Matthew Jednak

40133030

A3 report

For my CleverSIDC I decided to go with a hash map of ordered maps. My CleverSIDC has both size and threshold fields where size if the actual number of entries in the hash map and threshold is the maximum allowed number of entries in the hash map. I did this because a hash map would simplify the searching for an entry based on the hashing function. We would simply need to use the hashing function on the key we are looking for to instantly be pointed towards a bucket containing only a subset of the entries meaning we only have to look through a few entries which drastically speeds up searching. To further simplify things, each bucket of the hash map would contain an ordered map where the entries will be stored. This means that within each bucket the entries stored will already be in order so finding things like which is the next biggest key or the key smaller than a specific key or entire entry since it they will simply be the next and previous entry linked to a specific entry. For my ordered map it will store the entries in a sequence. I chose to use a sequence because it has the functionality of a linked list, array list and double ended queue. This then gives me the flexibility to access entries in a multitude of different ways depending on what I am trying to do with the ordered list as well as being able to have each ordered map in the hash map return a sequence to make a bit sequence of all the entries in the entire hash map. Furthermore, since all these subsequences are already sorted thanks to the ordered map, sorting the complete sequence is much easier. Lastly my entries are essentially typical nodes but instead of having a single element they have a string key and a string value, as well as the usual next and previous entries.

In terms of space and time complexity of the CleverSIDC methods, they never change as the underlying structure of CleverSIDC never changes however the size of the hash map does in fact change based on the threshold of CleverSIDC.

For getValues() it will very often have a time complexity of O(1) but at worst it will be O(n) for the specific bucket we are looking through. This is an important distinction to make between the O(n) of the entire hash map and the O(n) of the buckets since each bucket is an ordered map containing only miniscule fraction of the total elements in the hash map. This is because of the buckets of the hash map being ordered maps. In a situation where the key we are searching for is hashed to a bucket with a single entry, that entry is the one we are searching for. In the worst case scenario where there is more than one entry, it still isn’t that bad even though it is O(n) since that bucket of size n will only have a miniscule fraction of the total elements in the hash map.

For nextKey(), prevKey() and rangeKeys() they all have a time and space complexity of O(n) for the same reasons. Often it will be less than that again simply due to the nature of how a hash map functions but, in a situation, where for example we are searching for the next key of the biggest key in the hash map, we will need to compare it to every entry in the hash map to then conclude there is none bigger than it.

For remove() and add() they will both have a time and space complexity of very often O(1) and at worst O(n) of the bucket they are store in. The space and time complexity is as such for the exact same reasons as for getValues (i.e. the buckets of the hash map being ordered maps.)

For allKeys(), this one has a space and time complexity of O(n log n) since to sort my CleverSIDC I first return the sequence of the entire hash map and put each entry into a simple array which will take O(n). I then use Arrays.sort() on that array which uses merge sort which is O(n log n). Finally, I make a new sequence and then loop through the entire sorted array to add each of them one by one to the new sequence which will take O(n). In total this gives us (n log n + 2n) which is simply O(n log n)

Pseudo code for CleverSIDC functions

All of my CleverSIDC functions use a lot of functionsn from the underlying data structures to quickly accomplish what the functions needed to do such as getValues() which is simply a single line using the getValue method of the hash map. Because of this, we must assume that all the functions for the underlying classes are already implemented (i.e. all the functions for hash maps, ordered maps and sequences)

**allKeys()**

Input: none

Output: all keys in CleverSIDC as a sorted sequence

Algorithm allKeys()

Sequence s <- HashMap.entrySet()

arr[] with size of s

temp <- first entry from s

**rangeKeys()**

**add()**

**SetSIDCThreshold()**