Announcements

- Program 5 due tomorrow
- HW3 up today/tomorrow
Last time: Trees

Def: A tree $T$ is a set of nodes storing elements in a parent-child relationship.

$T$ has a special node $r$, called the root.
Each node (except $r$) has a unique parent.
More dfs

- child
- siblings: share common parent
- leaves: vertices with no children
- internal nodes: not leaves
- rooted subtree
- descendant/ancestor
Binary Tree

- Every node has \( \leq 2 \) children.

- left 2 pointers
  - right

code for this will be written next week
Nice trick

Can be pointers or array based!

left(x) = 2x + 1
right(x) = 2x + 1
**Depth + Height** - defined recursively

depth: \( \text{depth}(r) = 0 \)

\( \text{depth}(v) = \text{depth}(\text{parent}(v)) + 1 \)

\( \text{depth}(\text{tree}) = \text{max} \ \text{depth} \)

height: \( \text{height}(\text{leaf}) = 0 \)

\( \text{height}(v) = \text{max}(\text{height of children}) + 1 \)

\( \text{height}(\text{tree}) = \text{height}(r) \)

\( \text{height}(T) = \text{depth}(T) \)

\( \text{height}(v) \neq \text{depth}(v) \)
Data Structures (for trees)

Priority Queue: supports the following operations

\( \text{push} \)

- \( \text{insert}(e) \): adds element \( e \) to the data structure \( \Theta(n) \)

\( \text{pop} \)

- \( \text{removeMax()} \): removes maximum element
- \( \text{getMax()} \): returns maximum element (size, empty)

How to build?

Use lists or vectors & sort (something will take \( \Theta(n) \) time)
Heaps

A binary tree where:

- For every node \( v \) (other than root), the key stored at \( v \) is \( \leq \) key stored at \( v \)'s parent.

- The tree is complete: levels 0 to \( h-1 \) are full, and level \( h \) is filled in left to right order.
Max Heap

return this for getMax

know is ≥ children
Insert:

insert (11)

insert (52)

"bubbling up"
Insert: 6, 10, 2, 12, 60, 1
Remove Max  "bubble down"
Running times

How many comparisons/sweeps?

$O(n)$ (or $O(d)$)

Total $O(\log_2 n)$ for any operation

Versus $O(1)$ or $O(n)$
Binary Search Trees