CSE 241 Class 17

Jeremy Buhler

October 28, 2015

And now for something completely different!

1 A New Abstract Data Type

So far, we've described ordered and unordered collections.

- Unordered collections didn't support efficient **min** or **max**
- Ordered collections supported both, plus total ordering operations (**pred** and **succ**)
- Both types support **find**.

We'll now look at a new, partly-ordered data type: a **Priority Queue**. It supports:

- Q.insert(k) – insert key k into the queue (could be record with key k, but we'll just think about key)
- Q.max() – find maximum key in queue
- Q.ExtractMax() – delete the maximum key from the queue

What doesn't a priority queue support?

- *No support* for search, predecessor, or successor
- can have min/ExtractMin instead of max/ExtractMax, but **not both**

Priority Queue Examples:

- You've got a bunch of jobs x_i running on a computer, each with some priority p_i. Each time the CPU becomes available to run jobs, which job has the highest priority? Moreover, running the job changes its priority, so you need to repeatedly modify that job and find the new highest-priority job.
- Oooooor... for certain graph algorithms that we will study, you need to repeatedly find *and remove or modify* the largest- or smallest-weight vertex or edge in a graph quickly.
- Oooooor... You receive sales figures for a million books, and you want to report the top ten best-sellers (without sorting the whole set).

2 Heaps for Priority Queues

One implementation of a priority queue is a **binary heap**. Abstractly, it looks like a binary tree, but it is not totally ordered like a BST. Nodes are sequentially numbered top to bottom, left to right.

Example:

- Max-first heaps satisfy the following *heap property*:

  > For every node x in a heap, x's key is at least as large as the keys of every node in the subtree rooted at x.

- Tree representation of a heap is only useful for visualization.
- In practice, heaps are coded using arrays.

Example:

- For node at index i, its left and right children are at $2i$ and $2i + 1$.
- For node at index i, i.parent $= \lfloor i/2 \rfloor$
- Root node is at index 1 to keep index arithmetic simple

- 0th element of the array is **unused**.
- (It's actually not that hard to switch to 0-based arithmetic.)

Note that any heap element can be reached starting from root in at most $\Theta(\log n)$ steps. IOW, tree representation is *balanced*.

3 Heap Max and Insert

- **max**() is trivial – root always has maximal key.
- But how do we insert a key into the heap?

Visualize on the tree.

- INSERT(k) first creates an empty node at the bottom of the heap.
- While k is greater than key of empty node's parent, swap empty node with parent ("bubble up")
- Finally, write k into empty node.

Cost is surely $O(\text{height}) = O(\log n)$.
Example: insert(15)

Why didn't we write k into the empty node until it was correctly placed?

- In array rep, a node corresponds to an array cell.
- Swapping a node x with an empty node y becomes just "write x's key into y."
- Saves us from explicitly copying k each time.

4 Heapify

Heapify is a function used to correct violations of the heap property. We'll need it for, e.g., ExtractMax.

- Let i be a node for which both subtrees i.left and i.right are heaps (i.e. they satisfy the heap property).
- However, i might violate the heap property – i.e. its key might be less than that of one or both children.
- After calling HEAPIFY on i, the tree rooted at node i will be a heap.

Idea:

- If i's key is smaller than that of some child, swap i with largest child.
- Heap violation, if any, now pushed down the tree one level.
- Recursively heapify the subtree that received node i.

Example:

Pseudocode for heap stored in array A, starting from ith cell.

```
HEAPIFY(A, i)
  if i is not a leaf
      j ← index of larger of i's children
      if A[j] > A[i]
          swap(A[i], A[j])
          HEAPIFY(A, j)
```

- *Note*: suppose you know the size n of the heap (you'll need this to implement it).
- How do you test if a node with index i is a leaf?
- If node i has no children, we have that

$$2i > n$$

- Hence, i (an integer) is a leaf if

$$i > \lfloor n/2 \rfloor$$

Note that HEAPIFY does constant work per level of the heap and so requires time $O(\log n)$ for heap of size n. Note also that we can avoid some copies if we perform swaps as described for insertion.

5 ExtractMax

We can use HEAPIFY to implement EXTRACTMAX.

- Suppose we delete the root node of the heap.
- Replace this node with the *last* element of the heap (shrinks array size by one)
- Root may now violate heap property.
- However, its subtrees (if any) are both heaps.
- To fix the heap, call HEAPIFY starting at root.

Example:

ExtractMax(A)
  $A[1] \leftarrow A[n]$
  $n--$
  Heapify(A, 1)

Cost of ExtractMax is constant plus one call to Heapify, hence takes time $O(\log n)$.

6 Further Fun Facts About Heaps

- Can turn an arbitrary array into a heap in time $O(n)$.
- **Heapsort**: turn input array into a heap, then repeatedly remove smallest element
- Can support *increasing* the key value of any node in a max-first heap. We'll need this for graph algorithms – think about how!