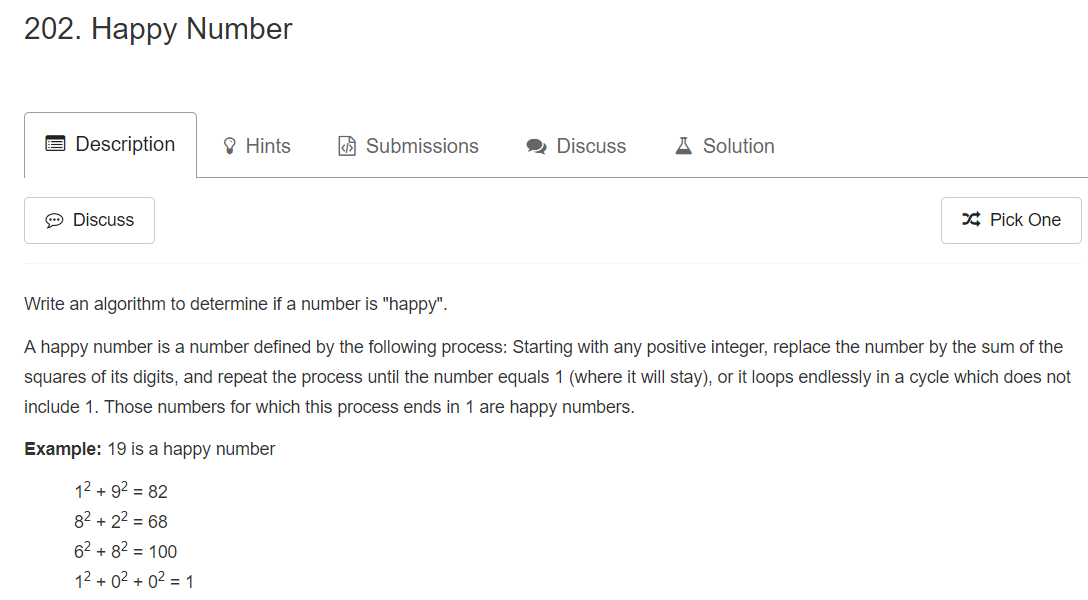
Day17. 安安，你快樂嗎?－LeetCode\_Happy Number

今天剛好有人請教我這題程式題，就來寫這個題目吧XDDD



廢話我就不多說惹，今天的題目長這樣



題目需求蠻簡單的，看起來就是把數字拆解然後把拆解出來的數字分別平方出來，最後拆出來的數字如果等於1進入無窮迴圈時

他就是快樂數了

至於他哪裏快樂我就不知道了，1這個數字看起來很孤單啊看起來都不快樂(被揍

現在就來拆一下題目吧。

1. 將數字拆解->toString方法
2. 拆解出來的數字平方並加總
3. 重複了沒? => 所以要將加總過的數字加入某個陣列進行判斷
4. 判斷重複可以用Contains方法

現在就來想一下最簡單的測試案例吧!

就先從0開始做輸入吧!

```

[TestMethod]

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

{

Assert.AreEqual(false,Solution.IsHappy(0));

}

```

而Production Code 也就是老樣子會長成這個樣子

```

public class Solution

{

public static bool IsHappy(int n)

{

throw new **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 IsHappy(int n)

{

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 Input\_1\_Should\_Be\_True()

{

Assert.AreEqual(true, Solution.IsHappy(1));

}

```

一樣是過了，因為原本的Production Code涵蓋了輸入0的需求。

所以我們現在來輸入1。

```

public static bool IsHappy(int n)

{

if (n == 1)

{

return true;

}

return false;

}

```

接下來跑一下測試吧，綠燈! Commit~

再來要寫一個拆解數字成字串然後將其中每一個數字平方加總後的方法

所以要寫這個方法的測試

```

[TestMethod]

public void Pow\_Input\_2\_Should\_Be\_4()

{

Assert.AreEqual(4,Solution.Pow(2));

}

```

然後加上這個方法

```

public static int Pow(int n)

{

return n \* n;

}

```

再來寫一個要切割字串並加總其所有元素平方的測試

```

[TestMethod]

public void SumAll\_Input\_10\_Should\_Be\_1()

{

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

}

```

然後再加一個方法SumAll的Code如下

```

public static int SumAll(int n)

{

var result = 0;

foreach (var c in Pow(n).ToString())

{

result += Pow(Convert.ToInt32(c.ToString()));

}

return result;

}

```

接下來要把剛才生出來的兩個方法加進我的Production Code並應用囉!

然後要完成新的需求

「重複了沒」

要建立一個新的int List並且用Contains判斷他重複過了沒

所以我們來寫個輸入10的測試吧!

```

[TestMethod]

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

{

Assert.AreEqual(true, Solution.IsHappy(10));

}

```

而Production Code長這樣，因為要顧慮到輸入為0的狀況，所以在最前面加上了一個判斷。

```

public static bool IsHappy(int n)

{

if (n == 0)

return false;

var recordList = new List<int>();

while (!recordList.Contains(SumAll(n)) || SumAll(n) != 1)

{

recordList.Add(SumAll(n));

n = SumAll(n);

}

if (n == 1)

return true;

return false;

}

```

寫完就來跑個測試，PASS，Commit一下唄!

再來寫一個輸入7的測試。

```

[TestMethod]

public void Input\_7\_Should\_Be\_True()

{

Assert.AreEqual(true,Solution.IsHappy(7));

}

```

很好 他進入了無窮迴圈了，因為我們將職責分離的關係，所以很輕鬆的可以知道是SumAll這個方法出了問題。

於是重新審視了Code發現SumAll這個方法裡面不小心多用了Pow的方法了，導致會沒辦法有輪迴且數字會越來越大。

所以將foreach(var c in Pow(n).toString())改成下面這個樣子

```

public static int SumAll(int n)

{

var result = 0;

foreach (var c in n.ToString())

{

result += Pow(Convert.ToInt32(c.ToString()));

}

return result;

}

```

接下來再寫一個輸入2的測試吧!

```

[TestMethod]

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

{

Assert.AreEqual(false, Solution.IsHappy(2));

}

```

會發現我又進入無窮迴圈了，這一次看起來就不像是SumAll中發生的問題，而是while迴圈發生了無窮迴圈，就發現了剛才自己不小心多打了一個or的判斷，導致他永遠都出不來的問題。

所以將Production Code中的n!=1去除。

```

public static bool IsHappy(int n)

{

if (n == 0)

return false;

var recordList = new List<int>();

while (!recordList.Contains(SumAll(n)))

{

recordList.Add(SumAll(n));

n = SumAll(n);

}

if (n == 1)

return true;

return false;

}

```

基本上目前的程式碼已經可以涵蓋到所有範圍，所以我們就多加幾個測試案例吧!

以下是今天所有的測試程式碼

```

[TestClass]

public class UnitTest1

{

[TestMethod]

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

{

Assert.AreEqual(false, Solution.IsHappy(0))

}

[TestMethod]

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

{

Assert.AreEqual(true, Solution.IsHappy(1));

}

[TestMethod]

public void Pow\_Input\_2\_Should\_Be\_4()

{

Assert.AreEqual(4, Solution.Pow(2));

}

[TestMethod]

public void SumAll\_Input\_10\_Should\_Be\_1()

{

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

}

[TestMethod]

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

{

Assert.AreEqual(true, Solution.IsHappy(10))

}

[TestMethod]

public void Input\_7\_Should\_Be\_True()

{

Assert.AreEqual(true, Solution.IsHappy(7));

}

[TestMethod]

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

{

Assert.AreEqual(false, Solution.IsHappy(2))

}

[TestMethod]

public void Input\_19\_Should\_Be\_True()

{

Assert.AreEqual(true,Solution.IsHappy(19));

}

[TestMethod]

public void Input\_4\_Should\_Be\_false()

{

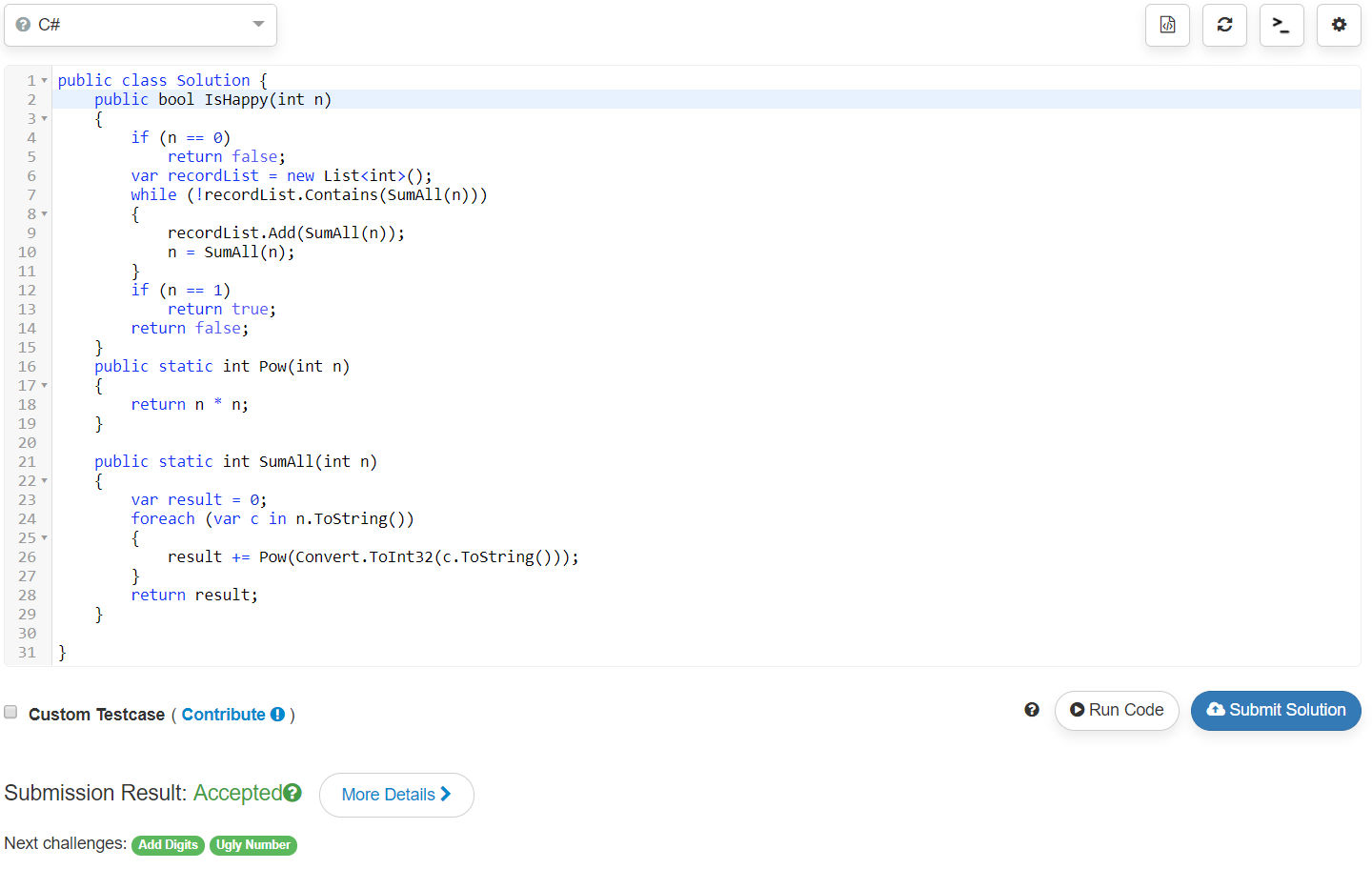
Assert.AreEqual(false,Solution.IsHappy(4));

}

}

```

然後在LeetCode上提交! Pass!!



今天的題目真的蠻有趣的XD

可以把一些職責分離

重點是測試一下子就可以幫助我釐清有沒有出錯

如果出錯了，我寫出來的Code可以很快速地幫助我理解哪裏出了問題。

超級棒的啦!!

Git url :

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

LeetCode Link:

<https://leetcode.com/problems/happy-number/description/>

下一題，明天見!