CS 211: Recursion

Chris Kauffman

Week 13-1
Front Matter

Goals Today
Recursion and Stacks and P6

Lab 13: Recursion Exercises

P6: Mouse Maze
- Search for paths from Mouse to Cheese
- Medium sized project
- Tricky problems

Reading
- BJP Ch 12: Recursion
- Lab Manual Ch 18: Recursion

Schedule

<table>
<thead>
<tr>
<th>Date</th>
<th>Day</th>
<th>Time</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>4/18</td>
<td>Mon</td>
<td>Recursion</td>
<td>Stacks Lab Exercises</td>
</tr>
<tr>
<td></td>
<td>Wed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4/25</td>
<td>Mon</td>
<td>Linear/Binary Search</td>
<td>Sorting Lab Review P6 Due</td>
</tr>
<tr>
<td></td>
<td>Wed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5/2</td>
<td>Mon</td>
<td>Queues</td>
<td></td>
</tr>
<tr>
<td>5/4</td>
<td>Wed</td>
<td>Review</td>
<td></td>
</tr>
</tbody>
</table>
Rabbits

A puzzle.\(^1\)

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

▶ A newly born pair of rabbits, one male, one female, are put on an island;
▶ 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;
▶ Rabbits never die and a mating pair always produces one new pair (one male, one female) every month from the second month on.

How many pairs will there be in one year?

\(^1\)Adapted from Wikipedia
Simulation

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></td>
<td>B0</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></td>
<td></td>
<td>B0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
</tr>
<tr>
<td>Pair 3</td>
<td></td>
<td></td>
<td></td>
<td>B0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
<td>M0</td>
</tr>
<tr>
<td>Pair 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B1</td>
<td>M1</td>
<td>M1</td>
<td>M1</td>
</tr>
<tr>
<td>Pair 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B0</td>
<td>M0</td>
<td>M0</td>
</tr>
<tr>
<td>Pair 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B1</td>
<td>M1</td>
<td>M1</td>
</tr>
<tr>
<td>Pair 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B2</td>
<td>M2</td>
</tr>
<tr>
<td>Pair 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B0</td>
</tr>
<tr>
<td>Pair 9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B1</td>
</tr>
<tr>
<td>Pair 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B2</td>
</tr>
<tr>
<td>Pair 11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B3</td>
</tr>
<tr>
<td>Pair 12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B4</td>
</tr>
</tbody>
</table>
How does the population of a month relate to previous months?
Recursively

Population for Month \( i = \) Pop. Month \( i-1 \) + Pop. Month \( i-2 \)
Better known as *Fibonacci Numbers*:

\[
\begin{align*}
  f_0 &= 0 \\
  f_1 &= 1 \\
  f_i &= f_{i-1} + f_{i-2}
\end{align*}
\]

```java
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

```java
public static int fibR(int n)
```

- Recursive implementation
- View Stack Trace of `fibR(4)`

Recursive

```java
public static int fibI(int n)
```

- Iterative implementation?
- View Stack Trace of `fibI(4)`

Point
Recursion utilizes the Stack to store information about history
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 it?
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

```bash
> 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?
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.

Subject of implementation in Project 6: AStack.
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

Questions

- Examples of stacks?
- How would you implement a stack using arrays?

Stacks are a LIFO: Last In First Out
Array Based Implementation of Stacks

- Similar to ArrayList, subject of P6 implementation
- 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

```java
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

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 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

...
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
```