Data structures so far

- Linked list
- Arrays/vectors
- Stacks
- Queues

Hierarchical data structures

- Allow for organization in more than one dimension
- Allow for hierarchical organization
- Benefits:
  - more natural form of organization
  - can help facilitate search process

Limitations of these data structures

- Array: sequential data storage
- Linked list: non-sequential data storage, but linear
- Stack: some hierarchical organization, but one-dimensional
- Queue: some hierarchical organization, but one-dimensional

Trees

October 6, 2003
Tree

- Hierarchical data structure
- Consists of:
  - nodes: data members
  - arcs: connect the nodes (separate the levels)
  - paths: the way to reach one node from another node

Tree: definitions

- **root**: the top node in a tree
- **leaves**: the bottom-most nodes in a tree
- **path length**: number of arcs in a path
- **level**: the path length from the root to a node, plus 1
  - also: the number of nodes in the path
- **height**: maximum level of a node in a tree

Tree: definitions

- **parent node**: the node above a given node in a tree
  - root node has no parent
- **child node(s)**: the node(s) below a given node in a tree
  - leaves have no children

Tree: formal definition

1. An empty structure is an empty tree.
2. If $t_1, \ldots, t_k$ are disjoint trees, then the structure whose root has as its children the roots of $t_1, \ldots, t_k$ is also a tree.
3. Only structures generated by rules 1 and 2 are trees.
Examples of trees

- Family trees
- Taxonomies
- Organization charts
- Database systems

What we'll cover

- Binary trees
- AVL trees
- Heaps
- B-Trees
- Tries

Binary trees

- Definition: a tree where each node has at most 2 children
- Each child node is a left child or a right child node
- There are at most $2^i$ nodes at level $i+1$ in a binary tree

Binary trees: definitions

- complete binary tree: all nodes (except for the leaves) have exactly 2 children
- decision tree: all nodes (except for the leaves) have either zero or 2 children
- binary search tree: a binary tree in which the values stored in all of the left subnodes of a node $n$ are less than the value stored at node $n$, and the values stored in all of the right subnodes of $n$ are greater than the value stored at node $n$
- a.k.a. ordered binary tree
- all values in the tree are unique
Implementations of binary trees

- **Array**
  - Each node contains:
    - data value
    - index of left child (-1 for null)
    - index of right child (-1 for null)
  - Root in first cell
- **Linked structure**
  - Each node contains:
    - data value
    - reference to left child
    - reference to right child

Binary tree node

```java
public class BinaryTreeNode {
    protected Comparable key;
    protected BinaryTreeNode left, right;
    public BinaryTreeNode(Comparable n) {
        this(n, null, null);
    }
    public BinaryTreeNode(Comparable n, BinaryTreeNode l, BinaryTreeNode r) {
        key = n;
        left = l;
        right = r;
    }
    public void visit() {
        System.out.print(key.toString() + " ");
    }
}
```

Binary tree

```java
public class BinaryTree {
    protected BinaryTreeNode root;
    public BinaryTree() {
        root = null;
    }
    // search methods
    // visiting node methods
    // insert node
    // delete node
}
```

Notes

- Need to use `Comparable` rather than `Object` so that we can compare two objects in a tree
  - `Comparable` is an interface
- To *visit* a node means to go to that node in the tree
- Operations on binary trees:
  - searching
  - traversal (visiting every node in a tree)
  - inserting a node into the tree
  - deleting a node from the tree
Searching a binary tree

```java
public BinaryTreeNode search(BinaryTreeNode n, Comparable obj) {
    int comparison;
    while (n != null) {
        comparison = obj.compareTo(n.key);
        switch (comparison) {
            case -1: // obj less than node key
                n = n.left;
                break;
            case  0: // obj equals node key
                return n;
                break;
            case  1: // obj greater than node key
                n = n.right;
                break;
        }
    }
    return null;
}
```

Complexity of searching a binary tree

- Best case: find item in root
- Worst case: not in tree, or in one of the leaves
- Average case: ???