Emmanuel Briones

Lab 6 Report

Introduction:

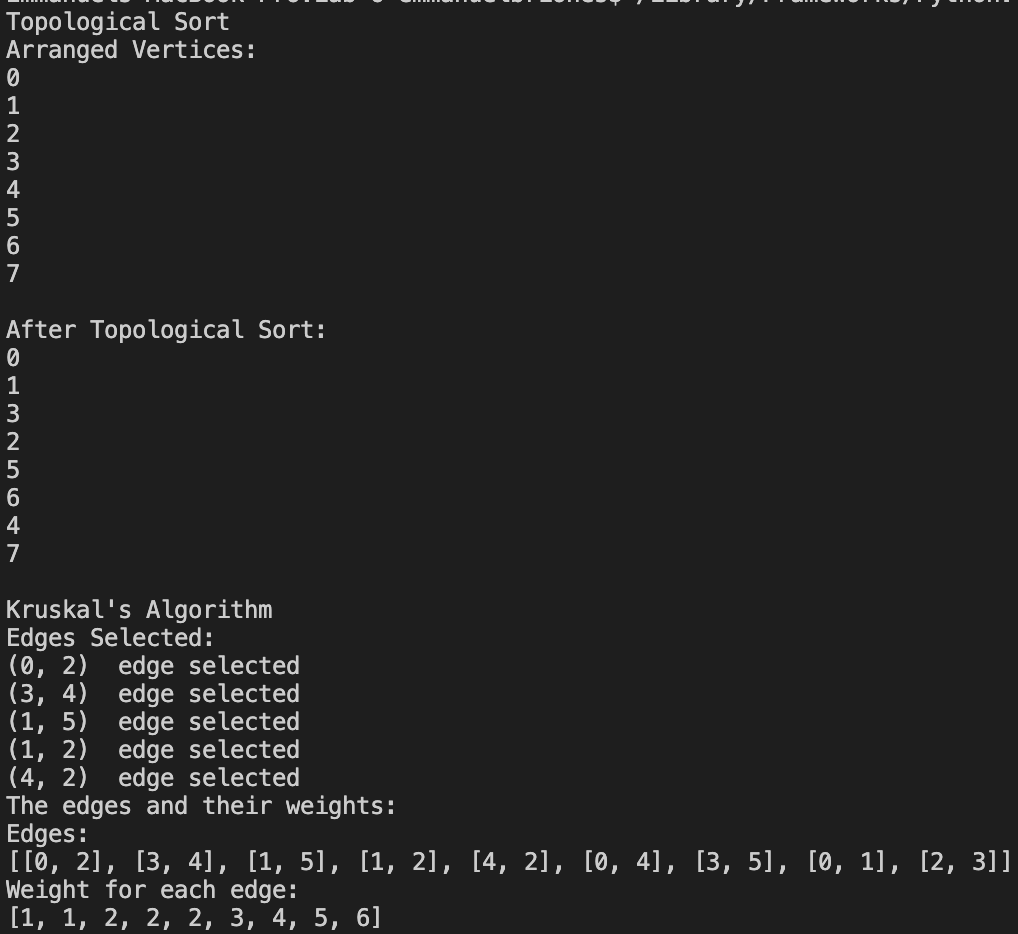
The problem that was to be solved was to implement the Kruskal’s algorithm and the topological sort algorithm. I had to develop a code that executes both algorithms at a very effective way. The Kruskal’s algorithm needs to find the least possible weight that connects to points. The code looks for the cheapest possible weight between points. The topological sort algorithm checks for any in degrees first. The first ones in the queue are the ones with no in degrees.

Proposed Solution Design and Implementation:

First, I created a method called Kruskal that first sorted the edges by their weights and then add the values to a list called vertices. Then while the number of vertices is greater than one, I check the edges and see which one is better and then select the two edges for a union. The method also prints the selected edges for the spanning tree. For the topological sort, I used Diego’s implementation and I just finished doing the methods that Diego did not provide. So, if there was a vertex that had no in degrees, it was the first to the queue and so on.

Experimental Results

Here are the results for the test cases that were used for both algorithms:



Appendix:

class kruskal\_graph:

edges = []

weight = []

vertices = []

def \_\_init\_\_(self, edge\_list, weight):

self.edges.append(edge\_list)

self.weight.append(weight)

def kruskal(self):

self.sort()

self.make\_set()

count = 0

i = 0

while len(self.vertices) > 1:

if self.find(self.edges[i][0]) != self.find(self.edges[i][1]):

print((self.edges[i][0], self.edges[i][1]), " edge selected")

count += 1

self.union(self.edges[i][0], self.edges[i][1])

i += 1

def add(self, edges, weight):

self.edges.append(edges)

self.weight.append(weight)

def sort(self):

if len(self.edges) != len(self.weight):

return

for i in range(1, len(self.weight)):

temp\_weight = self.weight[i]

temp\_edge = self.edges[i]

current = i - 1

while current >= 0 and temp\_weight < self.weight[current]:

self.weight[current + 1] = self.weight[current]

self.edges[current + 1] = self.edges[current]

current -= 1

self.weight[current + 1] = temp\_weight

self.edges[current + 1] = temp\_edge

def make\_set(self):

for i in range(len(self.edges)):

for j in range(len(self.edges[i])):

if self.edges[i][j] not in self.vertices:

self.vertices.append(self.edges[i][j])

for k in range(len(self.vertices)):

self.vertices[k] = [self.vertices[k]]

def find(self, vertex):

for i in range(len(self.vertices)):

for element in self.vertices[i]:

if element == vertex:

return i

return None

def union(self, vertex1, vertex2):

index1 = self.find(vertex1)

index2 = self.find(vertex2)

for element in self.vertices[index2]:

self.vertices[index1].append(element)

self.vertices.pop(index2)

def print(self):

print("Edges:")

print(self.edges)

print("Weight for each edge:")

print(self.weight)

class Queue:

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

self.items = []

def is\_empty(self):

return self.items == []

def enqueue(self, item):

self.items.insert(0, item)

def dequeue(self):

return self.items.pop()

def size(self):

return len(self.items)

def topological\_sort(graph):

all\_in\_degrees = graph.compute\_indegree\_every\_vertex()

sort\_result = []

q = Queue()

for i in range(len(all\_in\_degrees)):

if all\_in\_degrees[i] == 0:

q.enqueue(i)

while not q.is\_empty():

u = q.dequeue()

sort\_result.append(u)

for adj\_vertex in graph.vertices\_reachable\_from(u):

all\_in\_degrees[adj\_vertex] -= 1

if all\_in\_degrees[adj\_vertex] == 0:

q.enqueue(adj\_vertex)

if len(sort\_result) != graph.num\_vertices():

return None

return sort\_result

Conclusions:

What I learned from this lab assignment was to know how to implement both the Kruskal’s algorithm and the topological sort algorithm. It was interesting to see how both algorithms worked and it was also somewhat challenging. It was a fun lab to work on. This lab helped me to prepare for the final exam. I now know how the Kruskal and topological sort algorithms work.