CS180 - Trees

Note Title 10/13/2010

Announcements

- HW3 is due in 1 week
- No class Monday

Last time: Sorting - (Ch. 10)

- Insertion sort: - simple & easy to code
  - useful for smaller lists
  - $O(n^2)$ $O(n+k)$
- Merge Sort: - fastest worst case: $O(n \log n)$
  - difficult to run in-place
- Quicksort: - worst case $O(n^2)$
  - BUT $O(n \log n)$ in practice
- Bubble Sort: - Slow - $O(n^2)$
  - again, OK for small problems (but less useful than insertion sort)

Bucket Sort

$O(n+N)$

Suppose we have n numbers, all between $\underbrace{[0, N-1]}$.

Turn things around: use $0 \ldots N-1$ as keys.

Put element with key i in spot $A[i]$

$\underline{A[1]}$ $\underline{A[2]}$ $\underline{A[3]}$ . . . $\underline{A[26]}$ . . .

3 26

3

Radix Sort

Suppose we have n ordered pairs (1,3), (5,11), (3,1)

all numbers b/t 0 & N-1

(1,1), (1,3), (1,5)

1	2	3	4	5	. . .	11
(1,1)		(1,3)		(1,5)		(5,11)
(3,1)				(2,5)		
				(3,5)		

→ (1,1) (3,1) (1,3) (1,5) (2,5) (3,5) (5,11)

$\hookrightarrow$ (1,1) (3,1) (1,3) (1,5) (2,5) (3,5) (5,11)

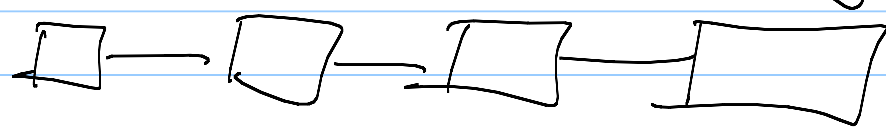
Bucket sort again: (on 1^{st} coord)

1	2	3	4	5
(1,1)	(2,5)	(3,1)		(5,11)
(1,3)		(3,5)		
(1,5)				

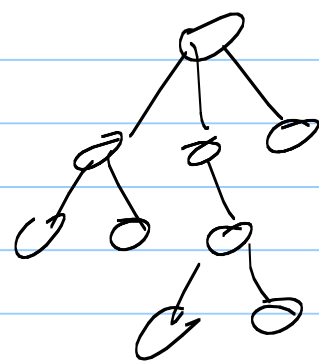
Ch 6 - Trees

All data structures so far have expressed linear orderings:

lists
vectors
stacks
queues

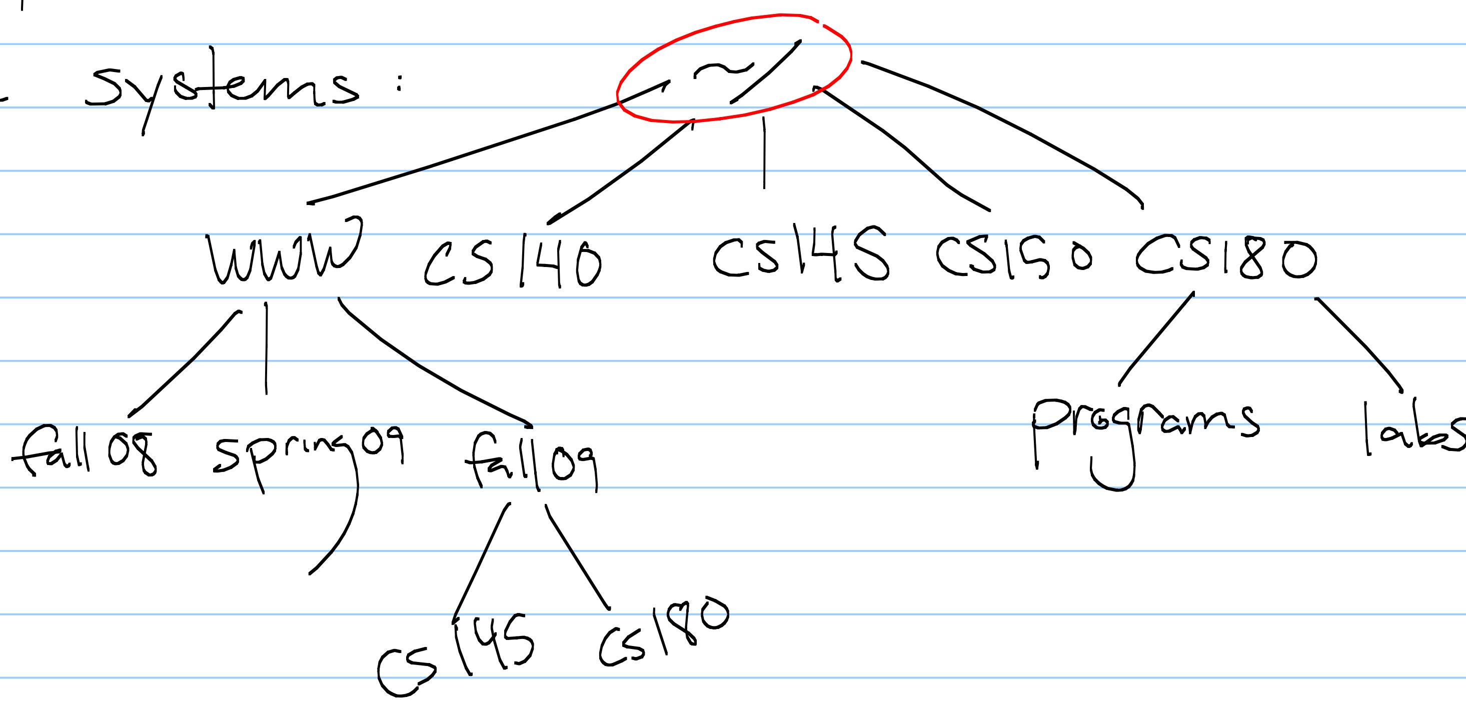


Some structures require more complex relations.



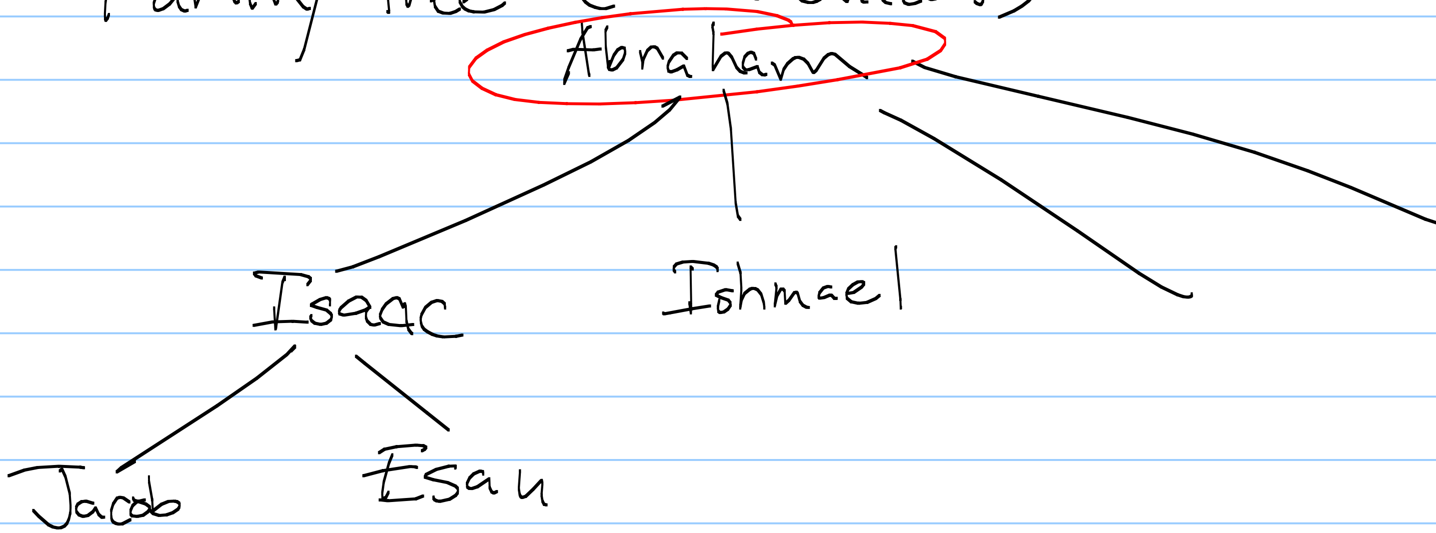
Examples:

- File systems:



Ex:

- Family tree (Patriachical)



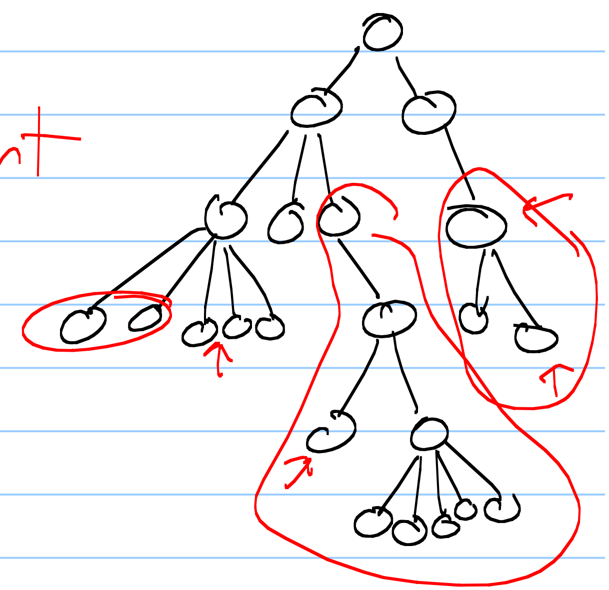
Definitions

A tree is set of nodes storing elements in a parent-child relationship.

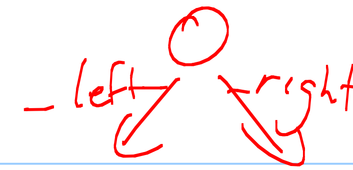
- T has a special node r, called the root
- Each node (except r) has a unique parent

More dfns

- child
- siblings - share a common parent
- leaves - have no children
- internal nodes - have at least 1 child
- rooted subtree
- ancestor - parent of a parent
- descendant - children of child



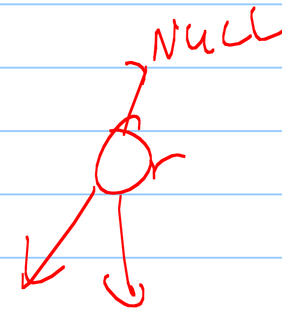
Tree Data Structure



What sort of data might a tree class need?

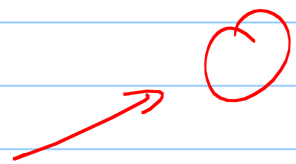
Tree class will need a root.

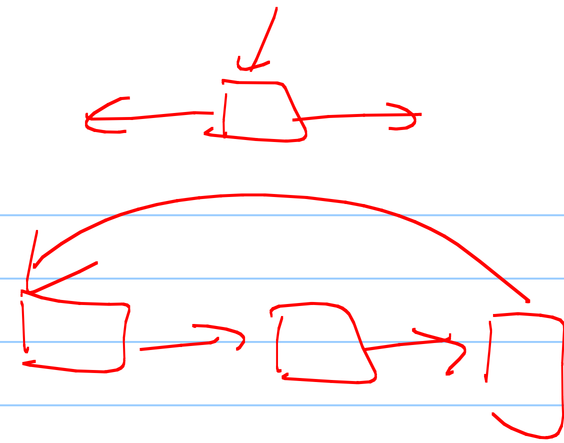
Node * _r;
int _size;



<u>Node struct</u>: Node* _parent;
Child pointers ← 2 pointers, _left + _right
Object _data;
int _aux;
depth or height

Iterators in Trees





up

down left

down right

private data:

```
Node* _location;
BinaryTree* _mytree;
```

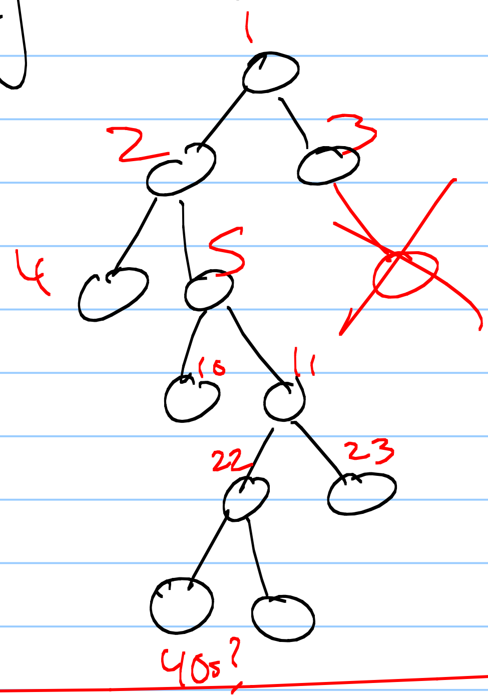
Code for trees

We'll come back to this after fall break.

Our first implementation will be of a special kind of tree, since we can avoid pointers in some cases.

Binary Trees

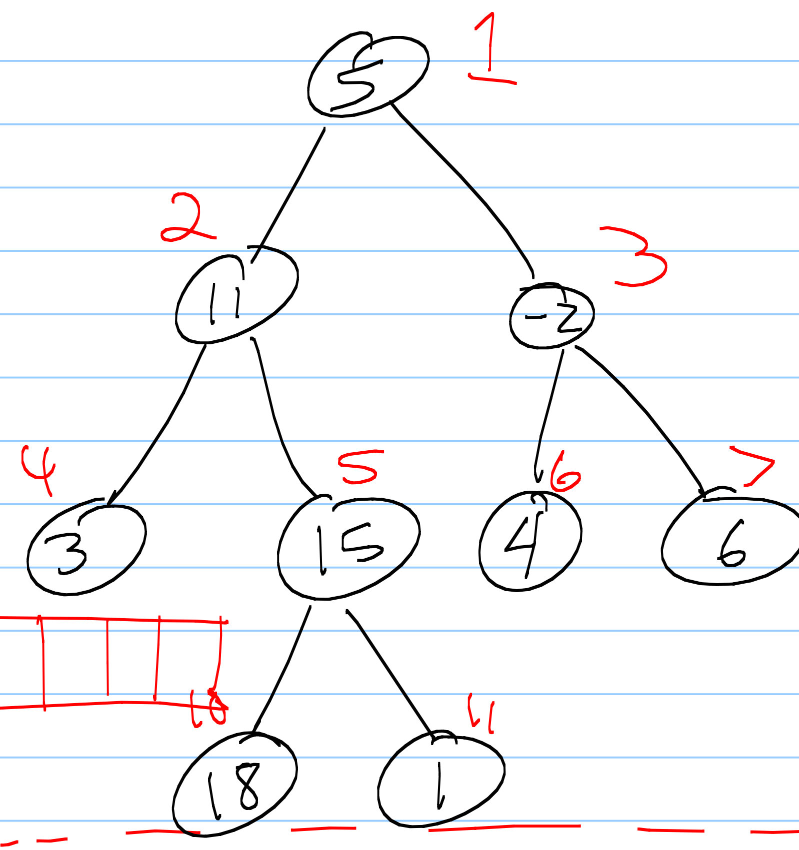
Here, every node has 0 or 2 children.



Array Based implementation:

Root is #~~1~~ 0

For any node v with number n, left child gets number $2 \cdot n$ and right child get $2 \cdot n + 1$



0														
5	11	-2	3	15	4	6			18	1				
							n							18

Each array will have size
& max capacity.
You have to double array if a
new level is added / ~~to~~ free.

Depth of a tree

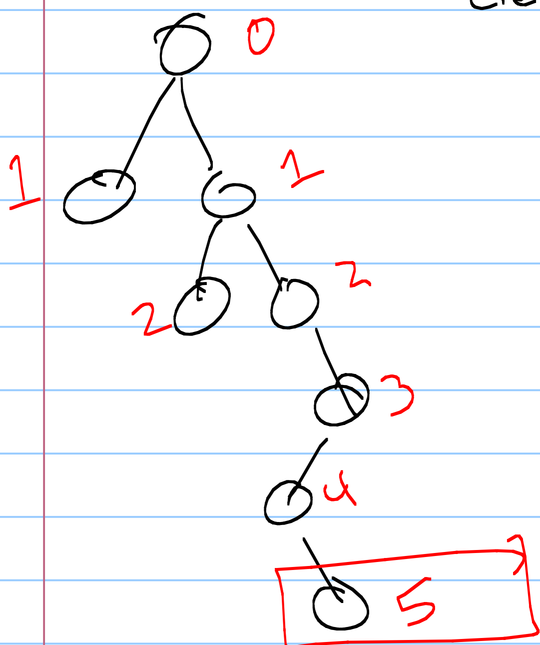
depth:

defined recursively:

root has depth 0

every other node:

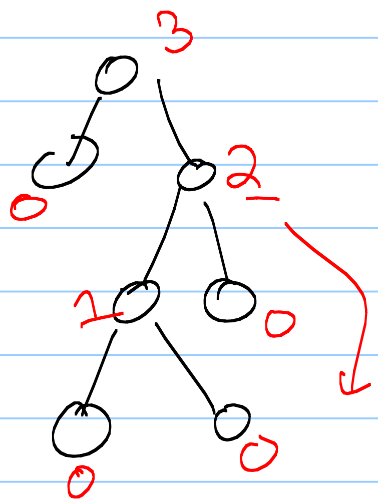
depth(v) = depth (parent(v)) + 1



(Easy to give recursive algorithm.)

O(depth of tree)

Height of a tree



Height of a leaf $= 0$

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

leads to recursive alg.

How long?

$O\left(\text{size of subtree rooted at } v\right)$