Viswaraje

21011102113

IOT B

Kruskal Algorithm

INF = float('inf')

g = [[INF, INF,2, INF, INF, INF],

[1,INF, INF, INF, INF,INF],

[INF, -2, INF, INF, INF,INF],

[-4, INF,-1, INF, INF, INF],

[INF, INF, INF, 1, INF,INF],

[10, INF, INF, INF,8,INF]]

V = 6

parent = [i for i in range(V)]

def find(i):

while parent[i] != i:

i = parent[i]

return i

# Does union of i and j. It returns

# false if i and j are already in same

# set.

def union(i, j):

a = find(i)

b = find(j)

parent[a] = b

def kruskalMST(cost):

mincost = 0 # Cost of min MST

# Initialize sets of disjoint sets

for i in range(V):

parent[i] = i

# Include minimum weight edges one by one

edge\_count = 0

while edge\_count < V - 1:

mi = INF

a = -1

b = -1

for i in range(V):

for j in range(V):

if find(i) != find(j) and cost[i][j] < mi:

mi = cost[i][j]

a = i

b = j

union(a, b)

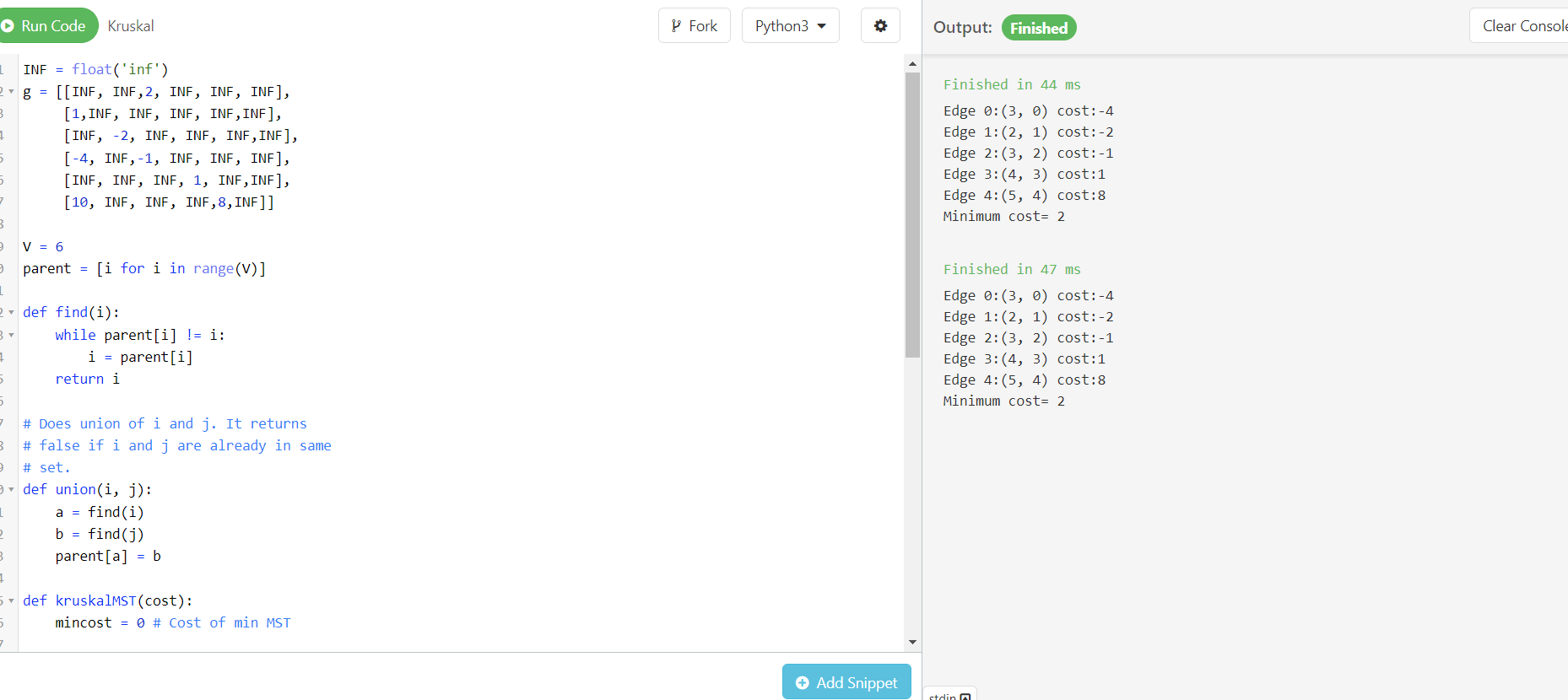
print('Edge {}:({}, {}) cost:{}'.format(edge\_count, a, b, mi))

edge\_count += 1

mincost += mi

print("Minimum cost= {}".format(mincost))

kruskalMST(g)



Minimum cost:2

**Prims Algorithm**

from collections import defaultdict

import heapq

def prims(graph, starting\_vertex):

mst = defaultdict(set)

visited = set([starting\_vertex])

edges = [

(cost, starting\_vertex, to)

for to, cost in graph[starting\_vertex].items()

]

heapq.heapify(edges)

min\_cost = 0

while edges:

cost, frm, to = heapq.heappop(edges)

min\_cost += cost

if to not in visited:

visited.add(to)

mst[frm].add(to)

for to\_next, cost in graph[to].items():

if to\_next not in visited:

heapq.heappush(edges, (cost, to, to\_next))

print("min cost : {}".format(min\_cost))

return mst

g = {

'A': {'C': 2},

'B': {'A': 1},

'C': {'B': -2},

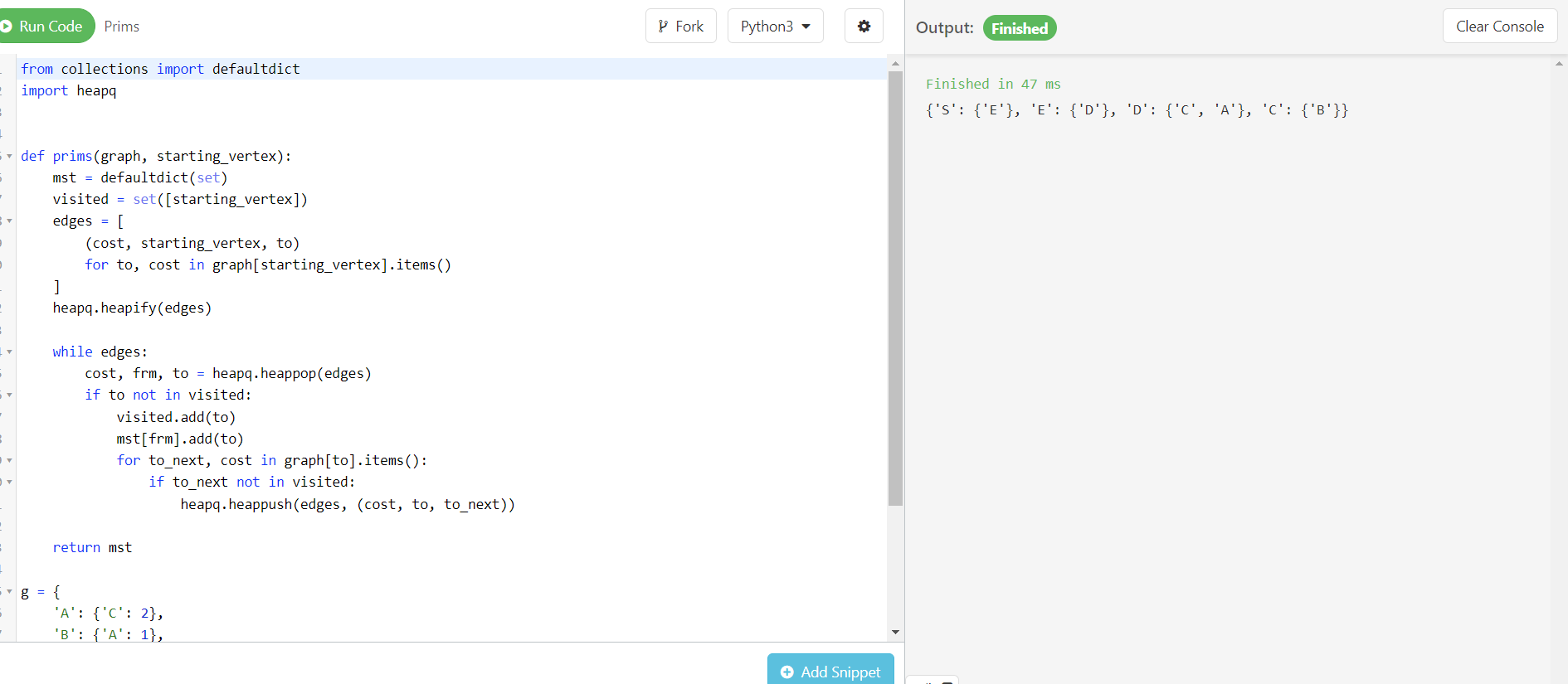
'D': {'A': -4, 'C': -1},

'E': {'D': 1},

'S': {'A': 10, 'E': 8}

}

print(dict(prims(g, 'S')))



Minimum cost:14

DIJKSTRA’S ALGORITHM

This algorithm does not work for negative weights.

\*I chose python programme for these problems to code.I am well versed in coding with python,   
Python Functions do not have restrictions on the type of the argument and the type of its return value.

\*Prim’s Algorithm grows a solution from a random vertex by adding the next cheapest vertex to the existing tree. Kruskal’s Algorithm grows a solution from the cheapest edge by adding the next cheapest edge to the existing tree / forest.

Prim’s Algorithm is faster for dense graphs. Kruskal’s Algorithm is faster for sparse graphs.