

加微信：642945106 发送“赠送”领取赠送精品课程

三 发数字“2”获取众筹列表

下载APP



10 | 动态规划（下）：如何求得状态转移方程并进行编程实现？

2019-01-04 黄申

程序员的数学基础课

[进入课程 >](#)



讲述：黄申

时长 09:51 大小 9.04M



你好，我是黄申。

上一节，我从查询推荐的业务需求出发，介绍了编辑距离的概念，今天我们要基于此，来获得状态转移方程，然后才能进行实际的编码实现。

状态转移方程和编程实现

上一节我讲到了使用状态转移表来展示各个子串之间的关系，以及编辑距离的推导。不过，我没有完成那张表格。现在我把它补全，你可以和我的结果对照一下。

	空B	m	o	u	s	e
空A	0	1	2	3	4	5
m	1	$\min(2, 2, 0)=0$	$\min(3, 1, 2)=1$	$\min(4, 2, 3)=2$	$\min(5, 3, 4)=3$	$\min(6, 4, 5)=4$
o	2	$\min(1, 3, 2)=1$	$\min(2, 3, 0)=0$	$\min(3, 1, 2)=1$	$\min(4, 2, 3)=2$	$\min(5, 3, 4)=3$
u	3	$\min(2, 4, 3)=2$	$\min(1, 3, 2)=1$	$\min(2, 2, 0)=0$	$\min(3, 1, 2)=1$	$\min(4, 2, 3)=2$
u	4	$\min(3, 5, 4)=3$	$\min(2, 4, 3)=2$	$\min(1, 3, 1)=1$	$\min(2, 2, 1)=1$	$\min(3, 2, 2)=2$
s	5	$\min(4, 6, 5)=4$	$\min(3, 5, 4)=3$	$\min(2, 4, 3)=2$	$\min(2, 3, 1)=1$	$\min(3, 2, 2)=2$
e	6	$\min(5, 7, 6)=5$	$\min(4, 6, 5)=4$	$\min(3, 5, 4)=3$	$\min(2, 4, 3)=2$	$\min(3, 3, 1)=1$

这里面求最小值的 \min 函数里有三个参数，分别对应我们上节讲的三种情况的编辑距离，分别是：替换、插入和删除字符。在表格的右下角我标出了两个字符串的编辑距离 1。

概念和分析过程你都理解了，作为程序员，最终还是要落脚在编码上，我这里带你做些编码前的准备工作。

我们假设字符数组 $A[]$ 和 $B[]$ 分别表示字符串 A 和 B ， $A[i]$ 表示字符串 A 中第 i 个位置的字符， $B[i]$ 表示字符串 B 中第 i 个位置的字符。二维数组 $d[]$ 表示刚刚用于推导的二维表格，而 $d[i, j]$ 表示这张表格中第 i 行、第 j 列求得的最终编辑距离。函数 $r(i, j)$ 表示替换时产生的编辑距离。如果 $A[i]$ 和 $B[j]$ 相同，函数的返回值为 0，否则返回值为 1。

有了这些定义，下面我们用迭代来表达上述的推导过程。

如果 i 为 0，且 j 也为 0，那么 $d[i, j]$ 为 0。

如果 i 为 0，且 j 大于 0，那么 $d[i, j]$ 为 j 。

如果 i 大于 0，且 j 为 0，那么 $d[i, j]$ 为 i 。

如果 i 大于 0，且 j 大于 0，那么 $d[i, j] = \min(d[i-1, j] + 1, d[i, j-1] + 1, d[i-1, j-1] + r(i, j))$ 。

这里面最关键的一步是 $d[i, j] = \min(d[i-1, j] + 1, d[i, j-1] + 1, d[i-1, j-1] + r(i, j))$ 。这个表达式表示的是动态规划中从上一个状态到下一个状态之间可能存在的一些变化，以及基于这些变化的最终决策结果。我们把这样的表达式称为**状态转移方程**。我上节最开始就说过，在所有动态规划的解法中，状态转移方程是关键，所以你一定要掌握它。

有了状态转移方程，我们就可以很清晰地用数学的方式，来描述状态转移及其对应的决策过程，而且，有了状态转移方程，具体的编码其实就很容易了。基于编辑距离的状态转移方程，我在这里列出了一种编码的实现，你可以看看。

我们首先要定义函数的参数和返回值，你需要注意判断一下 a 和 b 为 null 的情况。

 复制代码

```
1 public class Lesson10_1 {  
2  
3     /**  
4      * @Description: 使用状态转移方程，计算两个字符串之间的编辑距离  
5      * @param a- 第一个字符串，b- 第二个字符串  
6      * @return int- 两者之间的编辑距离  
7     */  
8  
9     public static int getStrDistance(String a, String b) {  
10        if (a == null || b == null) return -1;  
11    }
```

然后，初始化状态转移表。我用 int 型的二维数组来表示这个状态转移表，并对 i 为 0 且 j 大于 0 的元素，以及 i 大于 0 且 j 为 0 的元素，赋予相应的初始值。

 复制代码

```
1 // 初始用于记录化状态转移的二维表  
2     int[][] d = new int[a.length() + 1][b.length() + 1];  
3  
4     // 如果 i 为 0，且 j 大于等于 0，那么 d[i, j] 为 j  
5     for (int j = 0; j <= b.length(); j++) {  
6         d[0][j] = j;  
7     }  
8  
9     // 如果 i 大于等于 0，且 j 为 0，那么 d[i, j] 为 i  
10    for (int i = 0; i <= a.length(); i++) {  
11        d[i][0] = i;  
12    }
```

我这里实现的时候，i 和 j 都是从 0 开始，所以我计算的 $d[i+1, j+1]$ ，而不是 $d[i, j]$ 。而 $d[i+1, j+1] = \min(d[i, j+1] + 1, d[i+1, j] + 1, d[i, j] + r(i, j))$ 。

复制代码

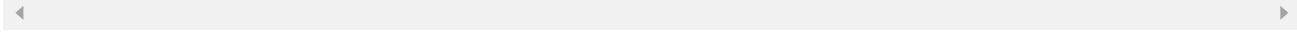
```
1 // 实现状态转移方程
2             // 请注意由于 Java 语言实现的关系，代码里的状态转移是从 d[i, j] 到 d[i+1, j+1]
3         for (int i = 0; i < a.length(); i++) {
4             for (int j = 0; j < b.length(); j++) {
5
6                 int r = 0;
7                 if (a.charAt(i) != b.charAt(j)) {
8                     r = 1;
9                 }
10
11                 int first_append = d[i][j + 1] + 1;
12                 int second_append = d[i + 1][j] + 1;
13                 int replace = d[i][j] + r;
14
15                 int min = Math.min(first_append, second_append);
16                 min = Math.min(min, replace);
17                 d[i + 1][j + 1] = min;
18
19             }
20         }
21
22     return d[a.length()][b.length()];
23
24 }
25
26 }
```



最后，我们用测试代码测试不同字符串之间的编辑距离。

复制代码

```
1 public static void main(String[] args) {
2     // TODO Auto-generated method stub
3     System.out.println(Lesson10_1.getStrDistance("mouse", "mouuse"));
4
5 }
```



从推导的表格和最终的代码可以看出，我们相互比较长度为 m 和 n 的两个字符串，一共需要求 $m \times n$ 个子问题，因此计算量是 $m \times n$ 这个数量级。和排列法的 m^n 相比，这已经降低太多太多了。

我们现在可以快速计算出编辑距离，所以就能使用这个距离作为衡量字符串之间相似度的一个标准，然后就可以进行查询推荐了。

到这里，使用动态规划来实现的编辑距离其实就讲完了。我把两个字符串比较的问题，分解成很多子串进行比较的子问题，然后使用状态转移方程来描述状态（也就是子问题）之间的关系，并根据问题的定义，保留最小的值作为当前的编辑距离，直到过程结束。

如果我们使用动态规划法来实现编辑距离的测算，那就能确保查询推荐的效率和效果。不过，基于编辑距离的算法也有局限性，它只适用于拉丁语系的相似度衡量，所以通常只用于英文或者拼音相关的查询。如果是在中文这种亚洲语系中，差一个汉字（或字符）语义就会差很远，所以并不适合使用基于编辑距离的算法。

实战演练：钱币组合的新问题

和排列组合等穷举的方法相比，动态规划法关注发现某种最优解。如果一个问题无需求出所有可能的解，而是要找到满足一定条件的最优解，那么你就可以思考一下，是否能使用动态规划来降低求解的工作量。

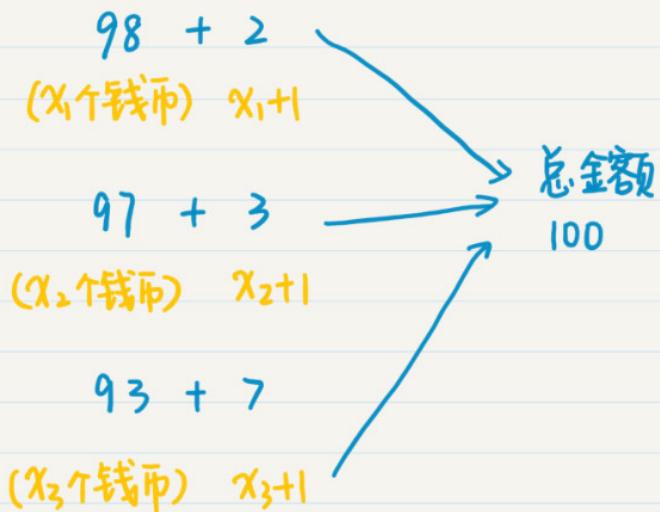
还记得之前我们提到的新版舍罕王奖赏的故事吗？国王需要支付一定数量的赏金，而宰相要列出所有可能的钱币组合，这使用了排列组合的思想。如果这个问题再变化为“给定总金额和可能的钱币面额，能否找出钱币数量最少的奖赏方式？”，那么我们是否就可以使用动态规划呢？

思路和之前是类似的。我们先把这个问题分解成很多更小金额的子问题，然后试图找出状态转移方程。如果增加一枚钱币 c ，那么当前钱币的总数量就是增加 c 之前的钱币总数再加上当前这枚。举个例子，假设这里我们有三种面额的钱币，2 元、3 元和 7 元。为了凑满 100 元的总金额，我们有三种选择。

第一种，总和 98 元的钱币，加上 1 枚 2 元的钱币。如果凑到 98 元的最少币数是 x_1 ，那么增加一枚 2 元后就是 $(x_1 + 1)$ 枚。

第二种，总和 97 元的钱币，加上 1 枚 3 元的钱币。如果凑到 97 元的最少币数是 x_2 ，那么增加一枚 3 元后就是 $(x_2 + 1)$ 枚。

第三种，总和 93 元的钱币，加上 1 枚 7 元的钱币。如果凑到 93 元的最少币数是 x_3 ，那么增加一枚 7 元后就是 $(x_3 + 1)$ 枚。



比较一下以上三种情况的钱币总数，取最小的那个就是总额为 100 元时，最少的钱币数。换句话说，由于奖赏的总金额是固定的，所以最后选择的那枚钱币的面额，将决定到上一步为止的金额，同时也决定了上一步为止钱币的最少数量。根据这个，我们可以得出如下状态转移方程：

拼课微信：176144365

$$c[i] = \arg \min_{j=1}^n (c[i - \text{value}(j)] + 1)$$

其中， $c[i]$ 表示总额为 i 的时候，所需要的最少钱币数，其中 $j=1,2,3,\dots,n$ ，表示 n 种面额的钱币， $\text{value}[j]$ 表示第 j 种钱币的面额。 $c[i - \text{value}(j)]$ 表示选择第 j 种钱币的时候，上一步为止最少的钱币数。需要注意的是， $i - \text{value}(j)$ 需要大于等于 0，而且 $c[0] = 0$ 。

我这里使用这个状态转移方程，做些推导，具体的数据你可以看下面这个表格。表格每一行表示奖赏的总额，前 3 列表示 3 种钱币的面额，最后一列记录最少的钱币数量。表中的 “/” 表示不可能，或者说无解。

总额\面额	2	3	7	最少钱币数c(x)
1	/	/	/	/
2	1	/	/	1
3	/	1	/	1
4	2	/	/	2
5	$c(3) + 1 = 2$	$c(2) + 1 = 2$	/	$\min(2, 2) = 2$
6	$c(4) + 1 = 3$	$c(3) + 1 = 2$	/	$\min(3, 2) = 2$
7	$c(5) + 1 = 3$	$c(4) + 1 = 3$	1	$\min(3, 3, 1) = 1$
8	$c(6) + 1 = 3$	$c(5) + 1 = 3$	/	$\min(3, 3) = 3$
9	$c(7) + 1 = 2$	$c(6) + 1 = 3$	$c(2) + 1 = 2$	$\min(2, 3, 3) = 2$
10	$c(8) + 1 = 4$	$c(7) + 1 = 2$	$c(3) + 1 = 2$	$\min(4, 2, 2) = 2$
11

这张状态转移表同样可以帮助你来理解状态转移方程的正确性。一旦状态转移方程确定了，要编写代码来实现就不难了。

小结

通过这两节的内容，我讲述了动态规划主要的思想和应用。如果仅仅看这两个案例，也许你觉得动态规划不难理解。不过，在实际应用中，你可能会产生这些疑问：什么时候该用动态规划？这个问题可以用动态规划解决啊，为什么我没想到？我这里就讲一些我个人的经验。

首先，如果一个问题有很多种可能，看上去需要使用排列或组合的思想，但是最终求的只是某种最优解（例如最小值、最大值、最短子串、最长子串等等），那么你不妨试试是否可以使用动态规划。

其次，状态转移方程是个关键。你可以用状态转移表来帮助自己理解整个过程。如果能找到准确的转移方程，那么离最终的代码实现就不远了。当然，最好的方式，还是结合工作中的项目，不断地实践，尝试，然后总结。

今日学习笔记

第10节 动态规划（下）

1. 状态转移方程是什么？

从上一个状态到下一个状态之间可能存在的一些变化，以及基于这些变化的最终决策结果。我们把这样的表达式称为状态转移方程。我上节最开始就说过，在所有动态规划的解法中，状态转移方程是关键，所以你一定要掌握它。

2. 基于编辑距离的算法有什么局限性？

基于编辑距离的算法也有局限性，它只适用拉丁语系的相似度衡量，所以通常只用于英文或者拼音相关的查询。如果在中文这种亚洲语系中，差一个汉字（或字符）语义就会差很远，所以并不适合使用基于编辑距离的算法。



黄申 · 程序员的数学基础课

思考题

对于总金额固定、找出最少钱币数的题目，用循环或者递归的方式该如何进行编码呢？

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 09 | 动态规划（上）：如何实现基于编辑距离的查询推荐？

下一篇 11 | 树的深度优先搜索（上）：如何才能高效率地查字典？

精选留言 (41)

写留言



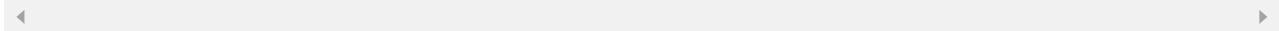
云开

2019-01-31

7

还是弄不明白编辑距离 为什么插入时是从空串开始 替换并不计算从空串到有字符的过程

作者回复: 你可以参考那张状态转移表，看看是从哪一格到哪一格，字符串是如何变换的，相邻格子的变换就三种方式，插入、删除和替换。替换可以将字符串中的某个字符替换成另一个字符



冰木

2019-01-26

4

老大，我可能没有得到要领，可以推到下，表格中，第一行，第二列吗？

作者回复: 是 $\min(3, 1, 2)$ 对吧 , 这个是mo和m的比较 , 3表示增加一个m再增加一个o , 再删掉一个o , 编辑距离是 $2+1=3$ 。1表示两个字符串都是m , 其中一个再增加一个o , 编辑距离是1。2表示一个m增加o , 一个从空集到m , 编辑距离是2。你可以顺着第9讲最后的表格来推导。



我心留

2019-01-05

3

```
public class Lesson10_2 {  
    /**  
     * 动态规划求最小钱币数  
     * @param c 用一维数组记录每一步的总金额      * @param value 用一维数组记录三种面  
     * 额的纸币    ...  
     */  
}
```

展开 ▼

作者回复: 代码的逻辑是对的



lianlian

2019-01-04

3

方法1 , 动态规划 , 最快。方法2递归有点慢 , 方法三递归 , 超级慢。在aim数值大于30的时候 , 三种写法 , 在我电脑速度快慢特别明显。用2元,3元,5元去找开100块 , 用递归方法 , 我的电脑要等到地老天荒O(∩_∩)O哈哈~

```
#include<iostream>  
#include<vector>...
```

展开 ▼

作者回复: 动手实验 , 比较不同的实现 , ↗



caohuan

2019-01-21

2

本篇所得 : 1.求解 最值可用动态规划 方法;
2.状态转移 可以把 大问题 分解为 小问题 , 再分解为 可以处理的问题 , 即 把 不可以处理的问题 分解为可以 处理的小问题 (也为子问题) ;
3.动态规划 适用于 下一个 状态与上一个状态有固定关系;

4.搜索引擎的 搜索词的查询推荐， 英文可用 编辑距离，中文 需要 转化 比如转为英文 ...
展开 ▾



mickey

2019-01-04

↳ 2

package Part01;

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;...
```

展开 ▾



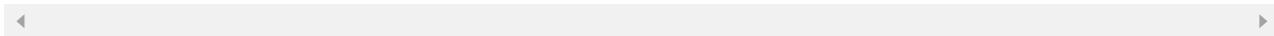
梅坊帝卿

2019-01-04

↳ 2

按照面值排序优先取最大的方法 不一定能取到解 除非有万能的面额1 比如 2 5 7 总数15

作者回复: 是的 可能无解 ↳



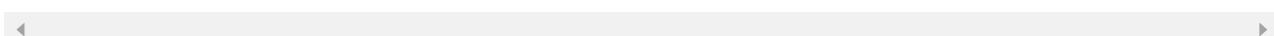
xiaobang

2019-01-15

↳ 1

min的三个参数应该分别是插入删除替换，或者插入插入替换吧

作者回复: 是的



Joe

2019-01-14

↳ 1

1.C++实现，对总金额100的最小纸币是15.

2.用递归法总金额为30就要算很久。

3.另外的数学办法可以用总金额依次对最大金额纸币求余数，直到为0.商相加为答案。如：
若 {1, 2, 3, 7} 为纸币金额，对于100，所需最小纸币数： $100/7=14$ 余2; $2/2 = 1$ 余0; 则纸
币数为 $14+1=15$

展开 ▾

作者回复: 答案正确 ✅



assert

2019-01-07

1

https://github.com/somenzz/geekbang/blob/master/mathOfProgramer/chapter10_实现了循环和递归，循环的方式快，递归的方式特别慢。

个人感觉递归是从后往前推导的，每一步的结果不论是否最优都保存在堆栈中，都占用了内存空间，算法上已经不属于动态规划。

...

展开 ▾

作者回复: 确实，动态规划使用循环更快



菩提

2019-01-06

1

思考题编码：

```
public static int least_count(int num) {  
    if (num < 0)  
        return -1;  
    int len = num;...
```

展开 ▾

作者回复: 思路正确 ✅，编码可以稍微再通用点，用循环访问不同的币值



lianlian

2019-01-04

1

老师早上好(^_^) / ，在第一题求r的时候，条件判断语句应该是a.charAt(i+1) != b.charAt(j+1)吧 😊 ?

作者回复: 应该是i和j没错，只是这里的i和j对应于d[i+1][j+1]。这里比较容易让人混淆，主要是因为d中的0下表表示的是空串，而不是字符串中的第一个字符。我之后将代码注释加一下



奔跑的蜗牛

2019-05-16



老大 如果是第一行第三列呢？是+m+o+u-u-o吗？

展开 ▼



jt

2019-05-14



c的来一个。

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

...

展开 ▼



You_can

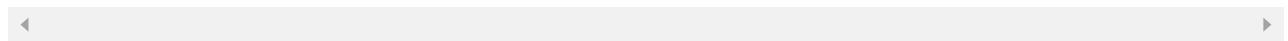
2019-04-07



```
/**
 * 交作业
 * @param moneyCnt 钱币总金额
 * @param coinValue 钱币面额
 * @return int...
```

展开 ▼

作者回复: 假期间坚持交作业，赞一个



叮当猫

2019-04-05



```
#include <vector>
#include <iostream>
using namespace std;
```

```
int dpSolve(int * a, int n, int aim){...
```

展开 ▼

作者回复: 代码写得很赞



枫林火山

2019-03-31



对钱币问题经过循环，递归和动态规划的编码实践。作为一个数学菜鸟我有以下几个感触
心得分享

1. 递归方法。

论性能是第一个就该出局的选择，但是他的分治思想是最容易理解和编码的。我觉得递归一个很大的作用是可以用快速得出的结果，帮助我们分析问题找出规律，来实现最优算...
展开 ▾

作者回复: 心得写得非常详细，赞一个。对于你最后的提问，我觉得暂时没有特别好的方法，除非我们可以接受某种近似解，并可以发现一种通过多个局部最优解能近似全局最优解的方法。



李凯

2019-03-27



黄老师您好 看不明白 您能讲一下 钱币组合表的8行和9行么 一行都没看懂

作者回复: 第8行表示一共要8元，8元钱可以由6元钱加上1枚2元硬币，那么就是 $c(6)+1$ ，而 $c(6)$ 查看第6行的最后一列，是2，也就是说6元钱最少需要2枚硬币，所以8元钱如果是6元钱的硬币外加一枚2元硬币组成的话，那么最少需要 $(2+1)$ 枚共3枚。不过，8元钱也可能由5元钱外加1枚3元硬币组成，所以我们还要看 $c(5)+1$ ，其他的以此类推，最后确定8元钱最少需要多少钱币。



梦倚栏杆

2019-03-23



有点想数学的归纳演义的感觉

展开 ▾



Lay Zhao

2019-03-22



交作业：支持任意数量面额

```
import Foundation
```

```
import XCTest...
```

展开▼

作者回复: 代码很详细 , 可以再贴一下运行结果吗 ?

