Programming Assignment #2

Programming assignments are to be done individually. You may discuss the problem and general concepts with other students, but there should be no sharing of code. You may not submit code other than that which you write yourself or is provided with the assignment. This restriction specifically prohibits downloading code from the Internet. If any code you submit is in violation of this policy, you will receive no credit for the entire assignment.

The goals of this lab are:

- Ensure that you understand graphs and corresponding algorithms.
- Explore the trade-offs between memory and run-time efficiency, in particular, by designing and analyzing an efficient representation of a graph.
- Explore the application of Dijkstra's algorithm in real world problems.

In this project, you will design an algorithm for a graph problem and write a small report about it. We have provided the Java code skeletons to help you start. Your job is to fill in the blanks with your own code. Please read through this document and the documentation in the starter code.
If you add your own print statements to any part of the solution other than in `Driver.java`, you are responsible for removing them before submitting your solution. If your debug output interferes with the grader, you may lose points for an otherwise correct submission.

Problem Description

You are a manager of an international water transport company providing cargo transport service between countries. Your company has ports all around the world and they are connected by different rivers or the ocean. Some rivers are small and can only carry small boats while the ocean can carry large ships. Through many years of business, your company now has the information about how long it would take from one port to another (assuming there's no difference between port A to port B and port B to port A, i.e. **undirected graph**) and the carrying capacity for every section. With these information, you will design routing schemes for services.
There are N ports labeled 0 to $N-1$ and M waterways. And the total number of waterways is M. Each waterway R_{ij} is described with the time of travel t_{ij} and capacity c_{ij}. Of course, all times and capacities will be finite postive integers.

Part 1: Construct a graph [15 points]

Provided with the information about all the waterways, your program should construct a graph out of them (using adjacency list or adjacency matrix). The input file will be like this:

```
5 4
0 1 5 10
1 2 6 10
2 3 8 15
2 4 5 12
```

The first line gives the number of ports N and the number of edges M. In the following M lines, the first and second number are port indexes i and j, the third integer is the travel time t_{ij} between i and j and the fourth integer is the carrying capacity c_{ij}.
When reading the input file, `Driver` will call the `inputEdge()` function which will be implemented by you in the `Graph` class to store the graph in an adjacency list or adjacency matrix.

Part 2: Find the time optimal route [25 points]

A part of your business orders are time sensitive, they require your company to provide fast delivery. Given two ports A, B, design the time optimal route for it.
Pseudo-code for Dijkstra's algorithm is as follows:

```
1: Function Dijkstra(Graph, source,sink)
2: for each vertex v in Graph // Initialization do
3:     dist[v] := infinity ; // Unknown distance function from source to v
4:     previous[v] := undefined ; // Previous node in optimal path from source
5: end for
6: dist[source] := 0 ; // Distance from source to source
7: Q := the set of all nodes in Graph ; // All nodes in the graph are unoptimized - thus are in Q
8: while Q is not empty: // The main loop
9: u := vertex in Q with smallest distance in dist[] ;
10: if dist[u] = infinity then
11:     break;
12: end if
13: remove u from Q ;
14: if u = sink then
15:     break;
16: end if
17: for each neighbor v of u: // where v has not yet been removed from Q. do
18:     alt := dist[u] + dist_between(u, v) ;
19:     if alt < dist[v]: // Relax (u,v,a) then
20:         dist[v] := alt ;
21:         previous[v] := u ;
22:         decrease-key v in Q; // Reorder v in the Queue
23:     end if
24: end for
25: end while;
26: return dist[sink] ;
27: end Dijkstra
```

priority queue

Part 3: Find the capacity optimal route [30 points]

Part of your business orders have large weights of cargo and rivers with low carrying capacity cannot bear the burden of large ships with highly weighted freights. Given two ports A, B, find the path between source and destination that can permit the largest capacity ships to pass. Your design only need to output the bottleneck capacity of the optimal path.

Part 4: Write a report [30 points]

Write a short report that includes the following information:

(a) Explain the trade off between using adjacency matrix representation of graph and adjacency list representation of graph. Give a Big-O analysis of the memory space efficiency of your graph representation in terms of nodes N and edges M.

(b) Give a Big-O analysis of the runtime complexity and memory complexity of the algorithm

you used to find the time optimal route.

(c) Write pseudo-code for the algorithm you have implemented to find the capacity optimal route and prove the correctness of your algorithm.

(d) Is Dijkstra's algorithm able to solve graph problem with negative weighted edges? Explain why?

Instructions

- Download and import the code into your favorite development environment. We will be grading in Java 1.8. **It is YOUR responsibility to ensure that your solution compiles with the provided Driver.java in the standard Java 1.8 (JDK 8).** For most (if not all) students this should not be a problem. If you have doubts, email a TA or post your question on Piazza.
- **Please make sure your program works with the original Driver.java. Any changes you make in Driver.java will not be included in grading.**
- Driver.java is the main driver program. Use command line arguments to choose between your delivery time optimal algorithm and your delivery capacity optimal algorithm. Use -d for time optimal, -w for the capacity optimal algorithm, and input file name for specified input (i.e. `java -classpath . Driver [-d] [-w] <filename>` on a linux machine).
- Do not make changes to `Driver.java` and `Program2.java`. In `Graph.java`, you should add your own data structure and implement the method `inputEdge()` to construct the graph. You should also implement `getNeighbors()`, `findTimeOptimalPath()` and `findCapOptimalPath()`. You may add methods or other classes.
- In this assignment, the worst case is $N = 2000$ and it could be a full graph. Please make sure both of your programs will finish in 10s (which is more than enough for an efficient solution). The Maximum capacity of one edge is bounded by 10,000.(i.e., $c_{i,j} < 10000$). The total time of one path is bounded by the `Integer.MAX_Value` which is defined by Java.
- In this assignment, the input graphs are all connected, so you do not need to worry about connectivity in your program.
- For the `findTimeOptimalPath()`, you only need to output the total time of the optimal route from source port to destination port. For the `findCapOptimalPath()`, you only need to output the bottleneck capacity of the optimal route from source port to destination port.
- Do not add a package to your code; use the default package. If you have a line of code at the top of your Java file that says `package <some package name>;` that is wrong.
- Make sure your program compiles on the LRC machines before you submit it.
- We will be checking programming style. A penalty of up to 5 points will be given for poor programming practices (e.g. do not name your variables foo1, foo2, int1, and int2).

What To Submit

You should submit a single zip file titled `eid_lastname_firstname.zip` that contains:

- Graph.java
- other .java files you create
- your pdf report `eid_lastname_firstname.pdf`

Do not put these files in a folder before you zip them (i.e. the files should be in the root of the ZIP archive). Your solution must be submitted via Canvas BEFORE 11:59pm on the day it is due.