CSCE 221 Assignment 5 Cover Page

First Name Jacob Last Name Hamilton UIN 725009698

User Name jake7054 E-mail address jake7054@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				
Web pages (provide URL)		cpp website		
Printed material		book		
Other Sources				

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 Jacob Hamilton Date 04/14/19

CSCE 221 Assignment 5

Due to eCampus April 11th at 11:59pm

Objective

This is an individual assignment involving implementation of the skip list ADT in C++.

Program (70 points)

- Write a program in C++ that efficiently implements a skip list that holds integers. Your program should:
  1. Accurately implement the skip list ADT using a random number generator and nodes that contain an integer as well as the addresses of adjacent nodes to the left, right, up, and down.
  2. Correctly implement the Insert, Search, and Delete operations.
  3. Read a series of unique, newline-delineated integers from a file and insert them (one at a time in the order provided) into your skip list. If the number of nodes is not greater than 2^4, print out the number of comparisons for each insertion, as well as the level at which each number is inserted (this should vary with each execution of your program!)
  4. Repeat this process for each input file in sorted, "perfect", and random order (these are the same input files you used for Assignment 4).
  5. If the number of nodes is not greater than 2^4, print a representation of your skip list to the console. Note: a simple series of space-separated numbers for each level is sufficient here. Overloading `operator<<` may be helpful but is not required.
  6. Remove all items from the skip list, and, if the number of nodes is not greater than 2^4, print out the number of comparisons for each deletion. (Hint: the number of comparisons required for deletion should be equal to the number of comparisons required to search for the node to be deleted)
  7. Calculate the average costs for insertion and deletion (sum up the number of comparisons for each operation and divide by the number of operations) and print them to console.
- Example:
  - Input data:

    ```
    9
    2
    7
    12
    ```

  - Create a skip list:
    Add 9 to skip list at level 1, # comparisons
    Add 2 to skip list at level 1, # comparisons
    Add 7 to skip list at level 0, # comparisons
    Add 12 to skip list at level 2, # comparisons

  - Print skip list starting from the top level (instead of ∞, print "infinity"):
    $-\infty$, 12, ∞
    $-\infty$, 2, 9, 12, ∞
    $-\infty$, 2, 7, 9, 12, ∞
  - Delete skip list:
    Delete 12 from skip list, # comparisons
    Delete 7 from skip list, # comparisons
    Delete 2 from skip list, # comparisons
    Delete 9 from skip list, # comparisons
  - Average insert cost: #
  - Average delete cost: #

  Done!

- Hints:
  - Read the text and slides about skip lists!
  - You will want to create a class for your skip list that holds all of the data members and relevant functions.
  - You may either dynamically allocate new levels as needed, or hard-code a pre-determined number of levels for your skip list. Note: If you do fix the size, make sure your skip list is large enough that it would be very improbable for the size to be exceeded, and be sure to include guards just in case it is!
  - In implementing the underlying data structure, you may use standard containers (vector/list) from the STL, or any other implementation using basic containers (or not!) that you wish.
    * If you use STL containers, you will have to deal with iterators. Sources for information on iterators: Dr. Stroustrup's textbook, cplusplus.com on iterators (http://www.cplusplus.com/reference/iterator/), list element iterators (http://www.cplusplus.com/reference/list/list/). You will also need to create your own struct to hold the extra pointers – the payload of your std::list nodes can hold them this way!
    * If you use the code from Assignment 3, you can modify the node class to hold your additional pointers (up and down).
    * In either case, a vector of linked lists is a reasonable underlying data structure for your skip list.
  - As C++ does not have a default integer representation for positive and negative infinity, "Edge" nodes may be represented by maximum/minimum data values. For a C++ integer these can be retrieved by `#including<limits>` and using `numeric_limits<int>::max()` and `numeric_limits<int>::min()`.
  - The ctime library (`#include <ctime>`) is helpful for random number generation. Use `srand(time(0))` to seed your number generator, and the `rand()` function to get your "coin flips." Or you can use the C++ class random (`#include <random>`).
  - You may want your search function to return a pointer/iterator to the item it finds (this can simplify your delete function); you may also have to pass your insert function an integer by reference so that you can keep track of the comparison count.
  - In order to remove items from your skip list, simply call the delete function using the numbers in the file (you can also store the numbers in a vector or somewhere when you read them in so that you don't have to read them in twice). DO NOT implement a function that deletes arbitrary elements from the skip list!

Report (30 points)

Write a brief report that includes the following:

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).

   Compile the Main.cpp, SkipList.h, and SkipList.cpp with c++ 11. It takes in 4 files: random.txt, sorted.txt, reverse.txt, and 4r. When running it will compute the average costs of random, reverse, and sorted. Each have 1000 unique numbers. It will print these to the screen, and nothing else as the print flag is off. Then it runs the 4r file which holds 15 numbers. It prints each insertion, search, and deletion of it, prints the list, and it also prints the averages.

   The deleteItem function deletes an element in the list. It uses a helper function not out of necessity but to make it cleaner. The helper function recursively finds and deletes the item. Much like the search helper, and the insert helper functions. stack() is the function that does the "coin tosses" and adds verticality to the list. The variable "cost" holds the cost of the last function call. So if insert it called it will hold insert cost and if delete is called it will hold delete cost. At the end of the call it will store the cost in its respected vector, ex "deleteCosts". The getAverage functions will use these vectors to return the average cost of a certain operation.

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).

   This is a skip list. It is designed to be more efficient in iterating through it in large data sets. It will do any operation insert, delete, and search, in a log(n) time. When an element is inserted it will use a .5 probability to decide whether to add a node above it or not. When iterating through it, it will start from the top and go towards the bottom.

3. A description of how you implemented the calculation of

   (a) insert cost - A "cost" variable holds the value of all the comparisons made when running an insert operation.

   (b) search cost - A "cost" variable holds the value of all the comparisons made when running an search operation.

   (c) delete cost. - A "cost" variable holds the value of all the comparisons made when running an delete operation.

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

   O(log(n)) for all functions in average case. O(n) for all functions in the worst case. O(1) for all functions in the best case.

5. Additionally, answer the following questions:

   (a) How likely is it that an item will be inserted into the n^{th} level of the skip list?

   2^{-n}provided that the bottom is the 0th level.

   (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?

   It would increase the average run times.

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

   The order of the data did little to affect the number of comparisons.

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

It is faster than a binary search tree in all categories. Except for the random worst case.

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

A BST elimates the uncertainty by not having randomness to its algorithm. It also takes up significantly less space.