Timothy P. McCrary
12/5/2018

CS 2302 Lab 6 – Option A

Introduction

For this lab the main goal is to implement Topological Sort and Kruskal's Algorithm on a directed graph and non-directed graph respectively.

Proposed Solution Design and Implementation

In order to solve this lab, Kruskal's Algorithm and Topological Sort needed to be implemented, using the professors code, along with a graph that stores vertices and edges through either an adjacency list or an adjacency matrix (we choose which implementation). Using the adjacency list, the first thing was to create the Topological sort in its own file, then the Kruskal's Algorithm in another file, which will be called and run by the main runner file.

For Topological sort, the main code is already given to us, with *compute_indegree_every_vertex* and *get_adj_vertices_list* functions needing to be completed. The *compute_indegree_every_vertex* function was created in the same file, and as the name states, it takes a graph and returns a list of the in degree of every vertex, with the index representing the vertex and the element representing the number of in degrees. Moreover, the *get_adj_vertices_list* function was created in the graph class, with it returning a list of all the adjacent vertices of a single vertex. Once these two functions where created, the rest of the code could be completed, with it returning a list of vertices from X to Y, with X always being before Y.

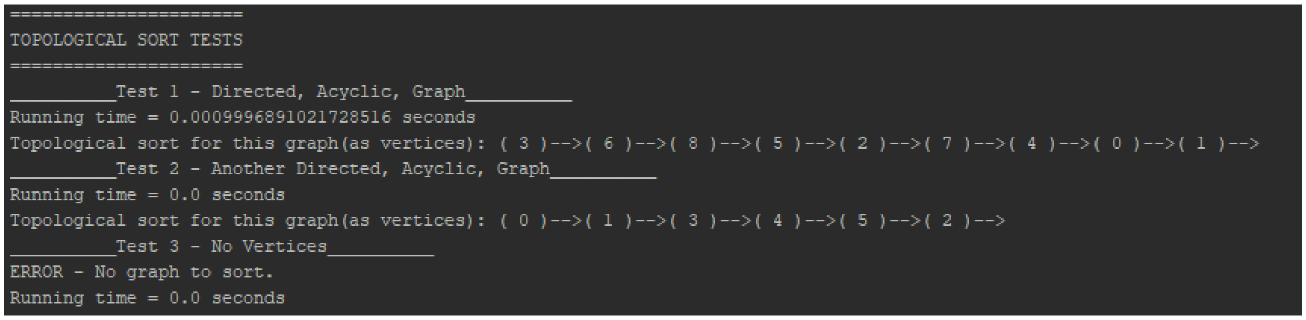
Kruskal's Algorithm was a tad hard to write, as the algorithm was given to use, not the actual code. The first thing created was a function to get all the edges by increasing cost, as this was the first step in Kruskal's Algorithm. Though not having the best running times, this function does successfully return the edges in order, along with the two vertices that edge connects. Once that function was completed the main algorithm was created. The second part of this algorithm is to place the weights in a list, unless they create a cycle within the graph (as we are trying to find the minimum spanning tree). To do this a disjoint set forest data structure was implemented, as it is a very simple and easy way to see if a vertex is connected to another vertex. With those two main parts completed the algorithm was completed, with it returning the minimum spanning tree as a list with the edge, and the two vertices that are connected by that edge.

Finally, a main runner file was created to call the other files in order to keep the code clean and organized. Since no user input was needed, the main runner just tests the algorithms on different graphs and prints out the results.

Experimental Results

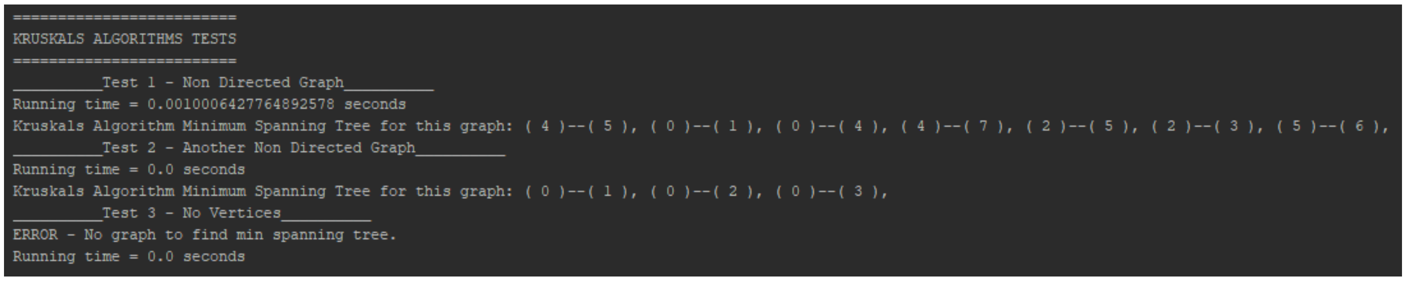
For this lab we are testing two algorithms, Topological Sort and Kruskal's Algorithm, so 3 tests for each algorithm were conducted, with Topological sort using a directed, acyclic graph, and Kruskal's using a non-directed graph. Test 1 for both algorithms used an ordinary medium sized graph, test 2 for both used another ordinary, but smaller, graph and test 3 for both used an empty graph.

Topological Sort: Tests 1, 2, and 3



```
======================
TOPOLOGICAL SORT TESTS
======================
__________Test 1 - Directed, Acyclic, Graph__________
Running time = 0.0009996891021728516 seconds
Topological sort for this graph(as vertices): ( 3 )-->( 6 )-->( 8 )-->( 5 )-->( 2 )-->( 7 )-->( 4 )-->( 0 )-->( 1 )-->
__________Test 2 - Another Directed, Acyclic, Graph__________
Running time = 0.0 seconds
Topological sort for this graph(as vertices): ( 0 )-->( 1 )-->( 3 )-->( 4 )-->( 5 )-->( 2 )-->
__________Test 3 - No Vertices__________
ERROR - No graph to sort.
Running time = 0.0 seconds
```

Kruskal's Algorithm: Tests 1, 2, and 3



```
=========================
KRUSKALS ALGORITHMS TESTS
=========================
__________Test 1 - Non Directed Graph__________
Running time = 0.0010006427764892578 seconds
Kruskals Algorithm Minimum Spanning Tree for this graph: ( 4 )--( 5 ), ( 0 )--( 1 ), ( 0 )--( 4 ), ( 4 )--( 7 ), ( 2 )--( 5 ), ( 2 )--( 3 ), ( 5 )--( 6 ),
__________Test 2 - Another Non Directed Graph__________
Running time = 0.0 seconds
Kruskals Algorithm Minimum Spanning Tree for this graph: ( 0 )--( 1 ), ( 0 )--( 2 ), ( 0 )--( 3 ),
__________Test 3 - No Vertices__________
ERROR - No graph to find min spanning tree.
Running time = 0.0 seconds
```

Conclusion

This lab was straight forward and challenging. Though we were given the implementation for both algorithms, the missing parts that needed to be created required some thinking, especially so it could fit in with the test of the code. This lab helped me really understand how graphs are represented in code with an adjacency list and an adjacency matrix.

Appendix

```
#
***********************************************************************************************
*********************************
# NAME: Timothy P. McCrary
# CLASS: CS 2302
# LAB 6 OPTION A
# INSTRUCTOR: Diego Aguirre
# TA: Manoj Pravaka Saha
# DATE: 12/3/2018
# PURPOSE: To use and manipulate a Graph data structure.
#
***********************************************************************************************
*********************************

import DisjointSetForest
import Graph
```

```python
import TopologicalSort
import KruskalsAlgorithm
import ExtraCredit
import time


def main():
    """
    Main function/Runner function.
    :return:
    """
    # ======================
    # TOPOLOGICAL SORT TESTS
    # ======================
    print('======================\nTOPOLOGICAL SORT TESTS\n======================')

    # __________TEST 1__________
    print('__________Test 1 - Directed, Acyclic, Graph__________')
    start_time = time.time()
    graph = Graph.GraphAL(9, True)

    graph.add_edge(0, 1)
    graph.add_edge(2, 1)
    graph.add_edge(3, 2)
    graph.add_edge(3, 6)
    graph.add_edge(4, 0)
    graph.add_edge(4, 1)
    graph.add_edge(5, 1)
    graph.add_edge(5, 2)
    graph.add_edge(5, 4)
    graph.add_edge(5, 7)
    graph.add_edge(6, 2)
    graph.add_edge(6, 5)
    graph.add_edge(6, 8)
    graph.add_edge(7, 4)
    graph.add_edge(8, 5)
    graph.add_edge(8, 7)

    top_sort = TopologicalSort.topological_sort(graph)
    print("Running time = %s seconds" % (time.time() - start_time))
    print_top_sort(top_sort)

    # __________TEST 2__________
    print('\n__________Test 2 - Another Directed, Acyclic, Graph__________')
    start_time = time.time()
    graph = Graph.GraphAL(6, True)

    graph.add_edge(0, 1)
    graph.add_edge(0, 3)
    graph.add_edge(0, 3)
    graph.add_edge(1, 2)
    graph.add_edge(1, 4)
    graph.add_edge(3, 4)
    graph.add_edge(4, 2)
    graph.add_edge(4, 5)
    graph.add_edge(5, 2)

    top_sort = TopologicalSort.topological_sort(graph)
    print("Running time = %s seconds" % (time.time() - start_time))
    print_top_sort(top_sort)

    # __________TEST 3__________
    print('\n__________Test 3 - No Vertices__________')
    start_time = time.time()
    graph = Graph.GraphAL(0, True)

    top_sort = TopologicalSort.topological_sort(graph)
    print("Running time = %s seconds" % (time.time() - start_time))
    print_top_sort(top_sort)

    # ========================
    # KRUSKALS ALGORITHM TESTS
    # ========================
    print('\n\n=========================\nKRUSKALS ALGORITHMS
TESTS\n=========================')

    # __________TEST 1__________
    print('__________Test 1 - Non Directed Graph__________')
    start_time = time.time()
    graph = Graph.GraphAL(8, False)  # Creates graph.

    # Adds edges between vertices.
    graph.add_edge(0, 1, 2)
    graph.add_edge(0, 4, 3)
    graph.add_edge(1, 2, 9)
    graph.add_edge(1, 4, 6)
    graph.add_edge(1, 5, 8)
    graph.add_edge(2, 3, 10)
    graph.add_edge(2, 5, 7)
    graph.add_edge(2, 6, 15)
    graph.add_edge(3, 6, 16)
    graph.add_edge(4, 5, 1)
    graph.add_edge(4, 7, 4)
    graph.add_edge(5, 6, 12)
    graph.add_edge(5, 7, 11)
    graph.add_edge(6, 7, 13)

    kru_alg = KruskalsAlgorithm.kruskals_algorithm(graph)
    print("Running time = %s seconds" % (time.time() - start_time))
    print_kru_alg(kru_alg)

    # __________TEST 2__________
    print('\n__________Test 2 - Another Non Directed Graph__________')
    start_time = time.time()
    graph = Graph.GraphAL(4, False)  # Creates graph.

    # Adds edges between vertices.
    graph.add_edge(0, 1, 1)
    graph.add_edge(0, 2, 2)
    graph.add_edge(0, 3, 3)
    graph.add_edge(1, 2, 3)
    graph.add_edge(1, 3, 4)
    graph.add_edge(2, 3, 5)

    kru_alg = KruskalsAlgorithm.kruskals_algorithm(graph)
    print("Running time = %s seconds" % (time.time() - start_time))
    print_kru_alg(kru_alg)

    # __________TEST 3__________
    print('\n__________Test 3 - No Vertices__________')
    start_time = time.time()
    graph = Graph.GraphAL(0, False)  # Creates graph.

    kru_alg = KruskalsAlgorithm.kruskals_algorithm(graph)
    print("Running time = %s seconds" % (time.time() - start_time))
    print_kru_alg(kru_alg)
    # # ============
    # # EXTRA CREDIT
    # # ============
    #
    # graph = Graph.GraphAL(8, False)  # Creates graph.
    #
    # # Adds edges between vertices.
    # graph.add_edge(0, 1, 2)
    # graph.add_edge(0, 4, 3)
    # graph.add_edge(1, 2, 9)
    # graph.add_edge(1, 4, 6)
    # graph.add_edge(1, 5, 8)
    # graph.add_edge(2, 3, 10)
    # graph.add_edge(2, 5, 7)
    # graph.add_edge(2, 6, 15)
    # graph.add_edge(3, 6, 16)
    # graph.add_edge(4, 5, 1)
    # graph.add_edge(4, 7, 4)
    # graph.add_edge(5, 6, 12)
    # graph.add_edge(5, 7, 11)
    # graph.add_edge(6, 7, 13)




def print_top_sort(top_sort):
    """
    Prints Topological sort of graph.
    :param top_sort:
    :return:
    """
    if top_sort is None:
        return

    print('Topological sort for this graph(as vertices): ', end='')

    for value in top_sort:
        print('(', value, ')-->', end='')


def print_kru_alg(kru_alg):
    """
    Prints the min spanning tree of graph obtained from Kruskals Algorithm.
    :param kru_alg:
    :return:
    """
    if kru_alg is None:
        return

    print('Kruskals Algorithm Minimum Spanning Tree for this graph: ', end='')

    for value in kru_alg:
        print('(', value[1], ')--(', value[2], '), ', end='')


if __name__ == '__main__':
    main()
```

“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 provided inappropriate assistance to any student in the class.”

X____Timothy P. McCrary____