Practical Work nr. 4 – Group 913, Geanovu Medeea-Elena

5. Write a program that, given an undirected connected graph, constructs a minumal spanning tree using the Kruskal's algorithm.

Code in the DirectedGraph class:

**def kruskal(self):**

#initialize an empty graph which will be the minimum spanning tree

min\_spanning\_tree = DirectedGraph()

# Create a set for each vertex to track the disjoint sets

disjoint\_sets = {}

for vertex in self.vertices\_iterator():

disjoint\_sets[vertex] = set([vertex])

# Sort the edges in ascending order of their costs

edges = sorted(

[(first\_vertex, second\_vertex, self.get\_cost\_of\_edge(first\_vertex, second\_vertex))

for first\_vertex in self.vertices\_iterator()

for second\_vertex in self.outbound\_vertices\_iterator(first\_vertex)],

key=lambda edge: edge[2]

)

# Iterate through each edge and add it to the min spanning tree if it doesn't create a cycle

for edge in edges:

first\_vertex, second\_vertex, cost = edge

# Check if the edge creates a cycle

if not self.creates\_cycle(first\_vertex, second\_vertex, disjoint\_sets):

# Add the edge to the minimum spanning tree

if not min\_spanning\_tree.is\_vertex(first\_vertex):

min\_spanning\_tree.add\_vertex\_isolate(first\_vertex)

if not min\_spanning\_tree.is\_vertex(second\_vertex):

min\_spanning\_tree.add\_vertex\_isolate(second\_vertex)

min\_spanning\_tree.add\_edge\_isolate(first\_vertex, second\_vertex, cost)

# Merge the sets of the two vertices

self.merge\_sets(first\_vertex, second\_vertex, disjoint\_sets)

return min\_spanning\_tree

**def creates\_cycle(self, first\_vertex, second\_vertex, disjoint\_sets):**

# Find the root of the set containing the first vertex

first\_vertex\_root = self.find\_set\_root(first\_vertex, disjoint\_sets)

# Find the root of the set containing the second vertex

second\_vertex\_root = self.find\_set\_root(second\_vertex, disjoint\_sets)

# If the roots are the same, it means the vertices are in the same set and adding the edge will create a cycle

return first\_vertex\_root == second\_vertex\_root

**def find\_set\_root(self, vertex, disjoint\_sets):**

# Find the root of the set containing the given vertex

for root, vertices\_set in disjoint\_sets.items():

if vertex in vertices\_set:

return root

**def merge\_sets(self, first\_vertex, second\_vertex, disjoint\_sets):**

# Find the root of the set containing the first vertex

first\_vertex\_root = self.find\_set\_root(first\_vertex, disjoint\_sets)

# Find the root of the set containing the second vertex

second\_vertex\_root = self.find\_set\_root(second\_vertex, disjoint\_sets)

# Merge the sets by combining the two sets into one

disjoint\_sets[first\_vertex\_root] = disjoint\_sets[first\_vertex\_root].union(disjoint\_sets[second\_vertex\_root])

# Remove the second vertex root from the disjoint sets dictionary

del disjoint\_sets[second\_vertex\_root]

Code in the UI (it prints the edges and total cost of the minimum spanning tree for the given graph):

**def kruskal(self):**

k\_graph = self.\_\_graph.kruskal()

total\_cost = 0

if k\_graph.count\_number\_of\_edges() == 0:

print("There is no minimum spanning tree")

return

print("Resulted Minimum Spanning Tree:")

for start\_vertex, end\_vertex, cost in k\_graph.edges\_iterator():

print(f"Edge: {start\_vertex} -> {end\_vertex}, Cost: {cost}")

total\_cost = total\_cost + cost

print(f"Total cost: {total\_cost}")