CSCI 3901 Assignment 3

Due date: Monday, November 7, 2022 at 11:59pm Halifax time. Submissions through your CS gitlab repository in https://git.cs.dal.ca/courses/2022-fall/csci-3901/assignment3/????.git where ???? is your CS id.

Problem 1

Goal

Work with inter-related information.

Background

We typically use version control systems like git to manage the changes to code in large software projects. Every change is logged as a separate commit point in the version control system and all files involved in one change are typically part of a single commit. Give a set of commits, we can seek to infer some of the interdependencies between our code files and some of the habits of the programmers who make the commits. Insights from these inferences may then inform how we restructure our code or if we change or software development processes.

We consider a version control system in which each commit contains

- The name of the person doing the commit
- The time and date when the commit is made
- The branch name to which the commit is made
- The set of file names that are part of the commit
- A string that is either a feature number or a bug number that speaks to the reason for the commit
- A comment

Features and bug fixes can be involved in multiple commits.

You will develop a Java class that will record and report on the commits in a project.

Problem

Implement a class called CommitManager that will accept information about commit operations in a version control system and will then report on the status of those commits.

For the sake of this assignment, time and date combinations will be a single integer that is the number of minutes from the time when the version control system was first deployed. We will also omit references to branch names and to comments to simplify this assignment. Last, identifiers for bugs or features are differentiated by the first letter of the string: bug numbers begin with “B-“ while features begin with “F-“.

Your CommitManager class will include the following methods, at a minimum:

- Constructor that accepts no arguments
- void addCommit(String developer, int commitTime, String task, Set<String> commitFiles) throws IllegalArgumentException
- boolean setTimeWindow (int startTime, int endTime)
- void clearTimeWindow( )
- boolean componentMinimum(int threshold)
- Set<Set<String>> softwareComponents ( )
- Set<String> repetionInBugs (int threshold)
- Set<String> broadFeatures (int threshold)
- Set<String> experts (int threshold)
- List<String> busyClasses (int limit)

The constructor sets up whatever your class needs.

The main reporting methods are softwareComponents, repetitionInBugs, broadFeatures, experts, and busyClasses. These four reports deal with commits made within a given time frame (set by setTimeWindow() and clearTimeWindow()) and often relate to the components that we detect in the system, which is goverened by componentMinimum( ).

Data methods:

addCommit(String developer, int commitTime, String task, Set<String> commitFiles) throws IllegalArgumentException

Define one commit to the system. One would typically get this commit information from the version control system, but we'll get it through a method for this assignment.

The parameters define a single commit:

- developer – the name of the person doing the commit
- commitTime – the time that the commit operation was made (in minutes since the start of the installation of the version control system)
- task – an identifier for the reason for the commit, as either fixing a bug or implementing a feature. All commits with the same task identifier are considered elements of the same bug or feature
- commitFiles – the names of the files that are involved in the commit

Commits are expected to come in chronological order.

Control methods:

boolean setTimeWindow(int startTime, int endTime)

Limit all reporting operations to commits that happened between the given start and end times (including the start and end times in the time range). This time window is in effect until it is cleared or a new time window is set.

Returns true if the time window is accepted and false if the time window is not one that is in operation.

clearTimeWindow( )

Reset the time window used for reporting operations. After clearTimeWindow, any reporting operation will consider all commits until another window is successfully set with setTimeWindow()

boolean componentMinimum(int threshold)

Some reporting operations, like "experts" and "components", deal with conceptual components of our system such as the user interface, the database component, or the permission management component. A good program design will have break out these components in the high-level design. In Java, these components may become "packages" within the code.

The version control system does not know about these components. Consequently, we will infer the components from the set of commits.

In a good design, any bug or feature will primarily deal with classes in the same component. We count the number of times that files occur together in commits for bugs or for features. Files that occur together often are part of the same component. Transitively, two pairs of classes that both occur together often and that share a class then form a larger single class. For example, Figure 1 depicts 8 classes (A through H) and shows how often classes have been seen together in the same commits within the current time window. If we have a threshold value of 5, meaning classes must be in 5 or more commits together to be considered part of the same class, then Figure 1 has 3 components: {A, B, D, F}, {C}, and {E, G, H}. Conversely, if the same figure is given a threshold value of 7 then we have four components: {A, B, D, F}, {C}, {E, H}, {G}.

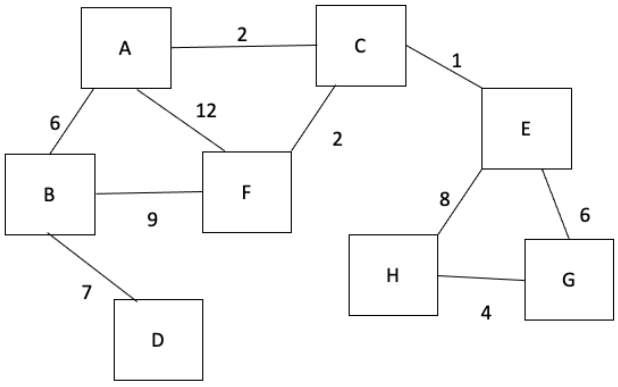

Figure 1 Class co-occurrence in bugs and features. A threshold of 5 gives software components {A, B, D, F}, {C}, and {E, G, H}

The componentMinimum() method defines the threshold to use for defining software components in the rest of the reporting methods.

It returns true if the minimum is accepted and false if the minimum is not accepted.

Reporting methods:

Each of the other methods operates as follows:

Set<Set<String>> softwareComponents ( )

Return all the software components identified by the commits in the current time window and considering the defined threshold for files being in the same commit to be considered as part of the same software component.

Examples of softwareComponents appear in the description of the componentMinimum() method.

Set<String> repetionInBugs (int threshold)

Return a set of bug fix task identifiers that include one or more files among all fixes for the bug at least "threshold" times. For example, consider bug "B-1328" that has 3 commits that include file A, 5 commits that include file B, and 2 commits that include file C. If repetitionInBugs is called with a threshold of 3 then the method returns "B-1328" in the set. If repetitionInBugs is called with a threshold of 8 then "B-1328" does **not** appear in the returned set.

Set<String> broadFeatures (int threshold)

Return the set of feature identifiers whose commits within the given time window includes one or more files from at least "threshold" components. Software components are as defined by the sofwareComponent() method.

Set<String> experts (int threshold)

Return the set of developer names whose commits within the given time window touch on one or more files from at least "threshold" components. Software components are as defined by the softwareComponent() method.

List<String> busyClasses (int limit)

Return the "limit" most seen file names, in descending order of use, that appear in commits within the given time window. In the case of ties, break ties alphabetically by file name. If a tie in number of occurrences of files happens at the "limit" boundary then include all file names with that tied number of occurrences.

In that latter situation, suppose that we have a limit of 2 and we have 5 files with the following occurrences: A – 10, B – 15, C – 10, D – 1, E – 3. In this case, the returned list is [B, A, C]. The list is longer than 2 because stopping at 2 would end with A, but C has the same number of occurrences as A and so C is also included in the list.

Assumptions

You may assume that

- All developer, file, and task identifiers are case sensitive.

Constraints

- You may use any data structures from the Java Collection Framework.
- If you use a graph then you may not use a library package to for your graph or for the algorithm on your graph.
- If in doubt for testing, I will be running your program on timberlea.cs.dal.ca. Correct operation of your program shouldn't rely on any packages that aren't available on that system.

Notes

- Pick the order in which you will implement your methods. Implement the simpler ones first to get familiar with your data representation.

Marking scheme

- Documentation (internal and external), program organization, clarity, modularity, style – 4 marks
- Thorough (and non-overlapping) list of test cases for the problem – 3 marks
- Efficiency of your data structures and algorithms for the given task, as explained by your own description of that efficiency (part of external documentation) – 2 marks
- Marks for each of the requested reporting methods (the control methods are marked indirectly by their effects on the reporting methods:
    - softwareComponents – 5 marks
    - repetionInBugs – 3 marks
    - broadFeatures – 3 marks
    - experts – 3 marks
    - busyClasses – 3 marks