**[Does This Linked List Have A Cycle?](https://t.dripemail2.com/c/eyJhY2NvdW50X2lkIjoiNDc3NDM4NCIsImRlbGl2ZXJ5X2lkIjoibXkwNmJqaWl3b3ptYzNsa3ZvMzQiLCJ1cmwiOiJodHRwczovL3d3dy5pbnRlcnZpZXdjYWtlLmNvbS9xdWVzdGlvbi9saW5rZWQtbGlzdC1jeWNsZXM_dXRtX3NvdXJjZT13ZWVrbHlfZW1haWxcdTAwMjZ1dG1fY2FtcGFpZ249d2Vla2x5X2VtYWlsXHUwMDI2dXRtX21lZGl1bT1lbWFpbFx1MDAyNl9fcz16c2Nib3NnZXF5Y3E4azg4cGFrZVx1MDAyNnV0bV9zb3VyY2U9ZHJpcFx1MDAyNnV0bV9tZWRpdW09ZW1haWxcdTAwMjZ1dG1fY2FtcGFpZ249SW50ZXJ2aWV3K0Nha2UrV2Vla2x5K1Byb2JsZW0rJTIzMjkwJTNBK0RvZXMrVGhpcytMaW5rZWQrTGlzdCtIYXZlK0ErQ3ljbGUlM0YifQ" \t "_blank)**

**You have a singly-linked list ↴ and want to check if it contains a cycle.**

A singly-linked list is built with nodes, where each node has:

* node.next—the next node in the list.
* node.value—the data held in the node. For example, if our linked list stores people in line at the movies, node.value might be the person's name.

For example:

class LinkedListNode(object):

def \_\_init\_\_(self, value):

self.value = value

self.next = None

A **cycle** occurs when a node’s next points back to a previous node in the list. The linked list is no longer linear with a beginning and end—instead, it cycles through a loop of nodes.

**Write a function contains\_cycle() that takes the first node in a singly-linked list and returns a boolean indicating whether the list contains a cycle.**

**Breakdown**

Because a cycle could result from the last node linking to the first node, we might need to look at every node before we even see the start of our cycle again. So it seems like we can’t do better than O(n)O(n) runtime.

How can we track the nodes we’ve already seen?

**Using a set, we could store all the nodes we’ve seen so far**. The algorithm is simple:

1. If the current node is already in our set, we have a cycle. Return True.
2. If the current node is None we've hit the end of the list. Return False.
3. Else throw the current node in our set and keep going.

What are the time and space costs of this approach? Can we do better?

**Find Duplicates**

**You left your computer unlocked and your friend decided to troll you by copying a lot of your files to random spots all over your file system.**

Even worse, she saved the duplicate files with random, embarrassing names ("this\_is\_like\_a\_digital\_wedgie.txt" was clever, I'll give her that).

Write a function that returns a list of all the duplicate files. We'll check them by hand before actually deleting them, since programmatically deleting files is really scary. To help us confirm that two files are actually duplicates, return a list of tuples ↴ where:

* the **first** item is the **duplicate** file
* the **second** item is the **original** file

For example:

[('/tmp/parker\_is\_dumb.mpg', '/home/parker/secret\_puppy\_dance.mpg'),

('/home/trololol.mov', '/etc/apache2/httpd.conf')]

You can assume each file was only duplicated once.

**Breakdown**

**No idea where to start?** Try writing something that just walks through a file system and prints all the file names. If you're not sure how to do that, look it up! Or just *make it up*. Remember, even if you can’t implement *working code*, your interviewer will still want to see you *think through* the problem.

One brute force ↴ solution is to loop over all files in the file system, and for each file look at every *other* file to see if it's a duplicate. This means n^2*n*2 file comparisons, where n*n* is the number of files. That seems like a lot.

Let's try to save some time. Can we do this in *one* walk through our file system?

Instead of holding onto one file and looking for files that are the same, we can just keep track of *all* the files we've seen so far. What data structure could help us with that?

**Reverse Words**

**You're working on a secret team solving coded transmissions.**

Your team is scrambling to decipher a recent message, worried it's a plot to break into a major European National Cake Vault. The message has been *mostly* deciphered, but all the words are backward! Your colleagues have handed off the last step to you.

Write a function reverse\_words() that takes a message as a list of characters and reverses the order of the words in place. ↴

Why a list of characters instead of a string?

The goal of this question is to practice manipulating strings *in place*. Since we're modifying the message, we need a **mutable ↴**type like a list, instead of Python 2.7's *immutable* strings.

For example:

message = [ 'c', 'a', 'k', 'e', ' ',

'p', 'o', 'u', 'n', 'd', ' ',

's', 't', 'e', 'a', 'l' ]

reverse\_words(message)

# Prints: 'steal pound cake'

print ''.join(message)

When writing your function, assume the message contains only letters and spaces, and all words are separated by one space.

**Graph Coloring**

**Given an undirected graph ↴ with maximum degree ↴ D*D*, find a graph coloring ↴ using at most D+1*D*+1 colors.**

For example:

This graph's maximum degree (D*D*) is 3, so we have 4 colors (D+1*D*+1). Here's one possible coloring:

Graphs are represented by a list of N*N* node objects, each with a label, a set of neighbors, and a color:

class GraphNode:

def \_\_init\_\_(self, label):

self.label = label

self.neighbors = set()

self.color = None

a = GraphNode('a')

b = GraphNode('b')

c = GraphNode('c')

a.neighbors.add(b)

b.neighbors.add(a)

b.neighbors.add(c)

c.neighbors.add(b)

graph = [a, b, c]

**Write a method to find the 2nd largest element in a binary search tree. ↴**

Here's a sample binary tree node class:

public class BinaryTreeNode {

public int value;

public BinaryTreeNode left;

public BinaryTreeNode right;

public BinaryTreeNode(int value) {

this.value = value;

}

public BinaryTreeNode insertLeft(int leftValue) {

this.left = new BinaryTreeNode(leftValue);

return this.left;

}

public BinaryTreeNode insertRight(int rightValue) {

this.right = new BinaryTreeNode(rightValue);

return this.right;

}

}

**You're working with an intern that keeps coming to you with JavaScript code that won't run because the braces, brackets, and parentheses are off. To save you both some time, you decide to write a braces/brackets/parentheses validator.**

Let's say:

* '(', '{', '[' are called "*openers*."
* ')', '}', ']' are called "*closers*."

Write an efficient method that tells us whether or not an input string's openers and closers are properly nested.

Examples:

* "{ [ ] ( ) }" should return **true**
* "{ [ ( ] ) }" should return **false**
* "{ [ }" should return **false**

**In order to win the prize for most cookies sold, my friend Alice and I are going to merge our Girl Scout Cookies orders and enter as one unit.**

Each order is represented by an "order id" (an integer).

We have our lists of orders sorted numerically already, in arrays. Write a method to merge our arrays of orders into one sorted array.

For example:

int[] myArray = new int[]{3, 4, 6, 10, 11, 15};

int[] alicesArray = new int[]{1, 5, 8, 12, 14, 19};

System.out.println(Arrays.toString(mergeArrays(myArray, alicesArray)));

// prints [1, 3, 4, 5, 6, 8, 10, 11, 12, 14, 15, 19]

**You have a linked list ↴**

Write a method kthToLastNode() that takes an integer kkk and the headNode of a singly-linked list, and returns the kkkth to last node in the list.

For example:

public class LinkedListNode {

public String value;

public LinkedListNode next;

public LinkedListNode(String value) {

this.value = value;

}

}

LinkedListNode a = new LinkedListNode("Angel Food");

LinkedListNode b = new LinkedListNode("Bundt");

LinkedListNode c = new LinkedListNode("Cheese");

LinkedListNode d = new LinkedListNode("Devil's Food");

LinkedListNode e = new LinkedListNode("Eccles");

a.next = b;

b.next = c;

c.next = d;

d.next = e;

kthToLastNode(2, a);

// returns the node with value "Devil's Food" (the 2nd to last node)

**Write a function for doing an in-place ↴ shuffle of a list.**

The shuffle must be "uniform," meaning each item in the original list must have the same probability of ending up in each spot in the final list.

Assume that you have a function get\_random(floor, ceiling) for getting a random integer that is >= floor and <= ceiling.

**Delete a node from a singly-linked list, ↴ given only a variable pointing to that node.**

The input could, for example, be the variable b below:

class LinkedListNode(object):

def \_\_init\_\_(self, value):

self.value = value

self.next = None

a = LinkedListNode('A')

b = LinkedListNode('B')

c = LinkedListNode('C')

a.next = b

b.next = c

delete\_node(b)

Y**ou are a renowned thief who has recently switched from stealing precious metals to stealing cakes because of the insane profit margins. You end up hitting the jackpot, breaking into the world's largest privately owned stock of cakes—the vault of the Queen of England.**

While Queen Elizabeth has a *limited number of types of cake*, she has an *unlimited supply of each type*.

Each type of cake has a weight and a value, stored in a tuple with two indices:

1. An integer representing the **weight** of the cake in kilograms
2. An integer representing the **monetary value** of the cake in British shillings

For example:

# Weighs 7 kilograms and has a value of 160 shillings

(7, 160)

# Weighs 3 kilograms and has a value of 90 shillings

(3, 90)

You brought a duffel bag that can hold limited weight, and you want to make off with the most valuable haul possible.

Write a function max\_duffel\_bag\_value() that takes **a list of cake type tuples**and **a weight capacity**, and returns **the *maximum monetary value* the duffel bag can hold.**

For example:

cake\_tuples = [(7, 160), (3, 90), (2, 15)]

capacity = 20

# Returns 555 (6 of the middle type of cake and 1 of the last type of cake)

max\_duffel\_bag\_value(cake\_tuples, capacity)

**Weights and values may be any non-negative integer.** Yes, it's weird to think about cakes that weigh nothing or duffel bags that can't hold anything. But we're not just super mastermind criminals—we're also meticulous about keeping our algorithms flexible and comprehensive.

**Your company delivers breakfast via autonomous quadcopter drones. And something mysterious has happened.**

Each breakfast delivery is assigned a unique ID, a positive integer. When one of the company's 100 drones takes off with a delivery, the delivery's ID is added to a list, delivery\_id\_confirmations. When the drone comes back and lands, the ID is again added to the same list.

After breakfast this morning there were only 99 drones on the tarmac. One of the drones never made it back from a delivery. **We suspect a secret agent from Amazon placed an order and stole one of our patented drones**. To track them down, we need to find their delivery ID.

**Given the list of IDs, which contains many duplicate integers and one unique integer, find the unique integer.**

The IDs are ***not*** guaranteed to be sorted or sequential. Orders aren't always fulfilled in the order they were received, and some deliveries get cancelled before takeoff.