CS 206
Introduction to Computer Science II

03 / 10 / 2016

Instructor: Michael Eckmann
Today’s Topics

• Questions? Comments?
• Review Recursion
• Binary Search trees
  – Review BSTs, search, insert
  – Code for these operations
  – Delete operation
Recursion

• 1. have at least one base case that is not recursive
• 2. recursive case(s) must progress towards the base case
• 3. trust that your recursive call does what it says it will do (without having to unravel all the recursion in your head.)
• 4. try not to do redundant work. That is, in different recursive calls, don't recalculate the same info.
Recursion

• some definitions are inherently recursive
• e.g.
• sum of the first \( n \) integers, \( n \geq 1 \).
• \( S(1) = 1 \)
• \( S(N) = S(N-1) + N \)

• recursive code for this
• iterative code for this
• simpler code for this -> \( N^*(N+1)/2 \)
• any problems with the recursive code? \( n=0 \) or large \( n \)?
Recursion

The last example showed that recursion didn't really simplify our lives, but there are times when it does.

e.g. If given an integer and you wanted to print the individual digits in order, but you didn't have the ability to easily convert an int >10 to a String.

e.g. int n=35672;

If we wanted to print 3 first then 5 then 6 then 7 then 2, we need to come up with a way to extract those digits via some mathematical computation.

It's easy to get the last digit n%10 gives us that.

Notice: 35672 % 10 = 2 also 2 % 10 = 2.

Any ideas on a recursive way to display all the numbers in order?
void printDigits(int n) {
    if (n>=10) {
        printDigits((n/10));
        System.out.println( n%10 );
    }
}

// what's the base case here?

// what's the recursive step here? Will it always approach the base case?
Recursion

• Now that last problem was “made up”, because Java (and most languages) allow us to print ints.
• However what if we wanted to print the int in a different base? Say base 2, 3, 4, 5, or some other base?
Recursion

• The fibonacci sequence is:
  0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

– Can you detect the pattern?
Recursion

• The fibonacci sequence is:
  0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

• Fibonacci numbers are simply defined recursively:
  \[ F_0 = 0 \]
  \[ F_1 = 1 \]
  \[ F_n = F_{n-1} + F_{n-2} \]

• How could we convert this to code to calculate the \( i \)th fibonacci number?
Recursion

• Fibonacci numbers are simply defined recursively:
  \[ F_0 = 0 \]
  \[ F_1 = 1 \]
  \[ F_n = F_{n-1} + F_{n-2} \]

• How could we convert this to code to calculate the \(i^{th}\) fibonacci number?
  ```
  int fib(int i)
  {
      if (i <= 1)
          return i;
      else
          return ( fib(i-1) + fib(i-2) );
  }
  ```
Recursion

```c
int fib(int i)
{
    if (i <= 1)
        return i;
    else
        return ( fib(i-1) + fib(i-2) );
}
```

Any problems with this code?
Recursion

int fib(int i)
{
    if (i <= 1)
        return i;
    else
        return (fib(i-1) + fib(i-2));
}

Any problems with this code?
Yes – it makes too many calls. And further, these calls are redundant.
It violates that 4th idea of recursion stated earlier: in different recursive calls, don't recalculate the same info.
Recursion

• We know what a tree is.
• Can anyone give a recursive definition of a tree?
Recursion

• We know what a tree is.

• Can anyone give a recursive definition of a tree?
  – A tree is empty or it has a root connected to 0 or more subtrees.
  – (Note a subtree, taken on its own, is a tree.)
Binary Search Trees

• Binary Search Trees (BSTs)
  – Definition
  – Example of a BST and verification that it is one
  – Are BST's unique for a given set of data?
  – Algorithms
    • print keys in ascending order
    • Search for a key
    • Find minimum key
    • Find maximum key
    • Insert a key
Binary Search Trees

• Binary Search Trees (BSTs)
  – A *tree* that is both a *binary tree* and a *search tree*.
  – we know the definition of a tree and a binary tree so:
  – We just need to define search tree.

• A *search tree* is a tree where
  – every subtree of a node has data (aka keys) less than any other subtree of the node to its right.
  – the keys in a node are conceptually between subtrees and are greater than any keys in subtrees to its left and less than any keys in subtrees to its right.
Binary Search Trees

• A *binary search tree* is a tree that is a binary tree and is a search tree. In other words it is a tree that has
  – at most two children for each node and
  – every node's left subtree has keys less than the node's key, and every right subtree has keys greater than the node's key.

– Definitions taken from http://www.nist.gov/ (The National Institute of Standards and Technology)
Binary Search Trees

- Do you see any recursion in BSTs?
Binary Search Trees

• Do you see any recursion in BSTs?
  – Every node itself is a BST
Binary Search Trees

• Each node in a BST has
  – key
  – left reference
  – right reference
public class BSTNode
{
    public int key;
    public BSTNode left;
    public BSTNode right;

    public BSTNode(int k)
    {
        key = k;
        left = null;
        right = null;
    }
}
Binary Search Trees

- Assume Alphabet ordering of letters.
- Let's verify whether it is indeed a binary search tree.
  - What properties does it need to have again?
  - Does it satisfy those properties?
Binary Search Trees

• Are BST's unique for a given set of keys?
• Let's build a tree for the list of keys
  – 13, 45, 10, 9, 54, 11, 42
  – Choose 13 to put in the root and go from there
Binary Search Trees

• Are BST's unique for a given set of keys?
• Let's build a tree for the list of keys
  – 13, 45, 10, 9, 54, 11, 42

  – What if we change the order that we insert the keys?
  – What if we choose a different key as the root to start?
Binary Search Trees

Let's look at algorithms to do the following

- Print keys in ascending order
- Search for a key
- Find minimum key
- Find maximum key
- Insert a key
- Delete a key
Print the Keys

To print the keys in increasing order we use inorder traversal. Recursive description of inorder traversal

In-order traversal of a tree
Start with x being the root

check if x is not null then
1) In-order traversal of left(x)
2) print key(x)
3) In-order traversal of right(x)

Let's apply this algorithm to the tree and see what it does.

What is the running time complexity of this algorithm for a tree that has n nodes?
Search for a key

• Any ideas on an iterative solution?
• First, let's assume this method lives in the BSTree class
• What parameter(s) will it take?
• What will it return?
Search for a key

• To search in a binary search tree for a key k, start with x being the root. Here's an iterative solution of the search

```java
public BSTNode searchKey(int searchkey)
{
    BSTNode temp = root;
    while (temp != null && temp.key != searchkey)
    {
        if (temp.key > searchkey)
            temp = temp.left;
        else
            temp = temp.right;
    }
    return temp;  // may be null or not (if null then key wasn't found)
}
```

What's the running time of this?
Search for a key

- Any ideas on a recursive solution?
- First, let's assume this method lives in the BSTree class
- What parameter(s) will it take?
- What will it return?
- Driver method necessary?
- Base case?
- Recursive step?
Search for a key

•To search in a binary search tree for a key k, start with x being the root. Here's a recursive solution of the search

```java
public BSTNode treeSearch(BSTNode temp, int k)
{
    if (temp == null || k == temp.key)
        return temp;
    if (k < temp.key)
        return treeSearch(temp.left, k)
    else
        return treeSearch(temp.right, k)
}
```

What's the running time of this?
Search for a key

• What's the running time of this?
  – On the order of the height of the tree.
• What if the binary search tree is complete (or full)?
Find Minimum key in a BST

- How might we devise an algorithm to find the minimum?
- Where in the tree is the minimum value?
Find Minimum key in a BST

• How might we devise an algorithm to find the minimum?
• Where in the tree is the minimum value?
  – It is in the leftmost node
  – Code to find it?
Find Minimum key in a BST

• How might we devise an algorithm to find the minimum?
• Where in the tree is the minimum value?
  – It is in the leftmost node

```java
while (temp.left != null)
    temp = temp.left;
return temp;
```
Find Maximum key in a BST

• How might we devise an algorithm to find the maximum?
• Where in the tree is the maximum value?
Find Maximum key in a BST

• How might we devise an algorithm to do find the maximum?
• Where in the tree is the maximum value?
  – It is in the rightmost node

```java
while (temp.right != null)
    temp = temp.right;

return temp;
```

• Running times of these?
Insert a key in a BST

• How might we devise an algorithm to insert a key into the tree?
• Can the key go anywhere?
Insert a key in a BST

• How might we devise an algorithm to insert a key into the tree?

• Can the key go anywhere? No, it has to follow the rules of BST's so the resulting tree after insert must be a BST.

• z is the node to insert and z.key is its key and its left and right are null.

• Need to keep track of where we are in the tree as we traverse it and the parent of where we are because we might have to go back up the tree.

• par will be a pointer to the parent of temp as we go through the tree
Insert a key in a BST

insert(BSTNode z) // insert node z in tree T
{
    BSTNode temp = root;
    BSTNode par = null;
    // search the tree to find the place it should go
    while (temp != null)
    {
        par = temp;
        if (z.key < temp.key)
            temp = temp.left;
        else
            temp = temp.right;
    }
    // continued on next slide
Insert a key in a BST

// now we know temp is null and the parent of temp is par

// insert the key either at the root or to the left or right of par
if (par == null)
    root = z;
else
    if (z.key < par.key)
        par.left = z;
    else
        par.right = z;

} // any fear of overwriting a node? What if par.left or par.right // contain a node?
Insert a key in a BST

// any fear of overwriting a node? What if par.left or par.right

// contain a node?

They can't contain a node because of the first half of the algorithm guarantees it

Examples:
Let's insert C into the original tree.
Then let's insert E.
Delete a key in a BST

- Let's do some examples of deleting from a BST
- Let's make some observations of different situations and how they can be handled
Delete a key in a BST

- Let's do some examples of deleting from a BST
  - Deleting a leaf is easy
    - Make parent of the delete node's (correct) child null
  - Deleting a node with only one child is easy
    - Make parent of the delete node's (correct) child be the delete node's non-null child
  - Deleting a node with two children
    - Find either the largest in the left subtree and put that in the delete node's place OR find the smallest in the right subtree and put that in the delete node's place
    - Then depending on which you replaced, delete the largest in the left subtree or the smallest in the right subtree
Delete a key in a BST

• Deleting a node with two children
  • In other words, find the rightmost node in the left subtree and replace deleteNode with it and delete it OR find the leftmost node in the right subtree and replace deleteNode with it and delete it.