Advanced Programming Techniques in Java

Recursion (cont.)

Lecture 21

Class Objectives

- More Recursion (Sections 12.1-12.3)

Backtracking (Section 12.5)

Review: Why learn recu

- "cultural experience" - A different way of thi
- Can solve some kinds of problems bet
- Leads to elegant, simplistic, short code

- Many programming languages ("functiona
as Scheme, ML, OCaml and Haskell) use
(no loops)

Review: Run-Time Stack Frames

- Javamaintains a run-time stack on
newinformation in the form of an
- Theactivation frame contains storage

- method arguments
- local variables (if any)
- the return address of the instruction

- Whenever a newmethod is called (re
  Javapushes a newactivation frame
  stack

Review: Recursion Versu

- There are similarities between recursion
- In iteration, a loop repetition co
  whether to repeat the loop body or
- In recursion, the condition usually tes
  case
- You can always write an iterative solutio
  that is solvable by recursion

- A recursive algorithm may be simpler algorithm and thus easier to write, read

Review: Efficiency ofRec

- Recursive methods often have slower relative to their iterativecounterparts

- The overhead for loop repetition is overhead for a method call and return
- If it is easier to conceptualize an alg recursion, then you should code it method
- The reduction in efficiency does not advantage of readable code thatis

Review: Design of a Binary

- A binary search can be performed thathas been sorted
- Base cases
  - The array is empty
  - Element being examined matches
- Rather than looking at the first ele search compares the middle elemen the target

- A binary search excludes the hal
  which the target cannot lie

Review: Testing Bin

- You should test arrays with
  - an even number of elements
  - an odd number of elements
  - duplicate elements

- Test each array for the followi
  - the target is the element at eac
    starting with the first position and end
  - the target is less than the smallest
  - the target is greater than the larg
  - the target is a value between
    the array

Recursive Data

Recursive Data Stru

- Computer scientists often encount are defined recursively – with ar a component
- Linked lists and trees can be defined a structures
- Recursive methods provide a natu processing recursive data struc

- The first language developed for artifi
  was a recursive language calle

Recursive Definition of a

- A linked list is a collection of n
  each node references another linke
  the nodes that follow it in the list
- The last node references an empty l

- A linked list is empty, or it cont
  called the list head, it stores data
  to a linked list

Class LinkedListRec

- We define a class LinkedLis
  list operations using recursive meth

```
public class LinkedListRec<E> {
  private Node<E> head;

  // inner class Node<E> here
  //
}
```

Recursive `size` Metho

```
/** Finds the size of a list.
    @param head The head of the current
    @return The size of the current list
*/
private int size(Node<E> head) {
    if (head == null)
        return 0;
    else
        return 1 + size(head.next);
}

/** Wrapper method for finding the size
    @return The size of the list
*/
public int size() {
    return size(head);
}
```

Recursive `toString`

```
/** Returns the string representation of
    @param head The head of the current
    @return The state of the current lis
*/
private String toString(Node<E> head) {
    if (head == null)
        return "";
    else
        return head.data + "\n" + toStri
}

/** Wrapper method for returning the str
    @return The string representation of
*/
public String toString() {
    return toString(head);
}
```

Recursive `replace` **Meth**

```
/** Replaces all occurrences of oldObj with newObj.
    post: Each occurrence of oldObj has been replac
    @param head The head of the current list
    @param oldObj The object being removed
    @param newObj The object being inserted
*/
private void replace(Node<E> head, E oldObj, E new(
    if (head != null) {
        if (oldObj.equals(head.data))
            head.data = newObj;
        replace(head.next, oldObj, newObj);
    }
}

/*  Wrapper method for replacing oldObj with newObj
    post: Each occurrence of oldObj has been replac
    @param oldObj The object being removed
    @param newObj The object being inserted
*/
public void replace(E oldObj, E newObj) {
    replace(head, oldObj, newObj);
}
```

Recursive `add` Meth

```java
/** Adds a new node to the end of a list.
    @param head The head of the current list
    @param data The data for the new node
*/
private void add(Node<E> head, E data) {
    // If the list has just one element, add
    if (head.next == null)
        head.next = new Node<E>(data);
    else
        add(head.next, data);      // Add to
}

/** Wrapper method for adding a new node to t
    @param data The data for the new node
*/
public void add(E data) {
    if (head == null)
        head = new Node<E>(data);  // List ha
    else
        add(head, data);
}
```

Recursive `remove` Metho

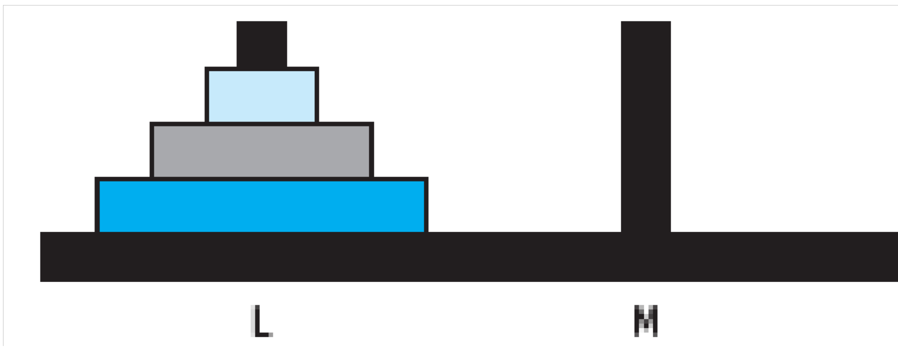
```
/** Removes a node from a list.
    post: The first occurrence of outData i
    @param head The head of the current lis
    @param pred The predecessor of the list
    @param outData The data to be removed
    @return true if the item is removed
            and false otherwise
*/
private boolean remove(Node<E> head, Node<E
    if (head == null)  // Base case - empty
        return false;
    else if (head.data.equals(outData)) {  /
        pred.next = head.next;  // Remove h
        return true;
    } else
        return remove(head.next, head, outD
}
```

Recursive `remove` Met

```
/** Wrapper method for removing a no
    post: The first occurrence of ou
    @param outData The data to be re
    @return true if the item is remo
            and false otherwise
*/
public boolean remove(E outData) {
    if (head == null)
        return false;
    else if (head.data.equals(outDat
        head = head.next;
        return true;
    } else
        return remove(head.next, hea
}
```

Problem Solving with
Simplified Towers of

- Move the three disks to a dif their order (largest disk top, etc.)
  - Only the top diskon a another peg
  - A larger diskcannot be placed smaller disk


L
M

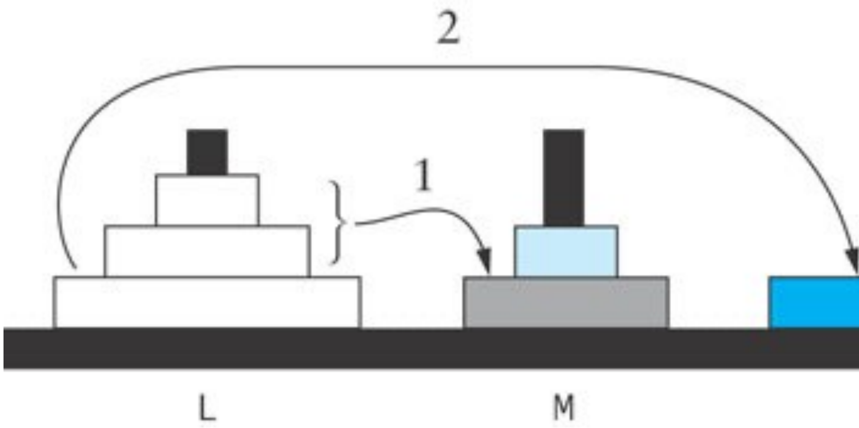
Towers ofHanoi

Problem Inputs
Number of disks (an integer)
Letter of starting peg: L (left), M (middle), or R
Letter of destination peg: (L, M, or R), but differe
Letter of temporary peg: (L, M, or R), but differe destination peg
Problem Outputs
A list of moves

Algorithm for Towers

Solution to Three-Disk Problem: Mo
from PegL to PegR

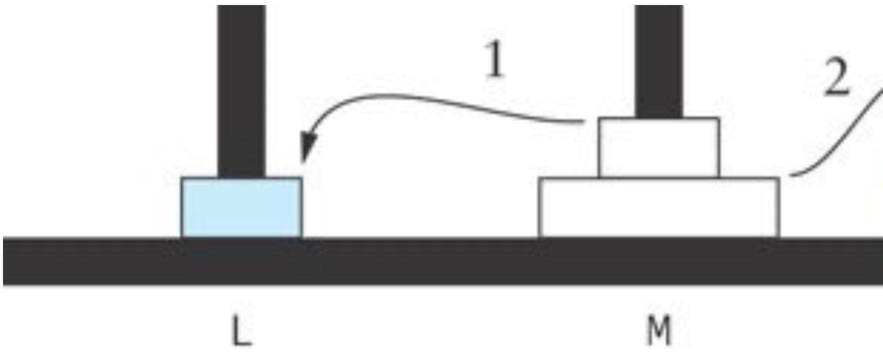
1. Move the top two disks from peg L
2. Move the bottom diskfrom peg L to
3. Move the top two disks from peg M



Algorithm for Towers

Solution to Two-Disk Problem: Move Peg R

1. Move the top disk from peg M to peg
2. Move the bottom disk from peg M to
3. Move the top disk from peg L to peg



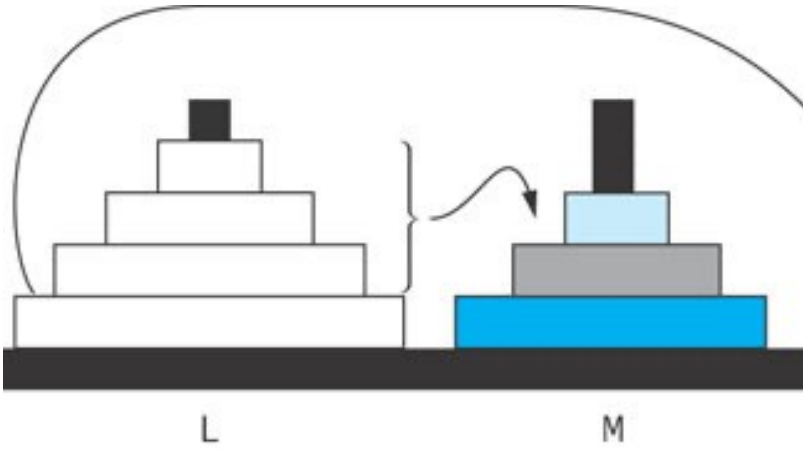
Algorithm for Towers

Solution to Four-Disk Problem: Mo
to Peg R

1. Move the top three disks from peg

2. Move the bottom disk from peg L to

3. Move the top three disks from peg



Recursive Algorithm for

Recursive Algorithm for n -Disk Problem:Mov
Peg to the Destination Peg `if` n is 1 move disk
the starting peg to the destination peg
`else`
move the top $n - 1$ disks from the starting
peg
(neither starting nor destination peg) move disk
disk at the bottom) from the starting peg to
destination peg
move the top $n - 1$ disks from the tempora
peg

Implementation of Recu
Hanoi

```java
/** Class that solves Towers of Hanoi problem.
public class TowersOfHanoi {
    /** Recursive method for "moving" disks.
        pre: startPeg, destPeg, tempPeg are di
        @param n is the number of disks
        @param startPeg is the starting peg
        @param destPeg is the destination peg
        @param tempPeg is the temporary peg
        @return A string with all the required
    */
    public static String showMoves(int n, char
                                   char destPeg
        if (n == 1) {
            return "Move disk 1 from peg " + st
                   " to peg " + destPeg + "\n"
        } else {  // Recursive step
            return showMoves(n - 1, startPeg, t
                   + "Move disk " + n + " from
                   + " to peg " + destPeg + "\n
                   + showMoves(n - 1, tempPeg,
        }
    }
}
```

Counting Cells ina

- Consider how we might process presented as a twodimensional values
- Information in the image may
  - an X-ray
  - an MRI
  - satellite imagery
  - etc.
- The goal is to determine the the image thatis considered abr its color values

Counting Cells in a Bl

- Given a two-dimensional grid of
  contains either a normal bac
  second color, which indicates
  an abnormality
- A *blob* is a collection of contigu
- A user will enter the x, y coo
  the blob, and the program will
  of all cells in that blob

Counting Cells ina Bl

- Problem Inputs
  - the two-dimensional grid of cell
  - the coordinates of a cell in a
- Problem Outputs
  - the count of cells in the blo

Counting Cells in a

Method	Behavior
void recolor(int x, int y, Color aColor)	Resets the colo
Color getColor(int x, int y)	Retrieves the c
int getNRows()	Returns the nu
int getNCols()	Returns the nu

Method	Behavior
int countCells(int x, int y)	Returns the nu

Counting Cells ina Bl
(cont.)

Algorithm for `countCells(x, y)`

```
if the cell at (x, y) is outside
      the grid the result is 0
else if the color of the cell at (x,
      abnormal color the result is 0
else
    set the color of the cell at (x,
      temporary color the result is 1 p
      cells in each piece of the blobt
      nearest neighbor
```

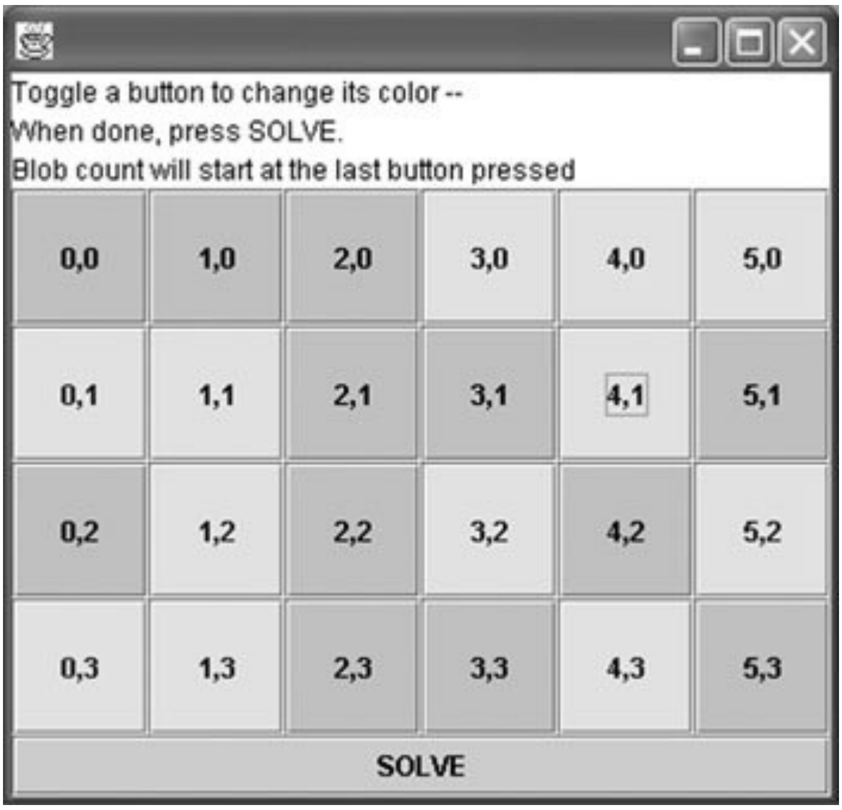
Counting Cells ina Blo

```
import java.awt.*;

/** Class that solves problem of countin
public class Blob implements GridColors

    /** The grid */
    private TwoDimGrid grid;

    /** Constructors */
    public Blob(TwoDimGrid grid) {
        this.grid = grid;
    }
```

Counting Cells ina Blo
(cont.)

```java
/** Finds the number of cells in the blob a
    pre: Abnormal cells are in ABNORMAL col
         Other cells are in BACKGROUND colo
    post: All cells in the blob are in the
    @param x The x-coordinate of a blob cel
    @param y The y-coordinate of a blob cel
    @return The number of cells in the blob
*/
public int countCells(int x, int y) {
    int result;

    if (x < 0 || x >= grid.getNCols()
            || y < 0 || y >= grid.getNRows(
        return 0;
    else if (!grid.getColor(x, y).equals(AB
        return 0;
    else {
        grid.recolor(x, y, TEMPORARY);
        return 1
            + countCells(x - 1, y + 1) + co
            + countCells(x + 1, y + 1) + co
            + countCells(x + 1, y) + countC
            + countCells(x, y - 1) + countC
    }
}
}
```

Counting Cells in a



Counting Cells ina Blo

- Verify thatthe code works for t
  - A starting cell that is on the edge
  - A starting cell that has no neighborin
  - A starting cell whose only abnormal connected to it
  - A "bull's-eye": a starting cell whose normal but their neighbors a
  - A starting cell that is normal
  - A grid that contains all abnormal c
  - A grid that contains all normal cells