csci 210: Data Structures

Recursion II
Summary

- **Topics**
  - previous
    - recursion overview
    - simple examples
    - Sierpinski gasket
    - Hanoi towers
  - today
    - counting blobs in a grid

- **READING:**
  - GT textbook chapter 3.5
Blob Check

• Problem: you have a 2-dimensional grid of cells, each of which may be filled or empty. Filled cells that are connected form a “blob” (for lack of a better word).

• Write a recursive method that returns the size of the blob containing a specified cell (i,j)

• Example

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

- BlobCount(0,3) = 3
- BlobCount(0,4) = 3
- BlobCount(3,4) = 1
- BlobCount(4,0) = 7

• Solution ??
  - essentially you need to check the current cell, its neighbors, the neighbors of its neighbors, and so on
  - think RECURSIVELY
• When you try to write a recursive methods, always start from the base case

• What are the base cases for counting the blob?
  • imagine a call BlobCheck(i,j)
  • when is there no need for recursion, and the function can return the answer immediately?

• calling BlobCheck(i,j)
  • (i,j) may be outside of grid
  • (i,j) may be EMPTY
  • (i,j) may be FILLED

• Base cases
  • (i,j) is outside grid
  • (i,j) is EMPTY
- **blobCheck(i,j)**
  - if (i,j) is FILLED
    - 1 (for the current cell)
    - + count blobs of its 8 neighbors

\[
blobc = 1 \\
\text{for } (l = -1; l <= 1; l++) \\
\quad \text{for } (k = -1; k <= 1; k++) \\
\quad \quad \text{//skip of middle cell} \\
\quad \quad \text{if } (l==0 \&\& k==0) \text{ continue;} \\
\quad \quad \text{blobc } += \text{blobCheck}(i+k, j+l);
\]

- **Example: blobCheck(1,1)**
  - blobCount(1,1) calls blobCount(0,2)
  - blobCount(0,2) calls blobCount(1,1)

- **Problem: infinite recursion**
- **Why?**
  - multiple counting of the same cell
Marking your steps

- **Idea**
  - once you count a cell, mark it so that it is not counted again by its neighbors

```
blobCheck(1,1)
```

- count it and mark it
- then find counts of neighbors, recursively
  - $+ \text{blobCheck}(0,0)$
  - $+ \text{blobCheck}(0,1)$
  - $+ \text{blobCheck}(0,2)$
  - ...

```
Correctness

- blobCheck(i,j) works correctly if the cell (i,j) is not filled
- if the cell (ij) is FILLED
  - the cell is marked
  - the blob of this cell is 1 + blobCheck of all neighbors
  - because the cell is marked, the neighbors will not see it as FILLED
  - ==> a cell is counted only once

- Why does this stop?
  - blobCheck(i,j) will generate recursive calls to neighbors
  - recursive calls are generated only if the cell is FILLED
  - when a cell is marked, it is NOT FILLED anymore, so the size of the blob of filled cells is one smaller
  - ==> the blob when calling blobCheck(neighbor-of-ij) is one smaller that blobCheck(i,j)

- Note: after one call to blobCheck(i,j) the blob of (i,j) is all marked
  - need to do one pass and restore the grid
Try it out

- download blobCheckSkeleton
- fill in method blobCount(i,j)