More on Lists & Trees

Lecture 9
CS211 – Fall 2005

Announcements

- Section 6 (Wednesdays at 2:30 in Upson B17) is switching rooms
  - New room is HO 110
  - Course website has been updated

A List Interface

```
public interface List {

   public void insert (Object element);

   public void delete (Object element);

   public boolean contains (Object element);

   public int size ();
}
```

- The interface specifies the methods without saying anything about the implementation
  - Matches idea of ADT (Abstract Data Type)
- Any class that *implements* List can be stored in a variable of type List

An Array Implementation of List

```
public class ArrayList implements List {

   private Object[] theArray;
   private int empty;

   public ArrayList (int maxSize) {
      theArray = new Object[maxSize];
      empty = 0;
   }

   public void insert (Object element) {
      theArray[empty++] = element;
   }

   public int size () {
      return empty;
   }

public void delete (Object element) {
   for (int i = 0; i < empty; i++) {
      if (theArray[i].equals(element)) {
         for (int j = i+1; j < empty; j++)
            theArray[j-1] = theArray[j];
         empty--;
         break;
      }
   }
}

   public boolean contains (Object element) {
      for (int i = 0; i < empty; i++)
         if (theArray[i].equals(element)) return true;
      return false;
   }
}
```

ListCell

```
class ListCell {

   public Object datum;     // Data for this cell
   public ListCell next;    // Next cell

   public ListCell (Object datum, ListCell next) {
      this.datum = datum;
      this.next = next;
   }
}
```

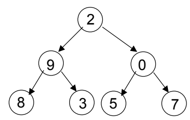
Linked-List Implementation of List

```
public class LinkedList implements List {

   ListCell head;

   public LinkedList () {
      head = null;
   }
   public void insert (Object element) {
      head = new ListCell(element, head);
   }
   public int size () {
      return size (head);
   }
   private static int size (ListCell cell) {
      if (cell == null) return 0;
      return 1 + size(cell.next);
   }

public void delete (Object element) {
      head = delete(element, head);
   }
private static ListCell delete (Object element, ListCell cell) {
      if (cell == null) return null;
      if (cell.datum.equals(element)) return cell.next;
      cell.next = delete(element, cell.next);
      return cell;
   }
public boolean contains (Object element) {
      return contains(element, head);
   }
private static boolean contains (Object element, ListCell cell)
   {
      if (cell == null) return false;
      if (cell.datum.equals(element)) return true;
      return contains(element, cell.next);
   }
}
```

Searching in a Binary Tree

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

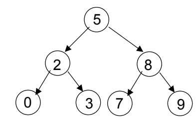
```
public static boolean treeSearch (Object x, TreeNode node) {
    if (node  == null) return false;
    return node.datum.equals(x) ||
                treeSearch(x, node.lchild) ||
                treeSearch(x, node.rchild);
}
```



Binary Search Tree (BST)

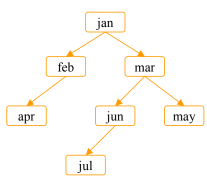
- Idea: tree nodes are ordered
  - All *left* descendents come *before* node
  - All *right* descendents come *after* node
- This makes it *much* faster to search

```
public static boolean treeSearch (Object x, TreeNode node) {
    if (node  == null) return false;
    if (node.datum.equals(x)) return true;
    if (node.datum.compareTo(x) < 0)
        return treeSearch(x, node.lchild);
    return treeSearch(x, node.rchild);
}
```



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
  - We insert months in calendar order
    - This way the months are in “random” order alphabetically



TreeNode

- This version is for a tree of Strings

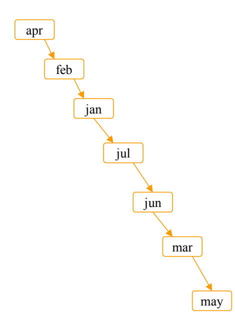
```
class TreeNode {

    String datum;              // Data for a node
    TreeNode lchild, rchild;     // Left and right children.

    public TreeNode (String datum) {
        this.datum = datum;
        lchild = null; rchild = null;
    }
}
```

BST Code

```
public class BST {
                              public void insert (String string) {
    private TreeNode root;         root = insert(string, root);
                                  }
    public BST () {
        root = null;          private static TreeNode insert (String string, TreeNode node) {
    }                                if (node == null) return new TreeNode(string);
                                     int compare = string.compareTo(node.datum);
                                     if (compare < 0) node.lchild = insert(string, node.lchild);
                                     else if (compare > 0) node.rchild = insert(string, node.rchild);
                                     return node;
                                  }
                              }
```

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 lchild 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

```
public void show () {
    show(root); System.out.println();
  }

private static void show (TreeNode node) {
    if (node == null) return;
    show(node.lchild);
    System.out.print(node.datum + " ");
    show(node.rchild);
  }
```

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

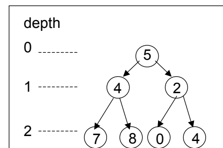
Some Useful Methods

```
// Determine if a TreeNode is a leaf node
public static boolean isLeaf (TreeNode node) {
   return (node != null) && (node.lchild == null) && (node.rchild == null);
}

// Compute height of tree using postorder traversal
public static int height (TreeNode node) {
    if (node == null) return -1; // Height is undefined for empty tree
    if (isLeaf(node)) return 0;
    return 1 + Math.max(height(node.lchild), height(node.rchild));
}

// Compute number of nodes in tree using postorder traversal
public static int nNodes (TreeNode node) {
   if (node == null) return 0;
   return 1 + nNodes(node.lchild) + nNodes(node.rchild);
}
```

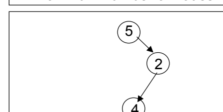
Useful Facts about Binary Trees

- Maximum number of nodes at depth d = 2^d
- If height of tree is h
  - Minimum number of nodes it can have = h+1
  - Maximum number of nodes it can have =

    $2^0 + 2^1 + \ldots + 2^h = 2^{h+1} - 1$
- *Full binary tree* (or *complete binary tree*)
  - All levels of tree are completely filled



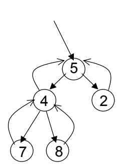
Height 2,
maximum number of nodes



Height 2,
minimum number of nodes

Tree with Parent Pointers

- In some applications, it is useful to have trees in which nodes can reference their parents
  - Tree 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?