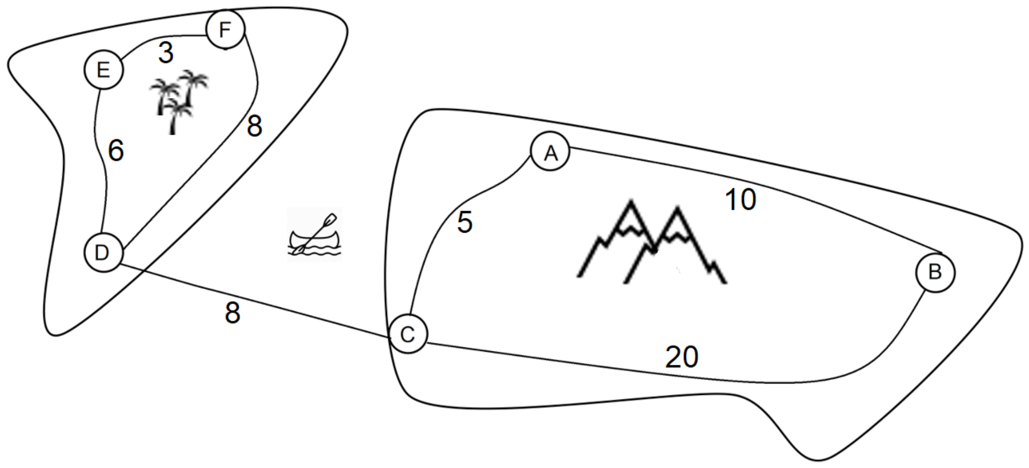
Project 5 - Tour Guide System
Due by 4/27/2018 (Hard deadline)

In project 5, you will be developing some tools for use by a Tourism company to help customers plan their trips and to identify which tourist spots are most important to the company. In the first part, you will implement a system that reads trip information and stores it. Next, you will implement a system to find the cheapest tickets for your customers. Finally, you will generate a list of company's most visited tourist places.

Milestone 1: Identification of connected components and separation Edges(bridges)
(Due 04/06/2018 at 11:59pm)

Part 1.1: Building the network

In this part you will read in the tourist places and the routes between them. Input will start with a number 1, representing part 1 of the project. You will be given the number of tourist places(n) and the interconnections among them (m), followed by m pairs of tourist places and the travel ticket price associated with them. Create a network (graph) from the given data and Identify sets of connected places. Two places are connected if they have a path between them. You can assume that each of the interconnection represent a two-way route. That is if there is a path from A to B, there is also a path from B to A at the exact same price. Here is a small example of the input modelled with 6 tourist spots and routes between them.



Example 1:

```
1
6 7
A B 10
B C 20
C A 5
E D 6
D C 8
D F 8
F E 3
```

This information should be stored as a graph in your project. The exact way you store this up to you; take a look at the class slides for suggestions on possible solutions.

Part 1.2: Identifying the critical routes

In this part you will identify the critical routes in the network that you build in part 1.1. Critical routes are analogous to separation edges (bridges) in a graph. A separation edge is an edge whose removal increases the number of connected components. Your output should be in the following format.

<number of connected components in the graph>

<number of separation edges>

<separation edges listed in lexicographical order>

i.e in the last example D C is a separation edge. Therefore, the expected output is:

```
1
1
C D
```

Also refer to the sample test cases provided at the end of the file.

Part 1: Input:

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

Part 1: Output:

<Number of connected components in the graph>

<Number of separation edges>

<Separation edges listed in alphabetical order>

Your program should work in O(m+n) to not hit a time-out.

Hint:

You can modify DFS to recursively find separation edges. To do that for each vertex v define the following evaluation function. It can be calculated in a constant time after visiting all the children of v and possible ancestors through the back edges.

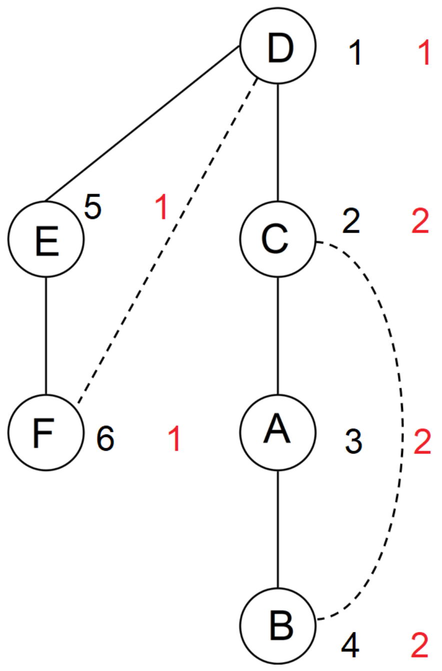
$$\text{evaFun}(v) = \mathbf{min}\ \{\ \text{disc}(v),\ \min\{\ \text{evaFun}(u) \mid (v, u) \text{ is a tree edge }\},\ \min\{\ \text{disc}(w) \mid (v, w) \text{ is a back edge}\}\ \}$$

in which disc(v) is the *discovery time* of vertex v.

A tree edge (s,t) is a separation edge if and only if evaFun(t) > disc(s).

Example:

In the Example 1, assume DFS started from vertex D. The following picture shows DFS tree in solid lines and the back edges in dashed lines. There are two numbers next to each vertex, the black one is the discovery time and the red one is the result of the evaluation function for that vertex. Here, the only tree edge whose discovery time of the parent is less than the evaluation function of the child is edge C D, so it is the separation edge of the given graph.



Milestone 2: Computing shortest path's using Dijkstra's Algorithm (Due 04/13/2018 at 11:59pm)

Part 2. Finding the cheapest ticket

Now that you have read in data, you will need to find tickets for your customers. In this project someone might want to travel between two places that requires a connection. That is, they may need to travel multiple cities to get to the destination as a direct route may not be available. You need to implement a Dijkstra's shortest path algorithm in a way that lets you search for the cheapest path between two different cities. For simplicity you can assume that there is only one cheapest route between two different cities.

Input/Output for this section will directly follow the network input and is formatted as follows:

<Source> <Destination>

e.g. IND DTW

You should continue reading in ticket queries until you read in the token "END".

Your output should be in the following format:

<source> <list of intermediate nodes> <destination> <cost up to 2 decimal places>.

Part 2: Input:

1. The integer "2" followed by two integers n and m, representing the number of places and the number of routes respectively.

2. m lines in the format <source> <destination> <cost>

3. 1 or more lines containing ticket queries in the form <source> <destination>

4. A line containing "END".

Part 2: Output:

Each line is 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. Number of lines is equal to the number of queries, and in the same order.

Milestone 3: Computing the Spanning Tree and Euler Tour (Due 04/27/2018 at 11:59pm – hard deadline)

Part 3: Going on a tour

In part 3 you will be constructing a network in the same way as in part 1. However, this time our objective will be to find a way for a passenger to visit every tourist place offered by the tourist company. 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 graph, you will construct a minimum spanning tree using either Prim-Jarnik or Kruskal's algorithm. The root of this tree (start point of the tour) will be provided in the input. Once you have the minimum spanning tree, you should do an Eulerian tour of the tree. An Euler tour on tree is a walk around the tree where each edge is visited exactly once, and the tour starts and ends at the same vertex. Euler tour in which we visit the nodes on left produces preorder traversal and when we visit the nodes on the right produces a postorder traversal of the tree. Eulerian tour of the tree can be implemented 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)
```

Part 3: Input:

1. The integer 3 followed by two integers ‘n’ and ‘m’ representing the number of tourist places and number of interconnections among them.
2. ‘m’ lines in the format of <source> <destination> <cost>.
3. One line containing the tourist place that will be the root of 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 Input / Output Format

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

1 3 3 A B 163.30 B C 238.30 C D 239.90	1 3 A B B C C D
2 3 3 A B 163.30 B C 238.30 A C 239.90 A B	A B 163.30 C A 239.90 C B 238.30

C A C B END	
2 4 2 A B 163.30 C D 150.00 A C END	not possible
3 5 4 IND ORD 163.30 ORD DTW 238.30 DCA DTW 150.00 DTW LAX 300.00 DTW	DTW DCA LAX ORD IND

Tests:

In the tests/ directory, there is a script "testall.sh" that you will use to test your project. You can also run it to test different parts of the project separately, for example "./testall.sh -p1" will test part-1 of the project only.

Setting up your environment:

Remote login to data by typing:
```
ssh <your-user-name>@data.cs.purdue.edu
```
Copy the initial project-5 files by typing:
```
cp /homes/cs251/Spring2018/project5/project5-src.tar .
tar -xvf project5-src.tar
cd project5-src
```

What's in the tarball:

You are provided with all the skeleton files you need to implement your project. The header files (*.h) contain the class declarations while the *.cpp files contain the definitions of the class methods. You are provided with some method declarations in the header files, but you need to add more methods to them (the provided header files are incomplete).

Turning in **(Due 04/27/2018 at 11:59pm)**

NOTE: You can use up to 4 late days to turn in project 5-3 (whether those late days include free late days or not). This means that no submission will be accepted after 5/1/2018.
To turn in your project (i.e. part 3) execute the following turnin command:
turnin -c cs251 -p project5-3 < your_folder_name>
(Eg: turnin -c cs251 -p project5-3 john_smith)
(Important: previous submissions are overwritten with the new ones. Your last submission will be the official one and therefore graded).
Verify what you have turned in by typing
turnin -v -c cs251 -p project5-3
(**Important**: Do not forget the -v flag, otherwise your submission would be replaced with an empty one).