Answers Lab2

Jakob Ristner, Oscar Kilberg and Lucas Karlsson

November 25, 2020

1 firstIndexOf()

Describe how your firstIndexOf() method in RangeBinarySearch.java finds the first index of a key that equals the search key.

We use a traditional binary search method taking an array, a key and a comparator as arguments. Them only thing that differs this from a normal one is that it does not stop once mid is equal to the key we are searching for. If this is the case, the algorithm will set hi to me equal to mid, therefor now "inching" our way down in the list one element at a time. This will eventually lead us to find the first occurrence of the key.

1.1 Implementation

```
int firstIndexOf(Term[] terms, Term key,
 Comparator<Term> comparator) {
  int lo = 0;
  int hi = terms.length - 1;

  while (lo <= hi) {
    int mid = (lo + hi) / 2;
    int comp = comparator.compare(key, terms[mid]);
    if (comp < 0) {
      hi = mid - 1;
    } else if (comp > 0) {
      lo = mid + 1;
    } else if (lo != mid){
      hi = mid;
    } else {
      return mid;
    }
  }
  return -1;
}
```

2 Complexity

What is the worst case time complexity in the number of compares that each of the operations in the Autocomplete data type make, as a function of the number of terms n and the number of matching terms m?

2.1 sortDictionary()

Sorting the dictionary is done prior to matching any terms and therefor the algorithm is not dependent on m. The algorithm used is the standard implementation in java for arrays. The complexity of this algorithm is $O(n \log n)$.

2.2 allMatches()

Here we search for matches twice with binary search, an algorithm with logarithmic complexity dependent on the size of input. The size of the input of our binary search calls is n. This is called twice and since big O notation does not care about constant, both of the binary search calls combined will still have logarithmic complexity $O(\log n)$.

The other algorithm used is sorting the matches by weight, this has linearithmic complexity but its input size is m this time, the amount of matches. The complexity for the entire method will therefor be additive $O(\log n + m \log m)$.

2.3 numberOfMactches()

This method does basically the same thing as the first part of the allMacthes method 2.2 with the two binary search calls but without sorting. Since both of these binary search algorithms have logarithmic complexity and do not depend on the number of matches, the entire method has linearithmic complexity $(O \log n)$.

3 Appendix

3.1 Time Spent

Jakob Ristner	2 Hours
Lucas Karlsson	2 Hours
Oscar Kilberg	2 Hours

3.2 Known bugs

We have found none so far, we are slightly worried about our implementation of the binary search to find the last occurrence of an item since we needed the division of

```
int mid = (lo + hi) / 2;
```

To ”round up”. Our solution was to implement it as follows.

```
int mid = (int) ((lo + hi) / 2.0 + 0.5);
```

This will work exactly the same as normal integer division but instead round upwards if presented with some floating point number. Since lo and hi are integers, this should not cause any adversary effectsoindent