Programming Assignment 5

Syed Abeer Hasan Zaidi

April 20, 2019

Program Description and Purpose of Assignment

The Assignment was to implement the Skip List ADT in c++, in order to improve our knowledge of Skip Lists and to demonstrate our understanding of what we had learned from the lectures. The program was to have insert, delete, and search operations, and would utilize nodes in order to travese the Skip List. The program should also compute the number of comparisons performed in each operation, and print it out if the number of nodes was less than 16. The program should also be able to calculate and return the average insertion and deletion cost. I compiled my skip list using a makefile, and ran it using the command ./run-slist in terminal.

The Data Structure

A skip list is essentially a series of linked lists connected to one another, with some common elements. It uses a quad node, which connects to nodes to the right, left, "above", and "below" itself. In my implementation of the skip list, I set a maximum height of 6. This is because chances of an inserted node reaching a height higher than that, which would call for the allocation of another level for the skip list, is less than 1%. The insertion and search functions of the data structure start from the top, and traverse down the several levels of linked lists, to find or insert the element with a specified value. For the insertion operation I made coinToss() function thatwould continuosly generate a random number(mod 2), increment a counter everytime this value equaled 1, and stop if the value wasn't equal to 1. With this coinToss function I was able to determine the maximum height of a certain element being inserted. The delete function utilizes the search function, which returns a pointer to the node at the base of the list, and works its way up the levels of the list to delete the element.

Insert, Search, and Delete Costs

Insert

The insertion cost was determined, by using a counter that incremented whenever a comparison was made between the value at one node against the value we were inserting.

Best case: O(logn)
Average case: O(logn)
Worst case: O(n)

Search

The search cost was determined, by using a counter that incremented whenever a comparison was made between the value at a certain node against the value we were searching for.

Best case: O(logn)
Average case: O(logn)
Worst case: O(n)

Delete

The deletion cost was determined, by adding the search cost for a certain node, to the height of that node. This is because the search function was used in the implementation of the delete function. So in the end it would have the same best, average, and worst case as the search function.

Best case: O(logn)
Average case: O(logn)
Worst case: O(n)

Additional Questions

1. How likely is it that an item will be inserted into the n^th level of the skip list?

   The probability is $(\frac{1}{2})^n$.

2. If you were to increase the probability of getting a "heads" (positive result, keep flipping the "coin"), what would this do to the average runtime of insert, search, and delete?

   Since this would increase the highest point of most of the elements in the list, it would increase the average runtime for all the operations. Their runtimes would be closer to that of a singly linked list than a skip list.

3. How does the order of the data (sorted, reverse sorted, random) affect the number of comparisons?

Sorted has the highest number of comparisons, since in my implementation we look for an element larger than it before inserting a node at that level, so a sorted input would have to be traversed more at each level.

Unsorted has the smallest runtime, since an element larger than the one being inserted will always be found after the Head of a level.

Random has a number of comparisons between that of sorted rather than unsorted, so it would be a good indicator of the average number of comparisons for a certain operation

4. How does the runtime compare to a Binary Search Tree for the insert, search, and delete operations?

   The runtime for a BST and a Skip list is similar for insertion and search, but deletion would be worse since you would have to find a new root to replace the deleted element.

5. In what cases might a Binary Search Tree be more efficient than a skip list? In what cases might it be less efficient?

   A Binary Search tree would be better for larger lists, since a large skip list would have search costs closer to that of a Linked List as the lower levels of the list would get really crowded, and the upper levels would also get more filled. In this scenario a binary search tree would be a much better option.