CPSC 420 - Algorithms - Final Project

This project will give you some practice with graph algorithms and their related data structures. The two problems contained within are thinly-veiled graph algorithm problems, in need of your new found analytical capabilities.

Each problem is provided with sample input, a description of that input, and sample output corresponding to that input. Your programs will read in the graph description, build an internal representation (either adjacency-list or adjacency-matrix, or both), and apply a graph algorithm of your choosing to solve it.

This write-up is a primarily an analytical exercise, but coding will help (and is required). Some analyses you *may* want to consider are:

- Is the graph connected? Is it complete?
- Is it cyclic or acyclic?
- Does it contain negative weight edges? Does it contain negative weight cycles?
- Are the edges directed or undirected?
- Does the graph have an Eulerian path? (Note: An Eulerian path is one that connects two vertices by traversing through each edge once and only once.) Is there an Eulerian cycle? (that is, can you step through each edge of the graph exactly once and edge back at your starting point?)

(For those of you that are ambitious, pedantic, compulsive and thinking of trying the Hamiltonian cycle/path problem...*don't!*)

Of course, you can calculate all sorts of interesting results in graphs: minimum spanning trees, shortest paths, depth and breadth first searches.

Write-Up

For this project, you will need to read up on graph algorithms and determine which algorithms are applicable. **For each problem, your write-up should contain one section for discussion, one section for your code.**

The discussion section is the most important. You should start with an analysis of the problem. (What graph-theoretic properties of the problem can you exploit?) You may want to add an illustration, by drawing the graphs given in the sample input. Then describe your approach to solving it, what data structures you're using, what algorithm you are choosing to employ, and how that algorithm is modified to solve the problem at hand. Be sure to include an analysis of the asymptotic complexity of your algorithm/code.

Deliverable: Hard Copy, due at the FINAL EXAM. Tuesday, Apr 28, 11:00am.

Problem 1

Who's Got Game?

Massively multiplayer online role-playing games (MMORPGs) have their roots in MUDs (multi-user dungeons) which gained initial popularity in the 1970's. Here's an early excerpt:

> You are standing on the edge of a cliff surrounded by forest to the north and a river to the south. A chill wind blows up the unclimbable and unscaled heights. At the base of the cliff you can just make out the shapes of jagged rocks.
>
> `> go west`
>
> As you approach the edge of the cliff the rock starts to crumble. Hurriedly, you retreat as you feel the ground begin to give way under your feet!
>
> `> leap`
>
> You are splattered over a very large area, or at least most of you is. The rest of your remains are, even now, being eaten by the seagulls (especially your eyes). If you had looked before you leaped you might have decided not to jump!
>
> Would you like to play again?

In order to make things interesting for the player, the best RPGs try to allow for non-linear game-play, so a user's decisions during the game allow for a different experience than others who play the game. However, certain elements of the game must be performed in sequence. For instance, a key must first be discovered before a hidden chest can be opened and a new (more powerful) item collected. The key doesn't appear until you talk to the mage in the water town. After all, you wouldn't want the most powerful weapons in the game to be available right from the start! (Where would the challenge be?) Keeping up with these sequences is no easy task, especially when teams of designers are dreaming up new scenes and items all the time... who can keep up?

In order to help out aspiring dungeon masters, you want to determine if a given set of items or tasks will allow for strictly linear game play, many possible game play sequences, or if a particular set of relationships will result in an unfeasible game.

Input

There will be multiple test cases in the input.

Each test case will begin with a line with two integers n ($1 \leq n \leq 2,000$) and m ($1 \leq m \leq 50,000$), where n is the number of items or tasks, and m is the number of relationships between items.

On each of the next m lines will be two integers, d and u ($1 \leq u,\ d \leq n,\ d \neq u$) which indicate that collecting item or performing action d allows access to item or action u.

The input will end with two 0's on their own line.

Output

For each test case, print a single line of output containing 'Infeasible game' if the proposed gameplay sequences are impossible, 'Linear gameplay' if exactly one sequence is possible, or 'Nonlinear gameplay possible' if multiple arrangements are possible (no matter how many arrangements there are).

Sample Input

```
5 4
1 5
5 2
3 2
4 3
5 4
3 1
4 2
1 5
5 4
2 2
1 2
2 1
0 0
```

Sample Output

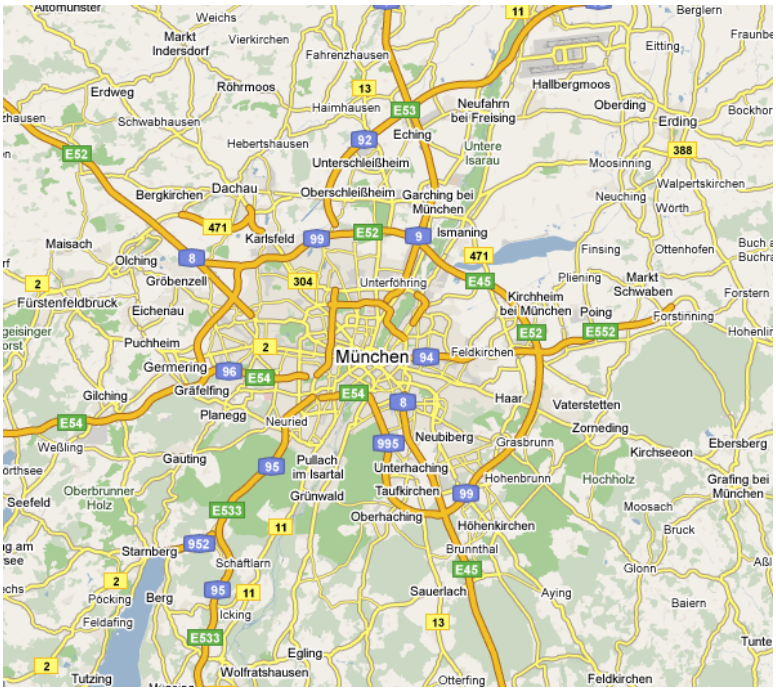
```
Nonlinear gameplay possible
Linear gameplay
Infeasible game
```

Problem 2

Fun, Fun, Fun "Auf der Autobahn"

Imagine spending a semester at the Technical University Munich, located (of course) in Munich, Germany. In order to get around this city, you could rely on the metro system, which provides great connections within the city and also far into the surrounding countryside. But then, you didn't come to Germany to only study Ohm and Siemens. No! Germany is also the country of Benz and Porsche. Furthermore, there is Germany's largest public entertainment park—the Autobahn! Combine the one with the other, and what do you get??? Fun, fun, fun, auf der Autobahn! Yes, can you imagine?!

And this is where this problem begins. It's about route finding while *increasing the fun factor*. Look at the map of Munich below, specifically at the street system. The thick lines are the Autobahn, the thinner streets are back roads. In order to get from any point A to any other point B, you can typically take many different routes, some consisting of only back roads, some including Autobahn segments. Of course, for fun purposes, you would always try to choose a route that would put you on the Autobahn, at least partially. On the other hand, you're a rational person, and you hate wasting time. That's why in the end you will choose the fastest route after all, and if it means that it is a back-roads-only route, well, there goes the fun... The question is now: Given any two points A and B, will you get to have fun when driving from A to B?



To solve this problem, you are given a map (in form of a graph), which contains all possible locations and intersections (these are the nodes of the graph), and the street segments (these are the edges). Each edge has a distance value associated with it, which specifies the length of this street segment in kilometers. Your average speed that you can drive on a street segment depends on whether it is a back road or the Autobahn. For this problem, assume that you can drive 80 km/h (about 50 mph) on a back road, and 160 km/h (100 mph) on the Autobahn. (Yes, this seems rather slow, as the real fun begins somewhere beyond 200 km/h. But then, it's just an average. . .)

Your task is to write a program that reads in the graph information and determines for several node pairs the total distance (in km) of the route, which takes the shortest time, as well as the number of Autobahn kilometers this route includes. In cases where there are several equally fast routes, your program should select the one(s) with the most Autobahn kilometers.

Input

The input starts with an integer specifying the number of nodes in the graph. In the next line, the node "names" (simple letter combinations) are listed. The next line contains the number of street segments that will be specified in the following lines. Each street segment consists of two nodes that this street connects, the length in kilometers of this segment (given as an integer), and whether it is a back road ("b") or Autobahn ("a"). Assume that all streets are two-way (i.e., bidirectional). Furthermore, for simplicity assume that only one edge can exist between any two nodes (i.e., there are no back roads running parallel to an Autobahn).

After the graph description comes the requests. At first, an integer specifies the number of route calculation requests. Each request then consists of two nodes.

Output

For each route request, your program should output the starting point and the destination point as well as the total travel distance in km and the total distance traveled on the Autobahn (also in km).

Sample Input

```
6
A B C D E F
7
A B 10 a
B C 10 a
D A 1 b
E B 5 b
F C 1 b
D E 10 b
E F 10 b
3
A B
D E
F D
```

Sample Output

```
A B 10 10
D E 10 0
F D 22 20
```

Extra Credit

Christopher Steiner's book, "How Algorithms Have Changed the World," is full of wonderful examples of contemporary applications of algorithms. However, it was written back in 2012—a lot has happened since then! (And of course, a lot happened that wasn't covered in the book.) For this extra credit assignment, write an essay paper (2 page minimum if single- spaced, 11pt font) on a topic, as if you were to add a chapter to Steiner's book. Start with an outline–the paper should come together pretty easily. You must use at least 3 references (including the book itself), cited properly. The best paper will be content that doesn't appear in the book, but if you want to formulate your paper around one of the ideas in the book, that is acceptable.

The paper can be research-based or a narrative prose. Also, give your paper a catchy title.

Deliverable

This paper is optional, and due via hard copy at the final exam. Minimum, 2 pages, 11pt font, single spaced, including references. (LaTeX 2 column prefered). A truly exceptional paper can receive up to 6 points.