CCC Junior Lesson 8

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 Reverse Integer

Given a 32-bit signed integer, reverse digits of an integer.

Example1:

```
Input: 123
Output: 321
```

Example2:

```
Input: -123
Output: -321
```

Example3:

```
Input: 120
Output: 21
```

Idea:

Use a loop. First get the sign of the whole thing then use its absolute value. It's easier than having two cases where you can do addition when it's positive and subtraction when it's negative, but that is also doable. Keep in mind to check overflow (should not be needed for CCC though).

Answer:

Uforse Education

Copyright@Uforse Education Group Inc.英狮教育集团 uforseteam@gmail.com 6472396116 wechatID: iloveufose

```
class Solution(object):
    def reverse(self, x):
        sign = -1 if x<0 else 1
        res = 0
        x = abs(x)
        while x:
            res = res*10 + (x%10)
            x /= 10
        # handle the overflow bound
        if res > 2**31+1 or res < -2**31-1:
            return 0
        return res*sign
```

Reference: leetcode #7

2 Palindrome Number

Determine whether an integer is a palindrome. An integer is a palindrome when it reads the same backward as forward.

Example1:

```
Input: 121
Output: true
```

Example2:

```
Input: -121
Output: false
Explanation: From left to right, it reads -121. From right to left,
```

Example2:

```
Input: 10
Output: false
Explanation: Reads 01 from right to left. Therefore it is not a palin
```

Idea:

Use a loop to construct the second half of the given number in the opposite

Uforse Education

Copyright@Uforse Education Group Inc.英狮教育集团 uforseteam@gmail.com 6472396116 wechatID: iloveufose

order of its first half and see if the are the same. Also need to check cases like 121 where the first half is 12 and second half is also 21. Thus checking x in (half, half / 10) makes things a lot easier.

Answer:

```
class Solution(object):
    def isPalindrome(self, x):
        if x < 0 or (x > 0 and x % 10 == 0): return False
        half = 0
        while x > half:
            half, x = half * 10 + x % 10, x / 10
        return x in (half, half / 10)
```

Reference: leetcode #9

3 Happy Number

Write an algorithm to determine if a number is "happy".
A happy number is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1. Those numbers for which this process ends in 1 are happy numbers.

Example:

```
Input: 19
Output: true
Explanation:
1^2 + 9^2 = 82
8^2 + 2^2 = 68
6^2 + 8^2 = 100
1^2 + 0^2 + 0^2 = 1
```

Idea:

Use a set then use a loop. In the loop, first check if the given number n is in the set. If so, you have an infinite loop then you need to exit. Otherwise you need to add n to the set for later use of checking infinite loop. Then calculate n for the next round. Exit when you get 1.

Uforse Education

Copyright@Uforse Education Group Inc.英狮教育集团 uforseteam@gmail.com 6472396116 wechatID: iloveufose

Answer:

```
class Solution(object):
    def isHappy(self, n):
        """
        :type n: int
        :rtype: bool
        """
        Hash = {}
        while n != 1:
            if n in Hash: return False
            else: Hash[n] = True
            n = sum([pow(int(c), 2) for c in str(n)])
        return True
```

Reference: leetcode #202

4 Count Primes

Count the number of prime numbers less than a non-negative number, n.

Example 1:

```
Input: 10
Output: 4
Explanation: There are 4 prime numbers less than 10, they are 2, 3,
```

Idea:

Use a list of bool to store if its index is a prime number, then starts from 2, once you find a prime number, put all of its duplicate to false.

Answer:

```
def countPrimes(self, n):
    if n <= 2:
        return 0
    res = [True] * n
    res[0] = res[1] = False
    for i in xrange(2, n):
        if res[i] == True:
```

Uforse Education

Copyright@Uforse Education Group Inc.英狮教育集团 uforseteam@gmail.com 6472396116 wechatID: iloveufose

```
                for j in xrange(2, (n-1)//i+1):
                    res[i*j] = False
        return sum(res)
```

Reference: leetcode #204

5 Ugly Number

Write a program to check whether a given number is an ugly number.
Ugly numbers are positive numbers whose prime factors only include 2, 3, 5.

Example 1:

```
Input: 6
Output: true
Explanation: 6 = 2     3
```

Example 2:

```
Input: 8
Output: true
Explanation: 8 = 2     2     2
```

Example 3:

```
Input: 14
Output: false
Explanation: 14 is not ugly since it includes another prime factor 7.
```

Idea:

dividing 2, 3, 5 from the given number as many times as you can, and see if it gets to 1 at the end.

Answer:

```
class Solution(object):
    def isUgly(self, num):
        for p in 2, 3, 5:
            while num % p == 0 < num:
                num /= p
        return num == 1
```

Reference: leetcode # 263

Uforse Education

Copyright@Uforse Education Group Inc.英狮教育集团 uforseteam@gmail.com 6472396116 wechatID: iloveufose

6 Missing Number

Given an array containing n distinct numbers taken from 0, 1, 2, ..., n, find the one that is missing from the array.
Note:
Your algorithm should run in linear runtime complexity. Could you implement it using only constant extra space complexity?
Example 1:

```
Input: [3,0,1]
Output: 2
```

Example 2:

```
Input: [9,6,4,2,3,5,7,0,1]
Output: 8
```

Idea:

Sum up the whole list and 1...n, minus the latter by the former gives you what you want.

Answer:

```
class Solution(object):
    def missingNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        n = len(nums)
        return n * (n+1) / 2 - sum(nums)
```

Reference: leetcode #268

7 Integer Break

Given a positive integer n, break it into the sum of at least two positive integers and maximize the product of those integers. Return

Uforse Education

Copyright@Uforse Education Group Inc.英狮教育集团 uforseteam@gmail.com 6472396116 wechatID: iloveufose

the maximum product you can get.

Example 1:

```
Input: 2
Output: 1
Explanation: 2 = 1 + 1, 1 * 1 = 1.
```

Example 2:

```
Input: 10
Output: 36
Explanation: 10 = 3 + 3 + 4, 3 * 3 * 4 = 36.
```

Idea: math behind the scenes

I saw many solutions were referring to factors of 2 and 3. But why these two magic numbers? Why other factors do not work? Let's study the math behind it.

For convenience, say n is sufficiently large and can be broken into any smaller real positive numbers. We now try to calculate which real number generates the largest product. Assume we break n into (n / x) x's, then the product will be $x^{n/x}$, and we want to maximize it.

Taking its derivative gives us $n * x^{n/x-2}$ * (1 - ln(x)). The derivative is positive when $0 < x < e$, and equal to 0 when x = e, then becomes negative when $x > e$, which indicates that the product increases as x increases, then reaches its maximum when $x = e$, then starts dropping.

This reveals the fact that if n is sufficiently large and we are allowed to break n into real numbers, the best idea is to break it into nearly all e's. On the other hand, if n is sufficiently large and we can only break n into integers, we should choose integers that are closer to e. The only potential candidates are 2 and 3 since $2 < e < 3$, but we will generally prefer 3 to 2. Why?

Of course, one can prove it based on the formula above, but there is a more natural way shown as follows. $6 = 2 + 2 + 2 = 3 + 3$. But $2*2*2 < 3*3$. Therefore, if there are three 2's in the decomposition, we can replace them by two 3's to gain a larger product. All the analysis above assumes n is significantly large. When n is small (say n ¡= 10), it may contain flaws. For instance, when n = 4, we have

$2 * 2 > 3 * 1$. To fix it, we keep breaking n into 3's until n gets smaller than 10, then solve the problem by brute-force.

Answer:

```
class Solution(object):
    def integerBreak(self, n):
        if n == 2 or n == 3:
            return n - 1
        if n % 3 == 0:
            return 3**(n/3)
        if n % 3 == 1:
            return 3**(n/3 - 1)*4
        if n % 3 == 2:
            return 3**(n/3)*2
```

Reference: leetcode #343

Uforse Education

Copyright@Uforse Education Group Inc.英狮教育集团 uforseteam@gmail.com 6472396116 wechatID: iloveufose