Assignment 10: Dijkstra
CS 758/858, Spring 2014
Due Monday, November 10 by **1:40pm**

Implementation

The skeleton code on the course web page is the start of a program for GPS navigation. The input will be a graph with weighted edges (where the weights represent approximate travel times), a source vertex, and a destination vertex. Your output will be the list of vertices along a least-cost path from the source to the destination. We will provide you with sample iput, including a (rather out-of-date and mildly inaccurate) graph of the roads of the continental United States.

We recommend implementing Dijkstra's algorithm. You may reuse some of your code from previous assignments if you wish.

Testing

Some sample graphs are (or soon will be) located in `~cs758/data/asn10/` on `agate.cs.unh.edu`. The US graph has almost 24 million vertices and is in `usa_times.bin`. A smaller graph of only the New York City area is in `ny_times.bin` and is probably better for preliminary testing and debugging. We also supply the following programs. As usual, `-help` is likely to yield usage information.

`map-route-harness` runs your program, checks its output, and optionally draws the planned routes. For example:

```
map-route-harness ~cs758/data/asn10/usa_times.bin input --full -o out.eps
```

will run your program on a full map of the united states. The routes that will be planned are specified in the `input` file as pairs of vertex numbers (see `nearest-nodes` below for finding vertex numbers from longitude and latitude). The `-o` option can generate either .eps or .ppm files that show the coordinates as points and the routes that were planned as lines. The `--full` option tells the harness to display the entire map, otherwise the drawn map will be fit to the routes that were planned (note: the .eps files are 'compressed' and should be much smaller than the .ppm files). Finally, the `--html` option will create a web page showing a (slightly lower quality) image of the planned routes. The web page is located at `http://www.cs.unh.edu/~cs758/asn10/`*username*.

`nearest-nodes` finds the vertex numbers that are the closest in the data set to given longitude and latitude pairs. Simply run `nearest-nodes` with 1 argument (the data file). You will then be prompted for longitude and latitude pairs. Each pair will output the nearest vertex number in the data file along with other information. Hint: to find latitude/longitude in Google Maps, click on the map and then look for the gray numbers at the bottom of the popup that appears below the search box.

`random-routes` takes a map file as its input and generates random route files. This can be convenient for testing when you don't want to have to generate input files by using the nearest-nodes program. For example, to generate a quick input file with 4 routes you can use:

```
./random-routes /home/cs758/data/asn10/ny_times.bin -n 4.
```

`map-route-greedy` is a reference solution that finds a 'greedy' path by searching the neighbors that are nearest to the goal vertex first. This will find paths quickly but they will be suboptimal. Note, to use the greedy solver with the harness you must pass the harness the `--no-check` option so that it doesn't complain about sub-optimal solution lengths.

`map-route-djikstra` is a reference solution that uses Djikstra's algorithm to find optimal paths.

Written Problems

1. Is there anything special that we should know when evaluating your implementation work?
2. The input data also includes the latitude and longitude of each vertex. The skeleton code implements a greedy heuristic algorithm that goes to the neighboring vertex that is closest to the destination. This is not guaranteed to produce optimal routes. How much slower do the routes returned the greedy algorithm tend to be, compared to your optimal solution? How much faster does the algorithm tend to run?
3. Exercise 24.2–4 in CLRS. (By 'analyze', let's assume that Cormen et al. mean 'prove the time complexity.')
4. Do the first part of exercise 15.3–6 (the $c_k = 0$ case where you show optimal substructure) and part a of problem 24–3 in CLRS.
5. (Those in 858 only) Do part b of problem 24–3 and the second part of exercise 15.3–6 (the $c_k \neq 0$ case).
6. Exercise 25.2–4 in CLRS.
7. What suggestions do you have for improving this assignment in the future?

Submission

Electronically submit your source code using the script on agate (eg, `~cs758/scripts/sub758 10 your-asn10-dir`). Also hand in a listing of your source code (2 pages per page, as with `a2ps -2`), along with your written work, to the TA in class.

Evaluation

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

5 Dijkstra

5 written problems