lab_heaps

Precarious Priority Queues

Due: Apr 07, 23:59 PM

Doxygen

Assignment Description

In this lab you will write some heap functions to implement a min heap from scratch.

The version of heap you will implement will have a tree implemented in an array. Your array can be 0-index or 1-indexed, where 0 and 1 are the indices of the root, depending on which one you choose.

The left and right children of the root will be at the positions immediately after the root. For example, if your root is at index 0, its left child will be at index 1 and 2. The root's left children will be at positions 3 and 4, and the root's right children will be at positions 5 and 6, and so on and so forth. You will need to determine the indices to store these children in leftChild() and rightChild(), and the indices of each element's parent in parent().

Remember that the big idea about a heap is that it must keep its heap property. In a min heap, this means that each element is smaller than both of its children. A max heap is the opposite- each element is larger than both of its children. Your maxPriorityChild() function will return the min child (out of an element's left and right children) in a min heap and the max child in a max heap. This is not a recursive function- it simply utilizes the higherPriority functor and returns whichever child out of the two is less than or greater than the other, depending on what the priority for this heap is. Your heapifyUp() and heapifyDown() functions will ensure the heap property of your heap, by swapping elements recursively up or down the tree to maintain the heap property.

Lab Insight

Heaps are an incredibly important data structure for prioritization functionality. Its performance in prioritization helps us implement path-finding algorithms efficiently. This same data structure is what helps power your navigation systems when you are trying to get from one place to another. To learn more about the practical importance of heaps, CS 374 and CS 473 delves into the applications of them in algorithmic contexts.

Checking Out The Code

From your CS 225 git directory, run the following on EWS:

```
git fetch release
git merge release/lab_heaps -m "Merging initial lab_heaps files"
```

If you're on your own machine, you may need to run:

```
git fetch release
git merge --allow-unrelated-histories release/lab_heaps -m "Merging initial lab_heaps
files"
```

Upon a successful merge, your lab_heaps files are now in your `lab_heaps` directory.

The code for this activity resides in the `lab_heaps/` directory. Get there by typing this in your working directory:

```
cd lab_heaps/
```

Check out the Doxygen for provided files.

root()

Doxygen for `root()`. Returns the index of the root- 0 for a 0-indexed heap, 1 for 1-indexed.

leftChild()

Doxygen for `leftChild()`. Returns the index of the left child of the element.

rightChild()

Doxygen for `rightChild()`. Returns the index of the right child of the element.

parent()

Doxygen for `parent()`. Returns the index of the parent of the element.

empty()

Doxygen for `empty()`. Determines if your heap is empty.

hasAChild()

Doxygen for `hasAChild()`. Determines if the current element has a child (aka if it isn't a "leaf node").

maxPriorityChild()

Doxygen for `maxPriorityChild()`. Determines the child with the maximum priority.

heapifyDown()

Doxygen for `heapifyDown()`. Maintains heap property by "sinking" a node down. When we pop an element, we heapifyDown() the new root so the heap's property is maintained.

heapifyUp()

Maintains heap property by "bubbling" a node up. When a node is added to the end of the array, we call heapifyUp() to recursively swap it into its proper place. We have written this for you as an example: Doxygen for `heapifyUp()`.

heap()

Doxygen for Constructor for building an empty heap. Doxygen for Constructor for building a heap.

pop()

Doxygen for `pop()`. Pops the element with the highest priority, as defined by the higherPriority functor. Maintains heap property by calling heapifyDown().

peek()

Doxygen for `peek()`. Returns the element with the highest priority.

`push()`

Doxygen for `push()`. Pushes an element into the heap, then calls heapifyUp() to maintain heap property.

`updateElem()`

Doxygen for `updateElem()`. Updates element in the heap array at the provided index (0-based, you may need to account for this based on your `root()` function.) while maintaining the heap property by appropriately "bubbling" up or down as need be. The index is relative to the first element in the heap, NOT necessarily the first element in the vector/array you are using for implementing the heap.

Testing Your Code

You can test your implementation by running:

```
./testheap            // Runs testheap. You can diff the output with soln_testheap.out
./testheap color      // colorizes the output of testheap (green for correct output
                      //    red for incorrect output, or red underlines for missing output)
./testheap > out.txt  // Redirects the output to the file "out.txt" so you can diff it with
soln_testheap.out
./testheap 10000      // Tests your heap with 10000 items instead of the default 15

make test             // Tests your heapify and constructor
```

Committing Your Code

Commit your code the usual way:

Guide: How to submit CS 225 work using git

Grading Information

The following files (and ONLY those files!!) are used for grading this lab:

- `heap.cpp`
- `heap.h`

If you modify any other files, they will not be grabbed for grading! And you will be sad, and we will be apathetic!