

05 | 递归（上）：泛化数学归纳，如何将复杂问题简单化？

2018-12-19 黄申

程序员的数学基础课

[进入课程 >](#)



讲述：黄申

时长 09:10 大小 8.41M



你好，我是黄申。上一节的结尾，我们用递归模拟了数学归纳法的证明。同时，我也留下了一个问题：**既然递归的函数值返回过程和基于循环的迭代法一致，我们直接用迭代法不就好了，为什么还要用递归的数学思想和编程方法呢？**这是因为，在某些场景下，递归的解法比基于循环的迭代法更容易实现。这是为什么呢？我们继续来看舍罕王赏麦的故事。

如何在限定总和的情况下，求所有可能的加和方式？

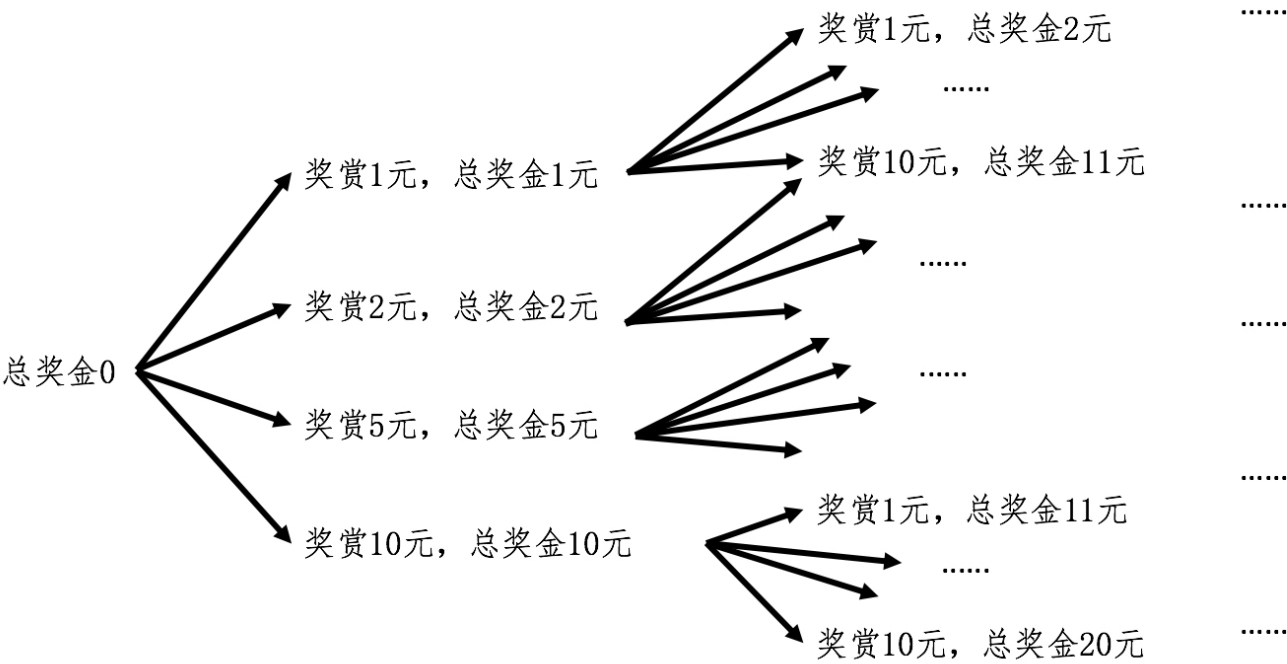
舍罕王和他的宰相西萨·班·达依尔现在来到了当代。这次国王学乖了，他对宰相说：“这次我不用麦子奖赏你了，我直接给你货币。另外，我也不用棋盘了，我直接给你一个固定数额的奖赏。”

宰相思考了一下，回答道：“没问题，陛下，就按照您的意愿。不过，我有个小小的要求。那就是您能否列出所有可能的奖赏方式，让我自己来选呢？假设有四种面额的钱币，1 元、2 元、5 元和 10 元，而您一共给我 10 元，那您可以奖赏我 1 张 10 元，或者 10 张 1 元，或者 5 张 1 元外加 1 张 5 元等等。如果考虑每次奖赏的金额和先后顺序，那么最终一共有多少种不同的奖赏方式呢？”

让我们再次帮国王想想，如何解决这个难题吧。这个问题和之前的棋盘上放麦粒有所不同，它并不是要求你给出最终的总数，而是在**限定总和的情况下，求所有可能的加和方式**。你可能会想，虽然问题不一样，但是求和的重复性操作仍然是一样的，因此是否可以使用迭代法？好，让我们用迭代法来试一下。

我还是使用迭代法中的术语，考虑 $k=1,2,3,...,n$ 的情况。在第一步，也就是当 $n=1$ 的时候，我们可以取四种面额中的任何一种，那么当前的奖赏就是 1 元、2 元、5 元和 10 元。当 $n=2$ 的时候，奖赏的总和就有很多可能性了。如果第一次奖赏了 1 元，那么第二次有可能取 1、2、5 元三种面额（如果取 10，总数超过了 10 元，因此不可能）。

所以，在第一次奖赏 1 元，第二次奖赏 1 元后，总和为 2 元；第一次奖赏 1 元，第二次奖赏 2 元后，总和为 3 元；第一次奖赏 1 元，第二次奖赏 5 元后，总和为 6 元。好吧，这还没有考虑第一次奖赏 2 元和 5 元的情况。我来画个图，从图中你就能发现这种情况在快速地“膨胀”。



你应该能看到，虽然迭代法的思想是可行的，但是如果用循环来实现，恐怕要保存好多中间状态及其对应的变量。说到这里，你是不是很容易就想到计算编程常用的**函数递归**？

在递归中，每次嵌套调用都会让函数体生成自己的局部变量，正好可以用来保存不同状态下的数值，为我们省去了大量中间变量的操作，极大地方便了设计和编程。

不过，这里又有新的问题了。之前用递归模拟数学归纳法还是非常直观的。可是，这里不是要计算一个最终的数值，而是要列举出所有的可能性。那应该如何使用递归来解决呢？上一节，我只是用递归编程体现了数学归纳法的思想，但是如果我们把这个思想泛化一下，那么递归就会有更多、更广阔的应用场景。

如何把复杂的问题简单化？

首先，我们来看，**如何将数学归纳法的思想泛化成更一般的情况**？数学归纳法考虑了两种情况：

1. 初始状态，也就是 $n=1$ 的时候，命题是否成立；
2. 如果 $n=k-1$ 的时候，命题成立。那么只要证明 $n=k$ 的时候，命题也成立。其中 k 为大于 1 的自然数。

将上述两点顺序更换一下，再抽象化一下，我写出了这样的递推关系：

1. 假设 $n=k-1$ 的时候，问题已经解决（或者已经找到解）。那么只要求解 $n=k$ 的时候，问题如何解决（或者解是多少）；
2. 初始状态，就是 $n=1$ 的时候，问题如何解决（或者解是多少）。

我认为这种思想就是将**复杂的问题，每次都解决一点点，并将剩下的任务转化成为更简单的问题等待下次求解，如此反复，直到最简单的形式**。回到开头的例子，我们再将这种思想具体化。

1. 假设 $n=k-1$ 的时候，我们已经知道如何去求所有奖赏的组合。那么只要求解 $n=k$ 的时候，会有哪些金额的选择，以及每种选择后还剩下多少奖金需要支付就可以了。
2. 初始状态，就是 $n=1$ 的时候，会有多少种奖赏。

有了这个思路，就不难写出这个问题的递归实现。我这里列一个基本的实现。

```

1 import java.util.ArrayList;
2
3 public class Lesson5_1 {
4
5     public static long[] rewards = {1, 2, 5, 10};    // 四种面额的纸币
6
7     /**
8      * @Description:    使用函数的递归（嵌套）调用，找出所有可能的奖赏组合
9      * @param totalReward- 奖赏总金额，result- 保存当前的解
10     * @return void
11     */
12
13     public static void get(long totalReward, ArrayList<Long> result) {
14
15         // 当 totalReward = 0 时，证明它是满足条件的解，结束嵌套调用，输出解
16         if (totalReward == 0) {
17             System.out.println(result);
18             return;
19         }
20         // 当 totalReward < 0 时，证明它不是满足条件的解，不输出
21         else if (totalReward < 0) {
22             return;
23         } else {
24             for (int i = 0; i < rewards.length; i++) {
25                 ArrayList<Long> newResult = (ArrayList<Long>)(result.clone());
26                 newResult.add(rewards[i]);
27                 get(totalReward - rewards[i], newResult);           // 剩下的
28             }
29         }
30
31     }
32
33 }

```


我们测试一下总金额为 10 元的时候，有多少种解。

```

1 public static void main(String[] args) {
2
3     int totalReward = 10;
4     Lesson5_1.get(totalReward, new ArrayList<Long>());
5
6 }

```

最终，程序运行后大致是这种结果：

 复制代码

```
1 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
2 [1, 1, 1, 1, 1, 1, 1, 1, 2]
3 [1, 1, 1, 1, 1, 1, 1, 2, 1]
4 [1, 1, 1, 1, 1, 1, 2, 1, 1]
5 [1, 1, 1, 1, 1, 1, 2, 2]
6 ...
7 [5, 5]
8 [10]
```

这里面每一行都是一种可能。例如第一行表示分 10 次奖赏，每次 1 元；第二行表示分 9 次奖赏，最后一次是 2 元；以此类推。最终结果的数量还是挺多的，一共有 129 种可能。试想一下，如果总金额为 100 万的话，会有多少种可能啊！

这个代码还有几点需要留意的地方，我再来解释一下：

1. 由于一共只有 4 种金额的纸币，所以无论是 $n=1$ 的时候还是 $n=k$ 的时候，我们只需要关心这 4 种金额对组合产生的影响，而中间状态和变量的记录和跟踪这些繁琐的事情都由函数的递归调用负责。
2. 这个案例的限制条件不再是 64 个棋格，而是奖赏的总金额，因此判断嵌套调用是否结束的条件其实不是次数 k ，而是总金额。这个金额确保了递归不会陷入死循环。
3. 我这里从奖赏的总金额开始，每次嵌套调用的时候减去一张纸币的金额，直到所剩的金额为 0 或者少于 0，然后结束嵌套调用，开始返回结果值。当然，你也可以反向操作，从金额 0 开始，每次嵌套调用的时候增加一张纸币的金额，直到累计的金额达到或超过总金额。

小结

递归和循环其实都是迭代法的实现，而且在某些场合下，它们的实现是可以相互转化的。但是，对于某些应用场景，递归确很难被循环取代。我觉得主要有两点原因：

第一，递归的核心思想和数学归纳法类似，并更具有广泛性。这两者的类似之处体现在：**将当前的问题化解为两部分：一个当前所采取的步骤和另一个更简单的问题。**

1. 一个当前所采取的步骤。这种步骤可能是进行一次运算（例如每个棋格里的麦粒数是前一格的两倍），或者做一个选择（例如选择不同面额的纸币），或者是不同类型操作的结合（例如今天讲的赏金的案例）等等。

2. 另一个更简单的问题。经过上述步骤之后，问题就会变得更加简单一点。这里“简单一点”，指运算的结果离目标值更近（例如赏金的总额），或者是完成了更多的选择（例如纸币的选择）。而“更简单的问题”，又可以通过嵌套调用，进一步简化和求解，直至达到结束条件。

我们只需要保证递归编程能够体现这种将复杂问题逐步简化的思想，那么它就能帮助我们解决很多类似的问题。

第二，递归会使用计算机的函数嵌套调用。而函数的调用本身，就可以保存很多中间状态和变量值，因此极大的方便了编程的处理。

正是如此，递归在计算机编程领域中有着广泛的应用，而不仅仅局限在求和等运算操作上。在下一节中，我将介绍如何使用递归的思想，进行“分而治之”的处理。

今日学习笔记

第5节 递归（上）

1. 在递归中，每次嵌套调用都会让函数体生成自己的局部变量，正好可以用来保存不同状态下的数值，为我们省去了大量中间变量的操作，极大地方便了设计和编程。

2. 递归就是将复杂的问题，每次都解决一点点，并将剩下的任务转化成为更简单的问题等待下次求解，如此反复，直到最简单的形式。

3. 递归和循环其实都是迭代法的实现，而且在某些场合下，它们的实现是可以相互转化的。



黄申 · 程序员的数学基础课

思考题

一个整数可以被分解为多个整数的乘积，例如，6 可以分解为 2×3 。请使用递归编程的方法，为给定的整数 n ，找到所有可能的分解（1 在解中最多只能出现 1 次）。例如，输入 8，输出是可以是 1×8 , 8×1 , 2×4 , 4×2 , $1 \times 2 \times 2 \times 2$, $1 \times 2 \times 4$,

欢迎在留言区交作业，并写下你今天的学习笔记。你可以点击“请朋友读”，把今天的内容分享给你的好友，和他一起精进。

程序员的数学基础课

在实战中重新理解数学

黄申

LinkedIn 资深数据科学家



新版升级：点击「👤 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 04 | 数学归纳法：如何用数学归纳提升代码的运行效率？

下一篇 06 | 递归（下）：分而治之，从归并排序到MapReduce

精选留言 (89)

写留言



黄申 置顶

2018-12-20

👍 6

由于暂时无法追加回复，我这里再回复一下网友debugtalk

我看了Python的实现，果然很简介👍

奖金的题目没问题。整数的因子分解有点小瑕疵，少了一些可能。比如，输入8，少了[1, 2, 2, 2]...

展开 ▾



杨景胜

2018-12-19

👍 13

从递归的两个原则到代码实现有点跳跃，想了半天还是没想明白这个代码的逻辑啊~



debugtalk

2018-12-19

👍 9

Python 实现赏金问题：<https://github.com/debugtalk/geektime-notes/blob/master/programmers-mathematics/chapter5.md>

Python 实现思考题：<https://github.com/debugtalk/geektime-notes/blob/master/programmers-mathematics/chapter5.py>

展开 ▾

作者回复: 很棒 我稍后sync下来看



李尧

2018-12-19

👍 7

思考题：请大神帮忙看下，输出少了个 [1,8]

输出：[2, 2, 2, 1] [2, 4, 1][4, 2, 1][8, 1]

```
import java.util.ArrayList;
```

...

展开 ▾

作者回复: 循环的时候不能少了1，可以在结果中判断是否已经涵盖1，我稍微修改了一下

```
public static void recursion(long total, ArrayList<Long> result) {
```

```
    if (total == 1) {
        if (!result.contains(1L)) result.add(1L);
        System.out.println(result);
        return;
    } else {
        for (long i = 1; i <= total; i++) {
            if ((i == 1) && result.contains(1L)) continue;
            ArrayList<Long> newList = (ArrayList<Long>) (result.clone());
            newList.add(Long.valueOf(i));
            if (total % i != 0) {
                continue;
            }
            recursion(total / i, newList);
        }
    }
}
```

```
}  
}
```



文リ共...

2018-12-19

👍 5

整数分解 - C++代码

```
#include <vector>
```

```
#include <iostream>
```

```
#include <algorithm>
```

...

展开 ▾

作者回复: 回答很棒，下次可以将运行结果也贴出来👍



松原

2018-12-19

👍 3

黄老师，这句“递归和循环其实都是迭代法的实现”我不太理解。

如果递归和循环都属于迭代法，那么就是说递归是从属于迭代法的。而我所理解的迭代法的核心是从1到n的递推，而递归是从n到1的逐步求解的过程，两者应该是并列的关系。希望老师能解答我的这个困惑。

展开 ▾

作者回复: 确实两者的递推方向是不一样的，不过递归在计算机的实现中，是使用的函数调用，在满足条件后，函数开始逐级返回，这时候又是正向递推了，所以我觉得这也是从1到n



杨景胜

2018-12-19

👍 3

//因数分解

```
public static void getMultiFactors(long multi,ArrayList<Long> result){
```

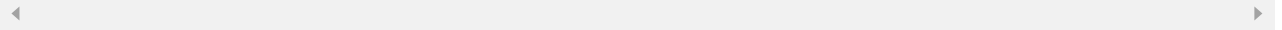
```
    if (multi == 1){
```

```
        result.add(multi);
```

```
        System.out.println(result);...
```

展开 ▾

作者回复: 如果循环从2开始, 可能会漏掉一些情况, 请参考为我给另一位网友李尧的回复



悬炫

2019-01-28

👍 2

JavaScript的实现:

赏金问题:

```
/**
```

```
*
```

```
* @description 赏金问题...
```

展开 ▾



icy

2018-12-24

👍 2

```
# -*- coding: utf-8 -*-
```

```
"""
```

```
Created on Mon Dec 24 14:30:38 2018
```

```
@author: icytanz
```

```
""" ...
```

展开 ▾



郑晨Cc

2018-12-20

👍 2

```
package 二进制;
```

```
import java.util.ArrayList;
```

```
public class Lesson5Test {...
```

展开 ▾



debugtalk

2018-12-20

👍 2

感谢 Sean 的指正, 之前整数因子分解的解答的确存在问题, 没有完整考虑整数 1 在各种位置的情况。

现已更正：<https://github.com/debugtalk/geektime-notes/blob/master/programmers-mathematics/chapter5.py>

展开 ▾



唐瑞甫

2018-12-19

👍 2

我来答下吧，案例中的n代表已经迭代的次数，在这个案例中，限制了每轮迭代只有四种选择



Sawyer

2019-03-12

👍 1

老师好，我实现了一个算法，但是没有打印出来1的情况。

还有个问题就是，如果使用老师的示例输入8，结果既有 2x4，又有 4x2 这不是重复了吗？

```
static void getFactorization(long product, ArrayList<Long> result) {  
    for (int i = 2; i <= product / 2; i++) {...
```

展开 ▾

作者回复: 这里作为教学案例，可以遍历所有情况，包括4x2和2x4。至于1，需要特殊处理一下，你可以思考看看，或者看看之前读者的留言

◀ ▶



方得始终

2018-12-29

👍 1

参考老师和其他同学的留言，下面是Python实现的思考题，应该是个较为简洁的版本。

```
import copy
```

```
def prod_factors(num, result=[]):  
    if num == 1:...
```

展开 ▾

作者回复: 同时考虑了1出现和不出现的情况 📖

◀ ▶



Youngggg

👍 1



2018-12-25

```
import copy
def get(num, result=[]):
    sum = 1
    for index in result:
        sum = sum * index...
```

展开 ∨



hallon

2018-12-21

👍 1

思考题，用js实现如下：

```
function mul(totalReward, result) {
    //i从自身开始循环到1，每次递减1
    for(var i=totalReward; i>0; i--) {
        if(totalReward % i == 0) { //能整除i...
```

展开 ∨



assert

2018-12-20

👍 1

这里贴代码对Python这种缩进语言来讲很不友好啊，建议可以提交图片或者支持md格式。说下自己的思路：一个整数num 对 num 到 1 之间的整数 i 分别求余，如果余数为 0，说明这是一个因子，将 i 添加到结果列表里，然后再让num 对 i 取整，得到 k，对整数 K 再次递归求解。退出条件，如果 num == 1，那么将 1 添加到结果列表里，并打印结果列表。这里要注意下，如果 i == 1，也是退出条件，此时将 num 加入结果列表并打印...

展开 ∨

作者回复: 少了一些可能。比如，输入8，少了

[1, 2, 2, 2]

[1, 2, 4]

[1, 4, 2]

[2, 1, 2, 2]



文リ共...

2018-12-19

👍 1

使用C++实现赏金问题

```
#include <vector>
```

```
#include <iostream>
```

```
using namespace std;...
```

展开 ▾



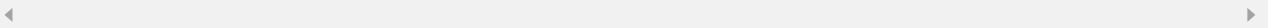
cocu

2018-12-19

👍 1

这个案例中的n到底是什么，是奖励总金额，还是取的纸币数？

作者回复: 这里我统一回答一下，是当前迭代的次数，也就是取的纸币数量



Gavin黄炯...

2019-05-10

👍

老师，假如不考虑先后顺序要怎么做？可以给点思路吗

展开 ▾

作者回复: 具体你是指哪个先后顺序？

