Today’s Topics:
✦ Binary Heaps
✦ Array Implementation
✦ FindMin/DeleteMin and Percolate Down
✦ Insert and Percolate Up
✦ BuildHeap, DecreaseKey, IncreaseKey
✦ Introduction to Binomial Queues
✦ Covered in Chapter 6 in the text

Heaps
✦ A binary heap is a binary tree that is:
  1. Complete: the tree is completely filled except possibly the bottom level, which is filled from left to right
  2. Satisfies the heap order property: every node is smaller than (or equal to) its children
✦ Therefore, the root node is always the smallest in a heap

Heaps: FindMin and DeleteMin Operations
  ➤ Run time = ?
✦ DeleteMin:
  ➤ Delete (and return) value at root node
  ➤ We now have a “Hole” at the root
  ➤ Need to fill the hole with another value
  ➤ Replace with smallest child?
  ◦ Try replacing 2 with smallest child and that node with its smallest child, and so on…what happens?
Heaps: DeleteMin Operation

✦ DeleteMin:
  - Delete (and return) value at root node
  - We now have a “Hole” at the root
  - Need to fill the hole with another value
  - Replace with smallest child?
    - Try replacing 2 with smallest child and so on, what happens?
    - Tree is no longer complete!
    - Let’s try another strategy...

Since heap is one node smaller, we need to empty the last slot

Steps:
  - Move last item to top; decrease size by 1
  - Percolate down the top item to its correct position in the heap

DeleteMin: Percolate Down

Replace with smaller child and go down one level
Done if both children are ≥ item or reached a leaf node
What is the run time?

Heaps: DeleteMin Operation

✦ DeleteMin:
  - Delete (and return) value at root node
  - We now have a “Hole” at the root
  - Need to fill the hole with another value
  - Since heap is one node smaller, we need to empty the last slot
  - Steps:
    - Move last item to top; decrease size by 1
    - Percolate down the top item to its correct position in the heap

Run time is O(depth of tree)
What is the depth of a complete binary tree of N nodes?
DeleteMin: Run Time Analysis

• Run time is \(O(\text{depth of heap})\)

• A heap is a complete binary tree

• What is the depth of a complete binary tree of \(N\) nodes?
  \(\Rightarrow\) At depth \(d\), you can have:
  \(N = 2^d\) (one leaf at depth \(d\)) to \(2^{d-1} - 1\) nodes (all leaves at depth \(d\))
  \(\Rightarrow\) So, depth \(d\) for a heap is: \(\log N \leq d \leq \log(N+1)-1\) or \(\Theta(\log N)\)

• Therefore, run time of DeleteMin is \(O(\log N)\)

Heaps: Insert Operation

Insert: Percolate Up

- Insert at last node and keep comparing with parent \(A[i/2]\)
- If parent larger, replace with parent and go up one level
- Done if parent \(\leq\) item or reached top node \(A[1]\)
- Run time?

Sentinel Values

• Every iteration of Insert needs to test:
  1. if it has reached the top node \(A[1]\)
  2. if parent \(\leq\) item

• Can avoid first test if \(A[0]\) contains a very large negative value (denoted by \(-\infty\))

• Then, test #2 always stops at top
  \(-\infty <\) item for all items

• Such a data value that serves as a marker is called a sentinel
  \(\Rightarrow\) Used to improve efficiency and simplify code
Summary of Heap ADT Analysis

✦ Consider a heap of N nodes
✦ Space needed: O(N)
  ➤ Actually, O(MaxSize) where MaxSize is the size of the array
  ➤ One more variable to store the size N
  ➤ With sentinel, array-based implementation uses N+2 space
  ➤ Pointer-based implementation: pointers for children and parent
    1. Total space = 3N + 1 (3 pointers per node + 1 for size)
✦ FindMin: O(1) time; DeleteMin and Insert: O(log N) time
✦ BuildHeap from N inputs: What is the run time?
  ➤ N Insert operations = O(N log N). Actually, can do better…
  ➤ O(N): Treat input array as a heap and fix it using percolate down
  ➤ See text for proof that this takes O(N) time.

Other Heap Operations

✦ Find(X, H): Find the element X in heap H of N elements
  ➤ What is the running time?
✦ FindMax(H): Find the maximum element in H
  ➤ What is the running time?

Other Heap Operations

✦ Find and FindMax: O(N)
✦ DecreaseKey(P, ∆, H): Decrease the key value of node at position P by a positive amount ∆. E.g. System administrators can increase priority of important jobs.
  ➤ First, subtract ∆ from current value at P
  ➤ Heap order property may be violated
  ➤ Percolate up or down?
  ➤ Running time?
✦ IncreaseKey(P, ∆, H): Add ∆ to current key value at P and percolate down. Running Time: O(log N)
  ➤ E.g. Schedulers in OS often decrease priority of CPU-hogging jobs
✦ Delete(P, H): E.g. Delete a job waiting in queue that has been preemptively terminated by user
  ➤ Use DecreaseKey followed by DeleteMin.
  ➤ How? (Idea: Decrease key to bubble node at P to the top)
  ➤ Running Time?
Other Heap Operations

- **Delete(P,H):** E.g. Delete a job waiting in queue that has been preemptively terminated by user
  - Use DecreaseKey($P\rightarrow\infty$,H) followed by DeleteMin(H).
  - Running Time: $O(\log N)$
- **Merge(H1,H2):** Merge two heaps $H_1$ and $H_2$ of size $O(N)$. $H_1$ and $H_2$ are stored in two arrays. E.g. Combine queues from two different sources to run on one CPU.
  1. Can do $O(N)$ Insert operations: $O(N \log N)$ time
  2. Better: Copy $H_2$ at the end of $H_1$ and use BuildHeap
     Running Time: $O(N)$
Can we do even better? (i.e. Merge in $O(\log N)$ time?)

Binomial Queues

- **Binomial queues support all three priority queue operations**
  - Merge, Insert and DeleteMin in $O(\log N)$ time
- **Idea:** Maintain a collection of heap-ordered trees
  - *Forest of binomial trees*
- **Recursive Definition of Binomial Tree** (based on height $k$):
  - Only one binomial tree for a given height
  - Binomial tree of height 0 = single root node
  - Binomial tree of height $k = B_k = \text{Attach } B_{k-1} \text{ to root of another } B_{k-1}$
- Binomial tree of height $k$ has exactly $2^k$ nodes (by induction)

The First Four Binomial Trees

<table>
<thead>
<tr>
<th>$B_0$</th>
<th>$B_1$</th>
<th>$B_2$</th>
<th>$B_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="B0" /></td>
<td><img src="image" alt="B1" /></td>
<td><img src="image" alt="B2" /></td>
<td><img src="image" alt="B3" /></td>
</tr>
</tbody>
</table>

Why are these trees called binomial? (hint: how many nodes at depth $d$?)

Why “Binomial”?

<table>
<thead>
<tr>
<th>$B_0$</th>
<th>$B_1$</th>
<th>$B_2$</th>
<th>$B_3$</th>
<th>$\ldots$</th>
<th>$B_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="B0" /></td>
<td><img src="image" alt="B1" /></td>
<td><img src="image" alt="B2" /></td>
<td><img src="image" alt="B3" /></td>
<td>$\ldots$</td>
<td><img src="image" alt="Bk" /></td>
</tr>
</tbody>
</table>

Number of nodes at different depths $d$ for $B_k = [1], [1 1], [1 2 1], [1 3 3 1], \ldots = \binom{k}{d}$

Binomial coefficients of $(a + b)^k = \binom{k}{d}d!$
Binomial Queues

Binomial Queue = “forest” of heap-ordered binomial trees

B₀ B₂

Binomial queue H₁
5 elements = 101 base 2
→ B₂ B₀

B₀ B₁ B₃

Binomial queue H₂
11 elements = 1011 base 2
→ B₃ B₁ B₀

Next Class:
How do we merge H₁ and H₂?
More on Binomial Heaps and Priority Queues

To Do:
Read Chapter 6
Programming Assignment #1 will be on-line tomorrow (due April 27)