Project 3, Min-Max Heaps

Due: Please check due date on BlackBoard

Objectives

The objective of this programming assignment is to have you work with heaps and to practice working with templated classes in C++.

Introduction

In this project, you will implement the min-max heap data structure described [here](http://faculty.cse.tamu.edu/slupoli/notes/DataStructures/homeworks/Fall2013HWs-Java/Homework%204,%20CMSC%20341%20Fall%202013.pdf). Four of these min-max heaps can be used to create a quartile heap — a data structure that can be used to collect percentile statistics.

A quartile heap supports insertion in *O*( log *n* ) time and can report the minimum, the 25th percentile, the median (or equivalently, the 50th percentile), the 75th percentile and the maximum value in the heap in constant time. Quartile heaps do not support deletion.

You can construct a quartile heap using 4 min-max heaps. The first min-max heap stores all the values in the lowest 25th percentile. The second min-max heap stores the values in the 25th to 50th percentile. The third min-max heap has the 50th to 75th percentile and the fourth min-max heap has the values above the 75th percentile. The intention is for these four heaps to hold the same number of items (off by one, at most). Furthermore, the values in the first heap should be smaller than or equal to the values in the second heap, which are in turn smaller than or equal to the values in the third heap, which are smaller than or equal to the values in the fourth heap.

In this arrangement, the minimum value in the quartile heap is simply the minimum value in the first heap. The 25th percentile value is the maximum value in the first heap. The median is the largest value in the second heap, and so on. Since we are using min-max heaps, these minimum and maximum values can be retrieved from the appropriate min-max heap in constant time.

Assignment

*Note: Running time is one of the most important considerations in the implementation of a data structure. Programs that produce the desired output but exceed the required running times are considered wrong implementations and will receive substantial deductions during grading.*

Your assignment is to implement a min-max heap data structure and to use these in turn to construct a quartile heap. Your implementation must use templateds.

Your min-max heap class should begin with the declaration:

|  |
| --- |
| template <typename T>  class MMHeap  {  T\*\* pData;  size\_t length;  ... |

MMHeap Requirements

Here are the requirements for your implementation of the MMHeap class.

1. You must have a constructor(s) with the following signature:

|  |
| --- |
| **MMHeap**() :pData(**nullptr**), length(0) {};  **MMHeap**(**const** **int**& initialSize) :length(0) { ... } |

This should be the only constructor for the class. The constructor should set aside an array big enough for initialSize items.

1. You must have a size() method that reports the number of items in the min-max heap:

|  |
| --- |
| size\_t **size**() **const** { **return** length; } |

The size() method should run in constant time.

1. You must have an insert() method:

|  |
| --- |
| **void** **insert**(**T**\* px) { ... } |

The insert() method should run in *O*( log *n* ) time. Insertion of duplicate values is allowed. Duplicate entries should appear multiple times in the MMHeap.

1. You must have a dump() method that prints out the contents of the min-max heap (along with other debugging information):

void dump();

1. You must have two methods, getMin() and getMax(), that return the smallest and the largest values currently in the min-max heap.

T\* getMin();

T\* getMax();

These two methods should run in constant time. If the heap is empty, these methods should throw ***an exception***.

1. You must have two methods, deleteMin() and deleteMax(), that remove and return the smallest and largest items, respectively:

T deleteMin();

T deleteMax();

These two methods should run in *O*( log *n* ) time.

QHeap Requirements

Here are the requirements for your quartile heap class.

1. The class must be declared as:

|  |
| --- |
| **template** <**typename** T>  **class** QHeap  {  MMHeap<T>\*\* heaps; |

1. You must implement a constructor(s) with the signature:

|  |
| --- |
| **public**:  **QHeap**() :heaps(**nullptr**) {};  **QHeap**(**const** **int**& initialSize) { ... } |

1. You must implement a method insert() that adds an item to the quartile heap:

void insert(T\* px);

The insert() method should run in *O*( log *n* ) time. Insertion of duplicate values is allowed. Duplicate entries should appear multiple times in the QHeap.

1. You must implement the statistics reporting methods:

T\* getMin();

T\* getMax();

T\* get25();

T\* get50();

T\* get75();

All of these methods must run in constant time. If there are fewer than 4 items in the quartile heap, then calls to these methods may result in an exception.

1. The sizes of the 4 min-max heaps must be within 1 of each other. Furthermore, if there is a size difference, the bigger heaps must be the lower percentile heaps. So, this is OK:

Heap 1 ( 0% - 25%) 15 items  
Heap 2 (25% - 50%) 15 items  
Heap 3 (50% - 75%) 14 items  
Heap 4 (75% - 100%) 14 items

But, this is not:

Heap 1 ( 0% - 25%) 15 items  
Heap 2 (25% - 50%) 14 items  
Heap 3 (50% - 75%) 15 items  
Heap 4 (75% - 100%) 14 items

*Note:* this does not mean you insert a new item in a lower percentile heap first, since the placement of an item in a heap depends on its value. For example, if you have to insert a value that is the largest value you have seen so far, then that value must go in the fourth heap. Now your fourth heap might be bigger than the third heap. You have to remove the smallest item in the fourth heap and insert that into the third heap. This might in turn make the third heap too big ...

1. You must implement a dump() method that prints out the statistics of the quartile heap and also prints out the contents of the 4 min-max heaps.

Here is a small sample program (Proj3Test.cpp and output Proj3MMHeapTest.txt) offered that should compile with your program.

*Note:* These are simple tests and do not check all the conditions necessary for deleteMin() and deleteMax(). You must *at least* also check all the cases from [this](http://faculty.cse.tamu.edu/slupoli/notes/DataStructures/homeworks/Fall2013HWs-Java/Homework%204,%20CMSC%20341%20Fall%202013.pdf).

Here's another test program for you to try out: Proj3CensusTest.cpp with [CensusCity.h](https://www.csee.umbc.edu/courses/undergraduate/341/fall13/projects/CensusCity.java), too. This test case uses census data from the United States Census Bureau. It also demonstrates that you can have objects of the same class compared in different ways. In this case the boolean data member use2011 determines whether the compare() method will use the 2010 census data or the 2011 population estimates.

Implementation Notes

*Here we list some recommendations and point out some traps and pitfalls.*

* You need some way to keep track of whether the last level of your min-max heap is an odd level or an even level. Think through this. If you mess this part up, then your min-max heap will be messed up.
* In deleteMin(), when the new item trickles down and hits bottom, there are *two* reasons that may force it to percolate back up the max heap levels. Make sure that you understand these two conditions. The situation is analogous for deleteMax().
* During a deleteMin(), there are 4 cases where you should stop the trickling down process:
  1. You should stop when the key is actually in the right place. (It is smaller than the min items in the left and right subtrees.)
  2. You should stop when the node has no children. (You actually hit bottom.)
  3. You should stop when the node has 1 or 2 children, but no grandchildren.
  4. You should stop when the node has 2 children, but only 1 or 2 grandchildren.

Note that it is impossible for a heap to have no right child, but the left child has children. Also, if you have 3 or 4 grandchildren, the only reason you would have stopped trickling down is that the key is already in the right place (Case 1).

You need to handle the four cases separately, because checking if you need to bounce up the maxheap levels has to be handled differently in each case. Also, Case 4 has subcases depends on whether the smallest item is the right child or one of the grandchildren. (Here's an [illustration of Case 4 on a separate page](http://faculty.cse.tamu.edu/slupoli/notes/DataStructures/projects/F19Proj3MinMaxHeap/Project%203,%204th%20Case.pdf).)

Of course, the situation is analogous for deleteMax().

* There are special cases for deleteMax() when the min-max heap has only one or two items.
* You are probably better off making 4 separate methods that trickle down the min heap levels, trickle down the max heap levels, percolate up the min heap levels and percolate up the max heap levels.
* In QHeap, C++ is perfectly happy to make a templated MMHeap object:

MMHeap<T> \*\*heaps = new MMHeap<T> \* [n]; // you need another step AFTER this to fully instantiate

* Let's just *define* the 25th percentile to be the maximum item in the first min-max heap, the 50th percentile to be the maximum item in the second min-max heap and the 75th percentile to be the maximum of the third min-max heap. Don't take the average of the maximum from one heap and the minimum from the next heap.

What to Submit

You should copy over all of your C++ source code and have your .cpp files in their own directories which are in turn under the src directory. You must also supply an ANT build file.

Make sure that your code is in the ~/csce221proj/proj3/ directory and not in a subdirectory of

~/csce221proj/proj3/. In particular, the following Unix commands should work. (Check this.)

cd ~/csce221proj/proj3

make

make run

make clean