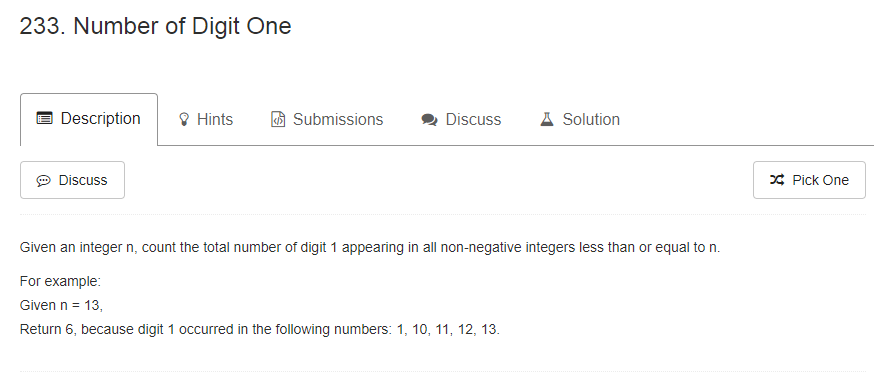
Day19. 今天上傳LeetCode失敗惹－LeetCode\_Number of Digit One

今天老樣子LeetCode

(LeetCode變老樣子了嗎XDD



廢話我就不多說惹，今天的題目長這樣



簡單來說找出一個數字以內，他的組成元素有1的有幾個數字。

EX: 1 算是1個、2不算、3不算、11~19算

這樣，很簡單吧!

現在就來拆一下題目吧。

1. 將數字拆解->toString方法
2. 判斷該元素是否包含字元”1”
3. 計算包含元素有幾個

一開始我們先來寫一個判斷元素是否包含1方法的測試吧!

就先從0開始做輸入吧!

```

[TestMethod]

public void HasDigitOne\_Input\_0\_Should\_Be\_false()

{

Assert.IsFalse(Solution.HasDigitOne(0));

}

```

而Production Code 也就是老樣子會長成這個樣子

```

public static bool HasDigitOne(int i)

{

throw new System.**NotImplementedException**();

}  
```

老樣子，跑個測試，沒過很正常，紅燈，commit一下

接下來把Production Code改一下，用最簡單的方式解決他!

```

public static bool HasDigitOne(int i)

{

return false;

}  
```

接下來跑個測試，PASS! Commit~

接下來寫一個輸入0過了，就來寫一下輸入1囉

```

[TestMethod]

public void HasDigitOne\_Input\_1\_Should\_Be\_True()

{

Assert.IsTrue(Solution.HasDigitOne(1));

}

```

而Production Code長這樣

```

public static bool HasDigitOne(int i)

{

if (i.ToString().Contains("1"))

{

return true;

}

return false;

```

再來寫10位數的HasDigitOne的測試!

```

[TestMethod]

public void HasDigitOne\_Input\_10\_Should\_Be\_True()

{

Assert.IsTrue(Solution.HasDigitOne(10));

}

```

一樣是過了，基本上所有的輸入都可以進行判斷了，接下來寫Production Code的測試吧!

這一個測試就是要去計算這個數以下的正整數有幾個包含1了

```

[TestMethod]

public void Input\_1\_Should\_Be\_1()

{

Assert.AreEqual(1,Solution.CountDigitOne(1));

}

```

再來就是改寫Production Code啦

```

public static int CountDigitOne(int n)

{

var result = 0;

for (int i = 1; i <= n; i++)

{

if (HasDigitOne(i))

{

result++;

}

}

return result;

}

```

接下來跑一下測試吧，綠燈! Commit~

基本上目前的Code已經涵蓋到所有輸入了。

所以我們來改一下Production Code吧!

像是HasDigitOne方法可以改成這個樣子

```

public static bool HasDigitOne(int i)

{

return i.ToString().Contains("1");

}

```

改完不要忘記跑測試，Pass，Commit!

再來就是改CountDigitOne方法啦!

我們可以用Enumerable的Range方法來跑迴圈，加上我們剛才寫好的方法就可以用一行解決我們的問題囉!

```

public static int CountDigitOne(int n)

{

return Enumerable.Range(1, n).Count(HasDigitOne);

}

```

接下來就可以提交到LeetCode上了，不過他沒有過，因為他有一個測試試輸入負號的，這時候自己家一個測試案例是負號的然後補上Production Code就可以了

這是最後的Production Code

```

public class Solution

{

public static bool HasDigitOne(int i)

{

return i.ToString().Contains("1");

}

public static int CountDigitOne(int n)

{

return n > 0 ? Enumerable.Range(1, n).Count(HasDigitOne) : 0;

}

}

```

原本以為這樣就結束了，提交上去還是錯的，原來他是計算他的元素”1”有幾個，而不是他元素有1的有幾個，沒關係，接下來我們可以改一下Code。

不過這時候測試案例也須要維護了，就從原本的測試案例開始修正吧!

前3個測試案例就要改成這個樣子，我們需要先計算他

```

[TestMethod]

public void HasDigitOne\_Input\_0\_Should\_Be\_false()

{

Assert.AreEqual(0, Solution.HasDigitOne(0));

}

[TestMethod]

public void HasDigitOne\_Input\_1\_Should\_Be\_True()

{

Assert.AreEqual(1, Solution.HasDigitOne(1));

}

[TestMethod]

public void HasDigitOne\_Input\_10\_Should\_Be\_True()

{

Assert.AreEqual(1, Solution.HasDigitOne(10));

}

```

所以HasDigitOne的Code就變成要使用Count計算他

```

public static int HasDigitOne(int i)

{

return i.ToString().Count(x => x == '1');

}

```

而Production Code理所當然地也不能使用Count了，而是要使用Sum

所以CountDigitOne變成這個樣子

```

public static int CountDigitOne(int n)

{

return n > 0 ? Enumerable.Range(1, n).Sum(HasDigitOne) : 0;

}

```

以下是今天所有的測試案例

```

[TestClass]

public class UnitTest1

{

[TestMethod]

public void HasDigitOne\_Input\_0\_Should\_Be\_false()

{

Assert.IsFalse(Solution.HasDigitOne(0));

}

[TestMethod]

public void HasDigitOne\_Input\_1\_Should\_Be\_True()

{

Assert.IsTrue(Solution.HasDigitOne(1));

}

[TestMethod]

public void HasDigitOne\_Input\_10\_Should\_Be\_True()

{

Assert.IsTrue(Solution.HasDigitOne(10));

}

[TestMethod]

public void Input\_1\_Should\_Be\_1()

{

Assert.AreEqual(1, Solution.CountDigitOne(1));

}

[TestMethod]

public void Input\_10\_Should\_Be\_2()

{

Assert.AreEqual(2, Solution.CountDigitOne(10));

}

[TestMethod]

public void Input\_f1\_Should\_Be\_0()

{

Assert.AreEqual(0,Solution.CountDigitOne(-1));

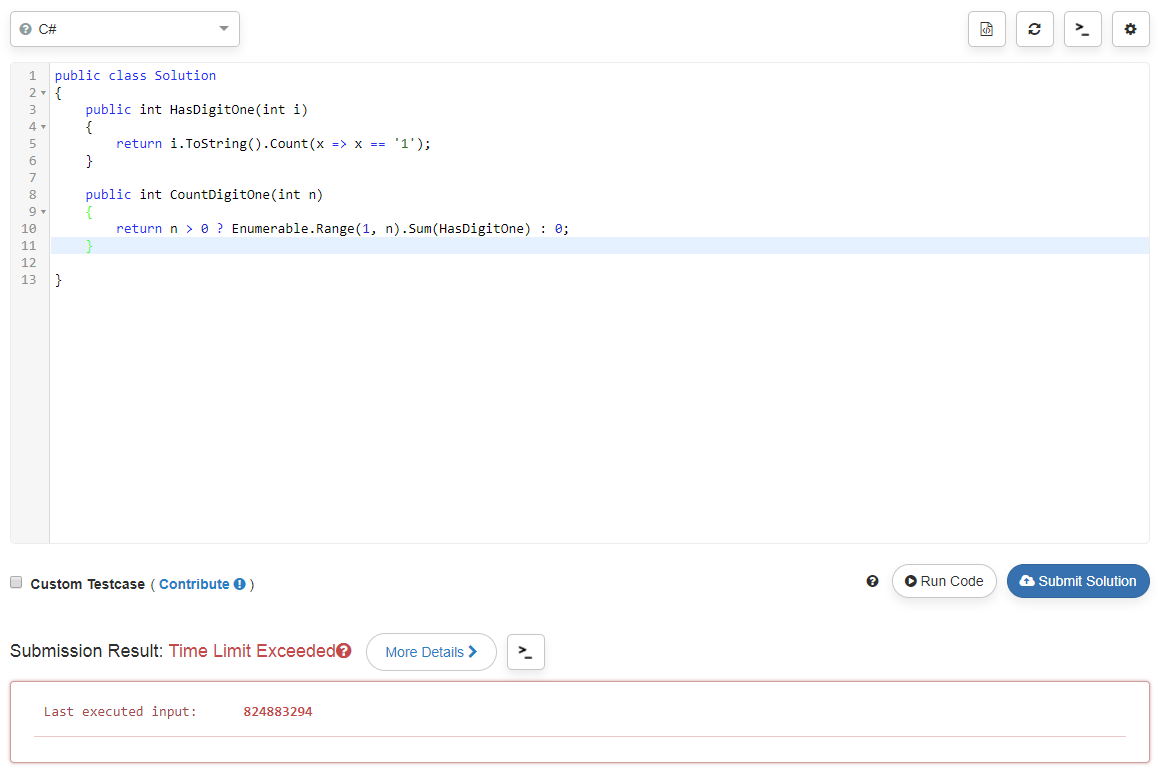
}

```

然後在LeetCode上提交!

還是不過啊!!!!!!!

為甚麼呢XDDD



今天的題目太過於考演算法啦QQ

小弟我演算法苦手

根本GG

沒辦法把這麼大的數處理好

不過基本上我拆解題目的方式是沒有錯的

但沒有考慮到時間複雜度的問題

或許今年可以好好考慮研究一下演算法的東西呢

不過今天除了LeetCode沒過這點之外

還有一件有趣的是，今天在實作的過程，也因為自己的測試Code而讓自己改程式碼改得理所當然、理直氣壯的不會怕改了A又壞了B之類的

但相對的也顯現了測試程式碼是需要被維護的，今天需求一但一開始就理解錯了，就連測試程式碼都會寫錯喔。

PS: 大家都按這題不喜歡倒是讓我欣慰了不少(被揍



Git url :

<https://github.com/SQZ777/LeetCode_NumberOfDigitOne>

LeetCode Link:

<https://leetcode.com/problems/number-of-digit-one/description/>

下一題，明天見!

今天老樣子LeetCode

(LeetCode變老樣子了嗎XDD

![https://ithelp.ithome.com.tw/upload/images/20180104/20107209ZbfvlPhu1R.png](https://ithelp.ithome.com.tw/upload/images/20180104/20107209ZbfvlPhu1R.png)

廢話我就不多說惹，今天的題目長這樣

![https://ithelp.ithome.com.tw/upload/images/20180104/20107209d98EWNy5Sn.png](https://ithelp.ithome.com.tw/upload/images/20180104/20107209d98EWNy5Sn.png)

簡單來說找出一個數字以內，他的組成元素有1的有幾個數字。

EX: 1 算是1個、2不算、3不算、11~19算

這樣，很簡單吧!

現在就來拆一下題目吧。

1. 將數字拆解->toString方法

2. 判斷該元素是否包含字元”1”

3. 計算包含元素有幾個

一開始我們先來寫一個判斷元素是否包含1方法的測試吧!

就先從0開始做輸入吧!

```

[TestMethod]

public void HasDigitOne\_Input\_0\_Should\_Be\_false()

{

Assert.IsFalse(Solution.HasDigitOne(0));

}

```

而Production Code 也就是老樣子會長成這個樣子

```

public static bool HasDigitOne(int i)

{

throw new System.NotImplementedException();

}

```

老樣子，跑個測試，沒過很正常，紅燈，commit一下

接下來把Production Code改一下，用最簡單的方式解決他!

```

public static bool HasDigitOne(int i)

{

return false;

}

```

接下來跑個測試，PASS! Commit~

接下來寫一個輸入0過了，就來寫一下輸入1囉

```

[TestMethod]

public void HasDigitOne\_Input\_1\_Should\_Be\_True()

{

Assert.IsTrue(Solution.HasDigitOne(1));

}

```

而Production Code長這樣

```

public static bool HasDigitOne(int i)

{

if (i.ToString().Contains("1"))

{

return true;

}

return false;

```

再來寫10位數的HasDigitOne的測試!

```

[TestMethod]

public void HasDigitOne\_Input\_10\_Should\_Be\_True()

{

Assert.IsTrue(Solution.HasDigitOne(10));

}

```

一樣是過了，基本上所有的輸入都可以進行判斷了，接下來寫Production Code的測試吧!

這一個測試就是要去計算這個數以下的正整數有幾個包含1了

```

[TestMethod]

public void Input\_1\_Should\_Be\_1()

{

Assert.AreEqual(1,Solution.CountDigitOne(1));

}

```

再來就是改寫Production Code啦

```

public static int CountDigitOne(int n)

{

var result = 0;

for (int i = 1; i <= n; i++)

{

if (HasDigitOne(i))

{

result++;

}

}

return result;

}

```

接下來跑一下測試吧，綠燈! Commit~

基本上目前的Code已經涵蓋到所有輸入了。

所以我們來改一下Production Code吧!

像是HasDigitOne方法可以改成這個樣子

```

public static bool HasDigitOne(int i)

{

return i.ToString().Contains("1");

}

```

改完不要忘記跑測試，Pass，Commit!

再來就是改CountDigitOne方法啦!

我們可以用Enumerable的Range方法來跑迴圈，加上我們剛才寫好的方法就可以用一行解決我們的問題囉!

```

public static int CountDigitOne(int n)

{

return Enumerable.Range(1, n).Count(HasDigitOne);

}

```

接下來就可以提交到LeetCode上了，不過他沒有過，因為他有一個測試試輸入負號的，這時候自己家一個測試案例是負號的然後補上Production Code就可以了

這是最後的Production Code

```

public class Solution

{

public static bool HasDigitOne(int i)

{

return i.ToString().Contains("1");

}

public static int CountDigitOne(int n)

{

return n > 0 ? Enumerable.Range(1, n).Count(HasDigitOne) : 0;

}

}

```

原本以為這樣就結束了，提交上去還是錯的，原來他是計算他的元素”1”有幾個，而不是他元素有1的有幾個，沒關係，接下來我們可以改一下Code。

不過這時候測試案例也須要維護了，就從原本的測試案例開始修正吧!

前3個測試案例就要改成這個樣子，我們需要先計算他

```

[TestMethod]

public void HasDigitOne\_Input\_0\_Should\_Be\_false()

{

Assert.AreEqual(0, Solution.HasDigitOne(0));

}

[TestMethod]

public void HasDigitOne\_Input\_1\_Should\_Be\_True()

{

Assert.AreEqual(1, Solution.HasDigitOne(1));

}

[TestMethod]

public void HasDigitOne\_Input\_10\_Should\_Be\_True()

{

Assert.AreEqual(1, Solution.HasDigitOne(10));

}

```

所以HasDigitOne的Code就變成要使用Count計算他

```

public static int HasDigitOne(int i)

{

return i.ToString().Count(x => x == '1');

}

```

而Production Code理所當然地也不能使用Count了，而是要使用Sum

所以CountDigitOne變成這個樣子

```

public static int CountDigitOne(int n)

{

return n > 0 ? Enumerable.Range(1, n).Sum(HasDigitOne) : 0;

}

```

以下是今天所有的測試案例

```

[TestClass]

public class UnitTest1

{

[TestMethod]

public void HasDigitOne\_Input\_0\_Should\_Be\_false()

{

Assert.IsFalse(Solution.HasDigitOne(0));

}

[TestMethod]

public void HasDigitOne\_Input\_1\_Should\_Be\_True()

{

Assert.IsTrue(Solution.HasDigitOne(1));

}

[TestMethod]

public void HasDigitOne\_Input\_10\_Should\_Be\_True()

{

Assert.IsTrue(Solution.HasDigitOne(10));

}

[TestMethod]

public void Input\_1\_Should\_Be\_1()

{

Assert.AreEqual(1, Solution.CountDigitOne(1));

}

[TestMethod]

public void Input\_10\_Should\_Be\_2()

{

Assert.AreEqual(2, Solution.CountDigitOne(10));

}

[TestMethod]

public void Input\_f1\_Should\_Be\_0()

{

Assert.AreEqual(0,Solution.CountDigitOne(-1));

}

```

然後在LeetCode上提交!

還是不過啊!!!!!!!

為甚麼呢XDDD

![https://ithelp.ithome.com.tw/upload/images/20180104/20107209xxP7Oy8xQl.png](https://ithelp.ithome.com.tw/upload/images/20180104/20107209xxP7Oy8xQl.png)

今天的題目太過於考演算法啦QQ

小弟我演算法苦手

根本GG

沒辦法把這麼大的數處理好

不過基本上我拆解題目的方式是沒有錯的

但沒有考慮到時間複雜度的問題

或許今年可以好好考慮研究一下演算法的東西呢

不過今天除了LeetCode沒過這點之外

還有一件有趣的是，今天在實作的過程，也因為自己的測試Code而讓自己改程式碼改得理所當然、理直氣壯的不會怕改了A又壞了B之類的

但相對的也顯現了測試程式碼是需要被維護的，今天需求一但一開始就理解錯了，就連測試程式碼都會寫錯喔。

PS: 大家都按這題不喜歡倒是讓我欣慰了不少(被揍

![https://ithelp.ithome.com.tw/upload/images/20180104/20107209OSOAR0CmsN.png](https://ithelp.ithome.com.tw/upload/images/20180104/20107209OSOAR0CmsN.png)

Git url :

https://github.com/SQZ777/LeetCode\_NumberOfDigitOne

LeetCode Link:

https://leetcode.com/problems/number-of-digit-one/description/

下一題，明天見!