Task1-Q2

2019年10月11日　0:59

分别用递归法和非递归法求Fibonacci数列的前1000位，并比较计算时间的差异.

斐波那契数列（Fibonacci sequence），又称黄金分割数列、因数学家列昂纳多·斐波那契（Leonardoda Fibonacci）以兔子繁殖为例子而引入，故又称为“兔子数列”，指的是这样一个数列：1、1、2、3、5、8、13、21、34、……在数学上，斐波那契数列以如下被以递推的方法定义：F(1)=1，F(2)=1, F(n)=F(n-1)+F(n-2)（n>=3n∈N*）在现代物理、准晶体结构、化学等领域，斐波纳契数列都有直接的应用

- **非递归方法**

  这种方法又叫做递推法，就是递增法。时间复杂度为O（n），呈线性增长，如果数据巨大，速度越拖越慢。比较好理解就，就是c=a+b，运算完成后，重新赋值。

  源代码：

```
#non_recursion
def fib_loop(n):
    a = 0
    b = 1
    c = 0
    for i in range(n):
        c = a + b
        a = b
        b = c
    return a

num_b = int(input("input a number : "))
if num_b==0:
    print("error!")
else:
    print(fib_loop(num_b))
```

运行截图：

随机数值0 1 30 1000

```
input a number : 0
error!
```

```
input a number : 1
1
```

```
input a number : 30
832040
```

```
input a number : 1000
43466557686937456435688527675040625802564660517371
```

由于第一千个数的数值太大显示不完全

- **递归法**

  递归法也比较好理解，从第三项开始，每一项数字都是前面两项数字的和，这种方式效率不是很高，因为有大量的重复计算，如f(5)需要计算f(3)与f(4)，f(6)又计算f(5)，f(5)又计算f(4)....，时间复杂度为`O(2^n)`。

  源代码：

```
#recursion
def fib_recursion(n):
    if  n == 1:
        return 1
    elif n == 2:
        return 1
    else:
        return fib_recursion(n-1) + fib_recursion(n-2)

num_a = int(input("input a number : "))
if num_a==0:
    print("error!")
else:
    print(fib_recursion(num_a))
```

运行截图：

测试如上随机数0 1 30 1000

```
input a number : 0
error!
```

```
input a number : 1
1
```

```
input a number : 30
832040
```

```
    if  n == 1:
RecursionError: maximum recursion depth exceeded in comparison
```

当计算第1000个数的时候我们会发现报错，超过最大递归深度。但是经过其他随机数的测试可得code并无错误，但是该怎么解决还在思考中。尝试计算第999个数计算时间非常长.

利用datetime模块可以计算并比较递归与非递归的时间差异

e.g. input a number：35 //已经将时间放大1000倍

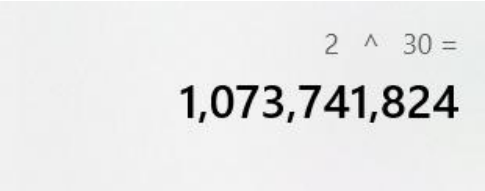
```
C:\Users\79453\Anaconda3\python.exe "D:/华师工程中心/研一/课程
 算法设计/coding tests/Fibonacci.py"
input a number : 30
832040
time : 500.0
input a number : 30
832040
time : 0.0
```

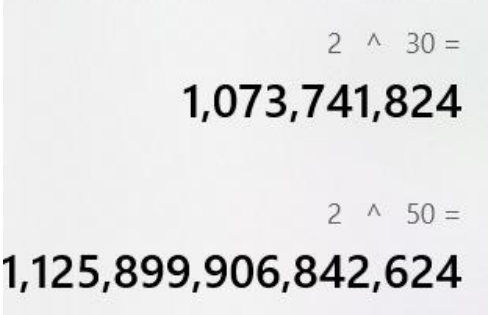
结果显而易见 非递归时间复杂度比递归时间复杂更低

- 类实现内部魔方方法

```python
class Fibonacci(object):
    """斐波那契数列迭代器"""

    def __init__(self, n):
        """
        :param n:int 指 生成数列的个数
        """
        self.n = n
        # 保存当前生成到的数据列的第几个数据，生成器中性质，记录位置，下一个位置的数据
        self.current = 0
        # 两个初始值
        self.a = 0
        self.b = 1

    def __next__(self):
        """当使用next()函数调用时，就会获取下一个数"""
        if self.current < self.n:
            self.a, self.b = self.b, self.a + self.b
            self.current += 1
            return self.a
        else:
            raise StopIteration

    def __iter__(self):
        """迭代器的__iter__ 返回自身即可"""
        return self


if __name__ == '__main__':
    num_c = int(input("input a number : "))
    fib = Fibonacci(num_c)
    for num in fib:
        print(num)
```

后记：老师更改要求 求出第五十个斐波那契数 其实50层迭代的斐波那契计算也需要非常长的时间……





第50个数的计算时间是第30个数的计算时间的1,048,576倍，
i7-7700HQ大概需要计算145小时