Computing Graphs BFS using Python

Homework #10

By

Mario Pendleton

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

November 1, 2019

### Problem Specification

Implement an undirected graph using adjacency lists.

Implement Breadth First Search algorithm on graphs.

**Problem**

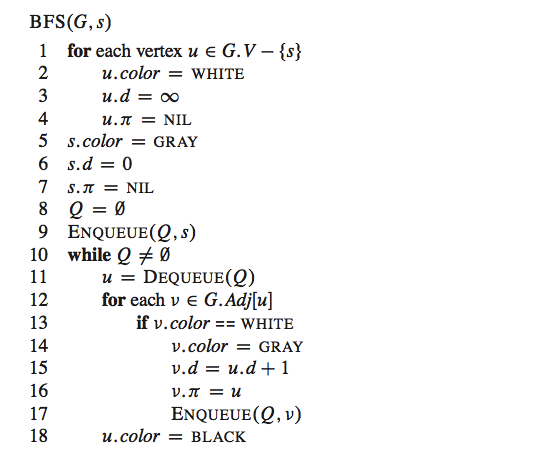
1. Write a program to implement an undirected graph using adjacency lists. Write a driver program to read the input files (mediumG.txt and largeG.txt) that contain the number of vertices, number of edges, and the corresponding list of edges and create an undirected graph. You can verify that the graph was constructed successfully by displaying the graph in the form of adjacency list for each vertex (see example below).

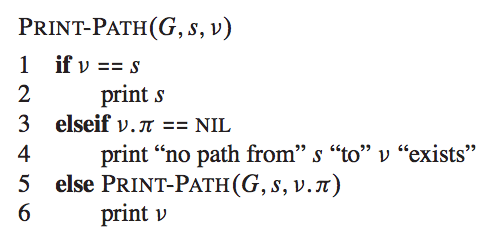
2. Implement BFS algorithm on the undirected graphs (created in Problem 1) following the pseudo-code given in the next page. Print the BFS paths from a source (you can pick any vertex as the source) to all the other nodes in the graph.

3. Write a detailed report on the performance of the BFS algorithm for the two input files.

### Program Design

This program requires a linked list of data that will be searched for the shortest distance in a Graph. 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, and Node distance

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 for Key value, Neighbors array, Parent Node, Node Color, and Node distance

1. Write a Graph class in python.

Class Graph

\_\_init\_\_(self) – Graph Dictionary

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input Values | Output Values | Actual Output | Reverse Bubble Sort |
| (a) | (ABCDEFG) | GACEF | GACEF | 0.000241 |
| (b) | load(path) path = mediumG.txt |  |  | 0.0061 |
| (c) | load(path) path = largeG.txt |  |  | 27.065 |
|  |  |  |  |  |

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

* LargeG.txt
* mediumG.txt

### Testing Plan

In the node testing sample nodes were created (A,B,C,D,E,F,G). A list of neighbors was established to situate the graph. Node G was created as a floating node and is later connected to node A using the addEdge() method. 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) were then imported into the graph and printed. The union between node A and G is created and node A’s updated list of neighbors is printed. Edges were then reinforced using the addEdges() method although not necessary. The graph is printed again using the printGraph() method. Now that the graph is complete; BFS() is called on node G (also tested all other nodes) and the printPath() is called on node G and F (also tested a other pars of nodes such as A & F). Testing is then performed on the files provided.

### 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

### BSF was a success.

### References

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

**Screen Shots**

