Author: Carlos Fernando Castaneda

Class: CS 2302

Date Modified: December 1, 2019

Instructor: Diego Aguirre

Assignment: Lab 6 Kruskal and Topological

TA: Gerardo Barraza

**Introduction:**

In programming, graphs is a data structure that represents connections among different items, which are represented as vertices, which care connected by edges. In this lab, we are looking at two specific implementations of graphs, the first is called kruskal’s algorithm, which determines as subset of a graph’s edges with the minimum sum of each edge weight; the other is called Topological sort, which determines the order in which every vertex contained in a graph is contained. The job in this lab is to implement both algorithms using a separate file to contain my test cases.

**Proposed solution design and implementation:**

A picture containing watch, indoor

Description automatically generatedBefore I begin, I want to note that the graphs that I will test the algorithms are the same ones that were given to us from previous homework in the class. Below you will find the graphs for analysis, the one on the left is the one used for Kruskal’s Algorithm, and the one on the right is the one used for Topological sort:

A clock in the middle of a watch

Description automatically generated

The implementation of this lab is pretty straightforward for me, since the lab sheet already gives me enough directions to know what to do. For the Kruskal Algorithm and the Topoligical sort, I plan to add the implementation found within zybooks. Hopefully that will be enough, but I wouldn’t be surprised if I have to alter the methods to fit within the graphs that I have to test.

Lastly, I want to add another section on the opening menu where the user can either problems from the lab. This will be very helpful to me later as it will help me determine the run times of the separate classes that will be included in my program.

**Experimental results**:

Before I begin explaining my results, like in the last lab, I wanted to note that I split my program into three separate files, one for the main method, one for the Kruskal Algorithm, and one for the Topological Sort implementation. I mainly did this in the basis that its easier for me to keep my methods organized, and the fact that I must keep my test cases in a separate file, so I choose to implement these cases on the main method, while the implementations in another.

The implementation of Kruskals Algorithm was pretty easy to turn it into code, but I made some alterations to fit within the goal that I was trying to reach. Fist thing, I added an addEdge method that not only inserts the edge with its parameters into the graph, But I also wrote it to display which edges were bring added so I could keep track of its initialization. Secondly, I added a method that searches for the parent of an item, that way the program knows in which direction it must go.

For the implementation of the Topological sort, it was also straightforward, but like Kruskal’s, I also added some alterations to the graph to fit better with the algorithm that I was trying to create. As above I added a method that adds an edge to the graph. Secondly, I created a utility method that checks for every edge in the graph if its already visited for it to keep track of its progress; it mainly Booleans to indicate if the edge has been visited, otherwise it keep it stored in a stack for the final result.

As mentioned above, I added a section in the user interface that will allow them to access either problem from the lab by either pressing a for the Kruskal’s algorithm, or b for topological sort. As for the run times, I have included the results at the end of the document before the conclusion.

Below you will find the results from various states in the program:

My results from running option a, kruskal’s algorithm:

**A screenshot of a cell phone

Description automatically generated**

My results from running option B, using topological sort to find my solution:

**A screenshot of a social media post

Description automatically generated**

**Run times (Binary Search Trees):**

|  |  |  |
| --- | --- | --- |
| **Time** | **LRU Cache** | **Heap** |
| **1** | **0.0049 seconds** | **0.0010 seconds** |
| **2** | **0.0049 seconds** | **0.0006 seconds** |
| **3** | **0.0049 seconds** | **0.0009 seconds** |

**Graph representation:**

Obviously looking at the data you can see that the cache runs way faster than the heap, but in all honesty, that wasn’t the intent of the lab. I thought it would be appropriate to shoe the running times of both methods not for comparison, but to prove that the methods of the cache ran at O(1) time, which was a requirement of this lab.

**Conclusions**:

With this lab, I was able to learn to code better using the Python language, including using multiple python files to replicate kruskal’s algorithm, as topological sort to play around with graphs. I was able to implement algorithms that act as the basic structure for both algorithms, as well as experiment with problems from the homework in this lab.

**Appendix :**

**Main.py**

'''

Author: Carlos Fernando Castaneda

Class : CS 2302

Date Modified: November 25, 2019

Instructor: Diego Aguirre

Assingment: Lab 6 Kruskal and Topological Sort (1 0f 3)

TA: Gerardo Barraza

Purpose: To practice using graphs to implement

Kruskal's Algorithm and Topological Sort.

'''

#Used to calculate the time for each option on the lab

import time

#Imports the kruskal and the top sort programs needed to implement the graphs

import kruskal as K

import top\_sort as topSort

#NOTE: The graphs created in the main file are based on the ones from the previous homework given to us in class

#Method to create a hard-coded graph so it can be sent to the kruskal.py file

def Kruskal\_Algorithm():

#Sets the number of vertices in the graph

n = 10

#Creates the maximum number of vertices for this graph

g = K.kruskal(n)

print("Applying a graph of ",n," vertices.")

print("")

#Sets the edges that are present in the graph, reading from, to, and its weight value

g.addEdge(0, 1, 4)

g.addEdge(0, 4, 3)

g.addEdge(1, 4, 2)

g.addEdge(1, 5, 5)

g.addEdge(1, 2, 4)

g.addEdge(2, 5, 6)

g.addEdge(2, 6, 9)

g.addEdge(3, 6, 13)

g.addEdge(4, 7, 1)

g.addEdge(4, 5, 12)

g.addEdge(5, 7, 21)

g.addEdge(7, 8, 14)

g.addEdge(5, 8, 11)

g.addEdge(5, 6, 17)

g.addEdge(6, 8, 10)

g.addEdge(8, 9, 16)

g.addEdge(3, 9, 20)

print("")

print("Creating the new graph using Kruskal's Algorithm...")

#Sends the new formed graph to be sorted using kruskal's algorithm. The next class will print its result

g.Kruskal\_al()

def Topological\_sort():

#Sets the number of vertices in the graph

n = 9

#Creates the maximum number of vertices for this graph

g2 = topSort.Topsort(n)

print("Applying a graph of ",n," vertices.")

print("")

#Sets the edges that are present in the graph, reading from, and to values

g2.add\_edge(0, 1)

g2.add\_edge(4, 0)

g2.add\_edge(4, 1)

g2.add\_edge(7, 4)

g2.add\_edge(5, 4)

g2.add\_edge(5, 7)

g2.add\_edge(8, 7)

g2.add\_edge(8, 5)

g2.add\_edge(5, 1)

g2.add\_edge(2, 1)

g2.add\_edge(2, 3)

g2.add\_edge(5, 2)

g2.add\_edge(6, 2)

g2.add\_edge(6, 3)

g2.add\_edge(6, 5)

g2.add\_edge(6, 8)

g2.add\_edge(3, 1)

print("")

print("Creating the new graph for Topological Sort...")

print("")

#Sends the new formed graph to be sorted using kruskal's algorithm. The next class will print its result

g2.topological\_sort()

def main():

print("Welcome to the Kruskal and Topological Sort program!")

print("")

print("Do you want to see an implementation of kruskal's algorithm, or of topological sort? Select from below:")

print("")

print("A. Kruskal's Algorithm")

print("B. Topological Sort")

user\_selection = input()

count = 0

if (user\_selection == "A" or user\_selection == "a"):

print("Applying Kruskal's Algorithm...")

print("")

start1 = time.time()

Kruskal\_Algorithm()

end1 = time.time()

print('Running time was: ', end1 - start1, 'seconds.')

elif (user\_selection == "B" or user\_selection == "b"):

print("Applying Topological Sort...")

print("")

start2 = time.time()

Topological\_sort()

end2 = time.time()

print('Running time was: ', end2 - start2, 'seconds.')

elif count == 3:

print("ERROR: Too many incorrect statements!")

print("Farewell.")

else:

print("ERROR: Input not valid!")

count+=1

main()

main()

**kruskal.py**

'''

Author: Carlos Fernando Castaneda

Class : CS 2302

Date Modified: November 25, 2019

Instructor: Diego Aguirre

Assingment: Lab 6 Kruskal and Topological Sort (2 0f 3)

TA: Gerardo Barraza

Purpose: To practice using graphs to implement

Kruskal's Algorithm and Topological Sort.

'''

#Creates the node values necessary to use kruskal's algorithm

class kruskal:

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

self.V= vertices

self.graph = []

#Method that sorts the graph to be constructed using kruskal's algorithm

def Kruskal\_al(self):

#This will store the resultant edges into a list to be used later

list = []

#Indexes used for keeping track of the sorted edges and for the list[]

i\_edge = 0

i\_list = 0

#Will sort all the edges in an non-decreasing order of their respective weight.

self.graph = sorted(self.graph,key=lambda item: item[2])

#Creates two new sets called parent and ranked that keep the values of their respective variables

parent = []

ranked = []

# Create a set of subsets of rank of the maximum vertices V with single elements as placebos

for i in range(self.V):

parent.append(i)

ranked.append(0)

# While the number of edges to be taken is equal to V-1, it will select the smallest edge and increase its index

while i\_list < self.V -1 :

From,To,weight = self.graph[i\_edge]

i\_edge = i\_edge + 1

x = self.find(parent, From)

y = self.find(parent ,To)

# If including this particular edge doesn't cause a cycle, it will include it in the result and increment its index for the result of the next edge, otherwise it will discard it

if x != y:

i\_list = i\_list + 1

list.append([From,To,weight])

self.union(parent, ranked, x, y)

#Will print the results gathered form the class, including the edges and their respected weights

print("Construction Complete!")

print("Here are the resulting edges:")

print("")

for From,To,weight in list:

print("Edge: ", From, "---", To, "Weight: ", weight)

print("")

# Method that serves as a function of uniting two sets x and y to form a singular set

def union(self, parent, ranked, x, y):

#Creates two roots from their respective sets to traverse

x\_root = self.find(parent, x)

y\_root = self.find(parent, y)

#Replace an object in order of ascending depending of the root of each set; if they are the same rank, then make them one as a root and increment its rank by a value one

if ranked[x\_root] < ranked[y\_root]:

parent[x\_root] = y\_root

elif ranked[x\_root] > ranked[y\_root]:

parent[y\_root] = x\_root

else :

parent[y\_root] = x\_root

ranked[x\_root] += 1

#Method that adds an edge to the graph to be used for kruskals algorithm.

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

#Prints out the current edge being added to the graph

print("Adding an edge from",u,"to",v,",with a weight of",w)

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

#Metod that uses a utility function to find set of an element i using a parent as its index

def find(self, parent, i):

if parent[i] == i:

return i

return self.find(parent, parent[i])

**top\_sort.py**

*'''*

*Author: Carlos Fernando Castaneda*

*Class : CS 2302*

*Date Modified: November 25, 2019*

*Instructor: Diego Aguirre*

*Assingment: Lab 6 Kruskal and Topological Sort (3 0f 3)*

*TA: Gerardo Barraza*

*Purpose: To practice using graphs to implement*

*Kruskal's Algorithm and Topological Sort.*

*'''*

*#Imports the default dictionary from collections so that it can be used in the topological sort costructor*

*from collections import defaultdict*

*#Class to represent a graph*

*class Topsort:*

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

*self.graph = defaultdict(list) #dictionary containing adjacency List*

*self.V = vertices #No. of vertices*

*#Method that adds an edge to the graph to be used for topological sort.*

*def add\_edge(self,u,v):*

*self.graph[u].append(v)*

*#Method used as a recursive function to apply the rules of the topological sorting*

*def topological\_sort\_utility(self,v,visited,stack):*

*# Marks the current node visited as visited.*

*visited[v] = True*

*#For i in range of the graph,if the visited is false, then recall itself over again*

*for i in self.graph[v]:*

*if visited[i] == False:*

*self.topological\_sort\_utility(i,visited,stack)*

*# Push the current vertex to the stack which will store the results*

*stack.insert(0,v)*

*#Method that sorts the stack to be constructed using topological sort*

*def topological\_sort(self):*

*#Makes a list called visited, and marks them all as false as we havent began to traverse through the methods yet*

*visited = [False]\*self.V*

*#Marks our stack that we will use to store our values*

*stack =[]*

*#For i in range of all vertices, if the visited edge is falsem it will store that edge using the utility method*

*for i in range(self.V):*

*if visited[i] == False:*

*self.topological\_sort\_utility(i,visited,stack)*

*#Finally, it will print out the contents of the stack to the displayed the final sort*

*print("Construction Complete!")*

*print("Here are the results of the Topological Sort of the given graph: ")*

*print(stack)*

*print("")*

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provide inappropriate assistance to any student in the class.

