EXP 12 -  Implementation of DAG

AIM:

* To perform the implementation of DAG using python program.

ALGORITHM:

Following is complete algorithm for finding shortest distances.

1) Initialize dist[] = {INF, INF, ….} and dist[s] = 0 where s is the source vertex.

2) Create a toplogical order of all vertices.

3) Do following for every vertex u in topological order.

Do following for every adjacent vertex v of u

if (dist[v] > dist[u] + weight(u, v))

dist[v] = dist[u] + weight(u, v)

CODE:

from collections import defaultdict

class Graph:

def \_\_init\_\_(self, vertices):

self.V = vertices # No. of vertices

# dictionary containing adjacency List

self.graph = defaultdict(list)

# function to add an edge to graph

def addEdge(self, u, v, w):

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

# A recursive function used by shortestPath

def topologicalSortUtil(self, v, visited, stack):

# Mark the current node as visited.

visited[v] = True

# Recur for all the vertices adjacent to this vertex

if v in self.graph.keys():

for node, weight in self.graph[v]:

if visited[node] == False:

self.topologicalSortUtil(node, visited, stack)

# Push current vertex to stack which stores topological sort

stack.append(v)

def shortestPath(self, s):

# Mark all the vertices as not visited

visited = [False] \* self.V

stack = []

for i in range(self.V):

if visited[i] == False:

self.topologicalSortUtil(s, visited, stack)

# Initialize distances to all vertices as infinite and

# distance to source as 0

dist = [float("Inf")] \* (self.V)

dist[s] = 0

# Process vertices in topological order

while stack:

# Get the next vertex from topological order

i = stack.pop()

# Update distances of all adjacent vertices

for node, weight in self.graph[i]:

if dist[node] > dist[i] + weight:

dist[node] = dist[i] + weight

for i in range(self.V):

print("%d" % dist[i]) if dist[i] != float("Inf") else "Inf",

g = Graph(6)

g.addEdge(0, 1, 5)

g.addEdge(0, 2, 3)

g.addEdge(1, 3, 6)

g.addEdge(1, 2, 2)

g.addEdge(2, 4, 4)

g.addEdge(2, 5, 2)

g.addEdge(2, 3, 7)

g.addEdge(3, 4, -1)

g.addEdge(4, 5, -2)

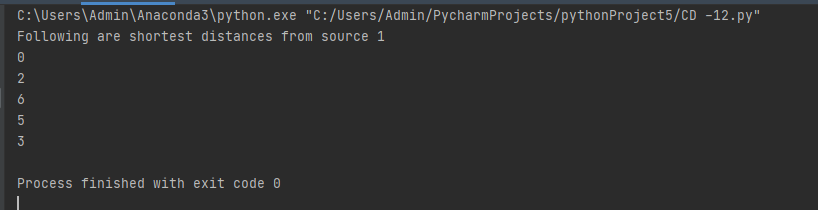
# source = 1

s = 1

print("Following are shortest distances from source %d " % s)

g.shortestPath(s)

OUTPUT:



RESULT:

* Hence implementation of DAG is performed using python program.