Bansilal Ramnath Agarwal Charitable Trust's
Vishwakarma Institute of Information Technology

Department of
Artificial Intelligence and Data Science

Name: Siddhesh Dilip Khairnar		
Class: TY	Division: B	Roll No: 372028
Semester: V	Academic Year: 2023-2024	
Subject Name & Code: Design and Analysis of Algorithm: ADUA31202		
Title of Assignment: Implement All Pair Shortest paths problem using Floyd's Algorithm.		
Date of Performance: 16-09-2023	Date of Submission: 22-09-2023	

ASSIGNMENT NO. 4

Page No.
Date

Assignment no-4 DAA

Name : Siddhesh Dilip Khairnar
Division : B(B2)
PRN no : 22110398

* Title :→ Implement All pairs shortest Path Problem using Floyd Algorithm.

* Objective :→ To understand and implement the Floyd's warshall algorithm using Dynamic programming.

* Theory :→

The 'All pair shortest path' problem is a Fundamental problem in graph theory & algorithm. It involves finding the shortest path between all pair of vertices in a weighted graph. These short path can be measured in term of distance, time, cost or any other relevant metric depending on the problem context.

The problem statement for the above is follows the given a undirected directed graph with weighted edges. find the shortest path between every pair of vertices in the graph.

There are various way to solve this problem such as
1) Floyd warshall Algorithm
2) Johnson Algorithm.

1) Floyd's warshall Algorithm :→

It is a dynamic programm Algorithm used for finding the shortest paths between all the vertices in a weighted directed graph. The floyd warshall algorithm is for solving all pairs of shortest-path problem. The problem is to find the shortest distance between every pair of vertices in a given edge-weighted directed Graph.

It is an algorithm for finding the shortest Path :→

```
void Floydwarshall()
{ int cost[N][N];
  int i, j, k
  for(i=0; i<N; i++)
  for(j=0; j<N; j++)
  cost[i][j] = costMat[i][i]
  for(k=0; k<N; k++)
  {
   for(i=0; i<N; i++)
   for(j=0; j<N; j++)
   if(cost[i][j] > cost[i][k] + cost[k][j];
      cost[i][j] = cost[i][k] + cost[k][i];
   }
   // display the matrix cost(N)(N);
}
```

Algorithm :→

1. Initialize the sol^n matrix same as the input graph matrix as a first step.
2. Then updates the sol^n matrix by considering all vertices as an intermediate vertex.
3. 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.
4. When we pick vertex no. k as an intermediate vertex, we already have considered vertices {0, 1, 2, ... k-1} as intermediate vertices
5. for every pair (i, j) of the source and destination vertices respectively there are two possible cases.

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

(ii) K is an intermediate vertex in shortest path from i to j. We keep the ~~valu~~ update the value of dist[i][j] as dist[i][k] + dist[k][j] if dist[i][j] > dist[i][k] + dist[k][j].

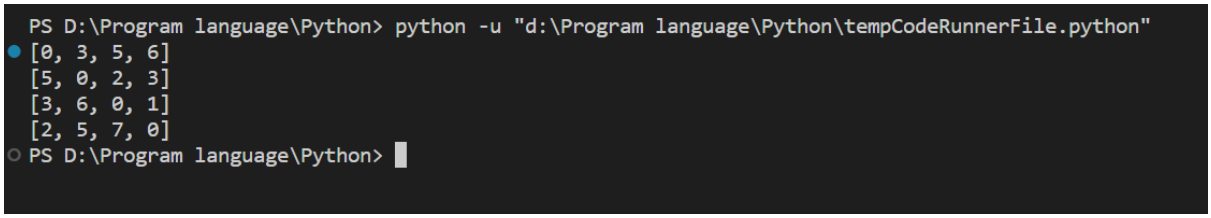
✓ ⑤ Bkeadan 07/10/23.

Aim: Implement All Pair Shortest paths problem using Floyd's Algorithm.

Program Code:

```
def floyd_warshall(graph):
    num_vertices = len(graph)

    # Initialize the distance matrix with the same values as the input graph
    dist = [[float('inf')] * num_vertices for _ in range(num_vertices)]

    for i in range(num_vertices):
        for j in range(num_vertices):
            if i == j:
                dist[i][j] = 0
            elif graph[i][j] != 0:
                dist[i][j] = graph[i][j]

    # Update the distance matrix using intermediate vertices
    for k in range(num_vertices):
        for i in range(num_vertices):
            for j in range(num_vertices):
                if dist[i][k] != float('inf') and dist[k][j] != float('inf')
and dist[i][k] + dist[k][j] < dist[i][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]

    return dist

# Example usage
inf = float('inf')
graph = [
    [0, 3, inf, 7],
    [8, 0, 2, inf],
    [5, inf, 0, 1],
    [2, inf, inf, 0]
]

result = floyd_warshall(graph)
for row in result:
    print(row)
```

Result:



Conclusion: Floyd's Algorithm finds shortest paths between all pairs of vertices in a weighted directed graph, suitable for medium-sized graphs (O(V^3) complexity). It detects negative weight cycles but becomes inefficient for large graphs. It has applications in network routing and transportation optimization but requires consideration of computational resources and negative cycles.