**Kirsten Madina**

**Section 502**

**Programming Assignment 5**

**1. A description of the assignment objective, how to compile and run your programs, and an explanation of your program structure (i.e. a description of the classes you use, the relationship between the classes, and the functions or classes in addition to those in the lecture notes).**

The objective of this assignment is to implement a skiplist using iterators instead of pointers. The program is implemented on a .h file ‘SkipList.h’ and a .cpp file ‘SkipList.cpp’. To compile the program, use the command **g++ SkipList.cpp -std=c++11** followed by the command **./a.out**.

The classes in these files include struct Node, which has two data members: **int** key & std::**list<Node>::iterator** down. The files also include the class and class definitions for class SkipList, which has four private data members (**vector<list<Node>>** skiplist; **vector<int>** searchcosts, **vector<int>** insertioncosts, **vector<int>** deletioncosts) and five public member functions (**void** insert(**int** a), **list<Node>::iterator** search(**int** a), **int** remove(**int** a), **void** printskiplist(), **void** avgcost()).

Two helper functions were also programmed in this assignment: **int** coinflip() & **int** average(vector<int> v);

**2. A brief description of the data structure you created (i.e. a theoretical definition of the data structure and the actual data arrangement in the classes).**

A skip list is a randomized data structure type involving a hierarchy of linear data structure types (such as linked list or array, etc) where each level of the hierarchy is connected to the level below it. Insertion into this data type is random; a random number generator determines the row in which the new node or element is inserted, and it is then inserted in every level below the initial level. Elements are inserted sorted so that each node/element is less than the node/element to the right of it.

I implemented a skiplist by creating a struct Node and using that as elements in a vector of lists. I have a fixed number of levels (10). I use iterators in order to travers through each list and connect each lest to the one below.

**3. A description of how you implemented the calculation of (a) insert cost (b) search cost (c) delete cost.**

As mentioned before, the SkipList class has three vector members containing the number of comparisons for each utilization of the search(), insert(), and remove() functions. Each time the function is called, an integer **comp\_num** counts the number of comparisons within the function and is pushed back into the corresponding vector. A helper function **average(vector<int>** a)takes the sum total of each element in the vector and divides that by the number of elements in the vector. The SkipList member function **avgcost()** calls the average function for each of the member vectors of integers.

**4. Best case, worst case, and average case theoretical runtimes (Big-O) for the insert, search, and delete functions.**

The Best Case and Average case are when the elements are inserted into varied rows.

The Worst Case of a Skip List is when the all inserted elements are inserted into the same row based on the random number generator/coinflip.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Best Case** | **Worst Case** | **Average Case** |
| **Insert** | O(logn) | O(n) | O(logn) |
| **Search** | O(logn) | O(n) | O(logn) |
| **delete** | O(logn) | O(n) | O(logn) |

**5. Additionally, answer the following questions:**

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

Using a fixed height of my skip list of 9 (10 rows) and a coinflip, the probability of it occurring can be found (1/2)^10 = .001 or .1% chance.

**(b) 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?**

By increasing the probability of heads, you increase the chances of a node being inserted into one of the higher levels of the skip list, and therefore the average runtime of the three functions would increase.

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

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

Best case and worst case for all the three functions are the same for each of the three functions.

**(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 an unbalance binary search tree, a skip list would be more effective. If the skiplist however is worst case and so its elements are all initially on the same row, it would be less or equally as inefficient.