Kiefer Kleister Project Discussion

Introuduction:

The basis of this project is to determine what the best data structure would be to store the data needed for an app about hiking trails. For this app, both users, trails, and hiking history to be stored. To start hiking history will need to grow as users add hikes they have been on. The hiking history must maintain the order that hikes are added. Based upon these two criteria arrays and all data structures based upon arrays, for example hashmap are eliminated. The reasoning behind this is those data structures cannot grow and must be initialized with a specific size. Other data structures that can be eliminate are anything tree based and priority queue. Trees will rearrange data based upon the compare to method that the data is given, and priority queue will return data based upon the priority it is given not the order it is placed. This leaves us with linkedlist, stack, and queue. Stack and queue are not data structures meant for holding data. They are more so a tool to make a specific operation happen. That is why linkedlist will be chosen as the candidate for the hiking history.

Trails have roughly the same requirements without having to maintain order. The data structure must grow and be able to search, delete, and insert. It must also return all elements that contain the search criteria and should allow duplicates of names of trails. Yet again any array-based data structure is eliminated from contention because it cannot grow. Two of the criteria, difficulty and trailtype, the number is known so hashmap will be used. The other criteria must be able to grow. For the purpose of the test treeset using a stream, and another data structure that has treemaps containing linkedlists for elevation and distance, linkedlist for names, and hashmap containing linkedlist for difficulty and trailtype will be compared.

Finally, the data structure for users will need to be able to grow and allow for fast searching to verify a user's password upon login. Yet again ignore all array-based data structures because they cannot grow. Instead treemap will be tested. Treemaps grow and do not have a limit when they are initialized. This is because treemap will allow usernames to be unique, and by searching the treemap with the username the password for login can be verified.

Hypothesis:

What I predict will happen is the multi structure data structure will perform better in searching and deletion while being slower for insertion because there are more five data structures that need to be added to versus one for the treeset, and treemap will have fast search times for users based upon searching by username.

Approach and Methods:

To test the data structures I ran the search, delete, and insertion methods at ten percent of the number of trails currently in them while timing each of the runs in nanoseconds. From there I

took an average of those times to account for best-case and worst-case scenarios occurring. I then increase the number of trails linearly in the data structures by the same amount we initially start with, and ignore the first run of the search, delete, and insertion methods as these are unreliable. In my testing I started with five hundred but ignore those results, and then increase to one thousand. After the initial run the methods are ran ten more times for a final number of fifty-five hundred trails in the data structures. The program then takes the times and put them into text files containing the results along with creating and displaying graphs for each of the search criteria.

Results:

The graphs that were generated showed the searching for elevation, distance, and trail type in the treemap data structure remaining generally flat despite the trail number being increased linearly. For distance search the initial time was 647 nanoseconds and the final search was 779. Elevation search was 1032 nanoseconds initially and ended with 1154 nanoseconds. Trail type started with 932 nanoseconds and ended at 1020 nanoseconds. The hashmap for difficulty started at 1909 nanoseconds but ended at 3347 nanoseconds and did not increase linearly, but still increased as we increased the number of trails. The last data structure in the multi data structure approach is linkedlist search using stream. This initially increased but then flattened. Initially 24545 nanoseconds it ended at 32395 nanoseconds. For the treeset with stream the lines on the graph were for the most part flat, and generally increasing as time went on. The final time results for each of them by search criteria were difficulty: 9543 nanoseconds, elevation: 6310 nanoseconds, distance 5618 nanoseconds, trail type: 5183 nanoseconds, and name: 5081 nanoseconds.

In deletion the graphs for the multi data structure approach were linear for the linkedlist of names, elevation treemap, and distance treemap. The graph of the hashmaps of difficulty and trail type were almost exponential. The start times for each were as follows difficulty: 220960270 nanoseconds, elevation: 4882718 nanoseconds, distance: 7753019 nanoseconds, trail type: 221195437 nanoseconds, and name: 804728 nanoseconds. Then end times were as follows difficulty: 6532511571 nanoseconds, elevation: 145669864 nanoseconds, distance: 228301239 nanoseconds, trail type: 6538594347 nanoseconds, and name: 4578952 nanoseconds. For deletion using stream in the treeset the graphs were all linear as we increased the number of trails. The start times were as follows difficulty: 837367 nanoseconds, elevation: 704832 nanoseconds, distance: 700162 nanoseconds, trail type: 839460 nanoseconds, and name: 5811243 nanoseconds. The end times are as follows difficulty: 4679560 nanoseconds, elevation: 3995219 nanoseconds, distance: 3987817 nanoseconds, trail type: 4757303 nanoseconds, and name: 32386367 nanoseconds.

Insertion for both the treeset and the multi structure approach were linear according to the graph, and their times almost overlapped. The initial times were multi structure: 16911155 nanoseconds and treeset: 16209000 nanoseconds. The end results were multi structure: 89158522 nanoseconds and treeset: 92676514 nanoseconds.

For the user treemap the search time and deletion times remained flat while the insertion time increased linearly. The start time for each are as follows search: 10834 nanoseconds, deletion: 8586 nanoseconds, and insertion: 68901297 nanoseconds. The end times were search: 11658 nanoseconds, deletion: 9538 nanoseconds, and insertion: 377163306 nanoseconds.

Discussion:

From the results we can start to compare the search time between the multi structure approach and the treeset approach. We can see that using the treemaps for elevation and distance, and hashmaps for difficulty and trail type we had search times that were four to five times faster than their counterparts in stream but using a linkedlist with stream for the names the times were much slower than using a treeset for the names up to six times slower.

Deletion between the two has stream being much more efficient, especially when deleting a large amount as is the case when deleting by trail type or difficulty. When deleting by this type nearly 1/3 of the data is being deleted. With the way deletion must be carried out in the multi structure approach everything deleted in one structure must be deleted in the others which increases the time complexity. This is not the case for the treeset. For the treeset you only go through the treeset once and remove all the elements based upon the criteria you have provided.

The results for the insertion are interesting. Trees generally are considered to have log n for insertion. There is an exception though which is when a tree must be rebalanced. That is what I assume is occurring here. Even though the multi structure approach needed to have five separate insertions occur, the result almost exactly the same as a single treeset.

For the treemap with users, the insertion behaved the same as the multi approach and the treeset. It increased linearly. The performance for searching was very good and remained flat even though we increased the number of users linearly this was the same result with removal.

Conclusion:

The multi structure approach proved to be better for searching in all categories except one versus the treeset. If the treeset approach were to be incorporated into the multi structure approach for searching by name instead of linkedlist that would yield the best overall search according to the results. For deletion when deleting a small amount of data, the multi structure approach has good efficiency, but for deleting larger amounts of data as seen in the hashmap difficulty and trail type it becomes bad. So treeset has better deletion. Insertion is almost the same for the two. That is why to decide between the two one would have to choose which is more important faster deletion, or faster search. If someone wanted the fastest search possible and would not being doing much deleting, the multi structure approach is the best, but if there will be a lot of deletion of data then treeset would be best. Treemap for user proved to be a good approach because it was proven to have a fast search time which was the main criteria. Linkedlist

will be the hiking history data structure because it is the best candidate, fulfills all the requirements, and is versatile.