Experiment No. 8
To implement All pair shortest Path Algorithm (Floyd Warshall Algorithm)
Name : Vaidehi D. Gadag
Branch/Div.: Comps-1 (C47)
Date of Performance: 04/04/2024
Date of Submission: 11//04/2024

Experiment No. 8

Title: All Pair Shortest Path

Aim: To study and implement All Pair Shortest Path Algorithm

Objective: To introduce dynamic programming-based algorithm

Theory: The Floyd-Warshall algorithm is a graph algorithm that is deployed to find the shortest path between all the vertices present in a weighted graph. This algorithm is different from other shortest path algorithms; to describe it simply, this algorithm uses each vertex in the graph as a pivot to check if it provides the shortest way to travel from one point to another.

Floyd-Warshall algorithm is one of the methods in All-pairs shortest path algorithms and it is solved using the Adjacency Matrix representation of graphs.

Floyd-Warshall Algorithm

Consider a graph, **G = {V, E}** where **V** is the set of all vertices present in the graph and E is the set of all the edges in the graph. The graph, **G**, is represented in the form of an adjacency matrix, **A**, that contains all the weights of every edge connecting two vertices.

Algorithm:

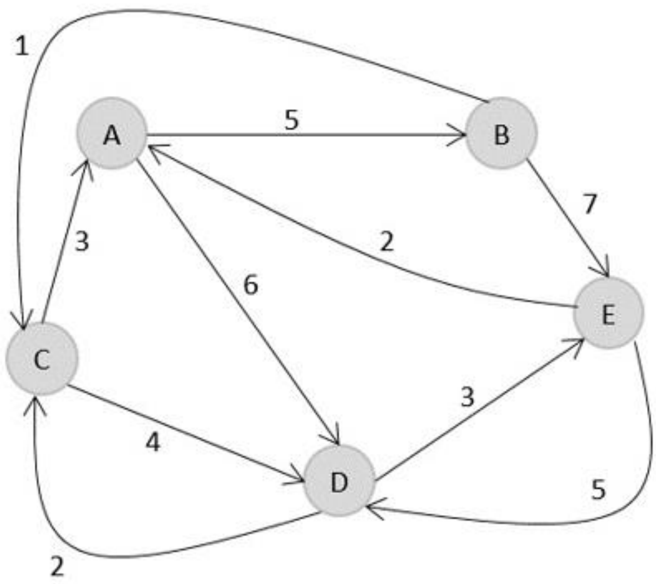
1. Construct an adjacency matrix **A** with all the costs of edges present in the graph. If there is no path between two vertices, mark the value as ∞.
2. Derive another adjacency matrix $\mathbf{A_1}$ from **A** keeping the first row and first column of the original adjacency matrix intact in $\mathbf{A_1}$. And for the remaining values, say $\mathbf{A_1[i,j]}$, if **A[i,j]>A[i,k]+A[k,j]** then replace $\mathbf{A_1[i,j]}$ with **A[i,k]+A[k,j]**. Otherwise, do not change the values. Here, in this step, **k = 1** (first vertex acting as pivot).
3. Repeat **Step 2** for all the vertices in the graph by changing the **k** value for every pivot vertex until the final matrix is achieved.
4. The final adjacency matrix obtained is the final solution with all the shortest paths.

Pseudocode:

```
Floyd-Warshall(w, n){ // w: weights, n: number of vertices
  for i = 1 to n do // initialize, D (0) = [wij]
    for j = 1 to n do{
      d[i, j] = w[i, j];
    }
    for k = 1 to n do // Compute D (k) from D (k-1)
      for i = 1 to n do
        for j = 1 to n do
          if (d[i, k] + d[k, j] < d[i, j]){
            d[i, j] = d[i, k] + d[k, j];
          }
    return d[1..n, 1..n];
}
```

Example:

Consider the following directed weighted graph **G = {V, E}**. Find the shortest paths between all the vertices of the graphs using the Floyd-Warshall algorithm.



Step 1: Construct an adjacency matrix **A** with all the distances as values.

$$A = \begin{matrix} 0 & 5 & \infty & 6 & \infty \\ \infty & 0 & 1 & \infty & 7 \\ 3 & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 2 & \infty & \infty & 5 & 0 \end{matrix}$$

Step 2: Considering the above adjacency matrix as the input, derive another matrix A_0 by keeping only first rows and columns intact. Take **k = 1**, and replace all the other values by ***A[i,k]+A[k,j]***.

$$A = \begin{matrix} 0 & 5 & \infty & 6 & \infty \\ \infty & & & & \\ 3 & & & & \\ \infty & & & & \\ 2 & & & & \end{matrix}$$

$$A_1 = \begin{matrix} 0 & 5 & \infty & 6 & \infty \\ \infty & 0 & 1 & \infty & 7 \\ 3 & 8 & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 2 & 7 & \infty & 5 & 0 \end{matrix}$$

Step 3:

Considering the above adjacency matrix as the input, derive another matrix A_0 by keeping only first rows and columns intact. Take **k = 1**, and replace all the other values by ***A[i,k]+A[k,j]***.

$$A_2 = \begin{array}{ccccc} & 5 & & & \\ \infty & 0 & 1 & \infty & 7 \\ & 8 & & & \\ & \infty & & & \\ & 7 & & & \end{array}$$

$$A_2 = \begin{array}{ccccc} 0 & 5 & 6 & 6 & 12 \\ \infty & 0 & 1 & \infty & 7 \\ 3 & 8 & 0 & 4 & 15 \\ \infty & \infty & 2 & 0 & 3 \\ 2 & 7 & 8 & 5 & 0 \end{array}$$

Step 4: Considering the above adjacency matrix as the input, derive another matrix $\boldsymbol{A_0}$ by keeping only first rows and columns intact. Take **k = 1**, and replace all the other values by ***A[i,k]+A[k,j]***.

$$A_3 = \begin{array}{ccccc} & & 6 & & \\ & & 1 & & \\ 3 & 8 & 0 & 4 & 15 \\ & & 2 & & \\ & & 8 & & \end{array}$$

$$A_3 = \begin{array}{ccccc} 0 & 5 & 6 & 6 & 12 \\ 4 & 0 & 1 & 5 & 7 \\ 3 & 8 & 0 & 4 & 15 \\ 5 & 10 & 2 & 0 & 3 \\ 2 & 7 & 8 & 5 & 0 \end{array}$$

Step 5: Considering the above adjacency matrix as the input, derive another matrix $\boldsymbol{A_0}$ by keeping only first rows and columns intact. Take **k = 1**, and replace all the other values by ***A[i,k]+A[k,j]***.

$$
\begin{array}{ccccc}
 & & & 6 & \\
 & & & 5 & \\
A_4 = & & & 4 & \\
5 & 10 & 2 & 0 & 3 \\
 & & & 5 &
\end{array}
$$

$$
A_4 = \begin{array}{ccccc}
0 & 5 & 6 & 6 & 9 \\
4 & 0 & 1 & 5 & 7 \\
3 & 8 & 0 & 4 & 7 \\
5 & 10 & 2 & 0 & 3 \\
2 & 7 & 7 & 5 & 0
\end{array}
$$

Step 6: Considering the above adjacency matrix as the input, derive another matrix ***A₀*** by keeping only first rows and columns intact. Take **k = 1**, and replace all the other values by ***A[i,k]+A[k,j]***.

$$
\begin{array}{ccccc}
 & & & & 9 \\
 & & & & 7 \\
A_5 = & & & & 7 \\
 & & & & 3 \\
2 & 7 & 7 & 5 & 0
\end{array}
$$

$$
A_5 = \begin{array}{ccccc}
0 & 5 & 6 & 6 & 9 \\
4 & 0 & 1 & 5 & 7 \\
3 & 8 & 0 & 4 & 7 \\
5 & 10 & 2 & 0 & 3 \\
2 & 7 & 7 & 5 & 0
\end{array}
$$

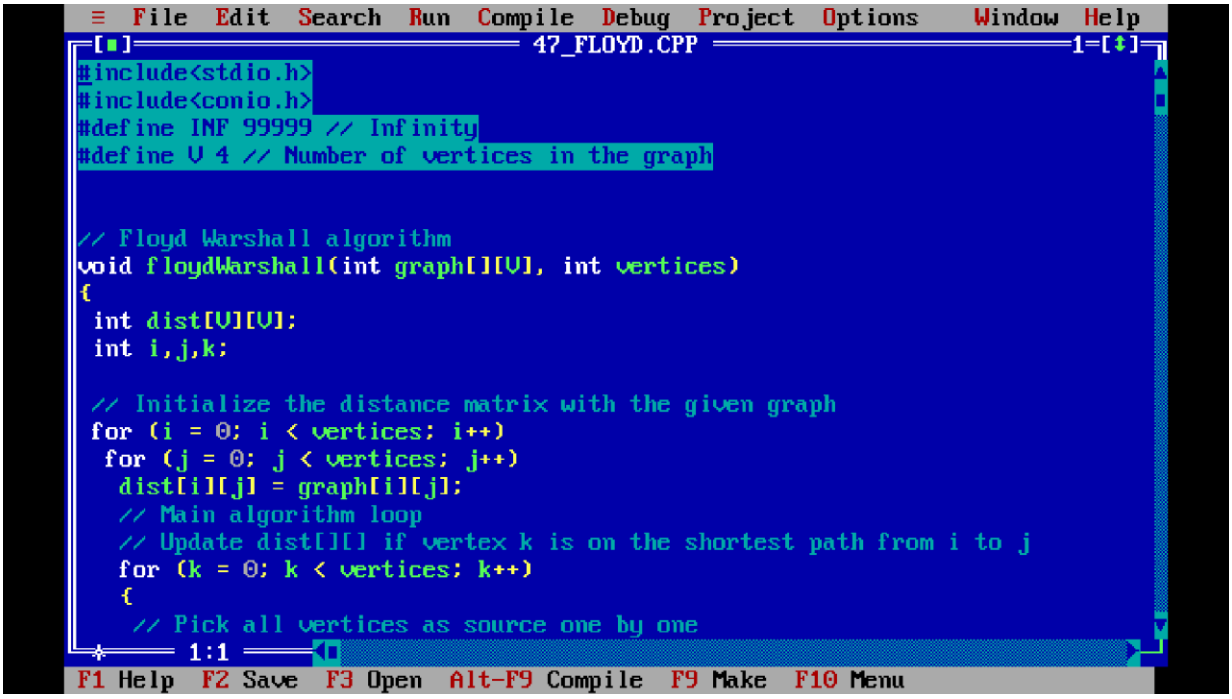
Time Complexity Analysis:

The algorithm uses three for loops to find the shortest distance between all pairs of vertices within a graph. Therefore, the **time complexity** is **$O(n^3)$**, where 'n' is the number of vertices in the graph. The **space complexity** of the algorithm is **$O(n^2)$**.

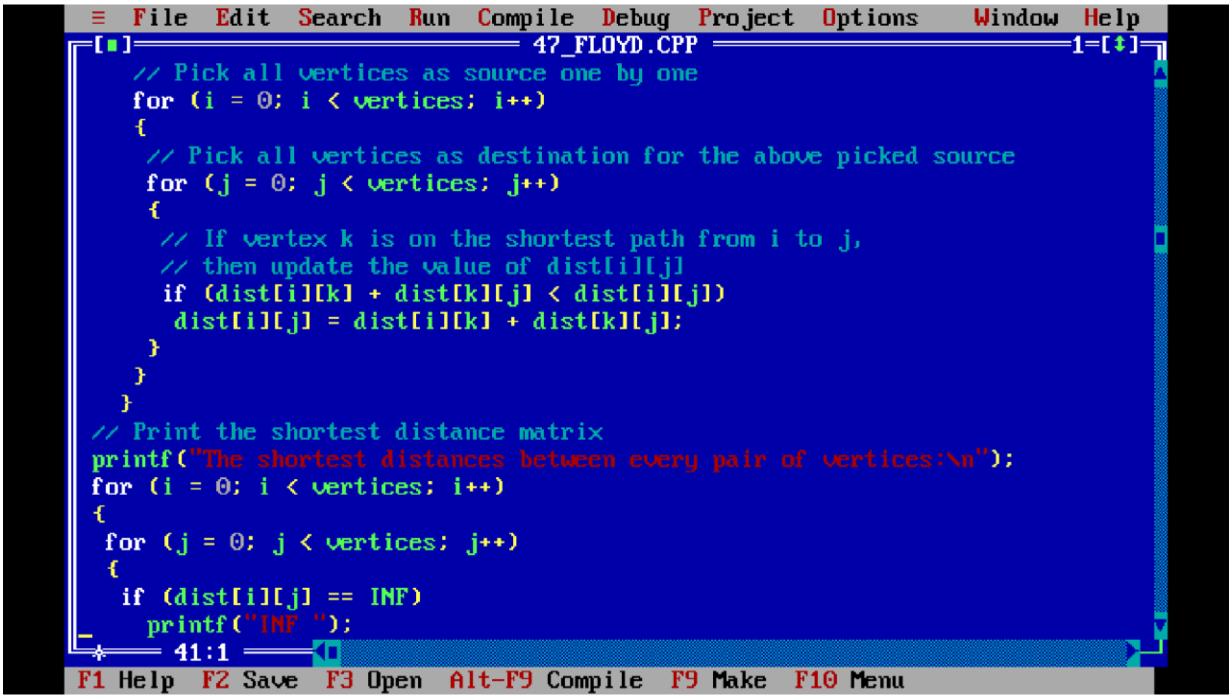
Program:

```
#include<stdio.h>
#include<conio.h>
#define INF 99999 // Infinity
#define V 4 // Number of vertices in the graph

// Floyd Warshall algorithm
void floydWarshall(int graph[][V], int vertices)
{
 int dist[V][V];
 int i,j,k;

 // Initialize the distance matrix with the given graph
 for (i = 0; i < vertices; i++)
  for (j = 0; j < vertices; j++)
   dist[i][j] = graph[i][j];
   // Main algorithm loop
   // Update dist[][] if vertex k is on the shortest path from i to j
   for (k = 0; k < vertices; k++)
   {
    // Pick all vertices as source one by one
    for (i = 0; i < vertices; i++)
    {
     // Pick all vertices as destination for the above picked source
     for (j = 0; j < vertices; j++)
     {
      // If vertex k is on the shortest path from i to j,
      // then update the value of dist[i][j]
      if (dist[i][k] + dist[k][j] < dist[i][j])
       dist[i][j] = dist[i][k] + dist[k][j];
     }
    }
   }
 // Print the shortest distance matrix
 printf("The shortest distances between every pair of vertices:\n");
 for (i = 0; i < vertices; i++)
 {
```

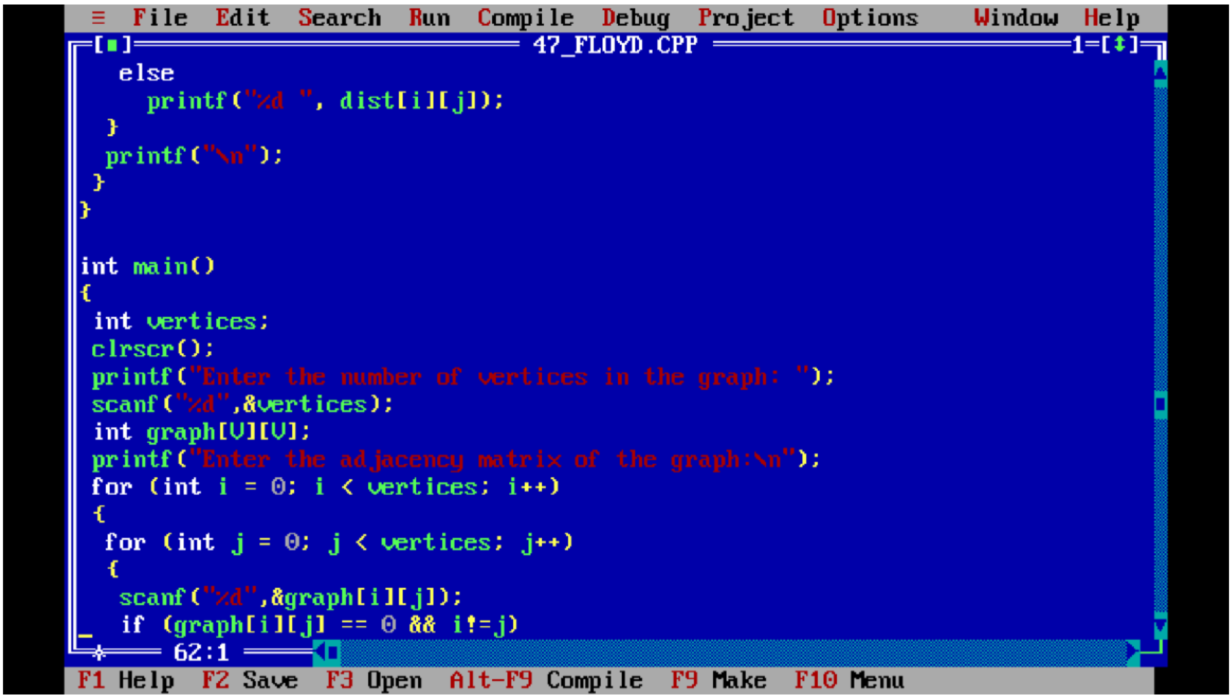
```
  for (j = 0; j < vertices; j++)
  {
   if (dist[i][j] == INF)
     printf("INF ");
   else
     printf("%d ", dist[i][j]);
  }
  printf("\n");
 }
}

int main()
{
 int vertices;
 clrscr();
 printf("Enter the number of vertices in the graph: ");
 scanf("%d",&vertices);
 int graph[V][V];
 printf("Enter the adjacency matrix of the graph:\n");
 for (int i = 0; i < vertices; i++)
 {
  for (int j = 0; j < vertices; j++)
  {
   scanf("%d",&graph[i][j]);
   if (graph[i][j] == 0 && i!=j)
    graph[i][j] = INF;
  }
 }
 // Run Floyd Warshall algorithm
 floydWarshall(graph, vertices);
 getch();
 return 0;
}
```

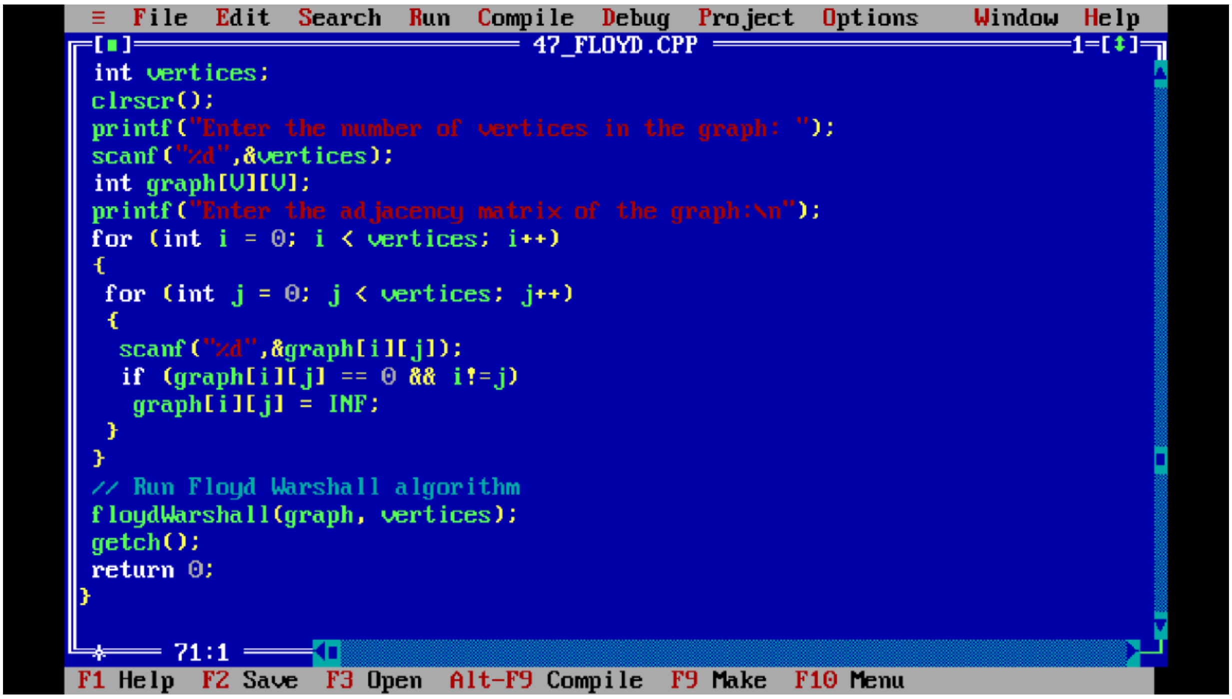


```
≡ File Edit Search Run Compile Debug Project Options Window Help
[■] 47_FLOYD.CPP 1=[↕]
#include<stdio.h>
#include<conio.h>
#define INF 99999 // Infinity
#define V 4 // Number of vertices in the graph


// Floyd Warshall algorithm
void floydWarshall(int graph[][V], int vertices)
{
 int dist[V][V];
 int i,j,k;

 // Initialize the distance matrix with the given graph
 for (i = 0; i < vertices; i++)
  for (j = 0; j < vertices; j++)
   dist[i][j] = graph[i][j];
   // Main algorithm loop
   // Update dist[][] if vertex k is on the shortest path from i to j
   for (k = 0; k < vertices; k++)
   {
    // Pick all vertices as source one by one
1:1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu
```



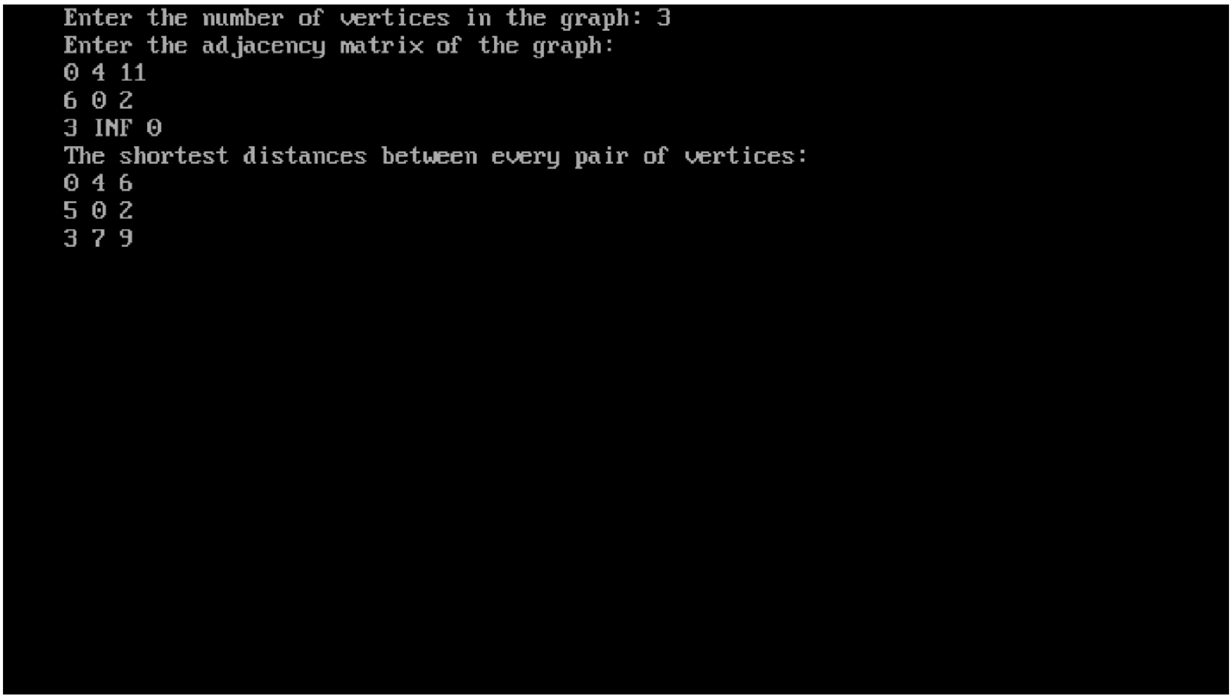
```
≡ File Edit Search Run Compile Debug Project Options Window Help
[■] 47_FLOYD.CPP 1=[↕]
    // Pick all vertices as source one by one
    for (i = 0; i < vertices; i++)
    {
     // Pick all vertices as destination for the above picked source
     for (j = 0; j < vertices; j++)
     {
      // If vertex k is on the shortest path from i to j,
      // then update the value of dist[i][j]
      if (dist[i][k] + dist[k][j] < dist[i][j])
       dist[i][j] = dist[i][k] + dist[k][j];
     }
    }
   }
 // Print the shortest distance matrix
 printf("The shortest distances between every pair of vertices:\n");
 for (i = 0; i < vertices; i++)
 {
  for (j = 0; j < vertices; j++)
  {
   if (dist[i][j] == INF)
    printf("INF ");
41:1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu
```


≡ File Edit Search Run Compile Debug Project Options Window Help
[■] 47_FLOYD.CPP 1=[↕]
else
printf("%d ", dist[i][j]);
}
printf("\n");
}
}
int main()
{
int vertices;
clrscr();
printf("Enter the number of vertices in the graph: ");
scanf("%d",&vertices);
int graph[V][V];
printf("Enter the adjacency matrix of the graph:\n");
for (int i = 0; i < vertices; i++)
{
for (int j = 0; j < vertices; j++)
{
scanf("%d",&graph[i][j]);
if (graph[i][j] == 0 && i!=j)
62:1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu


≡ File Edit Search Run Compile Debug Project Options Window Help
[■] 47_FLOYD.CPP 1=[↕]
int vertices;
clrscr();
printf("Enter the number of vertices in the graph: ");
scanf("%d",&vertices);
int graph[V][V];
printf("Enter the adjacency matrix of the graph:\n");
for (int i = 0; i < vertices; i++)
{
for (int j = 0; j < vertices; j++)
{
scanf("%d",&graph[i][j]);
if (graph[i][j] == 0 && i!=j)
graph[i][j] = INF;
}
}
// Run Floyd Warshall algorithm
floydWarshall(graph, vertices);
getch();
return 0;
}
71:1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

Output:

Enter the number of vertices in the graph: 3
Enter the adjacency matrix of the graph:
0 4 11
6 0 2
3 INF 0
The shortest distance between every pair of vertices:
0 4 6
5 0 2
3 7 9



```
Enter the number of vertices in the graph: 3
Enter the adjacency matrix of the graph:
0 4 11
6 0 2
3 INF 0
The shortest distances between every pair of vertices:
0 4 6
5 0 2
3 7 9
```

Enter the number of vertices in the graph: 4
Enter the adjacency matrix of the graph:
0 3 6 8
3 0 2 4
6 2 0 1
8 4 1 0
The shortest distance between every pair of vertices:
0 3 5 6
3 0 2 3
5 2 0 1
6 3 1 0

```
Enter the number of vertices in the graph: 4
Enter the adjacency matrix of the graph:

0 3 6 8
3 0 2 4
6 2 0 1
8 4 1 0
The shortest distances between every pair of vertices:
0 3 5 6
3 0 2 3
5 2 0 1
6 3 1 0
_
```

Conclusion:

In conclusion, the Floyd-Warshall algorithm is an efficient and reliable method for finding the shortest path between all vertices in a weighted graph. Its use of dynamic programming and the adjacency matrix representation of graphs allows it to compute all pairwise shortest paths in $O(n^3)$ time, making it suitable for dense graphs. Overall, the Floyd-Warshall algorithm is a valuable tool in graph theory and network analysis, providing a foundation for understanding and optimizing complex systems.