CS180 - Sorting + trees

Note Title 11/2/2011

Announcements

Exam 1

Average :

Sorting Algorithms

Why do we care?

- Insertion
- Selection
- Merge
- Bubble
- Quick
- Bucket
- Radix
- Shell
- van Em de Boas

⋮

Bucket Sort

n elements, each between 0 and $N-1$

Can we do better than $O(n \log n)$?

Radix Sort : for multiple-key sorting

Ex: (1, 5), (2, 1), (4, 2), (3, 3), (5, 4), (3, 1), (2, 2), (5, 1), (2, 4)

Sort lexicographically: (use repeated bucket sorts)

Practicalities

Expirimentally, quicksort runs faster than merge on small inputs.

Why?

More practicalities

- If implemented well, the running time of insertion sort is $O(m+n)$, where m = # of <u>inversions</u> (or out of order elements)

<u>Conclusion</u>:

- If the range of values is small, bucket sort (or radix sort) are faster.

Trees:

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

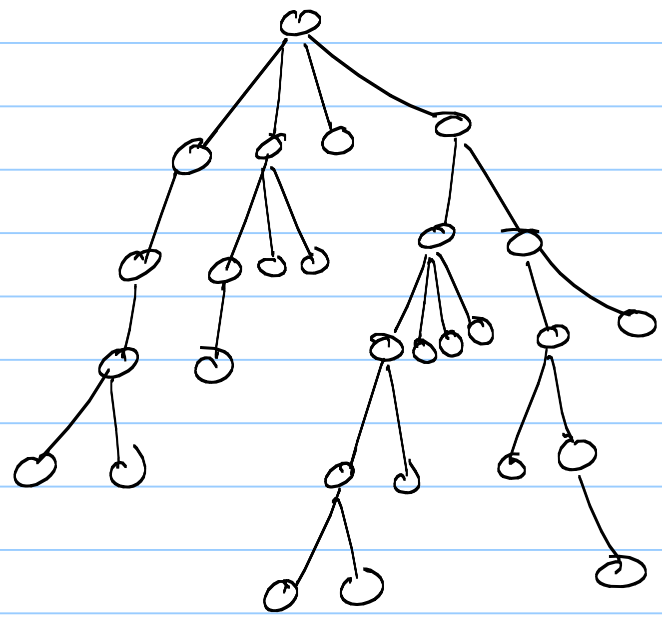
T has a special node r, called the **root**.

Each node (except r) has a unique **parent**.

Compare to lists:

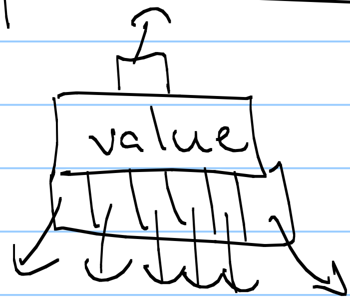
More dfns

- child
- siblings
- leaves
- internal nodes
- rooted subtree
- desendant / ancestor



General Tree Implementation

Pointer based:



Need a list of children in each node.

Applications

Anything where relationships are more complex than linear orderings!

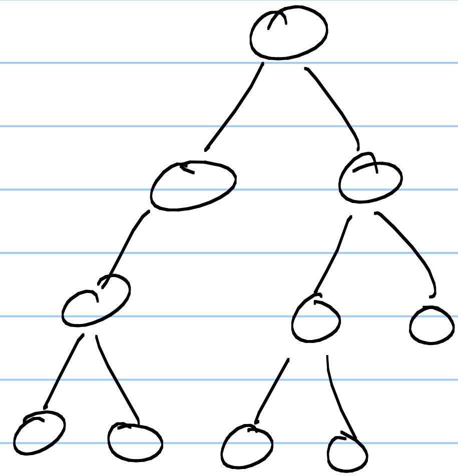
Ex:

- family tree
- file systems
- Numeric expressions

⋮

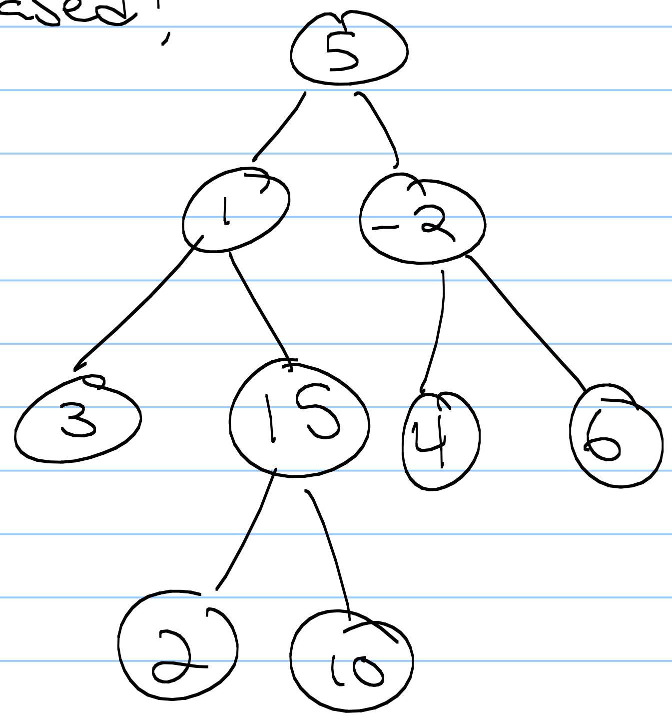
Binary Tree

- Every node has ≤ 2 children.



Nice trick

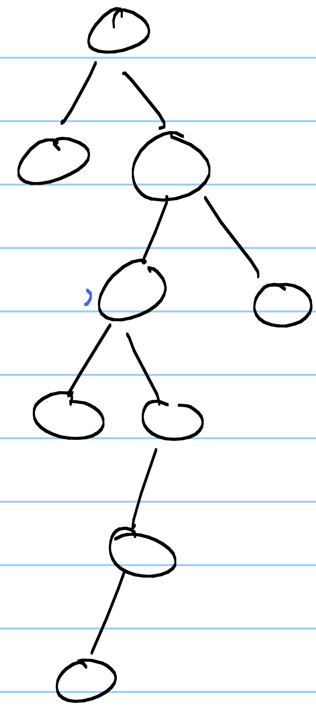
Can be pointers or array based!



Depth & Height - defined recursively

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

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



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

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