Comp 3760 Assignment 3: World Traveler v3

Due date: 11:00pm Friday December 5, 2008

This is an optional assignment. If you choose to complete this assignment your grade on it can be used to replace a lower grade from a previous assignment. If you choose not to complete this assignment your course grade will be based on the previous assignments that you have completed.

Introduction

Billy and Sam decide to go on an around the world trip. They are not experienced travelers, and both have concerns. Billy's biggest worry is money. He is concerned about getting ripped off when buying souvenirs, because he always confuses the currency exchange rates, and the street vendors are so good at changing up currencies in the middle of a barter.

Sam is not so concerned about money. Her worry is being exposed to a deadly drug-resistant disease. Even though she plans to avoid the high risk areas, she knows that she will come in contact with other travelers who may have been exposed, and she fears that she could be at risk because of this.

Both Billy and Sam are computing students, so they decide to write a program that will help alleviate their fears. Billy's program will allow him to query the exchange rate for any type of currencies he comes across. Sam's program will allow her to identify everyone who was potentially exposed to a disease after one or more of their fellow travelers gets sick.

Your Challenge

1. Write one of the following programs:
   a. ***Billy's Program*** : This program uses assertions and queries to determine exchange rates at any point in the journey. The specific requirements for ***Billy's Program*** are given below.
   b. ***Sam's Program*** : This program uses assertions and queries to determine contagious people at any point in the journey. The specific requirements for ***Sam's Program*** are given below.
2. Prepare a two to three page document that describes the design and algorithms used in your program. You should justify your choice of algorithms and how each was implemented. This document will be used during the grading of your assignment. The clearer and more articulate your are, the higher your grade will be (provided that what you say is in fact correct).

Billy's Program

The exchange rate between two currencies **`A`** and **`B`** is expressed as two positive integers **`m`** and **`n`**, and says that **`m`** of item **`A`** is worth **`n`** of item **`B`**. For example, 2 Euros might be worth 3 Cdn dollars. (Mathematically, 1 Euro is worth 1.5 Cdn, but fractions were always difficult for Billy, so he decided to always express exchange rates using integers.)

The challenge that Billy has is to write a program that, given a list of exchange rates, calculates the exchange rate between any two currencies.

The input is from a file that contains one or more commands, followed by a line beginning with a period that signals the end of the file. Each command is on a line by itself and is either an assertion or a query. Billy plans to add new assertions as he travels and when he discovers what the local exchange rates are. He will use a query whenever he needs to determine an exchange rate.

An assertion begins with an exclamation point and has the format

```
! m currency_a = n currency_b
```

where `currency_a` and `currency_b` are distinct item names and **`m`** and **`n`** are both positive integers less than 100. This command says that **`m`** of `currency_a` are worth **`n`** of `currency_b`. A query begins with a question mark, is of the form

```
? currency_a = currency_b
```

and asks for the exchange rate between `currency_a` and `currency_b`, where `currency_a` and `currency_b` are distinct currencies that have both appeared in previous assertions (although not necessarily the same assertion).

For each query, the program needs to output the exchange rate between `currency_a` and `currency_b` based on all the assertions made up to that point. Exchange rates must be in integers and must be reduced to lowest terms. If no exchange rate can be determined at the time a query is made, the output must display question marks instead of integers. Format all output exactly as shown in the example.

Note that the following constraints apply to this program:

- Currency names will have length at most 20 and will contain only lowercase letters (a-z).
- Only the singular form of an currency name will be used (no plurals).
- There will be at most 600 distinct currencies.
- There will be at most one assertion for any pair of distinct currencies.
- There will be no contradictory assertions. For example, "2 cdn = 1 euro", "2 euro = 1 usd", and "2 usd = 3 cdn" are contradictory.
- Assertions are not necessarily in lowest terms, but output must be.
- Although assertions use numbers less than 100, queries may result in larger numbers that will not exceed 10000 when reduced to lowest terms.

There are multiple tests per input file. Each test ends with a line containing a single dot (".").

Example input for Billy's program:

```
! 6 cdn = 15 franc
! 47 euro = 9 lira
? franc = cdn
? cdn = lira
! 2 franc = 1 euro
? lira = cdn
.
```

.***Example output from Billy's program:***

```
5 franc = 2 cdn
? cdn = ? lira
45 lira = 188 cdn
```

Sam's Program

Sam is pretty diligent about keeping track of the people she meets when traveling. She know that tourists almost always travel in groups, and that individual travelers often mix with a variety of tour groups. When traveling with a group, the group members tend to intercommunicate with each other frequently, which of course causes they all to be exposed to a disease if someone else in the group has been exposed. To make sure that she can identify at risk travelers, Sam intends to keep a detailed member list for each tour group. To be on the safe side she has decided to make the following rule:

If a member of a group has been exposed to a disease, all the other members of the group are also assumed to have been exposed.

This all seemed logical to Sam, until she realized that the intermingling of groups makes it potentially difficult to identify all the at risk travelers after a traveler has been exposed. The purpose of Sam's program is to identify these at risk traveler's as soon as someone is identified as having been exposed to a disease.

Furthermore, to protect peoples privacy Sam has decided to assign everyone a numerical ID. She will always be identified as 0, and Billy will be always be 1. Other travelers will be assigned integer ID's starting at 2.

The input is from a file that contains one or more commands, followed by a line beginning with a period that signals the end of the file . Each command is on a line by itself and is either an assertion or a query. Sam plans to add new assertions as she travels and when she discovers who is a member of which tour group. She will use a query whenever someone gets sick, to determine who else might have been exposed.

An assertion begins with an exclamation point and has the format

```
! { ID1 ID2 ID3 ... IDn }
```

where each integer between the curly braces identifies a different traveler. The above command says that all the travelers listed were on a tour together. A query begins with a question mark, is of the form

```
? * ID
```

and asks the program to determine the number of travelers who are at risk if the traveler known by **`ID`** was exposed to a disease, and to output their **`ID`**'s. This query will cause the number of at risk travelers to be output, followed by a colon, followed by the **`ID`**'s (sorted in ascending order) of at risk travelers, exactly as shown in the example below.

Note that the following constraints apply to this program:

- There will be at most 10000 travelers.
- There will be at most 100000 tour groups.
- A traveler can be a member of any tour group.
- A traveler can be member of multiple tour groups.
- A traveler does not need to be a member of any tour groups.

There are multiple tests per input file. Each test ends with a line containing a single dot (".").

Example input for Sam's program:

```
! { 0 1 }
! { 10 13 14 11 18 }
? * 11
! { 42 0 }
! { 99 1 }
? * 99
? * 9999
.
```

.***Example output from Sam's program:***

```
5: 10 11 13 14 18
4: 0 1 42 99
1: 9999
```

Other Requirements and Guidelines

1. This is an individual assignment. You must work by yourself.
2. For this assignment you must use Java.
3. You are free to use any of the data structures or algorithms in the Java Collections Framework, or you can develop your own structures if you prefer.
4. You are not allowed to use algorithm implementations and/or data structures that you find on the internet. You must write your own code. The only exception is a Graph class; you may use one of these found on the internet as long as you reference where you found it.
5. Each source code file must include a header block listing (minimally) the author(s), date, and description of the file contents.
6. Your program will be evaluated against instructors test files, which will not be provided in advance. It is up to you to understand the requirements and make sure your solution meets them.
7. Programming style, design, technique, and inline documentation are up to you. This is not a course in software engineering or software design, however, you are expected to write code that is well designed and that can be easily understood by someone else. Also, since this is an algorithm course, we will be looking at the overall efficiency and sophistication of the algorithms and structures that you use.
8. The requirements on the assignment are subject to change. Any clarifications or changes will be announced in lecture, and documented in a new version of this file - which will be placed on webct. It is your responsibility to attend lecture and/or monitor webct so that you are aware of any assignment updates or changes.

Submission Details

To receive full marks, your assignment must be submitted to WebCT before ***11:00pm Friday December 5, 2008.*** WebCT will be configured such that assignments submitted after this time will ***not be accepted***.

There will be no exceptions to this policy, regardless of your reason. It is the policy of the instructor to ensure that regular coursework (eg: assignments) do not interfere with studying or preparation for exams.

Your submitted assignment will consist of a single zip file. When your submitted zip file is unzipped, it should create a directory with your real name (initial & last name). For example, unzipping my assignment will create a directory called: rneilson (or something similar).

Within this directory I expect to find a directory containing all the Java source files required to build your program. Your submission must not include java SDK files, and must meet the following criteria:

a. Your "main" function must be in a class called Main, in a package called "travel", and in a subdirectory called "travel" under the directory created when your submitted file is unzipped. In the above example, this means main should be in the file rneilson\travel\Main.java
b. Note that packages correspond to subdirectories. This means you need to put the line: package travel; in Main.java. You will probably need to do the same with the other source files.

How do I test your assignment?

1. I don't use NetBeans. I just compile your program using javac. Your program must compile fine when I use the following commands to compile it: javac travel\Main.java (assuming I am in directory rneilson).
2. Next I will execute your program, passing the name of one of my input files, and redirecting the output to a file. For example: java travel/Main a3data.txt > rneilson_a3_out.txt

Grading and Evaluation

Your program will be graded according to the marks distribution shown below. The instructor's test data is designed to make sure you understood and followed the requirements. Don't assume anything!

Programs that do not compile will receive a grade of zero.

Programs that are submitted late will receive a grade of zero.

Programs that do not read the input file (ie: crash on input) will receive a grade of zero.

Programs that do not conform to the submission instructions will receive a grade of zero.

Programs that do not accept the file name as a command line argument will receive a grade of zero.

Programs that take longer than 180 seconds elapsed execution time will receive a grade of zero (we will assume your program has an infinite loop and does not work).

Mark Distribution

1: Program Correctness 60%

- marks will be assigned based on the correctness of the output produced by your program

2: Choice of Algorithms 20%

- marks will be assigned based on your choice of algorithms and data structures; more efficient solutions will receive grades

3: Program Design 20%

- marks will be assigned based on the design of your program; your program must use a reasonable level of decomposition and must be laid out and commented such that it is easy to understand and follow

Summary of Changes to this File

Note: if there are errors, omissions, or ambiguities in this file, please let me know by email and I will update the assignment description.