动态规划

2019年1月29日

1

递推

兔子们尽情地繁殖着
不小心忘记了环境容纳量的存在

“

斐波那契数列

[redacted]

问：一般而言，不缺草吃的小兔子在出生两个月后就有繁殖能力，且一对兔子每个月能生出一对小兔子来。这里，我们假设兔子还是长生不老的。你一旦得到了一对小兔子，那么在一年之后你最多可以得到多少只兔子？

斐波那契数列

[redacted]

问：一般而言，不缺草吃的小兔子在出生两个月后就有繁殖能力，且一对兔子每个月能生出一对小兔子来。这里，我们假设兔子还是长生不老的。你一旦得到了一对小兔子，那么在一年之后你最多可以得到多少只兔子？

解：设 $F(n)$ 为第 n 个月兔子的总对数。

我们有：$F(n)$ =上一个月活下来的兔子对数 + 新出生的小兔子数，

即 $F(n-1)+F(n-2)$，初始状态为 $F(0)=F(1)=1$。

[redacted]

问：求用 1×2 的骨牌铺满大小为 $2 \times n$ 的方格的方案总数。

问：求用 1×2 的骨牌铺满大小为 $2 \times n$ 的方格的方案总数。

解：设 $F(n)$ 表示对于 n 的铺骨牌方案数。

首先，可以知道 $F(0) = 1.$

$F(1)$ 的状态只能通过竖着摆放的一个骨牌得到，因此 $F(1) = 1.$

考虑最右面的骨牌，可能是两个骨牌横着摆放，也可能最右面的骨牌单独竖着摆放。因此，有 $F(n) = F(n-1) + F(n-2).$

[redacted]

平面分割问题

[redacted]

问：n 条直线最多能把平面分成多少区域？

平面分割问题

[redacted]

问：n 条直线最多能把平面分成多少区域？

解：设 $F(n)$ 为 n 条直线把平面分割成的区域个数，分析问题中每增加一条直线时，平面增加了多少区域。画图可知，每增加一条直线，新增加的平面数为新增加的直线被分成的段数。

递推式：$F(n) = F(n-1) + n = {}^{n\times(n+1)}/_{2} + 1.$

问：在二维平面上，从原点 $(0,0)$ 出发，每次只能向下或向右走一个单位，走到 (x,y) 的路径数量。

问：在二维平面上，从原点 $(0,0)$ 出发，每次只能向下或向右走一个单位，走到 (x,y) 的路径数量。

解：

$$\begin{cases} f[1][i] = f[i][1] = 1 & \\ f[i][j] = f[i-1][j] + f[i][j-1], & i,j > 1 \end{cases}$$

组合数？

问：在二维平面上，从原点 $(0,0)$ 出发，每次只能向下或向右走一个单位，走到 (n,n) 的路径数量。

解：相当于有 n 次向下的机会和 n 次向右走的机会，方案数为

$$\binom{2n}{n}$$

[redacted]

卡特兰数

[redacted]

问：依次让 $1, 2, 3, \ldots, n$ 进栈，可能的出栈序列共有多少种？

卡特兰数

[redacted]

问：依次让 $1, 2, 3, \ldots, n$ 进栈，可能的出栈序列共有多少种？

解：设 $C(n)$ 表示长度为 n 的出栈序列数量。如果规定最后一个出栈的元素必须是 x，则方案数是 $C(x-1) \times C(n-x)$.

$$C(n) = \sum_{i=1}^{n} C(i-1) \times C(n-i) = \sum_{i=0}^{n-1} C(i) \times C(n-i-1)$$

[redacted]

卡特兰数

[redacted]

$$\begin{cases} C(0) = 1, \\ C(n) = \sum_{i=0}^{n-1} C(i) \times C(n-i-1), \qquad n \geq 1 \end{cases}$$

递推式：

$$C(n) = \frac{2 \times (2n-1)}{n+1} \times C(n-1)$$

卡特兰数

[redacted]

问：n 对括号能组成多少种括号序列？

问：n 个结点能构成多少种不同的二叉树？

问：有多少种用 n 个长方形填充大小为 n 的阶梯的方案数？

问：圆周上给定 $2n$ 个点，有多少连接 n 条不相交线段的方式？

卡特兰数

[redacted]

问：在二维平面上，从原点 $(0,0)$ 出发，每次只能向下或向右走一个单位，且不能走到直线 $y=x$ 以上，问走到 (x,y) 的最短路径数量。

解：答案为从 $(0,0)$ 到 (n,n) 的路径条数减去经过了对角线的路径条数。可以发现，穿过了对角线的路径一定经过了直线 $y=x+1$.
将它们第一次经过直线 $y=x+1$ 之前的路径沿 $y=x+1$ 对称，发现其与从 $(-1,1)$ 到 (n,n) 的路径一一对应。答案即为

$$\binom{2n}{n}-\binom{2n}{n-1}=\frac{(2n)!}{(n+1)!\,n!}=C(n)$$

2

动态规划入门

正经的求最优化问题的方法！

“

INTRO

[redacted]

这篇课件用于**DP入门**。

前置技能：小学数学知识

约定几个英文缩写：

e.g.	例如	动物都是生物，e.g. 猫是生物。
etc.	等等	动物中有猫，狗，etc.
P.S.	备注	P.S. 这篇课件之后的内容很多是rxz做的。

硬币问题 [redacted]

您有无限多的硬币，硬币的面值为 $1,5,10,50,100,500$.

给定一个数额 w，问您最少用多少枚硬币可以凑出 w.

硬币问题

[redacted]

依据生活经验，我们可以采用这种策略：

先尽量用500的，然后尽量用100的……以此类推。

e.g. 666 = 1*500+1*100+1*50+1*10+1*5+1*1,共用10枚硬币。

这就是贪心了。

我们每次使用一个硬币，总能最大程度地解决问题（把剩下要凑的数额变小）。可是，贪心是一种**只考虑眼前情况**的策略。尽管这一套硬币面值可以采用贪心策略，但是迟早要栽跟头的。

硬币问题

[redacted]

我们考虑一组新的硬币面值：1,5,11.

于是有了一个反例：如果我们要凑出15，贪心策略是：

15 = 11+4*1，共用 5 枚硬币。

而最佳策略是：

15 = 3*5，共用3枚硬币。

硬币问题

[redacted]

贪心策略自此陷入困境：鼠目寸光。

在 w=15 时，贪心策略选择了面值 11 的硬币（因为这样可以尽可能降低要凑的数额）。

在选择了面值为 11 的硬币之后，我们只好面对 w=4 的处境。

硬币问题

[redacted]

我们重新分析刚刚的情况：

w=15时，我们取了11，接下来面对w=4的情况。

w=15时，如果我们取5，接下来就面对w=10的情况。

我们记“凑出n需要用到的最少硬币数量”为 $f(n)$.

那么，如果我们取了 11，则：

$\text{cost} = f(4) + 1 = 4 + 1 = 5$.

解释：我们用了一枚面值为 11 的硬币，所以加一；

接下来面对的是 w=4 的情况。$f(4)$ 我告诉你等于4.

相应地，如果我们选择取 5，则：

$\text{cost} = f(10) + 1 = 2 + 1 = 3$.

硬币问题

[redacted]

那么，w=15时，我们选哪枚硬币呢？

cost最低的那一个！

11：　$\text{cost} = f(4) + 1 = 4 + 1 = 5.$

5:　$\text{cost} = f(10) + 1 = 2 + 1 = 3.$

1:　$\text{cost} = f(14) + 1 = 4 + 1 = 5.$

选择5，$f(15) = 3$,即为答案！

[redacted]

硬币问题

[redacted]

我们注意到了一个很棒的性质：

$f(n)$ 只与 $f(n-1), f(n-5), f(n-11)$ 相关。

更确切地说：

$$f(n) = \min\{f(n-1), f(n-5), f(n-11)\} + 1$$

硬币问题

[redacted]

```c
int f[105],i,n,cost;
scanf("%d",&n);

f[0]=0;

for(i=1;i<=n;i++)
{
    cost=INF;
    if(i-1>=0) cost=min(cost,f[i-1]+1);
    if(i-5>=0) cost=min(cost,f[i-5]+1);
    if(i-11>=0) cost=min(cost,f[i-11]+1);
    f[i]=cost;
    printf("f[%d]=%d\n",i,f[i]);
}
```

C:\Users\Administrator\Desktop\讲课\code\coin.exe

```
15
f[1]=1
f[2]=2
f[3]=3
f[4]=4
f[5]=1
f[6]=2
f[7]=3
f[8]=4
f[9]=5
f[10]=2
f[11]=1
f[12]=2
f[13]=3
f[14]=4
f[15]=3
3

--------------------------------
```

硬币问题

[redacted]

这个做法和贪心的区别是：

这个算法对给定的w，会算出取1、5、11的代价，从而确定最终答案。

而贪心直接选择可选的最大硬币，一条路走到黑。

硬币问题

[redacted]

这个算法的时间复杂度显然是$O(n)$.为什么比暴力要快呢？

我们暴力枚举了“使用的硬币”，然而这属于冗余信息。

我们要的是答案，根本**不关心这个答案是怎么凑出来的。**

要求出$f(15)$，只需要知道$f(14), f(10), f(4)$的**值。**

其他信息不需要。

硬币问题

[redacted]

可见，我们的做法比暴力快，是因为我们**舍弃了冗余信息。**

我们只记录了对解决问题有帮助的信息—$f(n)$.

我们能这样干，取决于问题的性质：

求出$f(n)$，只需要知道几个**更小的**$f(c)$.

我们将求解$f(c)$称作求解$f(n)$的“子问题”。

这就是动态规划（DP, dynamic programming）.

将一个问题拆成几个子问题，分别求解这些子问题，即可推断出大问题的解。

[redacted]

一旦$f(n)$确定，“我们如何凑出$f(n)$”就再也用不着了。

要求出$f(15)$，只需要知道$f(14), f(10), f(4)$的值，而$f(14), f(10), f(4)$是如何算出来的，对之后的问题没有影响。

“未来与过去无关”，这就是**无后效性。**

P.S. 其严格定义：如果给定某一阶段的状态，则在这一阶段以后过程的发展不受这阶段以前各段状态的影响。

回顾我们对$f(n)$的定义：

我们记“凑出n需要用到的最少硬币数量”为$f(n)$.

$f(n)$的定义就已经蕴含了“最优”。

利用w=14,10,4的最优解，我们即可算出w=15的最优解。

大问题的最优解可以由小问题的最优解推出，这个性质叫做“最优子结构性质”。

什么情况下我们能使用DP呢？

我们能将大问题拆成几个小问题，且满足

- 无后效性
- 最优子结构性质

动态规划

[redacted]

给出一个数字三角形，从其中一个位置只能走到其正下方和右下方两个位置。现在你在三角形的左上角，请找出一条走向最后一行的路径使得这条路径经过的数字之和最大。输出这条路径经过数字的和。

```
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
```

动态规划

[redacted]

设 $f(i,j)$ 表示从左上角走到 (i,j) 经过的数字之和的最大值。由于 (i,j) 可以由 $(i-1,j)$、$(i-1,j-1)$ 两个位置到达，可以列出来式子：

$$f(i,j)=\max\{f(i-1,j),f(i-1,j-1)\}+a(i,j).$$

用一些阶段的状态求得另外一个阶段的状态的过程叫做状态转移，其转移式称为**状态转移方程。**

```
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
```

```
7
10 15
18 16 15
20 25 20 19
24 30 27 26 24
```

动态规划

[redacted]

```
for (int i = 1; i <= n; ++i)
    for (int j = 1; j <= i; ++j)
        f[i][j] = max(f[i-1][j], f[i-1][j-1])
                  + a[i][j];
for (int i = 1; i <= n; ++i)
    if (f[n][i] > ans)
        ans = f[n][i];
cout << ans << endl;
```

状态转移

```
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

7
10 15
18 16 15
20 25 20 19
24 30 27 26 24
```

[redacted]

最长上升子序列（LIS）问题：

给定长度为n的序列a，从a中抽取出一个子序列，这个子序列需要单调递增。问最长的上升子序列（LIS）的长度。

e.g. **1**,5,**3**,**4**,**6**,9,**7**,**8**：LIS长度为6。

[redacted]

我们记$f(x)$为以a_x结尾的LIS长度，那么答案就是$\max\{f(x)\}$.

那么，大问题$f(x)$如何拆成小问题呢？考虑比x小的每一个p：

如果$a_x > a_p$，那么$f(x)$可以取$f(p)+1$.

解释：我们把a_x接在a_p的后面，肯定能构造一个以a_x结尾的上升子序列，长度比以a_p结尾的LIS大1.

[redacted]

那么，我们可以写出状态转移方程了：

$$f(x)=\max_{p<x\,,\,a_p<a_x}\{f(p)\}+1$$

两层for循环，复杂度$O(n^2)$.

[redacted]

最长上升子序列

[redacted]

```
for (int i = 1; i <= n; ++i)
{
    f[i] = 1;
    for (int j = 1; j < i; ++j)
        if (a[j] < a[i])
            f[i] = max(f[i], f[j] + 1);
}
for (int i = 1; i <= n; ++i)
    if (f[i] > ans)
        ans = f[i];
```

[redacted]

如果要输出一个最长上升子序列呢？

只需要在更新时顺便记录一下每一个$f(x)$是从哪里来的。

最长上升子序列

[redacted]

```
if (a[j] < a[i] && f[j] + 1 > f[i])
    f[i] = f[j] + 1, from[i] = j;
... // 中略
if (f[i] > ans)
    ans = f[i], t = i;
stack<int> s;
while (t)
    s.push(t), t = from[t];
while (!s.empty())
    cout << s.top() << endl, s.pop();
```

最长上升子序列

[redacted]

N位同学站成一排，音乐老师要请其中的$(N-K)$位同学出列，使得剩下的K位同学排成合唱队形。

合唱队形是指这样的一种队形：设K位同学从左到右依次编号为$1,2,\ldots,K$，他们的身高分别为$T_1,T_2,\ldots,T_K$， 则他们的身高满足$T_1 < \cdots < T_i > T_{i+1} > \cdots > T_K(1\le i\le K)$。

你的任务是，已知所有 N 位同学的身高，计算最少需要几位同学出列，可以使得剩下的同学排成合唱队形。

P.S. 题目来源：NOIP2004 合唱队形

[redacted]

此问题可以优化到$O(n\log n)$.

这里安利一份阅读材料：《动态规划初步·各种子序列问题》

此文对初学者很有帮助。作者是Flower_pks.

https://pks-loving.blog.luogu.org/junior-dynamic-programming-dong-tai-gui-hua-chu-bu-ge-zhong-zi-xu-lie

最长公共子序列

[redacted]

在两个字符串中，有些字符会一样，可以形成的子序列也有可能相等，因此，长度最长的相等子序列便是两者间的最长公共字序列，其长度可以使用动态规划来求。

C[i][j]表示第一个序列到第i位，第二个序列到第j位可以形成的最长公共子序列的长度。

$$C[i,j]=\begin{cases}0 & \text{若 } i=0\text{或}j=0\\ C[i-1,j-1]+1 & \text{若 } i,j>0,x_i=y_j\\ \max\{C[i,j-1],C[i-1,j]\} & \text{若 } i,j>0,x_i\neq y_j\end{cases}$$

永恒の灵魂最近得到了面积为$n \times m$的一大块土地，他想在这块土地上建造一所房子，这个房子必须是正方形的。
但是，这块土地并非十全十美，上面有很多不平坦的地方（也可以叫瑕疵）。这些瑕疵十分恶心，以至于根本不能在上面盖一砖一瓦。
他希望找到一块最大的正方形无瑕疵土地来盖房子。

输入文件第一行为两个整数n, m（$1 \leq n, m \leq 1000$），接下来n行，每行m个数字，用空格隔开。0 表示该块土地有瑕疵，1 表示该块土地完好。
输出一个整数，最大正方形的边长。

解题步骤：

首先明确状态数组的含义：

f[i][j] = k 表示的含义为______________________________。

之后列出转移方程：

f[i][j] = min(__________, __________, __________) + 1.

f[i][j] 中的最大值即为最终答案。

3

背包问题

最经典的动态规划模型！

“

0-1背包问题

[redacted]

辰辰是个天资聪颖的孩子，他的梦想是成为世界上最伟大的医师。为此，他想拜附近最有威望的医师为师。医师为了判断他的资质，给他出了一个难题。医师把他带到一个到处都是草药的山洞里对他说：“孩子，这个山洞里有一些不同的草药，采每一株都需要一些时间，每一株也有它自身的价值。我会给你一段时间，在这段时间里，你可以采到一些草药。如果你是一个聪明的孩子，你应该可以让采到的草药的总价值最大。”

P.S. 题目来源：NOIP2004普及组 采药

[redacted]

问题简化：

有 N 件物品和一个大小为 M 的背包，每件物品有固定的体积 v 和价值 w。从这些物品中选出若干件放入背包，使得选出的物品价值总和最大。求这一最大价值。

0-1背包问题

[redacted]

状态：设 f[i][j] 表示在前 i 件物品中选出若干件放入背包，占用空间为 j 时所能获得的最大价值。

根据第 i 件物品是否放入背包的决策，可列出**动态转移方程**：

f[i][j] = max(f[i-1][j], f[i-1][j-v[i]] + w[i]).

（不放）　　　　　（放）

由于每件物品只有不选（0）或选（1）两种状态，因此称为0-1背包问题。

0-1背包问题

[redacted]

```
for (int i = 1; i <= n; ++i)
    for (int j = 1; j <= m; ++j)
    {
        f[i][j] = f[i-1][j];
        if (j >= v[i])
            f[i][j] = max(f[i][j], f[i-1][j-v[i]] + w[i]);
    }
```

[redacted]

有 N 种物品和一个大小为 M 的背包，每种物品都有**无限件**，有固定的体积 v 和价值 w。从这些物品中选出若干件放入背包，使得选出的物品价值总和最大。求这一最大价值。

完全背包问题

[redacted]

状态：与 0-1 背包相同，设 f[i][j] 表示在前 i 件物品中选出若干件放入背包，占用空间为 j 时所能获得的最大价值。

根据第 i 件物品是否放入背包的决策，可列出**动态转移方程**：

```
f[i][j] = max(f[i-1][j], f[_____][_____] + w[i]).
```

背包九讲：

https://wenku.baidu.com/view/519124da5022aaea998f0f22.html?sxts=1548741647895