Compsci 201
Lists, Trees, Recursion

Susan Rodger
March 4, 2020

Announcements

• APT 4 due yesterday! Still turn in today no penalty

• Assignment P4 DNA-Link
  • Part 1 due March 5 – Analysis, Partner form
  • Part 2 due March 19 – Code and more Analysis

• APT Quiz 1 – now on regular APT page
  • Not for credit, but finish if you didn’t

O is for …

• Object Oriented
  • Programming with inheritance and classes

• Open Source
  • Copyright meets the Creative Commons

• O-Notation
  • Measuring in the limit

Plan for WBSB

• Problem Solving: from Interviews to APTs
  • Example of an algorithmic concept or two

• DNA Catchup and Review
  • Part 1 and Strings/StringBuilder

• Recursion Review
  • When recursion makes code more simple
  • When recursion makes code more complex?

• Binary Trees
  • Search and more: best of array and linked lists
    • O(1) insert and O(log n) search
Want to read more on topics?

• Free online textbooks from OpenDSA
  • http://lti.cs.vt.edu/OpenDSA/Books/CS2
  • http://lti.cs.vt.edu/OpenDSA/Books/CS3

• Topics
  • See chapters on Linked Lists, Recursion, Trees

Interview Interlude (à la 201)

• Length of longest substring no repeated chars
  • https://leetcode.com/problems/longest-substring-without-repeating-characters/
  • Example: longest("abcdafgb")
  • Returns 6, since "bcdafg"
  • Example: longest("abcdafgbdch")
  • Returns 7, since "afgbdch"

• Make it run, make it right, make it fast
  • Finally? Make it small, e.g., mobile?

Thinking about it…

• Do you have to look at every substring?

• Could you write code that looks at every possible substring?
  • Look at every starting character
  • And for that char look at every possible ending character
  • Nested loop
Java-isms from $N^2$ solution

- Need to know index bounds on substring(x,y)
  - Conveniently length is y-x
  - Because y not included: substring(0,4)

- Primitive and Wrapper classes: char/Character
  - No easy way to use Arrays.asList to add all characters to a set 😊
  - But Wait!!! `Arrays.asList(s.split(""))`
    - Let's not revel in one-liners, not the point at all

Goal: linear solution $O(n)$

- Technique called "sliding window", develop with invariant to help reason about correctness
  - You shouldn't be expected to do on the fly
  - What if interviewer gives you help?

<table>
<thead>
<tr>
<th>'a'</th>
<th>'b'</th>
<th>'c'</th>
<th>'d'</th>
<th>'e'</th>
<th>'f'</th>
<th>'g'</th>
<th>'b'</th>
<th>'d'</th>
<th>'c'</th>
<th>'h'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<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>
</tr>
</tbody>
</table>

3/4/2020 Compsci 201, Spring 2020

Sliding Window Technique

- The window has a start index, initially 0
  - If repeat char?
    - Slide start

<table>
<thead>
<tr>
<th>'a'</th>
<th>'b'</th>
<th>'c'</th>
<th>'d'</th>
<th>'e'</th>
<th>'f'</th>
<th>'g'</th>
<th>'b'</th>
<th>'d'</th>
<th>'c'</th>
<th>'h'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<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>
</tr>
</tbody>
</table>

Developing Window with Invariant

- Slide start to where?
  - Reset start to index after duplicate char
- Invariant: characters in [start,index) are unique
  - We want to extend window: index$^th$ ch in window?
- Invariant: map.get(ch) == index of last ch
  - If duplicate in window? Reset window

<table>
<thead>
<tr>
<th>'a'</th>
<th>'b'</th>
<th>'c'</th>
<th>'d'</th>
<th>'e'</th>
<th>'f'</th>
<th>'g'</th>
<th>'b'</th>
<th>'d'</th>
<th>'c'</th>
<th>'h'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<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>
</tr>
</tbody>
</table>

3/4/2020 Compsci 201, Spring 2020
Developing Window with Invariant

- **Invariant**: characters in \([\text{start}, \text{index})\) are unique
  - Goal: extend window, what about \(\text{index}^{th}\) char
  - **Invariant**: \(\text{map.get(ch)} == \text{index of last ch}\)
  - **Invariant**: max is length of longest window

- Before loop: \(\text{start} = 0, \text{index} = 1, [0,1)\) unique
  - \(\text{map.put(s.charAt(0), 0)}\)

- As \(\text{index}\) increases from 1, 2, 3, ...
  - If unique? Keep invariant: \(\text{map.put(ch, index)}\)
  - If not unique? Reset start/window, remap

Linus Torvalds

- **Created Linux and Git**
  - 2012 Millennium Prize
  - 2014 IEEE Pioneer Award

This week people in our community confronted me about my lifetime of not understanding emotions. My flippant attacks in emails have been both unprofessional and uncalled for. Especially at times when I made it personal. In my quest for a better patch, this made sense to me. I know now this was not OK and I am truly sorry.

Linux Kernel Mailing List, 9/16/2018

WOTO


https://www.youtube.com/watch?v=5EGx4_WSMSE

Come to my window

LinkedList and linked list

- The former implements the java.util.List interface
  - Uses linked lists internally
  - \(O(1)\) add/remove at back or at front

- Nodes for linked lists
  - Insert/splice is \(O(1)\)
  - Always has \(\text{.next, sometimes has .prev}\)

Linked lists

In the real world:

Find (remove) minimal node in list

- [https://www2.cs.duke.edu/csed/newapt/removemin.html](https://www2.cs.duke.edu/csed/newapt/removemin.html)
- We can't do better than $O(n)$ for n-node list
  - Simple: find minimal node, then traverse to it
    - Remove requires linking around minimal node
- How to find minimal node?
  - Typical min over a sequence algorithm OR
  - Use recursion leveraging list structure

RemoveMin: First find min: Pass one

RemoveMin: Find node BEFORE min: Pass two

RemoveMin: REMOVE MIN
SPECIAL CASE: Min is first node

```
min: 2
```

Iterative Find Minimal Node

- If current value < minimal value? Update current
  - Canonical list traversal, null pointer exception?

```java
private ListNode findMin(ListNode list) {
    if (list == null || list.next == null) return list;
    ListNode after = findMin(list.next);
    if (list.info < after.info) {
        min = list;
    }
    list = list.next;
    return min;
}
```

Recursive Find Minimal Node

- What is the base case: typically 0 or 1 node list
- Otherwise: make recursive call, use that result
  - Find minimal node after first (recursively)
    - Can there be null pointer on after.info?
  - Compare to first, return final value

```java
private ListNode findMinRec(ListNode list) {
    if (list == null || list.next == null) return list;
    ListNode after = findMinRec(list.next);
    if (list.info < after.info) return list;
    return after;
}
```
Finishing Remove Min

- Advance current to the node before min
  - Then link around min node

```
31       ListNode current = list;
32       while (current.next != min) {
33           current = current.next;
34       }
35       current.next = min.next;
```

- Lists often have special case: no before node

```
28       if (min == list) {
29           return list.next;
30       }
```

- This is O(N) to find min and O(N) to remove
  - Can we do in one pass? Yes, keep prevNode pointer

Revisit: Understanding Recursion

- Visualized last time
  - The base case anchors the recursion
- There's no loop! Why?
  - Sequence of recursive calls
  - Stacked up until base returns
- The recursive call "decreases"
  - Must get toward base case

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

How did recursion work?

- Structure of a linked list is essential
  - For a non-null list:
    - compute # nodes OR find minimal node
- Use result of this call and treatment of first node
  - Count me: add one to result
  - Am I smaller? Return me, else return you
What is the DNA/LinkStrand assignment about?
- Why do we study linked lists?
- How do you work in a group?

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 "gggtttaaa" (green) splicees
  - For String/StringBuilder: memory: B*S (+ N)
  - For LinkedList: memory: S (re-use green!) (+ N)
    - B nodes used too

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

Let's look at strings... AGAIN

- See 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 + ... + 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;
  }
  ```

stringConcat(stringConcat("a",N),N)

- Length returned: `StringConcat("a",N)`
  - `ret.length() == N` first time
  - Then 2N, then 3N, then ...
  - $N(1 + 2 + ... + N)$ which is $O(N^3)$
  - Time for a + b string concat? $O(|a| + |b|)$

  ```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();
}
```

Bad Trees and Good Trees

Trees are Best of Both Worlds

• With arrays we can use binary search
  • This is $O(\log N)$, that’s really, really fast
  • Remember that $2^{10} = 1024$ so …
    • Search a billion sorted items with 30 comparisons
• With linked lists we can add/remove quickly
  • Cannot search, cannot index, can relink
• Can we get fast search and fast add/remove?

Binary Search Trees

• Nodes have left/right references: similar prev/next
  • At each node: $\leq$ goes left, $>$ goes right
• How do we search?
• How do we insert?
• Insert: “koala”
Tree Terminology

- **Root**: "top node", has no parent
  - "macaque". Subtrees also have a root
- **Leaf**: bottom nodes, have no children
  - "baboon", "lemur", "organutan"
- **Path**: sequence of parent-child nodes
  - "macaque", "chimp", "lemur"

Trees: Concepts and Code

- In a search tree: property holds at every node
  - Nodes in left subtree are < (or <=)
  - Nodes in right subtree are >

- To search or add: if not found?
  - Look left if <=
  - Look right if >
  - Iterative or recursive

A TreeNode by any other name…

- What does this look like? Doubly linked list?

```java
public class TreeNode {
    TreeNode left;  // left child
    TreeNode right; // right child
    String info;    // info
    TreeNode(String s, TreeNode llink, TreeNode rlink) {
        info = s;
        left = llink;
        right = rlink;
    }
}
```

Trees Performance

- Search for any value. Compare to root and …
- Similar to binary search. \(O(\log N)\) if tree "good"
  - Trees are generally well-behaved, but !!!
  - Guarantee? Balanced tree: AVL or Red-Black

- We get \(O(\log N)\) search and …
  - No shifting to add, find leaf
Good Search Trees and Bad Trees

http://www.9wy.net/onlinebook/CPrimerPlus5/ch17lev1sec7.html