Intro to Trees

or, doing even better than linear time
Linked Lists Are Linear Access Time

- Usually pretty good.

- But consider terabytes of data, like from daily satellite images. $2^{40}$ (or $\sim 10^{12}$)
  $$1,099,511,627,776$$

- If looking for a particular bit of information, the linked list is going to be mighty slow.
Enter Trees… (stage right)

- Growth rate is $O(\log N)$ on average.

- Wow, that means
  - $N = 1,099,511,627,776$ versus
  - $\log N = 12$

- Now which one is faster? And by a LOT!!!!!!!!

Tell the truth now… Did you realize that $\log N$ is *that* much faster?
A Typical Tree
(just upside down)

A

B

C

D

E

F

G

H

I

J

K

L

M

N

node

parent of L

child of G

root (of tree)

data (stored in node)
sibling of H and J
(has same parent)

leafs
Paths

- A **path** between A and B is any chain of tree nodes that gets you from A to B.
  - A path can only follow from parents to children, and vice versa. But no siblings or cousins or aunts or uncles…
  - Must be in one direction up or down the tree. Can’t be both.

- The **length of a path** is the number of connecting edges. i.e., the number of nodes in the path – 1.

- The **depth** of a node is the length of the path from the node to the tree’s root.

- **Height** of a node is the length of the longest path from the node to the farthest leaf (among its children).
Path Example (etc.)

- Path from D to L is “D, G, L”
- Length of Path from D to L is 2.
- Path from L to J? None! (Can’t go up and down the tree.)
- Depth of L is 3.
- Depth E is 1.
- Height of A is 3.
- Height of G is 1.
- Height of M is 0.
How Can We Define a Tree?

- Recursively! (groan… No really, it’s simple!)

Each of these triangles represents another tree.

The “base case” or “seed”.

root

subtree $T_1$

subtree $T_2$

subtree $T_3$
Subtree $T_1$
Tree Recursion

- So each subtree has a new “root” node.
- If expand out all of the subtrees, and subtrees of subtrees (etc.) then get the full tree.
Implementation of Trees

• Option 1: Make each parent hold a link to it’s children
  • Impractical! Could have 10 million children of one node and 2 children of another.
    – Too much wasted space if assume many children for each node.
    – Only works well when have small number of children per node.
      » We’ll come back to this when restrict to two children later.

• Option 2: Keep children in a linked list.
  • Each node has two pointers
    – points to it’s first child
    – and points to it’s next sibling.
      » So sibling #1 would point to sibling #2, etc.
Tree With Linked List Siblings

A --> B, C, D
A --> E, F
A --> G, H, I, J
A --> K

B --> C, D
B --> E, F
B --> G, H, I, J
B --> K

C --> D
C --> E, F
C --> G, H, I, J
C --> K

D --> E, F
D --> G, H, I, J
D --> K

E --> F
E --> G, H, I, J
E --> K

F --> G, H, I, J
F --> K

G --> H, I, J
G --> K

H --> I, J
H --> K

I --> J
I --> K

J --> K

this becomes this
Code for Linked List Sibling Tree

/* in Java */

```java
public class TreeNode {
    int value;
    TreeNode firstChild;
    TreeNode nextSibling;
}
```

Students of OO will prefer to add a constructor. That’s a good idea!

/* in C */

```c
typedef struct TreeNode *PtrToNode;

struct TreeNode {
    int value;
    TreeNode firstChild;
    TreeNode nextSibling;
}
```

Note “recursion”, just like we visualized defining the tree.
Using the TreeNode Code

- Easy to create a tree. Start “backwards” with node C, then B, then A.

```java
public static void main(String[] args) {
    TreeNode nodeC = new TreeNode();
    nodeC.value = 2;
    nodeC.firstChild = null;
    nodeC.nextSibling = null;

    TreeNode nodeB = new TreeNode();
    nodeB.value = 2;
    nodeB.firstChild = null;
    nodeB.nextSibling = nodeC;

    TreeNode nodeA = new TreeNode();
    nodeA.value = 1;
    nodeA.firstChild = nodeB;
    nodeA.nextSibling = null;
}
```

The code creates this tree.

And the code uses the linked list implementation.
Pre vs. Postorder Traversals

Consider example application: file system.
- How do you print all the file names?
- In what order do you “traverse” the tree?
- With linked lists, etc. The answer was obvious. Here there are many choices!
Preorder Traversals

- Option 1 (of many possibilities):
  1. *First* print the node.
  2. *Then* print the left subtree.
  3. *Then* print the next subtree to the right.
  4. *Continue* left to right through all of the subtrees.

- Called preorder traversal because the work on the node (in this case printing) *precedes* the work on the subtrees.

C
  file1.c
Winnt
  file2.c
  file3.c
Java
  file4.c
MyFolder
  file5.
  Stuff
  file6.c
  file7.c
Postorder Traversals

- Option 2 (of many possibilities):
  1. *First* print the left subtree.
  2. *Then* print the next subtree to the right.
  3. *Continue* left to right through all of the subtrees.
  4. *Then at the very end*, print the node.

- Called postorder traversal because the work on the node (in this case printing) happens *after* the work on the subtrees.
Can you move one line of code and make this a postorder print?
Example: Railroad Yard

What’s stored at the nodes? Switch information of course! (switch points left or right)

Nodes
(switches on the train track)

And on the leafs can put stacks (of train cars).
Example: Blood Vessels

What data do we store? Percentage of vessels that are blocked upstream. Blood flow (i.e., flux). Etc.
Example: Rivers

What do we store on the nodes? Total flux of water, other stream flow data…
Example: Glaciers

• I proved with a tree structure that glaciers all have a similar fractal tree design! [Show cool picture.]

• What did I store on each node?
  – The volume of ice upstream of the node.
  – The surface area of ice upstream of the node.
  – Other physical properties…

• Who cares? Helps predict sea-level rise!
Next: Binary Trees

• Notice something? Many of the most useful trees have just two children for one parent.
  • Restricted to two children per parent.

• Next: let’s do binary trees.
  • 0, 1, or 2 children only. Never more.