Tree Overview

- Tree: recursive data structure (similar to list)
  - Each cell may have zero or more successors (children)
  - Each cell has exactly one predecessor (parent) except the root, which has none
  - All cells are reachable from root
- Binary tree: tree in which each cell can have at most two children: a left child and a right child

Tree Terminology

- M is the root of this tree
- G is the root of the left subtree of M
- B, H, J, N, and S are leaves
- P is the parent of N
- M and G are ancestors of D
- P, N, S are descendants of W
- Node J is at depth 2 (i.e., depth = length of path from root = number of edges)
- Node W is at height 2 (i.e., height = length of longest path to a leaf)
- A collection of several trees is called a ...?

Class for Binary Tree Cells

```java
public class TreeCell<T> {
    private T datum;
    private TreeCell<T> left, right;
    public TreeCell(T x) { datum = x; }
    public TreeCell(T x, TreeCell<T> lft, TreeCell<T> rgt) {
        datum = x;
        left = lft;
        right = rgt;
    }
    more methods: getDatum, setDatum, getLeft, setLeft, getRight, setRight
}
```

... new TreeCell<String>("hello") ...

Class for General Tree nodes

```java
public class GTreeCell {
    private Object datum;
    private GTreeCell left;
    private GTreeCell sibling;
    appropriate getter and setter methods
}
```

Binary versus general tree

- In a binary tree each node has exactly two pointers: to the left subtree, and to the right one
  - Of course one or both could be null
- In a general tree a node can have any number of child nodes
  - Very useful in some situations...
  - ... one of which will be our assignments!
Applications of Trees

- Most languages (natural and computer) have a recursive, hierarchical structure
- This structure is implicit in ordinary textual representation
- Recursive structure can be made explicit by representing sentences in the language as trees: Abstract Syntax Trees (ASTs)
- ASTs are easier to optimize, generate code from, etc. than textual representation
- A parser converts textual representations to AST

Example

- Expression grammar:
  - $E \rightarrow \text{integer}$
  - $E \rightarrow (E + E)$
- In textual representation:
  - Parentheses show hierarchical structure
- In tree representation:
  - Hierarchy is explicit in the structure of the tree

Searching in a Binary Tree

- Analog of linear search in lists: given tree and an object, find out if object is stored in tree
- Easy to write recursively, harder to write iteratively

Binary Search Tree (BST)

- If the tree data are ordered -- in any subtree,
  - All left descendants of node come before node
  - All right descendants of node come after node
  - This makes it much faster to search

Building a BST

- To insert a new item:
  - Pretend to look for the item
  - Put the new node in the place where you fall off the tree
  - This can be done using either recursion or iteration
- Example:
  - Tree uses alphabetical order
  - Months appear for insertion in calendar order
What Can Go Wrong?

- A BST makes searches very fast, unless...
  - Nodes are inserted in alphabetical order
  - In this case, we're basically building a linked list (with some extra wasted space for the left fields that aren't being used)
- BST works great if data arrives in random order

Printing Contents of BST

Because of the ordering rules for a BST, it's easy to print the items in alphabetical order

- Recursively print everything in the left subtree
- Print the node
- Recursively print everything in the right subtree

Tree Traversals

- "Walking" over the whole tree is a tree traversal
  - This is done often enough that there are standard names
  - The previous example is an inorder traversal
    - Process left subtree
    - Process node
    - Process right subtree
  - Note: we're using this for printing, but any kind of processing can be done
  - There are other standard kinds of traversals
    - Preorder traversal
      - Process node
      - Process left subtree
      - Process right subtree
    - Postorder traversal
      - Process left subtree
      - Process right subtree
      - Process node
    - Level-order traversal
      - Not recursive
      - Uses a queue

Some Useful Methods

- Determine if a node is a leaf
  ```java
  public static boolean isLeaf(TreeCell node) {
    return (node != null) && (node.left == null)
          && (node.right == null);
  }
  ```
- Compute height of tree using postorder traversal
  ```java
  public static int height(TreeCell node) {
    if (node == null) return -1; //empty tree
    if (isLeaf(node)) return 0;
    return 1 + Math.max(height(node.left),
                        height(node.right));
  }
  ```
- Compute number of nodes using postorder traversal
  ```java
  public static int nNodes(TreeCell node) {
    if (node == null) return 0;
    return 1 + nNodes(node.left) + nNodes(node.right);
  }
  ```

Useful Facts about Binary Trees

- \(2^d\) = maximum number of nodes at depth \(d\)
- If height of tree is \(h\)
  - Minimum number of nodes in tree = \(h + 1\)
  - Maximum number of nodes in tree = \(2^h + 2^{h-1} + ... + 2 + 1\)
- Complete binary tree
  - All levels of tree down to a certain depth are completely filled

Tree with Parent Pointers

- In some applications, it is useful to have trees in which nodes can reference their parents
- Analog of doubly-linked lists
Things to Think About

- What if we want to delete data from a BST?
- A BST works great as long as it's balanced
  - How can we keep it balanced?

Suffix Trees

- Given a string s, a suffix tree for s is a tree such that
  - each edge has a unique label, which is a nonnull substring of s
  - any two edges out of the same node have labels beginning with different characters
  - the labels along any path from the root to a leaf concatenate together to give a suffix of s
  - all suffixes are represented by some path
  - the leaf of the path is labeled with the index of the first character of the suffix in s
  - Suffix trees can be constructed in linear time

Suffix Trees

- Useful in string matching algorithms (e.g., longest common substring of 2 strings)
- Most algorithms linear time
- Used in genomics (human genome is ~4GB)

Huffman Trees

- Fixed length encoding
  \[197^2 + 63^2 + 40^2 + 26^2 = 652\]
- Huffman encoding
  \[197^1 + 63^2 + 40^3 + 26^3 = 521\]
BSP Trees

- BSP = Binary Space Partition
- Used to render 3D images composed of polygons
- Each node \( n \) has one polygon \( p \) as data
- Left subtree of \( n \) contains all polygons on one side of \( p \)
- Right subtree of \( n \) contains all polygons on the other side of \( p \)
- Order of traversal determines occlusion!

Tree Summary

- A tree is a recursive data structure
  - Each cell has 0 or more successors (children)
  - Each cell except the root has at exactly one predecessor (parent)
  - All cells are reachable from the root
  - A cell with no children is called a leaf
- Special case: binary tree
  - Binary tree cells have a left and a right child
  - Either or both children can be null
- Trees are useful for exposing the recursive structure of natural language and computer programs