R is for …

• Recursion
  • Base case of wonderful stuff

• Refactoring
  • No new functionality
Announcements

• Discussion 10 Monday 3/30 out by Friday

• Assignment P4 DNA-Link
  • Part 2 due March 26 – Code and more Analysis

• Assignment P5 will be out Friday
  • Then P6 will be last assignment

• APT-5 is now out and due Tuesday, March 31
Plan for LFIM

• Linear Data Structures
  • Stacks and Queues
  • Classes, Interfaces, Applications

• Priority Queue API

• Counting blob to motivate DFS and BFS
Some (Linear) Data Structures

• Back in the day … weeks on these. Now?
  • Queue: FIFO, Stack: LIFO, Priority Queue: …
  • We can use standard java.util classes
Images of Stacks

Google search ...
Images of Queues

Google search …
In Brazil, It’s Fine for Seniors to Cut the Line

Law requires special treatment so they don’t wait; ‘here comes the old bag’

Fortaleza, a city in Brazil, passed a law allowing people aged 60 and up to cut to the front of any line at any time.
QueueLike Example: Level Order Traversal

• How do we visit nodes in level-order?
  • See TreeDemo.java

```java
62  public void whatOrder(TreeNode root) {
63      LinkedList<TreeNode> list = new LinkedList<>();
64      if (root != null) {
65          list.add(root);
66      }
67      while (list.size() > 0) {
68          root = list.remove(0);
69          System.out.printf("%s\n", root.info);
70          if (root.left != null) list.add(root.left);
71          if (root.right != null) list.add(root.right);
72      }
73  }
```
QueueLike Example: Level Order Traversal

• How do we visit nodes in level-order?
  • See TreeDemo.java

```java
public void whatOrder(TreeNode root) {
    LinkedList<TreeNode> list = new LinkedList<>();
    if (root != null) {
        list.add(root);
    }
    while (list.size() > 0) {
        root = list.remove(0);
        System.out.printf("%s\n", root.info);
        if (root.left != null) list.add(root.left);
        if (root.right != null) list.add(root.right);
    }
}
```
Level Order Dissected

• What is in list before loop starts?
  • List: [macaque]

• Take node out, add left and right node to end
Level Order Dissected

• What is in list before loop starts?
  • List: [macaque]

• Take node out, add left and right node to end
  • List: [chimp, monkey]
  • List: [monkey, baboon, lemur]
  • List: [baboon, lemur, tamarin]
  • List: [lemur, tamarin]
  • List: [tamarin]
  • List: [orangutan]
  • List: [ ]
Order Printed
See TreeDemo.java
https://coursework.cs.duke.edu/201spring20/classcode/

macaque chimp monkey baboon lemur tamarin orangutan
QueueLike Example: Level Order Traversal

• What is the running time of printing in level-order?
• See TreeDemo.java

```java
public void whatOrder(TreeNode root) {
    LinkedList<TreeNode> list = new LinkedList<>();
    if (root != null) {
        list.add(root);
    }
    while (list.size() > 0) {
        root = list.remove(0);
        System.out.printf("%s\n", root.info);
        if (root.left != null) list.add(root.left);
        if (root.right != null) list.add(root.right);
    }
}
```
QueueLike Example: Level Order Traversal

- What is the running time of printing in level-order?
- See TreeDemo.java

```java
public void levelOrder(TreeNode root) {
    LinkedList<TreeNode> list = new LinkedList<>();
    if (root != null) {
        list.add(root);
    }
    while (!list.isEmpty()) {
        TreeNode root = list.remove(0);
        System.out.printf("%s\n", root.info);
        if (root.left != null) list.add(root.left);
        if (root.right != null) list.add(root.right);
    }
}
```
Compsci 201
Stacks, Queues, Priority Queues
Part 2 of 4

Susan Rodger
March 27, 2020
Queue: FIFO in Java

- Scheduling: processes on CPU, events in simulation
  - $O(1)$ add (to back) and remove (from front)
- Queue<> is interface, use LinkedList<>  
  - Methods `add()`, `peek()`, `remove()`

```java
public void qdemo() {
    Queue<String> q = new LinkedList<>();
    String[] strs = {"compsci", "is", "wonderful"};
    for(String s : strs) q.add(s);
    while (! q.isEmpty()) {
        System.out.println(q.remove());
    }
}
```
Queue: FIFO in Java

• Scheduling: processes on CPU, events in simulation
  • O(1) add (to back) and remove (from front)
• Queue<> is interface, use LinkedList<>  
  • Methods add(), peek(), remove()

```java
public void qdemo() {
    Queue<String> q = new LinkedList<>();
    String[] strs = {"compsci", "is", "wonderful"};
    for(String s : strs) q.add(s);
    while (! q.isEmpty()) {
        System.out.println(q.remove());
    }
}
```

OUTPUT: compsci is wonderful
Stack Code (LIFO)

- See Treeliterator in TreeDemo to find successor
  - https://coursework.cs.duke.edu/201spring20/classcode
  - O(1) add at end and remove from end: LIFO
  - ArrayList: push and pop are veneer

```java
public void sdemo() {
    Stack<String> sta = new Stack<>();
    String[] strs = {"compsci", "is", "wonderful"};
    for(String s : strs) sta.push(s);
    while (!sta.isEmpty()) {
        System.out.println(sta.pop());
    }
}
```
Stack Code (LIFO)

- See TreeIterator in TreeDemo to find successor
  - https://coursework.cs.duke.edu/201spring20/classcode
- O(1) add at end and remove from end: LIFO
- ArrayList: push and pop are veneer

```java
public void sdemo() {
    Stack<String> sta = new Stack<>();
    String[] strs = {"compsci", "is", "wonderful"};
    for (String s : strs) sta.push(s);
    while (!sta.isEmpty()) {
        System.out.println(sta.pop());
    }
}
```

OUTPUT: wonderful is compsci
Stack: LIFO in Java

• Often used in place of recursion: runtime stack
  • Each method "stacks" on top of other calls
• Postfix notation: 3 + 5 * 7 and 5 7 * 3 +
  • Sequentially scan expression left to right
    • Number? Push/add
    • Operand? Pop/remove, Pop/remove, push result
• Stack in Java like wrapper over ArrayList
  • Use \texttt{s.pop()} not \texttt{s.remove(s.size()-1)}
Understanding ADT via API

- **Abstract Data Type**: implementation unknown
  - Application Programming Interface
    - How to call, what to expect
  - [https://coursework.cs.duke.edu/201spring20/classcode](https://coursework.cs.duke.edu/201spring20/classcode)
- See **StackQueuePriorityQueue.java**
  - Each class implements Collection interface
  - Queue and PriorityQueue: Queue interface
  - Stack: push and pop veneer for ArrayList/Vector
Understanding Interfaces

• Why can we make three calls to \texttt{load}? Interface

  • Each of \texttt{qu}, \texttt{st}, \texttt{pq} have a \texttt{.add} method

```java
public static void load(String[] words,
                        Collection<String> coll) {
    for(String s : words) {
        coll.add(s);
    }
}
```

```java
String[] words = {
    "aardvarks", "become", "cooperative", "dangerous",
    "elegant", "ferocious"
};
Collections.shuffle(Arrays.asList(words));

Queue<String> qu = new LinkedList<>();
Stack<String> st = new Stack<>();
PriorityQueue<String> pq = new PriorityQueue<>();
load(words, qu);
load(words, st);
load(words, pq);
```
Arrays.asList backed by array

```java
String[] words = {
    "aardvarks", "become", "cooperative", "dangerous",
    "elegant", "ferocious"
};
Collections.shuffle(Arrays.asList(words));

System.out.println("**** original ****");
for(String s : words) {
    System.out.printf("%s\n", s);
}
```

AFTER ONE RUN:
**** original ****

ferocious
dangerous
elegant
cooperative
aardvarks
become

AFTER A SECOND RUN:
**** original ****
elegant
dangerous
become
aardvarks
cooperative
ferocious
Overloaded Method

• Not `@Override`. Same name, different parameters
  • Compiler determines which to call
    • `printDeplete(stack);`
    • `printDeplete(queue);`

```java
public static void printDeplete(Queue<String> qu) {
    while (!qu.isEmpty()) {
        String s = qu.remove();
        System.out.printf("%s
", s);
    }
}

public static void printDeplete(Stack<String> st) {
    while (!st.isEmpty()) {
        String s = st.pop();
        System.out.printf("%s
", s);
    }
}
```
Let’s look at the code
See StackQueuePriorityQueue.java

https://coursework.cs.duke.edu/201spring20/classcode
What's printed by printDelete?

• For stack? Reverse order of list
  • Last in, First out

• For queue? Same order as list
  • First in, First out

• For priority queue? Alphabetical order
  • Smallest (minimal priority) out first unless ...

```java
PriorityQueue<String> pq =
    new PriorityQueue<>((Comparator.reverseOrder()));
```
WOTO

Ayanna Howard

• Educator, Researcher and Innovator
• Professor & Chair of the School of Interactive Computing, Georgia Tech
• Robotics – Robots and Bias, Robots changing lives of children with disabilities, Robots beyond part of the family
• Top 50 U.S. Women in Tech, Forbes, 2018

I believe that every engineer has a responsibility to make the world a better place. We are gifted with an amazing power to take people’s wishes and make them a reality.
Compsci 201
Stacks, Queues, Priority Queues
Part 3 of 4

Susan Rodger
March 27, 2020
Flood Fill, Blob Count

https://coursework.cs.duke.edu/201spring20/backtracking-sp20

data: lots of stars

program numbers blobs
Applications

• Using draw/paint program, fill a region
  • Redeye removal
  • Give image partially transparent background

• Finding edges or blobs in image processing
Blob Counting Ideas

• How do I count “my region” or “my blob” size?
  • Ask my neighbors their size
  • Add +1 to their result
  • Avoid double-counting!

• Colors indicate calls
  • White calls green
    • Green calls tan and gray
    • Gray calls yellow, …
Blob Counting Ideas

• How do I count “my region” or “my blob” size?
  • Ask my neighbors their size
  • Add +1 to their result
  • Avoid double-counting!

• Colors indicate calls
  • White calls green
    • Green calls tan and gray
    • Gray calls yellow, …
Blob Counting Ideas

- How do I count “my region” or “my blob” size?
  - Ask my neighbors their size
  - Add +1 to their result
  - Avoid double-counting!

- Colors indicate calls
  - White calls green
    - Green calls tan and gray
    - Gray calls yellow, …
Blob Counting Ideas

• How do I count “my region” or “my blob” size?
  • Ask my neighbors their size
  • Add +1 to their result
  • Avoid double-counting!

• Colors indicate calls
  • White calls green
    • Green calls tan and gray
    • Gray calls yellow, …
**Blob Counting Ideas**

- How do I count “my region” or “my blob” size?
  - Ask my neighbors their size
  - Add +1 to their result
  - Avoid double-counting!

- Colors indicate calls
  - White calls green
    - Green calls tan and gray
    - Gray calls yellow, …

```
BLOB  BLOB  BLOB
BLOB          BLOB
BLOB  BLOB
```
Blob Counting Ideas

- How do I count “my region” or “my blob” size?
  - Ask my neighbors their size
  - Add +1 to their result
  - Avoid double-counting!

- Colors indicate calls
  - White calls green
    - Green calls tan and gray
    - Gray calls yellow, …

3/27/2020 Compsci 201, Spring 2020
Blob Counting Ideas

• How do I count “my region” or “my blob” size?
  • Ask my neighbors their size
  • Add +1 to their result
  • Avoid double-counting!

• Colors indicate calls
  • White calls green
    • Green calls tan and gray
    • Gray calls yellow, …
Blob Counting Ideas

• How do I count “my region” or “my blob” size?
  • Ask my neighbors their size
  • Add +1 to their result
  • Avoid double-counting!

• Colors indicate calls
  • White calls green
    • Green calls tan and gray
    • Gray calls yellow, …
Blob Counting Ideas

- How do I count “my region” or “my blob” size?
  - Ask my neighbors their size
  - Add +1 to their result
  - Avoid double-counting!

- Colors indicate calls
  - White calls green
    - Green calls tan and gray
    - Gray calls yellow, …

```
Return 1
```
Blob Counting Ideas

• How do I count “my region” or “my blob” size?
  • Ask my neighbors their size
  • Add +1 to their result
  • Avoid double-counting!

• Colors indicate calls
  • White calls green
    • Green calls tan and gray
    • Gray calls yellow, …
Blob Counting Ideas

• How do I count “my region” or “my blob” size?
  • Ask my neighbors their size
  • Add +1 to their result
  • Avoid double-counting!

• Colors indicate calls
  • White calls green
    • Green calls tan and gray
    • Gray calls yellow, …
Blob Counting Ideas

• How do I count “my region” or “my blob” size?
  • Ask my neighbors their size
  • Add +1 to their result
  • Avoid double-counting!

• Colors indicate calls
  • White calls green
    • Green calls tan and gray
    • Gray calls yellow, …
Blob Counting Ideas

• How do I count “my region” or “my blob” size?
  • Ask my neighbors their size
  • Add +1 to their result
  • Avoid double-counting!

• Colors indicate calls
  • White calls green
    • Green calls tan and gray
    • Gray calls yellow, …
Blob Counting Ideas

• How do I count “my region” or “my blob” size?
  • Ask my neighbors their size
  • Add +1 to their result
  • Avoid double-counting!

• Colors indicate calls
  • White calls green
    • Green calls tan and gray
    • Gray calls yellow, …

Return 4
Blob Counting Ideas

- How do I count “my region” or “my blob” size?
  - Ask my neighbors their size
  - Add +1 to their result
  - Avoid double-counting!

- Colors indicate calls
  - White calls green
    - Green calls tan and gray
    - Gray calls yellow, …

Return 4
Blob Counting Ideas

• How do I count “my region” or “my blob” size?
  • Ask my neighbors their size
  • Add +1 to their result
  • Avoid double-counting!

• Colors indicate calls
  • White calls green
    • Green calls tan and gray
    • Gray calls yellow, …

Return 4
Blob Counting Ideas

• How do I count “my region” or “my blob” size?
  • Ask my neighbors their size
  • Add +1 to their result
  • Avoid double-counting!

• Colors indicate calls
  • White calls green
    • Green calls tan and gray
    • Gray calls yellow, …

Return 4
Blob Counting Ideas

• How do I count “my region” or “my blob” size?
  • Ask my neighbors their size
  • Add +1 to their result
  • Avoid double-counting!

• Colors indicate calls
  • White calls green
    • Green calls tan and gray
    • Gray calls yellow, …
Blob Counting Ideas

• How do I count “my region” or “my blob” size?
  • Ask my neighbors their size
  • Add +1 to their result
  • Avoid double-counting!

• Colors indicate calls
  • White calls green
    • Green calls tan and gray
    • Gray calls yellow, …
Blob Counting Ideas

• How do I count “my region” or “my blob” size?
  • Ask my neighbors their size
  • Add +1 to their result
  • Avoid double-counting!

• Colors indicate calls
  • White calls green
    • Green calls tan and gray
    • Gray calls yellow, …
Blob Counting Ideas

• How do I count “my region” or “my blob” size?
  • Ask my neighbors their size
  • Add +1 to their result
  • Avoid double-counting!

• Colors indicate calls
  • White calls green
    • Green calls tan and gray
    • Gray calls yellow, …

Blob is of size 8!
Bookkeeping Details

• Blobs marked with ‘*’ – aka asterisk
  • We fill or mark cells when counting
  • Filling avoids counting cells twice
  • Filling allows for visualization as well

• BlobModel.blobFill method: (row,col) for start
  • lookFor character, e.g., ‘*’
  • fillWith character, e.g., ‘1’ or ‘7’ or …
  • If blob not big enough? Erase: lookFor = fillWith
Run the program

- See code in:
  https://coursework.cs.duke.edu/201spring20/backtracking-sp20
Two Stage Analysis

• Before looking for blobs? Make a copy of data
  • Data stored in two-dimensional array `int[][]`
  • char values can be stored as int values

• Look at every possible blob starting location
  • If it's a blob? Mark with number
  • If it's not a blob? Erase any markings
    • There is a minimal size, erasing cleans up
Lookfor for Blobs Everywhere

- Two parameters in each call to blobFill

```c
int bcount = 0;
for (int j = 0; j < rows; j++) {
    for (int k = 0; k < cols; k++) {
        int size = blobFill(j, k, BLOB_ON, bcount + 1);
        if (size >= minSize) {
            bcount++;
        } else {
            blobFill(j, k, bcount + 1, BLOB_OFF);
        }
    }
}
```
Blobs starting at (row,col)

- Ask each of four neighbors for their blob-size
- Add me, +1, to what they report back

```java
protected int blobFill(int row, int col, int lookFor, int fillWith) {
    int size = 0;
    if (inRange(row, col)) {
        if (myGrid[row][col] != lookFor) {
            return 0;
        }
        myGrid[row][col] = fillWith; // mark pixel
        size = 1; // count this pixel, then scout for neighbors
        size += blobFill(row - 1, col, lookFor, fillWith) + blobFill(row + 1, col, lookFor, fillWith) + blobFill(row, col - 1, lookFor, fillWith) + blobFill(row, col + 1, lookFor, fillWith);
    }
    return size;
}
```
Base cases: when process stops

- If \((row, col)\) is not in bounds? Do nothing
- If \(grid[row][col] \neq \text{lookFor}\)? Do nothing

- Otherwise (what do we know here?)
  - In bounds AND looking for ‘*’ for example
  - Look at horizontal and vertical neighbors
  - Use results of recursive calls to create return
blobFill(row,col)

- Returns a value, make sure value used
  - Original call: is blob big enough? use value
  - Recursive calls: add values to return total

- How do we know this will terminate?
  - Each recursive call “marks” a cell
    - myGrid[row][col] = fillWith
  - Unless fillWith == lookFor, not infinite!
WOTO for Blobs

One way to think about the magnitude of the changes to come is to think about how you went about your business before powerful Web search engines.

You probably wouldn't have imagined that a world of answers would be available to you in under a second. The next set of advances will have a different effect, but similar in magnitude.

Compsci 201
Stacks, Queues, Priority Queues
Part 4 of 4

Susan Rodger
March 27, 2020
Level Order aka Breadth First

- What is in list before loop starts? List: [macaque]
- What is added to list first time through loop
  - List: [chimp, monkey]
  - next time? [monkey, baboon, lemur], …

```java
public void whatOrder(TreeNode root) {
    LinkedList<TreeNode> list = new LinkedList<>();
    if (root != null) {
        list.add(root);
    }
    while (list.size() > 0) {
        root = list.remove(0);
        System.out.println(root.info);
        if (root.left != null) list.add(root.left);
        if (root.right != null) list.add(root.right);
    }
}
```
Inorder with a Stack

- **Iterator interface**: `.next()` and `.hasNext()`
  - Two values: current node, location in node
  - The "next" node is easy in a linked list
- **Contrast to tree iteration**
  - What's the next node?
  - Can use parent pointer
  - Can use stack with parents
Inorder with a Stack

- **Iterator interface**: `.next()` and `.hasNext()`
  - Two values: current node, location in node
  - The "next" node is easy in a linked list

- **Contrast to tree iteration**
  - What's the next node?
  - Can use parent pointer
  - Can use stack with parents
  - **Inorder**: 8, 7, 4, 1, 3, 5, 2, 6
Use BFS instead of recursion

https://coursework.cs.duke.edu/201spring20/backtracking-sp20

- **IterativeBlobModel**
  - Extends BlobModel, `@Override` two methods
  - Track grid cells (row,col) create Pair class

```java
private Pair[][] myPairGrid;

@Override
public void initialize(int rows, int cols, int count){
    super.initialize(rows,cols,count);
    myPairGrid = new Pair[myGrid.length][myGrid[0].length];
    for(int r=0; r < myPairGrid.length; r++){
        for(int c=0; c < myPairGrid[0].length; c++){
            myPairGrid[r][c] = new Pair(r,c);
        }
    }
}
```
From Trees to Grids with BFS

- Level order queue: dequeue and enqueue children
- Blob BFS: dq then enqueue four neighbors
- Shortcut: deltas/offsets to find four neighbors
  - left, right, down, up as shown

```java
@override
def blobFill(row, col, lookFor, fillColor):
    size = 0
    rowDelta = [-1, 1, 0, 0]
    colDelta = [0, 0, -1, 1]
    if myGrid[row][col] != lookFor:
        return 0  # not part of blob
    Queue<Pair> qp = new LinkedList<>();
    myGrid[row][col] = fillColor  # mark pixel
    size++  # count pixel
    qp.add(myPairGrid[row][col]);
```
Level Order aka Breadth First

- What is in list before loop starts? List: [macaque]
- What is added to list first time through loop
  - List: [chimp, monkey]
  - next time? [monkey, baboon, lemur], ...

```java
public void whatOrder(TreeNode root) {
    LinkedList<TreeNode> list = new LinkedList<>();
    if (root != null) {
        list.add(root);
    }
    while (list.size() > 0) {
        root = list.remove(0);
        System.out.println("%s\n", root.info);
        if (root.left != null) list.add(root.left);
        if (root.right != null) list.add(root.right);
    }
}
```
Queue for breadth first search

- Blob BFS: dequeue then enqueue four neighbors
  - Don’t enqueue if seen or off grid

```java
while (qp.size() != 0){
    Pair p = qp.remove();
    for(int k=0; k < rowDelta.length; k++){
        row = p.row + rowDelta[k];
        col = p.col + colDelta[k];
        if (inRange(row, col) && myGrid[row][col] == lookFor){
            qp.add(myPairGrid[row][col]);
            myGrid[row][col] = fillWith;
            size++;
        }
    }
}
return size;
```
Coding trick

• Define the offsets for neighbors

```java
int[] rowDelta = {-1,1,0,0};
int[] colDelta = {0,0,-1,1};
```

• Use offsets to loop over four neighbors

```java
for(int k=0; k < rowDelta.length; k++){
    row = p.row + rowDelta[k];
    col = p.col + colDelta[k];
}
```
Coding trick

• Define the offsets for neighbors

```java
int[] rowDelta = {-1,1,0,0};
int[] colDelta = {0,0,-1,1};
```

• Use offsets to loop over four neighbors

```java
for(int k=0; k < rowDelta.length; k++){
    row = p.row + rowDelta[k];
    col = p.col + colDelta[k];
}
```
Coding trick

- Define the offsets for neighbors
  
  ```java
  int[] rowDelta = {-1, 1, 0, 0};
  int[] colDelta = {0, 0, -1, 1};
  ```

- Use offsets to loop over four neighbors
  
  ```java
  for(int k=0; k < rowDelta.length; k++){
      row = p.row + rowDelta[k];
      col = p.col + colDelta[k];
  }
  ```
Coding trick

- Define the offsets for neighbors

```
int[] rowDelta = {-1,1,0,0};
int[] colDelta = {0,0,-1,1};
```

- Use offsets to loop over four neighbors

```
for(int k=0; k < rowDelta.length; k++){
    row = p.row + rowDelta[k];
    col = p.col + colDelta[k];
}
```
Coding trick

- Define the offsets for neighbors

```java
int[] rowDelta = {-1,1,0,0};
int[] colDelta = {0,0,-1,1};
```

- Use offsets to loop over four neighbors

```java
for(int k=0; k < rowDelta.length; k++){
    row = p.row + rowDelta[k];
    col = p.col + colDelta[k];
}
```
How to search 8 neighbors?

- We need to make 8 recursive calls
  - W, NW, N, NE, E, SE, S, SW
  - See coding “trick” below

```java
int[] rd = {0, -1, -1, -1, 0, 1, 1, 1};
int[] cd = {-1, -1, 0, 1, 1, 1, 0, -1};
for(int d = 0; d < rd.length; d+= 1){
    int nr = row + rd[d];
    int nc = col + cd[d];
    size += blobFill(nr, nc, ...)
}
```
Queue = BFS, Stack = DFS

• With the queue we first search every neighbor one-away from start, then two-away, then ...  
  • Flooding in a ripple/wave from start

• With recursion or explicit stack: go down a path as far as possible, then back up and continue  
  • Potentially less storage then queue if many adjacent cells being explored
Next Assignment P5: Percolation

- We will be talking about this assignment next week.
- Look at different algorithms
  - DFS
  - BFS
  - UnionFind