Graphs : (Un)directed Breadth First Searching
Lab Assignment #10

by
Justin Anthony Timberlake

CS 303 Algorithms and Data Structures
November 1st, 2017

1. Problem Specification

The purpose of this lab was to implement DirectedGraph and UndirectedGraph classes, given the Graph class with the basic structure set up initially. Then, using an adjacency list, we were to print out all of the edges for each vertex. After this, we needed to implement the Breadth First Search algorithm on the given Graph of either type which was just instantiated. Finally, for the homework portion, we needed to build the Graph using an adjacency Matrix instead of just an adjacency list as before. Then, taking in the three sizes of graphs (based on the list of edges), compare the runtimes between using an adjacency list and using an adjacency matrix.

2. Program Design

The steps taken to develop the program were:

a) Using the classes given Graph.java, implement the UndirectedGraph and DirectedGraph classes as well as the Vertex class (within the Lab10 java file).

b) Use print statements to print out the read in vertex objects in the form of adjacency lists before the BFS.

c) Implement the BFS algorithm within the UndirectedGraph class.

d) Change adj (adjacency list) to a 2-dimensional array of booleans to show which vertices have edges to other vertices. Other than that, the BFS algorithm remains largely the same.

e) Generate printouts for the runtimes comparison between different sizes of adjacency lists versus adjacency matrices.

No additional methods were used in the primary (driver) class.

The following methods were used in secondary class, Vertex.

a) Vertex(int k) - constructor for objects of type Vertex, setting the key equal to k

b) getKey() - returns the integer key value of a given Vertex object.

c) getColor() - returns the String color value of a given Vertex object.

d) getParent() - returns the Vertex parent object of a given Vertex object.

e) setColor(String col) – sets the color value equal to the String col.

f) setDistance(int d) – sets the distance value equal to the integer d.

g) setParent(Vertex p) – sets the parent object equal to the Vertex p.

The following methods were used in secondary class, Graph.

a) Graph() – constructor for objects of type Graph, setting V = E = 0, for vertices and edges.

b) Graph(BufferedReader reader) – takes a BufferedReader object to read in the files, parse each line, and add a vertex for each entry in the files with an adjacency list.

c) addEdge(int v, int w) – method to be overloaded for adding an edge to a given Graph type.

d) tostring() - generates a string printout of an adjacency list from a particular graph and returns this string to be printed out.

The following methods were used in secondary class, UndirectedGraph.

a) UndirectedGraph(BufferedReader reader) – exactly the same as in Graph

b) addEdge(int v, int w) – Given two integers, adds an edge going in each direction from each of the values' corresponding vertices.

c) tostring() - exactly the same as in Graph

d) BFS (Vertex s) – launches a breadth first search from the given Vertex to all possible other values in the UndirectedGraph.

e) printPath (Vertex s, Vertex v) – Given a source vertex and an endpoint, will print out a post-BFS traversal of all paths from the Vertex v to s recursively.

The following methods were used in secondary class, DirectedGraph.

a) DirectedGraph(BufferedReader reader) – exactly the same as in Graph

b) addEdge(int v, int w) – Given two integers, adds an edge going in one direction from v's corresponding vertex to w's corresponding vertex.

c) tostring() - exactly the same as in Graph

The following methods were used in secondary class, MatrixGraph.

a) UndirectedGraph(BufferedReader reader) – exactly the same as in Graph, except the individual Vertex objects are read into a LinkedList array and the adjacency list is replaced by a two dimensional adjacency array (matrix) of booleans signifying whether or not one value within the range of 0 to V – 1 has a corresponding edge to any other given value within that range of Vertex objects.

b) addEdge(int v, int w) – Given two integers, adds an edge going in each direction from each of the values' corresponding vertices and marks their corresponding edge in the adjacency matrix as true.

c) tostring() - References positions in the adjacency matrix to generate a printable string of the paths between vertices very similarly as to how the Graph class does with this method.

d) BFS (Vertex s) – Largely the same algorithm as before, but using the list of Vertex objects instead of just an adjacency list by itself.

e) printPath (Vertex s, Vertex v) – exactly the same as in UndirectedGraph

3. Testing Plan

Once the given classes had all of their methods properly implemented, the only thing to do was to use print statements to compare the times it took to launch a BFS on the three varying sizes of graphs respectively with either an adjacency list or adjacency matrix and compare.'

4. Test Cases

For generating a graph with an adjacency matrix and then using a BFS for the graph on one particular value, the runtime was 2,619,263 nanoseconds with the tiny graph file as input, 12,452,756 nanosecond with the medium graph file as input, and 21,042,695,952 nanoseconds with the large graph file as input.

For the first two inputs, using an adjacency matrix helped the runtime significantly with 1,025,977 nanoseconds for the tiny graph input file and 5,150,821 nanoseconds for the medium graph input file. However, for the large graph input file, the program could not construct the MatrixGraph object as Java ran out of heap space, perhaps as a result of a restriction upon my computer or other limitations within Java itself.

5. Analysis and Conclusions

For every time which the graphs were constructed and BFS was run, there was a noticeable decrease in runtime when using the adjacency matrix instead of the list. This could have been because of the constant access time to predefined boundaries in a V by V sized two dimensional matrix (array). However, the space complexity for this clearly outsourced my personal computer as the last and largest graph file could not be completed with my implementation of the BFS by means of creating an undirected graph with an adjacency matrix. The exception was thrown when attempting to create a V by V size matrix for the adjacency booleans. This was a limitation which would have occurred no matter what as the table was not even being populated yet. So, this leads me to believe that while using an adjacency matrix may be faster for building and using BFS than an adjacency list by itself, there are space complexity restrictions on large values of V (apparently only good for less than 1,000,000 on my personal computer). To prevent the program from crashing, I have these lines of calculations commented out at the end of my driver program.

6. References

All code in this assignment was either presented in prior labs and assignments or provided through means of pseudocode within the in-class portion of the lab.