Computing Graphs MST Prim using Python

Homework #13

By

Mario Pendleton

CS 303 [Algorithms and Data Structures](https://uab.instructure.com/courses/1507655)

November 26, 2019

### Problem Specification

Prim’s Algorithm for minimum spanning trees.

**Problem**

1. Implement a weighted graph class. Write a driver program, which reads input file mediumDG.txt (downloadable from Canvas) and display the weighted graphs by printing adjacency list.

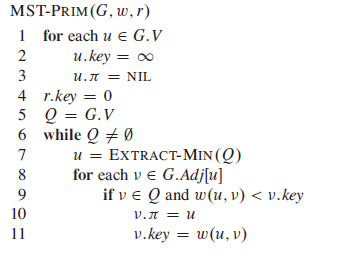
2. Implement Prim’s algorithm (provided below). Use your pre-lab assignment to check whether the output of your code is correct or not (tinyDG.txt).

3. Write a driver program, which reads input files mediumGraph.txt, LargeGraph.txt and XtraLargeGraph.txt (downloadable from Canvas) and run Prim’s algorithm on each of them to find the minimum spanning tree within these graphs. Record the times required for each of these graphs and include this in your report.

NB: For the following pseudo code, you need to use a priority queue. You can use your own code from the heap sort assignment or use the appropriate Java or Python packages for the priority queue.

Program Design

This program requires a linked list of data that will be searched using MST Prim. The core methods were design after the pseudo code below.



The following steps were required to develop this program:

1. Write a node class in python.

Class Node:

\_\_init\_\_(self, key) – Key value, Neighbors array, Parent Node, Node Color, Node distance, Node Finish

Methods within the class.

. printNode(self) – Prints a node stats.(key, data, left, right, parent, color)

. addNeighbor(self, v) – Adds an element to the array of neighbors.

. addNeighbors(self, vertices) – Appends an array of elements to the array of neighbors.

. Getters Key value, Neighbors array, Parent Node, Color, distance, Finish

1. Write a Graph class in python.

Class Graph

\_\_init\_\_(self) – Graph Dictionary , Q, edges

Methods with the class.

. addVertex(self, v) – Inserts a node into the graph

. addVertices(self, v) – Inserts an array of nodes into the graph

. addEdge(self, u, v) – Establishes a connection between nodes u and v making them neighbors.

. printGraph(self) – Prints the sorted graph’s keys and corresponding neighbors.

. BSF(self, s) – Performs a Breadth First Search of the graph starting with node s.

. printPath(self, s, v) – Prints the shortest distance from node s to v after the BSF

. DFS() – Preps the node and calls DFSV(node)

. DFSV() – Perform actual DFS search and queuing

. DFSQ() - Places the results of DFS in the into the graphs.Q

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Input Values | Output Values | Reverse Bubble Sort |
| (a) | Nodes in assignment | (0,6)(2,6)(3,1)(4,6)(5,4)(6,3)(7,0) | 0.00357 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Use the a method to read the following txt files and covert them into arrays to be sorted

* TinyDG.txt
* mediumDG.txt
* TinyDG.txt
* LargeDG.txt

### Testing Plan

In the node testing sample nodes were created (A,B,C,D,E,F,G,H). Corresponding the nodes found on lab 13 assignment. A list of neighbors was established to situate the graph. The getters were then tested on node A. To test the graph a new instance of Graph was established. The pointer to graph and the empty graph dictionary were printed. Nodes (A,B,C,D,E,F,G,H) were then imported into the graph and printed.

### Test Cases

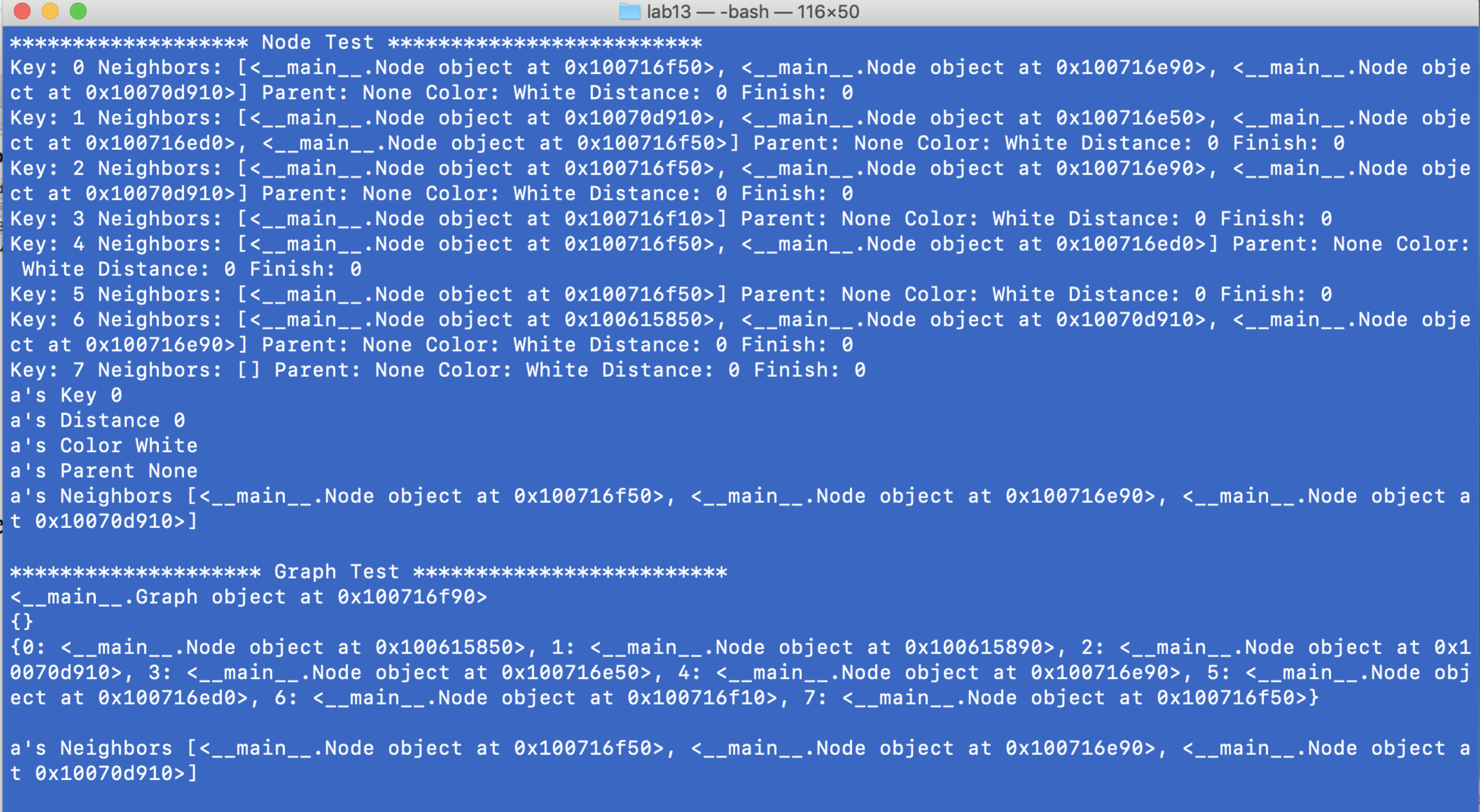
The test cases are shown in the table below using a MacBook Pro 16GB, 8 core 2.3 GHz Intel Core i9:

### Analysis and Conclusions

### References

Textbook, python.org, and examples provided in the assignment.

**Screen Shot**

****

