Design

We can use a map to store the word information.

The key of the map, is the perfix of words, the value of the map is all the words, start with the perfix.

```
TreeMap<String, TreeSet<String>> wordTable;
```

Time complexity:

Assume that the average length of word is m, and there are totally n words in the system.

(1) addWords method:

Each time we add a new word, we should add all the sub string(totally m) which starts from the first letter into the wordTable(totally m*n). So, time complexity: O(mlog(mn)), since the TreeMap is implemented by RB Tree, whose time complexity is O(logn).

(2) searchText method:

Each time we search a word from the map, the time complexity is O(logmn), for there are mn key in map.

For the value, we can use TreeSet<String> as value type. It has 2 advantages:

(1) It can remove the duplicate word, which can avoid a new word being added for several time.

(2) The String in TreeSet is in order. We can loop through all the word in TreeSet in order.

For example, we insert “desk” into the class:

The data map will become:

[“d”] => {“desk”}

[“de”] => {“desk”}

[“des”] => {“desk”}

[“desk”] => {“desk”}

Then, we insert “deck” into class:

The data map will become:

[“d”] => {“desk”, ”deck”}

[“de”] => {“desk” ,”deck”}

[“des”] => {“desk”}

[“desk”] => {“desk”}

[“dec”] => {”deck”}

[“deck”] => {”deck”}

If we search “de”, then we can simple return the {“desk” ,”deck”}