Homework 6

Due Monday, June 28th at 11:59 PM ET.
Late passes CANNOT be used on this homework.

Fun Sea Creature Fact: Electric eels can produce enough electricity to light 10 light bulbs!

Installing and Handing In

Accept the GitHub assignment **here**. Homework is handed in through Gradescope. For written portion, submit a PDF and match the questions accordingly. For the coding portion, submit ALL of the .py files.

1 Written Problems

Problem 6.1

Wheezie the Dragon

Purpose: More practice with dynamic programming
Resources: Dynamic Programming Lecture (May 25) and Section 2 Slides

Provide pseudocode for the most time-efficient algorithm you would use to solve the following problem. Your solution should run in $O(n)$. It should be clear and comprehensive.

On a promotional tour for her latest movie lasting n days in total, Wheezie the dragon knows that if she eats at *The Magic Scale Cafe* on day i she will make $d(i)$ dollars (some number of dollars, depending on the day). On this tour, she wants to rest for *at least* k days after every time she eats at the cafe. Describe an algorithm to find the best days for Wheezie to eat on her tour.

As an example, consider $n = 5$ and $k = 2$. Let the $d(i)$s be:
$d(1) = \$3, \quad d(2) = \$4, \quad d(3) = \$7, \quad d(4) = \$1, \quad d(5) = \$7.$
In this case, Wheezie can earn the most money by eating at the cafe on days 2 and 5, earning a total of \$11. Note that because $k = 2$, she is taking off 2 days (day 3 and day 4) between eating (though she could take off more if that was optimal).

Be sure to describe the input and output of your procedure, pay attention to indentation so that we understand the logic of your code, and use comments to clarify as needed. Double check that your pseudocode follows the pseudocode guidelines on the site.

Problem 6.2

Justifying Big-O

Purpose: More practice with runtime and Big-O analysis
Resources: Analysis Big-O Lecture (May 20), and all subsequent lectures that analyze runtimes of given algorithms

Wheezie's friend Sally the Shark is on the same promotional tour, but is rather impatient. We have provided pseudocode for her greedy solution below.

Justify the big-O runtime of her algorithm. (Hint: think about what a worst case input would look like.)

```
procedure findDaysToEat($n$, $d$, $k$):
Input: Takes in total number of days $n$, an array $d$ corresponding to how
       much she can earn on each day, and the least number of days $k$
       she needs to rest after eating.
       Note: The length of array $d$ should be $n$.
             The day count starts at 0.

Output: An array containing the days she should eat.

$eatDays$ = []
$index$ = 0
$currStartIndex = 0$

while $currStartindex < n$:
    $max$ = -infinity
    for $j = currStartIndex$ to $n - 1$:
        if $d[j] > max$:
            $max = d[j]$
            $index = j$
    $eatDays$.append($index$)
    $currStartIndex = index + k + 1$


return $eatDays$
```

2 Python Problems

Problem 6.3

Increment that

Purpose: More practice with recursion in python
Resources: Recursion Lecture (May 25)

Implement a *recursive* Python function called `increment(number)` that takes in a stack of 0's and 1's representing a binary number k, and returns another stack representing the binary number $k+1$. You should use a Python list as your stack representation, but you are only allowed to use the functions `len()`, `pop()` and `append()` (Python's version of `push()`).

Your code should be as neat and simple as possible (our solution is about 9 lines). You may mutate the input list directly. Note: there are several tricky edge cases to consider, so make sure to hand simulate your algorithm thoroughly!

In case you're unsure of how to add binary numbers, here are some links to help you out:

Wikihow.com/Add-Binary-Numbers
Courses.cs.vt.edu/AddingTwoBinaryNumbers

Remember to do this recursively and that you may not use list slicing! If you don't remember what a recursive function is, look back to the lecture on Recurrence, Induction, and Dynamic Programming!

Examples

- `increment([1,0,0,0])` $\rightarrow$ `[1,0,0,1]`
- `increment([1])` $\rightarrow$ `[1,0]`
- `increment([0,0,1])` $\rightarrow$ `[0,1,0]`

Testing

Write tests in the provided file `increment_test.py` to make sure your algorithm works. **DO NOT** write your tests within the example test functions we provide! Our scripts will skip the test functions we provide, so write your own functions to test your code thoroughly. To help you generate test cases, we've provided a helper function called `strToList(string)` which takes in a string of 0's and 1's and returns a list that you can input to your `increment` function. Note: you may assume that if your input list is not null or empty, then it contains only 0's and 1's. Don't forget to check for invalid lists (None and `[]`), though!

Example

- `strToList("1010")` → `[1,0,1,0]`

Problem 6.4

Binary Tree Traversals

Purpose: Hands-on python practice with binary tree traversals
Resources: Trees and Traversals Lecture (June 3)

Implement a preorder, inorder, postorder, and breadth-first traversal of a binary tree in Python. The `bt` object passed in to the traversals is a TA implementation of a binary tree.

Requirements

Each function should make a list of the nodes of a binary tree in the order of the respective traversal. Feel free to add helper methods to complete this task. You will use the TA binary tree implementation from Homework 5, and you are free to use Python's built-in Queue, or you can achieve the same Queue functionality (without importing anything) by instantiating a list and calling `list.insert(0, "hello")` and `list.pop(0)`.

As a reminder these are the functions you implemented when you created a binary tree in Homework 5, and therefore have access to: `parent, children, root, left, right, addRight, addLeft, hasRight, hasLeft, value, depth, addRoot,isEmpty, size, height,isInternal, isExternal, isRoot`

Functions

- preorder(bt)
- inorder(bt)
- postorder(bt)
- breadthfirst(bt)

Testing

You know the drill. Write tests in the provided file `traversals_test.py` and make sure your stuff works! **DO NOT** write your tests within the example test functions we provide! Our scripts will skip the test functions we provide, so write your own functions to test your code thoroughly. Don't forget (again) to raise an error for invalid inputs. You may assume, however, that if your input is not null, it is a tree.