

有图有真相

一言不合就上源码

RSS订阅

原

算法-动态规划 Dynamic Programming--从菜鸟到老鸟

2017年07月15日 22:58:29

阅读数：7954

前言

最近在牛客网上做了几套公司的真题，发现有关动态规划（Dynamic Programming）算法的题目很多。相对于我来说，算法里面遇到的问题里面感觉最难的也就是动态规划（Dynamic Programming）算法了，于是花了好长时间，查找了相关的文献和资料准备彻底的理解动态规划（Dynamic Programming）算法。一是帮助自己总结知识点，二是也能够帮助他人更好的理解这个算法。后面的参考文献只是我看到的文献的一部分。

动态规划算法的核心

理解一个算法就要理解一个算法的核心，动态规划算法的核心是下面的一张照片和一个小故事。

Those who cannot remember the past
are condemned to repeat it.

-Dynamic Programming
<http://blog.csdn.net/u013309870>

```
1  A * "1+1+1+1+1+1+1 = ? " *
2
3  A : "上面等式的值是多少"
4  B : *计算* "8!"
5
6  A *在上面等式的左边写上 "1+" *
7  A : "此时等式的值为多少"
8  B : *quickly* "9!"
9  A : "你怎么这么快就知道答案了"
10 A : "只要在8的基础上加1就行了"
11 A : "所以你不用重新计算因为你记住了第一个等式的值为8!动态规划算法也可以说是 '记住求过的解来节省时间'"
```

由上面的图片和小故事可以知道动态规划算法的核心就是记住已经解决过的子问题的解。

动态规划算法的两种形式

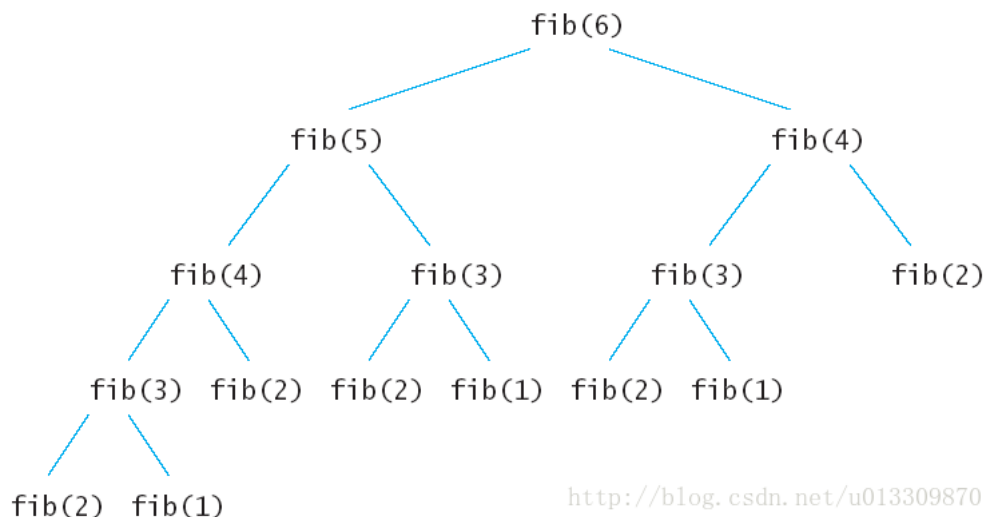
上面已经知道动态规划算法的核心是记住已经求过的解，记住求解的方式有两种：①自顶向下的备忘录法 ②自底向上。为了说明动态规划的这两种方法，举一个最简单的例子：求斐波拉契数列Fibonacci。先看一下这个问题：

```
1  Fibonacci (n) = 1;    n = 0
2
3  Fibonacci (n) = 1;    n = 1
4
5  Fibonacci (n) = Fibonacci(n-1) + Fibonacci(n-2)
```

以前学c语言的时候写过这个算法使用递归十分的简单。先使用递归版本来实现这个算法：

```
1 public int fib(int n)
2 {
3     if(n<=0)
4         return 0;
5     if(n==1)
6         return 1;
7     return fib( n-1)+fib(n-2);
8 }
9 //输入6
10 //输出: 8
```

先来分析一下递归算法的执行流程，假如输入6，那么执行的递归树如下：



<http://blog.csdn.net/u013309870>

上面的递归树中的每一个子节点都会执行一次，很多重复的节点被执行，fib(2)被重复执行了5次。由于调用每一个函数的时候都要保留上下文，所以空间上开销也不小。这么多的子节点被重复执行，如果在执行的时候把执行过的子节点保存起来，后面要用到的时候直接查表调用的话可以节约大量的时间。下面就看看动态规划的两种方法怎样来解决斐波拉契数列Fibonacci 数列问题。

①自顶向下的备忘录法

```
1 public static int Fibonacci(int n)
2 {
3     if(n<=0)
4         return n;
5     int []Memo=new int[n+1];
6     for(int i=0;i<=n;i++)
7         Memo[i]=-1;
8     return fib(n, Memo);
9 }
10 public static int fib(int n,int []Memo)
11 {
12
13     if(Memo[n]!=-1)
14         return Memo[n];
15     //如果已经求出了fib (n) 的值直接返回，否则将求出的值保存在Memo备忘录中。
16     if(n<=2)
17         Memo[n]=1;
18
19     else Memo[n]=fib( n-1,Memo)+fib(n-2,Memo);
20
21     return Memo[n];
22 }
```

备忘录法也是比较好理解的，创建了一个n+1大小的数组来保存求出的斐波拉契数列中的每一个值，在递归的时候如果发现前面fib (n) 的值计算出来了就不再计算，如果未计算出来，则计算出来后保存在Memo数组中，下次在调用fib (n) 的时候就不会重新递归了。比如上面的递归树中在计算fib (6) 的时候先计算fib (5) ，调用fib (5) 算出了fib (4) 后，fib (6) 再调用fib (4) 就不会在递归fib (4) 的子树了 因为fib (4) 的值已经保存在Memo[4]中

备忘录法还是利用了递归，上面算法不管怎样，计算fib（6）的时候最后还是要计算出fib（1），fib（2），fib（3）.....,那么何不先计算出fib（1），fib（2），fib（3）.....,呢？这也就是动态规划的核心，先计算子问题，再由子问题计算父问题。

```
1 public static int fib(int n)
2 {
3     if(n<=0)
4         return n;
5     int []Memo=new int[n+1];
6     Memo[0]=0;
7     Memo[1]=1;
8     for(int i=2;i<=n;i++)
9     {
10         Memo[i]=Memo[i-1]+Memo[i-2];
11     }
12     return Memo[n];
13 }
```

自底向上方法也是利用数组保存了先计算的值，为后面的调用服务。观察参与循环的只有i，i-1，i-2三项，因此该方法的空间可以进一步的压缩如下。

```
1 public static int fib(int n)
2 {
3     if(n<=1)
4         return n;
5
6     int Memo_i_2=0;
7     int Memo_i_1=1;
8     int Memo_i=1;
9     for(int i=2;i<=n;i++)
10     {
11         Memo_i=Memo_i_2+Memo_i_1;
12         Memo_i_2=Memo_i_1;
13         Memo_i_1=Memo_i;
14     }
15     return Memo_i;
16 }
```

一般来说由于备忘录方式的动态规划方法使用了递归，递归的时候会产生额外的开销，使用自底向上的动态规划方法要比备忘录方法好。你以为看懂了上面的例子就懂得了动态规划吗？那就too young too simple了。动态规划远远不止如此简单，下面先给出一个例子看看能否独立完成。然后再对动态规划的其他特性进行分析。

动态规划小试牛刀

例题：钢条切割

假定我们知道 Serling 公司出售一段长度为 i 英寸的钢条的价格为 $p_i(i=1, 2, \dots$ ，单位为美元)。钢条的长度均为整英寸。图 15-1 给出了一个价格表的样例。

长度 <i>i</i>	1	2	3	4	5	6	7	8	9	10
价格 <i>p_i</i>	1	5	8	9	10	17	17	20	24	30

图 15-1 钢条价格表样例。每段长度为 i 英寸的钢条为公司带来 p_i 美元的收益

钢条切割问题是这样的：给定一段长度为 n 英寸的钢条和一个价格表 $p_i(i=1, 2, \dots, n)$ ，求切割钢条方案，使得销售收益 r_n 最大。注意，如果长度为 n 英寸的钢条的价格 p_n 足够大，最优解可能就是完全不需要切割。

<http://blog.csdn.net/u013309870>

我们发现，将一段长度为 4 英寸的钢条切割为两段各长 2 英寸的钢条，将产生 $p_2 + p_2 = 5 + 5 = 10$ 的收益，为最优解。

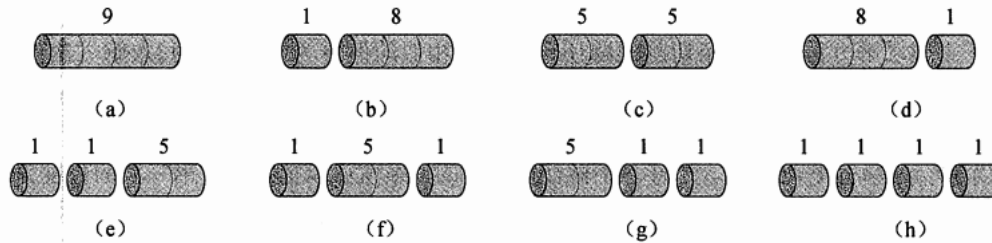


图 15-2 4 英寸钢条的 8 种切割方案。根据图 15-1 中的价格表，在每段钢条之上标记了它的价格。最优策略为方案(c)——将钢条切割为两段长度均为 2 英寸的钢条——总价值为 10。

更一般地，对于 $r_n (n \geq 1)$ ，我们可以用更短的钢条的最优切割收益来描述它：

$$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \dots, r_{n-1} + r_1) \quad (15.1)$$

第一个参数 p_n 对应不切割，直接出售长度为 n 英寸的钢条的方案。其他 $n-1$ 个参数对应另外 $n-1$ 种方案：对每个 $i=1, 2, \dots, n-1$ ，首先将钢条切割为长度为 i 和 $n-i$ 的两段，接着求解这两段的最优切割收益 r_i 和 r_{n-i} （每种方案的最优收益为两段的最优收益之和）。由于无法预知哪种方案会获得最优收益，我们必须考察所有可能的 i ，选取其中收益最大者。如果直接出售原钢条会获得最大收益，我们当然可以选择不做任何切割。

<http://blog.csdn.net/u013309870>



10

关于问题的最优解，并在所有可能的两段切割方案中选取组合收益最大者，构成原问题的最优解。问题称钢条切割问题满足最优子结构(optimal substructure)性质：问题的最优解由相关子问题的最优解组合而成，而这些子问题可以独立求解。



除了上述求解方法外，钢条切割问题还存在一种相似的但更为简单的递归求解方法：我们将钢条左边切割下长度为 i 的一段，只对右边剩下的长度为 $n-i$ 的一段继续进行切割(递归求解)，对剩下的一段则不再进行切割。即问题分解的方式为：将长度为 n 的钢条分解为左边开始一段，以及剩余部分继续分解的结果。这样，不做任何切割的方案就可以描述为：第一段的长度为 n ，收益为 p_n ；剩余部分长度为 0，对应的收益为 $r_0=0$ 。于是我们可以得到公式(15.1)的简化版本：

$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i})$$

<http://blog.csdn.net/u013309870>



微博

上面的内容来自于算法导论

关于题解就直接截图算法导论书上了这里就不展开讲。现在使用一下前面讲到三种方法来实现一下。

①递归版本

```
1 public static int cut(int []p,int n)
2 {
3     if(n==0)
4         return 0;
5     int q=Integer.MIN_VALUE;
6     for(int i=1;i<=n;i++)
7     {
8         q=Math.max(q, p[i-1]+cut(p, n-i));
9     }
10    return q;
11 }
```

递归很好理解，如果不懂可以看上面的讲解，递归的思路其实和回溯法是一样的，遍历所有解空间但这里和上面斐波拉契数列的不同之处在于，在每一层上都进行了一次最优解的选择， $q=\text{Math.max}(q, p[i-1]+\text{cut}(p, n-i))$ ；这个语句就是最优解选择，这里上一层的最优解与下一层的最优解相关。

②备忘录版本

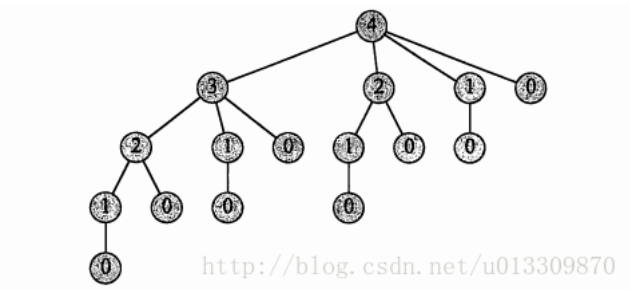
```
3         int []r=new int[p.length+1];
4         for(int i=0;i<=p.length;i++)
5             r[i]=-1;
6         return cut(p, p.length, r);
7     }
8     public static int cut(int []p,int n,int []r)
9     {
10         int q=-1;
11         if(r[n]>=0)
12             return r[n];
13         if(n==0)
14             q=0;
15         else {
16             for(int i=1;i<=n;i++)
17                 q=Math.max(q, cut(p, n-i,r)+p[i-1]);
18         }
19         r[n]=q;
20
21         return q;
22     }
```

有了上面求斐波拉契数列的基础，理解备忘录方法也就不难了。备忘录方法无非是在递归的时候记录下已经调用过的子函数的值。这道钢条切割问题的经典之处在于自底向上的动态规划问题的处理，理解了这个也就理解了动态规划的精髓。

③自底向上的动态规划

```
1 public static int buttom_up_cut(int []p)
2 {
3     int []r=new int[p.length+1];
4     for(int i=1;i<=p.length;i++)
5     {
6         int q=-1;
7         //①
8         for(int j=1;j<=i;j++)
9             q=Math.max(q, p[j-1]+r[i-j]);
10        r[i]=q;
11    }
12    return r[p.length];
13 }
```

自底向上的动态规划问题中最重要的是理解注释①处的循环，这里外面的循环是求r[1],r[2].....，里面的循环是求出r[1],r[2].....的最优解，也就是说r[i]中保存的是钢条长度为i时划分的最优解，这里面涉及到了最优子结构问题，也就是一个问题取最优解的时候，它的子问题也一定要取得最优解。下面是长度为4的钢条划分的结构图。我就偷懒截了个图。



动态规划原理

虽然已经用动态规划方法解决了上面两个问题，但是大家可能还跟我一样并不知道什么时候要用到动态规划。总结一下上面的斐波拉契数列和钢条切割问题，发现两个问题都涉及到了重叠子问题，和最优子结构。

①最优子结构

用动态规划求解最优化问题的第一步就是刻画最优解的结构，如果一个问题的解结构包含其子问题的最优解，就称此问题具有最优子结构性。因此，某个问题是否适合应用动态规划算法，它是否具有最优子结构性是一个很好的线索。使用动态规划算法时，用子问题的最优解

求该问题的最优解，因此必须求其子问题的最优解

②重叠子问题

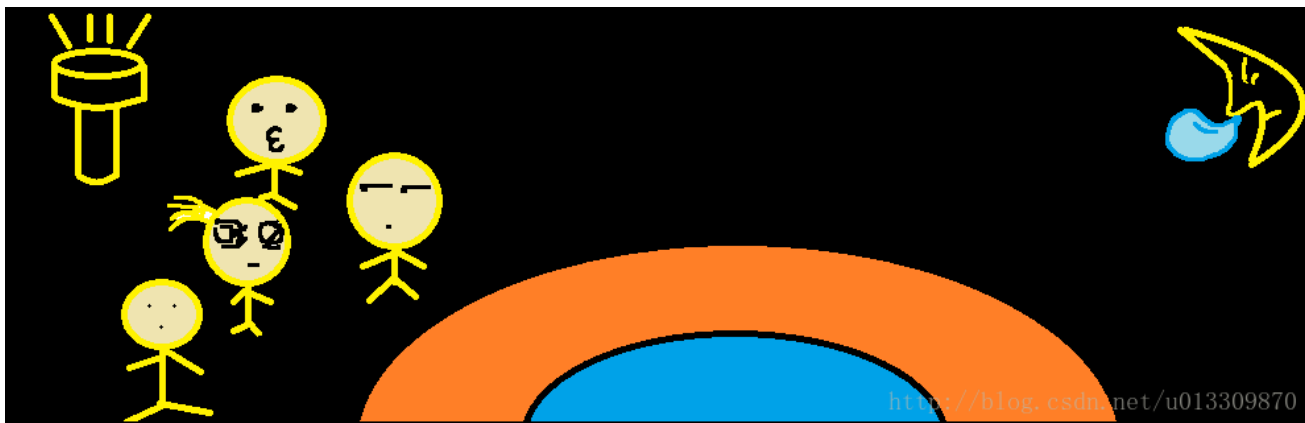
在斐波拉契数列和钢条切割结构图中，可以看到大量的重叠子问题，比如说在求fib（6）的时候，fib（2）被调用了5次，在求cut（4）的时候cut（0）被调用了4次。如果使用递归算法的时候会反复的求解相同的子问题，不停的调用函数，而不是生成新的子问题。如果递归算法反复求解相同的子问题，就称为具有重叠子问题（overlapping subproblems）性质。在动态规划算法中使用数组来保存子问题的解，这样子问题多次求解的时候可以直接查表不用调用函数递归。

动态规划的经典模型

线性模型

线性模型的是动态规划中最常用的模型，上文讲到的钢条切割问题就是经典的线性模型，这里的线性指的是状态的排布是呈线性的。【例题1】是一个经典的面试题，我们将它作为线性模型的敲门砖。

【例题1】在一个夜黑风高的晚上，有n（ $n \leq 50$ ）个小朋友在桥的这边，现在他们需要过桥，但是由于桥很窄，每次只允许不大于两人通过，他们只有一个手电筒，所以每次过桥的两个人需要把手电筒带回来，i号小朋友过桥的时间为T[i]，两个人过桥的总时间为二者中时间长者。问所有小朋友过桥的总时间最短是多少。



每次过桥的时候最多两个人，如果桥这边还有人，那么还得回来一个人（送手电筒），也就是说N个人过桥的次数为 $2*N-3$ （倒推，当桥这边只剩两个人时只需要一次，三个人的情况为来回一次后加上两个人的情况...）。有一个人需要来回跑，将手电筒送回来（也许不是同一个人，really？！）这个回来的时间是没办法省去的，并且回来的次数也是确定的，为N-2，如果是我，我会选择让跑的最快的人来干这件事情，但是我错了...如果总是跑得最快的人跑回来的话，那么他在每次别人过桥的时候一定得跟过去，于是就变成就是很简单的问题了，花费的总时间：

$$T = \min PTime * (N-2) + (totalSum - \min PTime)$$

来看一组数据 四个人过桥花费的时间分别为 1 2 5 10，按照上面的公式答案是19，但是实际答案应该是17。

具体步骤是这样的：

第一步：1和2过去，花费时间2，然后1回来（花费时间1）；

第二步：3和4过去，花费时间10，然后2回来（花费时间2）；

第三部：1和2过去，花费时间2，总耗时17。

所以之前的贪心想法是不对的。我们先将所有人按花费时间递增进行排序，假设前i个人过河花费的最少时间为opt[i]，那么考虑前i-1个人过河的情况，即河这边还有1个人，河那边有i-1个人，并且这时候手电筒肯定在对岸，所以 $opt[i] = opt[i-1] + a[1] + a[i]$ （让花费时间最少的人把手电筒送过来，然后和第i个人一起过河）如果河这边还有两个人，一个是第i号，另外一个无所谓，河那边有i-2个人，并且手电筒肯定在对岸，所以 $opt[i] = opt[i-2] + a[1] + a[i] + 2*a[2]$ （让花费时间最少的人把手电筒送过来，然后第i个人和另外一个人一起过河，由于花费时间最少的人在这边，所以下一次送手电筒过来的一定是花费次少的，送过来后花费最少的和花费次少的一起过河，解决问题）

所以 $opt[i] = \min\{opt[i-1] + a[1] + a[i], opt[i-2] + a[1] + a[i] + 2*a[2]\}$

区间模型

区间模型的状态表示一般为d[i][j]，表示区间[i, j]上的最优解，然后通过状态转移计算出[i+1, j]或者[i, j+1]上的最优解，逐步扩大区间的范围，最终求得[1, len]的最优解。

我们用 $d[i][j]$ 来表示 $A[i...j]$ 这个子串变成回文串所需要添加的最少的字符数，那么对于 $A[i] == A[j]$ 的情况，很明显有 $d[i][j] = d[i+1][j-1]$ （这里需要明确一点，当 $i+1 > j-1$ 时也是有意义的，它代表的是空串，空串也是一个回文串，所以这种情况下 $d[i+1][j-1] = 0$ ）；当 $A[i] != A[j]$ 时，我们将它变成更小的子问题求解，我们有两种决策：

1、在 $A[j]$ 后面添加一个字符 $A[i]$ ；

2、在 $A[i]$ 前面添加一个字符 $A[j]$ ；

根据两种决策列出状态转移方程为：

$d[i][j] = \min\{d[i+1][j], d[i][j-1]\} + 1$; (每次状态转移，区间长度增加1)

空间复杂度 $O(n^2)$ ，时间复杂度 $O(n^2)$ ，下文会提到将空间复杂度降为 $O(n)$ 的优化算法。

背包模型

背包问题是动态规划中一个最典型的问题之一。由于网上有非常详尽的背包讲解，这里只将常用部分抽出来。

【例题3】有 N 种物品（每种物品1件）和一个容量为 V 的背包。放入第 i 种物品耗费的空间是 C_i ，得到的价值是 W_i 。求解将哪些物品装入背包可使价值总和最大。 $f[i][v]$ 表示前 i 种物品恰好放入一个容量为 v 的背包可以获得的最大价值。决策为第 i 个物品在前 $i-1$ 个物品放置完毕后，是选择放还是不放，状态转移方程为：

$f[i][v] = \max\{f[i-1][v], f[i-1][v - C_i] + W_i\}$

时间复杂度 $O(VN)$ ，空间复杂度 $O(VN)$ （空间复杂度可利用滚动数组进行优化达到 $O(V)$ ）。

动态规划题集整理

1、最长单调子序列

[Constructing Roads In JG Kingdom](#)★★☆☆☆

[Stock Exchange](#) ★★☆☆☆

2、最大 M 子段和

[Max Sum](#) ★☆☆☆☆

[最长公共子串](#) ★★☆☆☆

3、线性模型

[Skiing](#) ★☆☆☆☆

总结

弄懂动态规划问题的基本原理和动态规划问题的几个常见的模型，对于解决大部分的问题已经足够了。希望能对大家有所帮助，转载请标明出处<http://write.blog.csdn.net/mdeditor#!postId=75193592>，创作实在不容易，这篇博客花了我将近一个星期的时间。

参考文献

1.算法导论

版权声明：本文为博主原创文章，转载请标明出处。 <https://blog.csdn.net/u013309870/article/details/75193592>

文章标签： [动态规划](#) [算法](#)

个人分类： [算法](#)

[查看更多>>](#)

想对作者说点什么？ [我来说一句](#)

博主，感谢您的分享 但是在过桥问题中的一个最优解是不是有瑕疵 $opt[i-2] + a[1] + a[i] + 2*a[2]$ 比如4个人过桥，时间序列为1，5，5，5 桥左剩余2人时，必然剩余5，5。每次都是第一个人折回送手电，剩余时间消耗为 $(1+5)*2=12$ 而按照这个最优解，剩余过桥时间消耗 $a[1] + a[i] + 2*a[2]=1+5+5*2=16$ 是不是我理解出了问题，非常期待您的解答

 IL_Programmer

2018-04-01 19:39:02 #2楼

博主，你的切钢条的第一个递归解法，当 $n>length$ 的时候数组p访问越界了，应该改一下 `public static int cut(int []p,int n) { if(n==0) return 0; int q=Integer.MIN_VALUE; for(int i=1;i<=n&&i<p.length;i++) { q=Math.max(q, p[i-1]+cut(p, n-i)); } return q; }`

 Elliotprl

2018-02-28 17:48:46 #1楼

[查看回复\(1\)](#)

你好，看了这篇博客很是受教，来来回回看来多次，但是本人基础比较薄弱，在看到动态规划——线性模型例题一（过桥问题）的时候对于状态转换方程有些不理解即： $opt[i] = \min\{opt[i-1] + a[1] + a[i], opt[i-2] + a[1] + a[i] + 2*a[2]\}$ 为什么： $opt[i-2] + a[1] + a[i] + 2*a[2]$ 是一个最优解，这里只考虑了岸上只留一个人（第i个人）和只剩两个人（第i个人和其他一个）这两种情况，会不会有其他的情况 比如考虑这边岸上有三个人的情况。 博主能否在有空之余给一个更加详细的说明或者该题最优结构的证明，链接或者思路都可以，感激不尽！

五大常用算法 ----DP 动态规划 (Dynamic Programming)

一、基本概念 动态规划过程是：每次决策依赖于当前状态，又随即引起状态的转移。一个决策序列就是在变化的状态中产生出来的，所以，这种多阶段最优决策解决问题的过程就称为动态规划。 二、基本思想...

 Caroline424

2016-07-24 22:35:21 阅读数：3679

动态规划 (dynamic programming) 初步入门

通过金矿模型介绍动态规划 点击下载01背包测试数据.rar 对于动态规划，每个刚接触的人都需要一段时间来理解，特别是第一次接触的时候总是想不通为什么这种方法可行， ...

 bit_zcy

2016-03-22 10:16:39 阅读数：4061

[NOI题库]4.5算法之动态规划9276:Connected Graph DP ----CSDN博客

[NOI题库]4.5算法之动态规划9276:Connected Graph DP 2017-11-02 16:35 4人阅读 评论(0) 收藏 举报 分类: DP(43) 作者同类文章X ...
2017-11-2

《面试--动态规划》 ---五种经典的算法问题 - CSDN博客

一 动态规划 动态规划问题是面试题中的热门话题,如果要求一个问题的最优解(通常是最大值或者最小值),而且该问题能够分解成若干个子问题,并且小问题之间也存在...

2018-6-5

动态规划(dynamic programming)原理

首先声明！鉴于之前有一段时间经常有朋友问我一些DP的问题，趁着现在有点时间，我在网上找了很多关于DP的资料，也希望大家可以通过这篇文章有一些收获！ 引入问题：现在我有1块，3块，5块的硬币，假设我...

 q623702748

2016-04-30 17:21:02 阅读数：2673

经典算法之一动态规划 - CSDN博客

动态规划是一种求解多阶段决策问题的最优化算法。一.多阶段决策过程事件的发展过程分为若干个相互联系的阶段。事件的发展总是初始状态开始,依次经过第一阶段、第二...

2018-5-20

经典算法题04-动态规划算法(背包问题) - CSDN博客

本来我是想解决公共最长子序列问题,但是在解题过程中需要先清楚动态规划算法,故先用动态规划算法中最经典的背包问题来说明。背包问题问题描述有N件物品和一个容量为...

2018-5-23

Dynamic Programming整理

动态规划相关问题整理

动态规划(Dynamic Programming)与贪心算法(Greedy Algorithms)

动态规划动态规划是用来解决优化问题的，即要作出一组选择以达到最优解。**动态规划**有两个标志：一是可以划分最优子问题，二是子问题有重叠性(可以自底向上存储这些子问题的解，把**算法**效率从指数时间降到多项式时间) ...

 wangxiaojun911 2009-10-18 13:46:00 阅读数：4973

算法之一动态规划 - CSDN博客

举报内容: **算法之一动态规划** 举报原因: 色情 政治 抄袭 广告 招聘 骂人 其他 原文地址: 原因补充: 最多只允许输入30个字加入...

2018-5-29


动态规划刷题(一) - CSDN博客

动态规划刷题(一) 阅读量:878 并查集(union-find)**算法**学习笔记 阅读量:220 ...举报内容: **动态规划**刷题(一) 举报原因: 色情 政治 抄袭 广告 招聘 骂人 其他...

2018-5-4

教你彻底学会动态规划——入门篇

动态规划相信大家都知道，**动态规划算法**也是新手在刚接触**算法**设计时很苦恼的问题，有时候觉得难以理解，但是真正理解之后，就会觉得**动态规划**其实并没有想象中那么难。网上也有很多关于讲解**动态规划**的文章，大多都是叙...

 baidu_28312631 2015-08-11 13:26:41 阅读数：144395

算法之动态规划 - CSDN博客

的状态中产生出来的,所以,这种多阶段最优化决策解决问题的过程就称为**动态规划**。...本题可转化为**动态规划算法**求解最长公共子序列问题,然后用总字符串长度... qq...

2018-2-17

【NOI题库 动态规划】 - CSDN博客

这个是部分NOI**题库**的**动态规划**先贴代码,解析挖个坑以后慢慢填。 1759:最长上升子序列#include <iostream> #include <cstdio> #include <queue> #include <string...

2018-5-31

动态规划求解最长公共子序列 (LCS)

看了《**算法导论**》中文第二版P208的**动态规划**求解LCS问题，觉得很赞，但总觉得算导写得有些晦涩，希望自己能写得简单易懂一些，纯当锻炼了，欢迎指导交流。 首先，子序列和子串是不一样的。子串是连续的，而子...

 zhouworld16 2013-11-15 20:23:47 阅读数：7200

LCS问题求解-动态规划

1.何为LCS问题：在求解LCS问题之前，我们需要先了解一下什么叫做最长公共子序列 最长公共子序列：用我们最容易通俗理解的话语来解释的话，最长公共子序列就是两个或者多个串中，最长的相同的子序列 ...

 ltyqljhwcm 2016-09-15 00:11:54 阅读数：3006

动态规划经典题目及解答(有代码),经典的不能再经典了

动态规划经典题目及解答(有代码) 1. 最长公共子序列 2. 计算矩阵连乘积 3. ...**动态规划**经典题目汇总 **动态规划**经典五题 **动态规划**经典例题之传纸条 **算法**:动态...

2018-5-9

动态规划题目整合 - CSDN博客

源博客链接:http://blog.csdn.net/cc_again/article/details/25866971 **动态规划**一直是ACM竞赛中的重点,同时又是难点,因为该**算法**时间效率高,代码量少,多元性强,...

2018-6-5

优化中的subgradient方法

哎，刚刚submit上paper比较心虚啊，无心学习，还是好好码码文字吧。 subgradient中文名叫次梯度，和梯度一样，完全可以多放梯度使用，至于为什么叫子梯度，是因为有一些凸函数是不可导的， ...

 lansatiankonoxxc 2015-06-06 11:03:15 阅读数：9092

《面试--动态规划》---五种经典的算法问题

一 **动态规划** **动态规划**问题是面试题中的热门话题，如果要求一个问题的最优解（通常是最大值或者最小值），而且该问题能够分解成若干个子问题，并且小问题之间也存在重叠的子问题，则考虑采用**动态规划**。使用动...

 tongxinzhazha 2017-08-19 11:02:49 阅读数：8654

Dynamic Programming

第一次听说**动态规划**(Dynamic programming)这个名词的人都会觉得它有点怪，其实它的本名应该是多阶段决策过程(multistage decision process)。根据Richar...

 jtop0 2010-05-03 22:46:00 阅读数：2475


LeetCode随笔之Dynamic programming

一. **Dynamic Programming** 1. 牛牛在农场饲养了n只奶牛,依次编号为0到n-1, 牛牛的好朋友羊羊帮牛牛照看着农场.有一天羊羊看到农场中逃走了k只奶牛,但是他只会告诉牛牛逃走的k...

 lcb_coconut 2017-08-20 17:54:33 阅读数：150


动态规划经典例题

关于**动态规划**的介绍很多，本文希望通过重复几个最经典的例题来理解**动态规划**。问题1 求一个字符串中的最长的回文子串 回文是指正着读和倒着读，结果一样，比如abcba或abba。分析：令状态方...

 ktigerhero3 2015-11-02 20:02:30 阅读数：3348

动态规划 (Dynamic Programming, DP) 简介

动态规划 (Dynamic programming , DP) 是一种在数学、计算机科学和经济学中使用的，通过把原问题分解为相对简单的子问题的方式求解复杂问题的方法。**动态规划**常常适用于有重叠子问题和最优子结...

 carson2005 2014-03-25 20:07:04 阅读数：2910

【动态规划】Dynamic Programming

动态规划 一、**动态规划** **动态规划** (Dynamic Programming) 是一种设计的技巧，是解决多阶段决策过程最优化问题的通用方法。基本思想：将待求解问题分解成若干个子问题，先求解子问题，然后从...

 lisong694767315 2014-11-27 16:41:49 阅读数：7183

递归、分治策略、动态规划以及贪心算法之间的关系

最近集中研究计算智能，其中涉及到递归和**动态规划**，**动态规划**实现中又用到了递归，忽然发现这两个概念的差别分得不太清楚。索性把递归、分治策略、**动态规划**、贪婪选择之间的联系与区别都一并搞清楚吧。...

 tyhj_sf 2017-01-01 23:01:59 阅读数：3392

dynamic programming by Bellman

2009年07月17日 3.16MB



算法导论：第15章 动态规划_1_2钢条切割_动态规划的两种解法

/* 钢条切割: **动态规划**与分治的相同点：组合子问题求解原问题 不同点：分治的子问题不重叠，做了重复工作，**动态规划**保存解到表中 **动态规划**的特点：1最优子结构：问题的...

 qingyuanluofeng 2016-03-10 14:29:43 阅读数：389

动态规划算法经典案例

动态规划的关键点：1、最优化原理，也就是最有子结构性质。这指的是一个最优化策略具有这样的性质，无论过去状态和决策如何，对前面的决策所形成的状态而言，余下的决策必须构成最优策略，简单来说就是一个最优化...

 uestclr 2016-02-29 20:14:52 阅读数：33319

几个经典的动态规划算法

个人资料



HankingHu

关注

原创	粉丝	喜欢	评论
168	60	73	46

等级： 访问：15万+
积分：3248 排名：1万+
勋章：

最新文章

python 实现DoS攻击
python爬虫豆瓣推理书籍及链接
人脸检测python
丰富的人脸相关数据库链接
github博客加访问量标签

个人分类

spring	1篇
设计模式	5篇
java	42篇
android	29篇
算法	53篇

展开

归档

2018年5月	2篇
2018年4月	6篇
2018年3月	1篇
2018年2月	1篇
2018年1月	3篇

展开

热门文章

Java中的File文件类详解
阅读量：18368
Java进阶--深入理解ArrayList实现原理
阅读量：9909
Java堆结构PriorityQueue完全解析
阅读量：8070
算法-动态规划 Dynamic Programming--从菜鸟到老鸟
阅读量：7871
笔试面试算法经典--全排列算法-递归&字典序实现（Java）
阅读量：5369

最新评论

加入CSDN，享受更精准的内容推荐，与500万程序员共同成长！

登录

注册



lewkyl Liu : ③在Integer.MIN_VALUE基础上加1输出的结果是个正值。 最小值减一才是正值

Java进阶--深入理解Array...

qq_39736758 : [reply]qq_39736758[/reply] java设置了最小容量 (minCapacit...

Java进阶--深入理解Array...

qq_39736758 : 楼主总结的非常好，但是我有一个疑问 [code=java] //3-----...

Java进阶--深入理解Array...

qq_39736758 : public ArrayList() { this.elementData = D...

Java堆结构PriorityQu...

Mario_Hu : poll的图有问题吧。8应该和2换，它每次都是与最小的子结点交换的