Announcements

- Exam 1 – Ask for Regrade in Gradescope by March 1
- Regrades assignments
  - if you pushed to github but did not resubmit in gradescope, fill out regrade form and we can look at your github if you have not modified it!
- Assignment P3 last chance today on time
- Assignment P4 out today with a Part1 and Part2
  - Part 1 due March 5, Part 2 due March 19
- APT 4 due Tuesday!
- APT Quiz 1 – now on regular APT page
  - Not for credit, but finish if you didn’t

N is for …

- new
  - Allocating memory from the heap

- null
  - Value when nothing has been allocated

PFtLFiF

- Introduction to Recursion
  - Canonical problem-solving/programming tool
  - Useful for lists, trees, and when structure is self-referential (algorithmic too, not today)

- Review linked lists in context of P4: DNA-link
  - You can work with a partner from your Discussion section
  - Choose next week, run code, finish after break
Modify and Return linked list

- If we pass a pointer to first node and ..
  - Want to "remove first"
  - We must return a pointer to modified list
- void change(ListNode first)
  - Call change(list)
    - first = first.next
  - list not changed after call

What does pass-by-value mean?

- Java passes parameters by value
  - Pass a copy of the variable
  - A copy of list1 is passed

list1 = ld.deleteAll(list1, "corn");

list1 = ld.deleteAll(list1,"squash");

Idiom: pass-and-return

- Change the list passed in, return the list.
  - Assign in the call, e.g. x = changeUp(x)

Thing xx = new Thing();
change(xx);
// can xx be different after call?
// can write xx.mutate()
// cannot assign to xx in change
xx = changeUp(xx);

Invariants

- Class level: true before each method executes
  - Established at construction
  - Re-established by each method

- Loop level: true before each loop guard evaluation
  - Established before first iteration of loop
  - Re-established after each loop iteration

- Reason formally and informally about code
What is the Internet?
- A network of networks ….

What is PageRank?
- What’s a good website link?

```java
public int calc(int n) {
    return n * calc(n - 1);
}
```

Self Reference and Recursion

- Does a Node reference itself?
  - No, there’s a .next field, but …
- Does a recursive method call itself?
  - No, calls clones of itself
  - Careful, could make “infinite” number of calls …
- What’s in a folder?
  - Files and other folders. Is that self-referential?

Google (DYM): Recursion

Did you mean …?

- Those software engineers …
  - Did you mean invented by Noam Shazeer, Duke 1998: Math and CompSci
When to use recursion

- The structure of the problem lends itself …
  - Folders/Directories contain …
  - Nodes in a linked list contain …

- The algorithmic structure lends itself …
  - Sorting algorithms as we’ll see …
  - Factorial? Just say no …

Size of a linked list

- You’ve seen a loop to do this
  - Goal: try to understand why this is correct
  - We’ll use example from arithmetic too

- Vocabulary with both structure and algorithm

```java
public int count(ListNode list) {
    if (list == null) return 0;
    return 1 + count(list.next);
}
```

Vocabulary

- All recursive code has a base case
  - A simple case where no recursion necessary
  - Example in linked list? null, no recursion
    - sometimes one node case too
  - Base case always identified with an if statement.
Understanding Recursion

- **Visualize:** RecDemo.java
  - The base case anchors the recursion
  - https://coursework.cs.duke.edu/201spring20/classcode/blob/master/src/RecDemo.java

- There's no loop! Why?
  - Sequence of recursive calls
  - Stacked up until base returns

- The recursive call "decreases"
  - Must get toward **base case**

---

RecDemo.java

```java
public class RecDemo {
    public class ListNode {
        int info;
        ListNode next;
        ListNode(int val, ListNode link) {
            info = val;
            next = link;
        }
    }

    public ListNode create(int n) {
        ListNode front = null;
        for (int k=0; k < n; k++) {
            front = new ListNode(k, front);
        }
        return front;
    }
}
```

---

RecDemo.java – rest of it

```java
    public int count(ListNode list) {
        if (list == null) return 0;
        int allButMe = count(list.next);
        return 1 + allButMe;
    }

    public void doit() {
        ListNode list = create(4);
        int n = count(list);
        System.out.println(n);
    }

    public static void main(String[] args) {
        RecDemo rd = new RecDemo();
        rd.doit();
    }
```

---

About to count # nodes in list

- **Call create(4) finished, call count(list)**
  - What does `list` point to? `list.next`?
First recursive call made

- Each method on the stack/pile of methods has its own local state: what does list reference?
  - Node 3 in doit/first call, node 2 in recursive call

Second recursive call made

- Three calls of count made: where is active call?
  - Parameter list points at 1, what happens?

\begin{lstlisting}[language=Java]
public int count(ListNode list) {
    if (list == null) return 0;
    int allButMe = count(list.next);
    return 1 + allButMe;
}

main:29
rd:

doit:24
this
list

count:19
this
list

What do we see?

- Each method invocation has its own state: parameter, local variables, line number

- Goal: trust recursion
  - Trust is hard
  - Debugging on trust? Not so easy

Last call: base case reached

- The active call has list == null
  - Base case reached! Return 0

- Where is this value returned?
  - To the call: the stack frame "above"

- Addition happens back up call-chain

\begin{lstlisting}[language=Java]
public int count(ListNode list) {
    if (list == null) return 0;
    int allButMe = count(list.next);
    return 1 + allButMe;
}

main:29
rd:

doit:24
this
list

count:19
this
list

\end{lstlisting}
How did recursion work?

- Structure of a linked list is essential
  - For a non-null list, # nodes is: count me, and recursively count the rest, add together

- Recursion in general: process one case, one number, one node. Make a recursive call, use result.
  - Code must use return value of recursive call
  - For lists? Deal with one node only in code

How do you calculate N!?

- Multiply 1 x 2 x 3 x ... x N

```java
public int fact(int n) {
    int prod = 1;
    for (int k=2; k <= n; k++) {
        prod *= k;
    }
    return prod;
}
```

Recursive Terminology

- Recursive methods must have a base case
  - Simple to do, don’t need “help”

- Recursive calls make progress toward base case
  - Some measure gets smaller, toward base case

- What’s n?
  - It’s n * (n-1)!
  - What’s the base case? 1! Is 1 (or 0! Is 1)

Don’t do this!

- int x = fact(4);
  - return 4*fact(3)

- The call of fact(3) calls a “clone” or “copy”
  - Doesn’t call “itself”, is re-entrant code

```java
public int fact(int n) {
    if (n == 1) return 1;
    return n*fact(n-1);
}
```
Don’t do this 2

```
public int fact(int n) {
    if (n == 1) return 1;
    return n * fact(n - 1);
}
```

• int x = fact(4);
  • return 4 * fact(3)

n=4

• return 4 * fact(3)

n=3

• return 3 * fact(2)

n=2

Don’t do this 3

```
public int fact(int n) {
    if (n == 1) return 1;
    return n * fact(n - 1);
}
```

• int x = fact(4);
  • return 4 * fact(3)

n=4

• return 3 * fact(2)

n=3

• When n is 2 ...?
  • return 2 * fact(1)

n=2

Base Case Reached

```
public int fact(int n) {
    if (n == 1) return 1;
    return n * fact(n - 1);
}
```

• return 2 * fact(1)
  • Evaluates to 2 * 1
  • Return to call of fact(1)

n=4

• return 2 * fact(1)

n=3

• return 2 * fact(1)

n=2

• return 2 * fact(1)

n=1

• return 2 * fact(1)

n=2
From PCR to linked lists

- Polymerase Chain Reaction
  - Make copies of a specific DNA segment
- Recombinant DNA
  - Insert DNA using restriction enzymes
- Loosely forms basis for DNA/Linked assignment
  - Big gains in efficiency using Linked Lists
  - Compare to array of chars, e.g. Strings

But first! Let's look at strings...

- See StringPlay.java
  - [https://coursework.cs.duke.edu/201spring20/classcode/blob/master/src/StringPlay.java](https://coursework.cs.duke.edu/201spring20/classcode/blob/master/src/StringPlay.java)
  - Runtime of `StringConcat("hello",N)`
  - Depends on size of ret: 5, 10, 15, ... 5*N
  - $5(1 + 2 + \ldots + N)$ which is $O(N^2)$

```java
public String stringConcat(String s, int reps) {
    String ret = "";
    for(int k=0; k < reps; k++) {
        ret += s;
    }
    return ret;
}
```
StringBuilder Examined

• Just say no to quadratic, use StringBuilder
  • String is immutable, StringBuilder is not
• Runtime of `builderConcat("hello", N)`
  • $5 + 5 + 5 + \ldots + 5$ a total of $N$ times: $O(N)$

```java
public String builderConcat(String s, int reps) {
    StringBuilder ret = new StringBuilder();
    for(int k=0; k < reps; k++) {
        ret.append(s);
    }
    return ret.toString();
}
```

Summary of Concatenation

• Using $s + t$ for two strings
  • Takes time $s.length() + t.length()$
  • Makes a new string, doesn't change $s$ or $t$
• Using StringBuilder is more efficient
  • Time for $s.append(t)$ is $t.length()$
  • Why? Just add $t.length()$ characters to $s$ – backed by array in $s$

Output from StringPlay

• Which is $O(N)$ and which is $O(N^2)$

<table>
<thead>
<tr>
<th>size</th>
<th>string</th>
<th>size</th>
<th>builder</th>
</tr>
</thead>
<tbody>
<tr>
<td>50000</td>
<td>0.169</td>
<td>50000</td>
<td>0.000</td>
</tr>
<tr>
<td>100000</td>
<td>0.314</td>
<td>100000</td>
<td>0.000</td>
</tr>
<tr>
<td>150000</td>
<td>0.533</td>
<td>150000</td>
<td>0.001</td>
</tr>
<tr>
<td>200000</td>
<td>0.784</td>
<td>200000</td>
<td>0.001</td>
</tr>
<tr>
<td>250000</td>
<td>1.341</td>
<td>250000</td>
<td>0.001</td>
</tr>
<tr>
<td>300000</td>
<td>1.769</td>
<td>300000</td>
<td>0.002</td>
</tr>
<tr>
<td>350000</td>
<td>2.496</td>
<td>350000</td>
<td>0.001</td>
</tr>
<tr>
<td>400000</td>
<td>3.326</td>
<td>400000</td>
<td>0.001</td>
</tr>
<tr>
<td>450000</td>
<td>4.371</td>
<td>450000</td>
<td>0.001</td>
</tr>
<tr>
<td>500000</td>
<td>5.437</td>
<td>500000</td>
<td>0.001</td>
</tr>
</tbody>
</table>

WOTO

Dr. danah boyd is a Senior Researcher at Microsoft Research, … a Visiting Professor at NYU, … Her work examines everyday practices involving social media, with specific attention to youth engagement, privacy, and risky behaviors. She recently wrote *Engaging the Ethics of Data Science in Practice*, coauthored *Isomorphism through algorithms: Institutional Dependencies in the case of Facebook*.

"Building new connections is a critical part of building a new economy. The American education system, as flawed as it is, is great for the creative class because of the way it mixes up networks."

**DNA Cut and Splice**

- Find enzyme like 'gat'
  - Replace with *splice* like 'gggtttaaa'
- Strings and StringBuilder for creating new strings
  - Complexity of "hello" + "world", or A+B
  - String: A + B, StringBuilder: B

**What do linked lists get us?**

- Faster run-time, much better use of memory
  - We splice in constant time? Re-use strings
  - Same as previous slide: sequential char view

**linked list improvement: memory**

- Suppose we have B "gat" (blue), in strand size N
  - Inserting size S "gggtttaaaa" (green) splicees
  - For String/StringBuilder: memory: B*S (+ N)
  - For LinkedList: memory: S (re-use green!) (+ N)
linked list improvement: time

- Suppose we have B "gat" (blue), in strand size N
  - Inserting size S "gggtttaaa" (green) splicees
- For String: time: $B^2S + N$, builder: $BS + N$
- For LinkedList: $B + N$

Theory and Practice

- The JVM can sometimes optimize your code
  - Don’t optimize what you don’t have to …

- Timings with `System.nanoTime()` are suspect
  - Other things going on in computer
  - JVM can go into garbage-collection mode
  - Other considerations

Thoughts on Exam 1

- Exam 1 – 80 points

Survey on Exam 1

- Review Grades for [Exam 1 CompSci 201 Spring 2020](#)
  - REGRADE REQUESTS
    - OPEN
    - GRADERS
    - PUBLISHED
  - MIN: 27.5  |  MEDIAN: 61.0  |  MAX: 78.5  |  MEAN: 60.65  |  STD DEV: 9.94
Big-Oh Questions

- Need to explain every line
- We will do big-Oh almost every day
- The more you do the better you will get at these

Storage Question

- Class has state – four items define an object
  - Int mySize
  - Int myCapacity
  - String[] myItems
  - HashSet<String> myUniqueItems
- Methods should update state appropriately
- Lot of points, but broken into small parts

Exam 1 Takeaways

- First, understand everything you missed
  - Get a blank sheet of paper, can you now write the code.
  - Need to do this before moving on
- Exam and solutions are on the Old Tests page
- Come in for office hours – go over your exam, concepts you are not solid on
  - Go on nights when an assignment/APT not due
  - Free four hours a night office hours

More Exam 1 Takeaways

- Midterm grades – most will pull up your grade with other things
  - A range – 119
  - B range – 135
  - C range – 30
- Help
  - Understand what you missed
  - Consulting/Office hours – it is free!
  - Peer Tutoring Center – group tutoring