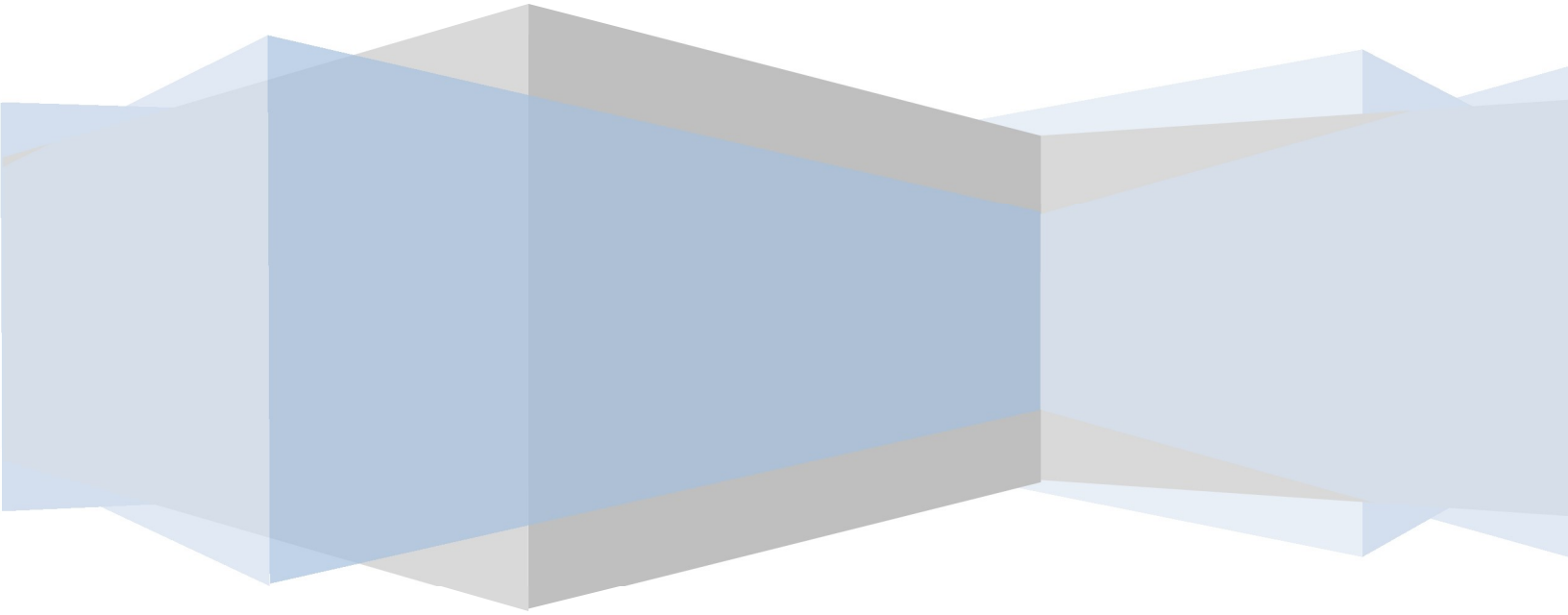
UNIT – 4: DYNAMIC PROGRAMMING



TOPICS:

1. *Introduction*
2. *Multistage Graph with Backward and Forward Algorithm*
3. *All Pair Shortest Path Algorithm*
4. *Single Source Shortest Path Algorithm*
5. *Travelling Salesman Problem*

INTRODUCTION & PRINCIPLE OF DYNAMIC PROGRAMMING METHODOLOGY:

This method of algorithm design is used when solution of problem is dependent on series of decisions. For such problems, it is necessary to try all the possible decisions and select the best decision. This may increase time and space requirement. The principle avoids the iterations related with the decisions, which will not result in optimal sequence.

Principle of Optimality: The solution is designed using series of decisions, will be optimal in nature, if all the sub-decisions involved in making the final decisions are also optimal.

For Example: If path from node "X" to "Z" is optimal, and if "p" to "q" is part of this path, then the path "p" to "q" must be optimal.

I) MULTI-STAGE GRAPH:

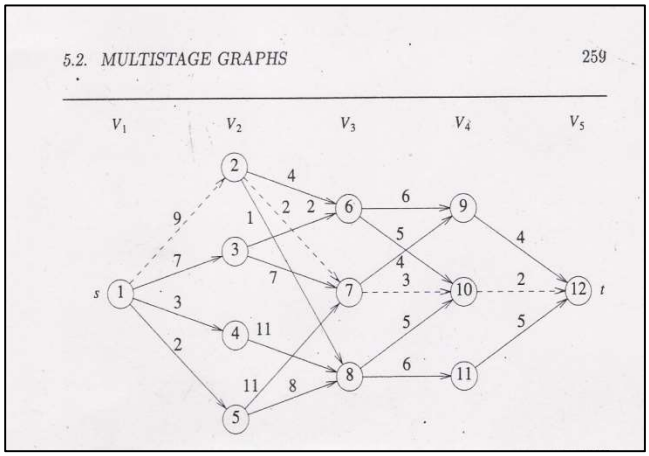
Features:

1) It is a directed graph consist of Vertices and Edges
2) In this graph the vertices are partitioned into k >= 2 disjoint sets. These sets are denoted as Vi, where 1 <= i <= k
3) If there is an edge <u, v>, then the vertex "u" belongs to set "Vi" and vertex "v" belongs to set "V_{i+1}"
4) The number of vertices in the first and last set i.e., V_1 and V_k are 1, that is there will be only one source and one destination vertex.

Objective:

The objective is to find the shortest path between the source and destination vertex. The path should be through all the stages of the graph. That is if there are FIVE stages in the graph, then there should be FIVE vertices in the shortest path, where each vertex belongs to one stage.

Example:



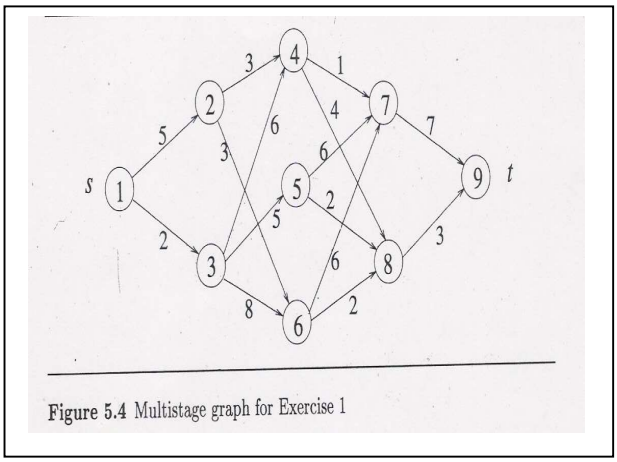


Figure 5.4 Multistage graph for Exercise 1

There are two ways to solve the Multi-stage Graph problem: Backward Algorithm and Forward Algorithm.

Backward Algorithm:

Using this method, the shortest path from source to destination, is generated using backward movement. The starting stage of the graph is denoted as stage 2, so that the backward movement can be implemented to stage 1.

The algorithm uses formula:

$$bcost(i, j) = \min_{l \in V_{i-1},\ \langle l, j\rangle \in E} \{ bcost(i-1, l) + cost(l, j) \}$$

In this: "i" represent : STAGE *"j" represents: VERTICES OF STAGE*
i -1 represents: PREVIOUS STAGE *"l" represenst: VERTICES OF PREVIOUS STAGE*
The graph is represented using "cost" matrix of size "nxn".

Solving for the graph given:

Step 1: Stage – 2

$$bcost(2, 3) = \min_{l \in V_{i-1},\ \langle l, j\rangle \in E} \{ bcost(i-1, l) + cost(l, j) \}$$

Hence: i=2, j=3,4,5,6 (all the vertices of Stage 2), l=1 (only vertex of stage 1) such that it has edge to vertex "j".

$$bcost(2, 3) = \min_{l \in 1,\ \langle 1,3\rangle \in E} \{ bcost(1, 1) + cost(1,3) \} = \min(0 + 9) = 9$$

Step 2: Stage – 3

Vertices of the stage – 3 are 6, 7, 8. Hence different values of "j" are 6, 7 and 8. Value of "i=3"
For vertex 6, the possible links from stage 2 are from vertices 2 and 3. Hence values of "l" are 2 and 3.

$$bcost(3, 6) = \min_{l \in 2,3,\ \langle 2,6\rangle, \langle 3,6\rangle \in E} \begin{cases} bcost(2,2) + cost(2,6) \\ bcost(2,3) + cost(3,6) \end{cases}$$

$$= \min\{(9+4), (7+2)\} = 9$$

For vertex 7, the possible links from stage 2 are from vertices 2, 3 and 5. Hence values of "l" are 2, 3, 5.

$$bcost(3, 7) = \min_{l \in 2,3,5,\ \langle 2,7\rangle, \langle 3,7\rangle \langle 5,7\rangle \in E} \begin{cases} bcost(2,2) + cost(2,7) \\ bcost(2,3) + cost(3,7) \\ bcost(2,5) + cost(5,7) \end{cases}$$

$$= \min\{(9+2), (7+7), (2+11)\} = 11$$

For vertex 8, the possible links from stage 2 are from vertices 2, 4 and 5. Hence values of "l" are 2, 4, 5.

bcost (3 , 8) = min $\begin{cases} bcost(2,2) + cost(2,8) \\ bcost(2,4) + cost(4,8) \\ bcost(2,5) + cost(5,8) \end{cases}$

l € 2,3,5

<2,8>, <4,8> <5,8> € E

= min {(9+1), (3+11), (2+8)} = 10

Step – 3: Stage 4

Vertices of the stage – 3 are 9, 10, 11. Hence different values of "j" are 9,10 &11. Value of "i=4"
For vertex 9, the possible links from stage 3 are from vertices 6 & 7. Hence values of "l" are 6 and 7.

bcost (4 , 9) = min $\begin{cases} bcost(3,6) + cost(6,9) \\ bcost(3,7) + cost(7,9) \end{cases}$

l € 6,7

<6,9>, <7,9> € E

= min {(9+6), (11+4)} = 15

For vertex 10, the possible links from stage 3 are from vertices 6, 7 & 8. Hence values of "l" are 6 , 7 and 8.

bcost (4 , 10) = min $\begin{cases} bcost(3,6) + cost(6,10) \\ bcost(3,7) + cost(7,10) \\ bcost(3,8) + cost(8,10) \end{cases}$

l € 6,7,8

<6,10>, <7,10> <8,10>€ E

= min {(9+5), (11+3) (10+5)} = 14

For vertex 11, the possible links from stage 3 is only from vertex 8, hence value of l=8.

bcost (4 , 11) = min $\{bcost(3,8) + cost(8,11)$

l € 8

<8,11>€ E

= (10+6) = 16

Step 4: Stage 5

Vertex at stage 5 is 12. This vertex is linked with vertices 9, 10 and 11 of stage 4, hence value of l=9, 10 and 11.

bcost (5 , 12) = min $\begin{cases} bcost(4,9) + cost(9,12) \\ bcost(4,10) + cost(10,12) \\ bcost(4,11) + cost(11,12) \end{cases}$

l € 9, 10, 11

<9,12> <10,12> <11,12> € E

= min {(15+4), (14+2), (16+5)} = 16

Hence the minimum distance from source to destination, visiting all the stages is 16.

Algorithm: Backward Method

Assumptions:

1) The graph is represented using cost matrix of size [nxn].
2) Array d[1..n] is a temporary array used to hold vertices information
3) Array p[1..k] is a output array of size "k", where "k" is number of stages in the graph.

```
Algorithm backward_cost(G, cost, d, n : p)
{
    Step 1:
        bcost[1] = 0 //Assume vertex 1 as start vertex
    Step 2:
        For j = 2 to n do
        {
        Function: Find vertex "r" such that <r,j> is an edge in the graph and bcost[r] + cost[r,j]
is minimum
                bcost[j] = bcost[r] + cost[r,j];
                d[j] = r;
        } //End of For loop
    Step 3: Finding minimum cost path
        p[1] = 1;        p[k] = n
        For j = k-1 to 2 do
                p[j] = d[p[j+1]];
} //End of Algorithm
```

Execution:

Step – 1:

bcost[1] = 0

Step – 2.1

j = 2, 3, 4, 5 (all the vertices are at stage 2 and for all these vertices value of r is vertex 1)

Hence: d[2]=d[3]=d[4]=d[5]=1

And bcost[2]=cost[1,2]=9 bcost[3]= cost[1,3]=7 bcost[4]=cost[1,4]=3, bcost[5]=cost[1,5]=2

bcost	0	9	7	3	2	9	11	10	15	14	16	16
Index	1	2	3	4	5	6	7	8	9	10	11	12
D	-	1	1	1	1	3	2	5	7	7	8	10
index	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*

Step-2.2: Stage 3 vertices (Refer calculations)

j = 6, 7, 8

For vertex 6: find the value of "r" satisfying the condition of minimum cost.

Vertex "r=3" hence d[6]=3 and bcost[6] = bcost[r] + cost[r,j] = bcost[3] + cost[3,6] = 7 + 2 = 9

Similarly for other vertices 7 and 8, bcost[7] = 11 and bcost[8] = 10, d[7]=2 & d[8]=5

Step 2.3: Stage 4 vertices (Refer calculations)

For vertex 9: Find the value of "r" satisfying the condition of minimum cost.

Vertex r=7 hence d[9]=7 and bcost[9] = bcost[7] + cost[7,11] = 11+4=15

FORWARD ALGORITHM:

Using this method, the shortest path from source to destination, is generated using FORWARD movement. The starting stage of the graph is denoted as stage K-1, so that the FORWARD movement can be implemented to stage K. (Here "k" represents number of stages).

The algorithm uses formula:

$$fcost(i, j) = \min_{l \in V_{i+1},\ \langle j,l \rangle \in E} \{ fcost(i+1, l) + cost(j,l) \}$$

In this: "i" represent : STAGE *"j" represents: VERTICES OF STAGE*
i +1 represents: Forward STAGE *"l" represenst: VERTICES OF Forward STAGE*
The graph is represented using "cost" matrix of size "nxn".

Solving for the graph given:
Step-1: Stage (k-1=4)

$$fcost(4, 9) = \min_{l \in V_{i+1},\ \langle j,l \rangle \in E} \{ fcost(i+1, l) + cost(j,l) \}$$

Here i=4 and j=9,10, 11 (all the vertices of stage 4). The connection of these vertices to stage 5 (l=5) vertices will be now checked in forward algorithm. That is the connection to vertex 12.

$$fcost(4, 9) = \min_{l \in 5,\ \langle 9,12 \rangle \in E} \{ fcost(5, 12) + cost(9,12) \} = \min(0 + 4) = 4$$

$fcost(4,9) = 4$

Similarly

$fcost(4,10) = 2$

$fcost(4,11) = 5$

Step – 2: Stage(k-2=3)

Here i=3, and j=6,7,8 (all the vertices of stage 3). The connection of these vertices to stage 5 will decide the value of "l".

For vertex 6, the connection of vertex 6 to next stage vertices is 9 & 10, hence l=9,10.

$$fcost(3, 6) = \min_{l \in 9,10,\ \langle 6,9 \rangle, \langle 6,10 \rangle \in E} \begin{cases} fcost(4,9) + cost(6,9) \\ fcost(4,10) + cost(6,10) \end{cases}$$

$= \min\{(4+6), (2+5)\}$

$= 7$

Similarly for vertex 7, the connection of vertex 7 are to vertex 9 & 10, hence l=9,10

$$fcost(3,7) = \min_{l \in 9,10,\ \langle 7,9 \rangle, \langle 7,10 \rangle \in E} \begin{cases} fcost(4,9) + cost(7,9) \\ fcost(4,10) + cost(7,10) \end{cases}$$

$= \min\{(4+4), (2+3)\} = 5$

Similarly for vertex 8, the connection of vertex 8 are to vertices 10 and 11, hence value of l=10, 11

fcost(3,8) = min $\begin{cases} fcost(4,10) + cost(8,10) \\ fcost(4,11) + cost(8,11) \end{cases}$

l € 10,11

<8,10>,<8,11> € E

= min{(2+5), (5+6)}

= 7

Step-3: Stage (k-3=2)

Students can solve it further based on above logic

```
Algorithm Forward_cost(G, cost, d, n : p)
{
    Step 1:
        fcost[n] = 0 //Assume vertex 1 as start vertex
    Step 2:
        For j = n-1 to 1    step -1 do
        {
        Function: Find vertex "r" such that <r,j> is an edge in the graph and bcost[r] + cost[r,j]
is minimum
                fcost[j] = fcost[r] + cost[j,r];
                d[j] = r;
        } //End of For loop
    Step 3: Finding minimum cost path
        p[1] = 1;          p[k] = n
        For j = 2 to k-1 do
                p[j] = d[p[j-1]];
} //End of Algorithm
```

II) ALL PAIR SHORTEST PATH ALGORITHM:

This algorithm is used to find shortest path from all the vertices present in the graph to all the vertices present the graph. With this algorithm, any vertex of the graph can act as source and all remaining vertices will be considered as destination.

A matrix is generated using algorithm, which represent the shortest distance from source to all the destination.

Steps:

1) The graph is initially represented in the form of a direct path matrix, which contains information about the direct paths present in the graph.
2) To calculate the new paths, a vertex is added as intermediate vertex in every phase of calculation and if the existing path is greater than the path generated after adding intermediate, then updation is performed.

Solution for Graph-1

The matrix representing direct path matrix for the given graph will be:

0	**4**	**11**
6	**0**	**2**
3	∞	**0**

With Vertex 1 as intermediate

0	4	11
6	0	2
3	7	0

With vertex 1 and Vertex 2 as intermediate

0	4	6
6	0	2
3	7	0

With Vertex 1, 2 and 3 as intermediate

0	4	6
5	0	2
3	7	0

Solution for Graph-2

Direct Path matrix [Given]

0	5	∞	∞
50	0	15	15
30	∞	0	15
15	∞	5	0

Vertex 1 as intermediate

0	5	∞	∞
50	0	15	5
30	35	0	15
15	20	5	0

Vertex 1, 2 as intermediate

0	5	20	10
50	0	15	5
30	35	0	15
15	20	5	0

Vertex 1, 2, 3 as intermediate

0	5	20	10
45	0	15	5
30	35	0	15
15	20	5	0

Vertex 1, 2, 3, 4 as intermediate

0	5	15	10
20	0	10	5
30	35	0	15
15	20	5	0

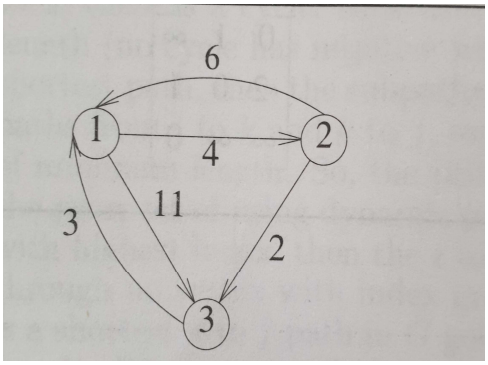
Algorithm: All pair shortest path Algorithm

Assumptions

1) The graph is represented using cost matrix of size nxn
2) The algorithm will generate output matrix in the form of matrix A of size nxn

```
Algorithm allpaths(G,cost,n: A)
{
   Step-1: Direct Path Matrix
           for i  = 1 to n do
             for j = 1 to n do
               A[i, j] = cost[i, j];
  Step – 2: Using intermediate vertices one by one
           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])
}//end of algorithm
```

Example for Home Work

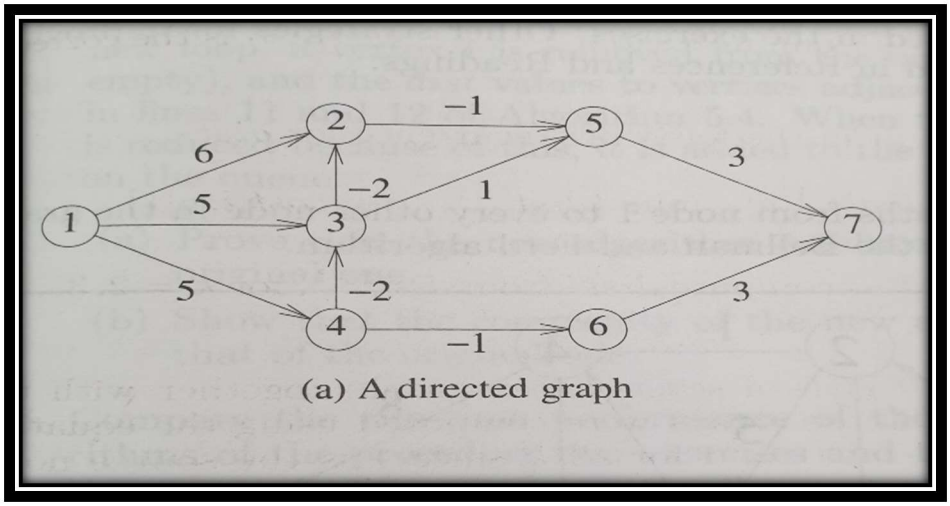


III) SINGLE SOURCE SHORTEST PATH ALGORITHM: BELLMAN FORD ALGORITHM

This algorithm will find shortest path from one source to all destination present in the graph. This algorithm is so designed that it also permits, edges with Negative Weights. When negative weights are permitted, it is essential that graph should not have any cycle.

When there is no cycle of negative length, the shortest path between any two vertices of "n" vertex graph, can at the most have n-1 edge in it.

Let $dist^{l}[u]$ be the length of shortest path from source vertex "v" to vertex "u", under the constraints that shortest path must contain "l" edges, then $dist^{1}[u] = cost[v,u]$. Similarly $dist^{n-1}[u]$ is shortest path from "v" to "u" of length "n-1".



Formula used for computing Shortest Path:

$dist^{k}[u] = min \{ dist^{k-1}[u] , min \{ dist^{k-1}[i] + cost[i, u]\}\}$

where "i" represents all the vertices except vertex "u" present in the graph. The vertex "i" should have connection to vertex "u" in the graph.

Execution:

Let k=1, then length of path between source to all possible vertices will be "1". This will also represent direct path. Value of "u" will be 2..7 and if path does not exist then distance will be 99 or infinity.

$dist^{1}[u] = min \{ dist^{0}[u] , min \{ dist^{0}[i] + cost[i, u]\}\}$

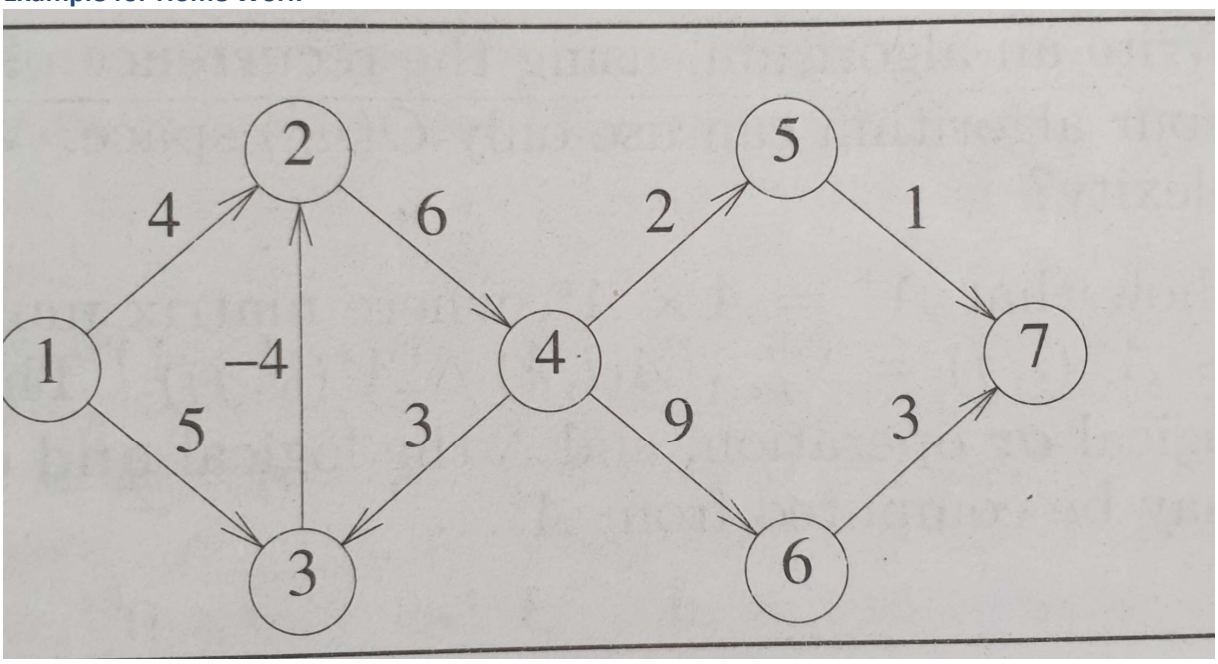
$dist^{1}[2] = min \{ dist^{0}[2] , min \{ dist^{0}[i] + cost[1, u]\}\}$
$dist^{1}[3] = min \{ dist^{0}[3] , min \{ dist^{0}[i] + cost[1, u]\}\}$
$dist^{1}[4] = min \{ dist^{0}[4] , min \{ dist^{0}[i] + cost[1, u]\}\}$
$dist^{1}[5] = min \{ dist^{0}[5] , min \{ dist^{0}[i] + cost[i, u]\}\}$
$dist^{1}[6] = min \{ dist^{0}[6] , min \{ dist^{0}[i] + cost[i, u]\}\}$
$dist^{1}[7] = min \{ dist^{0}[7] , min \{ dist^{0}[i] + cost[i, u]\}\}$

Let k=2, then length of path between source to all possible vertices will be “2”.During calculation, the values calculated for k=1, will be used to find values for k=2.

$dist^2[u] = min \{ dist^1[u] , min \{ dist^1[i] + cost[i, u]\}\}$

K	1	2	3	4	5	6	7
1	0	6	5	5	∞	∞	∞
2	0	3	3	5	5	4	∞
3	0	1	3	5	2	4	7
4	0	1	3	5	0	4	5
5	0	1	3	5	0	4	3
6	0	1	3	5	0	4	3

Example for Home Work



Algorithm:

```
Algorithm Bellman Ford Algo(v, cost, dist, n)
{
 Step 1
        For i= 1 to n do
                Dist[i] = cost[v,i]
Step 2
        For k = 2 to n-1 do
            For (each vertex “u” such that u≠v and “u” has at least one incoming edge) do
         // Let “i” represent the incoming edge
                For (each <i,u> in the graph) do
                        If (dist[u] > dist[i] + cost[i,u]) then
                                dist[u] = dist[i] + cost[i,u]
} // End of Algorithm
```

TRAVELLING SALEMAN PROBLEM: (TSP)

This is one of the classical problems, which can be solved using Dynamic Programming. The main objective is this problem is to find a cycle of shortest distance between source and destination, visiting all the vertices, once. This problem is implemented on Di-graph.

Applications: Postal System, Robotics, Manufacturing Industries.

Let G(V,E) be a digraph. Let Cij be the cost of the edge, then Cij>0, if the edge is present in the graph otherwise, Cij=00.

Methodology:

To find the shortest path, consider vertex 1 an the source vertex, then since cycle is to be generated, vertex 1 will be also destination vertex.

Step-1:

The process of obtaining the shortest length cycle starts with pre-final vertices of the path. For example if in a graph, there are "n" vertices and vertex "1" is source, then pre-final vertices will be 2..n.

The path from pre-final vertex will end to source vertex "1".

Step -2:

The process then consider, information obtained in step 1 and vertices which can be at index, pre to pre-final. Then this path can start with 2..n vertices and will use 2..n vertices as intermediate to reach to destination (source vertex) 1.

For example if vertex 3 is selected from the available vertices for pre to pre-final position, then remaining vertices 2 and 4..n will be used at pre-final position.

Formula used:

$$g(i,S) = \min_{j \in S} \{Cij + g(j - S-\{j\})$$

In this formula:

"i" represents source vertex

"j" represents the next vertex used as intermediate.

"S" represents the vertices other than "i" and if out of these vertices "j" vertex is used as next intermediate, then remaining vertices left will be S-{j}.

Example: Consider a complete Directed Graph, represented in the form of matrix

0	10	15	20
5	0	9	10
6	13	0	12
8	8	9	0

Step – 1:

Using this graph: if vertex "1" is assumed as source vertex, then the various possible vertices at pre-final position of the cyclic path will be 2,3,4 and these vertices will be connected with an edge to start vertex 1.

Hence

g(2, *) = C21 = 5

g(3,*) = C31 = 6

g(4,*) = C41 = 8

Step -2: Now if pre to pre-final position of path is considered then if at this index vertex 2 is used, then only possible path to continue to source 1, should be through vertex 3. Similarly it is possible to use vertex 3 and 4 at pre to pre-final positions of the path.
Vertex 2 at pre-to-pre-final position
g(2, {3}} = {C23 + g(3, *) } = 9 + 6 = 15
g(2, {4}} = {C24 + g(4, *) } = 10 + 8 = 18
Vertex 3 at pre-to-pre-final position
g(3, {2}} = {C32 + g(3, *) } = 13 + 5 = 18
g(3, {4}} = {C34 + g(4, *) } = 12 + 8 = 20
Vertex 4 at pre-to-pre-final position
g(4, {2}} = {C42 + g(2, *) } = 8 + 5 = 13
g(4, {3}} = {C43 + g(3, *) } = 9 + 6 = 15

Step – 3: Using the values calculated at step 1 and step 2 and finding the details for one position back in the path. Now if vertex 2 is used as selected vertex, then the remaining path will consist of vertex 3 and 4 and hence there will be two possibilities.
g(2, {3,4}} = MIN [{C23 + g(3,{ 4})}, C24 + g(4,{3}) } = MIN[9 + 20, 10 + 15] = 25

g(3, {2, 4}} = MIN[{C32 + g(2, {4})} , C34 + g(4, {2})] = MIN[13 + 18, 12 + 13] = 25

g(4, {2, 3}) = MIN[{C42 + g(2, {3})}, C43 + g(3, {2})}] = MIN[8+ 15, 9 + 18] = 23

Step 4: Finally the required path will be generated by considering the results of steps 1, 2 and 3

g(1, {2,3,4}) = MIN[{C12 + g(2, {3,4})}, C13 + g(3, {2, 4})} , C14 + g(4, {2,3})}
= MIN[10 + 25, 15 + 25, 30+23] = MIN[25, 40, 43] = 35

To find the actual path:

1 2 4 3 1 ➔ Cost = 35