Software Development Concepts

Assignment 3

Submitted by –

Name - Alen Santosh John

Banner ID – B00930528

Email – al283652@dal.ca

Assignment External Report

Problem -

Overview –
This program implements a version control manager that lets the user analyze various commits made. It can categorize the commit into multiple components and perform operations revolving around it.

Files and external data –
The program consists of the following java files -

1. **BroadFeatures.java** –
2. **BugTask.java** –
3. **BusyClasses.java** –
4. **CommitManager.java** –
5. **CommitObject.java** –
6. **Expert.java**
7. **GraphWeightCalculator.java**
8. **MatrixRelations**
9. **PopulateMatrix**
10. **VertexRelationBuilder**

Data Structure -

A graph has been implemented for solving this problem. The graph has been represented using an adjacency matrix. Maps, Lists and Sets are the most common data structures that have been used for this implementation.

The efficiency of the Algorithm.

This implementation has maintained the complexity at o(n^2). The reason for an n-square complexity is the usage of two loops in various methods. The time complexity was cubic (o(n^3)) before as three loops were involved to access the data. The complexity was reduced by properly handling data during the addCommit method.

Data could be stored in a Map that could be identified by the task id. That is all the commits could be stored in a map of String and ArrayList, where the ArrayList has all the commits with

the same task id is stored. To access the commit time or the set of files we have to use three for loops.

The implementation has methods that are invoked inside the add commit method. These add the required data to their respective global variables in the helper classes. This eliminates one layer of the loop as data is fed one by one in the add commit method.

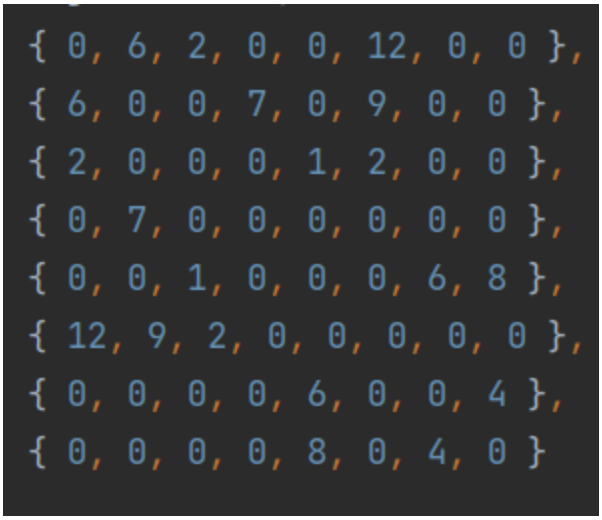
Essential Algorithms and Design Elements

Key Algorithm Overview -

1. All the commits are stored in a list of Commit objects. While invoking the addCommit method the data object is modified according to the requirement of the other methods, example - For the bug repetition method, the set of files and the task id are passed to the respective helper class. (**AddCommit**)

```
Set<String> filesCommitted;
bugTasks.generateMapForTasks(task, commitFiles);
expert.generateMapForExpert(developer, commitFiles);
broadFeatures.generateMapForFeatures(task, commitFiles);
busyClasses.storeFiles(commitFiles);
```

2. During the commit, the no of files is tracked this is then used to create a matrix. eg - [A, B, C, D, E, F, H] -> 7x7 matrix (**componentMinimum**)
3. The Files present in the individual commits like b-123 files - A, B, C are used to generate pairs like AB, BC, and AC. This is created for all the commits and the frequency of all the common pairs is stored in a hash map.
4. Using these values the previously created graph is populated with values. We end up creating the adjacency matrix for the commits. (**softwareComponents**)
5. After populating the matrix we iterate through a row to find the edges with weights greater than the threshold, and add them to the set.



in this graph, the sides are [0-7] which correspond to [A, B, C, D, E, F, G, H]

6. For the first row we get {A, B, F} (Where threshold = 5). If a row has no values that are greater or equal to the threshold value then we push that value into a set, these are the independent components, that we later use.
7. We get the following {A, B, F}, {A, B, D, F}, {C}, {B, D}, {E, G, H}, {A, B, F}, {E, G}, {E, H} -> this is pushed to a set to remove all the duplicate elements.
8. Using 2 sets I pop values from one set and push them into the second set. While adding new elements to the second set -
    a. if an intersection is possible - replace that element with the union of the set that was popped and the set that is to be replaced
    b. if an intersection is not possible just add a new entry in the second set.
9. once the iteration is completed combine the previously separated independent component to this set. The resulting set contains the components.
10. A similar approach has been followed for **repetitionInBugs**, **broadFeatures** and **experts**.

Limitations and improvement

1. Common classes could have been merged together to provide a better implementation.
2. Interface could have been used for the expert class and broadfeatures as both have identical methods with just one being different.
3. Use of access specifiers in the helper classes.

Test Cases

Input Validation -

1. **addCommit()**

   - String of developer is null

   - String of developer is empty

   - commit time is negative eg -1

   - commit time is 0

   - Task string is null

   - Task String is empty ""

   - Task identifier starts with the other char other than B, F

   - Set of committed files is null

   - Set passed is empty

2. **setTimeWindow()**

   - int start time is negative (-1)

   - int endTime is negative (-2)

   - int endtime is zero (0)

3. **ComponentMinimun()**

   - threshold provided is negative

4. **repetitioninBugs()**

   - threshold provide is negative

5. **broadFeatures()**

   - threshold is negative

6. **experts**()

   - threshold is negative

7. **busyClasses**()

   - limit is more than the no of total files

   - limit provided is negative.

Boundary Cases -

1. **addCommit()**
   - one commit added
   - multiple commit added
2. **setTimeWindow()**
   - start time = 1
   - start time = max value of int
   - end time = max value of int
   - end time =1
3. **ComponentMinimun()**

   - Max value of int passed

   - negative value of int passed

   5. **repetitioninBugs()**

   - threshold provide is negative

4. **broadFeatures()**

   - threshold is negative

5. **experts**()

   - threshold is negative

6. **busyClasses**()

   - limit is more than the no of total files

   - limit provided is negative.

Control Flow -

1. addCommit()

   - Same commit is again added

   - Invalid commit is added

   - String Task has lower case letter

   - String Task has an upper case letter

   -the commit time is not in the range of the time

2. SetTimeWindow()

   -the Start time is greater than the end time

   -The endtime set is less than the start time

   -both time is equal

3. clearTimeWindow()

   -no time window was set

4. ComponentMinimun()

   - called when no components are present.

5. repetitionInBugs()

   -no bug commits are made

6. broadFeatures()

   -no features were commited

7. experts()

   -only one developer present

   -multiple developer present.

8. busyClasses()

   -called with a big threshold for one file.

Data Flow -

1. experts()
   - invoked when no component exists
2. busyClasses()
   - invoked before componentMinimum
   - invoked before adding commits
3. softwarecomponents()
   - invoked before component minimum
4. broad features
   - invoked before software components
   - invoked before add commits
5. addCommits

-invoked before calling set time window.