COMP 3270 FALL 2021

**Programming Project: Autocomplete**

Name: Kevin Bumgarner Date Submitted: 10/29/21

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: **add**(word: String of **length n**, weight: double)

1. if word equals null then
2. throw new NullPointerException
3. if weight is less than 0 then
4. throw new IllegalArgumentException
5. Node currNode equals myRoot
6. for each character in word
7. if currNode mySubtreeMaxWeight is less than weight then
8. currNode mySubtreeMaxWeight equals weight
9. if currNodes child holding character equals null then
10. Node childNode equals new Node
11. put childNode in currNode children
12. currNode equals currNode child holding character;
13. currNode isWord equals true
14. set currNodes word
15. set currNodes weight

* Complexity analysis:

|  |  |
| --- | --- |
| Step # | Complexity stated as O(\_) |
| 1 | O(1) |
| 2 | O(1) |
| 3 | O(1) |
| 4 | O(1) |
| 5 | O(1) |
| 6 | O(n) |
| 7 | O(1) |
| 8 | O(1) |
| 9 | O(1) |
| 10 | O(1) |
| 11 | O(1) |
| 12 | O(1) |
| 13 | O(1) |
| 14 | O(1) |
| 15 | O(1) |
|  |  |
|  |  |
|  |  |

Complexity of the algorithm = O(n)

*topMatch*

* Pseudocode: **topMatch**(prefix: String of length s)

1. if prefix equals null then

2. throw NullPointerException

3. create String tm

4. create Node currNode equals myRoot

5. create Priority Queue nodeQ of type Node

6. for each character in prefix do

7. if currNode child holding character is not null

8. currNode equals currNode child holding character

9. else

10. return tm

11. double maxWeight equals currNode mySubtreeMaxWeight

12. if currNode isWord && currNode weight equals maxWeight

13. return currNode word

14. else

15. while size of nodeQ is greater than 0

16. if currNode isWord && currNode weight equals maxWeight

17. tm equals currNode word;

18. break loop

19. currNode equals head of nodeQ

20. add all children of currNode to nodeQ

21. return tm;

* Complexity analysis:

|  |  |
| --- | --- |
| Step # | Complexity stated as O(\_) |
| 1 | O(1) |
| 2 | O(1) |
| 3 | O(1) |
| 4 | O(1) |
| 5 | O(1) |
| 6 | O(s) s = length of prefix |
| 7 | O(1) |
| 8 | O(1) |
| 9 | O(1) |
| 10 | O(1) |
| 11 | O(1) |
| 12 | O(1) |
| 13 | O(1) |
| 14 | O(1) |
| 15 | O(n) |
| 16 | O(1) |
| 17 | O(1) |
| 18 | O(1) |
| 19 | O(1) |
| 20 | O(nlogn) |
| 21 | O(1) |

Complexity of the algorithm = O(n)

*topMatches*

* Pseudocode: **topMatches**(prefix: String of length s, k: int)

1. if prefix equals null then

2. throw new NullPointerException

3. Node currNode equals myRoot

4. Create ArrayList<Node> sortedNodeList

5. Create ArrayList<String> tMatchesList

6. Create PriorityQueue<Node> nodeQ

7. if k equals 0 then

8. return tMatchesList

9. for each character in prefix

10. if currNode.getChild(c) equals null then

11. currNode equals currNode.getChild(c)

12. else

13. return tMatchesList

14. nodeQ.add(currNode)

15. double maxWeight equals currNode mySubtreeMaxWeight;

16. while nodeQ.size() is greater than 0 do

17. if sortedNodeList.size() equals k && nodeQ.peek().mySubtreeMaxWeight is less thqan sortedNodeList.get(k - 1).getWeight()

18. break;

19. currNode equals nodeQ.poll();

20. if currNode isWord then

21. sortedNodeList.add(currNode)

22. nodeQ.addAll(currNode.children.values())

23. Collections.sort(sortedNodeList, Collections.reverseOrder())

24. if sortedNodeList.size() is greater than or equal k then

25. for int i = 0 to i is less than k

26. tMatchesList.add(sortedNodeList.get(i).getWord())

27. else

28. for each Node in nodeSortedList

29. tMatchesList.add(n.getWord())

30. return tMatchesList

* Complexity analysis:

|  |  |
| --- | --- |
| Step # | Complexity stated as O(\_) |
| 1 | O(1) |
| 2 | O(1) |
| 3 | O(1) |
| 4 | O(1) |
| 5 | O(1) |
| 6 | O(1) |
| 7 | O(1) |
| 8 | O(1) |
| 9 | O(s) s = length of prefix |
| 10 | O(1) |
| 11 | O(1) |
| 12 | None |
| 13 | O(1) |
| 14 | O(1) |
| 15 | O(1) |
| 16 | O(n) |
| 17 | O(1) |
| 18 | O(1) |
| 19 | O(1) |
| 20 | O(1) |
| 21 | O(1) |
| 22 | O(nlogn) |
| 23 | O(nlogn) |
| 24 | O(1) |
| 25 | O(k) |
| 26 | O(1) |
| 27 | O(1) |
| 28 | O(k) |
| 29 | O(1) |
| 30 | O(1) |

Complexity of the algorithm = O(n)

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:

1. 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), o(n2)
   * topMatch: O(n), o(n2)
   * topMatches: O(n), o(n2)
2. 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.
   * topMacthes() will vary linearly ‘O(n)’ based off the pseudocode analysis in the above questions
3. 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.)

* Increasing the size of prefix and source will decrease the runtime for topMatches and topMatch

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

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

Data:

File: fourletterwords.txt

Found 456976 words

Time to initialize BruteAutoComplete - 0.0477275

Time to initialize Autocomplete.BinarySearchAutocomplete - 0.0275037

Time to initialize - Autocomplete.TrieAutocomplete - 0.1166372

Benchmarking Autocomplete$BruteAutocomplete...

Time for topMatch("") - 7.612716E-4

Time for topMatch("nenk") - 0.003067454

Time for topMatch("n") - 0.0014748763

Time for topMatch("ne") - 0.0015448292

Time for topMatch("notarealword") - 0.0022848057

Time for topKMatches("", 1) - 0.0031228319

Time for topKMatches("", 4) - 0.003095387

Time for topKMatches("", 7) - 0.0031036353

Time for topKMatches("nenk", 1) - 0.0027073078

Time for topKMatches("nenk", 4) - 0.0026879848

Time for topKMatches("nenk", 7) - 0.0026958494

Time for topKMatches("n", 1) - 0.0027130355

Time for topKMatches("n", 4) - 0.0026930876

Time for topKMatches("n", 7) - 0.0026856974

Time for topKMatches("ne", 1) - 0.0026758676

Time for topKMatches("ne", 4) - 0.0026818283

Time for topKMatches("ne", 7) - 0.0026740902

Time for topKMatches("notarealword", 1) - 0.0024444001

Time for topKMatches("notarealword", 4) - 0.0024390745

Time for topKMatches("notarealword", 7) - 0.0024517826

Benchmarking Autocomplete$BinarySearchAutocomplete...

Time for topMatch("") - 8.3789E-4

Time for topMatch("nenk") - 1.0914E-6

Time for topMatch("n") - 9.2786E-6

Time for topMatch("ne") - 1.4368E-6

Time for topMatch("notarealword") - 2.2032E-6

Time for topKMatches("", 1) - 0.0013600462

Time for topKMatches("", 4) - 0.0013721782

Time for topKMatches("", 7) - 0.0013600908

Time for topKMatches("nenk", 1) - 4.716E-7

Time for topKMatches("nenk", 4) - 4.498E-7

Time for topKMatches("nenk", 7) - 4.614E-7

Time for topKMatches("n", 1) - 3.89466E-5

Time for topKMatches("n", 4) - 3.99974E-5

Time for topKMatches("n", 7) - 4.99428E-5

Time for topKMatches("ne", 1) - 2.5827E-6

Time for topKMatches("ne", 4) - 3.1143E-6

Time for topKMatches("ne", 7) - 3.4049E-6

Time for topKMatches("notarealword", 1) - 1.273E-6

Time for topKMatches("notarealword", 4) - 1.2692E-6

Time for topKMatches("notarealword", 7) - 1.2819E-6

Benchmarking Autocomplete$TrieAutocomplete...

Created 475255 nodes

Time for topMatch("") - 9.505E-7

Time for topMatch("nenk") - 6.566E-7

Time for topMatch("n") - 1.538E-7

Time for topMatch("ne") - 1.34E-7

Time for topMatch("notarealword") - 2.08E-7

Time for topKMatches("", 1) - 2.47138E-5

Time for topKMatches("", 4) - 3.36716E-5

Time for topKMatches("", 7) - 4.12774E-5

Time for topKMatches("nenk", 1) - 7.421E-7

Time for topKMatches("nenk", 4) - 5.742E-7

Time for topKMatches("nenk", 7) - 5.371E-7

Time for topKMatches("n", 1) - 3.4687E-6

Time for topKMatches("n", 4) - 1.37179E-5

Time for topKMatches("n", 7) - 1.44815E-5

Time for topKMatches("ne", 1) - 1.9701E-6

Time for topKMatches("ne", 4) - 3.6413E-6

Time for topKMatches("ne", 7) - 6.8848E-6

Time for topKMatches("notarealword", 1) - 3.455E-7

Time for topKMatches("notarealword", 4) - 3.314E-7

Time for topKMatches("notarealword", 7) - 3.296E-7

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Printing Summary of Results ...

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

prefix , Brute , Binary , Trie

-------------------------------------------------------------------

, 7.61272e-04, 8.37890e-04, 9.50500e-07

n\_1 , 2.71304e-03, 3.89466e-05, 3.46870e-06

n\_4 , 2.69309e-03, 3.99974e-05, 1.37179e-05

notarealword\_7 , 2.45178e-03, 1.28190e-06, 3.29600e-07

notarealword\_1 , 2.44440e-03, 1.27300e-06, 3.45500e-07

nenk , 3.06745e-03, 1.09140e-06, 6.56600e-07

n\_7 , 2.68570e-03, 4.99428e-05, 1.44815e-05

notarealword\_4 , 2.43907e-03, 1.26920e-06, 3.31400e-07

notarealword , 2.28481e-03, 2.20320e-06, 2.08000e-07

n , 1.47488e-03, 9.27860e-06, 1.53800e-07

nenk\_1 , 2.70731e-03, 4.71600e-07, 7.42100e-07

\_1 , 3.12283e-03, 1.36005e-03, 2.47138e-05

\_4 , 3.09539e-03, 1.37218e-03, 3.36716e-05

ne , 1.54483e-03, 1.43680e-06, 1.34000e-07

\_7 , 3.10364e-03, 1.36009e-03, 4.12774e-05

ne\_1 , 2.67587e-03, 2.58270e-06, 1.97010e-06

nenk\_7 , 2.69585e-03, 4.61400e-07, 5.37100e-07

ne\_7 , 2.67409e-03, 3.40490e-06, 6.88480e-06

nenk\_4 , 2.68798e-03, 4.49800e-07, 5.74200e-07

ne\_4 , 2.68183e-03, 3.11430e-06, 3.64130e-06

, 7.61272e-04, 8.37890e-04, 9.50500e-07

n\_1 , 2.71304e-03, 3.89466e-05, 3.46870e-06

n\_4 , 2.69309e-03, 3.99974e-05, 1.37179e-05

notarealword\_7 , 2.45178e-03, 1.28190e-06, 3.29600e-07

notarealword\_1 , 2.44440e-03, 1.27300e-06, 3.45500e-07

nenk , 3.06745e-03, 1.09140e-06, 6.56600e-07

n\_7 , 2.68570e-03, 4.99428e-05, 1.44815e-05

notarealword\_4 , 2.43907e-03, 1.26920e-06, 3.31400e-07

notarealword , 2.28481e-03, 2.20320e-06, 2.08000e-07

n , 1.47488e-03, 9.27860e-06, 1.53800e-07

nenk\_1 , 2.70731e-03, 4.71600e-07, 7.42100e-07

\_1 , 3.12283e-03, 1.36005e-03, 2.47138e-05

\_4 , 3.09539e-03, 1.37218e-03, 3.36716e-05

ne , 1.54483e-03, 1.43680e-06, 1.34000e-07

\_7 , 3.10364e-03, 1.36009e-03, 4.12774e-05

ne\_1 , 2.67587e-03, 2.58270e-06, 1.97010e-06

nenk\_7 , 2.69585e-03, 4.61400e-07, 5.37100e-07

ne\_7 , 2.67409e-03, 3.40490e-06, 6.88480e-06

nenk\_4 , 2.68798e-03, 4.49800e-07, 5.74200e-07

ne\_4 , 2.68183e-03, 3.11430e-06, 3.64130e-06

, 7.61272e-04, 8.37890e-04, 9.50500e-07

n\_1 , 2.71304e-03, 3.89466e-05, 3.46870e-06

n\_4 , 2.69309e-03, 3.99974e-05, 1.37179e-05

notarealword\_7 , 2.45178e-03, 1.28190e-06, 3.29600e-07

notarealword\_1 , 2.44440e-03, 1.27300e-06, 3.45500e-07

nenk , 3.06745e-03, 1.09140e-06, 6.56600e-07

n\_7 , 2.68570e-03, 4.99428e-05, 1.44815e-05

notarealword\_4 , 2.43907e-03, 1.26920e-06, 3.31400e-07

notarealword , 2.28481e-03, 2.20320e-06, 2.08000e-07

n , 1.47488e-03, 9.27860e-06, 1.53800e-07

nenk\_1 , 2.70731e-03, 4.71600e-07, 7.42100e-07

\_1 , 3.12283e-03, 1.36005e-03, 2.47138e-05

\_4 , 3.09539e-03, 1.37218e-03, 3.36716e-05

ne , 1.54483e-03, 1.43680e-06, 1.34000e-07

\_7 , 3.10364e-03, 1.36009e-03, 4.12774e-05

ne\_1 , 2.67587e-03, 2.58270e-06, 1.97010e-06

nenk\_7 , 2.69585e-03, 4.61400e-07, 5.37100e-07

ne\_7 , 2.67409e-03, 3.40490e-06, 6.88480e-06

nenk\_4 , 2.68798e-03, 4.49800e-07, 5.74200e-07

ne\_4 , 2.68183e-03, 3.11430e-06, 3.64130e-06

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