Computer Science 2530
April 20, 2020

Happy Monday, April 20.

This will be the last lecture that I send you. We will omit sorting. Today, we look at our last topic, heaps.

Exam 5 is Friday, April 24. I will make the exam available at 1:00 and expect you to email your answers to me by 3:30. Please send your answers as attachments and name *every file* that you attach starting with your last name, followed by your first name.

You can use the notes and the lectures during the exam. I am trusting your integrity and ethics for you to work independently. See the table of contents of the online notes for the topics that will be covered.

Heaps

As described on page **44A** in the notes, a ***heap*** is a binary tree. (This use of the word "heap" is unrelated to the heap as an area of memory. Both are standard terms.)

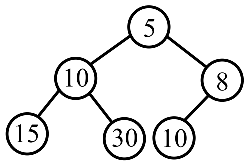
There are two kinds of heaps, min-heaps and max-heaps. In this page, I will only discuss min-heaps.

For assignment 6, you implemented priority queues using linked lists. Using that implementation, inserting an item into a priority queue of size n takes $\Theta(n)$ time. That can be very slow if n is large.

A min-heap is alternative representation of a priority queue. We will see that, using a min-heap, you can implement a priority queue in an efficient way, where insertion and removal both take time $\Theta(\log_2(n))$, where n is the current number of items in the priority queue. Inserting an item into a linked list of size 1000 takes 1000 steps. Inserting an item into a heap of size 1000 only takes about 10 steps.

The ordering requirement for min-heaps

Heaps are ordered vertically. If node u is the parent of node v in a min-heap, then the number in node u is no larger than the number in v. As you move downward in a min-heap, the numbers can only increase. Here is an example of a min-heap. Only the priorities are shown. Imagine a node to be a stack of two checkers, with the priority on top and the item beneath it. Whenever a priority is moved, its item is moved with it.



The structural requirement for heaps

To build up a heap, you are required to add nodes by levels, from top to bottom. The nodes in each level must be added from left to right. There cannot be any gaps in a level. In the example above, the first two levels have been completed. The bottom level has only 3 nodes. Notice that they are as far to the left as possible. If another node is added to that heap, it will be the fourth node on the bottom level. The next node after that will start a new level, and will be the leftmost possible node in that level.

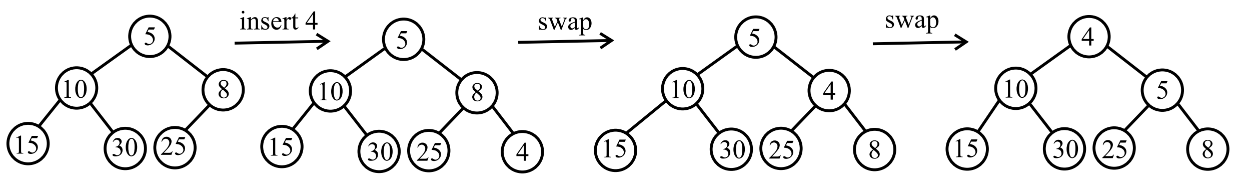
Inserting a value into a heap

Page **44C** shows how to insert an item and priority into a min-heap. There are two main steps of the insert operation.

1. Insert the new item and priority in the bottom level as far to the left as it can go. That might entail starting a new level. Don't worry that the modified tree does not satisfy the ordering requirement. That is handled at the next step.

2. Now do a ***reheap-up*** operation, starting at the newly inserted node. Don't move nodes. Move values among the nodes. If the priority in a node u is out of order with respect to u's parent, swap the item and priority in node u with the item and priority in u's parent. Keep moving the inserted item/priority that was inserted upwards until its parent has a smaller priority or the new value reaches the root of the tree.

Here is an example. Starting with the heap on the left, we do the first step of inserting priority 4 (and some associated item that is hidden). Then we do a sequence of swaps, moving the 4 up the tree.



Read page **44C**, but skip the implementation.

Removing from a heap

Recall that a priority queue only allows you to remove the item with the smallest priority. That item is surely located in the root of a min-heap.

To accomplish a removal, do the following steps.

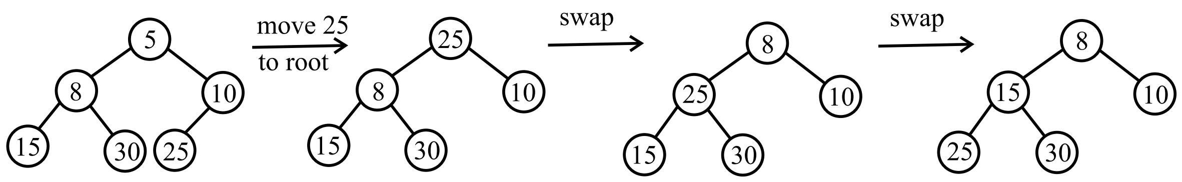
1. Remember the item and priority that are stored in the root. Those will be the item and priority that have been removed.

2. Find the rightmost node u in the bottom level. Move the item and priority from that node into the root. Then delete node u from the tree.

3. Now the ordering requirement will be violated. Do a ***reheap-down*** operation to move the value that is in the root downwards as far as it needs to go in the tree.

   Only move a priority in node u downward if it is larger than at least one of the priority in u's children. If a priority needs to be moved, swap it with the smaller of the values in the children. Then continue moving it downwards, as needed.

Here is an example. The goal is to remove the smallest priority, 5, from the heap at the left. The first step moves 25 to the root and removes the node that previously contained 25. Then the swapping steps of reheap-down are done.



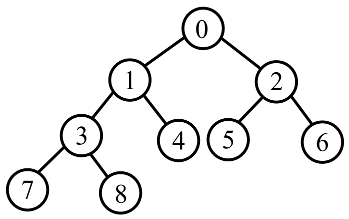
Notice that 25 moves downward in the tree until, in this case, it reaches a leaf, and cannot move downward any further.

Read page **44D** in the notes. Skip the implementation of remove.

Representing heaps

A heap represents a priority queue. But how can we represent a heap?

Using pointers, as we have done for binary search trees, does not work well for heaps. Instead, a heap is represented as an array and the logical size of that array. The following shows the indices where nodes are stored. For example, the root contains 0, so the root is stored in the array at index 0. The left child of the root is stored at index 1 in the array.



Using an array makes it easy to similate operations on the heap. Page **44B** defines functions

leftchild(i)	the index of the left child of the node with index i
rightchild(i)	the index of the right child of the node with index i
parent(i)	the index of the parent of the node with index i

What you should be able to do

You should be able to simulate insertion and removal operations. Exercises on page **44C** involve insertion. Exercises on page **44D** involve removal.