ECE368 Project #3: Map Routing

Milestones and final are due at 11:59pm EDT on the due dates
Please read the late submission policy on the syllabus

Important notice:

1. Backup your code regularly.
2. Make sure your ECN account works. Inform the instructor/TA any issue early on. Please refer to the late submission policy on the course syllabus.
3. Debugging: the instructor/TA will provide abundant, specific suggestions for debugging. However, to ensure fairness among all students, the instructor/TA cannot debug code for individual students.
4. Please use the ECN machine from the very beginning to code, debug, and submit your program.
5. If you prefer a local editor, you can map your ECN home folder locally as a network drive, see here.

Sample binary

Description

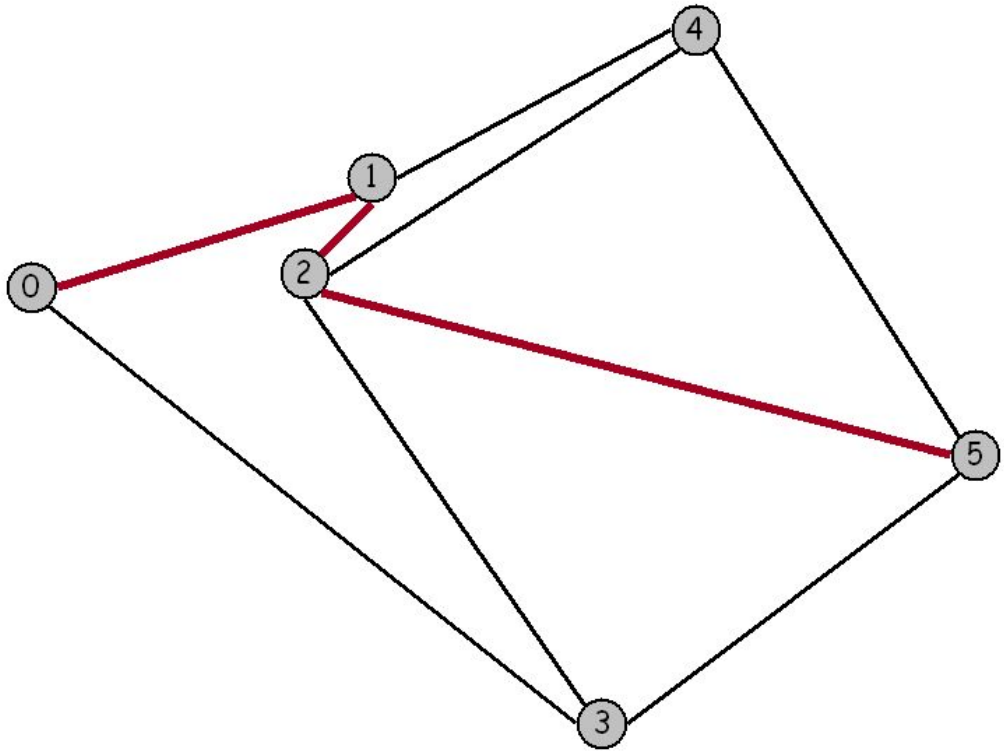
Summary

In this project, you will implement Dijkstra's shortest path algorithm for weighted undirected graphs. Variants/enhancements of this algorithm are widely used in geographic information systems including MapQuest and GPS-based car navigation systems. For a description of Dijkstra's algorithm, please see Wikipedia.

Input

You will be given two input files. The first input file will represent a map, which is an **undirected** graph whose vertices are points on a plane and are connected by edges whose weights are Euclidean distances. Think of the vertices as cities and the edges as roads connected to them. To represent such a map in a file, we list the number of vertices and edges on the first line, then list all the vertices (index followed by its x and y coordinates), and then list all the edges (pairs of vertices). For example, the input shown on the left below represents the map shown on the right below:

```
6 9
0 1000 2400
1 2800 3000
2 2400 2500
3 4000    0
4 4500 3800
5 6000 1500
0 1
0 3
1 2
1 4
2 4
2 3
2 5
3 5
4 5
```



Note: multiple nodes in a map may have identical coordinates.

The second input file contains a list of search queries with the first line containing the number of such queries (this is just to make your job of reading the file easier) and each of the following lines containing one query in the form of a source vertex and destination vertex pair. For example, the query input that is listed below contains two queries, one from node 0 to node 5, and the other from node 4 to node 3.

```
2
0 5
4 3
```

Goal: Given a map file and a query file as inputs, your goal is to compute the shortest path from each source listed in the query file to the corresponding destination using Dijkstra's algorithm. Your program takes the name of the map file and the name of the query file. Given these files, your program should then compute the shortest path for each query in the query file.

For example, given the files map6.txt and query2.txt, we expect your program (**shortestpath**) to produce the following output:

```
$ ./shortestpath map6.txt query2.txt
6273
0  1  2  5
5245
4  5  3
INF
6 7
```

In this example, your program prints out four lines, where:
Line 1 and 2 contain result for the 1st query in the input file `query2.txt`
    Line 1: the shortest distance from node 0 to node 5, which is 6273
    Line 2: all nodes on the shortest path from node 0 to node 5, delimited by a space character
Line 3 and 4 contain result for the 2nd query in the input file `query2.txt`

Line 3: the shortest distance from node 4 to node 3, which is 5245
Line 4: all nodes on the shortest path from node 4 to node 3, delimited by a space character
Line 5 and 6 contain result for the 3rd query in the input file `query2.txt`
Line 5 & 6: node 6 and 7 are disconnected and their distance is infinity.

Testing

	Map file	**Accompanying query file**
Case 1: A toy map	map5x5.txt	query5x5.txt
Case 2: The US continental map	usa.txt 87,575 vertices, each of which is a town; 121,961 edges, each of which is a road.	usa1.txt usa10.txt usa100.txt
Case 3: The California road network (source)	roadCA.txt 1,965,206 vertices, each of which is an intersection 5,533,214 edges, each of which is a road segment we found the sample binary crashed on this input, which may be due a bug inside the sample binary. If you encounter any issue on this input, report.	

Download test files

A sample output of Case 1 (path may not be unique):

```
7800
6 7 8 13 18 23
```

Big data challenge -- "Build Your Dungeon"

You submit a map file, so that your path finding algorithm outperforms all other "**big data challengers**" submissions on this particular map.

Please submit the map in a compressed format, e.g. yourmap.gz.

About the map:

Limitation: the map file must contains fewer than 500,000 lines. Under this constraint, we do not limit the maximum number of vertices or edges you may have.

The map file text format must be the same as specified above.

About queries:

TA will randomly generate 50 queries over the map.
In each query, both vertices are drawn independently and randomly from all the vertices your map contains.

About the total performance:

It is measured as the total runtime of all the queries.

About your program:

It must not depend on any external data file.
It must be the same one you submit for the regular requirements.
[For big data challenge only] The total executable binary size must be less than 100KB.

Deliverables

Milestone 1

Write a few paragraphs (around 400 words in total) to explain your overall approach to the problem. In particular, how are you going to parse the input files? What data structures are you going to use to represent graphs? Use your own words to briefly describe the Dijkstra algorithm.

To submit:

```
turnin -c ece368 -p proj3ms1 milestone1.pdf
```

Milestone 2

Submit the source file of a program (called **adjacent**) that takes in a map file and prints the adjacency list representation. For instance:

```
$ ./adjacent map6.txt
0: 1 3
1: 0 2 4
2: 1 3 4 5
3: 0 2 5
4: 1 2 5
5: 2 3 4
```

Each line starts with the index of a vertex, followed by the list of vertices that are adjacent to the first vertex.
To submit:

```
turnin -c ece368 -p proj3ms2 adjacent.c
```

Final

The project requires the submission (electronically) of the C code (source and header files) and a report detailing the results. A template is provided for the report. You should start with this template, fill in the various sections, and turn in a PDF version. Each report should not be longer than one page. To submit:

```
turnin -c ece368 -p proj3 file1.c file2.c ... file1.h file2.h ... report.pdf
```

where `file1.c`, `file2.c`, *etc.*, are the names of the files that contain your code. Make sure that you submit all your source files and also list them in the report. You will be responsible for any files that you forget to submit.

Grading

Your submission will be evaluated based on the correctness of your program, readability of your code, as well as the quality of your report. Your program will be compiled using the following command (if you use any special libraries, *e.g.,* math.h, which need additional compilation flags, explicitly mention those in your report):

```
gcc -Werror -Wall file1.c file2.c ... -o shortestpath
```

Rubrics

Correctness (75 points):

- Program doesn't crash on any inputs, up to the max number of 100,000 vertices with corresponding edges. (10 test cases, 1.5 points each)
- Gives the correct output. (10 test cases, 6 points each)

Performance (10 points):

- Speed (5 points): Is the speed within 120% of the provided binary.
  - If the provided binary takes 10 seconds, yours must run in 12 seconds. This will be tested on the largest test case
- Memory usage (5 points): Is memory usage within 120% of the provided binary.
  - If the provided binary uses 1,000 bytes on the program, yours can take a maximum of 1,200.
  - Will be tested using Valgrind total heap usage. This will be tested on the largest test case.

Report (15 points):

  - PDF format (5 points)
  - Clarity and quality of the report (10 points)

BONUS:

Submitting at least 24 hours early and scheduling an appointment with TA to explain how you did the project.

- Up to 5 points, must be able to answer algorithm, code, and conceptual questions on the project and background.

Final words

The Euclidean distance between vertex A with coordinates (x,y) and vertex B with coordinates (p,q) is given by $\sqrt{(p-x)^2+(q-y)^2}$. You are allowed to store the edge weights (Euclidean distances) as integers. You may also assume that the x and y coordinates of any vertex in the map file will be less than or equal to 10,000 and that the number of vertices in the map file will be less than or equal to 100,000.