CS 211: Recursion

Chris Kauffman

Week 13-1
Today

- P6 Questions
- Recursion, Stacks

Labs

- 13: Due today
- 14: Review and evals
- Incentive to attend lab 14, announce Tue/Wed

End Game

<table>
<thead>
<tr>
<th>Date</th>
<th>Day</th>
<th>Class Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>4/24</td>
<td>Mon</td>
<td>P6, Comparisons</td>
</tr>
<tr>
<td>4/26</td>
<td>Wed</td>
<td>Recursion</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lab 13 Recursion</td>
</tr>
<tr>
<td>5/1</td>
<td>Mon</td>
<td>Stacks/Queues</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lab 13 Due</td>
</tr>
<tr>
<td>5/3</td>
<td>Wed</td>
<td>Review/Evals</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lab 14 Review/Evals</td>
</tr>
<tr>
<td>5/7</td>
<td>Sun</td>
<td>P6 Due</td>
</tr>
<tr>
<td>Mon</td>
<td>5/15</td>
<td>Final Exams</td>
</tr>
<tr>
<td>002</td>
<td></td>
<td>10:30am-1:15pm</td>
</tr>
<tr>
<td>006</td>
<td></td>
<td>1:30pm-4:15pm</td>
</tr>
</tbody>
</table>
What are the built in search/sort routines in Java?
What classes are they in?
How can a new class be used with them?
How fast are these library routines?
- Linear search
- Binary search
- Sorting algorithm
Rabbits

A puzzle.\textsuperscript{1}

\textit{Consider the growth of an idealized (biologically unrealistic) rabbit population, assuming that:}

\begin{itemize}
  \item A newly born pair of rabbits, one male, one female, are put on an island;
  \item Rabbits are able to mate at the age of one month so that at the end of its second month a female can produce another pair of rabbits;
  \item Rabbits never die and a mating pair always produces one new pair (one male, one female) every month from the second month on.
\end{itemize}

How many pairs will there be in one year?

\textsuperscript{1}Adapted from Wikipedia
Write a program to simulate the rabbit population.

- First we should develop a general approach
- Look at some data for this
Tabularly

Mature pair produce baby pair the following month

BN  Baby pair from pair $N$

MN  Mature pair from pair $N$

<table>
<thead>
<tr>
<th>Month</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pairs</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>13</td>
</tr>
<tr>
<td>Pair 0</td>
<td>B1</td>
<td>M1</td>
<td>M1</td>
<td>M1</td>
<td>M1</td>
<td>M1</td>
<td>M1</td>
<td>M1</td>
</tr>
<tr>
<td>Pair 1</td>
<td>B0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
</tr>
<tr>
<td>Pair 2</td>
<td>B0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
</tr>
<tr>
<td>Pair 3</td>
<td>B0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
</tr>
<tr>
<td>Pair 4</td>
<td>B1</td>
<td>M1</td>
<td>M1</td>
<td>M1</td>
<td>M1</td>
<td>M1</td>
<td>M1</td>
<td>M1</td>
</tr>
<tr>
<td>Pair 5</td>
<td>B0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
</tr>
<tr>
<td>Pair 6</td>
<td>B1</td>
<td>M1</td>
<td>M1</td>
<td>M1</td>
<td>M1</td>
<td>M1</td>
<td>M1</td>
<td>M1</td>
</tr>
<tr>
<td>Pair 7</td>
<td>B2</td>
<td>M2</td>
<td>M2</td>
<td>M2</td>
<td>M2</td>
<td>M2</td>
<td>M2</td>
<td>M2</td>
</tr>
<tr>
<td>Pair 8</td>
<td>B0</td>
<td>B0</td>
<td>B0</td>
<td>B0</td>
<td>B0</td>
<td>B0</td>
<td>B0</td>
<td>B0</td>
</tr>
<tr>
<td>Pair 9</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
<td>B1</td>
</tr>
<tr>
<td>Pair 11</td>
<td>B3</td>
<td>B3</td>
<td>B3</td>
<td>B3</td>
<td>B3</td>
<td>B3</td>
<td>B3</td>
<td>B3</td>
</tr>
<tr>
<td>Pair 12</td>
<td>B4</td>
<td>B4</td>
<td>B4</td>
<td>B4</td>
<td>B4</td>
<td>B4</td>
<td>B4</td>
<td>B4</td>
</tr>
</tbody>
</table>
Pattern

How does the population of a month relate to previous months?
Recursively

Population for Month $i = \text{Pop. Month } i - 1 + \text{Pop. Month } i - 2$

Better known as *Fibonacci Numbers*:

$$f_0 = 0$$

$$f_1 = 1$$

$$f_i = f_{i-1} + f_{i-2}$$

```
public static int fib(int n)
```

- Recursive implementation?
- Iterative implementation?
- Call Stack behavior in each
Recursion is . . .

Something specified in terms of a smaller version of itself
Recursion involves

**Base Case**
The "smallest thing", where you can definitively say "here is the answer"

**Inductive/Recursive Case**
If I had the answer to a few smaller versions of this problem, I could combine them to get the answer to this problem.
Identify Base and Recursive Cases

Fibonacci

\[ f_0 = 0 \]
\[ f_1 = 1 \]
\[ f_i = f_{i-1} + f_{i-2} \]

Factorial

\[ \text{fact}(n) = n \times \text{fact}(n - 1) \]
\[ \text{fact}(0) = 1 \]
Examine Stack Trace for Fibonacci

Recursive

public static int fibR(int n)

▶ Recursive implementation
▶ View Stack Trace of fibR(4)

Iterative

public static int fibI(int n)

▶ Iterative implementation?
▶ View Stack Trace of fibI(4)

Point
Recursion utilizes the Stack to store information about history
Exercise: Show the stack trace of fib

```
public class Fib{
    static int CALLS = 0;
    public static void main(String args[]){
        int fn = fib(4);
        System.out.printf("%d %d\n",fn,CALLS);
    }
    public static int fib(int n){
        CALLS++;
        // Draw call stack here when CALLS==9
        if(n==0){ return 0; }
        if(n==1){ return 1; }
        else{
            int tmp1 = fib(n-1);
            int tmp2 = fib(n-2);
            return tmp1+tmp2;
        }
    }
}
```

- static var CALLS counts number times fib(n) is entered
- Show stack trace starting with fib(4)
- Show local vars n,tmp1,tmp2 in stack frames
- Stop when CALLS reaches 9
Other Uses for Recursion

Enumeration
Show me all possibilities of something
  ▶ All permutations of the numbers 1 to 10
  ▶ Print all games of Party Pong (hard problem from previous year)

Search Problems
Show me whether something exists and how its put together
  ▶ Does a number exist in an array?
  ▶ Does a path exist from point M to point C on a grid and what is that path?

```
M || || || || || || || || || C
```

```
Exercise: Sums

- Print all **permutations** of positive numbers which total 8 (order of numbers matters)
- Create a recursive helper called `totalsTarget()`
- Base and recursive cases?

Prototypes

```java
public static void sumsTo8(){..}

public static void totalsTarget(int target,
                                 int current,
                                 String history)
```

target: Eight!
current: current total
history: numbers used so far

Example output

```
> javac Sums.java
> java Sums
8 = 1 1 1 1 1 1 1 1
8 = 1 1 1 1 1 1 2
8 = 1 1 1 1 1 2 1
8 = 1 1 1 1 1 3
8 = 1 1 1 1 2 1 1
...
8 = 6 1 1
8 = 6 2
8 = 7 1
8 = 8
```

- 128 lines...
- Iterative version?
The "Power" of Recursion

Questions

▷ What problems can one solve with Recursion that *cannot* be solved with iteration (looping)
▷ Vice versa: loops can, recursion can’t?
Stacks and Stacks of...

- We will shortly examine a solution to the sums problem which does not use recursion.
- For that, we will need a data structure: a stack.
- Should be familiar at this point based on our discussions of function call stack.
Stacks

A data structure, supports a few operations

- T s.getTop(): return whatever is on top
- s.push(T x): put x on top
- void s.pop(): remove whatever is on top
- boolean s.isEmpty(): true when nothing is in it, false o/w

Stacks are a LIFO: Last In First Out

Questions

- Examples of stacks?
- How would you implement a stack using arrays?
Array Based Implementation of Stacks

- Must dynamically expand an internal array
- Following the textbook ArrayList implementation should make this easy
- Can check your work against java.util.Stack: should behave similarly

class AStack<T>{
    public AStack(); // Constructor
    public void push(T x); // Like add(x)
    public T pop(); // Like remove(size()-1)
    public T top(); // Like get(size()-1)
    // peek() is often a synonym for top()
    public int size();
    public int getCapacity();
}
Sums to 8 - No Recursion

Consider again the sums-to-8 problem

> javac Sums.java
> java Sums

8 = 1 1 1 1 1 1 1 1
8 = 1 1 1 1 1 1 2
8 = 1 1 1 1 1 2 1
8 = 1 1 1 1 1 3
8 = 1 1 1 1 2 1 1
..  
8 = 6 1 1
8 = 6 2
8 = 7 1
8 = 8

Use stacks to get the following

cur: 0 hist: ’’ toAdd: [8, 7, 6, 5, 4, 3, 2, 1]
cur: 1 hist: ’1’ toAdd: [7, 6, 5, 4, 3, 2, 1]
cur: 2 hist: ’1 1’ toAdd: [6, 5, 4, 3, 2, 1]
cur: 3 hist: ’1 1 1’ toAdd: [5, 4, 3, 2, 1]
cur: 4 hist: ’1 1 1 1’ toAdd: [4, 3, 2, 1]
cur: 5 hist: ’1 1 1 1 1’ toAdd: [3, 2, 1]
cur: 6 hist: ’1 1 1 1 1 1’ toAdd: [2, 1]
cur: 7 hist: ’1 1 1 1 1 1 1’ toAdd: [1]
cur: 8 hist: ’1 1 1 1 1 1 1 1’ toAdd: []
8 = 1 1 1 1 1 1 1

cur: 7 hist: ’1 1 1 1 1 1 1’ toAdd: []
cur: 6 hist: ’1 1 1 1 1 1’ toAdd: [2]
cur: 8 hist: ’1 1 1 1 1 1 2’ toAdd: []
8 = 1 1 1 1 1 1 2
...
...
8 = 6 2
cur: 6 hist: ’6’ toAdd: []
cur: 0 hist: ’’ toAdd: [8, 7]
cur: 7 hist: ’7’ toAdd: [1]
cur: 8 hist: ’7 1’ toAdd: []
8 = 7 1
Iterative Solutions

Use a little class to "simulate" a recursive call stack.

```java
public static void totalsTarget(int target){
    Stack<SumFrame> stack = new Stack<SumFrame>();
    SumFrame first = new SumFrame(0,target,"");
    stack.push(first);

    // Simulate the recursive call stack with a loop
    while(stack.size() > 0){
        SumFrame frame = stack.peek();
    }

    Store info about what should be done at each step in those frames

class SumFrame{
    public int current; // Current sum
    public Stack<Integer> toAdd; // Numbers remaining to add
    public String history; // History of adds that led here
}
```

Solution in SumsNoRecursion.java