CHAPTER -2
Shortest Path From A Specified Vertex To All Vertices

2.1 INTRODUCTION

Basically a shortest path from vertex to vertex has only unique solution, that means only specific path of the graph is covered, where as if the shortest path for any particular complete graph can be calculated from vertex to all vertices.

Sometimes it is interesting to find the shortest path between all \( n(n-1) \) ordered pairs of vertices in a digraph or \( n(n-1)/2 \) unordered pairs of vertices in an undirected Graph. If we use the Dijkstra's algorithm for this purpose, the Computation time would be proportional to \( n^4 \). There are several algorithms available that can do better. Among these, two are considered best, both being equally efficient, one of them is due to Dantziz[3] and the other is due to Floyd[6], based on a procedure by Warshall[13]. Both algorithms require computation time proportional to \( n^3 \).

The Problem of finding a shortest path from vertex to all pairs of vertices is an extension of the problem which was considered in Chapter 1. In continuation of the previous chapter, we described a Computer programming in C- language for the problem of finding the shortest path from specified vertex to all pairs of vertices. First, we find the shortest path for a weighted directed graph. Least weighted directed edges are considered from source vertex to terminal vertex. As the shortest path, in continuation, we have to findout all the pairs of shortest paths that are present in the given weighted directed graph.
Finally, if we calculate their costs, the one which is having the least weight is considered to be the feasible shortest-path. If there occurs, a tie between them it is considered to be the feasible shortest path, and the one having less cost is not considered. [5,7,7,8,7 is a shortest path cost is 7]. If there occurs, a tie of costs [like 7,7,11,11,14] then we will take the least tied cost path as the feasible shortest path. [Shortest path cost is 7].

Example :1
Consider the following Directed Weighted Graph

Fig (3.1): Weighted Graph
For this Graph, the input is in the form of Adjacency Directed Weighted Graph.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>8</td>
<td>2</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>∞</td>
<td>0</td>
<td>∞</td>
<td>4</td>
<td>3</td>
<td>∞</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
<td>1</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>E</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
<td>∞</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>10</td>
<td>4</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>G</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>2</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
</tr>
</tbody>
</table>

Here we consider shortest-path from vertex to all pairs.

Shortest-path from B to G

1) 

Cost of Shortest - path = 11
Root is B → C → F → G
2) Cost of Shortest path = 18
   Root is B → A → D → E → G

3) Cost of Shortest path = 11
   Root is B → A → E → G

4) Cost of Shortest path = 7
   Root is B → C → E → G
Now we will consider, least cost is the cost of the shortest-path from vertex to all vertices.
So cost is 11. (because 11, 11 tie occurs, so the tied values considered to be the Shortest-path).

Shortest-path from vertex B to another vertex G, cost is 11.

Shortest-path from vertex to all vertices costs are $\{11, 18, 11, 7, 10, 21\}$

Cost of Shortest-path = 10
Root is $B \rightarrow C \rightarrow F \rightarrow E \rightarrow G$

Cost of Shortest-path = 21
Root is $B \rightarrow A \rightarrow D \rightarrow A \rightarrow E \rightarrow G$
Example: 2

Consider the following Directed Weighted Graph

For this graph, the input is in the form of Adjacency Directed weighted Graph.

\[
\begin{array}{cccc}
  v_1 & v_2 & v_3 & v_4 \\
  v_1 & 0 & 2 & \infty & \infty \\
  v_2 & \infty & 0 & 3 & 2 \\
  v_3 & \infty & 4 & 0 & 5 \\
  v_4 & 1 & 0 & 6 & 0 \\
\end{array}
\]

Here the shortest-path in the form adjacency matrix and directed graph is vertex to all vertices \( v_1 \) to \( v_3 \)

1)

Cost of Shortest-path = 5
Root is \( v_1 \to v_2 \to v_3 \)
So shortest-path from vertex to all pairs of vertices so cost's are \{5,10\}

Now we will consider, least cost is the cost of the shortest-path from vertex to all vertices.

So cost is 5. Root is \(v_1 \rightarrow v_2 \rightarrow v_4 \rightarrow v_3\)
2.2 WARSHALL - FLOYD ALGORITHM:

Starting with the n by n matrix $D = [d_{ij}]$ of direct distances, n different matrices $D_1, D_2, D_3, \ldots, D_n$ are constructed sequentially. Matrix $D_k$, $1 \leq k \leq n$ may be thought of as the matrix whose $(i, j)$th entry gives the length of the shortest directed path among all directed paths from $i$ to $j$, with vertices $1, 2, \ldots, k$ allowed as the intermediate vertices. Matrix $D_k = [d_{ij}^{(k)}]$ is constructed from $D_{k-1}$ according to the following rule.

$$d_{ij}^{(k)} = \min \{ d_{ij}^{(k-1)}, [d_{ik}^{(k-1)} + d_{kj}^{(k-1)}] \}, \text{ for } k = 1, 2, \ldots, n \text{ and } d_{ii}^{(0)} = d_{ii}.$$

That is, in Iteration 1, vertex 1 is inserted in the path from vertex $i$ to $j$ if $d_{ii} > d_{i1} + d_{1j}$.

In Iteration 2, vertex 2 is inserted, and so on.

Suppose, for example, that the Shortest directed path from vertex 7 to 3 is 741953. The following replacements occur.

Iteration 1: $d_{49}^{(0)}$ is replaced by $(d_{41}^{(0)} + d_{19}^{(0)})$

Iteration 4: $d_{79}^{(3)}$ is replaced by $(d_{74}^{(3)} + d_{49}^{(3)})$

Iteration 5: $d_{91}^{(4)}$ is replaced by $(d_{95}^{(4)} + d_{51}^{(4)})$

Iteration 9: $d_{73}^{(6)}$ is replaced by $(d_{79}^{(6)} + d_{93}^{(6)})$

Once the Shortest distance is obtained in $d_{73}^{(6)}$. The value of this entry will not be altered in subsequent operations.

The algorithm described so far does not actually list the path, it only gives the shortest distances. Obtaining the path is slightly more involved than in previous Dijkstra's algorithm, because here, in this algorithm there are $n(n-1)$ paths required, not just one.
An efficient method of obtaining the intermediate vertices in each of the shortest paths is by constructing a matrix \( Z = [ Z_{ij} ] \) referred to as the optimal policy matrix, such that entry \( Z_{ij} \) is the first vertex from \( i \) along the shortest path from \( i \) to \( j \). The optimal policy matrix \( Z \) can be constructed as follows. Initially we set \( Z_{ij} = j \), if \( d_{ij} \neq 0 \),

\[
= 0, \text{ if } d_{ij} = \infty.
\]

In the \( k \)th iteration, if vertex \( K \) is inserted between \( i \) and \( j \), element \( Z_{ij} \) is replaced by the current value of \( Z_{ik} \), for all \( i \) and \( j \). Thus updating of the \( Z \) matrix is done during each iteration \( k \), where \( k = 1,2,\ldots,n \). At the end, the shortest path \( (i,v_1,v_2,\ldots,v_q,j) \) from \( i \) to \( j \) is derived as a sequence of vertex numbers from matrix \( Z \) as follows.

\[
v_1 = Z_{ij}, v_2 = Z_{v_1 j}, v_3 = Z_{v_2 j}, \ldots, j = Z_{v_q j}.
\]

For computational purpose, we need memory space for only one \( n \) by \( n \) matrix. Other constructed matrices can be overwritten on this matrix.

To estimate the execution time, note that we have to construct \( n \) matrices \( D_1, D_2, \ldots, D_n \) in sequential order. For each matrix \( D_k \), the number of elements to be computed is \((n-1)(n-2)\) because we already know that \( i \neq j, i \neq k, j \neq k \), although for simplicity in the Flow chart we have not taken advantage of this slight saving. Thus the execution time is proportional to \( n(n-1)(n-2) \approx n^3 \).

Whenever \( d_{ik}^{(k-1)} = \infty \), it is possible to circumvent \((n-1)\) additions and comparisons in exchange for an additional test.
Fig (2.2): Shortest path between every vertex-pair
2.3 ALGORITHM: SHORTEST PATH BETWEEN EVERY VERTEX-PAIR
/* This Algorithm takes n, the number of nodes in the Graph and the D, the
adjacency matrix of the graph as input. K,i and s,j are integers and dij is a record *

1. Read n, the number of nodes in the graph and D as the adjacency matrix.
2. K is First node or starting node is assigned 1.
3. \( i=1, dik < \infty \)
4. \( i = n, \text{state}[i].\text{predecessor} = n+1, =i+1 \)
5. Repeat through Step (6) for n = 1 to i
6. If \(( dik < \infty \) then \( s = dik + dkj \)
   \( \text{State}[i].\text{predecessor} = \text{state}[i].\text{length} \)
7. If(\( dkj < \infty \)) then \( s < dij \)
8. Repeat step(9) for \( j = 1 \) to \( n \)
9. if(\( \text{state}[i].\text{length} < \text{min} \)) then \( k \)
    then \( \text{state}(k)\).\text{length} = \text{state}(k)\).\text{length} + a[k][i] \)
10. \( k = 0 \) path \( (i) = k; j = j+1 \)
11. Repeat through step (13) for \( i = 1 \) to \( n \)
12. if ( \( i =0, i < \text{max}, i++ \)) then go to step(13)
13. if (\( \text{step}[i].\text{label} == 0 \)) \&\& (\( \text{state}[i].\text{length} < \text{min} \))
    then \( \text{state}[i].\text{length} = \text{state}(k)\).\text{length} + a[k][i] \)
14. \( a[k,i] = \text{min} \)
15. \( \text{min} = \infty \)
16. for \( i = 1 \) to \( n \) do step 8
17. \( dij = s \)
    \( a[k,i] = k; i = i + 1 \)
18. PRINT shortest path for \( j = 1 \) to \( n \) and also the tied distance path of \([i],\text{min}\)
2.4 PROGRAM

```c
#include <stdio.h>
typedef struct
{
    int predecessor;
    int length;
    int label;
} pi;
pi state[100];
int a[80][80], max1, max=1428;

main ( )
{
    int n, i, k, s, t, path[80], min;
    printf("ENTER THE NO OF NODES FOR SHORTEST PATH : \n");
    scanf("%d", &n);
    printf("READ THE MATRIX : \n");
    readln(n);
    printf("ENTER THE STARTING AND ENDING PATH'S : \n");
    scanf("%d %d", &s, &t);
    for(i=K, i<n4++)
    {
        state[i].predecessor = n+1;
        state[i].label = 0;
        state[i].length = max;
    }
    state[t].length = 0; state[t].label = 1;
    k=t;
    while(k!=s)
    {
        for(i=0; i<n; i++)
        {
            if ( ( a[k][i] != 0) && (state[i].label = = 0))
                if ( ( state[k].length+a[k][i] < state[i].length) )
                    state[i].predecessor = k;
                    state[i].length = state[k].length+a[k][i];
        }
    }
    k=0; min=max;
    for(i=0; i<n; i++)
    {
        if ((state[i].lable = = 0) && (state[i].length < min))
        {
            min = state[i].length;
            k = i;
        }
    }
    state[k].lable = 1;
}
k=s; i=0;
```
do
{
    path[i]=k;
    k=state[k].predecessor;
    i++;
}
while(k!=n+1);
max1=i;
printf("SHORTEST PATH OF THE GIVEN ADJACENT MATRIX \
");
for(i=0;i<max1;i++)
{
    printf(" %d",path[i]);
}
printf(" \
");
}
readln(n)
int n
{
    int i,j;
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    return;
}
2.5 Demonstration with an example

Consider the following Directed Weighted Graph

For this Graph, the input is in the form of Weighted Adjacency Matrix

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>5</td>
<td>8</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>3</td>
<td>∞</td>
</tr>
<tr>
<td>B</td>
<td>∞</td>
<td>0</td>
<td>10</td>
<td>7</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
<td>2</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>D</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
<td>∞</td>
<td>5</td>
<td>∞</td>
<td>∞</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>2</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>F</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>6</td>
<td>0</td>
<td>∞</td>
<td>5</td>
<td>∞</td>
</tr>
<tr>
<td>G</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>H</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>I</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>4</td>
<td>∞</td>
<td>0</td>
</tr>
</tbody>
</table>
Here the shortest-path is in the form of Adjacency matrix weighted directed graph is Now we consider, shortest-path form vertex A to all vertices E i.e (A → E)

1) 

\[ \text{Shortest-path cost} = 6 \]
\[ \text{Root is } A \rightarrow H \rightarrow G \rightarrow E \]

2) 

\[ \text{Shortest-path cost} = 21 \]
\[ \text{Root is } A \rightarrow C \rightarrow D \rightarrow F \rightarrow E \]

3) 

\[ \text{Shortest-path cost} = 28 \]
\[ \text{Root is } A \rightarrow B \rightarrow C \rightarrow D \rightarrow F \rightarrow E \]
Shortest-path cost = 20
Root is A, B, H, I, F, E

Shortest-path cost = 16
Root is A, B, I, F, E

Shortest-path cost = 26
Root is A, B, D, I, F, E
Shortest-path cost = 15
Root is A → H → G → F → E

Shortest-path cost = 21
Root is A → H → G → D → F → E

Shortest-Path from Vertex to all Vertices so cost's are \{6, 21, 28, 20, 16, 26, 15, 21\}
Now we will consider, least cost is the cost of the Shortest-path from Vertex to Vertex
so cost is 6
2.6 Result

ENTER THE NO OF NODES FOR SHORTEST PATH: 7
ENTER THE STARTING AND ENDING PATH'S : B G
GIVEN ADJACENCY MATRIX :

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>K</td>
<td>K</td>
<td>8</td>
<td>2</td>
<td>K</td>
<td>K</td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>K</td>
<td>K</td>
<td>K</td>
<td>K</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>K</td>
<td>0</td>
<td>K</td>
<td>K</td>
<td>3</td>
<td>K</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>K</td>
<td>K</td>
<td>0</td>
<td>1</td>
<td>K</td>
<td>K</td>
</tr>
<tr>
<td>E</td>
<td>K</td>
<td>K</td>
<td>K</td>
<td>K</td>
<td>0</td>
<td>K</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>K</td>
<td>K</td>
<td>K</td>
<td>10</td>
<td>4</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>G</td>
<td>K</td>
<td>K</td>
<td>2</td>
<td>K</td>
<td>K</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

SHORTEST PATH OF THE GIVEN ADJACENT MATRIX : B C F G
DISTANCE = 11