Recursion — Pt 2
Towers of Hanoi

Extra Credit: +15 to homework grade
http://165.106.10.159/~gtowell/Towers/
Must be done by May 5
Must submit a believable time.  +5 in-class final round.
LAB

• For a binary search over an array containing the numbers 1..20, what is the sequence of recursive function calls to find 11

• show all of the arguments to each recursive call
/**
 * Binary search, recursively on sorted internal array of ints
 * @param target the item to be found
 * @param lo the bottom of the range being searched
 * @param hi the top of the range being searched
 * @param steps the number of steps the search has taken
 * @return true if the target was found
 */

private boolean searchUtil(int target, int lo, int hi, int steps) {
    if (lo>hi) return false;
    int mid = (lo+hi)/2;
    System.out.println(target + " " + data[mid] + " " + lo + " " + hi + " " + steps);
    if (data[mid]==target) return true;
    if (data[mid]<target)
        return searchUtil(target, mid+1, hi, steps+1);
    else
        return searchUtil(target, lo, mid-1, steps+1);
}
### Calls to search Util

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
</tbody>
</table>

searchUtil(target, lo, hi, steps)

searchUtil(11, 0, 19, 0)
  mid = 9

searchUtil(11, 10, 19, 1)
  mid = 14

searchUtil(11, 10, 13, 2)
  mid = 11

searchUtil(11, 10, 10, 3)
  return true
Counter the number of occurrences of a letter in a string

```java
public int numOccur1(char ch, String str) {
    if (str == null || str.equals("")) {
        return 0;
    }
    int count = 0;
    if (str.charAt(0) == ch) {
        count++;
    }
    numOccur1(ch, str.substring(1));
    return count;
}
```

What does this return on “a”, “abc”
Why?
int acount = 0;

public int numOccur2(char ch, String str) {
    if (str == null || str.equals("")) {
        return 0;
    }
    if (str.charAt(0) == ch) {
        acount++;
    }
    numOccur2(ch, str.substring(1));
    return acount;
}

Correct answer, but a BAD solution
Occurrence count v3 and v4

```java
public int numOccur3(char ch, String str) {
    if (str == null || str.equals("")) { return 0; }
    int count = 0;
    if (str.charAt(0) == ch) { count = 1; }
    return count + numOccur3(ch, str.substring(1));
}

public int numOccur4(char ch, String str) {
    return numOccur4Util(ch, str, 0);
}

private int numOccur4Util(char ch, String str, int count) {
    if (str == null || str.equals("")) { return count; }
    if (str.charAt(0) == ch) { count++;
    return numOccur4Util(ch, str.substring(1), count);
}
```
public int numOccur5(char ch, String str) {
    if (str == null || str.length()==0)
        return 0;
    return numOccur5Util(ch, str, 0, 0);
}

private int numOccur5Util(char ch, String str, int loc, int count) {
    if (loc >= str.length())
        return count;
    if (str.charAt(loc) == ch) { count++; }
    return numOccur5Util(ch, str, loc+1, count);
}

public int numOccur6(char ch, String str) {
    if (str == null || str.length()==0)
        return 0;
    return numOccur6Util(ch, str, 0);
}

private int numOccur6Util(char ch, String str, int loc) {
    if (loc >= str.length())
        return 0;
    int cc = 0;
    if (str.charAt(loc) == ch) { cc=1; }
    return cc+numOccur6Util(ch, str, loc+1);
public ArrayList<Integer> rAccmulateB(int count)
{
    if (count <= 0)
        return new ArrayList<Integer>();
    ArrayList<Integer> alAcc = rAccmulate(count - 1);
    alAcc.add(count);
    return alAcc;
}

public ArrayList<Integer> rAccmulate(int count) {
    if (count <= 0)
        return new ArrayList<Integer>();
    ArrayList<Integer> ret = new ArrayList<Integer>(count);
    rAccmulateUtil(count, ret);
    return ret;
}

private void rAccmulateUtil(int count, ArrayList<Integer> arrLis) {
    if (count <= 0)
        return;
    arrLis.add(count);
    rAccmulateUtil(count - 1, arrLis);
}

public static void main(String[] args) {
    System.out.println("AA " + (new AB()).rAccmulate(5));
    System.out.println("BB " + (new AB()).rAccmulateB(5));
}

What is the output?
Recursion and Backtracking

• All problems considered so far progress steadily towards an answer.

• Consider a maze. Sometimes you need to "backtrack".

• Idea:
  • somehow make a copy of where you are,
  • try going forward using your copy,
  • If that fails back up and go some other direction using your original

• Alternately
  • when backing up, undo your change

• Twiddle
  • especially with mazes mark places you have been so you do not retry failed paths
N Queens problem

- Place N queens on an NxN chessboard such that no queen can take another
- Strategy:
  - on row N
    - move across columns trying a spot for OK
    - if OK, then recur with N+1
    - if have checked everything in a column and there is no place that is OK
      - backtrack
        - undo placement of queen in row N-1 and continue across that row
N Queens

• board just a 2d array of chars
• will do recursion with a private utility function

```java
public class NQueens {
    private char[][] board;
    private int size = 0;

    public NQueens(int siz) {
        size = siz;
        board = new char[size][size];
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                board[i][j] = '.';
            }
        }
    }

    private void showBoard() {
        for (int r = 0; r < size; r++) {
            for (int c = 0; c < size; c++) {
                System.out.print(board[r][c]);
            }
            System.out.println();
        }
    }

    public void doQueens() {
        doQueensUtil(0);
    }

    private void doQueensUtil(int i) {
        // Recursion logic here
    }
}
```
The base case(s) are:

- The row being asked to consider is off board
  - return true;

- In the row
  - Go across every column
    - Put queen in a column
      - Check if that is OK
        - If it is, go to recur to next row
        - If found solution return true;
      - If NOT OK, remove queen from column
    - If cannot find a place to put a queen, return false