CSCE 221 Assignment 5 Cover Page

First Name: Adrian Last Name: Gamez-Rodriguez UIN:126009409

User Name: adriangamez E-mail address: adriangamez@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more on Aggie Honor System Office website: http://aggiehonor.tamu.edu/

Type of Sources			
People	TA in Lab		
Web Pages	Stackoverflow.com	Geeksforgeeks.org	
Printed Material	Textbook	Lecture Slides	PDF from Andrew Lam

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work. On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.

Your Name: Adrian Gamez-Rodriguez Date:4/14/19

The purpose of this assignment was to implement a Skip list using STL containers and iterators. The purpose of creating a skip list is to guarantee a time complexity of O(logn) operation of insertion, searching, and deleting. A Skip list is useful in that it can skip some nodes and find a key faster than just traversing through a linked list. So, my program structure declares and defines a struct called Node. This node is the building block of the Skip list. Then I declare and define a class Skiplist that is the actual data structure to implement the skip list. It uses a combination of STL containers of vector and lists. Overall the structure is a vector of linked list of nodes. I decided to have a fixed max level of 10 lists. Each list has two nodes in the beginning, which are the negative infinity node and the positive infinity node. From there we just insert, delete, or search.

To run the program: go to the folder named code

Then: type command ***make all*** then type command: ***./main***

Note: All results for all input files should appear on screen.

The implementation of the Skip list data structure was attainable by using STL containers and iterators. The class Skiplist has a structure that is a vector of linked lists of nodes. That is to say, it is a vector of lists that contain the struct Node. The struct Node is the fundamental block of the skip list. Inside the Node struct it contains the data int and an iterator of list of nodes serving as a connection downward. So, then each list has linked Nodes and each list can be found by traversing through the vector that contains all 10 lists.

The method I used to calculate the insertion cost is declaring two private member variables. One of them kept track of the number of insertion calls made for the entire skip list. The other keeps track of the running total of comparisons for all insertion calls made. Then I simply divided total comparisons by the total calls made for every value inserted.

The method I used to calculate the average search cost was to create a function that called the search function for every valid node at the bottom row, as it would have been guaranteed to find that value in the skip list. I defined two private member variables to keep track of the running total of the comparisons made for every function and the number of functions called made. Then I simply divided the running total by the number of calls made.

Again, I defined two private member variables one to keep track of total number of comparisons made in every deletion function call and one to keep track of the number of function calls made for the entire skip list.

Time Complexities:

Insert Best Case: O(logn)

Insert Average Case: O(logn)

Insert Worst Case: O(n)

Search Best Case: O(logn)

Search Average Case: O(logn)

Search Worst Case: O(n)

Delete Best Case: O(logn)

Delete Average Case: O(logn)

Delete Worst Case: O(n)

(a) How likely is it that an item will be inserted into the nth level of the skip list?

The probability is $\frac{1}{2^n}$ for an item to be inserted into the nth level, so that's why inserting to a high level is statistically low.

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

For the all functions the average runtime would decrease because there are more nodes to be able to skip to get to the target. Because every function you need to compare to get either to the correct place or get to the target. In all cases you still need to traverse the skip list and more nodes on the higher levels means more skipping.

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

Sorted data would have decrease the number of comparisons, because it would skip nodes so it would take less comparisons. Reverse sorted data would increase the number of comparisons, because it would take more comparisons to find the right place. Random data would be the average of the number of comparisons of sorted and reverse sorted data.

(d) How does the runtime compare to a Binary Search Tree for the insert, search, and delete operations?

Generally, both data structures are very similar in terms of time complexity as the insert, search and deletion all have O(logn) running time. However, it would depend on whether the tree was balanced or not.

(e) In what cases might a Binary Search Tree be more efficient than a skip list? In what cases might it be less efficient?

In the case that the Binary tree is perfectly balanced then binary tree would be more efficient, because running time would be guaranteed be O(logn). In the case that if the Binary tree is one sided then the skip list would be more efficient.