Su09_CS61b_Week6Day2

42 Priority Queues and Binary Heaps	*(4 activities)*

42.1 Think about implementations of queues and priority queues.(7 steps)

42.1.1 (Display page) Overall road map

Overall road map

This set of activities involves exploration of priority queues (queues in which each element also has a priority that might bring it to the head of the queue more quickly) and their implementation. A data structure called a *max heap* turns out to be particularly suitable for implementing a priority queue. Activities include analysis of the abstract definition of a max heap, as well as debugging code that attempts to implement the abstract definition. Finally, you'll consider an application of a heap. Priority queues are an integral component of many algorithms for graph processing (which we'll cover next week). For example, the first four weeks of CS 170 all involve graph algorithms that use priority queues.

42.1.2 (Display page) Queue operations

Queue operations

A *queue* (pronounced "cue") is a data type that's like a waiting line. The queue has a "front", the head of the line, and a "back". Its operations include the following:

- construct an empty queue;
- (*enqueue*, pronounced "en-cue") add an element to the *back* of the queue;
- (*dequeue*, pronounced "de-cue") remove the front element of the queue;
- see if the queue is empty;
- access the front of the queue.

We've used a queue already to enumerate elements of a tree in breadth-first order. Queues also appear in any application that uses a waiting line, e.g. processes waiting their turn to run on the computer, simulations of service organizations, etc.

42.1.3 (Brainstorm) Queue implementations

What data structure makes the operations of enqueueing, dequeueing, checking for empty, and accessing the front of the queue most efficient? Briefly explain your answer. Correct your answer if necessary after reviewing the responses of your labmates.

42.1.4 (Display page) Priority queues

Priority queues

A *priority queue* is a queue in which every member has an associated *priority* (basically, just a number). Operations are

- construct an empty priority queue;
- add an element to the queue;
- remove the element with the highest priority (or one of the elements with the highest priority if there are more than one);
- see if the queue is empty;
- access a queue element whose priority is highest.

42.1.5 (Brainstorm) Linked list implementation of a priority queue

Explain how to use a linked list to implement a priority queue, and give running time estimates for the enqueue and dequeue operations. Correct your answer if necessary after reviewing the responses of your labmates.

42.1.6 (Brainstorm) BST implementation of a priority queue

Explain how to use a binary search tree to implement a priority queue, and give execution-time estimates for the enqueue and dequeue operations. Correct your answer if necessary after reviewing the responses of your labmates.

42.1.7 (Brainstorm) More or less work to balance the tree?

Your answer to the previous question probably assumed that the binary search tree was balanced. In the implementation you described, would it be less difficult or more difficult to keep the tree balanced compared to a situation where elements were being randomly added to and deleted from a binary search tree? Briefly explain your answer. Correct your answer if necessary after reviewing the responses of your labmates.

42.2 Work with binary heaps.*(10 steps)*

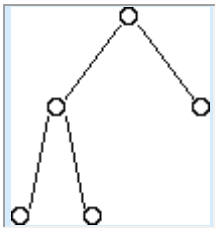
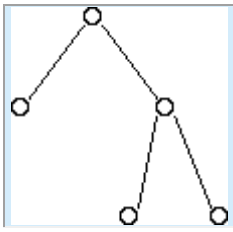
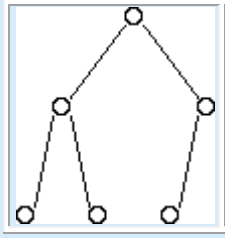
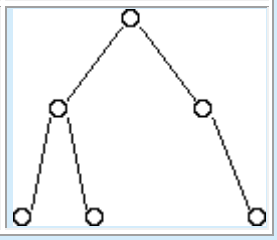
42.2.1 (Display page) Binary max heaps

Binary max heaps

An even better choice for implementing a priority queue is a special kind of binary tree, named a *binary max heap*. A binary max heap has the following characteristics.

- The element at the root—the "top of the heap"—is the largest element, or one of the largest elements, if there is a tie. (A "min heap" has the smallest item at the top.)
- All the subtrees of a max heap are themselves max heaps. (Recall a similar recursive definition for binary search trees.)
- Finally, a max heap is a *complete* tree, that is, completely filled except maybe for the bottom level. The bottom level is filled from left to right, with no missing nodes. ("Complete" is used here, as in section 8.5 of *Objects, Abstraction, Data Structures, and Design*, to mean what we called "maximally balanced" in an earlier homework.)

The table below lists some complete and incomplete trees.

complete trees	*incomplete trees*
	
	

In a complete tree of N nodes, the height is at most $1 + \log_2 N$. This is because a complete tree of height H has between 2^{H-1} and $2^H - 1$ nodes. That means that we can expect logarithmic worst-case behavior if we restrict changes in the structure to one path from the root to a leaf.

42.2.2 (Brainstorm) Heaps and BSTs

Describe as many different nonempty binary trees as you can that are both max heaps and binary search trees. ("Different" means different in shape as opposed to key values.) Assume that the largest value is at the top of the heap and at the rightmost position in the binary search tree.

42.2.3 (Brainstorm) Trying to verify that a binary tree is complete

You may have discovered in the day14 homework that it's not obvious how to verify that a linked binary tree is complete (what we called "maximally balanced" in that homework). A CS 61B student suggests the following recursive algorithm:

- A one-node tree is complete.
- A tree with two or more nodes is complete if its left subtree is complete and has depth k for some k, and its right subtree is complete and has depth k or k-1.

Find a counterexample to this claim. Correct your answer if necessary after reviewing the responses of your labmates.

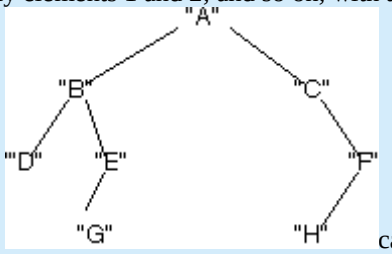
42.2.4 (Brainstorm) Another try

Another CS 61B student claims that a binary tree is complete if all the leaves in the tree come at the end of a breadth-first traversal of the tree. Give a counterexample to this claim. Correct your answer if necessary after reviewing the responses of your labmates.

42.2.5 (Display page) Using an array for heap elements

Using an array for heap elements

A binary tree can be represented in an array. The root of the tree would be array element 0; the two children of the root would be array elements 1 and 2; and so on, with array elements appearing breadth-first in the array.

For example, the tree 

can be represented as the array

"A"	"B"	"C"	"D"	"E"		"F"			"G"				"H"

This representation of course will waste memory if the tree isn't very "bushy". A max heap, however, is a complete binary tree, so we can store its elements in an array without wasting any space. Tree elements appear breadth-first in the array; element 0 is the top of the heap, elements 1 and 2 are its two children, and so on, with the children of element k being elements 2*k+1 and 2*k+2. Thus if we have the position of a node in the tree, we can access its children in constant time. We can also access its parent in constant time; the parent of the element in position k is at position (k-1)/2 (using integer division).

42.2.6 (Brainstorm) Accessing the third biggest item in a heap

Which elements of the underlying array might contain the *third largest* element of a max heap? Briefly explain your answer. Assume for simplicity that the heap does not contain any duplicate elements. Correct your answer if necessary after reviewing the responses of your labmates.

42.2.7 (Display page) Inserting and deleting an element

Inserting and deleting an element

Insertion and deletion both require attention to preserving the heap properties:

- the heap is a complete tree;
- the top of the heap is the largest element in the heap; and
- all the subtrees are heaps.

(We assume that we're working with max heaps here.) Insertion works as follows. The element to be inserted is added to the end of the heap array, thereby extending its size by 1. It is not necessarily in its proper place in

the heap, however, so it is "bubbled up" the tree until it's less than its parent item. Deletion proceeds by replacing the top of the heap by the last element and reducing the size by 1. Then the top of the heap is "bubbled down" to its proper position by exchanging it with its larger child. A complete example of how elements are inserted and deleted appears here.

42.2.8 (Display page) Experimenting with an animation

An animation of heap insertion/deletion

The web site http://www.student.seas.gwu.edu/~idsv/idsv.html provides an animation of the heap insertion and deletion operations. We suggest you work with a partner to experiment with the animation. Go to the web site, type your name, and click on "Priority queue" in the sidebar. The button labeled "Start Visualization" at the bottom right of the window starts the animation applet. Click "Single Step" and "I'll Pick" to gain full control of the animation. Enqueue 14, 15, and 50 to create a small heap. (It only accepts one- and two-digit integers.) Then try enqueuing a value that's larger than 50, another between 50 and 15, and a third that's less than 14. The "Step" button takes you through each step of the insertion. Then dequeue three elements. The "I'll Try" button provides a self-check facility for how the heap gets restored at each enqueue and dequeue operation. When you're done experimenting, click the "Stop Visualization" button in the browser window.

42.2.9 (Display page) Debugging a Heap class

Debugging a Heap class

The file ~cs61b/code/Heap.java partially implements a max heap of integers. The main method initializes a heap from a file whose name is provided as a command-line argument. (Integers appear one per line in the file.) It then checks that the values are in heap order, and repeatedly reads and inserts values into the heap. There is a bug in the isOK method and another in the insert method. Find and fix the bugs, changing the existing code as little as possible.

42.2.10 (Display page) Queue and PriorityQueue in java.util

Queue and PriorityQueue in java.util

The java.util library contains a Queue interface and a PriorityQueue class that implements Queue. Queue methods include the following:

- offer (*element*), which adds the element to the queue;
- peek ( ), which returns but does not remove the head of the queue;
- poll ( ), which retrieves and removes the head of the queue (returning null if the queue is empty).

The LinkedList class also implements Queue. PriorityQueues are implemented using binary min heaps. One might guess that from the online documentation:
"If multiple elements are tied for least value, the head is one of those elements—ties are broken arbitrarily. ... The Iterator provided in method iterator ( ) is not guaranteed to traverse the elements of the PriorityQueue in any particular order. ... Implementation note: this implementation provides O(log(n)) time for the insertion methods (offer, poll, remove ( ) and add) methods; linear time for the remove(Object) and contains(Object) methods; and constant time for the retrieval methods (peek, element, and size)."

42.3 Sort with a binary heap.*(4 steps)*

42.3.1 (Display page) Creating a heap all at once

Creating a heap all at once

Suppose you have an array of N values, and you want to turn them into a max heap. A straightforward approach is to insert them one by one into a heap that starts out empty. Since each insertion takes proportional to log N comparisons once the heap gets large enough, the whole process takes proportional to N log N time. However, there is a faster way that run in time proportional to N, involving the conversion of the collection of values to a max heap all at once. We start from the end of the array, building first tiny heaps, then combining them into larger heaps. (This is the first phase of a sorting algorithm named "Heapsort" that we'll revisit

shortly.) *Objects, Abstractions, Data Structures, and Design* doesn't mention it, but Wikipedia has the code and accompanying discussion. The process is animated here.

42.3.2 (Brainstorm) A way to sort with heaps?

A CS 61B student suggests the following method for printing the elements of an array in ascending order. First, make the array into a heap (whose maximum element is the top element). Then, since the top of the heap is the largest element, traverse the heap in postorder (traversing the left subtree before the right), printing each node value as it is encountered in the traversal. Unfortunately, the student is confused. Give an example for which the algorithm doesn't work.

42.3.3 (Display page) A better way to sort with heaps

A better way to sort with heaps

The *heap sort* algorithm to sort array values into ascending order has two phases. First, it creates a max heap out of the values. Then, it repeatedly removes the top element from the heap and places it at the front of the sorted sequence. There is a way to do this in place in the array that's explained in *Objects, Abstractions, Data Structures, and Design*, section 10.8. The web site accessible through the "Creating a heap all at once" step—http://www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/heapSort/heap.html—also provides an animation of the heap sort algorithm.

42.3.4 (Brainstorm) Isolating the smallest K out of N elements

Describe an algorithm that uses a heap of size K+1 to collect the K smallest elements in a list of length N in time proportional to N log K. Your description should be in detail sufficient for another CS 61BL student to recognize how your solution should be implemented in Java.

42.4 Homework*(3 steps)*

Reading

For the next lab please read chapters 30 and 31 from *Data Structures and Abstraction with Java*.

42.4.1 (Display page) A start on project 3

A start on project 3

Your t.a. will ask you on next Tuesday or Wednesday about who you'll be working in partnership, and how you intend to proceed on the project. Your answers to these questions will earn you up to 2 homework points. You should also supply a post and a response to each of the two discussions that follow.

42.4.2 (Discussion Forum) Questions about project 3

In a post (by Sunday night), supply a question that you or a labmate have about project 3. Then, provide a response by answering one of your labmates' questions or by significantly elaborating on an answer already provided.

42.4.3 (Discussion Forum) Likely challenges posed by project 3

In a post (by Sunday night), make a guess about what aspect of project 3 will be most difficult, and provide a brief explanation. Then, in response to one of your labmates, suggest a way to deal with or simplify the difficult aspect of the project that he or she described.

43 Quiz 21 *(1 activity)*

43.1 (Quiz) Review hashing.*(1 step)*

43.1.1 (Student Assessment) Quiz questions

1. Suppose that the Measurement class is correctly set up to allow Measurement objects to be keys in a java.util.HashMap object; that is, it has correctly defined equals and hashCode methods. What should be the contents of the hash table after execution of the following

program segment?

```
HashMap table = new HashMap ( );
table.put (new Measurement (6, 2), "clancy"); // Mike Clancy is 6'2"
table.put (new Measurement (6, 2), "hale"); // so is Paul Hale
```

2. The equals method in the Measurement class was originally defined something like this:

```
public boolean equals (Measurement m) {
  return m.myFeet == this.myFeet && m.myInches == this.myInches;
}
```

With Measurement.equals defined in that way, but with a correctly defined hashCode method, what is the contents of the hash table after execution of the following program segment?

```
HashMap table = new HashMap ( );
table.put (new Measurement (6, 2), "clancy"); // Mike Clancy is 6'2"
table.put (new Measurement (6, 2), "hale"); // so is Paul Hale
```