04讲数学归纳法：如何用数学归纳提升代码的运行效率

你好，我是黄申。

上次我们聊了迭代法及其应用，并用编程实现了几个小例子。不过你知道吗，对于某些迭代问题，我们其实可以避免一步步的计算，直接**从理论上证明某个结论**，节约大量的计算资源和时间，这就是我们今天要说的**数学归纳法**。

平时我们谈的“归纳”，是一种从经验事实中找出普遍特征的认知方法。比如，人们在观察了各种各样动物之后，通过它们的外观、行为特征、生活习性等得出某种结论，来区分哪些是鸟、哪些是猫等等。比如我这里列出的几个动物的例子。

动物	特征-皮毛	特征-是否会飞	特征-吃什么	类别
第1只麻雀	羽毛	是	小虫	鸟
第2只麻雀	羽毛	是	小虫	鸟
1头老鹰	羽毛	是	小动物	鸟
1只波斯猫	绒毛	否	小鱼,老鼠	猫
1只中国狸花猫	绒毛	否	小鱼,老鼠	猫

通过上面的表格，我们可以进行归纳，并得出这样的结论：

- 如果一个动物，身上长羽毛并且会飞，那么就是属于鸟；
- 如果一个动物，身上长绒毛、不会飞、而且吃小鱼和老鼠，那么就属于猫。

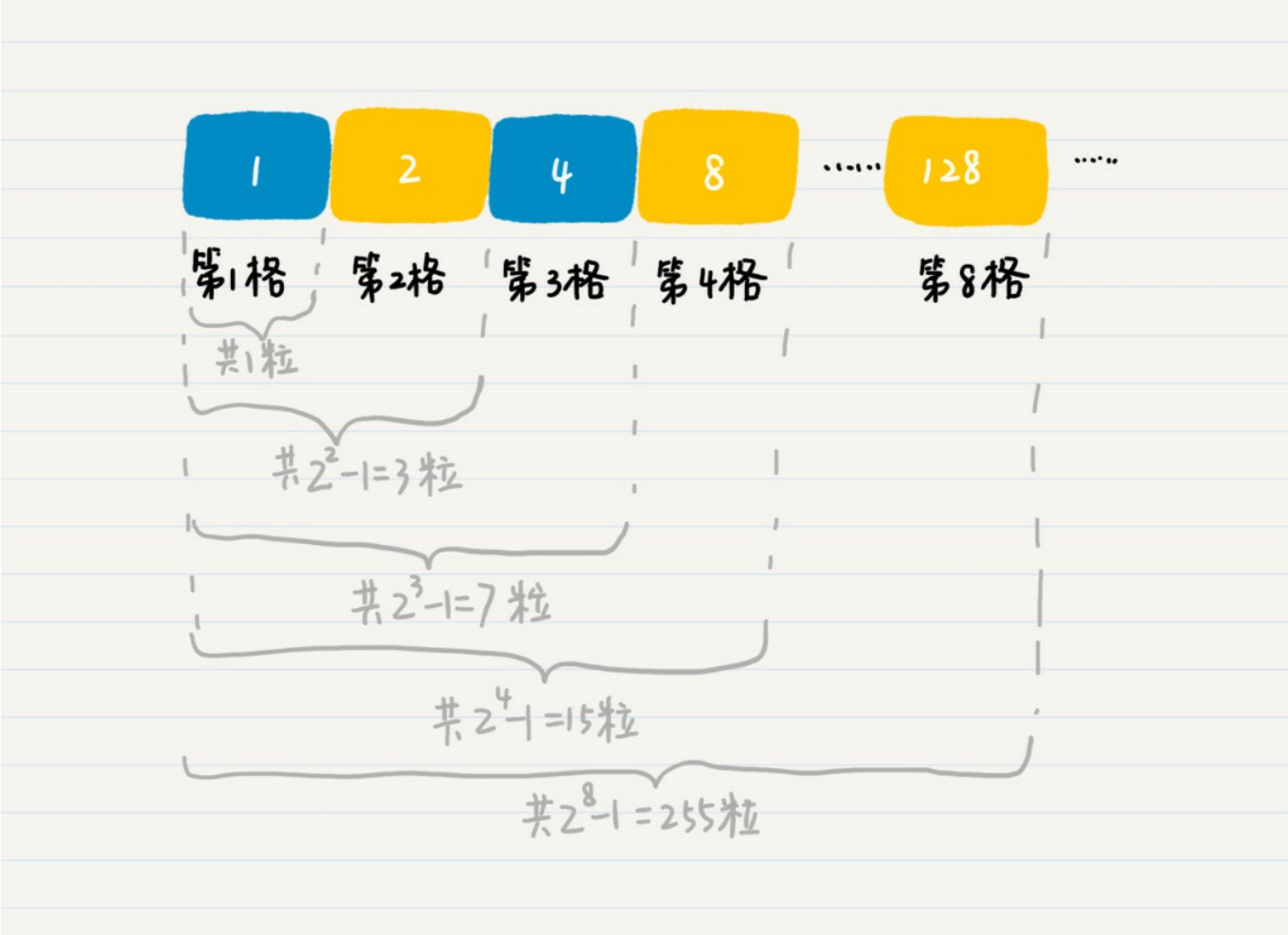
通过观察5个动物样本的3个特征，从而得到某种动物应该具有何种特征，这种方法就是我们平时所提到的归纳法。

我们日常生活中所说的这种归纳法和数学归纳法是不一样的，它们究竟有什么区别呢？具体数学归纳法可以做什么呢？我们接着上一节舍罕王赏麦的故事继续说。

什么是数学归纳法?

上节我们提到，在棋盘上放麦粒的规则是，第一格放一粒，第二格放两粒，以此类推，每一小格内都比前一小格多一倍的麦子，直至放满64个格子。

我们假想一下自己穿越到了古印度，正站在国王的身边，看着这个棋盘，你发现第1格到第8格的麦子数分别是：$1、2、4、8、16、32、64、128$。这个时候，国王想知道总共需要多少粒麦子。我们小时候都玩过“找规律”，于是，我发现了这么一个规律，你看看是不是这样?



根据这个观察，我们是不是可以大胆假设，前n个格子的麦粒总数就是$2^{n}-1$ 呢？如果这个假设成立，那么填满64格需要的麦粒总数，就是$1+2+2^{2}+2^{3}+2^{4}+……+ 2^{63}$
$＝2^{64}-1＝18446744073709551615$。

这个假设是否成立，我们还有待验证。但是对于类似这种无穷数列的问题，我们通常可以采用**数学归纳法**（Mathematical Induction）来证明。

在数论中，数学归纳法用来证明任意一个给定的情形都是正确的，也就是说，第一个、第二个、第三个，一直到所有情形，概不例外。

数学归纳法的一般步骤是这样的：

- **证明基本情况（通常是$n=1$的时候）是否成立；**
- **假设$n=k-1$成立，再证明$n=k$也是成立的（k为任意大于1的自然数）。**

只要学过数学，我想你对这个步骤都不陌生。但是，现在你需要牢记这个步骤，然后我们用这个步骤来证明下开头的例子。

为了让你更好地理解，我将原有的命题分为两个子命题来证明。第一个子命题是，第n个棋格放的麦粒数为2^{n-1}。第二个子命题是，前n个棋格放的麦粒数总和为$2^{n}-1$。

首先，我们来证明第一个子命题。

- 基本情况：我们已经验证了$n=1$的时候，第一格内的麦粒数为1，和2^{1-1}相等。因此，命题在$k=1$的时候成立。

- 假设第$k-1$格的麦粒数为2^{k-2}。那么第k格的麦粒数为第$k-1$格的2倍，也就是$2^{k - 2}*2=2^{k-1}$。因此，如果命题在$k=n-1$的时候成立，那么在$k=n$的时候也成立。

所以，第一个子命题成立。在这个基础之上，我再来证明第二个子命题。

- 基本情况：我们已经验证了$n=1$的时候，所有格子的麦粒总数为1。因此命题在$k=1$的时候成立。
- 假设前$k-1$格的麦粒总数为$2^{k-1}-1$，基于前一个命题的结论，第k格的麦粒数为2^{k-1}。那么前k格的麦粒总数为$(2^{k-1}-1)+(2^{k-1})=2*2^{k-1}-1=2^{k}-1$。因此，如果命题在$k=n-1$的时候成立，那么在$k=n$的时候也成立。

说到这里，我已经证明了这两个命题都是成立的。**和使用迭代法的计算相比，数学归纳法最大的特点就在于“归纳”二字。它已经总结出了规律。只要我们能够证明这个规律是正确的，就没有必要进行逐步的推算，可以节省很多时间和资源。**

说到这里，我们也可以看出，数学归纳法中的“归纳”是指的从第一步正确，第二步正确，第三步正确，一直推导到最后一步是正确的。这就像多米诺骨牌，只要确保第一张牌倒下，而每张牌的倒下又能导致下一张牌的倒下，那么所有的骨牌都会倒下。从这里，你也能看出来，这和开篇提到的广义归纳法是不同的。数学归纳法并不是通过经验或样本的观察，总结出事物的普遍特征和规律。

好了，对数学归纳法的概念，我想你现在已经理解了。这里，我对上一节中有关麦粒的代码稍作修改，增加了一点代码来使用数学归纳法的结论，并和迭代法的实现进行了比较，你可以看看哪种方法耗时更长。

```
public static void main(String[] args) {

  int grid = 63;
  long start, end = 0;
  start = System.currentTimeMillis();
  System.out.println(String.format("舍罕王给了这么多粒：%d", Lesson3_1.getNumberOfWheat(grid)));
  end = System.currentTimeMillis();
  System.out.println(String.format("耗时%d毫秒", (end - start)));

  start = System.currentTimeMillis();
  System.out.println(String.format("舍罕王给了这么多粒：%d", (long)(Math.pow(2, grid)) - 1));
  end = System.currentTimeMillis();
  System.out.println(String.format("耗时%d毫秒", (end - start)));

 }
```

在我的电脑上，这段代码运行的结果是：舍罕王给了9223372036854775807粒，耗时4毫秒。舍罕王给了这么多粒：9223372036854775806，耗时0毫秒。

你可能已经发现，当grid=63时，结果差了1个。这个是由于Math.pow()函数计算精度导致的误差。正确的结果应该是9223372036854775807。不过，基于数学归纳结论的计算明显在耗时上占有优势。虽然在我的笔记本电脑上只有4毫秒的差距，但是在生产项目的实践中，这种点点滴滴的性能差距都有可能累积成明显的问题。

递归调用和数学归纳的逻辑是一样的?

我们不仅可以使用数学归纳法从理论上指导编程，还可以使用编程来模拟数学归纳法的证明。如果你仔细观察一下数学归纳法

的证明过程，会不会觉得和函数的**递归调用**很像呢？

这里我通过总麦粒数的命题来示范一下。首先，我们要把这个命题的数学归纳证明，转换成一段伪代码，这个过程需要经过这样两步：

第一步，如果n为1，那么我们就判断麦粒总数是否为$2^{1-1}=1$。同时，返回当前棋格的麦粒数，以及从第1格到当前棋格的麦粒总数。

第二步，如果n为$k-1$的时候成立，那么判断n为k的时候是否也成立。此时的判断依赖于前一格$k-1$的麦粒数、第1格到$k-1$格的麦粒总数。这也是上一步我们所返回的两个值。

你应该看出来了，这两步分别对应了数学归纳法的两种情况。在数学归纳法的第二种情况下，我们只能假设$n=k-1$的时候命题成立。但是，在代码的实现中，我们可以将伪代码的第二步转为函数的递归（嵌套）调用，直到被调用的函数回退到$n=1$的情况。然后，被调用的函数逐步返回$k-1$时命题是否成立。

如果要写成具体的函数，就类似下面这样：

```
class Result {
 public long wheatNum = 0;  // 当前格的麦粒数
 public long wheatTotalNum = 0;  // 目前为止麦粒的总数
}

public class Lesson4_2 {

 /**
    * @Description: 使用函数的递归（嵌套）调用，进行数学归纳法证明
    * @param k-放到第几格，result-保存当前格子的麦粒数和麦粒总数
    * @return boolean-放到第k格时是否成立
    */

    public static boolean prove(int k, Result result) {

     // 证明n = 1时，命题是否成立
     if (k == 1) {
      if ((Math.pow(2, 1) - 1) == 1) {
       result.wheatNum = 1;
       result.wheatTotalNum = 1;
       return true;
      } else return false;
     }
     // 如果n = (k-1)时命题成立，证明n = k时命题是否成立
     else {

      boolean proveOfPreviousOne = prove(k - 1, result);
      result.wheatNum *= 2;
      result.wheatTotalNum += result.wheatNum;
      boolean proveOfCurrentOne = false;
      if (result.wheatTotalNum == (Math.pow(2, k) - 1)) proveOfCurrentOne = true;

      if (proveOfPreviousOne && proveOfCurrentOne) return true;
      else return false;

     }

    }

}
```

其中，类Result用于保留每一格的麦粒数，以及目前为止的麦粒总数。这个代码递归调用了函数prove(int, Result)。

从这个例子中，我们可以看出来，**递归调用的代码和数学归纳法的逻辑是一致的**。一旦你理解了数学归纳法，就很容易理解递归调用了。只要数学归纳证明的逻辑是对的，递归调用的逻辑就是对的，我们没有必要纠结递归函数是如何嵌套调用和返回的。

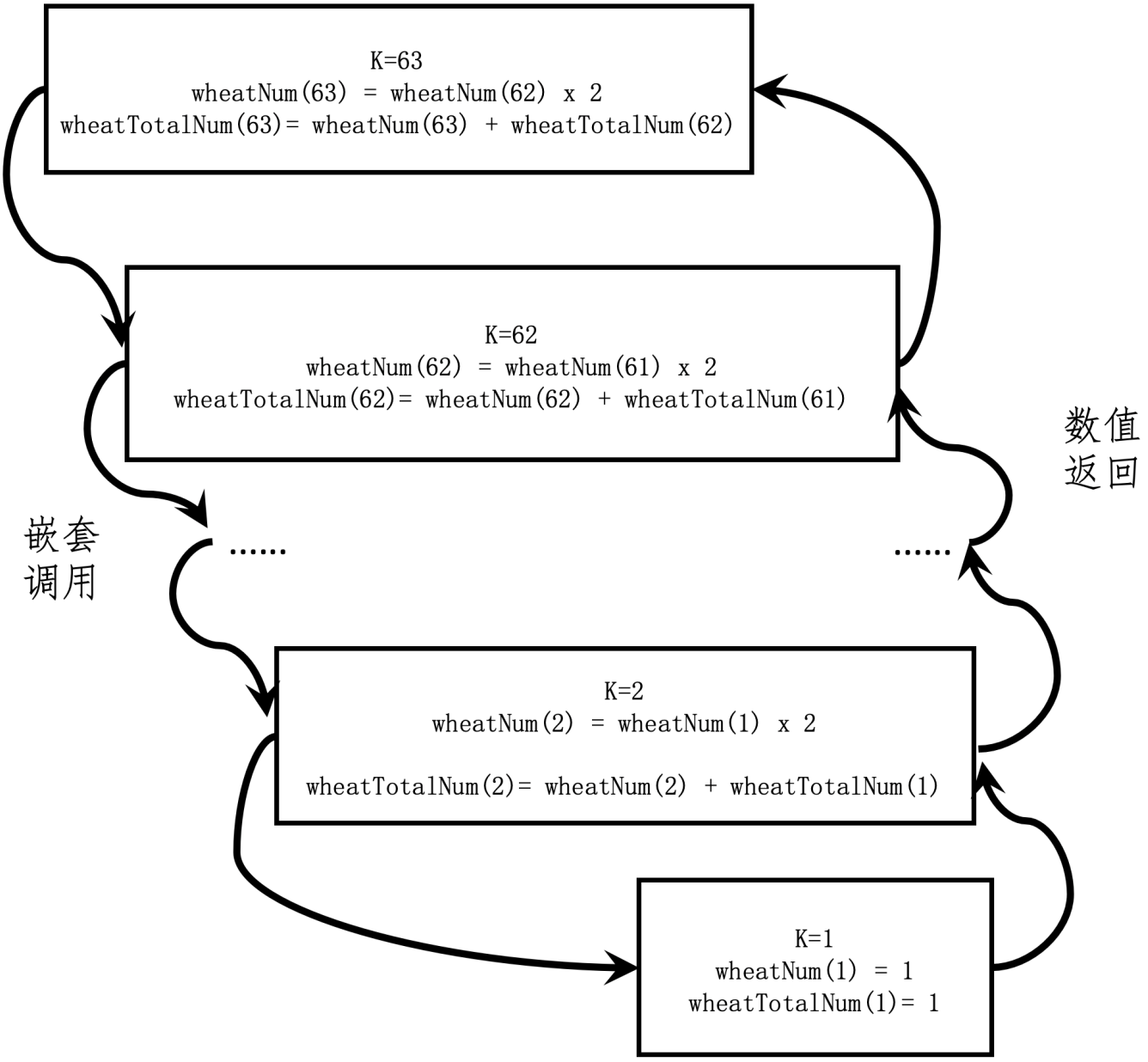
不过，和数学归纳证明稍有不同的是，递归编程的代码需要返回若干的变量，来传递$k-1$的状态到k。这里，我使用类Result来实现这一点。

这里是一段测试的代码。

```
public static void main(String[] args) {

  int grid = 63;

  Result result = new Result();
  System.out.println(Lesson4_2.prove(grid, result));

 }
```

我们最多测试到63。因为如果测试到第64格，麦粒总数就会溢出Java的long型数据。

你可以自己分析一下函数的调用和返回。我这里列出了一开始嵌套调用和到递归结束并开始返回值得的几个状态：



从这个图可以看出，函数从$k=63$开始调用，然后调用$k-1$，也就是62，一直到$k=1$的时候，嵌套调用结束，$k=1$的函数体开始返回值给$k=2$的函数体，一直到$k=63$的函数体。从$k=63, 62, ..., 2, 1$的嵌套调用过程，其实就是体现了数学归纳法的核心思想，我把它称为**逆向递推**。而从$k=1, 2, ..., 62, 63$的值返回过程，和上一篇中基于循环的迭代是一致的，我把它称为**正向递推**。

小结

今天，我介绍了一个编程中非常重要的数学概念：数学归纳法。

上一节我讲了迭代法是如何通过重复的步骤进行计算或者查询的。与此不同的是，数学归纳法在理论上证明了命题是否成立，而无需迭代那样反复计算，因此可以帮助我们节约大量的资源，并大幅地提升系统的性能。

数学归纳法实现的运行时间几乎为0。不过，数学归纳法需要我们能做出合理的命题假设，然后才能进行证明。虽然很多时候要做这点比较难，确实也没什么捷径。你就是要多做题，多去看别人是怎么解题的，自己去积累经验。

最后，我通过函数的递归调用，模拟了数学归纳法的证明过程。如果你细心的话，会发现递归的函数值返回实现了从$k=1$开始到$k=n$的迭代。说到这里，你可能会好奇：既然递归最后返回值的过程和基于循环的迭代是一致，那为什么还需要使用递

归的方法呢？下一节，我们继续聊这个问题。

今日学习笔记

第4节　数学归纳法

1. 数学归纳法和归纳有什么不一样？

平时我们谈的“归纳”，是一种从经验事实中找出普遍特征的认知方法。数学归纳法的一般步骤是：证明基本情况是否成立；再假设n=k-1成立，证明n=k也是成立的。

2. 数学归纳法和迭代法有什么不一样？

和迭代法相比，数学归纳法最大的特点就在“归纳”二字。它已经总结出了规律。只要我们能够证明这个规律是正确的，就没有必要进行逐步的推算，可以节省很多时间和资源。

3. 递归调用和数学归纳的逻辑是一样的！

只要数学归纳证明的逻辑是对的，递归调用的逻辑就是对的，没有必要纠结递归函数是如何嵌套调用和返回的。

黄申 · 程序员的数学基础课

思考题

在你日常工作的项目中，什么地方用到了数学归纳法来提升代码的运行效率？如果没有遇到过，你可以尝试做做实验，看看是否有提升？

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。

精选留言

oddrock

递归把计算交给计算机，归纳把计算交给人，前者是拿计算机的计算成本换人的时间，后者是拿人的时间换计算机的计算成本

2018-12-18 08:18

作者回复

精辟的总结

2018-12-18 09:15

cafu_chino

老师下次可以提供Python的代码示例吗？对Java的使用不是很熟练

2018-12-17 15:01

作者回复

可以，后期会整理出来

2018-12-17 23:09

Zoctopus

项目中还没碰到，但在以前做编程题时经常碰到找规律的问题，这时候只需抽象出一个公式即可。

另外可否给老师提个建议，就是在github上建一个仓库，这样我们可以fork下来，用自己熟悉的编程语言把老师讲的思路写进代码里push上去，方便大家相互学习 ^_^

2018-12-17 19:48

作者回复

这是个好建议，我先问下是否版权问题

2018-12-17 22:51

陈诚

我觉得加深学习数学效果的方式，就是自己实现一遍，我这里用C语言实现了一下递归求麦粒总数的问题以求抛砖引玉

```
#include <stdio.h>
#include <stdlib.h>

struct result {
long cur_grid_wheat;
long sum_of_wheat;
};

struct result wheat(int grid)
{
struct result rst;

if (grid == 1) {
rst.cur_grid_wheat = 1;
rst.sum_of_wheat = 1;
} else {
struct result last_rst;
last_rst = wheat(grid - 1);
rst.cur_grid_wheat = last_rst.cur_grid_wheat * 2;
rst.sum_of_wheat = last_rst.sum_of_wheat + rst.cur_grid_wheat;
}

return rst;
}

int main(int argc, char *argv[])
{
struct result rst;
int grid_index = 0;

if (argc != 2) {
printf("Usage: %s grid_index\n", argv[0]);
return -1;
}
grid_index = atoi(argv[1]);

rst = wheat(grid_index);

printf("grid_index: %d\ncur_grid_wheat: %ld\nsum_of_wheat: %ld\n",
grid_index, rst.cur_grid_wheat, rst.sum_of_wheat);

return 0;
}
```

2018-12-19 08:26

!不求甚解

我觉得看留言也是一种精进，以后多多看留言。

2018-12-25 13:03

陈诚

个人觉得有时候虽然用迭代和递归都能解决问题，但是使用递归时，由于有数学归纳法保证递归关系的正确性，所以只要专注于解决2个相邻层的关系就可以了，然后使用数学归纳法的基本情况作为递归出口。当然，在实际编程中，递归会增加函数调用栈的开销，也是要考虑的一方面

2018-12-17 07:55

WL

老师我感觉递归和归纳法还是有些区别的, 递归是自顶向下的拆解问题, 直到终止条件后再一层层返回结果, 但是归纳法好像没有自顶向下拆解问题的这个环节.

2018-12-17 07:41

失火的夏天

个人觉得动态规划就是归纳法思想的处理，一个问题分解为多个字问题的最优解

2018-12-17 07:27

鱼鱼鱼培填

用Python实现数学归纳法，一开始使用math.pow()函数发现不对，Python中该函数会使用科学技术法导致结果出错

```
#-*- coding:UTF-8 -*-
class result(object):
wheatNum = 0
wheatTotalNum = 0

class getWheatTotalNum(object):
'''
函数说明：使用递归嵌套， 进行数学归纳法证明
Param： k - 表示放到第几格 result - 表示当前格子的麦粒数
Return: boolean - 放到第K格时是否成立
'''
def prove(self, k, result):
if k == 1:
if (2 ** 1 - 1) == 1:
result.wheatNum = 1
result.wheatTotalNum = 1
return True
else:
return False

else:
proveOfPreviousOne = self.prove(k - 1, result)
result.wheatNum *= 2
result.wheatTotalNum += result.wheatNum
proveOfCurrentOne = False
if result.wheatTotalNum == (2 ** k - 1):
proveOfCurrentOne = True
if (proveOfPreviousOne & proveOfCurrentOne):
return True
else:
return False


if __name__ == '__main__':
grid = 64
result = result()
g = getWheatTotalNum()
print(g.prove(grid, result))
```

2018-12-19 23:35

作者回复

研究了细节，很赞

2018-12-20 01:24

田野

关于这节课讲的内容在编程中具体的应用能这样理解不？ 迭代法在实际应用中，如果迭代层次过深 ，会导致各种问题(耗时\内存占用等) ，遇到这种情况可以总结规律， 使用数学归纳法将其简化。(代码中不再使用迭代 使用数学归纳总结出来的结果)

2018-12-17 15:25

作者回复

是的，特别是递归的实现比较耗资源

2018-12-17 23:09

mikukuma

数据归纳法通俗易懂的说法就是一个个经典的数学公式的证明。我们要去证明这个公式是否正确，需要根据两个步骤，1：n=1时，公式是否正确；2: 假设n = k-1成立，那么去证明n=k公式是成立的。就跟数学做证明题一样的思路。然后我们去做这些算法题的时候，就可以直接根据已经证明的公式，很快就能得到我们需要的答案。

2018-12-17 11:29

未明。

数字全变成了math_progress_error了

2018-12-17 08:37

仁

老师~我没有学习JAVA，只学了C++和python。。。请问能否多发几种语言的代码哈

2019-01-06 20:49

作者回复

好的 后面我会统一加上

2019-01-07 10:17

fcb的鱼

每节课后可以多留一些编程题，每节课就2个例子感觉不过瘾！

2018-12-25 09:35

十指流玉

老师您好！如果有一些程序我们自己归纳出了规矩，那如何快速的应用到程序中呢，还是说程序中本身就有一些函数就是归纳法的体现?

2018-12-25 08:54

作者回复

这里你说的“有一些程序我们自己归纳出了规律”，具体是指什么?

2018-12-25 13:48

吴...

老师，您讲这些数学知识的时候能不能更加深入一点，最好能够结合工程项目或者一些著名的库函数呢?

2018-12-19 19:34

作者回复

后面会结合更多实际的项目。至于你说的库函数，具体是指哪方面？能否举个例子?

2018-12-19 22:51

吾本糊涂

python中计算n的阶乘

```
def func(n):
    if n == 0:
        return 1
    else:
        return n * func(n-1)

n! = func(n)
```

Python3默认递归的深度不能超过100层

2018-12-17 13:52

qinggeouye

清理服务器的文件夹（包括文件），清理文件夹里的文件夹（包括文件）...
～(￣▽￣～)～

2019-02-03 11:53

作者回复

对的，这就是递归的思想

2019-02-04 01:21

sloth-yp

个人理解： 计算机实现递归的过程，其实还是一层层迭代。本文是利用了计算机的递归调用的方式，来证明了舍罕王这个特定问题中，用“数学归纳法”得出的公式的正确性。但是编程实践中，应该尽可能用数学归纳法来总结出简化的算法，来节约计算机的资源。

2019-01-31 16:04

作者回复

是的

2019-02-01 01:33

悬炫

希望提供一下JavaScript的实现方式，或者像其他同学说的一样，提供一个仓库，让读者来贡献其他语言的实现方式，方便互相交流和学习

2019-01-24 16:51

作者回复

嗯，我们正在准备GitHub中，准备好了，及时通知大家

2019-01-24 23:33