Python迭代与递归算法

广州协和学校信息技术科

2024年12月

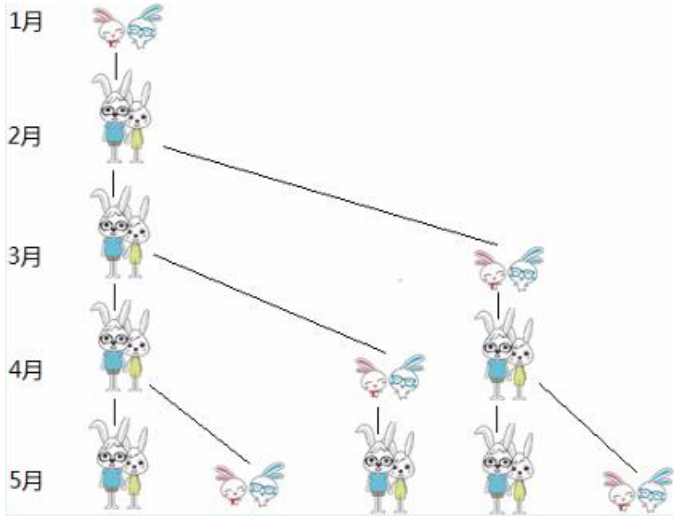
生活中的递归

求解斐波那契数列

- 斐波那契在《计算之书》中提出了一个有趣的兔子问题：假设一对兔子每个月可以生一对小兔子，一对兔子出生后第2个月就开始生小兔子。则一对兔子一年内能繁殖成多少对？10年呢？

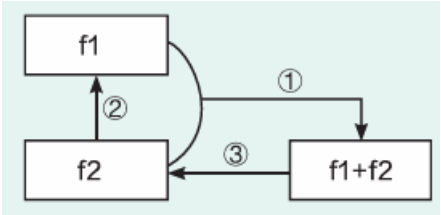
- 从第3个月起，每个月大兔子的对数等于上个月大兔子与小兔子的对数之和（即上个月兔子总对数），每个月小兔子的对数等于上个月大兔子的对数（即上上个月兔子总对数）。

- 第1个月和第2个月的兔子对数之和为第3个月的兔子对数，第2个月和第3个月的兔子对数之和为第4个月的兔子对数……，每个月的兔子对数是前两个月的兔子对数之和，又同时作为下一个月兔子对数的加数。这种重复反馈的过程称为迭代。



迭代法

- 迭代法也称辗转法，是用计算机解决问题的一种基本方法。迭代通常是为了接近并到达所需的目标或结果。每一次对过程的重复被称为一次“迭代”，而每一次迭代得到的结果会被用来作为下一次迭代的初始值。
- 由于在迭代系列中的每个月份兔子对数只跟前两个月有关，因此在编写程序时，只需两个变量$f1$和$f2$分别记录上上月和上月的数据。
- 迭代计算的示意图如下所示。




```
def fib(n):
    # 利用迭代算法求斐波那契数列第n项
    f1 = f2 = 1
    for i in range(3, n + 1):
        f1, f2 = f2, f1 + f2
    return f2
```

迭代算法的关键步骤与应用

- 利用迭代算法解决问题，有三个关键步骤：

（1）确定迭代变量，如求解斐波那契数列的f1、f2；

（2）建立迭代关系式；

（3）对迭代过程进行控制，这是编写选代程序必须考虑的问题，不能让迭代过程无休止地重复执行下去。

- 现代自然科学和工程电子技术的研究过程中，都离不开大规模的数学计算间题。例如：数学类课程中的线性方程求解、微分方程求解、概率统计等；实用性和实验性技术应用中的模拟核试验、油田开发、飞机设计等。

递归法

- 每月兔子的对数为 1,1,2,3,5,8,13.....，从第三个月起，当月兔子数为前两月兔子数之和。这个数列在数学上称做“斐波那契数列”。
- 分析：设定每月兔子对数为 $f(n)$,n 为月份，可以递归定义为：

$$f(n)=\begin{cases}1, & n=1,2\\ f(n-1)+f(n-2), & n>2\end{cases}$$

- 这种定义方式将问题分成了两种情况，其中一种比较简单，可以直接求出结果；另一种是把问题分解成规模更小的、具有与原问题相同解法的问题。在函数实现时，因为解决大问题的方法和解决小问题的方法往往是同一个方法，所以就产生了函数调用它自身的情况，这种使用方法我们称之为递归，这个函数就是递归函数。

递归

- 一个函数，自己调用自己，就是递归。
- 递归函数需要有终止条件，否则就会无穷递归导致程序无法终止甚至崩溃。
- 递归函数有两大要素：

（1）递归关系式：对问题进行递归形式的描述。

（2）递归终止条件：当满足该条件时以一种特殊情况处理，而不是用递归关系式来处理。

```
def 函数名(参数列表):
    if 递归结束条件:
        return 递归结束条件下的返回值
    else:
        ......
    return 递归计算公式
```

```
def f(n):  # 定义斐波那契函数
    if n == 1 or n == 2:
        return 1
    else:
        return f(n - 1) + f(n - 2)
```

递归求阶乘

- 利用递归算法求 n 的阶乘（$n! = 1 \times 2 \times \cdots \times n$），设函数 $f(n) = n!$，由数学知识可知，n 阶乘的递归定义为：

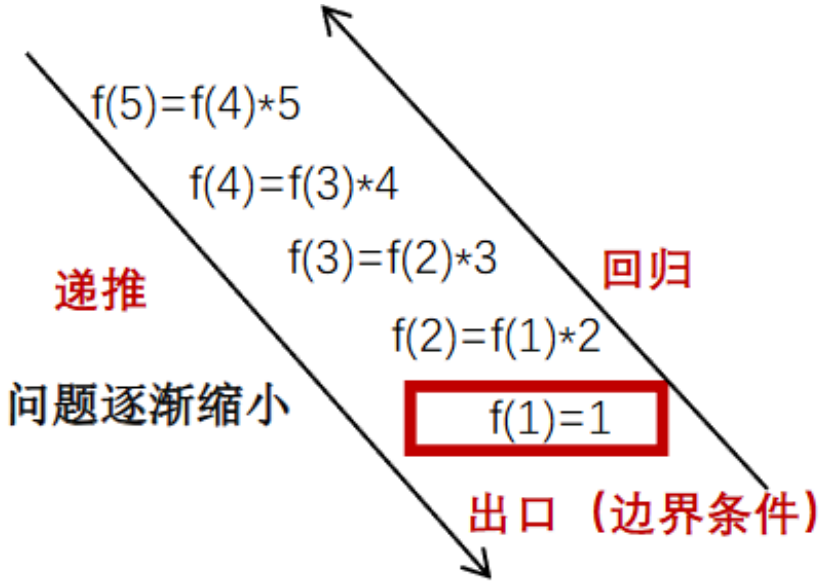
$$f(n) = \begin{cases} 1, & n = 1 \\ f(n-1) * n, & n > 1 \end{cases}$$

```
def f(n):
    if n == 1:
        return 1
    else:
        return f(n - 1) * n
```

- 递归是计算科学领域中一种重要的计算思维模式。它既是一种抽象表达的手段，也是一种问题求解的重要方法。

递归的基本思想

- 递归的基本思想是把规模较大的问题层层转化为规模较小的同类问题求解。对递归而言，递推与回归，二者缺一不可。
- 递归可用“分”，“治”，“合”三个字概括。
- 现在，我们再来看 5 的阶乘问题的递归过程：




```
def f(n):
    if n==1:
        return 1
    else:
        return f(n-1)*n
print(f(5))
```

递归的基本思想

- 递归关系式是递归的重要组成，而边界条件是递归的另一要素，它保证递归能在有限次的计算后得出结果，而不会产生无限循环的情况。
- 面对一个大规模复杂问题的求解，递归的基本思想是把规模较大的问题层层转化为规模较小的同类间题求解。对递归而言，递推与回归，二者缺一不可。结合分治策略，递归也可用“分”“治”“合”三个字概括。
- （1）分：将原问题分解成k个子问题。
- （2）治：对这k个子问题分别求解。如果子问题的规模仍然不够小，则将其再分解为k个子同题，如此进行下去，直到问题足够小时，就很容易求出子间题的解。
- （3）合：将求出的小规模问题的解合并为一个更大规模问题的解，首下而上逐步求出原同题的解。

求斐波那契数列时间对比

```python
import time


# 递归算法求斐波那契数列
def f(n):
    if n == 1 or n == 2:
        return 1
    else:
        return f(n - 1) + f(n - 2)


start = time.time()
print(f(40))
end = time.time()
print("程序运行时间为: %s 秒" % (end - start))
```

```
102334155
程序运行时间为: 13.436564207077026 秒
```

```python
import time


# 迭代算法求斐波那契数列
def f(n):
    f1 = f2 = 1
    for i in range(3, n + 1):
        f1, f2 = f2, f1 + f2
    return f2


start = time.time()
print(f(40))
end = time.time()
print("程序运行时间为: %s 秒" % (end - start))
```

```
102334155
程序运行时间为: 0.016675472259521484 秒
```

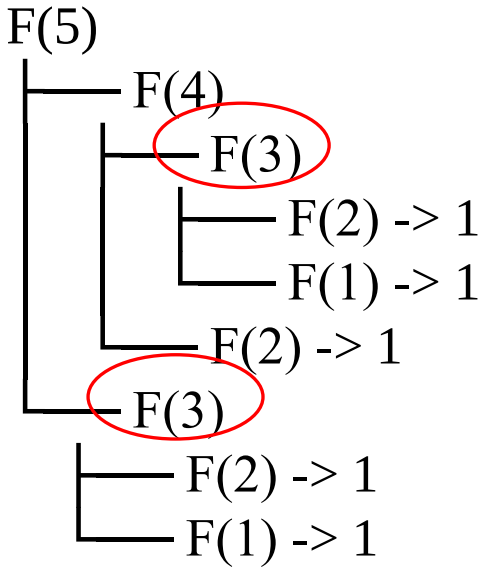
求斐波那契数列时间对比

迭代算法

- 通过一个循环从底向上计算斐波那契数列，逐步累加得到目标值。
- 每个斐波那契数只计算一次，时间复杂度是 $O(n)$。
- 例如，计算 $f(10)$ 只需要从 $f(1)$ 和 $f(2)$ 累加到 $f(10)$，共进行 10 次操作。

递归算法

- 从上向下递归地定义斐波那契数列，将问题不断拆解成子问题，直到到达基例$f(1)$和$f(2)$。
- 递归法的时间复杂度是 $O(2^n)$，因为在每次递归时都会进行两次子问题调用，存在大量的重复计算。
- 右图为递归算法的递归树。



比较迭代算法和递归算法

- 迭代算法与递归算法都需要重复执行某些代码，两者既有区别又有密切的联系。
- 迭代程序可以转换成等价的递归程序。

迭代	递归
重复方式：是重复反馈过程的活动，其目的通常是逼近所需目标或结果。	重复方式：重复调用函数自身。
结束方式：通常使用计数器结束循环。	结束方式：遇到满足终止条件的情况时逐层返回。