CSCI 335 Spring 2023
HW4: Binomial Queues & Hashing
65 points for deliverables (autograder) + 35 points for design

Total: 100 points

Due date: 4/27/2023
Gradescope will open on 4/17/2023

1. **Read and follow the contents of Programming Rules document on the blackboard (Course Information Section).**
2. **Submit only the files requested in the deliverables at the bottom of the description to Gradescope by the deadline.**
3. **Please note we will be using code comparison software across all submissions to ensure academic integrity requirements are strictly followed.**

Learning Outcome: The goal of this assignment is binomial queues associated with hashing.

Mutable binomial queues

Your task is to combine the implementation of a priority queue (binomial queue here) with that of hashing. To achieve this, one of the private members of your binomial queue class should be a hash-table. Use the Binomial Queue implementation provided. The goal is to be able to search for a specific key in the binomial queue (i.e. not only the minimum key), and to be able to remove a specific key (i.e., not only the minimum key).

Implementation hints: Keep a hash-table of the keys that already in the binomial queue. Use the quadratic hashing implementation provided. For any pair `<key,p>`, where p is the position of the key in the binomial queue (i.e. the pointer of the node that holds the `key`) , hash on the key, and store the pair `<key,p>` in your hash table. Any time you insert key into the binomial queue you should also insert the pair `<key,p>` into the hash table. Similarly, every time you delete a key from the binomial queue, you should also delete it from the hash table. To implement the removal of a given element, you have to add to each node in your Binomial Queue implementation a pointer to its parent node.

Below are specific instructions of what needs to be changed/added. Make sure you use the exact function names as described below, with the correct case (for example `Remove()` and not `remove()`). Otherwise, your main files will not compile.

Changes in `BinomialQueue` class in file `binomial_queue.h`:

a) Add a hash table as a private member variable:
HashTable<Comparable, BinomialNode*> positions_table_;

b) Add a parent pointer to the `BinomialNode`:
```
struct BinomialNode {
   …
  BinomialNode *parent;
  …
}
```

c) Change the return type of insert to bool, i.e.:
`bool insert(const Comparable &x);`
The insert will first search within the `positions_table_` for `x`. If `x` is in the `positions_table_` already, just return `false`. Otherwise, `x` will be inserted and `true` will be returned.

d) Add the public function:
`bool Find(const Comparable & x) const;`
That will return true if `x` is in the `positions_table_` and false otherwise.

e) Add the public function:
`bool Remove(const Comparable & x);`
The function first checks whether `x` is in `positions_table_`. It returns `false` if the item is not there.
Otherwise, removes `x` from both the binomial queue and the hash table. Note that the removal involves the use of the parent pointers. Also note that the removal may change the values of some of the elements within the hash table.

f) Make sure to keep the parent pointers and hash table are correctly maintained during all operations. In particular, make sure to update the parent pointer in function `CombineTrees()`.

Changes in `HashTable` class in file `quadratic_probing.h`:

a) Add a second template parameter:
```
template <typename HashedObj, typename HashedValue>
class HashTable {
  …
}
```
The first parameter will operate as before – no changes need to be made. The second parameter will contain the value for each hash key.

b) Add the value parameter in HashEntry:
```
struct HashEntry {
  …
  HashedValue value_;
  …
}
```

c) Modify the insert function, so that it now takes the value parameter as well:

`bool Insert(const HashedObj &x, const HashedValue &value);`
The `Insert` function will now also insert `value`.

d) Add a public Find function:
`bool Find(const HashedObj &key, HashedValue &found_value) const;`
The function will return `true` if `key` is in the table and `false` otherwise. If the key is in the table `found_value` should contain the value associated with the key (note that it is passed by reference).

e) Add the ability to change the `value` associated with a given (you can provide your own public function with the appropriate parameters for that).

Part 1 - search (40 points)

Implement the mutable binomial queue as described and test the search functionality.

You are given the file `test_insert_and_search.cc`

You should keep the `main()` and `TestInsertAndSearch()` as is. They will call the functions of the binomial queue what we described before. You can update these functions if you want to debug, but you should keep them as is for the submission. We will compile your code with this file on our side.

To run the program run:

```
./test_insert_and_search <input_file_to_create_queue>
<input_file_to_check_search>
```

The program reads all the integers in `<input_file_to_create_queue>` and insert them into the mutable binomial queue. It then reads the integers from file `<input_file_to_check_search>` and checks the search functionality.
By looking into function `TestInsertAndSearch()` you can see what is expected to be produced. We will give you input as well as expected output files.

Part 2 – remove (40 points)

Now we will test the remove functionality.

You are given the file `test_insert_and_delete.cc`

You should keep the `main()` and `TestInsertAndDelete()` as is. They will call the functions of the binomial queue what we described before. You can update these functions if you want to debug, but you should keep them as is for the submission. We will compile your code with this file on our side.

To run the program run:

```
 ./test_insert_and_delete <input_file_to_create_queue>
<input_file_to_check_delete>
```

The program reads all the integers in `<input_file_to_create_queue>` and insert them into the mutable binomial queue. It then reads the integers from file `<input_file_to_check_delete>` and checks the delete functionality.
By looking into function `TestInsertAndDelete()` you can see what is expected to be produced. We will give you input as well as expected output files.

Part 3 – improved insert (20 points)

Write a faster insert function for the binomial queue. You will need to specialize the merge functionality when one item is inserted.
So add the following function in the `BinomialQueue` class:

```
bool insertNoMerge(const Comparable & x);
```

In order to achieve that you have to add a modified merge function and make it specific to the merging a binomial queue with a queue containing one element only.

Test it using the given code in file `test_new_insert.cc`

You should keep the `main()` and `TestNewInsertAndSearch()` as is. You can update these functions if you want to debug, but you should keep them as is for the submission. We will compile your code with this file on our side. Note that if you use the regular insert you will receive no credit.

To run the program run:

```
 ./test_insert_and_search <input_file_to_create_queue>
<input_file_to_check_search>
```

The output should be exactly the same as in Part 1.

Source files provided:
`Makefile, test_insert_and_search.cc, test_insert_and_delete.cc, test_new_insert.cc, binomial_queue.h, quadratic_probing.h, dsexceptions.h`

Test files provided:
`<x>_numbers.txt, <x>_test_search.txt, <x>_test_delete.txt, <x>_search_out.txt, <x>_delete_out.txt`
Where `<x>=20, 100, 1000`

For example you can run (the input file contains 100 numbers):
`./test_insert_and_search 100_numbers.txt 100_test_search.txt > out1.txt`
The output `out1.txt` should be the same as `100_search_out.txt`
`./test_insert_and_delete 100_numbers.txt 100_test_delete.txt > out2.txt`
The output `out2.txt` should be the same as `100_delete_out.txt`
`./test_new_insert 100_numbers.txt 100_test_search.txt > out3.txt`
The output `out3.txt` should be the same as `100_search_out.txt`

Deliverables:

`README`
`binomial_queue.h`
`quadratic_probing.h`

Do not submit files `test_insert_and_search.cc`, `test_insert_and_delete.cc`, `test_new_insert.cc`. You should use them to make sure you are getting the expected output. Note that you should follow all coding instructions and functions as described, otherwise your submitted code will not compile on Gradescope.
