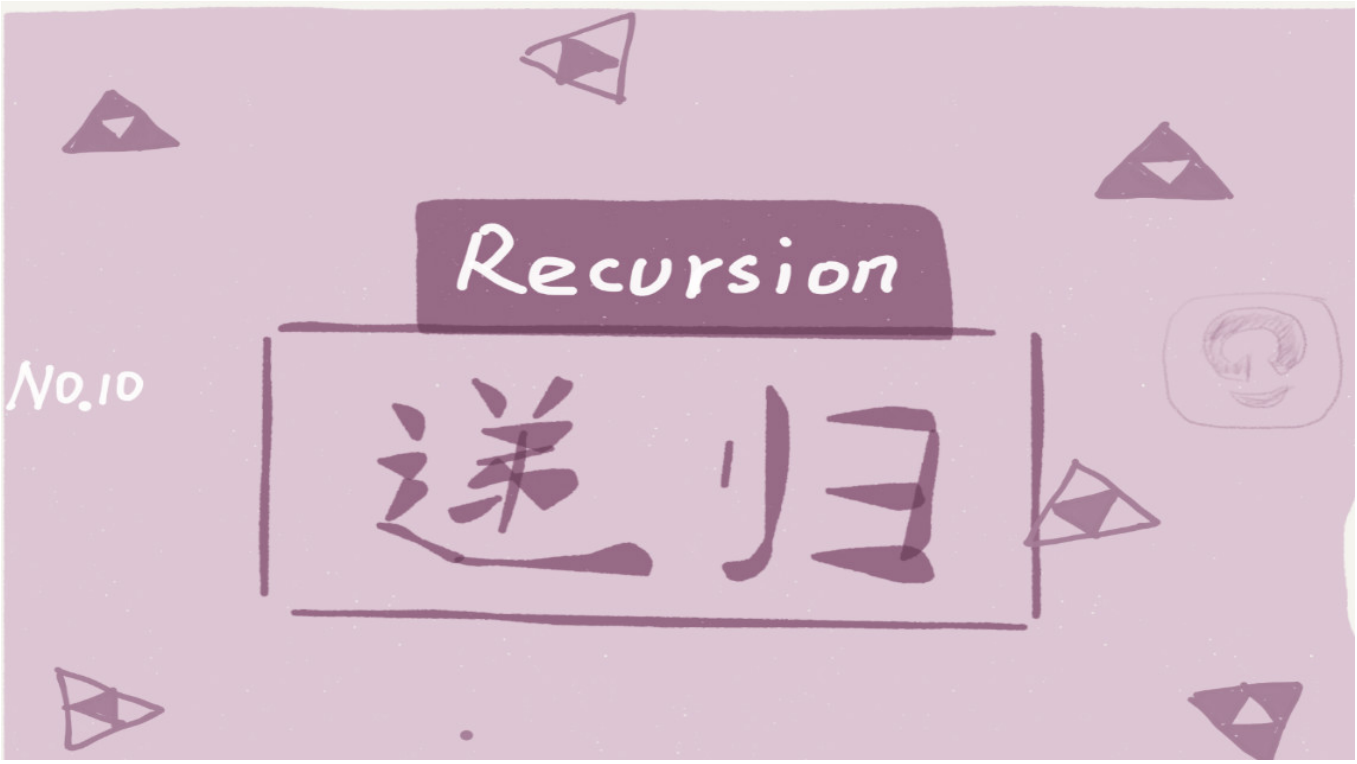
10 | 递归：如何用三行代码找到“最终推荐人”？

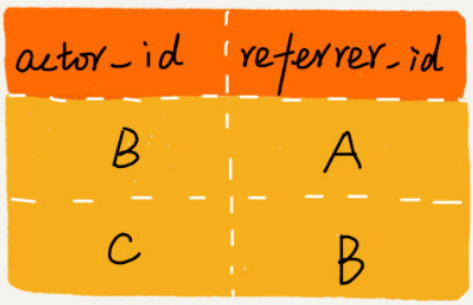
2018-10-12 王争



10 | 递归：如何用三行代码找到“最终推荐人”？
朗读人：修阳　15′35″ | 7.15M

推荐注册返佣金的这个功能我想你应该不陌生吧？现在很多 App 都有这个功能。这个功能中，用户 A 推荐用户 B 来注册，用户 B 又推荐了用户 C 来注册。我们可以说，用户 C 的“最终推荐人”为用户 A，用户 B 的“最终推荐人”也为用户 A，而用户 A 没有“最终推荐人”。

一般来说，我们会通过数据库来记录这种推荐关系。在数据库表中，我们可以记录两行数据，其中 actor_id 表示用户 id，referrer_id 表示推荐人 id。



基于这个背景，我的问题是，给定一个用户 ID，如何查找这个用户的“最终推荐人”？ 带着这个问题，我们来学习今天的内容，递归（Recursion）！

如何理解“递归”？

从我自己学习数据结构和算法的经历来看，我个人觉得，有两个最难理解的知识点，一个是**动态规划**，另一个就是**递归**。

递归是一种应用非常广泛的算法（或者编程技巧）。之后我们要讲的很多数据结构和算法的编码实现都要用到递归，比如 DFS 深度优先搜索、前中后序二叉树遍历等等。所以，搞懂递归非常重要，否则，后面复杂一些的数据结构和算法学起来就会比较吃力。

不过，别看我说了这么多，递归本身可是一点儿都不“高冷”，咱们生活中就有很多用到递归的例子。

周末你带着女朋友去电影院看电影，女朋友问你，咱们现在坐在第几排啊？电影院里面太黑了，看不清，没法数，现在你怎么办？

别忘了你是程序员，这个可难不倒你，递归就开始排上用场了。于是你就问前面一排的人他是第几排，你想只要在他的数字上加一，就知道自己在哪一排了。但是，前面的人也看不清啊，所以他也问他前面的人。就这样一排一排往前问，直到问到第一排的人，说我在第一排，然后再这样一排一排再把数字传回来。直到你前面的人告诉你他在哪一排，于是你就知道答案了。

这就是一个非常标准的递归求解问题的分解过程，去的过程叫“递”，回来的过程叫“归”。基本上，所有的递归问题都可以用递推公式来表示。刚刚这个生活中的例子，我们用递推公式将它表示出来就是这样的：

```
f(n)=f(n-1)+1 其中，f(1)=1
```

复制代码

f(n) 表示你想知道自己在哪一排，f(n-1) 表示前面一排所在的排数，f(1)=1 表示第一排的人知道自己在第一排。有了这个递推公式，我们就可以很轻松地将它改为递归代码，如下：

```
int f(int n) {
  if (n == 1) return 1;
  return f(n-1) + 1;
}
```

复制代码

递归需要满足的三个条件

刚刚这个例子是非常典型的递归，那究竟什么样的问题可以用递归来解决呢？我总结了三个条件，只要同时满足以下三个条件，就可以用递归来解决。

1. 一个问题的解可以分解为几个子问题的解

何为子问题？子问题就是数据规模更小的问题。比如，前面讲的电影院的例子，你要知道，“自己在哪一排”的问题，可以分解为“前一排的人在哪一排”这样一个子问题。

2. 这个问题与分解之后的子问题，除了数据规模不同，求解思路完全一样

比如电影院那个例子，你求解“自己在哪一排”的思路，和前面一排人求解“自己在哪一排”的思路，是一模一样的。

3. 存在递归终止条件

把问题分解为子问题，把子问题再分解为子子问题，一层一层分解下去，不能存在无限循环，这就需要有终止条件。

还是电影院的例子，第一排的人不需要再继续询问任何人，就知道自己在哪一排，也就是 f(1)=1，这就是递归的终止条件。

如何编写递归代码?

刚刚铺垫了这么多，现在我们来看，如何来写递归代码？我个人觉得，写递归代码最关键的是**写出递推公式，找到终止条件，**剩下将递推公式转化为代码就很简单了。

你先记住这个理论。我举一个例子，带你一步一步实现一个递归代码，帮你理解。

假如这里有 n 个台阶，每次你可以跨 1 个台阶或者 2 个台阶，请问走这 n 个台阶有多少种走法？如果有 7 个台阶，你可以 2，2，2，1 这样子上去，也可以 1，2，1，1，2 这样子上去，总之走法有很多，那如何用编程求得总共有多少种走法呢？

我们仔细想下，实际上，可以根据第一步的走法把所有走法分为两类，第一类是第一步走了 1 个台阶，另一类是第一步走了 2 个台阶。所以 n 个台阶的走法就等于先走 1 阶后，n-1 个台阶的走法 加上先走 2 阶后，n-2 个台阶的走法。用公式表示就是：

```
1 f(n) = f(n-1)+f(n-2)
```

复制代码

有了递推公式，递归代码基本上就完成了一半。我们再来看下终止条件。当有一个台阶时，我们不需要再继续递归，就只有一种走法。所以 f(1)=1。这个递归终止条件足够吗？我们可以用 n=2，n=3 这样比较小的数试验一下。

n=2 时，f(2)=f(1)+f(0)。如果递归终止条件只有一个 f(1)=1，那 f(2) 就无法求解了。所以除了 f(1)=1 这一个递归终止条件外，还要有 f(0)=1，表示走 0 个台阶有一种走法，不过这样子看起来

就不符合正常的逻辑思维了。所以，我们可以把 f(2)=2 作为一种终止条件，表示走 2 个台阶，有两种走法，一步走完或者分两步来走。

所以，递归终止条件就是 f(1)=1，f(2)=2。这个时候，你可以再拿 n=3，n=4 来验证一下，这个终止条件是否足够并且正确。

我们把递归终止条件和刚刚得到的递推公式放到一起就是这样的：

复制代码

```
f(1) = 1;
f(2) = 2;
f(n) = f(n-1)+f(n-2)
```

有了这个公式，我们转化成递归代码就简单多了。最终的递归代码是这样的：

复制代码

```
int f(int n) {
  if (n == 1) return 1;
  if (n == 2) return 2;
  return f(n-1) + f(n-2);
}
```

我总结一下，**写递归代码的关键就是找到如何将大问题分解为小问题的规律，并且基于此写出递推公式，然后再推敲终止条件，最后将递推公式和终止条件翻译成代码**。

虽然我讲了这么多方法，但是作为初学者的你，现在是不是还是有种想不太清楚的感觉呢？实际上，我刚学递归的时候，也有这种感觉，这也是文章开头我说递归代码比较难理解的地方。

刚讲的电影院的例子，我们的递归调用只有一个分支，也就是说“一个问题只需要分解为一个子问题”，我们很容易能够想清楚“递“和”归”的每一个步骤，所以写起来、理解起来都不难。

但是，当我们面对的是一个问题要分解为多个子问题的情况，递归代码就没那么好理解了。

像我刚刚讲的第二个例子，人脑几乎没办法把整个“递”和“归”的过程一步一步都想清楚。

计算机擅长做重复的事情，所以递归正和它的胃口。而我们人脑更喜欢平铺直叙的思维方式。当我们看到递归时，我们总想把递归平铺展开，脑子里就会循环，一层一层往下调，然后再一层一层返回，试图想搞清楚计算机每一步都是怎么执行的，这样就很容易被绕进去。

对于递归代码，这种试图想清楚整个递和归过程的做法，实际上是进入了一个思维误区。很多时候，我们理解起来比较吃力，主要原因就是自己给自己制造了这种理解障碍。那正确的思维方式应该是怎样的呢？

如果一个问题 A 可以分解为若干子问题 B、C、D，你可以假设子问题 B、C、D 已经解决，在此基础上思考如何解决问题 A。而且，你只需要思考问题 A 与子问题 B、C、D 两层之间的关系即可，不需要一层一层往下思考子问题与子子问题，子子问题与子子子问题之间的关系。屏蔽掉递归细节，这样子理解起来就简单多了。

因此，**编写递归代码的关键是，只要遇到递归，我们就把它抽象成一个递推公式，不用想一层层的调用关系，不要试图用人脑去分解递归的每个步骤。**

递归代码要警惕堆栈溢出

在实际的软件开发中，编写递归代码时，我们会遇到很多问题，比如堆栈溢出。而堆栈溢出会造成系统性崩溃，后果会非常严重。为什么递归代码容易造成堆栈溢出呢？我们又该如何预防堆栈溢出呢？

我在“栈”那一节讲过，函数调用会使用栈来保存临时变量。每调用一个函数，都会将临时变量封装为栈帧压入内存栈，等函数执行完成返回时，才出栈。系统栈或者虚拟机栈空间一般都不大。如果递归求解的数据规模很大，调用层次很深，一直压入栈，就会有堆栈溢出的风险。

比如前面的讲到的电影院的例子，如果我们将系统栈或者 JVM 堆栈大小设置为 1KB，在求解 f(19999) 时便会出现如下堆栈报错：

```
Exception in thread "main" java.lang.StackOverflowError
```

复制代码

那么，如何避免出现堆栈溢出呢？

我们可以通过在代码中限制递归调用的最大深度的方式来解决这个问题。递归调用超过一定深度（比如 1000）之后，我们就不继续往下再递归了，直接返回报错。还是电影院那个例子，我们可以改造成下面这样子，就可以避免堆栈溢出了。不过，我写的代码是伪代码，为了代码简洁，有些边界条件没有考虑，比如 x<=0。

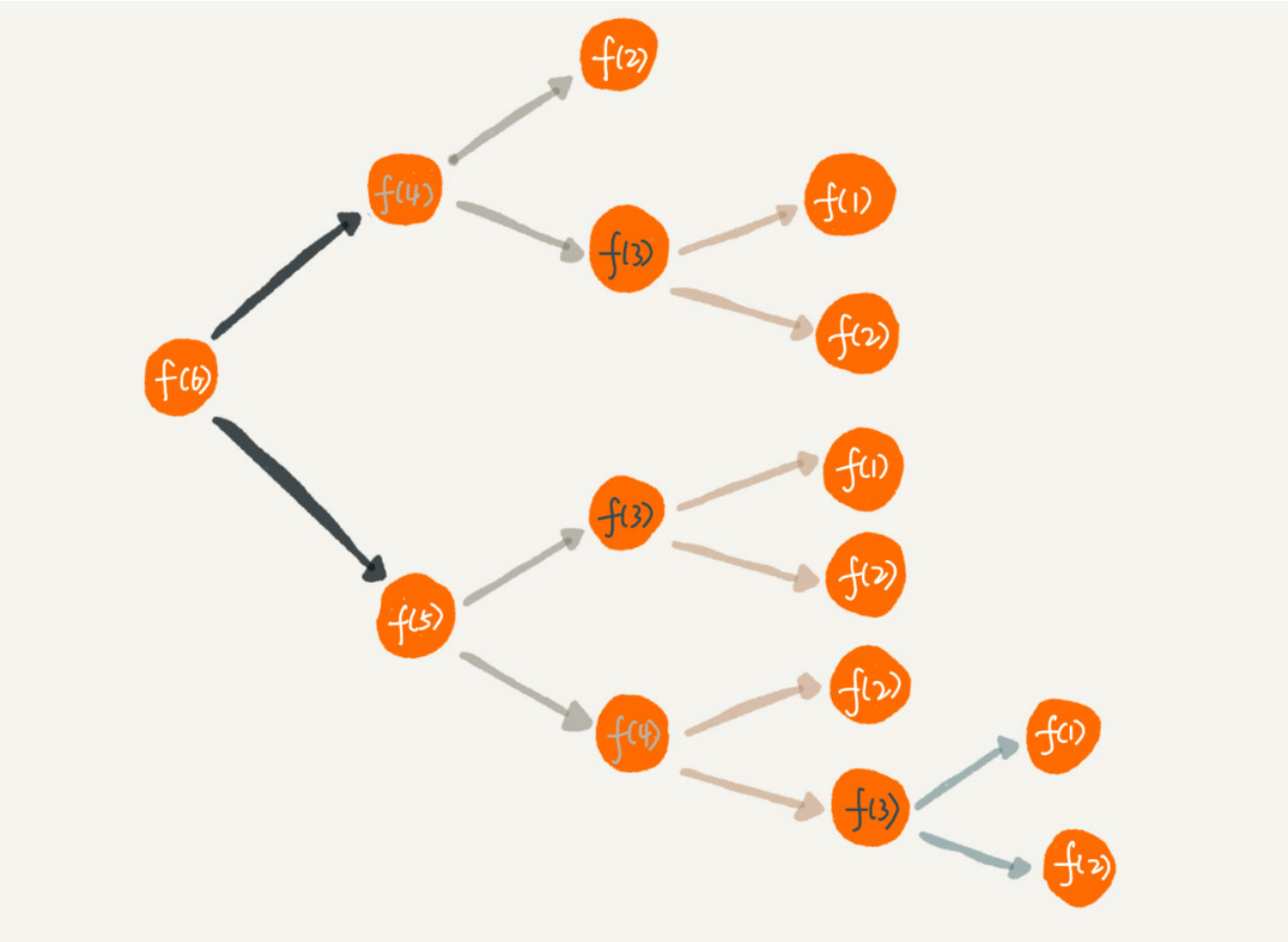
```
// 全局变量，表示递归的深度。
int depth = 0;

int f(int n) {
  ++depth;
  if (depth > 1000) throw exception;

  if (n == 1) return 1;
  return f(n-1) + 1;
}
```

复制代码

但这种做法并不能完全解决问题，因为最大允许的递归深度跟当前线程剩余的栈空间大小有关，事先无法计算。如果实时计算，代码过于复杂，就会影响代码的可读性。所以，如果最大深度比较小，比如 10、50，就可以用这种方法，否则这种方法并不是很实用。

递归代码要警惕重复计算

除此之外，使用递归时还会出现重复计算的问题。刚才我讲的第二个递归代码的例子，如果我们把整个递归过程分解一下的话，那就是这样的：



从图中，我们可以直观地看到，想要计算 f(5)，需要先计算 f(4) 和 f(3)，而计算 f(4) 还需要计算 f(3)，因此，f(3) 就被计算了很多次，这就是重复计算问题。

为了避免重复计算，我们可以通过一个数据结构（比如散列表）来保存已经求解过的 f(k)。当递归调用到 f(k) 时，先看下是否已经求解过了。如果是，则直接从散列表中取值返回，不需要重复计算，这样就能避免刚讲的问题了。

按照上面的思路，我们来改造一下刚才的代码：

复制代码

```
1 public int f(int n) {
2   if (n == 1) return 1;
3   if (n == 2) return 2;
4
```

```
  // hasSolvedList 可以理解成一个 Map，key 是 n，value 是 f(n)
  if (hasSolvedList.containsKey(n)) {
    return hasSovledList.get(n);
  }

  int ret = f(n-1) + f(n-2);
  hasSovledList.put(n, ret);
  return ret;
}
```

除了堆栈溢出、重复计算这两个常见的问题。递归代码还有很多别的问题。

在时间效率上，递归代码里多了很多函数调用，当这些函数调用的数量较大时，就会积聚成一个可观的时间成本。在空间复杂度上，因为递归调用一次就会在内存栈中保存一次现场数据，所以在分析递归代码空间复杂度时，需要额外考虑这部分的开销，比如我们前面讲到的电影院递归代码，空间复杂度并不是 O(1)，而是 O(n)。

怎么将递归代码改写为非递归代码?

我们刚说了，递归有利有弊，利是递归代码的表达力很强，写起来非常简洁；而弊就是空间复杂度高、有堆栈溢出的风险、存在重复计算、过多的函数调用会耗时较多等问题。所以，在开发过程中，我们要根据实际情况来选择是否需要用递归的方式来实现。

那我们是否可以把递归代码改写为非递归代码呢？比如刚才那个电影院的例子，我们抛开场景，只看 f(x) =f(x–1)+1 这个递推公式。我们这样改写看看：

复制代码

```
int f(int n) {
  int ret = 1;
  for (int i = 2; i <= n; ++i) {
    ret = ret + 1;
  }
  return ret;
}
```

同样，第二个例子也可以改为非递归的实现方式。

复制代码

```
int f(int n) {
  if (n == 1) return 1;
  if (n == 2) return 2;

  int ret = 0;
  int pre = 2;
  int prepre = 1;
  for (int i = 3; i <= n; ++i) {
    ret = pre + prepre;
    prepre = pre;
```

```
    pre = ret;
  }
  return ret;
}
```

那是不是所有的递归代码都可以改为这种**迭代循环**的非递归写法呢?

笼统地讲，是的。因为递归本身就是借助栈来实现的，只不过我们使用的栈是系统或者虚拟机本身提供的，我们没有感知罢了。如果我们自己在内存堆上实现栈，手动模拟入栈、出栈过程，这样任何递归代码都可以改写成看上去不是递归代码的样子。

但是这种思路实际上是将递归改为了“手动”递归，本质并没有变，而且也并没有解决前面讲到的某些问题，徒增了实现的复杂度。

解答开篇

到此为止，递归相关的基础知识已经讲完了，咱们来看一下开篇的问题：如何找到“最终推荐人”？我的解决方案是这样的：

复制代码

```
long findRootReferrerId(long actorId) {
  Long referrerId = select referrer_id from [table] where actor_id = actorId;
  if (referrerId == null) return actorId;
  return findRootReferrerId(referrerId);
}
```

是不是非常简洁？用三行代码就能搞定了，不过在实际项目中，上面的代码并不能工作，为什么呢？这里面有两个问题。

第一，如果递归很深，可能会有堆栈溢出的问题。

第二，如果数据库里存在脏数据，我们还需要处理由此产生的无限递归问题。比如 demo 环境下数据库中，测试工程师为了方便测试，会人为地插入一些数据，就会出现脏数据。如果 A 的推荐人是 B，B 的推荐人是 C，C 的推荐人是 A，这样就会发生死循环。

第一个问题，我前面已经解答过了，可以用限制递归深度来解决。第二个问题，也可以用限制递归深度来解决。不过，还有一个更高级的处理方法，就是自动检测 A-B-C-A 这种“环”的存在。如何来检测环的存在呢？这个我暂时不细说，你可以自己思考下，后面的章节我们还会讲。

内容小结

关于递归的知识，到这里就算全部讲完了。我来总结一下。

递归是一种非常高效、简洁的编码技巧。只要是满足“三个条件”的问题就可以通过递归代码来解决。

不过递归代码也比较难写、难理解。编写递归代码的关键就是不要把自己绕进去，正确姿势是写出递推公式，找出终止条件，然后再翻译成递归代码。

递归代码虽然简洁高效，但是，递归代码也有很多弊端。比如，堆栈溢出、重复计算、函数调用耗时多、空间复杂度高等，所以，在编写递归代码的时候，一定要控制好这些副作用。

课后思考

我们平时调试代码喜欢使用 IDE 的单步跟踪功能，像规模比较大、递归层次很深的递归代码，几乎无法使用这种调试方式。对于递归代码，你有什么好的调试方法呢?

欢迎留言和我分享，我会第一时间给你反馈。

写留言

精选留言

墨墨 4

老师好，你的github地址可以发下吗？我在前面的章节没看到

2018-10-12

作者回复

github上搜wangzheng0822

2018-10-12

张三的锅锅 1

递归时间复杂度觉得比较复杂

2018-10-12

作者回复

后面会讲两种分析方法 在排序和树两节课中

2018-10-12

Monday 1

原以为老师会先讲完10个基本的数据结构再讲十种基本的算法。没想到老师会穿插着讲。冒昧的问下老师设计课程的思路。谢谢

2018-10-12

作者回复

因为后面的内容会用到递归 而递归不依赖后面的内容 拓扑排序了解一下😄

2018-10-12

博金 47

调试递归:

1.打印日志发现，递归值。

2.结合条件断点进行调试。

2018-10-12

作者回复

答案正确 大家可以把这一条顶上去

2018-10-13

一步 19

哈哈，在电影院看是第几排，我直接看电影票，直接用索引找到了

2018-10-12

作者回复

哈哈😄

2018-10-12

zl 12

Debug不行就打日志

2018-10-12

一步 7

说的对的，每次写递归代码，或者看递归代码，都会不自觉的在大脑中复现整个递和归的过程

2018-10-12

姜威 6

总结

一、什么是递归?

1.递归是一种非常高效、简洁的编码技巧，一种应用非常广泛的算法，比如DFS深度优先搜索、前中后序二叉树遍历等都是使用递归。

2.方法或函数调用自身的方式称为递归调用，调用称为递，返回称为归。
3.基本上，所有的递归问题都可以用递推公式来表示，比如
f(n) = f(n–1) + 1;
f(n) = f(n–1) + f(n–2);
f(n)=n*f(n–1);

二、为什么使用递归？递归的优缺点？

1.优点：代码的表达力很强，写起来简洁。
2.缺点：空间复杂度高、有堆栈溢出风险、存在重复计算、过多的函数调用会耗时较多等问题。

三、什么样的问题可以用递归解决呢？

一个问题只要同时满足以下3个条件，就可以用递归来解决：
1.问题的解可以分解为几个子问题的解。何为子问题？就是数据规模更小的问题。
2.问题与子问题，除了数据规模不同，求解思路完全一样
3.存在递归终止条件

四、如何实现递归？

1.递归代码编写
写递归代码的关键就是找到如何将大问题分解为小问题的规律，并且基于此写出递推公式，然后再推敲终止条件，最后将递推公式和终止条件翻译成代码。
2.递归代码理解
对于递归代码，若试图想清楚整个递和归的过程，实际上是进入了一个思维误区。
那该如何理解递归代码呢？如果一个问题A可以分解为若干个子问题B、C、D，你可以假设子问题B、C、D已经解决。而且，你只需要思考问题A与子问题B、C、D两层之间的关系即可，不需要一层层往下思考子问题与子子问题，子子问题与子子子问题之间的关系。屏蔽掉递归细节，这样子理解起来就简单多了。
因此，理解递归代码，就把它抽象成一个递推公式，不用想一层层的调用关系，不要试图用人脑去分解递归的每个步骤。

五、递归常见问题及解决方案

1.警惕堆栈溢出：可以声明一个全局变量来控制递归的深度，从而避免堆栈溢出。
2.警惕重复计算：通过某种数据结构来保存已经求解过的值，从而避免重复计算。

六、如何将递归改写为非递归代码？

笼统的讲，所有的递归代码都可以改写为迭代循环的非递归写法。如何做？抽象出递推公式、初始值和边界条件，然后用迭代循环实现。

2018-10-12

Smallfly 5

界定问题能否用递归解决
1. 一个问题的解可以分解为几个子问题的解；
2. 这个问题与分解子问题的求解思路完全相同；
3. 存在终止条件

编写递归代码的技巧
1. 终止条件
2. 递推公式
3. 清理现场

编写递归的关键是思考终止条件，把问题抽象成一个递推公式，并信任它一定能帮我们完成任务，不用想一层层的调用关系，试图用人脑分解递归是反人类的，最多只能想两三层。

递归的缺点
递归会利用栈保存临时变量，如果递归过深，会造成栈溢出。解决方案是控制递归的深度。

递归要警惕重复计算，递归分解的子问题、子子问题可能存在相同的情况，如果都一一计算的话，就会发生重复计算。解决方案是使用散列表来保存结算结果，每次开始计算前检查散列表是否已经有结算结果。

笼统地讲，递归代码都能用迭代循环来替换。

免费加入知识星球「极客星球」讨论算法问题。

2018-10-12

失火的夏天 5

检测环可以构造一个set集合或者散列表(下面都叫散列表吧，为了方便)。每次获取到上层推荐人就去散列表里先查，没有查到的话就加入，如果存在则表示存在环了。当然，每一次查询都是一个自己的散列表，不能共用。不过这样请求量大的话，会不会造成内存空间开辟太多？这里老师能帮忙解答一下吗？

2018-10-12

作者回复

你这种办法可行的 👍。实际情况 内存也不会耗太多

2018-10-12

刘強 4

那个陷入思维误区的说法产生共鸣了，原来总以为自己脑容量不足，看来牛人也一样。

2018-10-12

lovetechlovelife 3

检测环的答案其实老师已经在文章中的一个例子中讲过了，就是用一个数据结构把查询过的元素放到这个数据结构里面，新来的就先查这个数据结构里面有没有，有就返回，没有就放入到这个数据结构里面。

2018-10-12

风起 3

调试递归就像写递归一样，
不要被每一步的细节所困，
重点在于确认递推关系与结束条件是否正确，
用条件断点着重调试最初两步与最终两步即可。

2018-10-12

柠檬C 2

递归和数学归纳法非常像，所以可以利用数学归纳法的思路，先验证边界条件，再假设n-1的情况正确，思考n和n-1的关系写出递推公式

2018-10-12

范柏柏 2

希望老师多分享一些经典习题。比如链表那一章课后所说的，掌握这几道题就基本掌握了链表。

2018-10-12

跬行 2

递归调试 确实好难 每次都是日志输出 每一步的结果 看问题在哪里

2018-10-12

微秒 2

不是很懂第二个例子非递归的实现方式是什么意思，可以讲的清楚一点吗?

2018-10-12

作者回复

你可以网上搜下斐波那契数列的代码实现

2018-10-13

汪木木 2

递推公式+边界条件

2018-10-12

Jessie 2

n=4,
f(n)=f(n-1)+f(n-2);
所以,
f(4)=f(3)+f(2)
=f(2)+f(1)+f(2)

=2+1+2=5

4个台阶，变5个台阶了，请问以上推理哪里出问题了?

2018-10-12

L 2

解答楼上的问题，数据规模较大的情况用循环，也就是老师讲的非递归代码

2018-10-12

作者回复

是的 👍

2018-10-12

涛 2

终于在认知层面得到了提升，递归是什么，在我看来递归就是用栈的数据结构，加上一个简单的逻辑算法实现了业务功能。

2018-10-12

作者回复

👍

2018-10-12

Geek_8a2f3f 2

王老师，你好！说那个限制递归深度的做法只适合规模比较小的情况，那如果规模大了，怎么限制呢?

2018-10-12

作者回复

自己模拟一个栈 用非递归代码实现

2018-10-12

Monday 1

老师，递归算法的时间复杂度，请给我们像前面章节那样分析分析。

2018-10-12

作者回复

在排序和树那两节课会讲两种递归代码的时间复杂度分析方法

2018-10-13

跬行 1

递归调试 确实好难 每次都是日志输出 每一步的结果 看问题在哪里

2018-10-12

传说中的成大大 1

1. 关于环形推荐人那个问题我又想到了判断链表环的,快慢指针法 每次查询分两步比如都从A开始 下步操作中 快的查C的推荐人慢的查B的推荐人 如果某一次查的人相同 比如都是A或者B或者C即为有环

2. 我觉得还有一种办法 叫最终推荐人法，即我们每一次查询的时候都记录下当前查找的人比如A 做一个映射map[A] = ?, 推荐人为B 则map[A] = B, 如果B还有推荐人C 我们更新map[A] = C 如果没有推荐人了 清空map，如果C还有推荐人则继续更新map[A] = ?,再每一次查询完了过后我们验证key是否等于value 则可得出是否有环，这方法的复杂度比上面方法要好多的多就是O(1),如果要查B则映射map[b] = ?以此类推

补充一下：
只检查A有点牵强，我思考过后又觉得 比如我最开始检查A 先map[A] = ?,如果得到A的推荐人B 则再做map[B] = ?，保留map[A] = ?,再接下来的检查推荐人过程中map[A] map[B] 都要更新value,并且每一个映射都要检查key是否等于value则可以得出是否有环，这次应该考虑周全了

2018-10-12

Max 1

pre和prepre代表的什么?

2018-10-12

作者回复

f（n-1）和f（n-2）

2018-10-12

哲人之石 1

定期存款利息计算可以用递归吗?
首先找到递推公式是：benjin =本金+本金*利率,
其次退出条件是：存5年
Int fuli(int benjin, int year, double lilv):
Benjin=benjin+benjin*lilv
Year=year-1
If (year < 5){
Return fuli(bebjin,year,lilv)
}else{
Return benjin
}
我这个程序写的对吗？而且感觉很复杂，请老师指导?

2018-10-12

作者回复

好像有点不对 先写递推公式吧 然后再写代码 你写的那个有点不是规整的递推公式

2018-10-12

along 1

最终推荐人避免出现死循环，可以用前面学的链表检测环的方式来实现

2018-10-12

DZuo 1

在路上，想了想，短点打到结束语条件位置，向前调试

2018-10-12

肖小强 0

老师，那个电影票递归的空间复杂度是O(n),如果用尾递归的话，就是O(1)吧

2018-10-14

秋天，不远了 0

哈哈，我就是文中说的那种人，写递归时要一步一步地想清楚，调试的时候用ide一步一步看清！！！

2018-10-14

Amari 0

int f(int n) {
if (n == 1) return 1;
return f(n-1) + 1;
}
老师，f(n-1)表示前一排，那再加1是啥意思，没懂，求解释

2018-10-14

张驰 0

关于非递归徒增实现复杂度这点我有意见：1）非递归可共享变量空间，大幅度降低空间复杂度。2）解决堆栈溢出除了限制迭代次数外，首要考虑的应该是尾递归优化。望指正观点。

2018-10-14

张驰 0

像楼梯那种，拆分到多个子问题的迭代算法，都是O（2^n）的时间复杂度，在真实场景中很难落实吧？感觉看到这种思路都会直接使用近视的估算算法了。王老师在现实实践中怎么解决这类问题?

2018-10-14

Rain 0

Re Jessie:
----
n=4,
f(n)=f(n-1)+f(n-2);
所以,
f(4)=f(3)+f(2)
=f(2)+f(1)+f(2)
=2+1+2=5
4个台阶，变5个台阶了，请问以上推理哪里出问题了?

不是4个台阶变5个台阶，是4个台阶有5种下法。

2018-10-14

往事随风，顺其自然 0

两个节点之间赋值，p=q和p-next=q有什么区别，一直没没分清

2018-10-13

wean 0

调试递归代码的方式，同样适合调试循环轮次很高的情况

2018-10-13

周平 0

递归的错误思路是，在脑中把递归展开。

正确的应该是， 找到如何将大问题分解为小问题的规律，并且基于此写出递推公式，然后再推敲终止条件，最后将递推公式和终止条件翻译成代码。

递归的错误的思维方式，还需要多多练习才行。

老师有没有一些递归的题，供我们练习一下啊?

2018-10-13

insist 0

递归是一种将大问题分解为小问题的编程技巧，优点是代码简洁，容易理解；缺点是如果递归层次过多，容易导致堆栈溢出，可能存在重复计算问题，空间复杂度也比较大。写递归代码思路：将大问题分解为数据规模小一些，但处理逻辑一致的小问题，接着推导出递推公式，推导出终止条件，最后翻译成代码。

这次课对递归有了新的理解，分为递和归两个步骤。就像坐公交，从后面上时候，让前一个人把公交卡递过去刷一下，刷完了再原路归回来。

2018-10-13

前进前进 0

老师后续会讲尾递归？尾递归能减少空间复杂度，避免堆栈溢出。

2018-10-13

阳仔 0

学习反馈：

递归是计算机算法里广泛使用的一种算法。它的实现代码简洁，但理解“递归”需要下点功夫。

递归算法需要满足的三个条件：

1、规模大的问题可以分解为规模较小的问题；

2、规模大的问题与规模较小问题的求解方式是一样的；

3、要有终止条件。

如何写一个递归算法：

关键点是将大规模问题分解成小规模问题，推导出递推公式，并找到终止条件，最后把递推公

式和终止条件翻译成代码。
递归算法需要注意的问题：
1、栈溢出
由于递归算法是基于计算机内部的函数调用栈的数据结构，每调用一次函数，就执行入栈操作，函数返回时才执行出栈操作，因此当函数调用层次太多，就有可能发生栈溢出的问题。
如何避免栈溢出
在递归代码中使用最大深度变量来控制。这种方式不是特别好，因为要知道计算机栈的深度，也使得编码复杂度变高。
2、避免重复计算
这个问题可以使用散列表来缓存已经计算过的值来解决。
3、其他问题
时间复杂度和空间复杂度都会有很高的开销
递归算法是一种简洁有效的编码思想，但是递归算法也有缺点，空间和时间复杂度都比较大。一般递归代码都可以写成迭代循环的形式。因此在实际编码中，选择递归或者迭代需要具体问题具体分析

2018-10-13

天若有情天亦老 0
费布那齐汤 就是递归的思想

2018-10-13

纯洁的憎恶 0
好留言不能做笔记，只能临时记一下了……

调试递归:
1.打印日志发现，递归值。
2.结合条件断点进行调试。

2018-10-13

cjling 0
systemtap等动态调试工具

2018-10-13

Jason 0
老师，看到有同学通过递归计算定期存款，我试着写了一下，麻烦老师看看。
……
定期存款利息计算可以用递归吗?
首先找到递推公式是：benjin =本金+本金*利率,
其次退出条件是：存5年
Int fuli(int benjin, int year, double lilv):
Benjin=benjin+benjin*lilv
Year=year-1

If (year < 5){
Return fuli(bebjin,year,lilv)
}else{
Return benjin
}
我这个程序写的对吗？而且感觉很复杂，请老师指导？
.....
以上是那位同学的留言，下面是我写的：
假如在银行定存一笔钱，一年以上每年固定利率为lilv，根据所存年数计算存息和（存款和利息总和）
首先找到递推公式是：存息 = 本金 * (1+利率)
f(n) = f(n–1)*(1+lilv)
终止条件有两个，1.定存不到一年，f(0) = 本金;2.定存只有一年，(1) = 本金 * (1+利率)
把递归终止条件和刚刚得到的递推公式放到一起就是这样的：
double f(int y){
if(y == 0) return 本金；
if(y == 1) return 本金* (1+lilv)；
return f(y–1)*(1+lilv)；
}
老师，这样分析对吗？

2018–10–13

Jason 0

如何来检测环的存在呢？
可否用前面学的“栈”存储结构，每次入栈时，判断栈里是否有这个元素存在，例如：A–B–C–A，当A再次入栈时，栈内存中已存在元素A，说明此时递归是一个环,栈顶的A就是最终推荐人；或者用队列为存储结构，当A再次入队时，A–B–C–A，队列头部的A就是最终推荐人。
老师，这种分析思路对吗？

2018–10–13

无名 0

最终推荐人有环的情况可以用递归的深度超过表记录的总长度的方法解决，感觉。

2018–10–13

liangjf 0

数据规模小结果正确，而数据规模大结果出错。面对递归这种 公式固定的算法，出错的原因应该是系统相关的吧，栈溢出，切换时保存现场出错，内存换页等原因吧？

2018–10–13

nil 0

想问老师，尾递归的方式是否能解决递归的堆栈溢出的问题

2018–10–13

liangjf

liangjr

0

看完觉得顿悟了，之前是用人脑模拟递归，现在看来是不合适的。

而应该像调试递归那样，人脑给定条件限定，"断点"调试模拟最初的一两步，和最终的一两步。

2018-10-13

Zhangwh

0

之前都没做笔记，但是这一篇写的实在是很动人，不自觉的就记下一些老师的经验总结，来加深理解

2018-10-13

崇拜

0

首先你得有一个女朋友才能解决看电影问题

2018-10-13

女神，come on

0

老师那个递归调试的两个方法，没看懂，能详细的解释一下吗？比如：如火热结合条件断点？这里的条件断点指的是什么？打印日志，是将递归值都打印出来吗？这样如果堆栈溢出还能打印出来吗？会不会编译的时候程序直接报错？我是小白，有时间请详细解释一波

2018-10-13

.小、挂念

0

看了老师的文章，就想着把前几天弄的那个菜单树换成用递归去实现。代码是简洁了，但极易死循环，调适起来也不是很容易

2018-10-13

DDT

0

我觉得现在老师已经把我从思维误区里面带出来了 我现在是怎么把一个大问题分成几个小问题的时候 会卡壳...

2018-10-13

吴...

0

老师，我想请教一下，你觉得机器学习与传统的数据结构算法这些有什么联系吗？如果我想学机器学习是否需要很强的编程能力？

2018-10-13

为你而来

0

用之前讲的链表来检测环😄

2018-10-13

yaxin

0

递归步骤：
1.大问题拆成小问题
2.找到终止条件
3.写代码

递归算法的空间复杂度为n，很容易内存溢出。解决方法是将已经算出结果的存起来，下次直接用。

2018-10-13

墨梵 0

之前写递归时候就是陷入了层层递归的计算误区、结果把自己陷进去了，老师这篇文章，颇有醍醐灌顶的感觉

2018-10-13

晚风·和煦 0

老师您好，一名有过其他语言学习的学生，在您看来应该如何系统的学好一门语言。

2018-10-13

Realm 0

n=4,
f(n)=f(n−1)+f(n−2);
所以,
f(4)=f(3)+f(2)
=f(2)+f(1)+f(2)
=2+1+2=5
4个台阶，变5个台阶了，请问以上推理哪里出问题了?

这里最后的结果是步数，不是台阶数，其实结果是一个1 2 3 5 8...肥波拉切

2018-10-13

侯金彪 0

老师我还有个问题，就是在所有操作递归调用本函数之前和在递归调用本函数之后的思维方式是不是不太一样呢？比如用递归处理链表反转和递归处理斐波那契数列

2018-10-13

ADkun 0

老师，将第二个递归算法改成非递归的那段代码不太明白，可否注释一下？每个变量分别表示什么？循环里面操作的意义？

2018-10-12

作者回复

我下周添加些注释，让编辑再重新更新下代码吧

2018-10-13

snakorse 0

部分编译器支持对尾递归的堆栈优化，可以避免堆栈溢出

2018-10-12

作者回复

尾递归也并不能完全避免堆栈溢出的

2018-10-13

mj 0

我对台阶问题的理解是:到达n阶只可能来自n-1和n-2,所以f(n)=f(n-1)+f(n-2)

2018-10-12

作者回复

你理解的也对

2018-10-13

侯金彪 0

在debug的时候不要过多关注具体的局部变量，重点关注递归函数的返回值，这样可以吗?

2018-10-12

作者回复

有时候这样子还不够能定位到问题

2018-10-13

leo 0

利用老师讲的方法分析下汉诺塔问题:
有a、b、c三个柱子，a柱子上从小到大码放n个盘子
要将盘子从a移动到c，依然从小到大码放
移动期间小盘子不能放到大盘子下面

1.问题拆分子问题
如果要将n个盘子从from柱借助assist柱移动到to柱，需要分三步
第一步：将第n个盘子上面的n-1个盘子（即n上面的盘子）从from柱子借助to柱移动到assist柱
第二步：将第n个盘子从from柱移动到to柱
第三步：将之前在assist柱子上面的n-1盘在从assist柱借助from柱移动到to柱

2.公式
move(n, from, to, assist)=
move(n-1, from, assist, to)
move n 从 from 到 to
move(n-1, assist, to, from)

3.终止条件
当只剩下一个盘子时，只需要将这个盘子从from柱移动到to柱

代码如下（Golang）：

```
func Hanoi(n uint) {
if n == 0 {
return
}
move(n, "a", "c", "b")
}

func move(num uint, from, to, assist string) {
if num == 1 {
fmt.Printf("move %+v from %+v to %+v\n", 1, from, to)
return
}
move(num-1, from, assist, to)
fmt.Printf("move %+v from %+v to %+v\n", num, from, to)
move(num-1, assist, to, from)
}
```

2018-10-12

作者回复

👍

2018-10-13

Geek_d5ff62 0

终于把困惑已久的知识点补上了，不过还是需要多看几遍仔细琢磨其精髓，再加上多加练习才能深刻领悟，希望老师可以多加一些练习题的内容，感谢。

2018-10-12

勤劳的小胖子-libo 0

大爱，对于递归原来不应想着展开，会被绕进去。多谢🙏，解决多年来的毛病。

课后问题，可以试着断点，以及动态改变参数的值

2018-10-12

陈亦凡 0

用python，最近学习不写项目都是vim/vscode+pudb

2018-10-12

小景 0

老师讲得真的很精彩，特别是思维误区那里，我以前看递归总是会陷进去，分析问题也因此多花很多时间。要是以后的课程也多这样总结思维层面的东西就好了，理解起来也更有高度。

2018-10-12

作者回复

也不是所有的都能总结出你的痛点，还是很难的，多包涵吧，兄弟。

2018-10-13

杨智晓 ✝

0

典型的递归例子是求阶乘吧f(n)=n*f(n-1)

2018-10-12

作者回复

嗯嗯

2018-10-13

青城

0

前面那个银行利息问题的，大概可以这样写递归?

```
#include<iostream>
using namespace std;
double money(double benjin,double rates,int year){
if(year<1) return benjin;
benjin=money(benjin,rates,year-1)*(1+rates);
return benjin;
}
int main(){
double benjin=10;
double rates=0.5;
cout<<money(10,0.05,5);
return 0;
}
```

2018-10-12

作者回复

正确

2018-10-13

等风来

0

老师:走台阶问题的递归实现的时间复杂度O(n^2),空间复杂度为O(n)

```
int f(int n) {
if (n == 1) return 1;
if (n == 2) return 2;
return f(n-1) + f(n-2);
}
```

使用哈希表记录中间值的话,时间复杂度O(n),空间复杂度为O(n);分析的对吗?

规模比较大、递归层次很深的递归代码可以通过记录日志方式调试.

2018-10-12

作者回复

递归的时间复杂度是O(2^n)指数型的。后面我会分析的，不要着急。

2018-10-13

传说中的成大大

0

验证递归代码 肯定是传一个数据规模较小的数据进去便于验证啊 因为算法固定 数据规模小的都正常了，那么大的也应该没问题,除开那种数据大到堆栈溢出 应该没问题

2018-10-12

作者回复

你说的也对，不过有些bug是在数据规模比较大的时候才能触发

2018-10-13

刘远通

0

把他想成数学归纳法 最重要的就是k到k+1是怎么过去的

2018-10-12

Hurt

0

老师 分析分析 递归的时间复杂度和空间复杂度呗😄

2018-10-12

along

0

老师，第二个例子，优化重复计算问题后时间还变长了很多，是数据规模还不够大吗?

```
private static int f1(int n){
Map<Integer,Integer> hasSolvedList = new HashMap<>();
if (hasSolvedList.containsKey(n)) {
return hasSolvedList.get(n);
}
if(n == 1){
return 1;
}
if (n == 2) {
return 2;
}
int ret = f1(n-1)+f1(n-2);
hasSolvedList.put(n, ret);
return ret;
}
```

print:
当n=40时:
结果:165580141
优化前耗时:765ms
结果:165580141
优化后耗时:5672ms

2018-10-12

喜欢你的笑

0

我的想法是最开始电影院的例子，每一排其实就是相当于是个链表，每一排维护一个链表，想要知道自己第几排，查找链表就可以知道第几排。这个时间复杂度是O（n），空间复杂度也是O(n)，不知道对不对?
还有如果求最终推荐人因为可能有分支，这个是否可以用也用链表实现呢?

2018-10-12

yaya 0

打印？递归解答问题的步骤:找到子问题，找到递归出口。我😂每次遇到问题也喜欢人脑模拟过程。。。尤其在回溯问题的时候。。不是很懂。也明白应该假设小问题都被解决但是还是有点转不过来。。请问有什么好的办法和步骤吗?

2018-10-12

作者回复

没办法了 自己多写多练吧

2018-10-12

ryp 0

最终推荐人怎么能死循环呢，必须通过其他手段避免掉，比如每个人都得记录他前边有n个人，这样递归n次就可以了

2018-10-12

双木公子 0

第二个例子的在没做改进和改进之后的时间复杂度与空间复杂度分别为多少？老师能分析一下吗?

2018-10-12

作者回复

改进前O（2^n） 改进后O（n）

2018-10-12

喜欢你的笑 0

求最终推荐人是否可以用一个链表来进行查找?

2018-10-12

作者回复

具体说说你的思路

2018-10-12

Listen To Me 0

一般会用条件断点debug

2018-10-12

凉粉 0

断点加条件

2018-10-12

沉睡的木木夕

0

作为递归的一种优化方案，尾递归是不是也讲解一下
至于是否有环，可以从gc算法中借鉴，当多个类存在深层次的循环引用，那是不是就是永远无法垃圾回收？其实gc算法中用了引用标记法来解决这种情况

2018-10-12

Linuxer

0

相同的数据规模应该在同一层，如果不同层出现相同数据规模认为是出现环

2018-10-12

Linuxer

0

相同的数据规模应该在同一层，如果不同层出现相同数据规模认为是出现环

2018-10-12

城

0

可以考虑增加调试日志

2018-10-12

bluesea

0

递归出现了问题，但是没有说出解决方案，只讲了一堆理论

2018-10-12

作者回复

我哪个问题没讲解决思路？！是我眼拙吗

2018-10-12

Linuxer

0

我想到的是小规模情况验证，打点两种方式

2018-10-12

□

0

我是小白哈。问下从全局来说，求解斐波那契数，用递归和通项公式，哪个更优？

2018-10-12

作者回复

通项公式 避免递归的副作用

2018-10-12

adapt

0

程序员哪来的女朋友😅😂

2018-10-12

作者回复

那例子就换成男朋友吧😄

2018-10-12

null

0

程序员没女朋友，男朋友有一堆

2018-10-12