SECURITY LOG MANAGER

PROPOSAL

**Minglan Zhang** **Shane Carson**

**CSC316: Data Structures for Computer Scientists**

# **mzhang17@ncsu.edu** **slcarso2@ncsu.edu**

**North Carolina State University**

**Department of Computer Science**

**1/18/2018**

BLACK BOX TEST PLAN

For my black-box test cases, I will use the following test files named:

**activity.txt**

**USERNAME, TIMESTAMP, ACTION, RESOURCE**

jtking, 1/18/2018 1:22:21PM, view, prescription information

mbbrown, 1/18/2018 1:23:47PM, create, immunization order ssoulcrusher, 1/18/2018 1:22:01PM, delete, prescription information jdschmidt, 1/18/2018 1:24:21PM, view, prescription information

jtking, 1/18/2018, 12:58:14PM, delete, demographics information

**one.txt**

**USERNAME, TIMESTAMP, ACTION, RESOURCE**

jtking, 1/18/2018 1:22:21PM, view, prescription information

To start the program, run securityLogManagerGUI.java.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **Expected** | **Actual** |
| **testLoadFile**    **(ECP – loading file)** | **Preconditions**:  · securityLogManagerGUI has been loaded successfully. · The file activity.txt exists **Steps**:  1. Click “Load from file” 2. Browse to select the activity.txt file 3. Click submit | The activity file loads  and displays usernames:  · jtking  · mbbrown  · ssoulcrusher  · jdschmidt  · jtking | The activity file loads  and displays usernames:  · jtking  · mbbrown  · ssoulcrusher  · jdschmidt  · jtking |
| **testNotExistFile**    **(DT-loading a not existing flight file))** | **Preconditions**:  · securityLogManagerGUI has been loaded successfully.  · The file none.txt doesn’t exist **Steps**:   1. Click “Load from file” 2. Browse to select the none.txt file 3. Click submit | The software reprompts the user to specify a new file. | The software reprompts the user to specify a new file. |

|  |  |  |  |
| --- | --- | --- | --- |
| **testGenerateProfile**    **(ECP – output specific information)** | **Preconditions**:  · securityLogManagerGUI has been loaded successfully. · The files activity.txt have been  uploaded **Steps**:   1. Click “Generate Operational   Profile”   1. Choose time between   “1/17/2018 00:00:00 and  1/20/2018 12:00:00” | The program would output:  OperationalProfile    view prescription information: frequency: 2, percentage: 40.0% create immunization order: frequency: 1, percentage: 20.0% delete demographics information: frequency: 1, percentage: 20.0% delete prescription information: frequency: 1, percentage: 20.0% | The program would output:  OperationalProfile    view prescription information: frequency: 2, percentage: 40.0% create immunization order: frequency: 1, percentage: 20.0% delete demographics information: frequency: 1, percentage: 20.0% delete prescription information: frequency: 1, percentage: 20.0% |
| **testUserReport**    **(ECP – produce user report )** | **Preconditions**:  · securityLogManagerGUI has been loaded successfully. · The files activity.txt have been  uploaded **Steps**:   1. Click “produce user report” 2. Choose user “jtking” | The program would output:  Activity Report for jtking  1/18/2018  12:58:14PM - delete demographics information  1/18/2018 1:22:21PM - view prescription information | The program would output:  Activity Report for jtking  1/18/2018  12:58:14PM - delete demographics information  1/18/2018 1:22:21PM - view prescription information |
| **testMinimumActivity**    **(BVA - loading the file with minimum activity)** | **Preconditions**:  · securityLogManagerGUI has been loaded successfully. · The files one.txt have been  uploaded **Steps**:  1. Click “Load from file” 2. Browse to select the one.txt file  3. Click submit | one.txt successful loaded | one.txt successful loaded |

# DATA STRUCTURES

You must determine which data structure(s) would be the best choice for implementing an efficient solution to the problem. You must:

For this project, we will use the following data structures

* Users/Logs: an array-based list
* Activities: an array

For the array-based list, we have following methods to be implemented:

* lookUp(name): return all the activities with given name
* add(user): add user information to the end of the list

For the array, we have following methods to be implemented:

* lookUp(startTime, endTime): return all the activities with given range of time
* add(activity): add activity based on the timestamp

|  |  |  |
| --- | --- | --- |
| **Operation** | **Run time** |  |
| lookUp(name) | O(n) |
| add(user) | O(1) |
| lookUp(startTime, endTime) | O(n) |
| add(activity) | O(n) |
|  |  | |

ALGORITHM DESIGN

PROPOSED ALGORITHM

**Algorithm** generateOperationalProfile(E, start, end) **Input E:** an unsorted list of log entries the start time to filter the log entries the end time to filter the log entries

**Output S:** a string representing the operational profile

1. S ← null
2. for i ← 0 to E.size() - 1 do
3. if E.get(i).time is between start and end
4. information = E.get(i).time + E.get(i). action + E.get(i).resource
5. S += information
6. return S

**Algorithm** getLogsForUser(E, user) **Input E:** an unsorted list of log entries

the user for which to get logs

**Output L:** a sorted list of logs for the given user

1. L ← new list
2. for i ← 0 to E.size() - 1 do
3. if user = E.get(i).getUser() then
4. L.insert( E.get(i))
5. L ← mergesort(L) // sort by action and resource, alphabetically
6. return L

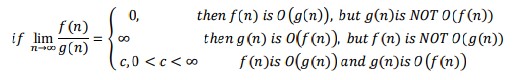
ALGORITHM ANALYSIS

|  |  |  |
| --- | --- | --- |
| Line | #operations | Description |
| 1 | 1 | Variable assignment |
| 2 | 1 | Variable assignment (i = 0) |
| 2 | 1 | Access E.size() |
| 2 | n | Comparison( i < E.size() - 1 ) |
| 3 | 2n + 2 | access E.get(i).time and compare it to the start and end |
| 4 | 3n + 1 | access 3 elements in E.get(i) |
| 5 | n | adding up strings |
| 6 | 1 | return the string |
| Total | 7n + 7 |  |

The estimated running time T(n) for the generateOperationalProfile(E) algorithm is:

which is O(n) as shown below using the limit test:

f(n) = 7n + 7 g(n) = n According to the limit test:

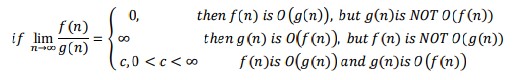


Therefore, since: the limit for f(n)/g(n) = (7n + 7n)/n = 14 f(n) is O(g(n)) as well as O(n)

|  |  |  |
| --- | --- | --- |
| Line | #operations | Description |
| a | 1 | Variable assignment |
| b | 1 | Variable assignment (i = 0) |
| b | 1 | Access E.size() |
| b | n | Comparison( i < E.size() - 1 ) |
| c | n | Adding i = i + 1 each iteration |
| c | n | Array access E.get(i).getUser |
| c | n | Compare user with E.get(i).getUser |
| d | n | adding to the list |
| e | n log n | mergesort |
| f | 1 | return the list |
| Total | 5n + nlogn + 4 |  |

The estimated running time T(n) for the getLogsForUser(E, user) algorithm is: which is O(n) as shown below using the limit test:

f(n) = 5n + nlogn + 4 g(n) = nlogn According to the limit test:



Therefore, since:

the limit for f(n)/g(n) = (5nlogn + nlogn + 4nlogn)/nlogn = 10 then f(n) is O(g(n)) as well as O(nlogn)

SOFTWARE DESIGN

In this section, you must present your software design for implementing your proposed algorithm. You must provide a UML class diagram.

For this software, we will be using the Model-View-Controller design, in order to keep the operations and how the software works away from the user. In our design, the SecurityLogManager, will hold an array-based list of logs, will generate the action list, collect the frequency, and get the user report, while the

SecurityLogManagerGUI will serve as the view to the users. SecurityLogManager will also read in the input file and generate the operational profile. Log will serve as the Model class to store information, such as the username, timestamp, action, and resource the user accessed at the time.

**The UML image:**

