Laboratory 10: Cover Sheet

Name Catherine Pollock Date 10/27/14

Section 1001

Place a check mark in the *Assigned* column next to the exercises your instructor has assigned to you. Attach this cover sheet to the front of the packet of materials you submit following the laboratory.

Activities	**Assigned:** Check or list exercise numbers	**Completed**
Implementation Testing	✓	
Programming Exercise 1	✓	
Programming Exercise 2		
Programming Exercise 3		
Analysis Exercise 1	✓	
Analysis Exercise 2	✓	
	Total	

Laboratory 10: Analysis Exercise 1

Name Catherine Pollock Date 10/27/14

Section 1001

Given a hash table of size *T*, containing *N* data items, develop worst-case, order-of-magnitude estimates of the execution time of the following Hash Table ADT operations, assuming they are implemented using singly-linked lists for the chained data items and a reasonably uniform distribution of data item keys. Briefly explain your reasoning behind each estimate.

insert O(N/T)

Explanation:

In the case that a hash table consisted of singly-linked lists and the data item keys were reasonably distributed, it would take O(N/T) to insert the item. This would be the case when the element must be inserted as the last element of its specific linked list (decided by the hash), meaning the insertion search algorithm checked every item of data in that linked list. Average, rather than worst-case, would of course be different, and not have to search so many elements to insert.

retrieve O(N)

Explanation:

In the instance that a hash table consists of singly-linked lists and the data items are reasonably distributed, the worst-case order of magnitude estimate would be N. In the worst case scenario, the element searching for would not be in the hash table or it would be the last element of the last list. This means that the retrieve would have to check every element of every list, making the worst case scenario algorithm O(N). Average, rather than worst-case, would of course be different, and not have to search so many elements to insert.

What if the chaining is implemented using a binary search tree instead of a singly-linked list? Using the same assumptions as before, develop worst-case, order-of-magnitude estimates of the execution time of the following Hash Table ADT operations. Briefly explain your reasoning behind each estimate.

insert O(N/T)

Explanation:

In the worst-case scenario, the binary search tree would resemble a linked list (where each node has one child) and the new data item to be inserted would search through the tree like it were a linked list. In the worst-worst case scenario it would be inserted at the bottom or final node of the binary search tree, so it would have traveled through all of that nodes in that tree. Since it only had to be inserted at that specific table element, the execution time would be represented by O(N/T). This is only the worst case-scenario, though. The average (and worst case) execution time would be O(1) to find the tree to insert to, since the element would know exactly which tree to go to. The insertion into the tree would average something (not sure exactly) along the lines of O(($\log_2$N) / T), unlike the worst case.

retrieve O(N)

Explanation:

Retrieving from a worst-case scenario of a hash table of binary search trees would be exactly like retrieving from a hash table of linked lists. The worst case scenario would have a binary search tree where each node has only one child, resembling a linked list. Also, in the worst-case scenario, each tree would have to be searched until the element is not found, or if it is found in the last tree as the last node. That would mean every element of every tree was searched. That means execution time can be estimated by O(N). Of course, this is not the average case. In a more average case, the element would be found in a tree more towards the beginning of the table, making the execution time estimate also something to do with $\log_2$N.

Laboratory 10: Analysis Exercise 2

Name Catherine Pollock Date 10/27/14

Section 1001

Part A

For some large number of data items—e.g., *N*=1,000,000—would you rather use a binary search tree or a hash table for performing data retrieval? Explain your reasoning.

For such a large number of data items I would use a hash table. The hash table would probably be made of something to handle collision, such as chaining, but the hash table would have a hash function that would make data retrieval easy, since we would know exactly which element to go to in the table.

Part B

Assuming the same number of data items given in Part A, would the binary search tree or the hash table be most memory efficient? Explain your assumptions and your reasoning.

Although the hash table would be more time efficient for data retrieval, a single binary search tree would be more memory efficient. This is because the hash table would be creating more trees which are stored in array. Although they would have the same number of elements total, the hash table itself would use more space.

Part C

If you needed to select either the binary search tree or the hash table as the general purpose best data structure, which would you choose? Under what circumstances would you choose the other data structure as preferable? Explain your reasoning.

I would personally choose a binary search tree, because it is more straightforward, viewable, user friendly, and hash tables often use it. By that I mean that each element in a hash table is a binary search tree. Although this is not always the case, it was so in our project and makes me lean towards thinking that binary search trees are more flexible in terms of use.