Finishing tree traversals
and
Implementing priority queues: Heaps

2/29/16
Announcements

• Last HW out: Due Friday
  – Design problems used for review next Monday

• Reading:
  – Wednesday: Chapter 28
  – Friday: Chapter 29

• Quiz this week: heap operations

• Next week:
  – Monday: Your design problems and review
  – Tuesday: No lab; final hopefully available
Recall: Tree traversals

```java
void traverse(Node n) {
    if(n == null) return;
    //preorder: print here
    traverse(n.left);
    //inorder: print here
    traverse(n.right);
    //postorder: print here
}
```
Draw two trees with preorder traversal a, b, c, d, e, f, g, h
Draw two trees with inorder traversal a,b,c,d,e,f,g,h
Draw the unique tree with both

- preorder traversal:
  a,b,c,d,e,f,g,h

- inorder traversal:
  b,c,a,f,e,g,d,h
Alternate: Level-order traversal

- Visits the nodes one level at a time
  (the root, its children, its grandchildren...)

- Implemented using a queue
Recall: Priority Queue ADT

- Store collection of values and read them out in priority order (as given by a Comparator)

- Operations:
  - void enqueue(value)  //add value to collection
  - value dequeue()      //remove & return next val
  - value peek()         //return next val w/o removing
  - int size()           //get collection size
Heaps

• A heap is a data structure specifically for priority queues

• Displayed as a complete binary tree (two children for every node except possibly nodes on the right of the bottom level)

• Each parent is <= either child (by Comparator)
Which of the following is a valid heap?

A) 
```
1
  12
  97
  121
  102
```

B) 
```
3
  8
  46
  123
```

C) 
```
6
  9
  45
```

D) 
```
9
  7
  73
  62
```

E) Not exactly one of the above
Which of the following is a valid heap?

A)  
```
1
/  
12 97
/  
121 102
```

B)  
```
3
/  
8
/  
46 123
```

C)  
```
6
/  
9 45
```

D)  
```
9
/  
7
/  
62 73
```

E) Not exactly one of the above
Efficient representation of a heap

• Values stored in an array indexed from 1
• Root stored in index 1
• Can move thru the tree via index arithmetic:
  – left child of value in index i: index 2i
  – right child of value in index i: index 2i+1
  – parent of value in index i: index i/2 (round down)
Which of the following is a valid heap?

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>12</td>
<td>18</td>
<td>142</td>
<td>215</td>
</tr>
<tr>
<td>B.</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>12</td>
<td>8</td>
<td>142</td>
<td>7</td>
<td>18</td>
<td>215</td>
</tr>
<tr>
<td>C.</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>142</td>
<td>12</td>
<td>8</td>
<td>6</td>
<td>18</td>
<td>215</td>
</tr>
<tr>
<td>D.</td>
<td>4</td>
<td>8</td>
<td>5</td>
<td>12</td>
<td>142</td>
<td>6</td>
<td>215</td>
<td>7</td>
<td>18</td>
</tr>
<tr>
<td>E. Not exactly one of the above</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Which of the following is a valid heap?

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>12</td>
<td>18</td>
<td>142</td>
<td>215</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>12</td>
<td>8</td>
<td>142</td>
<td>7</td>
<td>18</td>
<td>215</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>142</td>
<td>12</td>
<td>8</td>
<td>6</td>
<td>18</td>
<td>215</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>8</td>
<td>5</td>
<td>12</td>
<td>142</td>
<td>6</td>
<td>215</td>
<td>7</td>
<td>18</td>
</tr>
</tbody>
</table>

E. Not exactly one of the above (A and B)
How do we add a value to a heap?
Upheap: letting values rise up

```c
void upheap(int i) {
    //i is index that may need to move up
    while((i > 1) && (A[i/2] > A[i])) {
        swap A[i] and A[i/2]
        i = i / 2;
    }
}
```
What heap results when 64 is added to the heap below?
How do we dequeue the minimum value?
Downheap: letting values sink down

```c
void downheap(int i) {
    while(2*i <= size) {
        int j = 2*i; // smaller child’s index (val to replace A[i])
        if((j < size) && (A[j] > A[j+1]))
            j++;
        if(A[i] <= A[j])
            break;
        swap A[i] and A[j]
        i = j;
    }
}
```