Floyd Warshall Algorithm | DP-16

Difficulty Level : Medium • Last Updated : 28 Apr, 2021

The Floyd Warshall Algorithm is for solving the All Pairs Shortest Path problem. The problem is to find shortest distances between every pair of vertices in a given edge weighted directed Graph.

Example:

```
Input:
       graph[][] = { {0,   5,  INF, 10},
                    {INF,  0,  3,  INF},
                    {INF, INF, 0,   1},
                    {INF, INF, INF, 0} }
which represents the following graph
             10
       (0)------->(3)
        |         /|\
      5 |          |
        |          | 1
       \|/         |
       (1)------->(2)
            3
```

Related Articles

```
      INF      0      3      4
      INF    INF      0      1
      INF    INF    INF      0
```

Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

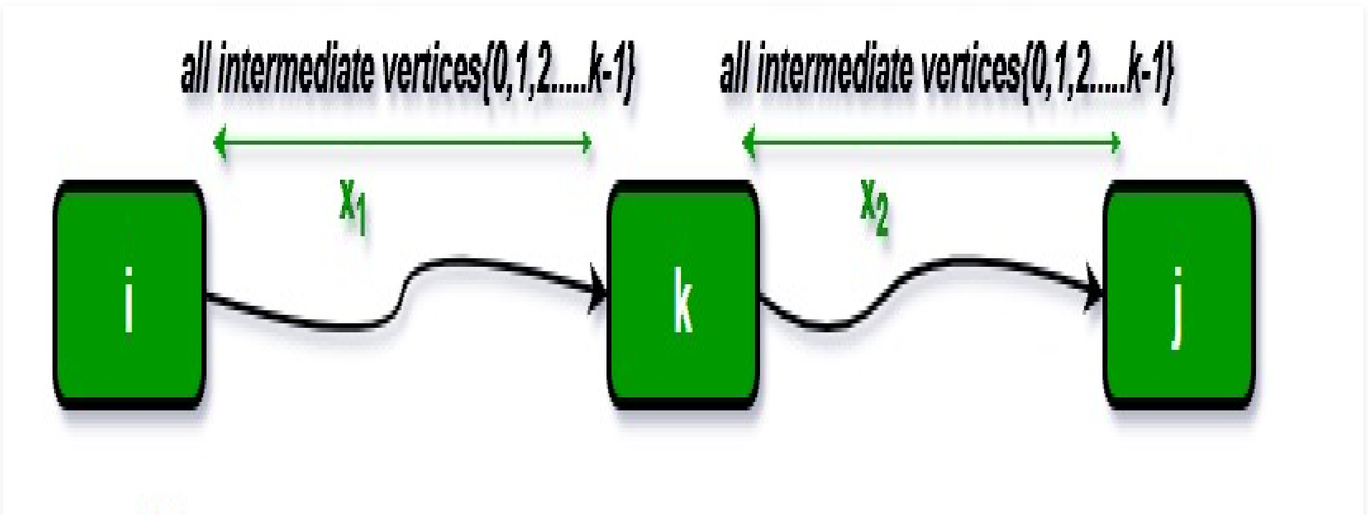
Floyd Warshall Algorithm

We initialize the solution matrix same as the input graph matrix as a first step. Then we update the solution matrix by considering all vertices as an intermediate vertex. The idea is to one by one pick all vertices and updates all shortest paths which include the picked vertex as an intermediate vertex in the shortest path. When we pick vertex number k as an intermediate vertex, we already have considered vertices {0, 1, 2, .. k-1} as intermediate vertices. For every pair (i, j) of the source and destination vertices respectively, there are two possible cases.

1) k is not an intermediate vertex in shortest path from i to j. We keep the value of dist[i][j] as it is.

2) k is an intermediate vertex in shortest path from i to j. We update the value of dist[i][j] as dist[i][k] + dist[k][j] if dist[i][j] > dist[i][k] + dist[k][j]

The following figure shows the above optimal substructure property in the all-pairs shortest path problem.


all intermediate vertices{0,1,2.....k-1}
all intermediate vertices{0,1,2.....k-1}
x_1
x_2
i
k
j

Following is implementations of the Floyd Warshall algorithm.

C++

```
// C++ Program for Floyd Warshall Algorithm
#include <bits/stdc++.h>
using namespace std;

// Number of vertices in the graph
#define V 4

/* Define Infinite as a large enough
value.This value will be used for
vertices not connected to each other */
#define INF 99999

// A function to print the solution matrix
void printSolution(int dist[][V]);

// Solves the all-pairs shortest path
// problem using Floyd Warshall algorithm
void floydWarshall(int graph[][V])
{
    /* dist[][] will be the output matrix
    that will finally have the shortest
    distances between every pair of vertices */
    int dist[V][V], i, j, k;

    /* Initialize the solution matrix same
    as input graph matrix. Or we can say
    the initial values of shortest distances
    are based on shortest paths considering
    no intermediate vertex. */
    for (i = 0; i < V; i++)
        for (j = 0; j < V; j++)
            dist[i][j] = graph[i][j];

    /* Add all vertices one by one to
    the set of intermediate vertices.
    ---> Before start of an iteration,
    we have shortest distances between all
    pairs of vertices such that the
    shortest distances consider only the
    vertices in set {0, 1, 2, .. k-1} as
    intermediate vertices.
    ----> After the end of an iteration,
    for (k = 0; k < V; k++) {
        // Pick all vertices as source one by one
        for (i = 0; i < V; i++) {
            // Pick all vertices as destination for the
            // above picked source
            for (j = 0; j < V; j++) {
                // If vertex k is on the shortest path from
                // i to j, then update the value of
                // dist[i][j]
                if (dist[i][j] > (dist[i][k] + dist[k][j])
                    && (dist[k][j] != INF
                        && dist[i][k] != INF))
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }

    // Print the shortest distance matrix
    printSolution(dist);
}

/* A utility function to print solution */
void printSolution(int dist[][V])
{
    cout << "The following matrix shows the shortest "
            "distances"
            " between every pair of vertices \n";
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            if (dist[i][j] == INF)
                cout << "INF"
                     << "     ";
            else
                cout << dist[i][j] << "     ";
        }
        cout << endl;
    }
}

// Driver code
int main()
{
    /* Let us create the following weighted graph
            10
    (0)------->(3)
        |     /|\
    5 |     |
    int graph[V][V] = { { 0, 5, INF, 10 },
                        { INF, 0, 3, INF },
                        { INF, INF, 0, 1 },
                        { INF, INF, INF, 0 } };

    // Print the solution
    floydWarshall(graph);
    return 0;
}

// This code is contributed by Mythri J L
```

C

```
// C Program for Floyd Warshall Algorithm
#include<stdio.h>

// Number of vertices in the graph
#define V 4

/* Define Infinite as a large enough
  value. This value will be used
  for vertices not connected to each other */
#define INF 99999

// A function to print the solution matrix
void printSolution(int dist[][V]);

// Solves the all-pairs shortest path
// problem using Floyd Warshall algorithm
void floydWarshall (int graph[][V])
{
    /* dist[][] will be the output matrix
      that will finally have the shortest
      distances between every pair of vertices */
    int dist[V][V], i, j, k;

    /* Initialize the solution matrix
      same as input graph matrix. Or
       we can say the initial values of
       shortest distances are based
       on shortest paths considering no
       intermediate vertex. */
    for (i = 0; i < V; i++)
        for (j = 0; j < V; j++)
      ---> Before start of an iteration, we
      have shortest distances between all
      pairs of vertices such that the shortest
      distances consider only the
      vertices in set {0, 1, 2, .. k-1} as
      intermediate vertices.
      ----> After the end of an iteration,
      vertex no. k is added to the set of
      intermediate vertices and the set
      becomes {0, 1, 2, .. k} */
    for (k = 0; k < V; k++)
    {
        // Pick all vertices as source one by one
        for (i = 0; i < V; i++)
        {
            // Pick all vertices as destination for the
            // above picked source
            for (j = 0; j < V; 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
    printSolution(dist);
}

/* A utility function to print solution */
void printSolution(int dist[][V])
{
    printf ("The following matrix shows the shortest distances"
            " between every pair of vertices \n");
    for (int i = 0; i < V; i++)
    {
        for (int j = 0; j < V; j++)
        {
            if (dist[i][j] == INF)
                printf("%7s", "INF");
            else
                printf ("%7d", dist[i][j]);
        }
        printf("\n");
    }
```

```
{
    /* Let us create the following weighted graph
            10
       (0)------->(3)
        |         /|\
      5 |          |
        |          | 1
       \|/         |
       (1)------->(2)
            3           */
    int graph[V][V] = { {0,   5,  INF, 10},
                        {INF, 0,   3, INF},
                        {INF, INF, 0,   1},
                        {INF, INF, INF, 0}
                      };

    // Print the solution
    floydWarshall(graph);
    return 0;
}
```

Java

```
// A Java program for Floyd Warshall All Pairs Shortest
// Path algorithm.
import java.util.*;
import java.lang.*;
import java.io.*;


class AllPairShortestPath
{
    final static int INF = 99999, V = 4;

    void floydWarshall(int graph[][])
    {
        int dist[][] = new int[V][V];
        int i, j, k;

        /* Initialize the solution matrix
           same as input graph matrix.
           Or we can say the initial values
           of shortest distances
           are based on shortest paths
           considering no intermediate
        /* Add all vertices one by one
           to the set of intermediate
           vertices.
          ---> Before start of an iteration,
               we have shortest
               distances between all pairs
               of vertices such that
               the shortest distances consider
               only the vertices in
               set {0, 1, 2, .. k-1} as
               intermediate vertices.
          ----> After the end of an iteration,
                vertex no. k is added
                to the set of intermediate
                vertices and the set
                becomes {0, 1, 2, .. k} */
        for (k = 0; k < V; k++)
        {
            // Pick all vertices as source one by one
            for (i = 0; i < V; i++)
            {
                // Pick all vertices as destination for the
                // above picked source
                for (j = 0; j < V; 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
        printSolution(dist);
    }

    void printSolution(int dist[][])
    {
        System.out.println("The following matrix shows the shortest "+
                         "distances between every pair of vertices");
        for (int i=0; i<V; ++i)
        {
            for (int j=0; j<V; ++j)
            {
                if (dist[i][j]==INF)
            System.out.println();
        }
    }

    // Driver program to test above function
    public static void main (String[] args)
    {
        /* Let us create the following weighted graph
           10
        (0)------->(3)
        |         /|\
        5 |          |
        |           | 1
        \|/         |
        (1)------->(2)
           3           */
        int graph[][] = { {0,   5,  INF, 10},
                          {INF, 0,   3, INF},
                          {INF, INF, 0,   1},
                          {INF, INF, INF, 0}
                        };
        AllPairShortestPath a = new AllPairShortestPath();

        // Print the solution
        a.floydWarshall(graph);
    }
}

// Contributed by Aakash Hasija
```

Python

```
# Python Program for Floyd Warshall Algorithm

# Number of vertices in the graph
V = 4

# Define infinity as the large
# enough value. This value will be
# used for vertices not connected to each other
INF = 99999

# Solves all pair shortest path
# via Floyd Warshall Algorithm
```

```
        have the shortest distances
        between every pair of vertices """
    """ initializing the solution matrix
    same as input graph matrix
    OR we can say that the initial
    values of shortest distances
    are based on shortest paths considering no
    intermediate vertices """

    dist = list(map(lambda i: list(map(lambda j: j, i)), graph))

    """ Add all vertices one by one
    to the set of intermediate
     vertices.
     ---> Before start of an iteration,
     we have shortest distances
     between all pairs of vertices
     such that the shortest
     distances consider only the
     vertices in the set
    {0, 1, 2, .. k-1} as intermediate vertices.
      ----> After the end of a
      iteration, vertex no. k is
     added to the set of intermediate
     vertices and the
    set becomes {0, 1, 2, .. k}
    """
    for k in range(V):

        # pick all vertices as source one by one
        for i in range(V):

            # Pick all vertices as destination for the
            # above picked source
            for j in range(V):

                # If vertex k is on the shortest path from
                # i to j, then update the value of dist[i][j]
                dist[i][j] = min(dist[i][j],
                                 dist[i][k] + dist[k][j]
                                 )
    printSolution(dist)


# A utility function to print the solution
def printSolution(dist):
    print "Following matrix shows the shortest distances\
                print "%7s" % ("INF"),
            else:
                print "%7d\t" % (dist[i][j]),
            if j == V-1:
                print ""


# Driver program to test the above program
# Let us create the following weighted graph
"""
            10
       (0)------->(3)
        |         /|\
      5 |          |
        |          | 1
       \|/         |
       (1)------->(2)
            3           """
graph = [[0, 5, INF, 10],
         [INF, 0, 3, INF],
         [INF, INF, 0,   1],
         [INF, INF, INF, 0]
         ]
# Print the solution
floydWarshall(graph)
# This code is contributed by Mythri J L
```

C#

```
// A C# program for Floyd Warshall All
// Pairs Shortest Path algorithm.

using System;

public class AllPairShortestPath
{
    readonly static int INF = 99999, V = 4;

    void floydWarshall(int[,] graph)
    {
        int[,] dist = new int[V, V];
        int i, j, k;

        // Initialize the solution matrix
        // same as input graph matrix
        // vertex
        for (i = 0; i < V; i++) {
            for (j = 0; j < V; j++) {
                dist[i, j] = graph[i, j];
            }
        }

        /* Add all vertices one by one to
        the set of intermediate vertices.
        ---> Before start of a iteration,
             we have shortest distances
             between all pairs of vertices
             such that the shortest distances
             consider only the vertices in
             set {0, 1, 2, .. k-1} as
             intermediate vertices.
        ---> After the end of a iteration,
             vertex no. k is added
             to the set of intermediate
             vertices and the set
             becomes {0, 1, 2, .. k} */
        for (k = 0; k < V; k++)
        {
            // Pick all vertices as source
            // one by one
            for (i = 0; i < V; i++)
            {
                // Pick all vertices as destination
                // for the above picked source
                for (j = 0; j < V; 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
        printSolution(dist);
    }

    void printSolution(int[,] dist)
```

```
        {
            for (int j = 0; j < V; ++j)
            {
                if (dist[i, j] == INF) {
                    Console.Write("INF ");
                } else {
                    Console.Write(dist[i, j] + " ");
                }
            }

            Console.WriteLine();
        }
    }

    // Driver Code
    public static void Main(string[] args)
    {
        /* Let us create the following
           weighted graph
              10
        (0)------->(3)
        |         /|\
        5 |         |
        |         | 1
        \|/         |
        (1)------->(2)
             3           */
        int[,] graph = { {0, 5, INF, 10},
                        {INF, 0, 3, INF},
                        {INF, INF, 0, 1},
                        {INF, INF, INF, 0}
                        };

        AllPairShortestPath a = new AllPairShortestPath();

        // Print the solution
        a.floydWarshall(graph);
    }
}

// This article is contributed by
// Abdul Mateen Mohammed
```

PHP

```
// using Floyd Warshall algorithm
function floydWarshall ($graph, $V, $INF)
{
    /* dist[][] will be the output matrix
    that will finally have the shortest
    distances between every pair of vertices */
    $dist = array(array(0,0,0,0),
                  array(0,0,0,0),
                  array(0,0,0,0),
                  array(0,0,0,0));

    /* Initialize the solution matrix same
    as input graph matrix. Or we can say the
    initial values of shortest distances are
    based on shortest paths considering no
    intermediate vertex. */
    for ($i = 0; $i < $V; $i++)
        for ($j = 0; $j < $V; $j++)
            $dist[$i][$j] = $graph[$i][$j];

    /* Add all vertices one by one to the set
    of intermediate vertices.
    ---> Before start of an iteration, we have
    shortest distances between all pairs of
    vertices such that the shortest distances
    consider only the vertices in set
    {0, 1, 2, .. k-1} as intermediate vertices.
    ----> After the end of an iteration, vertex
    no. k is added to the set of intermediate
    vertices and the set becomes {0, 1, 2, .. k} */
    for ($k = 0; $k < $V; $k++)
    {
        // Pick all vertices as source one by one
        for ($i = 0; $i < $V; $i++)
        {
            // Pick all vertices as destination
            // for the above picked source
            for ($j = 0; $j < $V; $j++)
            {
                // If vertex k is on the shortest path from
                // i to j, then update the value of dist[i][j]
                if ($dist[$i][$k] + $dist[$k][$j] <
                                    $dist[$i][$j])
                    $dist[$i][$j] = $dist[$i][$k] +
                                    $dist[$k][$j];
            }
        }
}

/* A utility function to print solution */
function printSolution($dist, $V, $INF)
{
    echo "The following matrix shows the " .
             "shortest distances between " .
                "every pair of vertices \n";
    for ($i = 0; $i < $V; $i++)
    {
        for ($j = 0; $j < $V; $j++)
        {
            if ($dist[$i][$j] == $INF)
                echo "INF " ;
            else
                echo $dist[$i][$j], " ";
        }
        echo "\n";
    }
}

// Driver Code

// Number of vertices in the graph
$V = 4 ;

/* Define Infinite as a large enough
value. This value will be used for
vertices not connected to each other */
$INF = 99999 ;

/* Let us create the following weighted graph
        10
(0)------->(3)
    |     /|\
5 |      |
    |     | 1
\|/      |
(1)------->(2)
        3     */
$graph = array(array(0, 5, $INF, 10),
               array($INF, 0, 3, $INF),
               array($INF, $INF, 0, 1),
               array($INF, $INF, $INF, 0));

// Print the solution
floydWarshall($graph, $V, $INF);
```

Output:

```
Following matrix shows the shortest distances between every pair of verti
      0      5      8      9
    INF      0      3      4
    INF    INF      0      1
    INF    INF    INF      0
```

Time Complexity: O(V^3)
The above program only prints the shortest distances. We can modify the solution to print the shortest paths also by storing the predecessor information in a separate 2D matrix.
Also, the value of INF can be taken as INT_MAX from limits.h to make sure that we handle maximum possible value. When we take INF as INT_MAX, we need to change the if condition in the above program to avoid arithmetic overflow.

```
#include

#define INF INT_MAX
..........................
if ( dist[i][k] != INF &&
     dist[k][j] != INF &&
     dist[i][k] + dist[k][j] < dist[i][j]
    )
 dist[i][j] = dist[i][k] + dist[k][j];
...........................
```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Previous Next

RECOMMENDED ARTICLES

Page : 1 2 3

01 **Finding shortest path between any two nodes using Floyd Warshall Algorithm**
30, Jun 20

02 **Detecting negative cycle using Floyd Warshall**
12, Oct 17

03 **Comparison of Dijkstra's and Floyd–Warshall algorithms**
19, Oct 17

04 **Boruvka's algorithm for Minimum Spanning Tree**
29, Mar 11

05 **Push Relabel Algorithm | Set 1 (Introduction and Illustration)**
04, Apr 16

06 **Union-Find Algorithm | Set 2 (Union By Rank and Path Compression)**
28, Oct 12

07 **Kruskal's Minimum Spanning Tree Algorithm | Greedy Algo-2**
30, Oct 12

08 **Dijkstra's shortest path algorithm | Greedy Algo-7**
25, Nov 12

Article Contributed By :

GeeksforGeeks

Current difficulty : Medium

Easy Normal Medium Hard Expert

Improved By : Abdul Mateen Mohammed, abhayscholar1995, ankthon, rathbhupendra, mythri1020, mitrukahitesh, NikitaRana07

Article Tags : Samsung, Shortest Path, Dynamic Programming, Graph

Practice Tags : Samsung, Dynamic Programming, Graph, Shortest Path

Improve Article Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments