CCC Junior Lesson 7

Derrick Guo

Note: This course note is prepared based on questions from www.leetcode.com, under the answer of each question, you can find a reference number of that question on leetcode. When you teach students using slides, let them go to leetcode.com and copy there solutions there to run tests against their programs. That way we can provide better test cases in both quantity and quality.

1 Roman to Integer

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

For example, two is written as II in Roman numeral, just two one's added together. Twelve is written as, XII, which is simply X + II. The number twenty seven is written as XXVII, which is XX + V + II.
Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:
I can be placed before V (5) and X (10) to make 4 and 9.
X can be placed before L (50) and C (100) to make 40 and 90.
C can be placed before D (500) and M (1000) to make 400 and 900.
Given a roman numeral, convert it to an integer. Input is guaranteed to be within the range from 1 to 3999.

Example1:

```
Input: "III"
Output: 3
```

Example2:

```
Input: "IV"
Output: 4
```

Example3:

```
Input: "IX"
Output: 9
```

Example4:

```
Input: "LVIII"
Output: 58
Explanation: L = 50, V= 5, III = 3.
```

Example5:

```
Input: "MCMXCIV"
Output: 1994
Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.
```

Idea:

Defining a dictionary is very helpful in this case. Use the roman letter as key and its corresponding numerical value as value.

Answer:

```
def romanToInt(self, s):
    d = {'I':1, 'V':5, 'X':10, 'L':50, 'C':100, 'D':500, 'M':1000}
    r = d[s[len(s)-1]]
    for i in range(len(s)-1):
        if d[s[i]] < d[s[i+1]]:
            r -= d[s[i]]
        else:
            r += d[s[i]]
    return r
```

Reference: leetcode #13

2 Longest Common Prefix

Write a function to find the longest common prefix string amongst an array of strings.
If there is no common prefix, return an empty string "".

Example1:

```
Input: ["flower","flow","flight"]
Output: "fl"
```

Example2:

```
Input: ["dog","racecar","car"]
Output: ""
Explanation: There is no common prefix among the input strings.
```

Idea:

Solve this using loops. Exit whenever there is something that don't match.

Answer:

```
class Solution(object):
    def longestCommonPrefix(self, strs):
        if not strs: return ''
        first = min(strs)
        for i in range(len(first)):
            for s in strs:
                if s[i] != first[i]:
                    return first[:i] if i > 0 else ''
        return first
```

Reference: leetcode #14

3 Reverse String

Write a function that reverses a string. The input string is given as an array of characters char[].

Do not allocate extra space for another array, you must do this by modifying the input array in-place with O(1) extra memory.
You may assume all the characters consist of printable ascii characters.

Example 1:

```
Input: ["h","e","l","l","o"]
Output: ["o","l","l","e","h"]
```

Example 2:

```
Input: ["H","a","n","n","a","h"]
Output: ["h","a","n","n","a","H"]
```

Idea:

Use a loop to add the string in reverse order.

Answer:

```
class Solution:
    def reverseString(self, s):
        out = ""
        for i in range(len(s)):
            out = out + s[len(s)-i-1]
        return out
```

Reference: leetcode #344

4 First Unique Character in a String

Given a string, find the first non-repeating character in it and return it's index. If it doesn't exist, return -1.
Note: You may assume the string contain only lowercase letters.

Example 1:

```
s = "leetcode"
return 0.
```

```
s = "loveleetcode",
return 2.
```

Idea:

You can go through the string twice to solve this. First time find number of appearance for each character and store it in a dictionary, or (if you prefer array) you can create an array that has size 26 (matching 26 alphabets) and for example increment array[0] if you see an appearance of 'a'. Then go through the string again to find the first unique char.

Answer:

```
class Solution:
    def firstUniqChar(self, s):
    if len(s) == 0:
        return -1
    if len(s) == len(set(s)):
        return 0
    map = {}
    for letter in s:
        if letter not in map:
            map[letter] = 1
        else:
            map[letter] += 1
    for index, letter in enumerate(s):
    # no need to use enumerate,
    # you can just use for i in s, then check for map[i]
        if map[letter] == 1:
            return index
    return -1
```

Reference: leetcode #387

5 Robot Return to Origin

There is a robot starting at position (0, 0), the origin, on a 2D plane. Given a sequence of its moves, judge if this robot ends up at (0, 0) after it completes its moves.
The move sequence is represented by a string, and the character moves[i] represents its ith move. Valid moves are R (right), L (left), U (up), and D (down). If the robot returns to the origin after it finishes all of its moves, return true. Otherwise, return false.
Note: The way that the robot is "facing" is irrelevant. "R" will always make the robot move to the right once, "L" will always make it move left, etc. Also,

assume that the magnitude of the robot's movement is the same for each move.

Example 1:

```
Input: "UD"
Output: true
Explanation: The robot moves up once, and then down once. All moves have the sam
```

Example 2:

```
Input: "LL"
Output: false
Explanation: The robot moves left twice. It ends up two "moves" to the left of t
```

Idea:

Count the appearence of each 'U', 'D', 'L', 'R' and see if they can cancel.

Answer:

```
def judgeCircle(self, moves):
    return moves.count('L') == moves.count('R') and moves.count('U') == moves.co
```

This is an really tricky solution that can be applied to First Unique Character in a String, too. (With worse performance though). You can also do this question by going through the string twice like you did in question 4.

Reference: leetcode # 657

6 Add Binary

Given two binary strings, return their sum (also a binary string). The input strings are both non-empty and contains only characters 1 or 0.

Example 1:

```
Input: a = "11", b = "1"
Output: "100"
```

Example 2:

```
Input: a = "1010", b = "1011"
Output: "10101"
```

Idea:

Use recursion. Define a add function with extra parameters carryover and current result and use them for addition.

Answer:

```
def add(a,b,carryover,result):
if len(a)>0 and len(b)>0:
    sum = int(a[0])+int(b[0])+carryover
    carryover = sum / 2
    result += str(sum % 2)
    a = a[1:]
    b = b[1:]
    return add(a,b,carryover,result)
elif len(a)>0:
    if carryover==1:
        sum = int(a[0])+carryover
        carryover = sum / 2
        result +=  str(sum % 2)
        a = a[1:]
        return add(a,b,carryover,result)
    else:
        result += a
        return result
elif len(b)>0:
    if carryover==1:
        sum = int(b[0])+carryover
        carryover = sum / 2
        result +=  str(sum % 2)
        b = b[1:]
        return add(a,b,carryover,result)
    else:
        result += b
        return result
else:
    if carryover==1:
        result += '1'
    return result


class Solution:
    # @param a, a string
    # @param b, a string
    # @return a string
```

```
def addBinary(self, a, b):
    a = a[::-1]
    b = b[::-1]
    sum = add(a,b,0,'')[::-1]
    return sum
```

Reference: leetcode #67
Emphasize on recursion :)

7 Valid Parentheses

Given a string containing just the characters '(', ')', '', '', '[' and ']', determine if the input string is valid.

An input string is valid if:

Open brackets must be closed by the same type of brackets.
Open brackets must be closed in the correct order.
Note that an empty string is also considered valid.

Example 1:

```
Input: "()"
Output: true
```

Example 2:

```
Input: "()[]{}"
Output: true
```

Example 3:

```
Input: "(]"
Output: false
```

Example 4:

```
Input: "([)]"
Output: false
```

Example 5:

```
Input: "{[]}"
Output: true
```

Idea:

Use a stack to track the number of open braces and close braces. Careful when encounter the cases that an close brace shows up before any of its corresponding open brace (especially when student uses counters)

Answer:

```
class Solution:
        # @param {string} s
        # @return {boolean}
        def isValid(self, s):
            stack=[]
            for i in s:
                if i in ['(','[','{']:
                    stack.append(i)
                else:
                    if not stack or {')':'(',']':'[','}':'{'}[i]!=stack[-1]:
                        return False
                    stack.pop()
            return not stack
```

Reference: leetcode #20