COMP 3270 FALL 2020

Programming Project: Autocomplete

Name: _____Wenzhuo Fan_____ Date Submitted: _____11/11/2020_____

1. **Pseudocode**: Understand the strategy provided for *TrieAutoComplete*. State the algorithm for the functions precisely using numbered steps that follow the pseudocode conventions that we use. Provide an approximate efficiency analysis by filling the table given below, for your algorithm.

Add

- Pseudocode:

```
1.  currentNode = myRoot
2.          for (each chars in word)
3.              if weight > currentNodesubtree max weight
4.                  currentNodesubtree max weight = weight
5.              child = currentNode.getChild(chars)
6.              if child = null
7.                  child = new Node(chars, currentNode,weight)
8.                  currentNode.children.put(chars, child)
9.              currentNode = child
10.     currentNode.setWord(word)
11.     currentNode.setWeight(weight)
12.     currentNode.isWord = true
```

- Complexity analysis:

Step #	Complexity stated as O( )
1	Complexity stated as O(1)
2	Complexity stated as O(n)
3	Complexity stated as O(1)
4	Complexity stated as O(1)
5	Complexity stated as O(1)
6	Complexity stated as O(1)
7	Complexity stated as O(1)
8	Complexity stated as O(1)
9	Complexity stated as O(1)
10	Complexity stated as O(1)
11	Complexity stated as O(1)
12	Complexity stated as O(1)

Complexity of the algorithm = O(n)

topMatch

- Pseudocode:

1. ArrayList nodes = new ArrayList
2. currentNode = myRoot
3. for (each chars in prefix)
4. if(currentNode's children does not contain chars)
5. return ""
6. currentNode = currentNode.getChild(chars)
7. while (currentNode not a word and currentNodeweight≠subtreemax weight)
8. for (each child in currentNode's children)
9. if (childsubtreemaxweight = currentNodesubtreemaxweight)
10. currentNode = child
11. break
12. return currentNode.getWord

- Complexity analysis:

Step#	Complexity stated as O(1)
1	Complexity stated as O(1)
2	Complexity stated as O(1)
3	Complexity stated as O(n)
4	Complexity stated as O(1)
5	Complexity stated as O(1)
6	Complexity stated as O(1)
7	Complexity stated as O(n)
8	Complexity stated as O(n)
9	Complexity stated as O(1)
10	Complexity stated as O(1)
11	break
12	Complexity stated as O(1)

Complexity of the algorithm = O(n^2)

topMatches

- Pseudocode:

1. List matches = new ArrayList
2. PriorityQueue nodes = new PriorityQueue
3. for (each chars in prefix)
4. if currentNode.getChild(chars) = null
5. return empty ArrayList
6. currentNode = currentNode.getChild(chars)
7. if (currentNode is a word)

8. matches.add(new Term(currentNode's word, currentNode's weight))
9. for (each child in currentNode's children)
10. if (child does not equal null) nodes.offer(child)
11. while (nodes.size > 0)
12. currentNode = nodes.poll
13. if(currentNode is a word)
14. matches.add(new Term(currentNode's word, currentNode's weight))
15. for (each child in currentNode's children)
16. if (child does not equal null) nodes.offer(child)
17. matches.sort(terms in reverse order)
18. ArrayList kmatches = new ArrayList
19. if (k > matches.size())
20. kMatches.addAll(matches)
21. else
22. kMatches.addAll(matches.sublist(0, k))
23. ArrayList topKMatches = new ArrayList
24. for (each term in kMatches)
25. topKMatches.add(term's word)
26. return topKMatches

- Complexity analysis:

Step#	Complexity stated as O(_)
1	Complexity stated as O(1)
2	Complexity stated as O(1)
3	Complexity stated as O(n)
4	Complexity stated as O(1)
5	Complexity stated as O(1)
6	Complexity stated as O(1)
7	Complexity stated as O(1)
8	Complexity stated as O(1)
9	Complexity stated as O(n)
10	Complexity stated as O(1)
11	Complexity stated as O(n)
12	Complexity stated as O(1)
13	Complexity stated as O(1)
14	Complexity stated as O(1)
15	Complexity stated as O(n)
16	Complexity stated as O(1)
17	Complexity stated as O(1)
18	Complexity stated as O(1)
19	Complexity stated as O(1)
20	Complexity stated as O(1)

21	Complexity stated as O(1)
22	Complexity stated as O(1)
23	Complexity stated as O(1)
24	Complexity stated as O(n)
25	Complexity stated as O(1)
26	Complexity stated as O(1)

Complexity of the algorithm = O(n^2)

2.**Testing**: Complete your test cases to test the *TrieAutoComplete* functions based upon the criteria mentioned below.

Test of correctness:

Assuming the trie already contains the terms {”ape, 6”, ”app, 4”, ”ban, 2”, ”bat, 3”, ”bee, 5”, ”car, 7”, ”cat, 1”}, you would expect results based on the following table:

Query	k	Result
””	8	{”car”, ”ape”, ”bee”, ”app”, ”bat”, ”ban”, ”cat”}
””	1	{”car”}
””	2	{”car”, ”ape”}
””	3	{”car”, ”ape”, ”bee”}
”a”	1	{”ape”}
”ap”	1	{”ape”}
”b”	2	{”bee”, ”bat”}
”ba”	2	{”bee”, ”bat”}
”d”	100	{}

3.**Analysis**: Answer the following questions. Use data wherever possible to justify your answers, and keep explanations brief but accurate:

i. What is the order of growth (big-Oh) of the number of compares (in the worst case) that each of the operations in the *Autocompletor* data type make?

Add: O(n)

topMatch: O(n^2)

topMatches: O(n^2)

ii. How does the runtime of *topMatches()* vary with k, assuming a fixed prefix and set of terms? Provide answers for

BruteAutocomplete and *TrieAutocomplete*. Justify your answer, with both data and algorithmic analysis.

```
Benchmarking Autocomplete$BruteAutocomplete...
Time for topMatch("") - 8.71398E-4
Time for topMatch("nenk") - 0.0046221656
Time for topMatch("n") - 0.0030767099
Time for topMatch("ne") - 0.0031714402
Time for topMatch("notarealword") - 0.0045045005
Time for topKMatches("", 1) -  0.0049078661
Time for topKMatches("", 4) -  0.0049059788
Time for topKMatches("", 7) -  0.0048961293
Time for topKMatches("nenk", 1) -  0.0041896526
Time for topKMatches("nenk", 4) -  0.0042263688
Time for topKMatches("nenk", 7) -  0.0042187739
Time for topKMatches("n", 1) -  0.0042610398
Time for topKMatches("n", 4) -  0.0042348345
Time for topKMatches("n", 7) -  0.0043183233
Time for topKMatches("ne", 1) -  0.00450598
Time for topKMatches("ne", 4) -  0.0046498949
Time for topKMatches("ne", 7) -  0.0044189608
Time for topKMatches("notarealword", 1) -  0.0040932056
Time for topKMatches("notarealword", 4) -  0.0041054981
Time for topKMatches("notarealword", 7) -  0.0040987204
```

```
Benchmarking Autocomplete$TrieAutocomplete...
Created 475255 nodes
Time for topMatch("") - 1.45902E-5
Time for topMatch("nenk") - 2.462E-7
Time for topMatch("n") - 9.6422E-6
Time for topMatch("ne") - 4.1023E-6
Time for topMatch("notarealword") - 2.244E-7
Time for topKMatches("", 1) -  6.6649E-6
Time for topKMatches("", 4) -  1.67407E-5
Time for topKMatches("", 7) -  2.14221E-5
Time for topKMatches("nenk", 1) -  3.226E-7
Time for topKMatches("nenk", 4) -  3.119E-7
Time for topKMatches("nenk", 7) -  2.987E-7
Time for topKMatches("n", 1) -  2.8288E-6
Time for topKMatches("n", 4) -  9.0933E-6
Time for topKMatches("n", 7) -  1.59511E-5
Time for topKMatches("ne", 1) -  2.0076E-6
Time for topKMatches("ne", 4) -  4.0082E-6
Time for topKMatches("ne", 7) -  7.7045E-6
Time for topKMatches("notarealword", 1) -  1.885E-7
Time for topKMatches("notarealword", 4) -  1.87E-7
Time for topKMatches("notarealword", 7) -  4.30838E-5
```

iii. How does increasing the size of the source and increasing the size of the prefix argument affect the runtime of *topMatch* and *topMatches*? (Tip: Benchmark each implementation using fourletterwords.txt, which has all four-letter combinations from aaaa to zzzz, and fourletterwordshalf.txt, which has all four-letter word combinations from aaaa to mzzz. These datasets provide a very clean distribution of words and an exact 1-to-2 ratio of words in source files.)

By looking at the above charts of fourletterwords and fourletterwordhalf, we can conclude that adding prefix parameters and increasing source size will make trieutomplete match faster. However, it should be noted that the time difference within the trie data structure in the diagram is very small and almost negligible, because they work in the numerical range of 10 ^ - 6 to 10 ^ - 8, which is different from brute and binary.

```
****************************************************************
Printing Summary of Results ...

****************************************************************
prefix         , Brute          , Binary         , Trie
         -----------------------------------------------------------------
               ,    8.71398e-04,    8.93352e-04,    1.45902e-05
n_1            ,    4.26104e-03,    4.29270e-05,    2.82880e-06
n_4            ,    4.23483e-03,    4.39775e-05,    9.09330e-06
notarealword_7 ,    4.09872e-03,    1.17820e-06,    4.30838e-05
notarealword_1 ,    4.09321e-03,    1.19210e-06,    1.88500e-07
nenk           ,    4.62217e-03,    1.09310e-06,    2.46200e-07
n_7            ,    4.31832e-03,    4.48940e-05,    1.59511e-05
notarealword_4 ,    4.10550e-03,    1.18650e-06,    1.87000e-07
notarealword   ,    4.50450e-03,    1.77390e-06,    2.24400e-07
n              ,    3.07671e-03,    1.65923e-05,    9.64220e-06
nenk_1         ,    4.18965e-03,    4.83200e-07,    3.22600e-07
_1             ,    4.90787e-03,    1.65881e-03,    6.66490e-06
_4             ,    4.90598e-03,    1.63317e-03,    1.67407e-05
ne             ,    3.17144e-03,    1.37300e-06,    4.10230e-06
_7             ,    4.89613e-03,    1.63083e-03,    2.14221e-05
ne_1           ,    4.50598e-03,    2.65920e-06,    2.00760e-06
nenk_7         ,    4.21877e-03,    4.82900e-07,    2.98700e-07
ne_7           ,    4.41896e-03,    3.32810e-06,    7.70450e-06
nenk_4         ,    4.22637e-03,    4.84900e-07,    3.11900e-07
ne_4           ,    4.64989e-03,    3.08800e-06,    4.00820e-06
               ,    8.71398e-04,    8.93352e-04,    1.45902e-05
n_1            ,    4.26104e-03,    4.29270e-05,    2.82880e-06
n_4            ,    4.23483e-03,    4.39775e-05,    9.09330e-06
notarealword_7 ,    4.09872e-03,    1.17820e-06,    4.30838e-05
notarealword_1 ,    4.09321e-03,    1.19210e-06,    1.88500e-07
nenk           ,    4.62217e-03,    1.09310e-06,    2.46200e-07
n_7            ,    4.31832e-03,    4.48940e-05,    1.59511e-05
notarealword_4 ,    4.10550e-03,    1.18650e-06,    1.87000e-07
notarealword   ,    4.50450e-03,    1.77390e-06,    2.24400e-07
n              ,    3.07671e-03,    1.65923e-05,    9.64220e-06
nenk_1         ,    4.18965e-03,    4.83200e-07,    3.22600e-07
_1             ,    4.90787e-03,    1.65881e-03,    6.66490e-06
_4             ,    4.90598e-03,    1.63317e-03,    1.67407e-05
ne             ,    3.17144e-03,    1.37300e-06,    4.10230e-06
_7             ,    4.89613e-03,    1.63083e-03,    2.14221e-05
ne_1           ,    4.50598e-03,    2.65920e-06,    2.00760e-06
nenk_7         ,    4.21877e-03,    4.82900e-07,    2.98700e-07
ne_7           ,    4.41896e-03,    3.32810e-06,    7.70450e-06
nenk_4         ,    4.22637e-03,    4.84900e-07,    3.11900e-07
ne_4           ,    4.64989e-03,    3.08800e-06,    4.00820e-06
               ,    8.71398e-04,    8.93352e-04,    1.45902e-05
n_1            ,    4.26104e-03,    4.29270e-05,    2.82880e-06
n_4            ,    4.23483e-03,    4.39775e-05,    9.09330e-06
notarealword_7 ,    4.09872e-03,    1.17820e-06,    4.30838e-05
notarealword_1 ,    4.09321e-03,    1.19210e-06,    1.88500e-07
nenk           ,    4.62217e-03,    1.09310e-06,    2.46200e-07
n_7            ,    4.31832e-03,    4.48940e-05,    1.59511e-05
notarealword_4 ,    4.10550e-03,    1.18650e-06,    1.87000e-07
notarealword   ,    4.50450e-03,    1.77390e-06,    2.24400e-07
n              ,    3.07671e-03,    1.65923e-05,    9.64220e-06
nenk_1         ,    4.18965e-03,    4.83200e-07,    3.22600e-07
_1             ,    4.90787e-03,    1.65881e-03,    6.66490e-06
_4             ,    4.90598e-03,    1.63317e-03,    1.67407e-05
ne             ,    3.17144e-03,    1.37300e-06,    4.10230e-06
_7             ,    4.89613e-03,    1.63083e-03,    2.14221e-05
ne_1           ,    4.50598e-03,    2.65920e-06,    2.00760e-06
nenk_7         ,    4.21877e-03,    4.82900e-07,    2.98700e-07
ne_7           ,    4.41896e-03,    3.32810e-06,    7.70450e-06
nenk_4         ,    4.22637e-03,    4.84900e-07,    3.11900e-07
ne_4           ,    4.64989e-03,    3.08800e-06,    4.00820e-06

 ----jGRASP: operation complete.
```

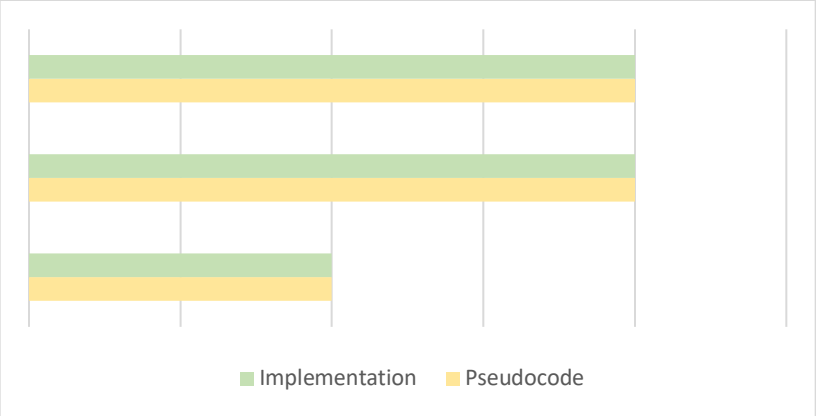
```
****************************************************************
Printing Summary of Results ...

****************************************************************
prefix          , Brute           , Binary          , Trie
               -----------------------------------------------------------------
                ,     6.10911e-04,     2.88282e-04,     1.46397e-05
a               ,     5.60254e-04,     1.31324e-05,     6.37880e-06
notarealword_7  ,     1.78703e-03,     9.94000e-07,     8.59990e-08
ae              ,     5.86133e-04,     1.25370e-06,     5.51090e-06
notarealword_1  ,     1.72334e-03,     9.87099e-07,     9.87000e-08
notarealword_4  ,     1.72738e-03,     9.97300e-07,     8.59990e-08
notarealword    ,     3.58037e-03,     1.63760e-06,     7.67000e-08
aenk            ,     4.15117e-03,     1.06600e-06,     2.21600e-07
aenk_7          ,     3.40601e-03,     4.58900e-07,     2.88400e-07
ae_7            ,     1.84666e-03,     3.21540e-06,     8.92240e-06
_1              ,     4.20994e-03,     6.87776e-04,     7.91600e-06
a_1             ,     3.39111e-03,     4.15361e-05,     2.70840e-06
ae_4            ,     1.88933e-03,     2.78790e-06,     5.59720e-06
_4              ,     3.94758e-03,     6.76033e-04,     1.29528e-05
a_4             ,     3.53931e-03,     4.11444e-05,     4.66366e-05
_7              ,     4.12283e-03,     6.88643e-04,     2.11976e-05
a_7             ,     3.56257e-03,     4.21725e-05,     1.61389e-05
aenk_1          ,     3.45465e-03,     4.66099e-07,     3.09600e-07
ae_1            ,     2.05641e-03,     2.42240e-06,     2.03840e-06
aenk_4          ,     3.41064e-03,     4.59300e-07,     2.81300e-07
                ,     6.10911e-04,     2.88282e-04,     1.46397e-05
a               ,     5.60254e-04,     1.31324e-05,     6.37880e-06
notarealword_7  ,     1.78703e-03,     9.94000e-07,     8.59990e-08
ae              ,     5.86133e-04,     1.25370e-06,     5.51090e-06
notarealword_1  ,     1.72334e-03,     9.87099e-07,     9.87000e-08
notarealword_4  ,     1.72738e-03,     9.97300e-07,     8.59990e-08
notarealword    ,     3.58037e-03,     1.63760e-06,     7.67000e-08
aenk            ,     4.15117e-03,     1.06600e-06,     2.21600e-07
aenk_7          ,     3.40601e-03,     4.58900e-07,     2.88400e-07
ae_7            ,     1.84666e-03,     3.21540e-06,     8.92240e-06
_1              ,     4.20994e-03,     6.87776e-04,     7.91600e-06
a_1             ,     3.39111e-03,     4.15361e-05,     2.70840e-06
ae_4            ,     1.88933e-03,     2.78790e-06,     5.59720e-06
_4              ,     3.94758e-03,     6.76033e-04,     1.29528e-05
a_4             ,     3.53931e-03,     4.11444e-05,     4.66366e-05
_7              ,     4.12283e-03,     6.88643e-04,     2.11976e-05
a_7             ,     3.56257e-03,     4.21725e-05,     1.61389e-05
aenk_1          ,     3.45465e-03,     4.66099e-07,     3.09600e-07
ae_1            ,     2.05641e-03,     2.42240e-06,     2.03840e-06
aenk_4          ,     3.41064e-03,     4.59300e-07,     2.81300e-07
                ,     6.10911e-04,     2.88282e-04,     1.46397e-05
a               ,     5.60254e-04,     1.31324e-05,     6.37880e-06
notarealword_7  ,     1.78703e-03,     9.94000e-07,     8.59990e-08
ae              ,     5.86133e-04,     1.25370e-06,     5.51090e-06
notarealword_1  ,     1.72334e-03,     9.87099e-07,     9.87000e-08
notarealword_4  ,     1.72738e-03,     9.97300e-07,     8.59990e-08
notarealword    ,     3.58037e-03,     1.63760e-06,     7.67000e-08
aenk            ,     4.15117e-03,     1.06600e-06,     2.21600e-07
aenk_7          ,     3.40601e-03,     4.58900e-07,     2.88400e-07
ae_7            ,     1.84666e-03,     3.21540e-06,     8.92240e-06
_1              ,     4.20994e-03,     6.87776e-04,     7.91600e-06
a_1             ,     3.39111e-03,     4.15361e-05,     2.70840e-06
ae_4            ,     1.88933e-03,     2.78790e-06,     5.59720e-06
_4              ,     3.94758e-03,     6.76033e-04,     1.29528e-05
a_4             ,     3.53931e-03,     4.11444e-05,     4.66366e-05
_7              ,     4.12283e-03,     6.88643e-04,     2.11976e-05
a_7             ,     3.56257e-03,     4.21725e-05,     1.61389e-05
aenk_1          ,     3.45465e-03,     4.66099e-07,     3.09600e-07
ae_1            ,     2.05641e-03,     2.42240e-06,     2.03840e-06
aenk_4          ,     3.41064e-03,     4.59300e-07,     2.81300e-07
```

4. Graphical Analysis: Provide a graphical analysis by comparing the following:

   i. The big-Oh for *TrieAutoComplete* after analyzing the pseudocode and big-Oh for *TrieAutoComplete* after the implementation.

      The big-Oh after analyzing the pseudocode was $O(n^2)$. After implementation is still appears to be $O(n^2)$.



ii. Compare the *TrieAutoComplete* with *BruteAutoComplete* and *BinarySearchAutoComplete*.