain the optimum Solution.
4.	In Greedy Method there is no as Such guarantee of getting optimum Solution	It is guaranteed that the Dynamic programming will generate optimal solution using principal of optimality.

STEPS OF DYNAMIC PROGRAMMING:

→ Characterize the structure of optimal Solution. that means develop a Mathematical notation that can express any Solution and Subsolution for the given problem.

→ Recursively define the Value of an optimal Solution.

→ By using bottom up technique Compute Value of optimal Solution. For that you have to develop a recurrence relation that relates to its Subsolutions, using the mathematical notation of step 1

→ Compute an optimal solution from Computed Information.

** PRINCIPLE OF OPTIMALITY:

The Dynamic programming algorithm obtains the solution using principle of optimality.

The principle of optimality states that "in an optimal sequence of decisions or choices, each subsequence must also be optimal.

When it is not possible to apply the principle of optimality it is almost impossible to obtain the solution using the Dynamic programming approach.

ALL PAIRS SHORTEST PATH PROBLEM:

PROBLEM DESCRIPTION:

When a weighted graph, represented by its weight Matrix W then objective is to find the distance between every pair of nodes.

Step 1: we will decompose the given problem into subproblems.

let,

$A^k_{(i,j)}$ be the length of shortest path from node i to node j such that the label for every Intermediate node will be <=k

We will Compute A^k for k=1-----n for n nodes.

Step 2: For solving all pair shortest path, the principle of optimality is used. That means any subpath of shortest path is a shortest path between the end nodes. Divide the paths from i node to j node for every intermediate node, say 'k' the there arises two cases.

(i) path going from i to j via k

(ii) path which is not going via k. Select only shortest path from two cases.

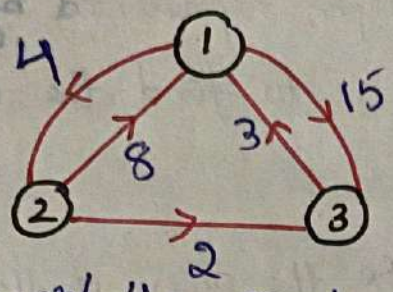
Step 3: The shortest path can be computed using bottom up Computation method.

Intially : $A^0 = W[i,j] \rightarrow A^K[i,j] =$

Next Computations : $\min \left\{ A^{K-1}_{(i,j)}, A^{K-1}_{(i,k)} + A^{K-1}_{(k,j)} \right\}$

Where $1 \leq K \leq n$

EX: Solve Minimum distances (or) All pairs shortest path problem using Dynamic programming.



→ Now Convert the graph into Matrix form

$$A^0 = \begin{bmatrix} 0 & 4 & 15 \\ 8 & 0 & 2 \\ 3 & \infty & 0 \end{bmatrix}$$

Now find out A^1 Matrix

$$A^1 = \begin{bmatrix} 0 & 4 & 15 \\ 8 & 0 & - \\ 3 & - & 0 \end{bmatrix}$$

$A^1_{11} = 0$

$A^1_{12} = \text{Min}\{A^{1-1}[1,2], A^{1-1}[1,1] + A^{1-1}[1,2]\}$

$= \text{Min}\{A^0[1,2], A^0[1,1] + A^0[1,2]\}$

$= \text{Min}\{4, 0+4\}$

$= 4$

$A^1_{13} = \text{Min}\{A^{1-1}[1,3], A^{1-1}[1,1] + A^{1-1}[1,3]\}$

$= \text{Min}\{15, 0+15\}$

$= 15$

$A^1_{21} = \text{Min}\{A^{1-1}[2,1],\ A^{1-1}[2,1] + A^{1-1}[1,1]\}$

$= \text{Min}\{8, 8+0\}$

$= \underline{8}$

$A^1_{22} = 0$

$A^1_{23} = \text{Min}\{A^{1-1}[2,3],\ A^{1-1}[2,1] + A^{1-1}[1,3]\}$

$= \text{Min}\{2, 8+15\}$

$= \text{Min}\{2, 23\}$

$= 2$

$A^1_{31} = \text{Min}\{A^{1-1}[3,1],\ A^{1-1}[3,1] + A^{1-1}[1,1]\}$

$= \text{Min}\{3, 3+0\}$

$= 3$

$A^1_{32} = \text{Min}\{A^{1-1}[3,2],\ A^{1-1}[3,1] + A^{1-1}[1,2]\}$

$= \text{Min}\{\infty, 3+4\}$

$= \text{Min}\{\infty, 7\}$

$= 7$

$A^1_{33} = 0$

$$A^1 = \begin{bmatrix} 0 & 4 & 15 \\ 8 & 0 & \underline{2} \\ 3 & \textcircled{7} & 0 \end{bmatrix} \qquad A^2 = \begin{bmatrix} 0 & 4 & - \\ 8 & 0 & 2 \\ - & 7 & 0 \end{bmatrix}$$

$A^2_{11} = \text{Min}\{A^{2-1}[1,1],\ A^{2-1}[1,2] + A^{2-1}[1,1]\}$

$= \text{Min}\{0, 4+8\}$

$= \underline{0}$

$$A^2_{12} = \text{Min}\{A^{2-1}[1,2],\ A^{2-1}[1,2] + A^{2-1}[2,2]\}$$
$$= \text{Min}\{4,\ 4+0\}$$
$$= 4$$

$$A^2_{13} = \text{Min}\{A^{2-1}[1,3],\ A^{2-1}[1,2] + A^{2-1}[2,3]\}$$
$$= \text{Min}\{15,\ 4+2\}$$
$$= 6$$

$$A^2_{21} = \text{Min}\{A^{2-1}[2,1],\ A^{2-1}[2,2] + A^{2-1}[2,1]\}$$
$$= \text{Min}\{8,\ 0+8\}$$
$$= 8$$

$$A^2_{22} = 0$$

$$A^2_{23} = \text{Min}\{A^{2-1}[2,3],\ A^{2-1}[2,2] + A^{2-1}[2,3]\}$$
$$= \text{Min}\{2,\ 0+2\}$$
$$= 2$$

$$A^2_{31} = \text{Min}\{A^{2-1}[3,1],\ A^{2-1}[3,2] + A^{2-1}[2,1]\}$$
$$= \text{Min}\{3,\ 7+8\}$$
$$= \text{Min}\{3,\ 15\} = 3$$

$$A^2_{32} = \text{Min}\{A^{2-1}[3,2],\ A^{2-1}[3,2] + A^{2-1}[2,2]\}$$
$$= \text{Min}\{7,\ 7+0\}$$
$$= 7$$

$$A^2_{33} = 0$$

$$A^2 = \begin{bmatrix} 0 & 4 & 6 \\ 8 & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix} \qquad A^3 = \begin{bmatrix} 0 & - & 6 \\ - & 0 & 2 \\ 3 & 7 & 0 \end{bmatrix}$$

Now find A^3 Matrix, here $K=3$

$A^3_{11} = 0$

$A^3_{12} = \text{Min}\{A^2[1,2], A^2[1,3] + A^2[3,2]\}$

$= \text{Min}\{4, 6+7\}$

$= 4$

$A^3_{13} = \text{Min}\{A^2[1,3], A^2[1,3] + A^2[3,3]\}$

$= \text{Min}\{6, 6+0\}$

$= 6$

$A^3_{21} = \text{Min}\{A^2[2,1], A^2[2,3] + A^2[3,1]\}$

$= \text{Min}\{8, 2+3\}$

$= 5$

$A^3_{22} = 0$

$A^3_{23} = \text{Min}\{A^2[2,3], A^2[2,3] + A^2[3,3]\}$

$= \text{Min}\{2, 2+0\}$

$= 2$

$A^3_{31} = \text{Min}\{A^2[3,1], A^2[3,3] + A^2[3,1]\}$

$= \text{Min}\{3, 0+3\}$

$= 3$

$A^3_{32} = \text{Min}\{A^2[3,2], A^2[3,3] + A^2[3,2]\}$

$= \text{Min}\{7, 0+7\} = 7$

$A^3_{33} = 0$

$$A^3 = \begin{bmatrix} 0 & \underline{4} & 6 \\ (5) & 0 & 2 \\ 3 & 7 & 2 \end{bmatrix}$$

Now find A^3 Matrix, here $k=3$

$A^3_{11} = 0$

$A^3_{12} = \text{Min}\{A^2[1,2], A^2[1,3] + A^2[3,2]\}$

$= \text{Min}\{4, 6+7\}$

$= 4$

$A^3_{13} = \text{Min}\{A^2[1,3], A^2[1,3] + A^2[3,3]\}$

$= \text{Min}\{6, 6+0\}$

$= 6$

$A^3_{21} = \text{Min}\{A^2[2,1], A^2[2,3] + A^2[3,1]\}$

$= \text{Min}\{8, 2+3\}$

$= 5$

$A^3_{22} = 0$

$A^3_{23} = \text{Min}\{A^2[2,3], A^2[2,3] + A^2[3,3]\}$

$= \text{Min}\{2, 2+0\}$

$= 2$

$A^3_{31} = \text{Min}\{A^2[3,1], A^2[3,3] + A^2[3,1]\}$

$= \text{Min}\{3, 0+3\}$

$= 3$

$A^3_{32} = \text{Min}\{A^2[3,2], A^2[3,3] + A^2[3,2]\}$

$= \text{Min}\{7, 0+7\} = 7$

$A^3_{33} = 0$

$$A^3 = \begin{bmatrix} 0 & \underline{4} & 6 \\ \textcircled{5} & 0 & 2 \\ 3 & 7 & 2 \end{bmatrix}$$

ALGORITHM

```
Algorithm All-pair (W,A)
{
  for i=1 to n do
    for j=1 to n do
      A[i,j] = W[i,j]
  for k=1 to n do
    {
    for i=1 to n do
      {
       for j=1 to n do
        {
       A[i,j] = min (A[i,j], A[i,k]+A[k,j]);
      }
    }
  }
}
```

ANALYSIS:

From above algorithm

→ The first double for-loop takes $O(n^2)$ time.

→ The nested three for loop takes $O(n^3)$ time

→ Thus, the Whole algorithm takes $O(n^3)$ time.

0/1 KNAPSACK PROBLEM:

PROBLEM DESCRIPTION:

IF we are given n objects and a knapsack or a Bag in which the object i that has weight w_i is to be placed. The knapsack has a capacity W. Then the profit has that can be earned is $P_i X_i$. The objective is to obtain finally of knapsack with Maximum profit earned.

Maximized $\sum_{n}^{1} P_i X_i$ Subject to Contruct $\sum_{n}^{1} w_i x_i \leq W$ where $1 \leq i \leq n$ and n is total number of objects. And $X_i = 0$ or 1

The greedy method does not work for this problem. To solve this problem using dynamic programming method.

Intially Compute

$$S^0 = \{0,0\}$$

$$S_1^i = \{(P,W) / (P - P_i, W - \omega_i) S^i\}$$

S^{i+1} Can be Computed by Merging S^i and S_1^i

PURGING RULE: (OR) DOMINANCE RULE:

IF S^{i+1} Contains (P_j, W_j) and (P_K, ω_K); these two pairs such that

$P_j \leq P_K$ and $W_j \geq W_K$, then (P_j, W_j) Can be eliminat This purging rule is also called as dominance rule. In purging rule basically the dominated tuples gets purged. In short, remove the pair with less profit and More weight.

Ex: Consider $n=4$, $m=8$ $(P_1, P_2, P_3, P_4) = (1,2,5,6)$ and weights $(\omega_1, \omega_2, \omega_3, \omega_4) = (2,3,4,5)$

→ Consider $S^0 = \{(0,0)\}$

Now $S_1^0 = \{$ Select first pair (P_1, ω_1) and add it with $S^0\}$

$= \{S^0 + (1,2)\}$

$S_1^0 = \{(1,2)\}$

$S^1 = \{$ Merge S^0 and $S_1^0\}$

$S^1 = \{(0,0), (1,2)\}$

$S_1^2 = \{$ Select next pair (P_2, ω_2) and add it with $S^1\}$

$= \{S^1 + (2,3)\}$

$S_1^2 = \{(2,3), (3,5)\}$

$S^2 = \{(0,0), (1,2), (2,3), (3,5)\}$

$S_1^3 = \{$ Select next pair $(5,4)$ and add it with $S^2\}$

$= \{S^2 + (5,4)\}$

$= \{(5,4), (6,6), (7,7), (8,9)\}$

$S^3 = \{$ Merge S^2 and $S_1^3\}$

$S^3 = \{(0,0), (1,2), (2,3), (3,5), (5,4), (6,6), (7,7), (8,9)\}$

apply purging rule $(3,5)$ and $(5,4)$

$3 \leq 5$ (T) and $(5 \geq 4)$ (T)

Hence remove $(3,5)$

$S^3 = \{(0,0), (1,2), (2,3), (5,4), (6,6), (7,7)\}$

S_1^4 = { select next pair (5,5) and add it with S^3 }

= $\{S^3 + (6,5)\}$

= {(6,5) (7,7) (8,8) (9,10) (11,9), (12,11), (13,11), (13,12) (14,14)}

S^4 = { Merge S^3 and S_1^4 }

{(0,0) (1,2) (2,3) (5,4) (6,6) (6,5) (7,4) (7,7) (8,9) (8,8) (9,10), (11,9), (12,11), (13,11), (13,12), (14,14) }

apply purging rule on (7,7) and (7,7)

$7 \leq 7$ (T) and $7 \geq 7$ (T)

Hence remove (7,7)

apply purging rule on (6,6) and (5,5)

$6 \leq 6$ (T) and $6 \geq 5$ (T)

Hence remove (6,6)

S^4 = {(0,0), (1,2), (2,3), (5,4), (6,5), (7,7), (8,8)}

from S^4 Consider (8,8) pair

(i) $(8,8) \in S^4$

$(8,8) \notin S^3$ ∴ select fourth object pair ∴ $x_4 = 1$

$((8,8),(8-5)) = (2,3)$

(ii) $(2,3) \in S^3$

$(2,3) \in S^2$ ∴ Don't select Third object because it was belongs to S^2 ∴ $x_3 = 0$

(iii) $(2,3) \in S^2$

$(2,3) \notin S^1$ ∴ select 2nd object pair ∴ $x_2 = 1$

$((2,-2),(3,-3)) = (0,0)$

4(iv) $(0,0) \in S^1$ ∴ Don't select first object pair

$(0,0) \in S^0$ it was belongs to S^0 ∴ $x_1 = 0$

∴ $\{0, 1, 0, 1\}$ → final Answer.

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

ALGORITHM DKP (P, ω, n, m)

{

$S^0 = \{(0,0)\}$;

for i=1 to n-1 do

{

$S_1^{i-1} = \{(P,\omega) / (P-P_i, W-\omega_i) \in S^{i-1}$ and $W \le m\}$;

S^i = Merge purge (S^{i-1}, S_1^{i-1});

}

(PX, WX) = Last pair in S^{n-1};

$(PY, WY) = (P'+P_n, W'+\omega_n)$ where W' is the Largest W in any pair in S^{n-1} such that $W + \omega_n \le m$;

if $(PX > PY)$ then $x_n = 0$;

else $x_n = 1$;

TrackBack for $(x_{n-1}, \dots x_1)$;

}

*** → Time Complexity $= \sum_{0 \le i \le n-1} |S^i| = \sum_{0 \le i \le n-1} 2^i = 2^n - 1$

∴ Time Complexity = $O(2^n)$

SINGLE SOURCE SHORTEST PATH PROBLEM:

→ Single Source shortest path problem in Dynamic programming to solve by using Bellman & FORD Algorithm.

→ Greedy Single Source shortest path problem does not allow negative cost. But where as Bellman Ford algorithm allows Negative.

→ To solve Single Source shortest path problem in Dynamic programming Bellman & Ford Introduced Relaxation rule. that is

if $d(u) + cost(u,v) < dv$ then

change as $d(v) = d(u) + cost(u,v)$

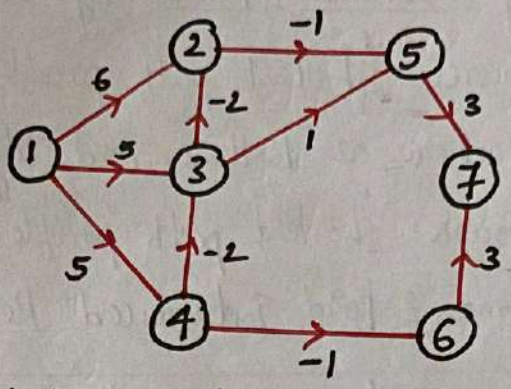
→ The differences between Single Source shortest path in Greedy Method and Single Source shortest path in D.P.

SSSP IN G.M	SSSP IN D.P
→ Greedy Single Source shortest path problem is solved by using Dijkstra's algorithm.	→ Dynamic programming Single Source shortest path problem is Solved by using Bellman Ford algorithm.
→ It is Connected and weighted graph	→ It is also a Connected and weighted graph.
→ In greedy SSSP we can Obtain an optimal solution from possible No. of feasible solutions	→ In Dynamic programming SSSP we Can Satisfy all choices (or) decisions.
→ In greedy SSSP we Can't Solve -ve edge cost in the given graph.	→ In Dynamic programming we Can Solve the problem even the edge cost May be -ve.

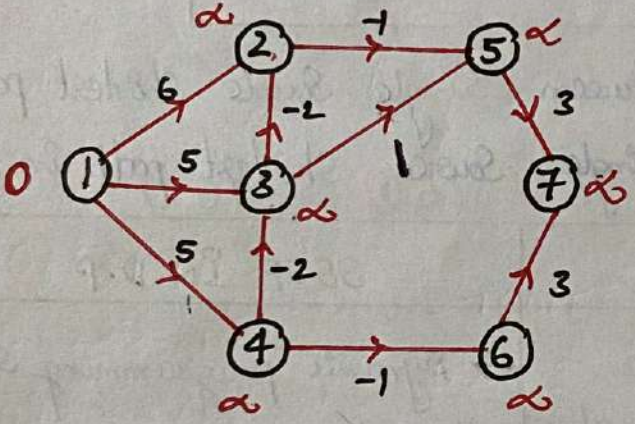
→ In greedy SSSP finally we can written shortest path and Minimum distance.

→ In Dynamic SSSP we can written individual Minimum distances.

EX:



→ Intially the distance of Source Vertex is '0' and remaining Vertex distances is '∝'



$d(1) = 0$
$d(2) = \infty$
$d(3) = \infty$
$d(4) = \infty$
$d(5) = \infty$
$d(6) = \infty$
$d(7) = \infty$

→ Now form edge list is

(1,2), (1,3), (1,4), (2,5), (3,2), (3,5), (4,3), (4,6), (5,7), (6,7)

i.e $d(u) + cost(u,v) < d(v)$ then change as

$d(v) = d(u) + Cost(u,v)$

① $0+6 < \infty$ (T) then
$d(v) = 6$

$0+5 < \infty$ (T) then
$d(v) = 5$

$0+5 < \infty$ (T) then
$d(v) = 5$

$6-1 < \infty$ (T)
$d(v) = 5$

$5-2 < 6$ (T)
$3 < 6$ (T)
$d(v) = 3$

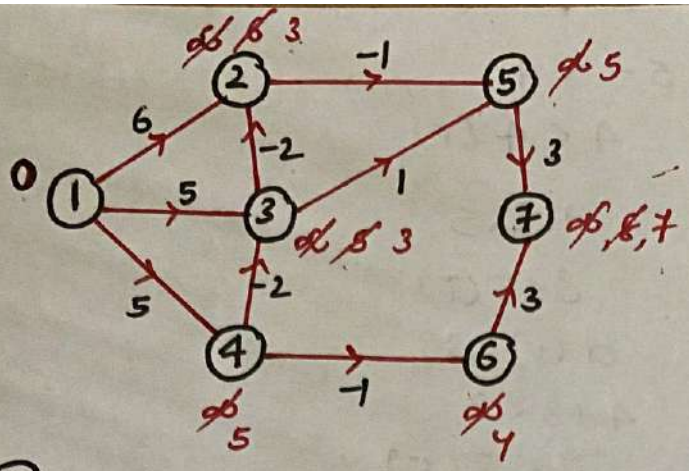
$5+1 < 5$
$6 < 5$ (F)
$d(v) = 4$

$5-2 < 5$
$3 < 5$ (T)

$5-1 < \infty$
$4 < \infty$ (T)
$d(v) = 4$

$5+3 < \infty$
$8 < \infty$ (T)
$d(v) = 8$

$4+3 < 8$
$7 < 8$ (T)



$d(1) = 0$
$d(2) = 3$
$d(3) = 3$
$d(4) = 5$
$d(5) = 5$
$d(6) = 4$
$d(7) = 7$

②

$0+6 < 3$ (F)

$0+5 < 3$ (F)

$0+5 < 5$ (F)

$3-1 < 5$

$2 < 5$ (T)

$d(v) = 2$

$3-2 < 3$

$1 < 3$ (T)

$d(v) = 1$

$3-1 < 2$

$2 < 2$ (F)

$5-2 < 3$

$3 < 3$ (F)

$5-1 < 4$

$4 < 4$ (F)

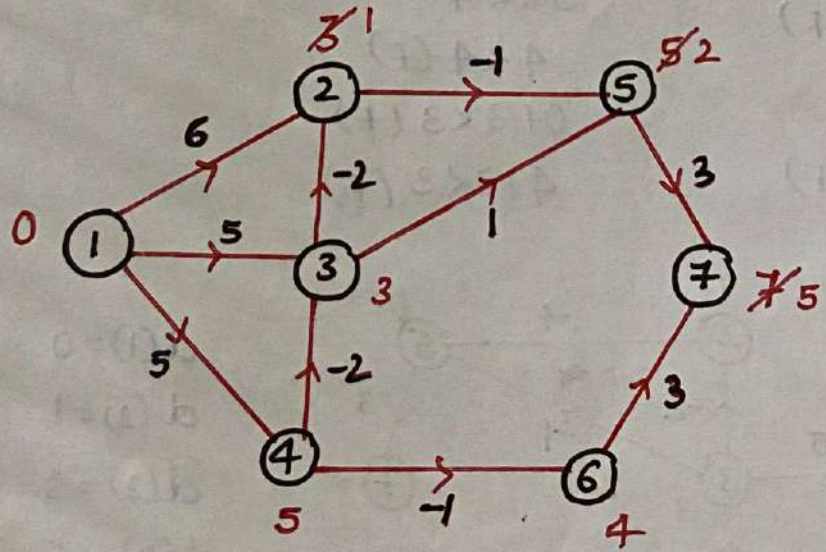
$2+3 < 7$

$5 < 7$ (T)

$d(v) = 5$

$4+3 < 5$

$7 < 5$ (F)



$d(1) = 0$
$d(2) = 1$
$d(3) = 3$
$d(4) = 4$
$d(5) = 2$
$d(6) = 4$
$d(7) = 5$

③

$0+6 < 1$ (F)

$0+5 < 3$ (F)

$0+5 < 5$ (F)

$1-1 < 2$

$0 < 2$ (T)

$d(v) = 0$

$3-2<1$

$1<1$ (F)

$3-1<0$

$2<0$ (F)

$5-2<3$

$3<3$ (F)

$5-1<4$

$4<4$ (F)

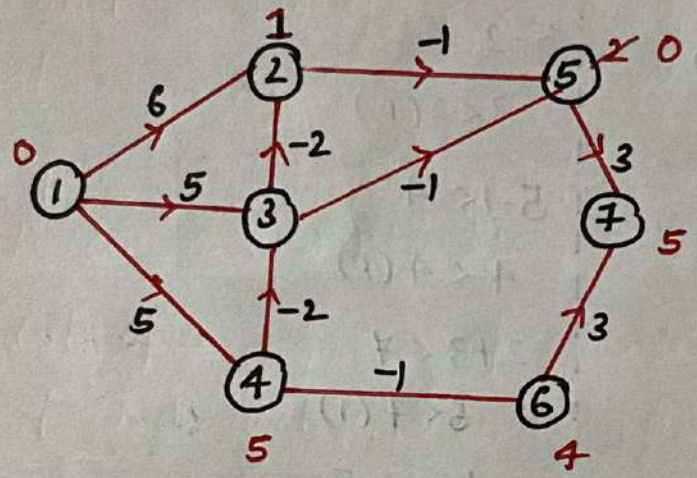
$0+3<5$

$3<5$ (T)

$d(v)=3$

$4+3<3$

$7<3$ (F)



$d(1)=0$

$d(2)=1$

$d(3)=3$

$d(4)=5$

$d(5)=0$

$d(6)=4$

$d(7)=3$

④ $0+6<1$ (F)

$0+5<3$ (F)

$0+5<5$ (F)

$1-1<0$

$0<0$ (F)

$3-2<1$

$1<1$ (F)

$3-1<0$ (F)

$5-2<3$

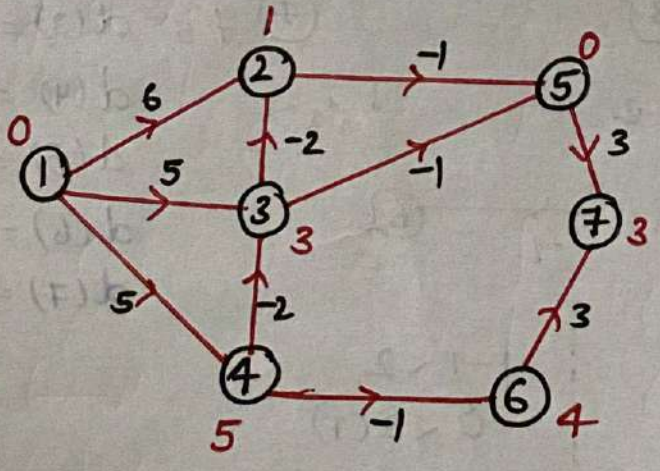
$3<9$ (F)

$5-1<4$

$4<4$ (F)

$0+3<3$ (F)

$4+3<3$ (F)



$d(1)=0$

$d(2)=1$

$d(3)=3$

$d(4)=5$

$d(5)=0$

$d(6)=4$

$d(7)=3$

ALGORITHM:

```
1. Algorithm BellmanFord (V, cost, dist, n)
2. {
3.   for i=1 to n do
4.     dist[i] = cost[V,i];
5.   for k=2 to n-1 do
6.     for each u such that u ≠ v and u has
7.       at least one Incoming edge do
8.       for each <i, u> in the graph do
9.         if dist[u] > dist[i] + cost[i,u] then
10.          dist[u] = dist[i] + cost[i,u];
11. }
```

TIME COMPLEXITY:

→ Each Iteration of the for loop of lines 5 to 10 takes $O(n^2)$ time if adjacency matrices are used and $O(e)$ time if adjacency lists are used.

→ Here e is the Number of edges in the graph.

→ The Overall Complexity is $O(n^3)$ when adjacency Matrices are used and $O(ne)$ when adjacency lists are used.

STRING EDITION (MINIMUM EDIT DISTANCE):

→ The given two strings $X = x_1, x_2, x_3 \cdots x_n$ and $Y = y_1, y_2, y_3, \cdots y_m$ where x_i is $1 \le i \le n$ y_j is $1 \le j \le m$ are members of a finite set of symbols

→ we want to transform XY using a sequence of edit operations on X.

→ The permissiable edit operations are 1) Insert, 2) delete (or) Remove and 3) Replace (or) change.

→ The cost of a sequence of operations is the sum of the costs of Individual operations in the sequence.

→ The problem of string editing is to Identify a Minimum cost sequence of editing operations that will transform 'X' into 'Y'

→ let $D(x_i)$ be the cost of deleting the symbol x_i from X

→ let $I(y_j)$ be the cost of Inserting the symbol y_j into X

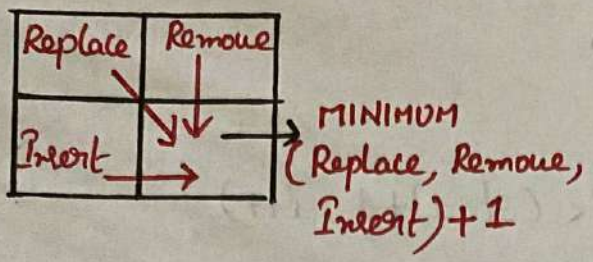
→ let $C(x_i, y_j)$ be the cost of changing the symbol x_i of X into y_j of Y.

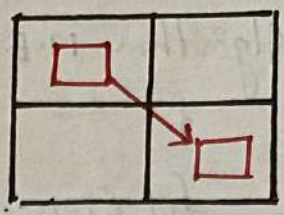
→ To solve string edition problem we can use the formula

$$\text{Cost}(i,j) = \min \begin{cases} \text{Cost}(i-1,j) + D(x_i), \\ \text{Cost}(i-1,j-1) + C(x_i, y_j), \\ \text{Cost}(i,j-1) + I(y_j) \end{cases}$$

- Insert (→)
- Remove (↓)
- Replace (↘)

→ 1. If $x \neq c$ 2. If $x = c$





Just copy the diagonal element

Ex: Consider the string edition problem $X = aabab$, $Y = babb$

→

i \ j	NULL	b	a	b	b
NULL	0	1	2	3	4
a	1	(1)	1	2	3
a	2	2	(1)	2	3
b	3	2	2	(1)	2
a	4	3	2	(2)	2
b	5	4	3	2	(2) → ANSWER

$X = a\ a\ b\ a\ b$

$Y = b\ a\ b\ b$

→ If b=b then apply replace operation without adding 1, Hence we can not perform any operation on x & y

→ If a≠b then apply remove operation and add with 1 Hence the string a is removed from 'x'.

→ If b=b then apply replace operation without adding 1 Hence we can not perform any operation on 'x' & 'y'

→ If a=a then apply replace operation without adding 1 Hence we can not perform any operation on x & y.

→ If a≠b then apply replace operation from Minimum cost add with 1

Finally we can perform two operations on 'x' transfer to 'y'. i.e one remove operation a
one replace operation a→b

Hence final Minimum cost is 2.

ALGORITHM :

```
Algorithm M.E.D
  {
  for (i=0; i< (str1)+1; i++)
  {
    for (j=0; j< leng(str2)+1; j++)
    {
      if (i==0 && j==0)
        T[i][j]=0;
    else
        (i==0)
      T[i][j] = T[i][j-1]+1;
    else (j==0)
      T[i][j] = T[i-1][j]+1;
    else
    { if (str1[i-1] == str2[j-2])
        T[i][j] = T[i-1][j-1];
      else
        T[i][j] = 1+ Min ( T[i][j-1], Insert
                           T[i-1][j], Remove
                           T[i-1][j-1], Replace )
    }
  }
}
```

∴ Time Complexity = $O(m.n)$

= $O(n.n)$

= $\underline{O(n^2)}$