Data Structures 89-120, FINAL EXAM

MOED A

Instructor: *Prof. Amihood Amir*
Length of Exam: 2 hours
Time: July 17th, 2014, 15:30
NO OUTSIDE MATERIAL ALLOWED!!!

1. (20 points) Given a *free binary tree* (unrooted) with the condition that there is no vertex of degree 2. Prove that if the tree has k leaves ($k > 1$), then it has $2k - 2$ nodes.

   Answer:

   Remark: A similar problem was shown in the tutorial classes, but for rooted binary trees. Free trees were defined in the lecture and the definition even appears in the lecture presentations. Nevertheless, the object of this question was to see that you know how to write a proof, so the definition was explained in the exam for whomever asked.

   Proof: By induction on k.
   Induction base: For $k = 2$ the only possible graph where the condition holds is where the graph has two nodes connected to each other by an edge. Clearly this graph has $(2 \cdot 2) - 2 = 2$ nodes.

   Induction Step: The hypothesis is that every free binary graph of k leaves where the condition holds, has $2k-2$ nodes. We have to prove this for a graph of $k+1$ leaves. We use an auxiliary claim.

   Claim: Every free binary tree of k leaves, $k > 2$, has at least one node that is adjacent to two leaves.
   Proof of Claim: Let $v_0, ..., v_m$ be a simple path in the tree where both v_0 and v_m are leaves and where the length of the path is largest. Because $k > 2$ every such path has an internal node, and because of the degree condition every internal node has degree 3. Consider v_{m-1} one of its neighbors is v_m and the other is v_{m-1}. if must have a third neighbor x. x must be a leaf, otherwise it creates a longer path than $v_0, ..., v_m$ which contradicts the fact that $v_0, ..., v_m$ was a maximum-length path. ∎

   Back to the induction proof. Given a free binary tree T with $k+1$ leaves. Choose a node x that is adjacent to two leaves a, b. Construct a new tree T' where a and b and the edges to them are deleted. Tw has k leaves, thus by induction it has $2k - 2$ nodes. T has two more nodes than T' (a and b), thus T has $2k = 2(k+1) - 2$ nodes. ∎

   Note: An alternate proof is by reduction to the rooted binary tree case that we had proven in the tutorial. Below is a brief synopsis of how such a proof should proceed: Choose a leaf a adjacent to node x. Partition the free tree T to a tree with a single node a, and to a rooted binary tree rooted at x. Prove that x satisfies the conditions and apply the claim proven in the tutorial. Then add a.

   Note about the errors below: I was very lenient in deducting points. If people had a correct glimmer of an idea, they got points. As a consequence, full points were given even to proofs that are not very well written.

   Errors:

1. Induction step based on adding two leaves to a tree of k leaves rather than deleting two leaves from a tree of $k+1$ leaves. We had spoken in class about the pitfalls in this action. (-5 points)
2. In reduction method - problem with definition of rooted tree. (-5 points)
3. In induction step, chooses a leaf and deletes it. If the parent of the leaf is not adjacent to another leaf, it is "contracted". The contraction needs to be well defined and it needs to be proven that the remaining tree fits the required conditions. (-3 points)
4. In induction step, continues to remove leaves until a tree satisfying the requirements is achieved. No proof that this process ends in a reasonable tree, and no reasonable "putting together" of the various deleted nodes. (-10 points)
5. No proof of claim that there exists a node adjacent to two leaves. (-3 points)
6. Assumes that all internal nodes are adjacent to two leaves. (-4 points)
7. Starts induction base case at $k = 1$. (-10 points)
8. Starts induction base case at $k = 3$. (-3 points)
9. Error in induction base case, or no induction base case. (-10 points)
10. Assumes, rather than proves, the induction step. (-5 points)
11. Induction on degree of the tree. (-10 points)
12. No proof to induction base case. (-10 points)
13. Erroneous proof. Number of leaves in the tree is greater at end of proof that at the beginning. (-10 points)
14. Starts induction base case at $k = 4$. (-5 points)
15. Partitions the tree to two trees. Either no explanation of how, or erroneous explanation. (-8 points)
16. Incomplete proof of claim that a node adjacent to two leaves exists. (-1 point)
17. Induction hypothesis is applied to subtrees that don't satisfy the requirement. (-8 points)
18. Proof deals with "left" and "right" subtrees. They don't exist in free trees. (-20 points)
19. Does not know the definition of a tree. (-20 points)
20. Jumps in induction step from k to $k+2$. (-10 points)
21. Wrong handling of induction hypothesis. (-8 points)
22. Wrong or nonexistent induction step. (-10 points)

2. (20 points) Describe how one can sort n whole numbers from the range $\{0, ..., n^2\}$ in time $O(n)$.

   Answer:

   Note that this problem appeared in the assignments so there was absolutely no reason for anyone not to get the full points.

   The idea is that limited ranges may allow for faster sorting algorithms (e.g. bucket sort). All that is necessary is to represent the numbers by a pair of integers in the range $\{0, ..., n\}$ and then do *radix sort* on them. In particular, do two bucket sorts starting from the LSB. The stability of the bucket sort will guarantee a final sorting. So we have two bucket sorts, each costing $O(n)$.

Errors:

1. Uses regular bucket sort from 1 to n, no explanation of where to put numbers greater than n. (-15 points)
2. Uses bucket sort but assumes a constant range per bucket. Thus number of buckets $O(n^2)$. (-15 points)
3. Uses a pivot and then sorts to its right and left. Pivot is taken as $\sqrt{n}$) or the median. This means worst case of $O(n^2)$. (-15 points)
4. Uses radix sort but sorts from MSB to LSB. (-5 points)
5. Uses a claim that an array of length $O(\sqrt{n})$ can be sorted in constant time. (-15 points)
6. Does bucket sort blithely not noticing that $O(n^2)$ buckets are necessary. (-15 points)
7. Uses a hash table of size $O(n^2)$. (-15 points)
8. Implicitly sorts n elements in time $O(\log n)$. (-15 points)
9. Uses radix sort. Does not provide number of digits. (-8 points)
10. Does radix sort using a constant base. time is $O(n \log n)$. (-10 points)
11. Uses heapsort (not linear time). (-15 points)
12. Sorts by MSB and then time $O(x \log x)$ for each bucket. (-13 points)
13. Considers base n as a constant. (-3 points)
14. No time complexity analysis. (-10 points)
15. Basically does bucket sort by MSB and base n representation. Ignores the part withing the buckets. (-13 points)
16. Radix sort but confusion about the number of digits and the base. (-10 points)
17. Radix sort. No base given. (-10 points)
18. Algorithm that is linear only for uniform distribution of inputs. (-13 points)
19. Quicksort with median as pivot. Time $O(n \log n)$. (-15 points)
20. Bucket sort by n buckets, then counting sort, for time $O(n^2)$. (-15 points)

3. (15 points) Draw an AVL tree that satisfies the following three conditions:

   (a) The tree has exactly 11 nodes.

   (b) There are no pair of nodes that, if extracted one after the other, will cause the height decrease by 1.

   (c) There is no key whose insertion will increase the height by 1.

Answer:

The aim of this unbelievably easy question was only to see if people know the definition of an AVL tree. A number of trees are correct. For an example see Figure 1.

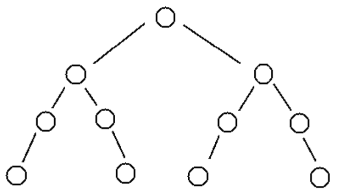

Figure 1: Example of solution to Question 3.

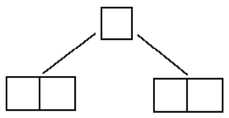

Figure 2: Solution to Question 4.

Errors:

1. Conditions (a) and (b) hold, but there is an addition that increases the height. (-8 points)
2. Conditions (b) and (c) hold but the number of nodes is not 11. (-8 points)
3. Not a valid AVL tree. (-10 points)

4. (15 points) Draw a 2-3 tree (a B-tree where the maximum number of children of a node is 3) that satisfies the following conditions:

   (a) The tree has two leaves.

   (b) If we add four keys the height will increase by 1.

   Answer:

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

   Errors:

   1. Not a 2-3 tree. (-15 points)
   2. Illegal operation on a 2-3 tree. (-8 points)
   3. Insertion not done by 2-3 tree insertion algorithm. (-8 points)
   4. Inserting 4 keys does not increase height. (-10 points)
   5. A 2-3 tree with 6 keys does not look like this. (-8 points)
   6. There are more than two leaves. (-15 points)

7. There is only one leaf. (-15 points)
8. A 2-3 tree with 5 keys does not look like this. (-7 points)
9. A 2-3 tree with 4 keys does not look like this. (-8 points)
10. A 2-3 tree with 7 keys does not look like this. (-10 points)

5. We have static data (the data is given initially and does not change) where the keys are natural numbers. We would like 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 entire data structure fits into memory, what data structure would you use? Explain why your solution is correct and compute its time complexity.

   (b) (20 points) Assuming the size of our data structure is 100GB, that the memory space is 2GB, that every read from the disc is a block of 1GB, and that the time for reading a block of data from the disc is equal to 2000 operations in memory. what data structure would you use? Describe an algorithm that answers the query. Explain why it is correct and compute its time complexity.

   Answer:

   This question tests the essense of understanding data structures. While the answer is, in fact, quite simple and does not require fancy data structures, it serves to emphasize the point that the existence of data structures is to enable efficient queries. In this context, any solution whose time is $O(n)$, where n is the size of the data structure, should have earned 0 points. Nevertheless, since there was an additional insight involved, that of hierarchical storage, I gave partial credit to $O(n)$ solutions.

   (a) Since the data structure fits in memory, then conceivably, any data structure that allows logarithmic search would do. Nevertheless, we would like to minimize space and time. since we know that the data is static, then the best data structure is a table, sorted by the keys. A binary search on the table either finds j, in which case that is the answer, or does not find j, but that means it narrowed the range to one or two elements and none of them is j. However, a constant check of which of the elements in the final range or immediately adjacent to it is closest to j will provide the correct answer. The time is $O(\log n)$ memory operations, where n is the number of records.

   Errors: I decided to grade this problem conceptually rather than detail oriented. Thus, any reasonable data structure, if all details were handled correctly, entitled to 8 points out of 10. On incomplete details I deducted 2 points off the maximum total (i.e. 2 out of 10 if a sorted table was used, and 2 out of 8 if another reasonable data structure was used. After all this, if the final query was of complexity $O(n)$, I awarded half the number of remaining points.

   1. Balanced search tree. (8 points)
   2. AVL tree. (8 points)
   3. Skip list. (8 points)
   4. Hashing. Since we did not discuss the very heavy question of order-preserving hashing, the students have no way of finding the closest key. Thus using hashing is conceptually totally incorrect for this question. (0 points)

5. $O(n)$ time query. (half the remaining points)
6. Incomplete details. (-2 points)
7. Sorting that is dependent on the key size (such as radix sort, bucket sort, or tries). (5 points)
8. A solution that works only if j is a key. (-5 points)

(b) As discussed in class, the B-tree is the natural answer to hierarchical storage. However, to calculate the complexity for our example we need to decide what parameter to use. Clearly, the more shallow the tree, the better. Since we have 2GB memory, and since every read is a block of 1GB, it is natural to make every node of length 1GB. This means that we can have a root of size 100, and one level of leaves of size 1GB each. This would make the complexity equal to two disc acesses.

However, we can do better yet, by having a single disc access. Consider the data on the disc in sorted order. However, keep **in memory** a table of size 200 that gives for every disc block the smallest and largest key in every disc block. Given a key j, find the block it is in. If it is larger than the largest key in block i (say k_1) and smaller than the smallest key in block $i+1$ (say k_2) then we can calculate the solution without a disc read (take the key $k \in \{k_1, k_2\}$ that minimizes $|k-j|$). Otherwise read the appropriate block and do a binary search as in question 5.(b). The time for a search is then one disc read and $O(\log 1,000,000)$ memory operations. Note that this is negligible relative to the one disc read.

Errors:

This one was also graded conceptually. However, since the only logical data structure to use here, other than the table method, was the B-tree, use of any other data structure gave no points.The only exception was skip lists where the student explicitly kept the higher levels of the list in memory and only the bottom level on the disc. This solution is still not as economical as the table but it is decent. B-tree started with 14 points because it still takes twice as long as the table method (unless the root is in memory, which no one did). Points were deducted for various errors and, as in the case of 5.(a), if the query time ended up $O(n)$ then half the remaining points were deducted.

1. B-tree. (14 points)
2. Heaps, AVL tree, binary search tree, skip list. (0 points)
3. Skip list but keeping the higher levels in memory. (14 points)
4. Problems with details. (-5 points)
5. Hashing. Again, using hashing is conceptually totally incorrect for this question. (0 points)
6. $O(n)$ time query. (half the remaining points)
7. Sorted array, but incorrect algorithm. (-12 points)
8. B-tree with no details, (-15 points)
9. Binary search on disc blocks. (15 points)
10. A solution that works only if j is a key. (10 points)

GOOD LUCK