CSC207.01 2013F, Class 49: Heaps

Overview

- Preliminaries.
  - Admin.
  - Questions.
- Priority queues, revisited.
- Recent implementation techniques.
- Heaps.
- Adding elements to heaps.
- Removing elements from heaps.
- Heap sort.

Preliminaries

Admin

- Exam 2 returned.
  - I tried to be very detailed in my comments, even though I used a coarse-grained grading system.
  - Please turn in your academic honesty statements.
  - Please follow formatting conventions.
  - I hope to distribute additional notes later this week.
- Upcoming extra credit opportunities:
  - Learning from Alumni Thursday: Erik Hanson (in person)
  - CS Extras Thursday: Summer Opportunities in CS
  - CS Table Friday: TBD.
  - Swim meet Friday/Saturday.
  - Any self-care week activity.
  - *One Grinnell* rally on December 4 at 4pm (unless you are taking photos).
    - And yes, I've sent a note to Dean Arora about the scheduling.

Priority queues, revisited

ADTs

- Philosophy/Purpose
- Practicum (Use Cases)
- Procedures (Methods)

Priority queues

- Philosophy:
  - Put things in in any order, get them out in a highest-priority-first order.
- Use Cases:
  - Lots of them. News articles ... in order of popularity or date or ....
  - If we can *change* priority, gives fairness in, say, a printer queue.
    - We generally don't allow priorities to change.
  - Sorting!
- Procedures:
  - put, get, peek

Implementation:

- Unordered list/array
  - put: O(1)
  - get: O(n)
  - peek: O(n)
- Ordered array/list
  - put: O(n)
  - get: O(1)
  - peek: O(1)

Recent implementation techniques

- Trees - Two dimensional linked structures
- Hash tables - Clever uses of arrays

Heaps

- Modified binary search tree
  - Highest priority item at the top
  - And balanced
- A heap is a binary tree
  - With the *heap property*
    - The root is >= the root of each subtree
    - Each subtree has the heap property
  - That is 'nearly complete'
    - Every level except the last level is complete
    - The last level is shoved all the way to the left (or complete)
- See whiteboard for sample heaps

Adding elements to heaps

- Two invariants to maintain: Nearly complete and heap property
- Nearly complete seems harder to reconstruct if we're doing other stuff, so we'll prioritize that.
- Add the element at the end of the last level (or the beginning of the next level, if the last level is full).
- Yay! It's still nearly complete.
- But it doesn't satisfy the heap property.
- If the thing we just inserted is larger than the parent,
  - swap with the parent
  - and recurse up the tree
- Problem: How do we get the parent? Magic.

Removing elements from heaps

- The largest element is at the top
- Grab it (and be ready to return it)
- Grab the last thing on the last level and put it at the top
- The tree is now nearly complete
- Swap with larger child
- And recurse

Outstanding problems that we've relied on magic to resolve

- How do you get the parent? (Parent pointer?)
- How do you get the last element on the last level?
- Where do you insert the next element before swapping up?
- The amazing TN tree representation: Put it into an array in breadth-first, left-to-right order, top-down order
- If we also store size, the number of elements in the tree
  - The next element goes in position size++
  - The last element is in position size-1
  - The left child of p is at position $2p + 1$
  - *The right child of p is at position 2*p + 2
  - The parent is (p minus 1 or 2)/2
    - In C, this should 'floor it', and we'll be ok
  - Can you tell if a node at position p is a left child or right child?
    - Left child is odd
    - Right child is even

Heap sort

- We have an array
- We want to sort it
- Turn it into a heap

Here's the code:

```
// Turn the array into a heap
for (int i = 1; i < values.length; i++) {
   swapUp(i);
} // for
// Grab the largest element out of the heap and put them at
// the end.
for (int pos = values.length - 1; pos > 0; pos--) {
   swap(pos, 0);
   swapDown(0);
} // for
```

Analysis:

- Adding an element is log_2(n)
- Removing an element is log_2(n)
- Adding all of the elements is O(nlogn)
- Moving all the elements into the sorted position is O(nlogn)

Yay! Another O(nlogn) algorithms

- Quicksort is *expected* O(nlogn), can be O(n^2)
- Merge sort is O(nlogn) but requires extra space
- Heap sort is O(nlogn) and needs almost no extra memory