```
/** ---------------------------------------------------------------

                    TRANSITIVE CLOSURE OF A GRAPH

Given a directed graph, find out if a vertex j is reachable from another vertex
i for all vertex pairs (i, j) in the given graph. Being reachable mean that
there is a path from vertex i to j. The reachability matrix is called
transitive closure of a graph.

The all-pairs shortest path problem is the determination of the shortest graph
distances between every pair of vertices in a given graph. The problem can be
solved using applications of Dijkstra's algorithm or all at once using the
Floyd-Warshall algorithm.

The Floyd–Warshall algorithm is an algorithm for finding shortest paths in a
weighted graph with positive or negative edge weights (but with no negative
cycles).


                      FLYOD-WARSHALL ALGORITHM

The Floyd–Warshall algorithm is an algorithm for finding shortest paths in a
weighted graph with positive or negative edge weights (but with no negative
cycles). A single execution of the algorithm will find the lengths (summed
weights) of the shortest paths between all pairs of vertices.

The Floyd–Warshall algorithm is an example of dynamic programming, and was
published in its currently recognized form by Robert Floyd in 1962. However, it
is essentially the same as algorithms previously published by Bernard Roy in
1959 and also by Stephen Warshall in 1962 for finding the transitive closure of
a graph, and is closely related to Kleene's algorithm (published in 1956) for
converting a deterministic finite automaton into a regular expression. The
modern formulation of the algorithm as three nested for-loops was first
described by Peter Ingerman, also in 1962.
The algorithm is also known as Floyd's algorithm, the Roy–Warshall algorithm,
the Roy–Floyd algorithm, or the WFI algorithm.


PATH RECONSTRUCTION using WFI Algorithm

The Floyd–Warshall algorithm typically only provides the lengths of the paths
between all pairs of vertices. With simple modifications, it is possible to
create a method to reconstruct the actual path between any two endpoint
vertices. While one may be inclined to store the actual path from each vertex
to each other vertex, this is not necessary, and in fact, is very costly in
terms of memory. Instead, the shortest-path tree can be calculated for each
node in Theta(|E]) time using Theta(|V]) memory to store each tree which allows
us to efficiently reconstruct a path from any two connected vertices.

Let,
    dist[][] be a |V| x |V| array of minimum distances initialized to INF
    next[][] be a |V| x |V| array of vertex indices initialized to NULL

procedure FloydWarshallWithPathReconstruction(G)
    for each edge (u,v)
        dist[u][v] ← w(u,v)  // the weight of the edge (u,v)
        next[u][v] ← v
```

```
    for k from 1 to |V| // standard Floyd-Warshall implementation
        for i from 1 to |V|
            for j from 1 to |V|
                if dist[i][j] > dist[i][k] + dist[k][j] then
                    dist[i][j] ← dist[i][k] + dist[k][j]
                    next[i][j] ← next[i][k]

procedure Path(u, v)
   if next[u][v] = NULL then
       return []

   path = [u]

   while u ≠ v
       u ← next[u][v]
       path.append(u)

   return path

-------------------------------------------------------------------- **/

#include <stdio.h>

#define NF 786
#define MX 10

void showMatrix(int graph[][MX], int vertices, const char *text){
    int i, j;

    printf("\n\t%s is....\n", text);
    printf("\t\t   u|v |");
    for(i=0; i < vertices; i++)
        printf("%4d ", i);
    printf("\n");

    printf("\t\t-------");
    for(i=0; i < vertices; i++)
        printf("-----");
    printf("\n");

    for(i=0; i < vertices; i++){
        printf("\t\t%5d  |", i);
        for(j = 0; j < vertices; j++)
            printf("%4d ", graph[i][j]);
        printf("\n");
    }
    //printf("\n");

}


void tcFloydWarshall (int graph[][MX], int vertices){
    int dist[MX][MX], i, j, k;
    int next[MX][MX];

    for (i = 0; i < vertices; i++){
        for (j = 0; j < vertices; j++){
            dist[i][j] = graph[i][j];
```

```c
                next[i][j] = j;
            }
        }

        printf("\nAt Initialization ...");
        showMatrix(dist, vertices, "Cost Matrix");
        showMatrix(next, vertices, "Path Matrix");

        for (k = 0; k < vertices; k++){
            printf("\nHit Enter key to Proceed ...");
            getc(stdin);
            getc(stdin);

            printf("ITERATION %2d\n", k+1);
            for (i = 0; i < vertices; i++){
                for (j = 0; j < vertices; j++){
                    if (dist[i][j] > dist[i][k] + dist[k][j]){
                        dist[i][j] = dist[i][k] + dist[k][j];
                        next[i][j] = next[i][k];
                    }
                }
            }
            sleep(3);

            showMatrix(dist, vertices, "Cost Matrix");
            showMatrix(next, vertices, "Path Matrix");
        }
    }


    void initMatrix(int graph[][MX]){
        int i, j, weight;
        for(i=0; i < MX; i++)
            for(j = 0; j < MX; j++)
                graph[i][j] = NF;
    }


    int createGraph(int graph[][MX]){
        int i, j, vCnt=0, weight;
        int u, v, vertices, type;

        printf("\n\tGraph Creation [Undirected/Directed]...\n");

        printf("\t\tType of Graph [0: UnDirected] := ");
        scanf("%d", &type);

        if(type != 0)
            type = 1;

        do{
            printf("\t\tHow Many Vertices [upto %d vertices]?? ", MX);
            scanf("%d", &vertices);
        }while(vertices < 1 || vertices > MX);

        printf("\n");
```

```c
    printf("\n\tVertices starts at 0 and terminates at %d\n", vertices-1);
    printf("\t\tVertex ID of -1 terminates Input\n");
    printf("\n\tEnter Existing Edges in the Graph\n\n");
    printf("\t\t------------------------------------------------------\n");
    printf("\t\tEdge#      'u'       'v'      Cost     Remark\n");
    printf("\t\t------------------------------------------------------\n");


    do{
        do{
            printf("\t\t   %2d         ", vCnt+1);
            scanf("%d%d%d", &u, &v, &weight );

            if(u == v && weight < 0)
                printf("\t\t\t\t\t\t\t\t\t\t\tNegative Cycle
\n");
        }while(u == v && (u != -1 && v != -1) && weight < 0);

        printf("  \t   ");

        if((u != -1 || v != -1) && u < vertices && v < vertices){
            if(graph[u][v] == NF){
                if(type)
                    graph[u][v] = weight;
                else
                    graph[u][v] = graph[v][u] = weight;

                printf("\t\t\t\t\t\t\t\t\t\tEdge Taken\n");

            } else
                printf("\t\t\t\t\t\t\t\t\t\tEdge Exists\n");

        }else
            printf("\t\t\t\t\t\t\t\t\t\tInvalid Edge\n");

        vCnt++;

    }while(u != -1 || v != -1);


    for(i = 0; i < MX; i++)
        if(graph[i][i] == NF)
            graph[i][i] = 0;

    return vertices;
}


int printPathWFI(int dist[][MX], int next[][MX], int u, int v){
    int pathCost = NF;


    return pathCost;
}



int main(){
```

```
    int graph[MX][MX];

    int vertices;
    initMatrix(graph);

    vertices = createGraph(graph);

    printf("\nGraph with %2d vertices ...\n", vertices);
    showMatrix(graph,  vertices, "Adjacency Matrix");
    tcFloydWarshall(graph,  vertices);


    return 0;
}


/** ----------------  E X E C U T I O N   T R A I L ---------------


    Graph Creation [Undirected/Directed]...
        Type of Graph [0: UnDirected] := 4
        How Many Vertices [upto 10 vertices]?? 4

    Vertices starts at 0 and terminates at 3
        Vertex ID of -1 terminates Input

    Enter Existing Edges in the Graph
        ------------------------------------------------------
        Edge#     'u'       'v'     Cost    Remark
        ------------------------------------------------------
           1       0         2        3
                                             Edge Taken
           2       2         2        1
                                             Edge Taken
           3       2         3        2
                                             Edge Taken
           4       2         1       -2
                                             Edge Taken
           5       3         2        4
                                             Edge Taken
           6       1         0        4
                                             Edge Taken
           7       1         3        3
                                             Edge Taken
           8      -1        -1       12
                                             Invalid Edge

    Graph with  4 vertices ...

        Adjacency Matrix is....
              u|v |   0    1    2    3
            --------------------------
               0  |   0  786    3  786
               1  |   4    0  786    3
               2  | 786   -2    1    2
               3  | 786  786    4    0

    At Initialization ...
```

```
        Cost Matrix is....
               u|v |   0    1    2    3
             ---------------------------
                 0  |   0  786    3  786
                 1  |   4    0  786    3
                 2  | 786   -2    1    2
                 3  | 786  786    4    0

        Path Matrix is....
               u|v |   0    1    2    3
             ---------------------------
                 0  |   0    1    2    3
                 1  |   0    1    2    3
                 2  |   0    1    2    3
                 3  |   0    1    2    3

    ITERATION  1
        Cost Matrix is....
               u|v |   0    1    2    3
             ---------------------------
                 0  |   0  786    3  786
                 1  |   4    0    7    3
                 2  | 786   -2    1    2
                 3  | 786  786    4    0

        Path Matrix is....
               u|v |   0    1    2    3
             ---------------------------
                 0  |   0    1    2    3
                 1  |   0    1    0    3
                 2  |   0    1    2    3
                 3  |   0    1    2    3

    ITERATION  2
        Cost Matrix is....
               u|v |   0    1    2    3
             ---------------------------
                 0  |   0  786    3  786
                 1  |   4    0    7    3
                 2  |   2   -2    1    1
                 3  | 786  786    4    0

        Path Matrix is....
               u|v |   0    1    2    3
             ---------------------------
                 0  |   0    1    2    3
                 1  |   0    1    0    3
                 2  |   1    1    2    1
                 3  |   0    1    2    3

    ITERATION  3
        Cost Matrix is....
               u|v |   0    1    2    3
             ---------------------------
                 0  |   0    1    3    4
                 1  |   4    0    7    3
                 2  |   2   -2    1    1
                 3  |   6    2    4    0
```

```
          Path Matrix is....
                 u|v |   0    1    2    3
              ---------------------------
                  0  |   0    2    2    2
                  1  |   0    1    0    3
                  2  |   1    1    2    1
                  3  |   2    2    2    3

      ITERATION  4
          Cost Matrix is....
                 u|v |   0    1    2    3
              ---------------------------
                  0  |   0    1    3    4
                  1  |   4    0    7    3
                  2  |   2   -2    1    1
                  3  |   6    2    4    0

          Path Matrix is....
                 u|v |   0    1    2    3
              ---------------------------
                  0  |   0    2    2    2
                  1  |   0    1    0    3
                  2  |   1    1    2    1
                  3  |   2    2    2    3

      aahika@DAB-PC:~/Desktop$

  **/
```