CS2100

Trees
Priority Queues

Recap

- HW due on 3/18
- Next HW: pen/paper one
- Lab due Friday

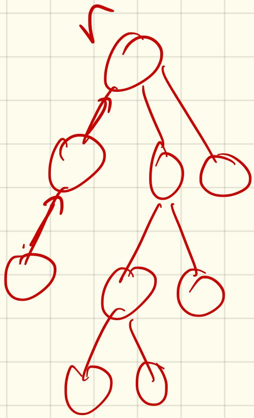
Trees:

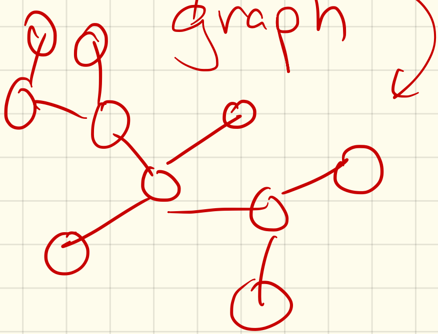
Dfn: A **tree** (in data structures) is a set of nodes which store elements in a parent-child relationship.

Any tree has a special uppermost node called the **root**.

Each node (except the root) has a unique parent.

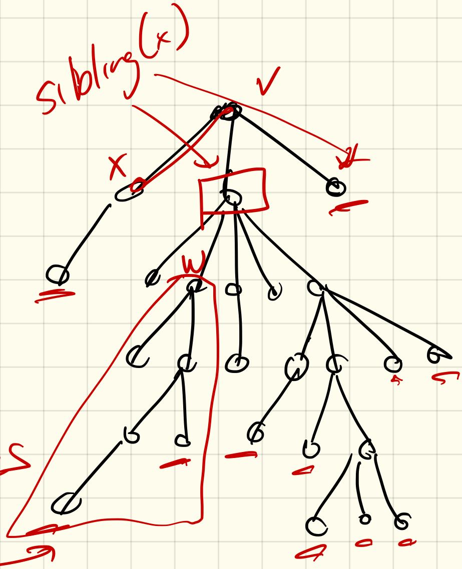
Note: **Not** quite the same as in graph theory! → acyclic graph





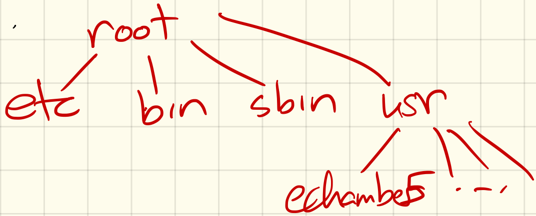
More dfns

- child
- sibling
- leaves: no children
- internal nodes: not leaves
- rooted subtree
- descendent / ancestor



<u>Practical examples</u>: anything w/ parent-child relationship (rather than linear):

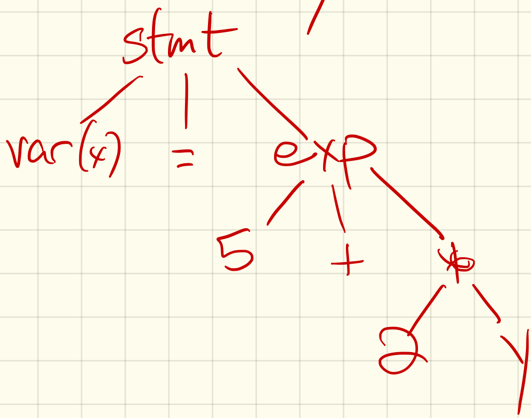
- File systems:



- Family trees

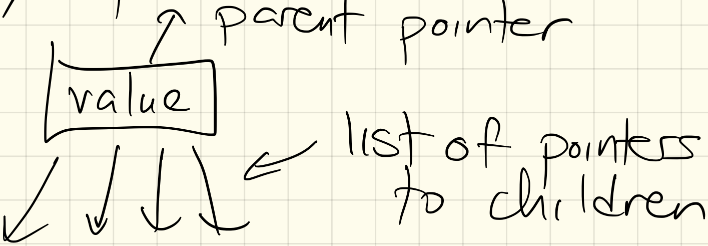
- Parse trees for sentences or equations

$x = 5 + 2 * y;$

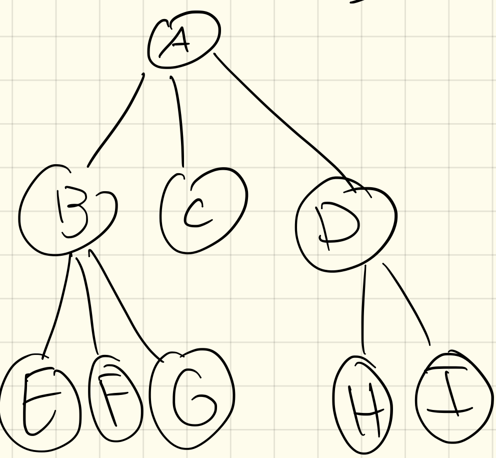


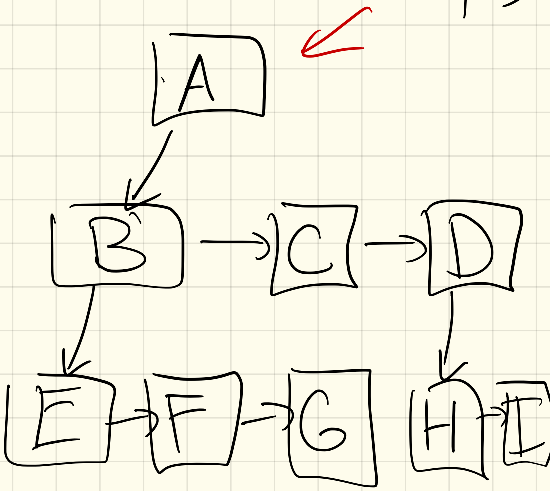
A general tree implementation:

Usually pointer based



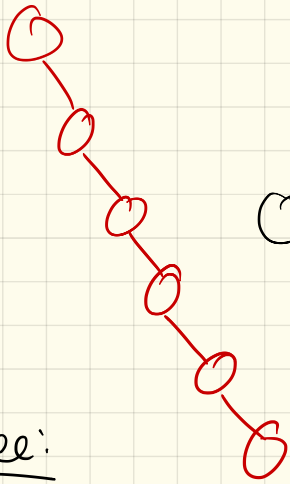
Or sibling based (so no list of ptrs)

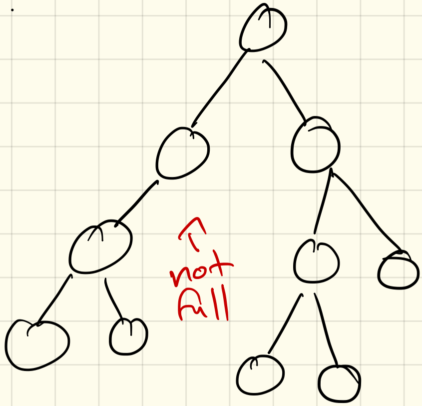




Binary Trees

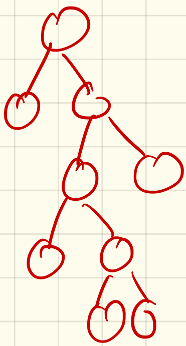
- Every node has ≤ 2 children:





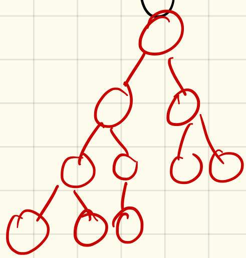
Full tree:

Every node has 0 or 2.



Complete tree:

Every node has 2 except perhaps lowest level, which fills left to right.



Depth + Height:

Both defined recursively.

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

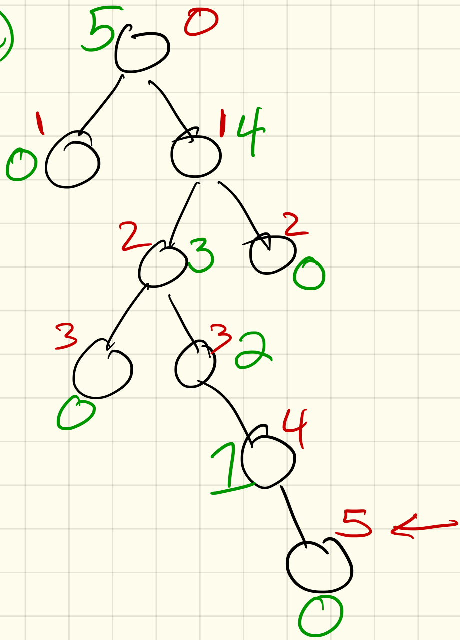
$\text{depth}(v) = \text{depth}(\text{parent}(v)) + 1$

$\text{depth}(T) = \max_v \text{depth}(v)$

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

$\text{height}(v) = \max\{\text{height of } v\text{'s child}\} + 1$

(we say -1 if not there)

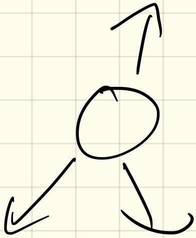


$\text{height}(T) = \text{height}(r) = \max_v h(v)$

Implementation:

- Pointer based: 3 pointers

Node:
data
parent
left
right

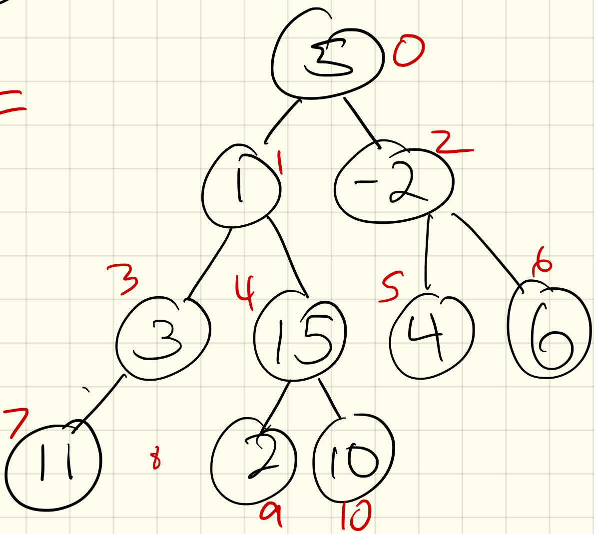


- Array based:

$$\text{left}(v) = 2v+1$$

$$\text{right}(v) = 2v+2$$

$$\text{parent}(v) = \left\lceil \frac{v}{2} \right\rceil - 1$$

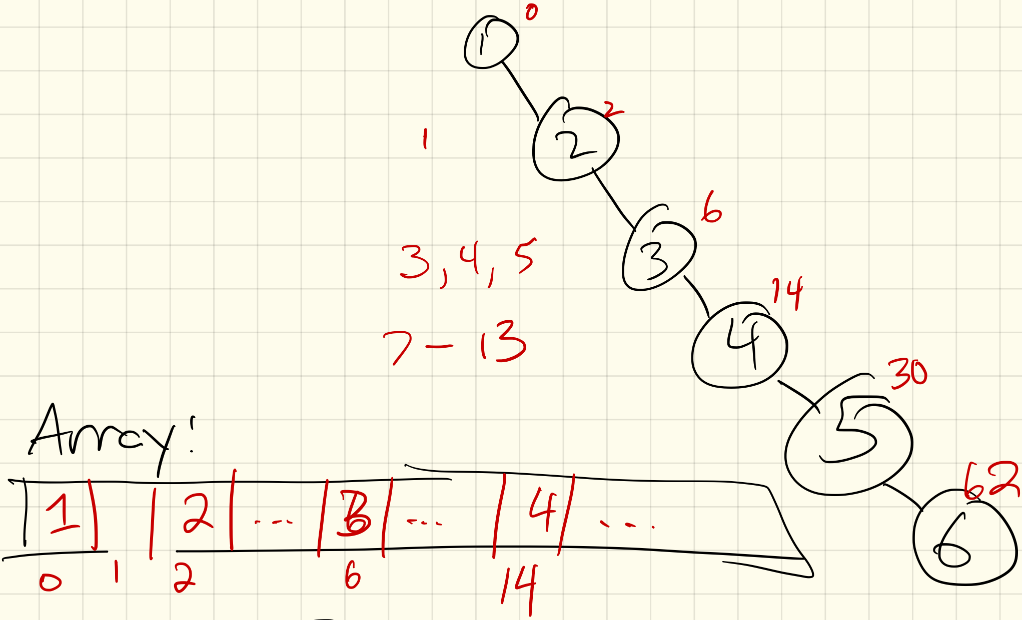


A:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
5	1	-2	3	15	4	6	11		2	10				

Potential downside of array:

Space!



How big?

n values → $O(2^n)$

Don't use for "sparse" trees.

First data structure:

- Priority queues:

Operations: Given priority queue PQ:

- insert(e, k): adds e to PQ w/ priority k
- getMax(): returns maximum ~~value~~ key in PQ
- removeMax():

(plus size & empty)

Why would this be useful?

How to implement?

Vector or list

```
class PQ {
private:
    Vector<int> values;
```

O(n) [getMax

linear Search]

O(n) [removeMax

~~li~~near Search

remove]

O(1) [insert(e)

amortize ↓

push_back]

Another way:

- keep data sorted

$O(n)$ [insert: binary search ← $\log n$
call insert ← $O(n)$

$O(1)$ [getMax:
return last element

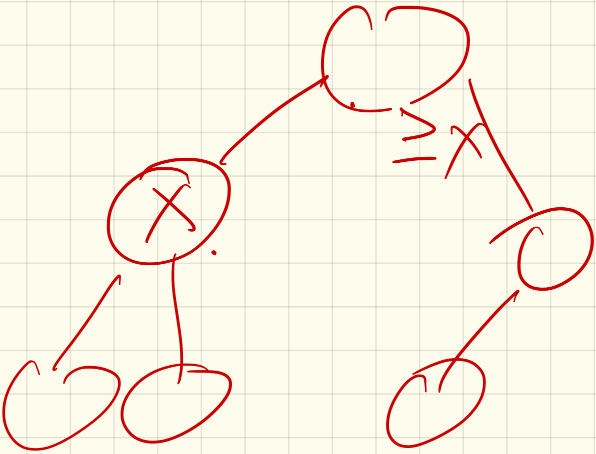
$O(1)$ [removeMax
pop_back

Heap: A binary tree where:
One way to do a priority queue

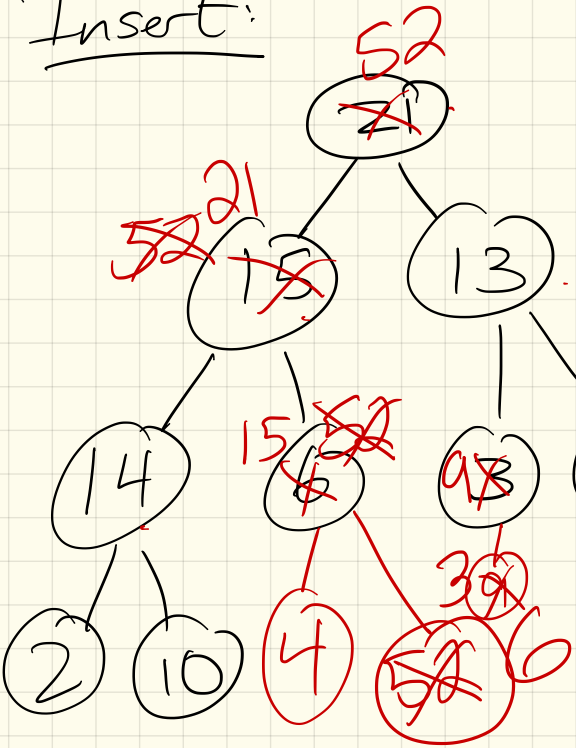
#1 • For every node v (other than r) the key stored at v is ≤ key stored at v's parent

#2 • The tree is complete:
levels 0... h-1 are full & h is filled in left to right.

Ex:



Insert:



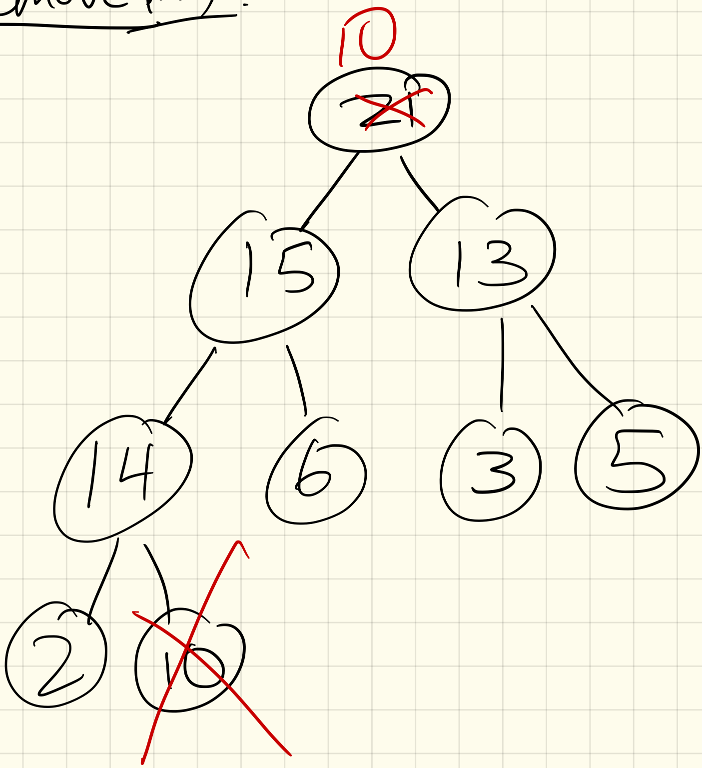
(not a search tree)

- insert(4)
- insert(52) – breaks #1

insert(9)

"bubbling up"

Remove Max:



"Bubble down"

Next time:

Coding & implementing!

We'll do array based.

Why?