CSCI 335 Section 2
Programming Project 2

Due: Friday, November 13, 2015 @ 11:59pm

1 Introduction

In this assignment, you have to implement a Binomial Queue that allows fast `remove` and `decreaseKey` operations. To ensure that these operations are indeed fast, we need a fast method of locating the element in the binomial queue (so that they can be percolated up). The most efficient way to implement this is to keep a `(key, pointer)` key-value pairs in a hash table, where `pointer` is a pointer to `key`'s node in the binomial queue. Locating an element in the queue then becomes an $O(1)$ operation.

Code from the textbook for the binomial queue class and a quadratic probing hash table has been provided to you. You can extend the binomial queue's functionality by adding the functions for `remove` and `decreaseKey`.

For testing, add the strings provided in the accompanying text file.

2 Tasks

Complete the following tasks:

1. Implement the binomial queue. Implement a hash-table of the keys that are already in the priority queue. So, for any pair `(key, pointer)`, where `pointer` is the pointer of the the node that holds `key` in the priority queue, hash on the `key`, and store the pair `(key, pointer)` in your hash table.

   Note that the examples of hash tables we have seen in class contain only a single element, and we hash the element. However, in real situations, each entry in a hash table is a key and value, and the hash depends only on the key.

   Also note that for every `insertion`, `deleteMin`, or `merge` the hash table needs to be updated as well. As part of this implementation you have to create a private function `<Pointer to bq node> find( <Type of Key> key)`. This function will return the pointer `p` that points to the node of `bq` that holds the key, or `nullptr` if key is not in the `bq`.

2. Count and print out the total number of comparisons and assignments executed for the insertions of all the N elements into the binomial queue.

3. Test the `deleteMin()` operation by applying a sequence of 10 deleteMin() and by printing out the result.

4. Test the function `find()` as follows: Prompt the user to input a string `key`. Execute the private function `find(key)`. If find returns a pointer to a node that indeed holds key printout that find() was successful. Otherwise, printout that `find()` did not find the key.

5. You are ready to implement now the `remove` and `decreaseKey` operations. Keep in mind that `remove` is a simple extension of `decreaseKey`: first decrease the key of the target so that it percolates to the top (i.e., becomes the minimum key), then simply call `deleteMin`. Note that percolating up involves accessing (and possibly moving) the parent node. You may modify the given code so that each node keeps a pointer to its parent.

   To implement `decreaseKey`, for a given key, find its position `p` (`p` is a pointer) in the priority queue using the hash table. If the key is found percolate it to the top of the binomial queue. You are free to search the internet for pseudocode of the decreaseKey algorithm for binomial queue (however, do try to spend some time thinking about how it can be done).

6. Test your implementation by applying a sequence of `remove(key)` operations, and verify their correctness. For instance after `remove(key)`, `find(key)` should return "not found". Prompt the user 5 times to input a key. Execute the `remove(key)`operation and verify (by printing whether the removal was successful or not).

3 Other Information

Like Programming Project 1, you have to split the given source code in to .cpp and .h files, and include a Makefile (I have not included one in this project).

You will be making your own class/struct to store the `(key, pointer)` pair and there is no default hash function for it. You can modify the `myhash()` function so that it returns the default hash for the string member variable.

Ideally, you should have one class (say "BinQueuePlusPlus") that contains a binomial queue and a hash table, and exposes the functions of the binomial queue. This allows us to encapsulate the entire structure in one class, and when using this class we wouldn't have to worry about two distinct data structures (hash table and queue). Try to keep the actual binomial queue as "pure" as possible, i.e., it shouldn't be involved in looking up the hash table when decreasing a key. This is BinQueuePlusPlus' job, and the actual binomial queue should just take a pointer to a node and percolate that node.

Submit your work in zip file named YOUR_LAST_NAME.zip.