AMRITA
VISHWA VIDYAPEETHAM

Design & Analysis Of Algorithm

Lab Experiment -4

NAME : SURIYAPRAKASH.C

ROLL NO:CH.EN.U4CSE20170

SUBJECT: DESIGN & ANALYSIS OF ALGORITHM

SUBJECT CODE: 19CSE302

Submitted to – Mrs. Ashwini,
Department of CSE,
ASE Chennai campus.

EX NO: 4

Bellman Ford algorithm and Floyd warshall algorithm.

AIM:

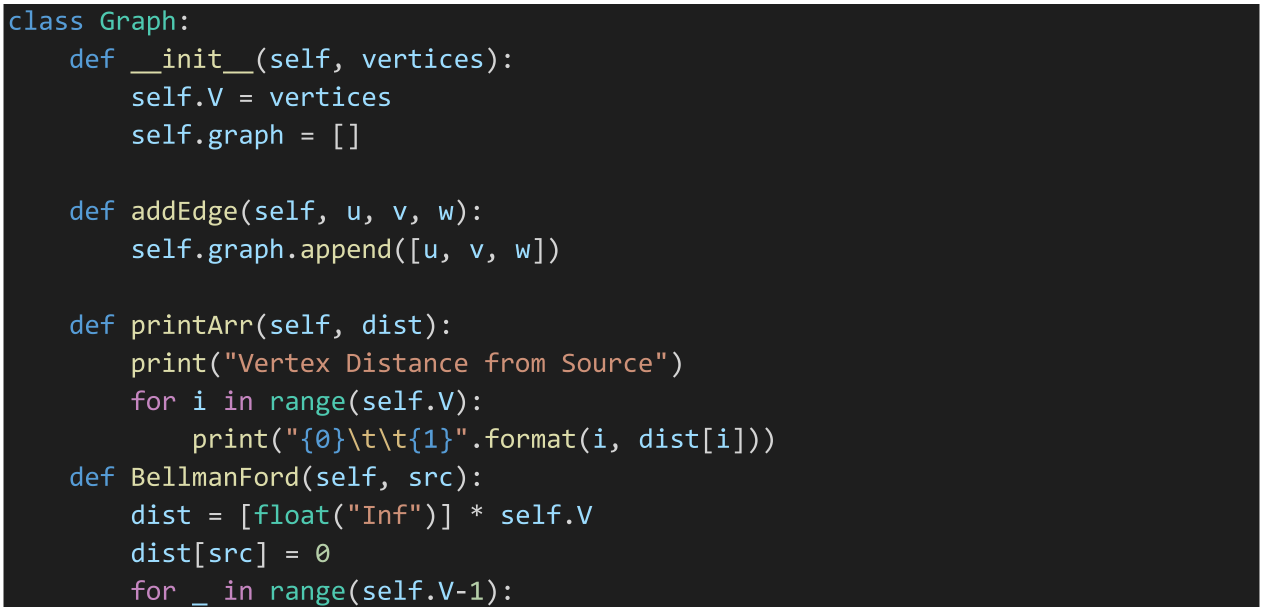
To write an algorithm to implement Bellman Ford algorithm.

ALGORITHM:

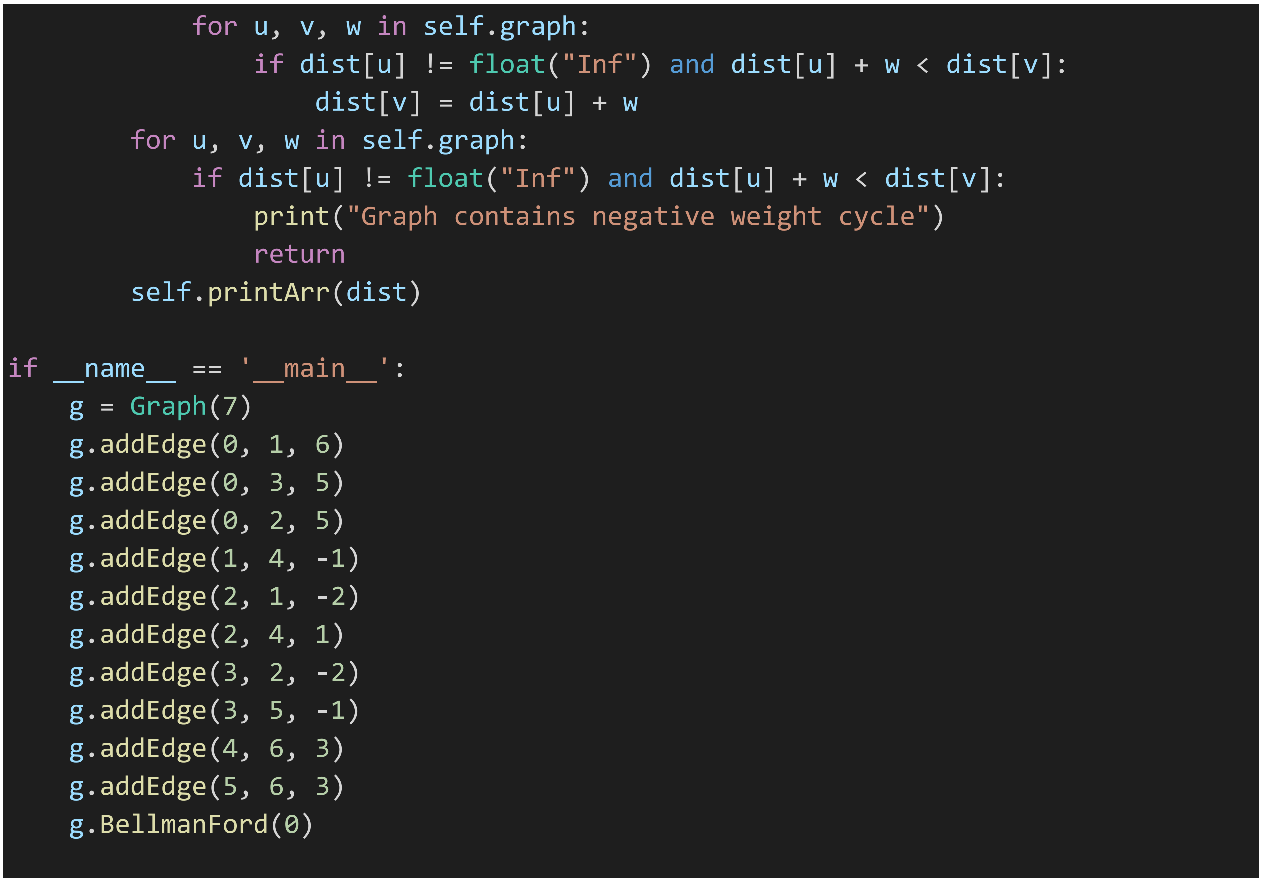
1) Declare and Define a class Graph with members –Value and an array to hold the graph.
2) The member function add_edge()is used to append the graph edge in the array.
3) The member function print Graph()is used to print the graph edges once the main operation is completed.
4) The member function bellman Ford():
    a. Sets all the distances as infinity except the source node.
    b. Calculates the shortest distance from source node to all nodes by using
       Dist [u]+ w < dist [v]

   And also checks for negative cycle.

CODE SCREEN:

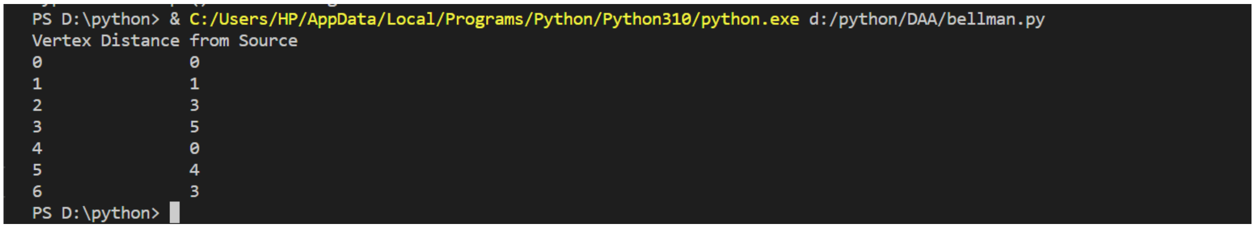


```
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []

    def addEdge(self, u, v, w):
        self.graph.append([u, v, w])

    def printArr(self, dist):
        print("Vertex Distance from Source")
        for i in range(self.V):
            print("{0}\t\t{1}".format(i, dist[i]))
    def BellmanFord(self, src):
        dist = [float("Inf")] * self.V
        dist[src] = 0
        for _ in range(self.V-1):
```



```python
            for u, v, w in self.graph:
                if dist[u] != float("Inf") and dist[u] + w < dist[v]:
                    dist[v] = dist[u] + w
        for u, v, w in self.graph:
            if dist[u] != float("Inf") and dist[u] + w < dist[v]:
                print("Graph contains negative weight cycle")
                return
        self.printArr(dist)

if __name__ == '__main__':
    g = Graph(7)
    g.addEdge(0, 1, 6)
    g.addEdge(0, 3, 5)
    g.addEdge(0, 2, 5)
    g.addEdge(1, 4, -1)
    g.addEdge(2, 1, -2)
    g.addEdge(2, 4, 1)
    g.addEdge(3, 2, -2)
    g.addEdge(3, 5, -1)
    g.addEdge(4, 6, 3)
    g.addEdge(5, 6, 3)
    g.BellmanFord(0)
```

OUTPUT SCREEN :



```
PS D:\python> & C:/Users/HP/AppData/Local/Programs/Python/Python310/python.exe d:/python/DAA/bellman.py
Vertex Distance from Source
0               0
1               1
2               3
3               5
4               0
5               4
6               3
PS D:\python>
```

TIME COMPLEXITY:

O(V*E), where V = No. of vertices, E = No. of edges

RESULT:

I have studied and understood the Bellman Ford algorithm

in python language and executed the program successfully.

Floyd Warshall Algorithm

AIM:

To write an algorithm to implement Floyd Warshall algorithm.

ALGORITHM:

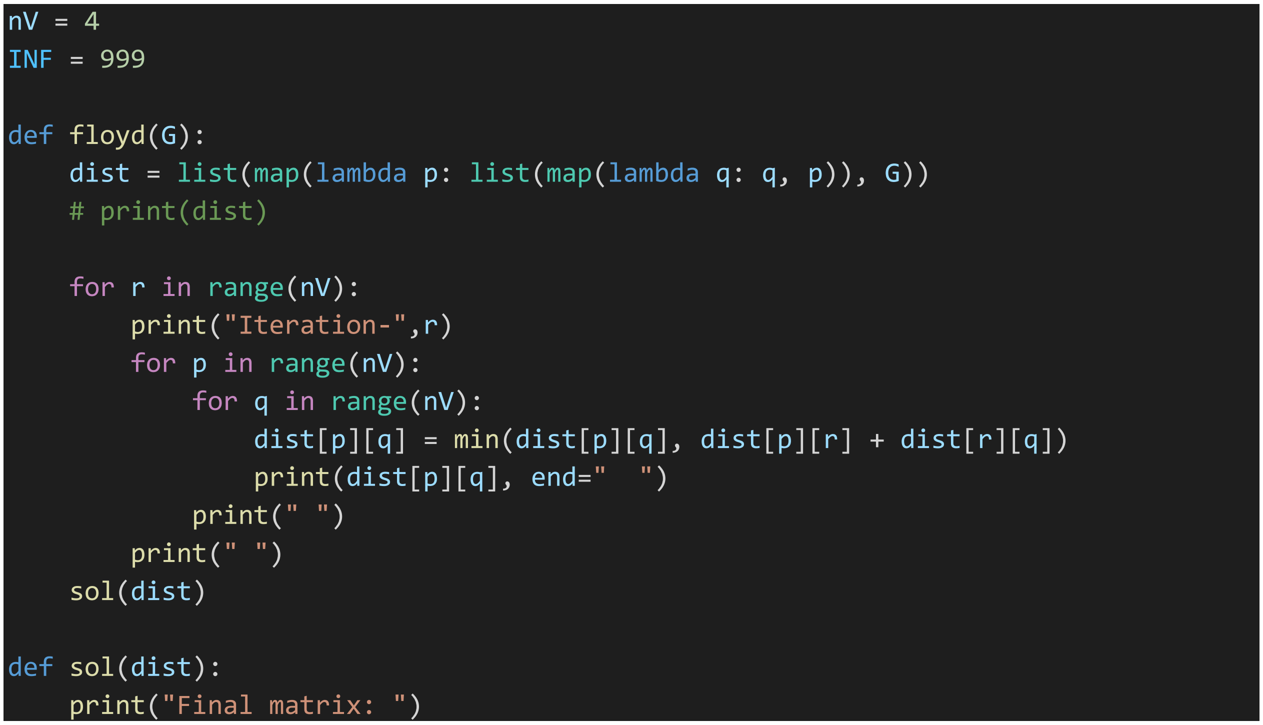
1) Give the no. of vertices and declare a variable inf and give some big value.
2) Declarea 2D array to represent the Adjacency Matrix.
3) In the functionalgo():

    For every pair of source and destination vertex, find the shortest path.

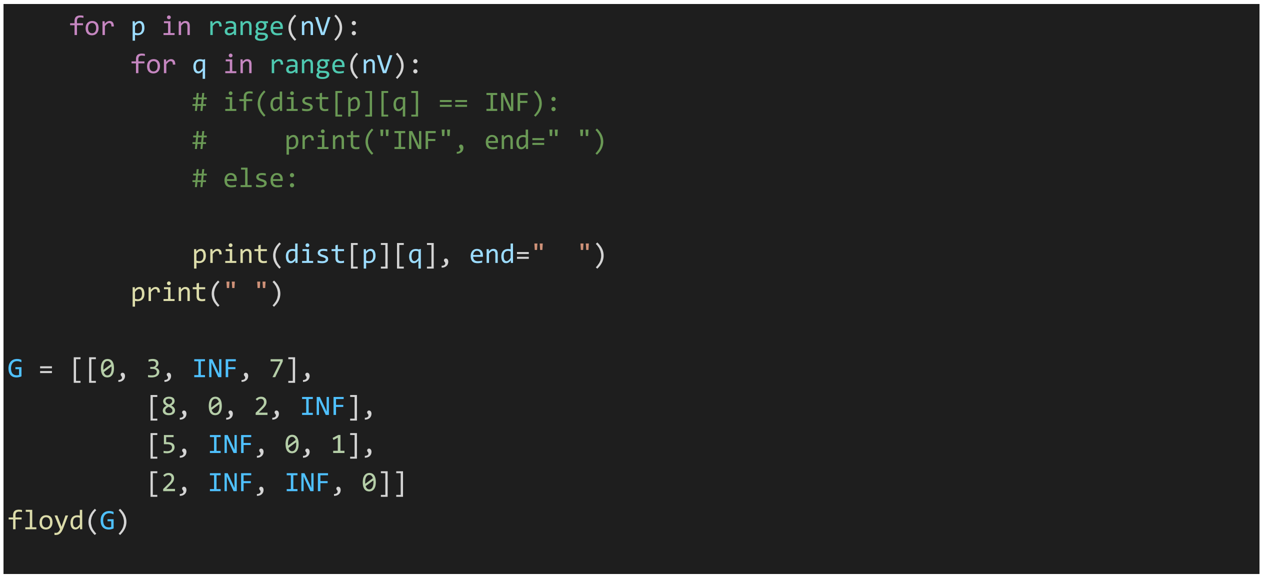
    If there is an intermediate path producing a shorter path, replace it with the previous path.

4) Print the final matrix which has the shortest path from node to node.

CODE SCREEN:

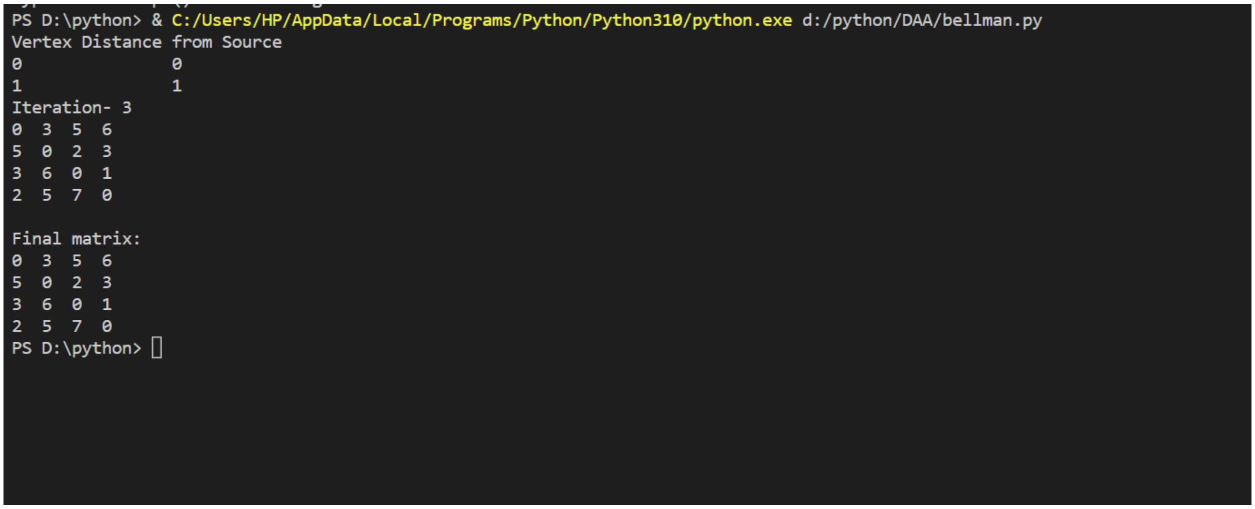


```
nV = 4
INF = 999

def floyd(G):
    dist = list(map(lambda p: list(map(lambda q: q, p)), G))
    # print(dist)

    for r in range(nV):
        print("Iteration-",r)
        for p in range(nV):
            for q in range(nV):
                dist[p][q] = min(dist[p][q], dist[p][r] + dist[r][q])
                print(dist[p][q], end="  ")
            print(" ")
        print(" ")
    sol(dist)

def sol(dist):
    print("Final matrix: ")
```



```
    for p in range(nV):
        for q in range(nV):
            # if(dist[p][q] == INF):
            #     print("INF", end=" ")
            # else:

            print(dist[p][q], end="  ")
        print(" ")

G = [[0, 3, INF, 7],
         [8, 0, 2, INF],
         [5, INF, 0, 1],
         [2, INF, INF, 0]]
floyd(G)
```

OUTPUT SCREEN :



```
PS D:\python> & C:/Users/HP/AppData/Local/Programs/Python/Python310/python.exe d:/python/DAA/bellman.py
Vertex Distance from Source
0               0
1               1
Iteration- 3
0  3  5  6
5  0  2  3
3  6  0  1
2  5  7  0

Final matrix:
0  3  5  6
5  0  2  3
3  6  0  1
2  5  7  0
PS D:\python> 
```

TIME COMPLEXITY:

O(n3), where n= Adjacent matrixsize

RESULT:

I have studied and understood the Floyd Warshall algorithm

THANK YOU !!