Brenden Lynch 10/21/2020

CSE 248 Hiking App Project

Report on Data Structure Choices.

(Introduction.) The Java API provides a number of well-designed and efficient models of various data structures. Each has its own benefits and drawbacks and should be used under only certain circumstances specific to the nature of a project's requirements. This project currently calls for the implementation of efficient structures that provide user authentication capabilities [TreeMapUserStorageContainer.java, LinkedListUserStorageContainer.java], trail storage and completed [HikingHistoryStorageContainer.java]. The data structures brought under consideration for use in this project include HashMap, HashSet, TreeMap, TreeSet, LinkedList, ArrayList, Queue and Stack. HashSet and HashMap are two options easily eliminated from the available choices for two reasons. First, both of these data structures need to re-arrange their internal contents when data is inserted into them as a result of their array-based nature. This is detrimental to the overall efficiency of the program, and ArrayList, Stack and Queue can be eliminated for the same reason. Second, only the UserStorageContainer classes call for functionality remotely similar to what is provided by HashMap and HashSet, and TreeMap is a much better choice as it dynamically extends itself as necessary in a manner similar to a LinkedList. TreeSet is just a TreeMap without keys, and so it provides little use to this application. Since the TrailStorageContainer and HikingHistoryStorageContainer classes must be able to return multiple results from their internal data structures based on one query, TreeMap and TreeSet would not be of much use here (for these classes specifically).

(Hypotheses.) Upon easy elimination of various data structures as described in the previous section, only the TreeMap and LinkedList data structures remain as viable choices. It is believed that TreeMap will be the best choice of internal data structure for the UserStorageContainer class, and that LinkedList will be the best choice of internal data structure for the TrailStorageContainer and HikingHistoryStorageContainer classes.

(Approach and Methods.) All testing was conducted on a Late-2017 model Apple MacBook-Pro, with a 3.5 GHz Intel Core i7 processor, 16 GB of DDR4 RAM, and a 512 GB SSD. The Java Version was 14.0.1. During testing, all other applications were closed besides Eclipse and all unnecessary background services and processes were stopped. Test methods were executed from the TestingMethods.java class found in the "utilities" package. Data needed for generation of random Trail and User objects was extracted from three .txt files provided earlier in the semester, which were placed in the project's root directory. All tests were run three separate times, and the average values for each test were calculated and recorded. Data is presented on the last page, see *Fig.1, Fig.2, Fig.3, Fig.4*. Considering the fact that, by elimination, the best data structure for the TrailStorageContainer and HikingHistoryStorageContainer is LinkedList, the TrailStorageContainer was only tested with this data structure, and the HikingHistoryStorageContainer was not tested due to its extreme operational similarity to the TrailStorageContainer. The LinkedList-based UserStorageContainer and TreeMap-based UserStorageContainer were assessed on three fronts: the time it took to add 2000 randomly generated users, the time it took to authenticate a statically generated and added user, and the

time it took to remove a user. The TrailStorageContainer was also assessed on three fronts: the time it took to add 2000 randomly generated trails, the time it took to search for trails based on the input criteria “a”, and the time it took to delete a specific trail.

(Results.) The TreeMap-based UserStorageContainer was notably faster than the LinkedList-based UserStorageContainer in all three tests. The TrailStorageContainer took, on average, 94846200 nanoseconds to add 2000 randomly generated trails, 9785973 nanoseconds to search for trails based on input criteria “a”, and 277502 nanoseconds to delete a specified trail.

(Discussion and Conclusion.) As a result of these tests, it was determined to use a TreeMap as the underlying data structure for the UserStorageContainer. It was vastly more efficient at all 3 tested tasks than the LinkedList-based UserStorageContainer, which was somewhat expected, as the LinkedList-based version will need to sort itself each time a user is added or deleted in order to have the best login-performance once the application is fully built. It was also decided that the TrailStorageContainer and HikingHistoryStorageContainer will use a LinkedList, and no other data structures were identified as viable competitors.

(Data.)