CMPSC 130A: Spring 2017

Programming Assignment 1

Assigned: April 17 **Due: Midnight, April 27 (11:59 PM)**

This assignment asks you to implement a data structure combining the functionality of Hash Tables and Heaps. Hash tables can implement insert, delete, and lookup in expected $O(1)$ time, while a Heap can implement insert and deleteMax in $O(\log n)$ worst-case time. However, deleting an arbitrary key in the Heap requires worst-case linear time, while Hash table will take linear time for a deleteMax. Your assignment is to build a data structure that interlinks the hash table and the Heap representations so that all four operations (insert, delete, lookup, and deleteMax) can be performed in time $O(1)$ expected plus $O(\log n)$, that is, just the sum of a hash table and a heap operation. You will also implement a *print* function that prints out all the items of the set in sorted (decreasing) order. Some specific implementation details:

1. Assume all elements to be inserted are integers (but they can be negative).
2. For the Hash Table, use **separate chaining** for collision resolution.
3. For the Max-Heap, use the array implementation.
4. Your executable should be called **prog1**.
5. You should implement all functions of this assignment yourself, and not use STL.

Input Format and Test Data

1. For the basic assignment, assume that size of the Hash Table is fixed as 97. (Note that 97 is a prime, so you can use the hash function "x mod 97".)
2. Your program should accept input from stdin, where the first line contains a single number 97 to tell you the size of the hash table.
3. The rest of the input file contains one command per line, where each command has one of the following formats for some integer value i.
   - insert i
   - lookup i
   - deleteMax
   - delete i
   - print (This will be the **last** command of the input file.)
4. "insert i" adds i to the set *if i is not present already*, and otherwise return an error.

5. “lookup i” returns “found i” if i is in the set, and “i not found” otherwise.
6. “deleteMax” removes the lasrgest item from the data structure and returns its value.
7. “delete i” removes i from the set, or returns an error that *the item is not present.*
8. Finally, on command “print” your program should print out the entire set in the **sorted** (decreasing) order.

Extra Credit

If you want to challenge yourself, you could enhance your program in a number of possible ways.

- First, design the data structure so that it handles an arbitrary size set. In this case, the input file can be any number n, which is an upper bound on the size of the set, and you should use a prime number p larger than n for the hash table. By a classical result in number theory, there is always a prime between n and $2n$, and there are lists of prime number maintained at various websites.
- In order to print the set in decreasing, if you perform repeated deleteMax operations, the operation destroys the data structure. Try to implement so that you can print without destroying the data structure.

The Extra Credit will NOT compensate for any points lost in the basic assignment. So, please make sure that basic functions all work correctly before spending time on extensions. The Extra Credit will also NOT give you points in excess of 100. However, it will be used in the final grade calculation, for instance to decide if you are on the grade boundary, and a few points of extra credit puts you over the limit.

Testing

Your programs will be autograded, so make sure that your output adheres to the specified format. We should should be able to run your program as `./prog1 < input.txt`.

The input file will have the following structure:

- The first line will be an integer N, the size of the hashtable. Note that N will be set to 97 for the basic version.
- The next line will be an integer S, the number of operations that will follow.
- The next S lines will be one of the operations mentioned before. For the basic version, `print` will be the last statement.

Following is an example input file `sample.txt`.

```
97
10
insert 5
insert 2
insert 8
insert 3
delete 7
insert 5
delete 5
deleteMax
lookup 8
print
```

The expected output is:

```
user@csil~] ./prog1 < sample.txt
error : item not present
error : item already exists
8
8 not found
3 2
```

Grading

Your grade will comprise of following components:

- There will be four testcases, one for each of the functions: `insert, delete, lookup, deleteMax`. The last statement in each of these testcases will be `print`. The autograder will measure the time taken to execute the tests. This accounts for a total of 60 points subject to the output being correct and **all** the operations meeting the expected $O(\log n)$ bound.
- There will be another testcase where the four operations above will be ordered randomly. The last statement will again be `print`. This accounts for a total of 40 points subject to the output being correct and meeting the expected time bounds.

Submission Instructions

1. Make sure to include a **README** file. This should *must* contain the following information in the given format.

   ```
   CSIL login :  [Your username here], UCSB Email :  [Your ID]@umail.ucsb.edu
   ```

   This is needed to map your CSIL usernames to your records.

2. If you have attempted the extra credits, make sure to have the text: `‘Attempted extra credits’` in your **README** file.
3. The recommended programming language is `C++`, however you may write your program in other languages. Always make sure to include a `‘makefile’`, so that running `‘make’` compiles your source files and generates the executable `‘prog1’`.
4. In order to electronically turnin your project files, use the following command:

   ```
   turnin prog1@cs130a LIST
   ```

   where LIST is the list of all your files, i.e., your makefile, all your source and header files. (The turnin program is located in `/usr/bin/turnin` on `csil.cs.ucsb.edu` only.) If you turnin more than once then the version that will be graded is the last version you turned in before the deadline.