CS251 - Project 5:

Out: March 28, 2016 @ 9:00 am

Due: April 25, 2016 @ 9:00 am

Overview:

In project 5, you will be developing some tools for use by an airline company to help customers plan their flights and to identify which airports are most important to the company. In the first part, you will implement a system that reads in the airlines' flight offerings and stores it. Next, you will implement a system to find the cheapest tickets for your customers. Finally, you will generate a list of the airlines' most used airports.

Part 1a: Building the airline

In this part, you will read in which airports your airline works with and what routes they offer. Input will start with the number 1, representing part 1 of the project. You will be given the number of airports (n) that the airline works with followed by the number of routes (m) the airline offers. Following will be m lines giving the details of the routes including the two connected airports and the ticket price. Each of these lines will represent a **two-way** route between cities. That is, if there is a route from A to B, there is also a route from B to A at the exact same price. Here is a small example with only 3 airports, modelled after the routes between Indianapolis, Chicago, and Detroit:

```
1
3 3
IND ORD 163.30
ORD DTW 238.30
IND DTW 239.90
```

This information should be stored as a graph in your project. The exact way you store this is up to you; take a look at the class slides for suggestions on possible solutions. Also keep in mind that the way you choose to store the data will affect the difficulty of writing later parts of this project. Plan ahead!

Part 1b: Finding the right ticket

Now that you have read in the data, you will need to find tickets for your customers. In this project someone might want to fly between two airports in a way that requires a connection. That is, they may need to fly between multiple airports to get to their destination because a direct flight is not available.

This is where the search algorithms we've been working with come in. Implement Dijkstra's algorithm in a way that lets you search for the cheapest path between any two airports.

For the ticket-finding portion, we will make a few assumptions:

- We ignore aircraft capacity / flight times. You only need to find the cheapest route.
- There is only one cheapest route between airports
- The total ticket price is the sum of the prices of the legs.

Input/Output for this section will directly follow the network input, and is formatted as follows:
<source airport> <destination airport>
e.g
IND DTW

Your output should be in the following format:
<source airport> … <intermediate nodes> … <destination airport> <price to 2 decimal places>
e.g, for flying from IND to Washington DC (DCA) you might get something like:
IND DTW DCA 389.90

You should continue reading in flights until you read in the first token as END, meaning we are done generating tickets for passengers.

Part 2: Going on a tour

In part 2 you will be constructing a network for an airline in the same way as in part 1. However this time our objective will not be to generate individual tickets for passengers, but to find a way for a passenger to visit every city the airline services. It turns out that this problem, a variation of a famous problem known as the travelling salesman problem, cannot currently be solved perfectly in a reasonable amount of time. Therefore you will be implementing a simplified version of a solution to this problem, rather than an optimal one.

Your solution will be found in two steps. First, given the airline graph, you will construct a minimum spanning tree using either Prim-Jarnik or Kruskal's algorithm. The root of this tree will be provided in the input. Once you have the minimum spanning tree, you should do an Eulerian tour of the tree according to the following pseudocode:

```
void eulerTour(Minimum_Spanning_Tree G):
      Print root node
      For each child of the root, in alphabetical order:
            eulerTour(child)
```

Input / Output Format

All input/output for project 5 will go through standard in and standard out.

Part 1:

Input:

1. The integer "1" followed by two integers n and m, representing the number of airports and the number of routes respectively.

2. m lines in the format <source airport> <destination airport> <cost>
3. 1 or more lines containing ticket queries in the form <source> <destination>
4. A line containing “END”

Output:

1. m lines containing the optimal route between the source and destination in the format <source > … <intermediate nodes> … <destination> <total cost to 2 decimal places> or the line “not possible” if it is not possible to reach the destination

Part 2:

Input:

1. The integer “2” followed by two integers n and m, representing the number of airports and the number of routes respectively.
2. m lines in the format <source airport> <destination airport> <cost>
3. One line containing the airport that will be the root of the MST

Output:

The eulerian traversal according to the given pseudocode, or the line “not possible” if it is not possible to visit all cities.

Sample Test Cases

All input files will be well-formatted and valid test cases!

1 3 3 IND ORD 163.30 ORD DTW 238.30 IND DTW 239.90 IND ORD DTW IND DTW ORD END	IND ORD 163.30 DTW IND 239.90 DTW ORD 238.30
1 4 2 IND ORD 163.30 DCA DTW 150.00 IND DCA END	not possible
2 5 4 IND ORD 163.30 ORD DTW 238.30 DCA DTW 150.00 DTW LAX 300.00 4 DTW	DTW DCA LAX ORD IND

Programming Environment and Grading

Assignments will be tested in a Linux environment. You will be able to work on the assignments using the Linux workstations in HAAS and LAWSON (use your username and password). Compilation will be done using g++ and makefiles. You must submit all the source code as well as the Makefile that compiles your provided source code into an executable named "program".

Your project must compile using the standard g++ compiler (v 4.9.2) on data.cs.purdue.edu. For convenience, you are provided with a template Makefile and C++ source file. You are allowed to modify such file at your convenience as long as it follows the I/O specification. Note some latest features from C++14 are not available in g++ 4.9.

The grading process consists of:

1. Compiling and building your program using your supplied makefile.
2. The name of produced executable program must be "program" (must be lowercase)
3. Running your program automatically with several test input files we have pre-made according to the strict input file format of the project and verifying correct output files thus follow the above instruction for output precisely – do not "embellish" the output with additional characters or formatting – if your program produces different output such as extra prompt and space, points will be deducted.
4. Inspecting your source code.

Input to the programming projects will be via the command line:

```
program < input-test1.txt > output-test1.txt
```

The file output-test1.txt will be tested for proper output.

Important:

1. **If your program does not compile, your maximum grade will be 50% of the grade (e.g., 5 out of 10) -- no exceptions.**
2. **Plagiarism and any other form of academic dishonesty will be graded with 0 points as definitive score for the project and will be reported to the corresponding office.**

Submit Instructions

The project must be turned in by the due date and time using the turnin command. **Follow these directions closely, incorrect submission format can result in a lowered grade:**

1. Login to data.cs.purdue.edu (you can use the labs or a ssh remote connection).
2. Create a directory named with your username and copy your solution (makefile, all your source code files including headers and any other required additional file) there.
3. Go to the upper level directory and execute the following command:

   turnin -c cs251 -p project5 your_username

   (Important: previous submissions are overwritten with the new ones. Your last submission will be the official and therefore graded).
4. Verify what you have turned in by typing turnin -v -c cs251 -p project5
   (Important: Do not forget the -v flag, otherwise your submission would be replaced with an empty one). If you submit the wrong file you will not receive credit.