IT5003: Data Structure and Algorithm

Semester I, 2019/2020

Practical Exam (30th November, 2019)

Task 1. Candy Crush [10 marks]

Let's play a game of **candy crush**, shall we? In this game, you are given a list of candies (represented as **integers between 1 to 4, inclusive**). These candies has an interesting property: *when the same kind of candy meets, they will be "crushed" into one and become the next higher candy* (i.e. candy *3* **meets** candy *3* ➔ candy *4*). Of course, two candies of "4" will just wraps around and become **candy *1***. Note that this process continues if the new candy meet candy of the same type right after crushing.

Your task is to use a **stack** to simulate this candy game and generate the final result. **Restriction: you can use only a single stack and use NO MORE than O(1) extra memory space.** The skeleton code has setup all the necessary input / output, your task is simply to put in the candy crushing algorithm. The output is generated from the stack simply by popping the candies from top to bottom.

Submit **only the `candyCrush`**() function.

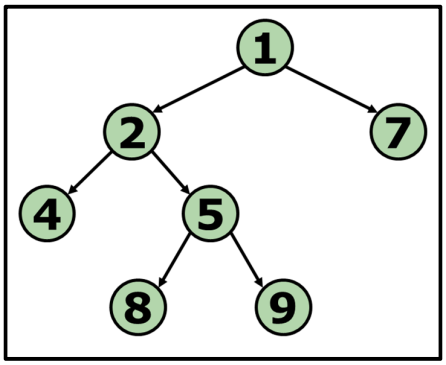
Sample Runs:

Given List	Result	Explanation
`[1, 2, 3, 4]`	`[4, 3, 2, 1]`	No crushing of candy, output is simply the candies in reverse order.
`[1, 1]`	`[2]`	1 and 1 crushed into 2
`[4, 4]`	`[1]`	4 and 4 crushed into 1
`[3, 1, 1, 3]`	`[3, 2, 3]`	1 and 1 crushed into 2
`[2, 1, 1]`	`[3]`	1 and 1 crushed into 2, then 2 and 2 crushed into 3
`[4, 3, 2, 1, 1]`	`[1]`	1 and 1 crushed into 2, then 2 and 2 crushed into 3, then 3 and 3 crushed into 4, then 4 and 4 crushed into 1

Task 2. The many paths of BT [10 marks]

This questions looks at **paths** (from root node to leaf node) in a **binary tree**. There are **three level of difficulties** for this task. **You only need to attempt ONE of them**. If you attempt more than one level of difficulty, we will only check the **HIGHEST LEVEL difficulty** that you managed to get completely correct. NO partial mark for higher / lower difficulty level that is incomplete or not fully correct.

Let's use binary tree on the right to illustrate the 3 level of difficulties.



Level 1 (**5 marks**): Return the length of the **longest path** in the binary tree
Method: `_pathV1(self, ...)`
Sample output with the binary tree shown: `4` **Explanation:** The longest paths from root to any leaf are either "1-2-5-8" and "1-2-5-9", both are of length **4**. This version always return a **single integer**.

Level 2 (**8 marks**): Return length of **ALL paths** in the binary tree as a list. The values are ordered by the order of the leaf nodes from left to right in the tree.
Method: `_pathV2(self, ...)`
Sample output with the binary tree shown: `[3, 4, 4, 2]` **Explanation:** From left to right, the paths to leaf nodes are "1-2-4" (length 3), "1-2-5-8" (length 4) "1-2-5-9" (length 4) and "1-7" (length 2). This version always return a list of integers.

Level 3 (**10 marks + 2 bonus marks**): Return length of the **ALL paths and the paths themselves** in the binary tree as a list.
Method: `_pathV3(self, ...)`
Sample output with the binary tree shown: `[ (3, '4-2-1'), (4, '8-5-2-1'), (4, '9-5-2-1'), (2, '7-1')]` Explanation: Essentially an enhanced version of two. Every path is captured as a 2-tuple, where the first value is the length, and the second value is the path **from leaf to root as a <u>string</u>.** This version always return a list of 2-tuple.

About implementations:

You are given a **`BTSimple`** class, which is a cut-down version of a complete binary tree implementation. The method skeleton for all 3 versions are given. Note that each version takes the form of a wrapper method and the actual recursive internal helper. e.g.:

```
    #this is the internal helper method
    def _pathV1( self ):
        pass  #change to your implementation

    #this is the wrapper method
    def pathV1(self):
        return self._pathV1( )
```

IMPORTANT: You CAN change the internal helper method parameters (i.e. takes in more argument if needed). However, the wrapper method parameter MUST stay the same i.e. take in ONLY "self" as shown above.

~~~ End of PE Question ~~~