04/18/2011
*what we've done
- linked list (chain, circular, doubly)
- sparse matrix with linked list
- equivalence class with linked list
- Reading Assignments:
  more on list operations
  more on sparse matrix
(Sec 5.1) Trees.

array (indexed access) stack/queue (restricted access)

list (sequential access)
trees: hierarchical access

* similarly, directories/files in your filesystem

* formal definition:

$$T = (\text{root}, T_1, T_2, \ldots, T_n)$$

note: recursive definition

Smallest tree

$$T^{(0)} = (\text{root})$$

disjoint: no cross links

* ancestors of $$K$$

degree of node: # of children; degree of tree: max degree of any node
* representing trees
  by definition
  \[ T = (\text{root}) \ T_1, T_2, \ldots, T_n \]

```c
struct Tree {
    Data root;
    struct Tree * subtrees[MAXDEG];
};
```

- wasted

* how about dynamic array for \( T_1, \ldots, T_n \)
  
  add
  ```
  int n_subtree;
  struct Tree * subtrees[];
  ```
  - more complicated allocations
  - still hard to insert/delete

* how about linked list for \( T_1, \ldots, T_n \)
  
  struct Tree {
      Data root;
      struct Tree * child();
      struct Tree * sibling();
  };
  
  called left-child, right-sibling
* how about call child by left sibling by right

A

B

E

F

C

G

D

... a binary tree that equivalently represents the original tree

* task: what's the binary tree representation of the binary tree
* Binary Tree

  tree w/ = 2 binary subtrees
  and ordered (left/right)
  that can be empty

  ![Diagram of binary tree]

(usually don't draw ꞌ ꞌ)

* how many nodes/edges?

  \( n_0 = 0 \) child (leaf)
  \( n_1 = 1 \) child
  \( n_2 = 2 \) child

\[
\begin{align*}
\# \text{node} &= n_0 + n_1 + n_2 \\
\# \text{edge} &= n_1 + 2n_2 = \# \text{node} - 1
\end{align*}
\]

\[ \Rightarrow n_2 = n_0 - 1 \]

* max # of nodes for depth-\( k \) tree? \( 2^k - 1 \) (full bin. tree)

  min # ? \( k \)

* binary tree representation

  ![Diagram of binary tree representation]

  * can get whole tree from root
  * tracing back?

    Can add 'parent' link

    (like doubly linked)
* full binary tree

```
   1
  /\  /
 /  \ /  \
/    /    \
2    3
```

Node #1 in full binary tree in binary code:

```
1 [path code]
```

- can "pack" the tree into an array

```
input  output  output
  ↓      ↓      ↓
append 0  append 1
```

* Binary Expression Tree

```
3 + 5  =⇒  
   +  
  / \  
 /   \ 
/     \
3     5
```

```
3 * 5 + 7  =⇒  
   +  
  / \  
 /   \ 
/     \
3     (5+7)
```

Infix Print (T) {
    Infix Print (T→left);
    output (T→data) j;
    Infix Print (T→right);
}

Postfix (T) {
    Postfix (T→left);
    Postfix (T→right);
    output (T→data) j;
}

Prefix (T) {
    output (T→data) j;
    Prefix (T→left);
    Prefix (T→right);
}
* more generally (on any binary tree)

In-order Traversal
Post-order Traversal
Pre-order Traversal

In: DBAECGFIH
post: DBEGIHFCA
pre: ABDCFEHGFI