Data Structures 89-120, FINAL EXAM
MOED B

Instructor: *Prof. Amihood Amir*
Length of Exam: 2 hours
Time: Sept. 4th, 2014, 15:30
NO OUTSIDE MATERIAL ALLOWED!!!

1. (25 points) We defined the *level* of a node in a rooted tree as follows: The level of the root is 0. The level of node x is the level of x's father $+1$.

   Describe an algorithm to compute the sum of the levels of all nodes in a tree. What is the idea of your algorithm? What is the time complexity of the algorithm?

   Answer:

   Remark: There are many ways to solve this. Some involving auxilliary fields defined on the tree, some recursive, some using data structures (stacks or queues). No one method was considered preferable.

   The general idea is to traverse the tree by DFS or BFS and add the levels. One possibility is to increase the level of a node whenever a son of the node is encountered for the first time and add that level to the sum. This can be done by recursive call with the level and root node of a subtree given as input to the recursive subroutine and the sum of the levels of the subtree returned. The initial call is to the root of the tree with level 0. The main part of the subroutine is a recursive call to all the children of the root, with the child and the input level+1 given as input and the sum of all returned sum returned.

   Other possibilities are DFS or BFS using stack or queue, resp. But then you need to be explicit about when exactly you do the summation, so that in implementations where elements are put more than once in the data structure you don't sum more than necesary.

   The time for all these traversals is linear, i.e. $O(n)$. If a recursive function was used, the time needs to be proven inductively or by finding the closed form of a recurrence.

   Errors:

   1. Anyone who had the idea of a linear traversal for writing the level and another traversal for summing the levels got 10 points.
   2. Did not observe that in a tree $|E| = |V|$ and thus making the time linear in the number of nodes. (-5 points)
   3. Missing details on when and how the level value is updated. (-5 points)
   4. Used but did not explicitly define auxilliary arguments and fields. (-2 points)
   5. Missing initialization. (-3 points)
   6. Assumes that the tree is binary (-6 points)
   7. Incorrect level calculations. (-5 points)
   8. Recursion without proof of linear time. (-7 points)
   9. Does not know complexity of DFS. (-10 points)

10. Technical problems in algorithm. (-10 points)
11. Incorrect recurrence formula. (-5 points)
12. Missing or faulty description of the tree traversal mechanism. (-15 points)

2. (15 points) Given a balanced coin. Define:
P_i = { the probability that i consecutive flips will be "heads" and the $i+1$st flip will be "tails". }
Q_i = { the probability that the first i consecutive flips of the coin will be "heads". }

Compute P_4, Q_4, and Q_5, and show that $P_4 = Q_4 - Q_5$.

Answer:

Since the coin is balanced then the probability of "heads" (or of "tails") is 1/2. Because the events are independent, the probability of i consecutive "heads" (Q_i) is the product of the probabilities, i.e. $\frac{1}{2^i}$. The probability of i consecutive "heads" followed by a "tail" (P_i) is $\frac{1}{2^{i+1}}$.

Therefore, it is clear that

$$Q_i - Q_{i+1} = \frac{1}{2^i} - \frac{1}{2^{i+1}} = \frac{2}{2^{i+1}} - \frac{1}{2^{i+1}} = \frac{1}{2^{i+1}} = P_i.$$

In our case, $i = 4$, thus $P_4, Q_5 = \frac{1}{32}$, $Q_4 = \frac{1}{16}$.

Errors:

1. 6 points were given to a correct computation of P_4.
2, 3 6 points were given for a correct computation of Q_4 and Q_5.
4. 3 points were given for showing that $P_4 = Q_4 - Q_5$.
5. Major computation errors. (-8 points)
6. Minor computation errors. (-3 points)
7. Does not calculate the probability of getting a "heads" on the first coin flip. (-7 points)

3. (10 points) Assume we have a hash table of size 7 (the entries are $0, ..., 6$). We are using the double hashing with probing method. Our hash function is:

$$h(k, i) = (h_1(k) + ih_2(k)) \text{mod } 7.$$

Where: $h_1(k) = k \text{mod } 7$, and $h_2(k) = (1 + k) \text{mod } 5$.

The following keys arrive: $8, 1, 15, 4, 16$. How will the hash table look?

Answer:

The aim of this easy question was only to see if people know the definition of double hashing with probing. For the correct result see Figure 1.

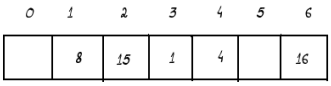

Figure 1: Solution to Question 3.

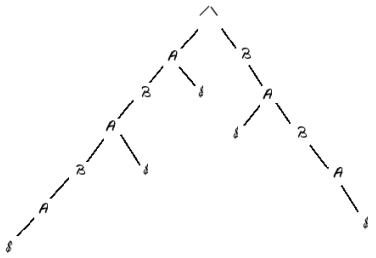

Figure 2: Solution to Question 4.

Errors:

1. Every item correctly placed gave 2 points.
2. Table of size 7. (-3 points)

4. (10 points) The *suffix* S_i of string $S = S[1], ..., S[n]$ is the substring $S[i], ..., S[n]$. The *suffix tree* of S is the trie of the suffixes $S_1, ..., S_n$. Draw the suffix tree of $S = ABABA$.

   Answer:

   Again, an easy question whose sole aim is to see if people know the definition of a trie. The answer can be seen in Figure 2.

   Errors:

   1. 1 point was given for starting with Λ.
   2. 1 point was given for ending with \$.
   3. Incorrect tree. (-10 points)
   4. 3 points were given for correctly identifying the suffixes.
   5. Compressed tree was drawn. (-3 points)
   6. Too many \$'s. (-2 points)
   7. Not enough \$'s. (-2 points)
   8. Not a tree. (-10 points)

5. We would like to use a dynamic data structure (records come in and out) where the keys are natural numbers. We need to answer queries of the following form:
Given a number j, find the key k that is closest to j (i.e. $|k-j|$ is smallest) where k is a key in the data structure.

   (a) (10 points) Assuming our data structure is a skip list, describe an algorithm that answers the query. Explain why your solution is correct and compute its time complexity.

   Answer:

   In a skip list, it is easy to answer this query. Simply look for input key j. If it is found, it is the answer. If not, we have arrived at the lowest level, where all keys reside in sorted order. Thus, our search for the closest key k to j, is a search on one "block" (the sequence between two keys in the level above it) on the lowest level. Since the size of these blocks is expected to be constant, and since the skip list is expected to have $O(\log n)$ levels, the expected running time for query is $O(\log n)$.

   Errors:

   1. Wrong time complexity. (-4 points)
   2. Does not acknowledge that the $O(\log n)$ running time is in the expectation. (-1 point).
   3. Compares if $|k_i - j| > |k_{i+1} - j|$ to decide whether to continue on a level or go down. This is a problem because for example, if $j = 10$ and the keys are $1, 11$, the algorithm would continue to check between 11 and the number that follows it, rather than go down a level. (-3 points) updated. (-5 points)
   4. This is an algorithm for a regular list, not using the special properties of a skip list. (-7 points)
   5. No details on how to handle a case where j is not in the list. (-4 points)
   6. No search algorithm (-6 points)
   7. No details on how to handle a case where j is in the list. (-2 points)
   8. An $O(n)$ time query processing algorithm. (-7 points)
   9. Does not give the time complexity but algorithm is correct. (-2 points)
   10. An $O(\sqrt{n})$ time query processing algorithm. (-6 points)
   11. Error in handling last two keys. (-2 points)
   12. Not a search on a skip list. (-10 points)

   (b) (30 points) Assuming the data structure is an AVL tree. Describe an algorithm that answers the query. Explain why it is correct and compute its time complexity.

   Answer:

   The idea is to look for input key j in the tree. If it is found, it is the answer. If not, we have arrived at a leaf in the AVL tree. We need to find the closest key k to j. The key point here is to prove:

   Claim: If j is not in the AVL tree, then the closest key k to j is on the path taken by the search for j on the search tree.

   Proof: Going down a path in the search tree, if we make a right turn at node v, it means that $j > v$ and therefore v is greater than the entire tree rooted at v's left son, so we need not consider the subtre to the left. Similarly, if we make a left turn at v,

then $j < v$ but v is smaller than the entire subtree rooted at v's right son, thus the right subtree need not be considered. The conclusion is that only the nodes on the search path are relevant to the query. ∎

Since the height of an AVL tree is $O(\log n)$, it means that we need only work $O(\log n)$ time to find the node whose value is closest to j.

Errors:

1. Algorithm only works if j is a key in the AVL tree. (-12 points)
2. $O(n)$ time query complexity. (-15 points)
3. $O(n)$ time query complexity but the claim is made that the time is $O(\log n)$. (-22 points)
4. Used the fact that the closest element lies on search path, but gave no proof. (-3 points)
5. Error in proving the closest key. (-15 points)
6. No proof that algorithm finds the closest key. (-20 points)
7. The closest key is not achieved (on one or the other side). (-15 points)
8. $O(\log^2 n)$ time query complexity. (-3 points)

GOOD LUCK