CS 130A Data Struc & Alg 1

Hashing is very useful if we are only interested in `insert`, `delete`, and `find` operations; e.g., `delete (30)` or `insert (15)` or `find (-10)`. But a hash table provides no assistance for searches that depend on the order or rank of an element in a set, e.g., find the smallest element, delete the largest element, etc. For the latter type of operation, heaps are more appropriate. On the other hand heaps do not support a general delete operation, e.g., `delete (-10)`; they only support deleting the min element.

In this programming assignment, you will develop a "compound" data structure called *Quash*, which is composed of both a min-heap (priority queue) and a hash table. The Quash supports inserts, delete and lookup using its hash component, and `deleteMin` using its heap component. In particular, each element in the set will be inserted in **both** the heap and the hash table. You will need to include pointers in both directions between the 2 instances of the element in the heap and in the hash table.

A **count** should be stored along with each element, in order to allow insertion of more than one copy of the same element. When an element is inserted for the first time, its `count` is `1` and every subsequent insertion increments the `count`. On deletion, the `count` is decremented. When the `count` becomes zero due to a deletion, then the element is completely deleted.

Operations should be executed as follows:

- `insert(i)`: Insert element `i` in both the heap and the hash table, and link them correctly. Your program should return either "item successfully inserted, count = 1" or "item already present, count = n", where n is the count for that item

- `lookup (i)`: Use the hash table to determine if `i` is in the data structure. Your program should return either "item found, count = n", where n is the count for that item or "item not found."

- `deleteMin`: Use the heap to find the minimum element and decrement its count. If its count becomes zero, delete it from the heap and use the pointer to delete the element in the hash table. Your program should return the key value of the item deleted.

- `delete(i)`: Use the hash table to determine where `i` is and decrement its count. If the count becomes zero, delete it from the hash table and use the pointer to delete it from the heap. (Note you will need to generalize the heap operations to support this "internal" delete operation.) You program should return either "item count decremented, new count = n", "item successfully deleted" or "item not present in the table."

- `print()`: Print out the heap (in the array format).

Some specific implementation issues:

- Assume all elements to be inserted are integers (but they can be negative).

- Use the array implementation for the min-heap.

- For this assignment, you can use "mod 43" as the hash function. (In general, it would be better to use an internal resize operation, which doubles the table size when the table becomes too full, and shrinks it when it becomes too empty, using appropriate thresholds for the full and empty. You are not required to implement resize, but you are welcome to try it if you like.)
- For collision resolution, use **separate chaining**.

For consistency, you can assume that the sequence of operations is provided in ascii format, with each command on a separate line. In addition, your program should print out (one line per command) information after each operation, as follows:

- `insert(i)`: "item successfully inserted, count = 1" or "item already present, new count = n"
- `lookup(i)`: "item found, count = n" or "item not found"
- `deleteMin`: "min item = x, count decremented, new count = n" or "min item x successfully deleted" or "min item not present since table is empty", where x is the value of the min element
- `delete(i)`: 'item count decremented, new count = n" or "item successfully deleted" or "item not present in the table"
- `print()`: prints out the heap array with elements separated by a whitespace.

Example

Sample Input

```
insert 10
insert -50
insert 76
lookup 12
insert 12
lookup 12
insert 12
lookup 12
print
deleteMin
delete -50
print
delete 10
print
deleteMin
print
delete 12
deleteMin
print
deleteMin
```

Sample Output

```
item successfully inserted, count = 1
item successfully inserted, count = 1
item successfully inserted, count = 1
item not found
item successfully inserted, count = 1
item found, count = 1
item successfully inserted, count = 2
item found, count = 2
-50 10 76 12
min item -50 successfully deleted
item not present in the table
10 12 76
item successfully deleted
12 76
min item = 12, count decremented, new count = 1
12 76
item successfully deleted
min item 76 successfully deleted

min item not present since table is empty
```

The TA/Readers should be able to run your program as "prog1" with input from stdin. Be sure to include a Makefile, and name the executable prog1. The TA/Readers should be able to terminate your program by Ctrl-D (EOF for linux). When we test using files, the EOF will be there.

The programs are tested automatically so make sure that you provide the output exactly as specified and shown above with no extra space lines to the output. The program should output to stdout. The input provided above is the subset of the input your program would be tested against. So be sure to check your program exhaustively for different cases.

This programming assignment has to be implemented in C++.

Submission

Deliver the assignment using `turnin` before midnight on the due date. You need to submit all the files having your implementation and a makefile using `turnin` command. The command to be used is:

```
turnin PA1@cs130a [list of files to be submitted]
```

Note that when running `turnin`, just provide a list of files. Do not create a directory and place all the files in the directory.