

Recursion

CMSC 433

Bill Pugh and

Nelson Padua-Perez

Fixed schedule

- Project 5 - Dense bags and Markov text
 - Due next Thursday, April 6th
- 2nd Midterm, Monday, April 10th
- Readings from now to midterm:
 - Chapter 7: Recursion
 - Section 8.1: Tree terminology

Recursion

aka divide and conquer

- General approach to solving problems
- If the problem instance is simple/trivial, solve it directly
- Otherwise,
 - break the problem instance down into one or more smaller instances
 - solve them
 - combine solutions from smaller instances to get solution for entire problem

How many ways are there to order the numbers 1..n?

- $\text{permutation}(1) = 1$
- $\text{permutation}(n) =$
 - n different numbers that could occur first
 - remaining elements could appear in $\text{permutation}(n-1)$ different orders
- thus, $\text{permutation}(n) = n * \text{permutation}(n-1)$
 $= n!$

Iteration/Recursion

- Iteration over a list/sequence is very similar to using recursion to combine the result from the first element with the result from the rest of the list
 - ```
int countVowels(String s) {
 if (s.length() == 0) return 0;
 int tailResult = countVowels(s.substring(1));
 switch (s.charAt(1)) {
 case 'a': case 'e': case 'i': case 'o': case 'u':
 return tailResult+1;
 default:
 return tailResult;
 }
}
```

# Counting vowels with iteration

```
– int countVowels(String s) {
 int result = 0;
 for(int i = 0; i < s.length(); i++) {
 switch (s.charAt(i)) {
 case 'a': case 'e': case 'i': case 'o': case 'u':
 result++; break;
 }
 }
 return result;
}
```

# Boring recursion

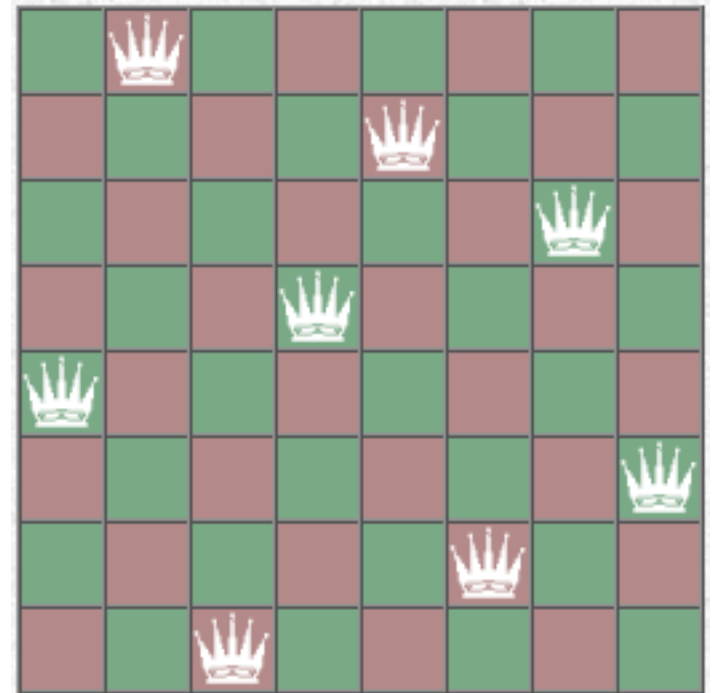
- Using recursion where iteration would work just as well isn't that exciting
  - sometimes, a little to think about it as a recursive problem
  - In Java, it isn't particularly efficient for long lists or deep recursion
  - In some languages (such as Scheme) that support tail recursion, it is as efficient as iteration. In fact, it can be the only/preferred way to implement iteration
    - has to be a special kind of recursion, called tail recursion

# Some interesting recursion

- Binary search
- Quicksort
- Mergesort

# nQueens

- Place queens on a board such that every row and column contains one queen, but no queen can attack another queen
  - place queens on  $n \times n$  board
  - recursive approach: assume you've already placed  $k$  queens



# recursive permutation generation

- Define a function  
    `calculatePermutations(int [] a)`
- that generates each permutation of `a` and calls  
    `handlePermutation(int [] a)`
- with each generated permutation
- Hint: define a recursive helper function  
    `calculatePermutations(int [] a, int k)`
- permutes the values in `a[k ... a.length-1]`

# Change making

- We've been asked to help devise a new set of coins for a country ruled by a mathematician.
- Our task is to figure out what the denominations of the coins should be.
  - For example, in the US, the coins have denominations (in pennies) of: 1, 5, 10, 25 (we'll ignore half dollars for now)
- Assume we want to have only three coins.
- What values should the coins have so as to minimize the average number of coins needed to make any number of cents from 0 to 99.

# Greedy vs. nonGreedy

- Greedy approach:
  - use as many of the largest coin possible, then go to the next largest coin
- Greedy is not always optimal
  - If your coins are 25, 10, 1, then to make change for 30 cents, the greedy approach requires 6 coins whereas only 3 dimes are required
- For US coins, greedy is optimal.