CSCI 136 Data Structures & Advanced Programming

Bill Jannen

Lecture 27

April 21, 2017

Administrative Details

- Lab 8: Lexicon
  - Due Monday
  - Turn in one lab per group
- Lab 9: Simulating Business (Queues)
  - Lab 13.7 from the book
    - Bank vs. Supermarket
  - Handout with instructions/hints will be posted *later* than usual so you have time to come up with your own designs

Last Time

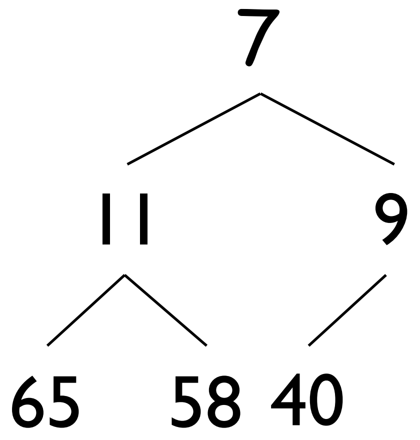
- Compact tree representation
- Priority queue interface
- OrderedVector Priority queue
- Introduced heaps

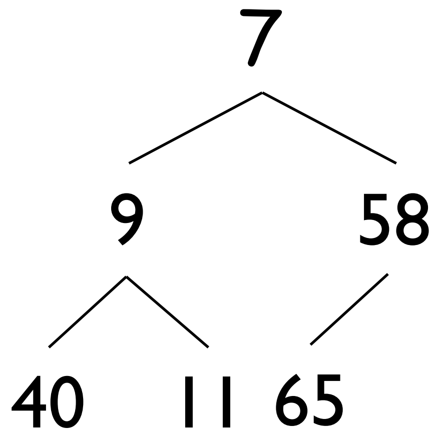
Today’s Outline

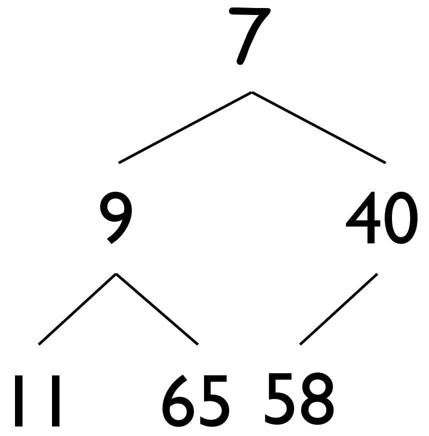
- Heap implementations

Heap

- **Definition:** A binary tree whose root is a minimum value and whose subtrees are also heaps
  - node.value() <= node.left.value()
  - node.value() <= node.right.value()
- **Implication:** any path from root to leaf is in ascending order
- **Implication:** several valid heaps for same data set (no unique representation)


7
11
9
65
58
40


7
9
58
40
11
65


7
9
40
11
65
58

Implementing Heaps

What we know:

- We can compactly store complete binary trees in Vectors
- Heaps are complete binary trees with added invariants

- …We can compactly store a heap in a vector!
  - class VectorHeap in structure5

Implementing Heaps

Mapping a Binary Tree to a Vector:

- root at position 0
- left child at position 2i+1
- right child at position 2i+2
- parent at (i-1)/2

Mapping the heap invariants to a Vector

- data[i] <= data[2i+1]
- data[i] <= data[2i+2]

Implementing Heaps

```
public class VectorHeap<E extends Comparable<E>> implements PQ<E> {
    protected Vector<E> data;

    public VectorHeap() {
        data = new Vector<E>();
    }

    protected static int parent(int node) {
        return (node - 1) / 2;
    }
    protected static int left(int node) {
       return 2*node + 1;
    }

    protected static int right(int node) {
       return 2*node + 2;
    }
}
```

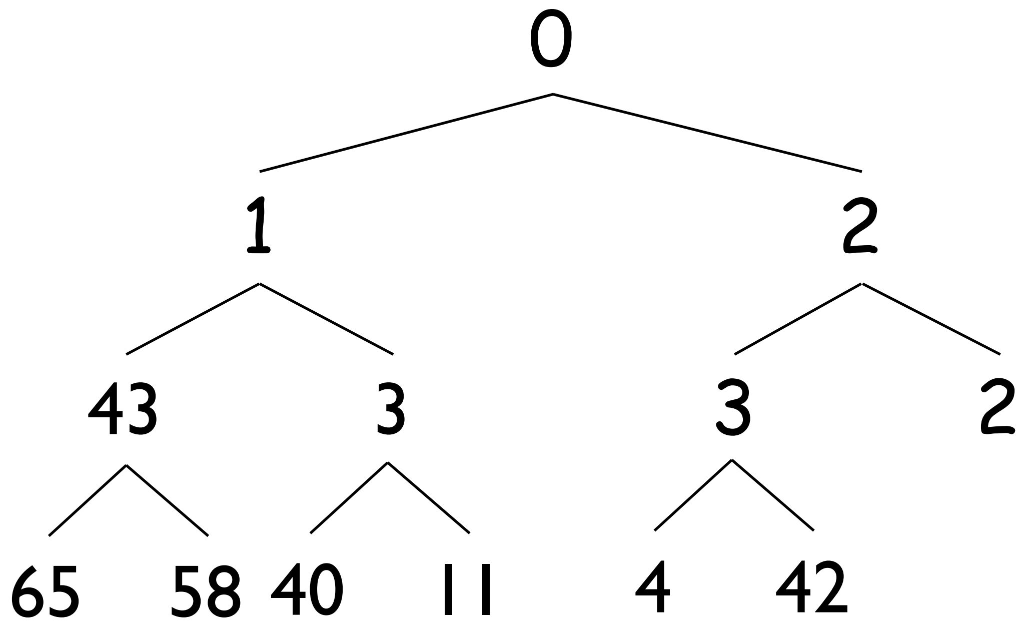
Implementing Heaps: add(obj)

- Assume we start with a complete heap
    - (not hard…empty heap is complete)
- After adding a value, resulting heap should still be a complete heap

Strategy:

1. Place the element in the tree so that the tree is complete (but not necessarily a heap)
2. Perform operations to make it a heap (while maintaining completeness)

heap.add(-1)

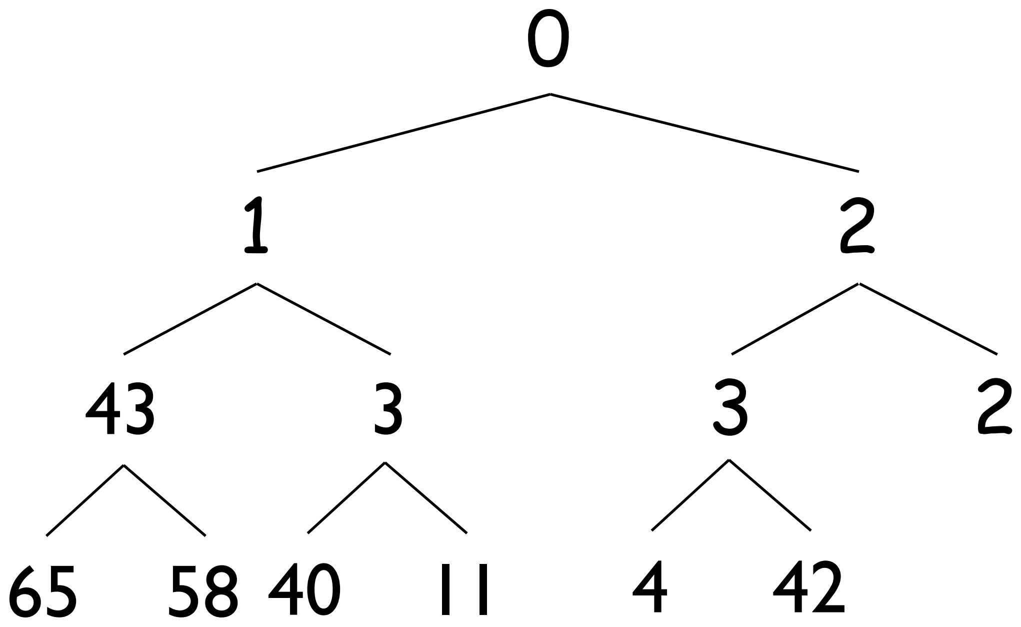


Insertion

Implementing Heaps: remove()

- Same strategy: preserve tree completeness, then swap elements to “heapify”
  1. Remove top (root) element
  2. Replace with rightmost leaf (last element)
  3. “Push down” until heap is valid again (by always swapping element with *smallest* child…why?)

heap.remove()



VectorHeap Summary

- [Look at implementation of PQ using VH]

- Add/remove are both O(log n)
- Data is not completely sorted
  - Partial order is maintained

Skew Heap

- What if heaps are not complete BTs?
- We can implement PQs using skew heaps instead of "regular" complete heaps
- Key differences:
  - Rather than use Vector as underlying data structure, use BT
  - Need a merge operation that merges two heaps together into one heap
- Details in book…

Tree Recap So Far…

- General Binary Trees
  - Express hierarchical relationships
  - Ordering is based on some external notion
    - i.e., ancestry, game boards, decisions, etc.
- Heap
  - Partially ordered (complete) binary tree based on priorities
  - Node invariants: parent has higher priority than both children