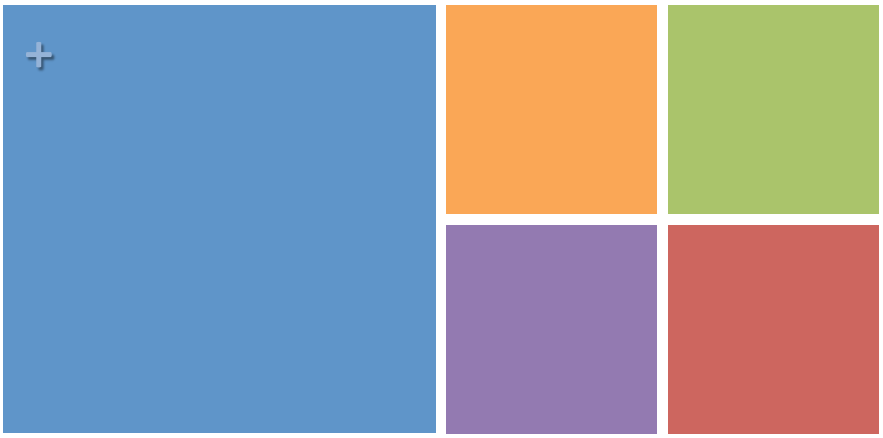


Lecture 17: Priority Queues

Today

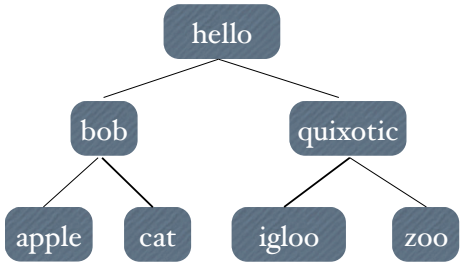
- Reading
  - JS 13.1-13.4.1 (Priority Queues)
- Objectives
  - Build up to Priority Queues
    - Array representation of trees
    - Heaps

Recap Tree Traversals

- Unlike lists, there is no standard order of traversal for trees
- A *tree traversal* is a specific order in which to traverse a tree structure
- Pre-order Traversal
  - node, left subtree, right subtree
- In-order Traversal
  - left subtree, node, right subtree
- Post-order Traversal
  - left subtree, right subtree, root

Recap Tree Traversals

```
// Prints values using pre-order traversal
public void preOrderTraversal() {
    if(!isEmpty()) {
        System.out.println(val);
        left.preOrderTraversal();
        right.preOrderTraversal();
    }
}
```



Priority Queues

- Priority Queue allows access to only the smallest (largest) element
- Implementation based on trees
- Contrasting priority queues
  - Unlike stacks and queues, order in does not determine order out
  - Unlike lists, cannot control where element is stored
  - Cannot traverse a priority queue
- Applications
  - Running processes on the CPU (top)
  - Search (A*)

Priority Queues Interface

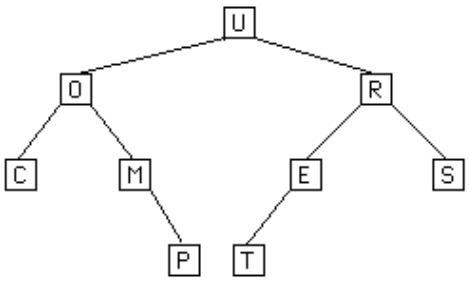
```
public interface PriorityQueue<E extends Comparable<E>> {

    // returns the minimum value
    public E getFirst();

    // removes and returns the minimum value
    public E remove();

    // adds a value to the priority queue
    public E add();
    …
}
```

Array Representations of Trees

- Priority queues can be efficiently implemented using binary trees!
- We've seen a recursive implementation of a binary tree
- We can, in fact, represent an entire binary tree using one array!
- Let `data` be an array
  - `data[0]` contains the root
  - the left child of `data[i]` is in `data[2*i+1]`
  - the right child of `data[i]` is in `data[2*i+2]`

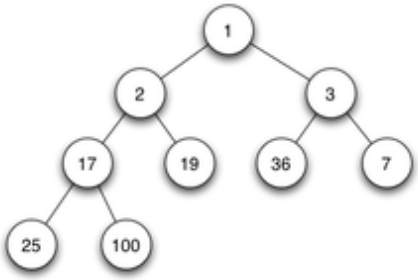
Array Representations of Trees

- Let `data` be an array
  - `data[0]` contains the root
  - the left child of `data[i]` is in `data[2*i+1]`
  - the right child of `data[i]` is in `data[2*i+2]`



Array Representations of Trees

- Let `data` be an array
  - `data[0]` contains the root
  - the left child of `data[i]` is in `data[2*i+1]`
  - the right child of `data[i]` is in `data[2*i+2]`

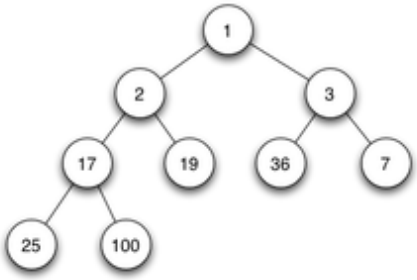


Array Representation of Trees

- A tree of height h requires 2^{h+1}-1 spots in the array regardless of the number of nodes N
- Bad for long, skinny trees
  - since N is only O(h)
- Good for full or complete trees
  - since N is $O(2^{h+1})$ anyways

Heaps

- A heap is a complete binary tree whose root contains the minimum value and whose subtrees are, themselves, heaps
- A heap is a complete binary tree whose values are in ascending order on every path from the root to the leaf



from the course webpage!

Implementing a Priority Queue

- Heaps provide an excellent implementation of a priority queue!
- Heaps themselves can be efficiently implemented using an array representation of trees
- Operations
  - `add()`
  - `remove()`