Assignment 3

Overview

This assignment combines several different data abstractions and algorithms that we have covered in class, including priority queues, online disjoint set operations, hashing, and sorting. The problem is a simplification of the problem of inferring relationships from social networking data.

In this assignment, you are to write a program that processes communication data. The input is obtained from a file named on the command line. The file consists of lines of text of various types. A ***data line*** consists of the word `data` followed by two *distinct* telephone numbers followed by a positive integer. Each telephone number is in the form *xxx-xxx-xxxx*, where each '*x*' is a decimal digit. The positive integer represents an amount of money transferred from the first number to the second number, in whole dollar amounts. For example, the line

```
data    807-444-2100    201-222-1200    15400
```

represents the fact that $15,400 was transferred from the owner of 807-444-2100 to the owner of 201-222-1200. The number more generally represents the strength of the relationship between the two numbers. A telephone number x is ***linked*** to a telephone number y if either x and y are the same number or a transfer of money was made between them. Two numbers x and y are ***connected*** if x and y are ***linked*** or if there is a number z such that x and z are linked and z is connected to y. A ***cohort*** is the largest set of telephone numbers such that every number in the set is connected to every other number in the set. This implies that if a telephone number is not in the cohort, then no transfer was ever made between that number and any number in the cohort. It also implies that every pair of cohorts is disjoint, and that no cohort is empty.

From these definitions, we can conclude that the *connected* relation is symmetric, reflexive, and transitive, and hence, an equivalence relation, and that the set of all cohorts forms a partition of the set of all telephone numbers.

The ***size*** of a cohort is the cardinality of the set (the number of telephone numbers in it.) The ***volume*** of a cohort is the total amount of money transferred between members of the cohort. From the definition of connectivity, cohorts of size 1 have volume 0. The ***activity*** of a cohort is the volume divided by the total number of unordered pairs of distinct numbers in the cohort, which is $N(N-1)/2$ where N is the size of the cohort; it measures the average amount of money that has been transferred between any pair of members of the cohort.

When a data line is read, the program must update all cohorts to reflect possible new relations. The update must include any changes in the size, volume, and activity of all cohorts. Every cohort must be identified by a ***positive integer ID***, called its ***cohort-id***.

The other types of lines that may appear in the input file are as follows. For each command, your program must take the action indicated in the right hand column. In the table, **boldface** indicates command keywords and *italics* represent placeholders for data. `Fixed font` indicates actual data.

Command	Argument Type	Description
`find`	*phone-number*	The program should display the ID of the cohort to which the phone-number belongs, in the form *phone-number*`:` *ID* If there is no such number, it should display *phone-number*`: no cohort`

Command	Argument Type	Description
`members`	*cohort-id*	The program should display a list of all telephone numbers in the given cohort, one per line. If there is no such cohort, it should display a line indicating there is no such cohort.

`max`	one of the following strings:
	`activity`
	`size`
		The program should display the cohort-id, activity, size, and volume of the cohort having the maximum value of the given property. If two or more cohorts have the same maximum value, then all that are maximal should be displayed. If there are no cohorts, it should display a line indicating there are no cohorts.

`cohort-ids`		The program should display a list of all cohort ids in order of increasing id.
`info`	*cohort-id* / `0`	The program should display 1. the cohort-id, 2. the activity, 3. the size, and 4. the volume of the cohort whose cohort-id is given. If no such id exists, it should display an error message. If the argument is 0, then the program should display this information for all of the cohorts, sorted by increasing cohort-id. These metrics should be listed on a single line, in the above order, separated by tab characters.

Further Details About Input And Output

The program must get the name of the input file from its only command line argument. Specifically, the program must parse the command line and extract the input file name from the first command line argument. If there is no command line argument, it must report this as an error and exit. If the file name is supplied but cannot be opened for any reason, it must report this error and exit. The program is to put all output on the standard output stream, not in any file; to repeat this, it is not to place its output into a file.

Telephone numbers will have no space in them, and will have hyphens as indicated above. Tokens in the input line may be preceded or followed by any number of white space characters. The number of distinct telephone numbers will not exceed 5000. From the preceding information, you can see that there are a total of six different words that can start a line in the file. The program should catch any line that does not start with one of these words and display an error message on the ***standard error stream*** for each such line that it finds. It can assume that the remainders of all lines are in the proper form. It should continue to the next line of the file after flagging the bad commands.

Implementation Requirements

1. Telephone numbers are strings, not integers. The union-find algorithm description is based on sets whose names are array index values. How are strings associated with these index values? When a telephone number is given to the program, the program needs to store it and access it easily. Although you could use a vector and some type of $O(\log n)$ look-up structure for it, the proper solution is to devise a hash table and hashing scheme that will allow nearly $O(1)$ look-ups. For the union-find algorithm, each new set should occupy the next free position in an array, which suggests that some

type of object must keep track of that position and associate it with each new telephone number, which is a set of size 1. The idea is therefore that, given a telephone number, you associate a new set id with it and store it in such a way that you can retrieve the set id from the telephone number in $O(1)$ time. But you also need to find all of the telephone numbers in a given set easily, which suggests that the array used for the union-find algorithm does not store just simple integer values.

2. You must use the union-find algorithm, with path compression and union-by-size, to solve this problem. All set manipulation must be carried out in a suitably defined class.

3. You must use a priority queue for finding the maximum cohort when the `max` command is issued. Since the maximum may be with respect to one of two different properties, this suggests using some type of indirection in the queue nodes. When sets are updated as a result of reading new data, the priority queue may become invalid because one set may become larger or smaller with respect to one or more properties than before, and because a new set may be created, or an existing one removed. Although you could update the priority queues after each data operation, you could use the on-demand approach of leaving them in their old states, and only when the `max` command is issued, heapifying and find the maximum element. For the sake of efficiency, you might consider keeping a 'dirty' flag to test if it is necessary to re-heapify. This is your choice; you can either update your queue after every operation that changes the properties of the sets, or only "on-demand."

4. You must use heapsort to sort cohorts for display.

5. All interaction with the file system and the terminal must be performed by the main program. Classes are free to create strings to pass to the main program, or to write onto ostream objects that are passed as parameters.

6. Activity must be computed with floating point division and reported to two decimal places of precision. Size and volume are whole numbers.

Programming Constraints

- You are not permitted to use any features of C++-11; the program must compile with the GNU C++ compiler in the lab, version 4.7.2 without the need to specify C++-11.

- Your program must conform to the programming rules described in the Programming Rules document on the course website. It is to be your own work alone.

Testing Your Program

You should design your own input files and test your program using your own input. You should carefully check that the output of your program is correct for the inputs you gave to it. Include files with bad lines, files with no lines, files that cannot be opened, files with all kinds of spacing, and so on. Include data that creates several cohorts of equal size, volume, and activity.

Grading

The program will be graded based on the following rubric.

- If the program does not compile on a cslab machine, it receives only 25%.

- For programs that compile:

  - Correctness 40%
  - Conformance to Implementation Requirements 30%
  - Design (modularity and organization) 10%
  - Documentation: 10%
  - Style and proper naming: 10%

Submitting the Assignment

This assignment is due by the end of the day (i.e. 11:59PM, EST) on May 12, 2014. Create a directory named named *username*`_hwk3`. Put all project-related source-code files into that directory. ***Do not place any executable files or object files into this directory.*** You will lose 1% for each file that does not belong there, and you will lose 2% if you do not name the directory correctly. With all files in your directory, run the command

```
zip -r username_hwk3.zip ./username_hwk3
```

This will compress all of your files into the file named `username_hwk3.zip`.

Before you submit the assignment, make sure that it compiles and runs correctly on one of the `cslab` machines. Do not enhance your program beyond this specification. Do not make it do anything except what is written above.

You are to put your zip file into the directory

```
/data/biocs/b/student.accounts/cs335_sw/projects/project3
```

Give it permission 600 so that only you have access to it. To do this, `cd` to the above directory and run the command

`chmod 600` *`username_hwk3.zip`*

where *`username_hwk3.zip`* is the name of your zip file.

If you put a file there and then decide to change it before the deadline, just replace it by the new version. Once the deadline has passed, you cannot do this. I will grade whatever version is there at the end of the day on the due date. You cannot resubmit the program after the due date.