Dynamic Programming | Set 16 (Floyd Warshall Algorithm)

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
Note that the value of graph[i][j] is 0 if i is equal to j
And graph[i][j] is INF (infinite) if there is no edge from vertex i to j.

Output:
Shortest distance matrix
      0      5      8      9
    INF      0      3      4
    INF    INF      0      1
    INF    INF    INF      0
```

Floyd Warshall Algorithm

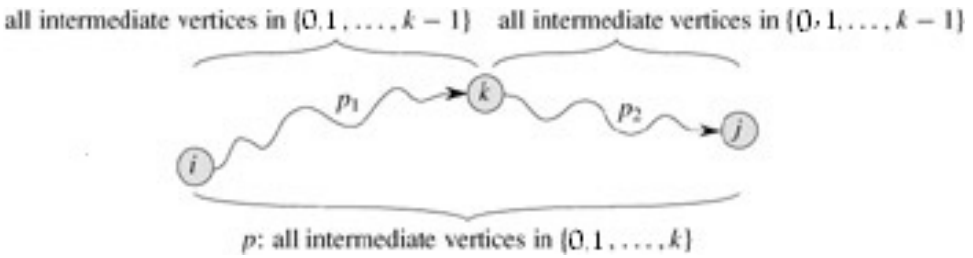
We initialize the solution matrix same as the input graph matrix as a first step. Then we update the solution matrix by considering all vertices as an intermediate vertex. The idea is to one by one pick all vertices and update all shortest paths which include the picked vertex as an intermediate vertex 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 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].

The following figure is taken from the Cormen book. It shows the above optimal substructure property in the all-pairs shortest path problem.



Following is C implementation of the Floyd Warshall algorithm.

```
// 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 floydWarshell (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 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);
}

/* A utility function to print solution */
void printSolution(int dist[][V])
{
    printf ("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");
    }
}

// driver program to test above function
int main()
{
    /* 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
    floydWarshell(graph);
    return 0;
}
```

Run on IDE

Output:

```
Following matrix shows the shortest distances between every pair of vertices
      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 arithmatic overflow.

```
#include<limits.h>

#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];
.........................
```

Run on IDE

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

53 Comments Category: Graph Tags: Dynamic Programming, Graph

Related Posts:

- Hopcroft–Karp Algorithm for Maximum Matching | Set 2 (Implementation)
- Hopcroft–Karp Algorithm for Maximum Matching | Set 1 (Introduction)
- Length of shortest chain to reach a target word
- Find same contacts in a list of contacts
- Karger's algorithm for Minimum Cut | Set 2 (Analysis and Applications)
- Steiner Tree Problem
- Karger's algorithm for Minimum Cut | Set 1 (Introduction and Implementation)
- Greedy Algorithms | Set 9 (Boruvka's algorithm)