Assignment 10: Dijkstra
CS 758/858, Fall 2024
Due at **11:30pm on Mon Nov 4**

Implementation

The skeleton code on the course web page implements lousy algorithms for GPS navigation. You will add an implementation of Dijkstra's algorithm. You may reuse code from your previous assignments if you wish.

The input will be a directed graph with weighted edges, followed by a sequence of queries, each line specifying source and destination vertices. Your output for each query will be the path cost and the sequence of vertices along a least-cost path from the source to the destination.

We provide you with two test harnesses: one allows you to specifying the source and destination as textual addresses (eg, `Kingsbury Hall, Durham, NH`) and the other runs more comprehensive tests (on random graphs).

Testing

To test your program from the command line, something like this will provide a graph and a sample query to your program:

```
cat ~cs758/data/asn10/tiny.graph ~cs758/data/asn10/tiny.query | ./path_debug dijkstra
```

Most graphs we provide are in a compact binary format (`.graph`). Sample graphs are located in `~cs758/data/asn10/` on `agate.cs.unh.edu`:

gX-Y-#.graph a random geometric graph, where each of X vertices embedded on the plane are connected to their Y nearest neighbors

tiny.graph the example graph in the Dijkstra section of CLRS. `tiny.txt` shows the graph in human readable format, `tiny.query` is an example query on the graph.

kings.graph the area around Kingsbury Hall

durham, seacoast, nh, northeast, usa additional excerpts from OpenStreetMaps. Arc weights represent distance in meters. Only roads allowing motor vehicles are included.

In addition to a reference solution, we also provide:

graph-info prints info about the graph provided on stdin. For example, `usa.graph` has 17,595,665 vertices and 44,310,980 arcs. If a path is specified with `-o`, generates a PDF drawing of the graph.

ui-harness uses your program to drive a complete navigation service from Unicode addresses to PDF map. Calls OpenStreetMaps to resolve textual names to latitude and longitude. To get reasonable output, the graph provided on stdin should contain the addresses in the query. For example, after running

```
ui-harness -e ./path_debug -f 'Market Basket, Lee, NH' -t 'Hampton Beach, NH' < seacoast.graph
```

then `output.pdf` should contain a map showing the shortest path. Try `--help` to see the options. To reduce memory and time, when drawing large maps, only the portion of the map near the path is drawn. Since it uses an API run by a non-profit, please don't use the program as part of a script (see `https://operations.osmfoundation.org/policies/nominatim/` for the exact usage policy).

test-harness runs multiple queries and multiple algorithms and generates plots of the results. Does not verify that you found the shortest path, but this should be apparent from the plot. Generates some small graphs internally but needs you to specify a big connected graph with `-g`. Specify your executable with `-e` and the reference with `-r`. As in:

```
test-harness -e ./path -r ./path-reference -g ~cs758/data/asn10/g1m-8-1.graph
```

Note that there is a small probability of 1) depth-first search taking forever, even on a small graph, and 2) the random graphs being disconnected thus causing no paths to be found. If either of these low-probability events seem to occur, trying again is likely to solve the problem.

There are some additional programs provided if you want to fool around:

txt2graph converts a file like `tiny.txt` into a `.graph` file. Good for trying small examples.

subgraph will extract only vertices and arcs that are within a specified radius of a specified lat/lon.

random-graph I haven't optimized this so generating a large graph can take a while

osm2graph converts an `.osm.pbf` file (like those at `download.geofabrik.de`) into a `.graph` file. uses multiple cores and lots of RAM (USA took 120 GB).

Written Problems

Please put all your responses in a single `written.pdf` file in your electronic submission.

1. Briefly list any parts of your program which are not fully working. Include transcripts or plots showing the successes or failures. Is there anything else that we should know when evaluating your implementation work?
2. Compare the performance of `dfs_first`, `dfs_shortest`, and `dijkstra`. Why do the algorithms behave as they do? (include a plot)
3. Part a of problem 22–3 in CLRS.
4. (Those in 858 only) a) Show that the problem of finding the best sequence of trades from currency 1 to currency n in the arbitrage problem of the previous question exhibits optimal substructure. b) Show that, if there exists a schedule of fees such that c_k is the fee for making k trades, the problem does not necessarily exhibit optimal substructure.
5. Exercise 23.2–4 in CLRS.
6. What suggestions do you have for improving this assignment in the future?

Submission

Electronically submit using the script on agate (eg, `~cs758/scripts/sub758 10-undergrad your-asn10-dir`).

Remeber to verify that your program has everything it needs to compile when make is run in the submission directory.

Evaluation

In addition to correctness, your work will be evaluated on clarity and efficiency.

Tentative breakdown:

6 Dijkstra

4 written problems