PQ Yeah Hours

Alexander De Baets

If you are looking at these online, please watch the video! This week's YEAH hours had a lot of drawing on the board!

Helpful Resources

- Pointer & Linked Nodes Lecture
- Linked Lists Lecure
- Priority Queue and Heap Data Structure Lecture

What is a Priority Queue?

Deliverables

- SLinkedPriorityQueue.h/.cpp:
- USLinkedPriorityQueue.h/.cpp
- HeapPriorityQueue.h/.cpp
- analysis.pdf
- pqueue-main.cpp:

What methods do I need to implement?

Member	Description
`pq.enqueue(value, priority);`	In this function you should add the given string value into your priority queue with the given priority. Duplicates are allowed. Any string is a legal value, and any integer is a legal priority; there are no invalid values that can be passed.
`pq.dequeue()`	In this function you should remove the element with the most urgent priority from your priority queue, and you should also return it. You should throw a string exception if the queue does not contain any elements.
`pq.peek()`	In this function you should return the string element with the most urgent priority from your priority queue, without removing it or altering the state of the queue. You should throw a string exception if the queue does not contain any elements.
`pq.peekPriority()`	In this function you should return the integer priority that is most urgent from your priority queue (the priority associated with the string that would be returned by a call to peek), without removing it or altering the state of the queue. You should throw a string exception if the queue does not contain any elements.

***pq*.changePriority(*value*, *newPriority*);**	In this function you will modify the priority of a given existing value in the queue. The intent is to change the value's priority to be more urgent (smaller integer) than its current value. If the given value is present in the queue and already has a more urgent priority to the given new priority, or if the given value is *not* already in the queue, your function should throw a string exception. If the given value occurs multiple times in the priority queue, you should alter the priority of the first occurrence you find when searching your internal data from the start.
***pq*.isEmpty()**	In this function you should return true if your priority queue does not contain any elements and false if it does contain at least one element.
***pq*.size()**	In this function you should return the number of elements in your priority queue.
***pq*.clear();**	In this function you should remove all elements from the priority queue.
out* << *pq	You should write a << operator for printing your priority queue to the console. The elements can print out in any order and must be in the form of "***value***":***priority*** with {} braces, such as {"t":2, "b":4, "m":5, "q":5, "x":5, "a":8} . The PQEntry and ListNode structures both have << operators that may be useful. Your formatting and spacing should match *exactly*. Do not place a \n or endl at the end.

The Sorted Linked List

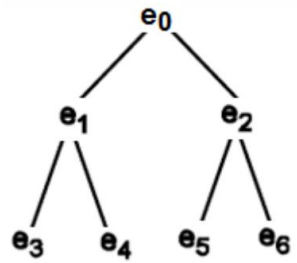
- Only one private member variable: a pointer to the front of the list!
- You are NOT allowed to have a variable that contains the number of elements in the queue.
- ~On board drawings & explanations~

The Unsorted Linked List

- Only one private member variable: a pointer to the front of the list!
- You are NOT allowed to have a variable that contains the number of elements in the queue.
- ~On board drawings & explanations~

The Heap PQ

Fact summary: Binary heap in an array

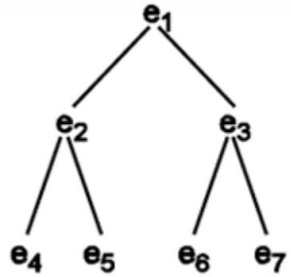


0-based:

For tree of height h, array length is 2^h-1

For a node in array index i:

- Parent is at array index: (i – 1)/2
- Left child is at array index: 2i + 1
- Right child is at array index: 2i + 2



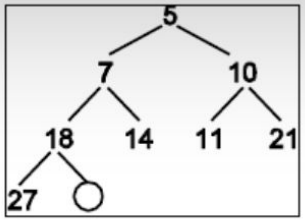
1-based:

For tree of height h, array length is 2^h

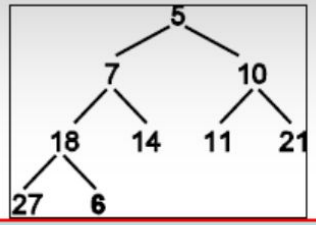
For a node in array index i:

- Parent is at array index: i /2
- Left child is at array index: 2i
- Right child is at array index: 2i + 1

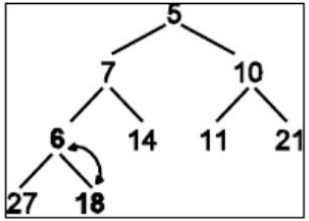
[Binary heap insert reference page]



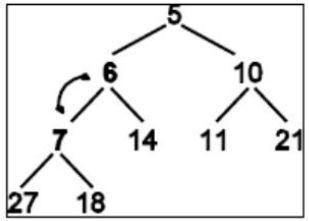
(a) A minheap prior to adding an element. The circle is where the new element will be put initially.



(b) Add the element, 6, as the new rightmost leaf. This maintains a complete binary tree, but may violate the minheap ordering property.



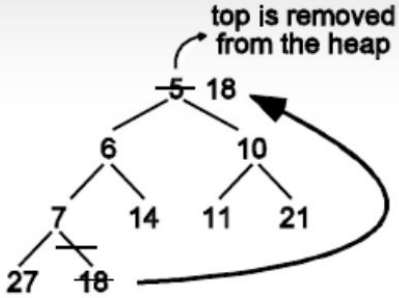
(c) "Bubble up" the new element. Starting with the new element, if the child is less than the parent, swap them. This moves the new element up the tree.



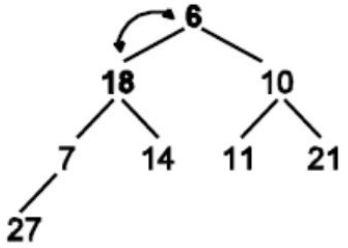
(d) Repeat the step described in (c) until the parent of the new element is less than or equal to the new element. The minheap invariants have been restored.

[Binary heap delete + “trickle-down” reference page]

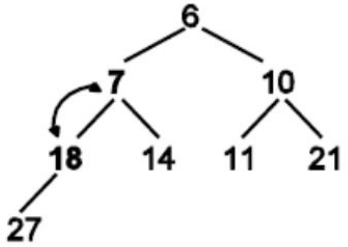
(a) Moving the rightmost leaf to the top of the heap to fill the gap created when the top element (5) was removed. This is a complete binary tree, but the minheap ordering property has been violated.



(b) “Trickle down” the element. Swapping top with the smaller of its two children leaves top’s right subtree a valid heap. The subtree rooted at 18 still needs fixing.



(c) Last swap. The heap is fixed when 18 is less than or equal to both of its children. The minheap invariants have been restored



Analysis

- Test Enqueue and Dequeue with 5 different N values to see how they react.
- How do you choose your N values?
- Check out our TImer-class

http://stanford.edu/~stepp/cppdoc/Timer-class.html

Good Luck!