EX3: BFS, DFS, & Dijkstra's

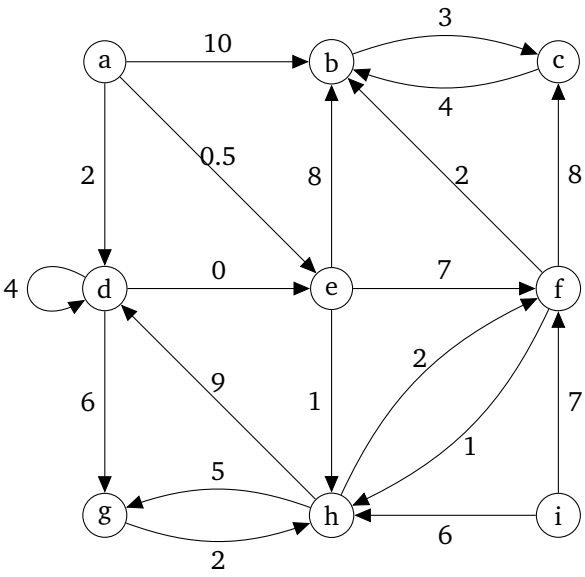
Due date: Friday August 7, 2020 at 11:59 pm

Instructions: Submit your responses to the "EX 3: BFS, DFS, & Dijkstra's" assignment on Gradescope here: https://www.gradescope.com/courses/141341/assignments/577261. Make sure to log in to your Gradescope account using your UW email to access our course.

These problems are meant to be done **individually**. If you do want to discuss problems with a partner or group, make sure that you're writing your answers individually later on. Check our course's collaboration policy if you have questions.

1. Dijkstra's Algorithm

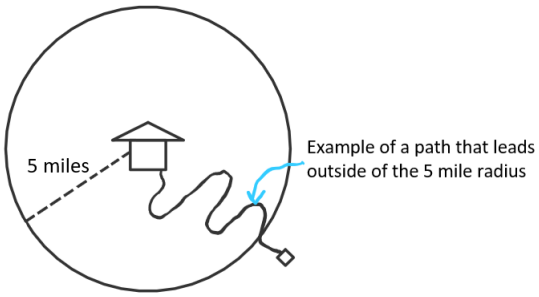
Consider the following graph:



Run Dijkstra's algorithm on this graph to compute the shortest path tree for this graph starting on node a. For each vertex in the graph (a - i), you should give the final result and show your work for the distTo and edgeTo data that gets stored. Make sure to show the intermediate values you had during the algorithm. To do this in Gradescope, you'll list out all the proposed distances / predecessor edges as a list in chronological order as they occur during the algorithm, where the last item in the list is the actual best distance / predecessor edge.

2. Graph Modeling

Imagine you have a goal to visit one new place each day but you've already seen everything within a 5-mile circular radius of your house. Today you want to find some path to a new place (i.e. a place that's outside your 5-mile radius). Below is an image describing what we mean by 5-mile radius.



Note that paths don't have to be straight lines, so it's possible to travel x miles along a path but end up less than x miles away from the starting point.

Explain how you would model this scenario as a graph to help plan a route that goes outside of the 5 mile radius.

(a) What do your vertices represent? If you store extra information in each vertex, what is it?

(b) What do your edges represent? Your answer may be a real-world object or an abstract description of what edges will exist. If you store extra information in each edge, what is it?

(c) Is your graph directed or undirected? Briefly explain why in 1-2 sentences.

(d) Is your graph weighted or unweighted? Briefly explain why in 1-2 sentences.

(e) Do you permit self-loops (i.e. edges from a vertex to itself)? Parallel edges (i.e. more than one copy of an edge between the same location)? Briefly explain why in 1-2 sentences?

(f) Describe how you would use DFS to discover the path to a place outside of the 5 mile radius, and return that path. Your answer should include a description of how you use any information in the graph described above and a description of how you stop your algorithm.

(g) What is the tight Big-Theta bound for the worst-case running time of your algorithm? Assume your graph has n nodes (vertices) and m edges.

3. Implementing BFS for Real Data

In this problem, you'll implement BFS with a small modification, run it over a real-world data set, and answer high-level questions analyzing the output. In doing so, you'll get some more practice thinking about data represented as a graph and working with graphs in code. Additionally, you'll get to practice turning pseudocode into real code that fits your context and data types. This is a common situation in programming: you often have a reference point for a similar problem and solution, but there are technical formatting issues that require you to adapt your implementation.

3.1. Context:

In the current global state of quarantine, the New York City (NYC) community started a trend in March in which people make noise at 7pm each day to cheer and show support for essential workers. You might imagine this happening as follows: one person starts making noise, then soon nearby neighbors who can hear that noise chime in and make noise of their own. This process repeats and the sound wave keeps spreading. This is the process that we're going to model our graph computation after – we're going to use BFS to simulate this sound wave spreading across NYC.

For our graph, each vertex represents a building and each edge that exists between buildings A and B represents that the clapping from building A is audible in building B and vice-versa.

The graph that you'll be working with is a real dataset published by the City of New York: https://data.cityofnewyork.us/City-Government/NYC-Address-Points/g6pj-hd8k. It contains close to 1 million NYC building address data points. By default, the graph you'll be working with contains a subset of the data, specifically focusing on ZIP code 10128.

Note: in this problem, edges are constructed following an approximation of how sound travels, but there are many more factors we could have considered. Even though we could come up with a more complex and realistic model, this setup works well enough for our purposes – and making simplifying assumptions of some type is often required when workin with real data.

3.2. Instructions:

Visit the repo here: https://gitlab.cs.washington.edu/cse373-20su-students/exercise-3 and clone it to get started. See https://courses.cs.washington.edu/courses/cse373/20su/projects/cse143review/setup/ for a reminder on how to clone a new repository. The `BFS.java` file is where you'll implement your BFS and do any extra analysis on the output. The other provided source file, `LoadAddressData.java`, is where we load the data and convert it to a graph format. If you want to If you want to optionally explore the data beyond the questions asked below, this is where you can make changes to the ZIP code or more generally choose what information gets included in the graph. Note that there are also tests for your BFS implementation located in `BFSTests.java`.

As mentioned in lecture, BFS and DFS by themselves don't do anything since they're just standard ways of traversing graphs. So to get useful information out of the BFS, you'll make a modification to record how many levels away each address is from the source (i.e. the starting source of the sound wave is distance 0, the buildings one edge away are distance 1, the next level out is distance 2, etc.). Your BFS implementation will return a `Map<String, Integer>` where the keys represent the building addresses and the values represent the associated distance. We recommend you use slide 31 of https://courses.cs.washington.edu/courses/cse373/20su/lectures/15.pdf as your starting point for implementing BFS.

3.3. Questions:

Answer these questions after implementing BFS. You should play around with your returned output in the main method to do some more investigation to answer the following questions. It's possible to obtain the answers to these questions with a few lines of code for each question.

Note: you do not need to turn in your code to Gradescope – just the answers to these questions.

(a) Which address(s) hears the clapping started by `212 E  95 ST, NY` last/the latest? If there are multiple addresses, then state all of them.

(b) Are there any addresses in the 10128 Zip code that the BFS sound wave is unable to reach when we start from `212 E  95 ST, NY`? What property of the algorithm could you examine to find this answer? For full credit, describe what property/properties you could check in terms of the general graph formulation and BFS result for this problem itself, and not your specific code.

(c) (EXTRA CREDIT) If you have time, feel free to explore the data some more; you could look up some of the points on Google Maps (copy-paste the addresses), or explore different ZIP codes, or explore the meaning of your BFS output more. Is there anything else you learned about the data while exploring that you want to share? Particularly involved answers will receive extra credit.