[COMP250]

-

Assignment 3

-

Question 2

Due: Sun, Nov. 16, 2012 at 23:59

Question 2 Running time analysis of Tries (30 points)

In this question, you will analyze the complexity of two of methods implemented in Question 1.. Suppose you are given a set of words, where each word is of length at most m characters. Let T1 denote an instance of the trie implemented in Question 1, and let N be the number of nodes of T1. Note that N typically does not equal n.

Answer the following questions. In each case, give the tightest bound you can. For example, if a runtime is O(n) and you write O(nm) then you have not given the tightest bound. In each case, *you must also provide a short written justification for your answer*. Notice that we are not asking for formal (mathematical) big O and big Omega proofs here.

Your answer should be in terms of the variables n, m, k, N which are defined in this PDF. Note that even though k=NUMCHILDREN is a constant in the code, we are asking you to treat it as a variable.

1) What is the O( ) bound of *loadKeys()* on T1?

We are inserting n keys made out of m characters at most (N nodes in general) to the Trie T1.
Let check out the insert() method routine:
- Get the prefix.
- Fill in the missing nodes starting from the prefix.

The routine travels through m nodes max.
Inserting n keys of m length at most will take n*m nodes-travel.

Thus O( ) bound of *loadKeys()* is O(nm).

2) What is the O( ) bound of *contains()* on T1?

When calling contains, you are inputting a key of length at most m. The algorithm then travels through m nodes at most, checking if they match on the fly.
It travels through m nodes at most.

Thus O( ) bound of *contains()* is O(m).

Now recall that, in *TrieNode.java*, the children of each node is implemented using an array of size k = | C | which is the maximum number of children. In many tree data structures, however, the set of children is implemented using a linked list. Suppose we were to modify **TrieNode.java** so that each node's children were implemented using a LinkedList, rather than an array. Let T2 denote the modified trie data structure. Again, let N be the number of nodes in T2. *Note that this is the same N as above since the number of* **TrieNode** *objects will be the same*.

3) What is the O( ) bound of *loadKeys()* on T2 ?

We are inserting n keys made out of m characters at most (N nodes in general) to the Trie T2.
Let check out the insert() method routine:

- Get the prefix.
- Fill in the missing nodes starting from the prefix.

The routine travels through m nodes, but in order to get those nodes it should go through all the previously recorded ones in the LinkedList at the parent node. The LinkedList contains at most k nodes. At max, the algorithm will then go through m*k nodes.

Inserting n keys of m length will take at most n*(m*k) nodes-travel.

Thus O( ) bound of *loadKeys()* is O(nmk).

4) What is the O( ) bound of *contains()* on T2 ?

When calling contains, you are inputting a key of length at most m. The algorithm then travels through m nodes, checking if they match on the fly. However to get the correct nextNode at each and every node, the algorithm will go through all the previously recorded ones in the LinkedList at the parent node, thus at max, it will go through m*k nodes.

Thus O( ) bound of *contains()* is O(mk).

Finally, we can compare the runtime of tries to the runtimes of binary search trees (BST). Suppose we were to consider a BST data structure to store the same set of keys. Each node in this BST would correspond to a key, such that the left (or right) child of each node is lexicographically smaller (or bigger) than its parent.

Note that the running time of operations on a BST depends on how “balanced" the tree is. (A tree is well balanced if the number of descendents of each left child is approximately the same as the number of descendents of each right child. This is not an exact definition, but it suffices for this question.)
Using the O( ) notation for *worst-case* and Ω() notation for *best-case*:

5) what is the O( ) bound of *loadKeys()* for a BST ?

> We are inserting n keys in the BST.
> If the BST is not well balanced, all the nodes will be either smaller or greater than the node we would like to insert.
> The algorithm will then have to go through each and every node of the BST, comparing lexicographically m times at most at every steps.
> Considering the fact that there is N nodes in the BST, O( ) bound of *loadKeys()* will be O(mN).

6) what is the Ω( ) bound of *loadKeys()* for a BST ?

> We are inserting n keys in the BST.
> If the BST is well balanced, we have a complete Binary Tree.
> The algorithm will then have to go through log_2() node of the BST, comparing lexicographically m times at most at every steps.
> Considering the fact that there is N nodes in the BST, Ω( ) bound of *loadKeys()* will be O(log_2(mN)).

7) what is the O( ) bound of *contains()* for a BST ?

> If the BST is not well balanced, all the nodes will be either smaller or greater than the node we would like to insert.
> The algorithm will then have to go through each and every node of the BST, comparing lexicographically at most m times at every step.
> Considering the fact that there is N nodes in the BST, O( ) bound of *contains()* will be O(mN).

8) what is the Ω( ) bound of *contains()* for a BST ?

> If the BST is well balanced, we have a complete Binary Tree.
> The algorithm will then have to go through log_2() node of the BST, comparing lexicographically at most m times at every step.
> Considering the fact that there is N nodes in the BST, Ω( ) bound of *loadKeys()* will be O(log_2(mN)).

Assume that these methods perform the same role as they do for tries i.e. *loadKeys()* populates a BST, and *contains()* searches for a key in BST.
Hint: Testing whether one string of length m is lexicographically bigger or smaller or the same as another string of length m requires at least one comparison and at most m comparisons.

Your Task

Submit a text file A3Q2.txt with answers to the questions. If you wish, you may write Omega( ) instead of Ω ( ).