动态规划入门

2019年1月25日

黄哲威 hzwer

北京大学16级计算机科学

自我介绍

- 北京大学16级计算机科学方向
- 计算概论 A，数据结构与算法 A 助教
- NOI银牌，CTSC金牌，ACM区域赛金牌(Rank 4)
- 旷视科技(Face++) Research Intern 计算机视觉与强化学习
- hzwer.com

第一节目标

- 掌握动态规划的基本概念
- 掌握动态规划的两种实现

- 简单的划分，棋盘动态规划的实现

基本概念

- 动态规划是运筹学中用于求解决策过程中的最优化数学方法。

- 动态规划过程是：每次决策依赖于当前状态，决策引发状态的转移。

- 动态规划经常常使用于解决最优化问题，这些问题多表现为多阶段决策。

基本概念

- 状态:每个阶段所面临的条件，同一阶段可能会有不同的状态
- 决策 (转移):对于一个阶段的某个状态，从该状态演变到下一阶段的某个状态的选择
- 边界:决策过程中的初始情况
- 策略:由每个阶段所做的所有决策组成的序列称为策略。所有可行策略中
- 使得目标达到最佳情况的策略称为最优策略

基本概念

- 当你企图使用计算机解决一个问题时，其实就是在思考如何将这个问题表达成状态（存储哪些数据）以及如何在状态中转移（怎样根据一些数据计算出另一些数据）。

- 所以所谓的空间复杂度就是为了支持你的计算所必需存储的状态最多有多少，所谓时间复杂度就是从初始状态到达最终状态中间需要多少步。

适用的情况

- 最优子结构：假设问题的最优解所包括的子问题的解也是最优的，就称该问题具有最优子结构，即满足最优化原理。

- 无后效性：即某阶段状态一旦确定。就不受这个状态以后决策的影响。也就是说，某状态以后的过程不会影响曾经的状态。仅仅与当前状态有关。

举个例子

- 假设你要走一个网格状迷宫，迷宫的某些地方是墙，给定了起点和终点，只能往下或往右走，问最少需要多少步?

- 这个问题的解是显然的。

- 我们用动态规划的思路来研究一下。

举个例子

- 假设你要走一个网格状迷宫，迷宫的某些地方是墙，给定了起点和终点，只能往下或往右走，问最少需要多少步?

- 首先考虑状态的表示。假设你和朋友在一起研究这个迷宫游戏，你要告诉他，你走这个迷宫到了某一个“状态”，那么要传达哪些信息呢?

- F(x, y)=k，(x,y)表示坐标，k表示一个步数。那么你的朋友可以根据这些信息，完全得知你的游戏状态。

举个例子

- 假设你要走一个网格状迷宫，迷宫的某些地方是墙，给定了起点和终点，只能往下或往右走，问最少需要多少步?

- 如果这是个魔塔游戏，可能你需要告诉他更多的信息，比如人物属性，获得消耗的道具，剩余的怪物等等。

- 当然他不一定要知道所有的状态，但是需要知道一些必要的状态才能把这个游戏继续玩下去。

- 我们考察一下这个设计是否满足动态规划适用情况。

举个例子

- 假设你要走一个网格状迷宫，迷宫的某些地方是墙，给定了起点和终点，只能往下或往右走，问最少需要多少步?

- 无后效性。可以理解成，状态之间满足一种拓扑序关系，之后走迷宫的进程，不会影响到之前的某个状态，或者说之前的状态不依赖于之后的决策。

- 最优子结构。F(x,y)=min(F(x-1,y),F(x,y-1) + 1)

时间复杂度

- 往往可以通过增加状态的维数，记录更多的关键信息来满足无后效性与最优子结构，但维数的增加会导致重叠子问题减少而影响时间效率
- 复杂度 = 状态数 × 决策数目 × 转移代价
- 复杂度 = 实际状态数 + 总转移费用

序列型 dp

序列型 dp

- 本类的状态是基础的基础, 大部分的动态规划都要用到它, 成为一个维

- 一般来说, 有两种编号的状态
  1 状态[i]表示前 i 个元素决策组成的一个状态
  2 状态[i]表示用到了第i个元素，和其他在1到i−1间的元素，决策组成的一个状态，例如floyd

- 同样有状态、转移方程、无后效性等特点但无最优性决策过程的递推往往也纳入动态规划的研究范围，如计算方案数

斐波那契数列

- 计算斐波那契数列的第 n 项
- f(i) = f(i - 1) + f(i - 2)
- 可以用for循环或者递归的方式实现
- 一般实现可以用记忆化搜索。记忆化搜索是一种非常常见的动态规划实现方式，好处是比较自然，可以方便处理无效状态。

最长上升子序列

- 给出一个序列a1,a2,…,an,找到序列a的子序列a(i1),a(i2),...,a(il)，1<=i1<i2<...<il<=n，a(i1)<a(i2)<...<a(il),使得l最大

最长上升子序列

- 状态：f(i)表示序列a1,a2,…,ai的包含ai的最长的子序列的长度
- 阶段：按i从小到大划分阶段，即按照i从小到大的顺序计算f(i)，计算f(i)时只依赖于前面计算过的f值
- 决策：枚举f(i)对应的子序列a(i1),a(i2),…,a(i(l-1)),i中i(l-1)的值，设i(l-1)=j,则问题转化为在a1,a2,…,aj中找最长上升子序列

最长上升子序列

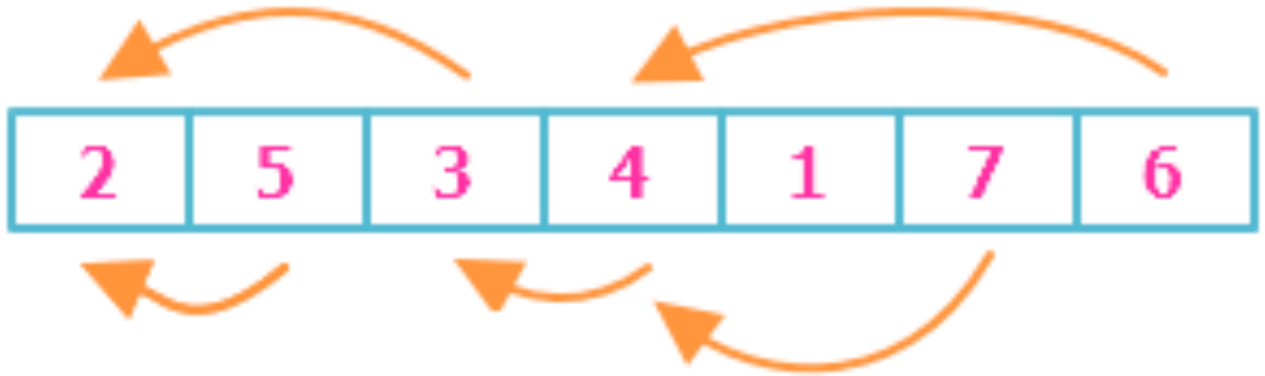
- 状态转移方程： f(i)=max{f(j)+1},(1<=j<i,aj<ai)

- 时间复杂度： O(n^2)

- 可以用树状数组优化到 O(nlogn)

最长上升子序列

转移过程



num的值　1　2　2　3　1　4　4

- 我们可以维护一个栈，表示长度当前为k的最长上升子序列的最末一个数的最小值，显然它是单调的。

- 每次用二分查找来更新，复杂度O(nlogn)

数字三角形

- N层数字三角形, N<=15
- 从顶部出发，在每一结点可以选择向左走或得向右走，一直走到底层。
- 要求找出一条路径，使路径上的值最大。
- CodeVS1220

数字三角形

- 暴力搜索?
- 每一层枚举向左or向右
- 因为有N层 每层都要选择
- 复杂度为O(2^N)

数字三角形

```
int F(int x,int y)
{
    if(x==n) return a[x][y];
    return max(F(x+1,y),F(x,y+1))+a[x][y];
}
```

“重复子问题”

子结点会被重复搜到，但是答案不变。

>>>记忆化搜索

数字三角形

f[N][N]，记录数组，初始值为-1；

并令f[n][i]=a[n][i];

```
int F(int x,int y)
{
    if(f[x][y]!=-1) return;
    return f[x][y]=max(F(x+1,y),F(x,y+1))+a[x][y];
}
```

数字三角形

- 递推
- 另一种实现思路
- 记忆化是由上而下 思路直观
- 递推是从底层往上走，不需要递归的额外开销

数字三角形

递推

f[n][i]=a[n][i];

```
for(int i=n-1 to 1)
    for(int j=i to 1)
        f[i][j]=max(f[i+1][j],f[i+1][j+1])+a[i][j];
```

划分型 dp

数的划分

- 将整数n分成k份，n<=100。
- 每份不能为空，任意两种划分方案不能相同(排序后相同也算相同)。
- 输出划分方案数。
- CodeVS 1039

数的划分

- f[i][k][x]，将i分成k个数，并且其中最大数为x的方案数。
- for(y=1 to x) f[i][k][x]+=f[i-x][k-1][y];
- answer=sigma f[N][K][x]; (1<=x<=N);

数的划分

- 进一步优化到O(n^2)。
- f[i][k]=f[i-1][k-1]+f[i-k][k];

数的划分

- 将整数n分成k份，n<=100。
- 每份不能为空，任意两种划分方案不能相同(排序后相同也算相同)。
- 输出划分方案数。
- CodeVS 1039

抄书问题

将长度为N的序列ai分成K段，使得最大段最小。

1 2 3 4 5 6 7 8 9 分 3段

1 2 3 4 | 5 6 7 | 8 9 -> 18

1 2 3 4 5 | 6 7 | 8 9 -> 17(更优)

CodeVS 3162

抄书问题

f(i,j)代表前i个数分为j份的最优方案

怎么转移?

枚举上一段的末尾k　k+1~i分为一段

max(f(k,j-1),sum[i]-sum[k]))。

抄书问题

状态：f[i][j]，将前i个数分为j段，子问题的最优值。

转移：for(int k=1 to i-1)//k为上一段末尾，k+1~i为一段

f[i][j]=min(f[i][j],max(f[k][j-1],sum[i]-sum[k]));

//sum为前缀和，快速计算一段的和

f[N][K]即为答案。

棋盘型 dp

方格取数

设有 N*N 的网格图 (N<=10) ， 我们将其中的某些方格中填入正整数，而其他的方格中则放入0。

某人从图的左上角的A点出发，可以向下行走，也可以向右走，直到到达右下角的B点。在走过的路上，他可以取走方格中的数（取走后方格中变为0）。

某人从A到B走一次，求取得数最大的方案。

CodeVS 1043

方格取数

先想想怎么算每次向下或向右走，从左上走到右下的方案数?

f(i,j)表示从(1,1)走到(i,j)的方案数。

f(i,j)=f(i-1,j)+f(i,j-1)

方格取数

f(i,j)表示从(1,1)走到(i,j)的最大值。

f(i,j)=max(f(i-1,j),f(i,j-1))+a[i][j]。

练习题

教主的后花园

- 教主有着一个环形的花园，他想在花园周围均匀地种上n棵树，但是教主花园的土壤很特别，每个位置适合种的树都不一样，一些树可能会因为不适合这个位置的土壤而损失观赏价值。教主最喜欢3种树，这3种树的高度分别为10，20，30，并且会告诉你每块地种每种树的观赏价值。

- 教主希望这一圈树种得有层次感，所以任何一个位置的树要比它相邻的两棵树的高度都高或者都低，并且在此条件下，教主想要你设计出一套方案，使得观赏价值之和最高。

- 1 <= n <= 10^5

教主的后花园

- 假设我们知道了前i棵树的种植方案，怎么向第i+1棵递推
- 需要知道第i棵树的高度，以及前一棵树的高度
- 实际上可以简化成四种情况
- 第i棵树高10；第i棵树高20且后一棵要更高；第i棵树高20且后一棵要更矮；第i棵树高30

教主的后花园

- 若用F(i, 0 ~ 3)表示，则转移方程

```
f[i][0]=max(f[i-1][2],f[i-1][3])+c[i][0];
f[i][1]=f[i-1][3]+c[i][1];
f[i][2]=f[i-1][0]+c[i][1];
f[i][3]=max(f[i-1][0],f[i-1][1])+c[i][2];
```

- 对于环状的情况，实际上只要枚举第一棵的高度，多次动态规划即可

CF543A. Writing Code

有 n 个程序员要完成 m 行的代码，要求出现的 bug 数目不超过 b。

给定每个程序员写一行代码会出现的 bug 数，问你有多少种方案完成这 m 行代码。

$1 \le n, m \le 500$

CF543A. Writing Code

有 n 个程序员要完成 m 行的代码，要求出现的 bug 数目不超过 b。

给定每个程序员写一行代码会出现的 bug 数，问你有多少种方案完成这 m 行代码。

$1 \le n, m \le 500$

我们可以设 F(i,j,k) 为前 i 个人完成 j 行代码 bug 数目为 b 的状态

F(i,j,k)=F(i-1,j-1,k-bug[i])

空间不够用？上滚动数组，其实本质是个完全背包

CF698A.Vacations

Vasya 假期有两种活动，打比赛和运动。每天都有可以选择的情况，如 0，则只能休息，1，则只能打比赛或者休息，2，则只能去运动或者休息，3 则可以运动，也可以打比赛，可以休息。要求相邻两天不重复相同的活动，问 Vasya 最少休息几天。

$1 \leq n \leq 100$

CF698A.Vacations

Vasya 假期有两种活动，打比赛和运动。每天都有可以选择的情况，如 0，则只能休息，1，则只能打比赛或者休息，2，则只能去运动或者休息，3 则可以运动，也可以打比赛，可以休息。要求相邻两天不重复相同的活动，问 Vasya 最少休息几天。

$1 \le n \le 100$

如果定义问题为到第 i 天最少休息几天，可以发现无法实现状态的转移。我们可以发现，我们在第 $i+1$ 天的决策还与第 i 天做出的决策相关

我们定义问题为到第 i 天，且第 i 天选择休息/打比赛/运动 的最少休息时间。

互质

给定一个正整数序列 a1, a2, ..., an。 你需要挑出这个序列的一个子序列，使得这个子序列的任意两个相邻元素不互质。

n ≤ 1000，ai ≤ 100000。

互质

给定一个正整数序列 a1, a2, ..., an。 你需要挑出这个序列的一个子序列，使得这个子序列的任意两个相邻元素不互质。
n ≤ 1000，ai ≤ 100000。

设 fi 表示选择到第 i 个元素，并且以这个元素结尾时，最长的子序列长度。

则 fi = max{fj} + 1(如果第 i 个元素和第 j 个元素不互质)。 时间复杂度 O(n^2)。
n ≤ 100000?

优化的关键在于:两个数不互质，只需要它们有共同的一个质因子即可。

互质

求出 100000 内的所有质数。
设 g(i, j)表示只用前 i 个元素(这时不需要保证第 i 个元素在结尾),
并且最后一个元素包含第 j 个质数。
如果 a_i 不包含包含第 j 个质数，那么 g(i, j)= g(i-1, j)。

如果 a_i 包含包含第 j 个质数，同时它包含第 k 个质数，那么 $g_{i,j}$ = max{g(i-1,k)} + 1。

鸡蛋的硬度（紫书）

- 如果一个蛋从高楼的第a层摔下来没摔破，但是从a+1层摔下来时摔破了，那么就说这个蛋的硬度是a。

鸡蛋的硬度（紫书）

- 如果一个蛋从高楼的第a层摔下来没摔破，但是从a+1层摔下来时摔破了，那么就说这个蛋的硬度是a。

- “假如有很多同样硬度的鸡蛋，那么我可以用二分的办法用最少的次数测出鸡蛋的硬度”，小A对自己的这个结论感到很满意，不过很快麻烦来了，“但是，假如我的鸡蛋不够用呢，比如我只有1个鸡蛋，那么我就不得不从第1层楼开始一层一层的扔，最坏情况下我要扔100次。如果有2个鸡蛋，那么就从2层楼开始的地方扔……等等，不对，好像应该从1/3的地方开始扔才对，嗯，好像也不一定啊……3个鸡蛋怎么办，4个，5个，更多呢……”。

鸡蛋的硬度（紫书）

- 如果一个蛋从高楼的第a层摔下来没摔破，但是从a+1层摔下来时摔破了，那么就说这个蛋的硬度是a。

- “假如有很多同样硬度的鸡蛋，那么我可以用二分的办法用最少的次数测出鸡蛋的硬度”，小A对自己的这个结论感到很满意，不过很快麻烦来了，“但是，假如我的鸡蛋不够用呢，比如我只有1个鸡蛋，那么我就不得不从第1层楼开始一层一层的扔，最坏情况下我要扔100次。如果有2个鸡蛋，那么就从2层楼开始的地方扔……等等，不对，好像应该从1/3的地方开始扔才对，嗯，好像也不一定啊……3个鸡蛋怎么办，4个，5个，更多呢……”。

- 给 n，m <= 100，表示最高层数和鸡蛋数，问使用最优策略在最坏情况下所需要的扔鸡蛋次数。

鸡蛋的硬度（紫书）

- 先想一个简单问题：只有两个鸡蛋的问题

- 两个软硬一样但未知的鸡蛋。有座100层的建筑，要你用这两个鸡蛋确定哪一层是鸡蛋可以安全落下的最高位置。可以摔碎两个鸡蛋。

鸡蛋的硬度（紫书）

- 先想一个简单问题：只有两个鸡蛋的问题

- 两个软硬一样但未知的鸡蛋。有座100层的建筑，要你用这两个鸡蛋确定哪一层是鸡蛋可以安全落下的最高位置。可以摔碎两个鸡蛋。

- 这是典型的动态规划问题。假设f[n]表示从n层楼找到摔鸡蛋不碎安全位置的最少判断次数。假设第一个鸡蛋第一次从第i层扔下，如果碎了，就剩一个鸡蛋，为确定下面楼层中的安全位置，必须从第一层挨着试，还需要i-1次；如果不碎的话，上面还有n-i层，剩下两个鸡蛋，还需要f[n-i]次（子问题，n层楼的上n-i层需要的最少判断次数和n-i层楼需要的最少判断次数其实是一样的）。因此，最坏情况下还需要判断max(i-1,f[n-i])次。

鸡蛋的硬度（紫书）

- 先想一个简单问题：只有两个鸡蛋的问题

- 两个软硬一样但未知的鸡蛋。有座100层的建筑，要你用这两个鸡蛋确定哪一层是鸡蛋可以安全落下的最高位置。可以摔碎两个鸡蛋。

- 这是典型的动态规划问题。假设f[n]表示从n层楼找到摔鸡蛋不碎安全位置的最少判断次数。假设第一个鸡蛋第一次从第i层扔下，如果碎了，就剩一个鸡蛋，为确定下面楼层中的安全位置，必须从第一层挨着试，还需要i-1次；如果不碎的话，上面还有n-i层，剩下两个鸡蛋，还需要f[n-i]次（子问题，n层楼的上n-i层需要的最少判断次数和n-i层楼需要的最少判断次数其实是一样的）。因此，最坏情况下还需要判断max(i-1,f[n-i])次。

- 状态转移方程：f[n] = min{1+max(i-1,f[n-i]) | i=1…n }

- 初始条件: f[0]=0（或f[1]=1）

鸡蛋的硬度（紫书）

- 推广成n层楼，m个鸡蛋

- 还是动态规划。假设f[n,m]表示n层楼、m个鸡蛋时找到摔鸡蛋不碎的最少判断次数。则一个鸡蛋从第i层扔下，如果碎了，还剩m-1个鸡蛋，为确定下面楼层中的安全位置，还需要f[i-1,m-1]次（子问题）；不碎的话，上面还有n-i层，还需要f[n-i,m]次（子问题，n层楼的上的n-i层需要的最少判断次数和n-i层楼需要的最少判断次数其实是一样的）。

鸡蛋的硬度（紫书）

- 推广成n层楼，m个鸡蛋

- 还是动态规划。假设f[n,m]表示n层楼、m个鸡蛋时找到摔鸡蛋不碎的最少判断次数。则一个鸡蛋从第i层扔下，如果碎了，还剩m-1个鸡蛋，为确定下面楼层中的安全位置，还需要f[i-1,m-1]次（子问题）；不碎的话，上面还有n-i层，还需要f[n-i,m]次（子问题，n层楼的上的n-i层需要的最少判断次数和n-i层楼需要的最少判断次数其实是一样的）。

- 状态转移方程：f[n,m] = min{1+max(f[i-1,m-1], f[n-i,m]) | i＝1..n}

- 初始条件：f[i,0]=0（或f[i,1]=i），对所有i

鸡蛋的硬度（紫书）

```
void calc()
{
    int current;
    for (int eggs = 1; eggs <= 10; eggs++) //鸡蛋数
        for (int level = 0; level <= 100; level++) //楼层数
            if (eggs == 1)    // 只有一个鸡蛋的情况，边界条件
                f[level][eggs] = level;    //最坏的情况是有多少层试多少次
            else if (level == 0)     // 0层楼的情况，边界条件
                f[level][eggs] = 0; //没有楼层的情况，一次都不用试
            else   //鸡蛋数大于1，楼层大于0的情况
            {   f[level][eggs] = MAX;  //先假设一个最大值
                for (int tryl = 1; tryl <= level; ++tryl) //枚举第一次尝试的层次
                { current = 1 + max(f[tryl - 1][eggs - 1],  //鸡蛋碎了的情况
                                    f[level-tryl][eggs]); //鸡蛋没有碎的情况
                    if (current < f[level][eggs])
                        f[level][eggs] = current;
                }
            }
}
```

鸡蛋的硬度（紫书）

```
int calc(int level, int eggs)
{
    if(f[level][eggs] < MAX) // 如果不是最大值，说明计算过了
        return f[level][eggs]; // 不加这个会 time out
    if(eggs == 1) // 只有一个鸡蛋的情况，边界条件
        return level; //最坏的情况是有多少层试多少次
    if(level == 0)     // 0层楼的情况，边界条件
        return 0; //没有楼层的情况，一次都不用试
    for(int i = 1; i <= level; i++)//枚举第一次尝试的层次
    {
        int current = 1 + max(calc(i - 1, eggs - 1), //鸡蛋碎了的情况
                              calc(level - i, eggs)); //鸡蛋没有碎的情况，注意此处与动态规划的区别
        if (current < f[level][eggs])
            f[level][eggs] = current;
    }
    return f[level][eggs];
}
```

最美妙的矩阵

- 在一个 n * m 的矩阵中找一个最大子矩阵，满足横竖单调递增（可以相等，也就是对于每一个最优子矩阵的元素都要满足：
- a[i, j]>=a[i - 1, j] and a[i, j]>=a[i, j - 1]
- 0<n, m<=200

最美妙的矩阵

- 预处理column[i][j]以及can[i][j][k]数组。
- 其中column[i][j]表示坐标为i，j的点能向上单调递减的长度。
- 用can[i][j][k]表示从第i行到第j行中的第k列能否接到第i行到第j行的k-1列中。
- 接下来用f[i][j][k]表示从第i行到第j行中，第k列的最大子矩阵。

最美妙的矩阵

- 预处理column[i][j]以及can[i][j][k]数组。
- 其中column[i][j]表示坐标为i，j的点能向上单调递减的长度。
- 用can[i][j][k]表示从第i行到第j行中的第k列能否接到第i行到第j行的k-1列中。
- 接下来用f[i][j][k]表示从第i行到第j行中，第k列的最大子矩阵。

  F[i][j][k] =

  f[i][j][k-1]+j-i+1 (can[i][j][k]=true)

  j-i+1 (can[i][j][k]=false and column[j][k]>=j-i+1)

  0 （else）

ZJOI2006. 物流运输

- 物流公司要把一批货物从码头 A 运到码头 B。由于货物量比较大，需要 n 天才能运完。货物运输过程中一般要转停好几个码头。物流公司通常会设计一条固定的运输路线。由于各种因素的存在，有的时候某个码头会无法装卸货物。这时候就必须修改运输路线，让货物能够按时到达目的地。但是修改路线是一件十分麻烦的事情，会带来额外的成本。因此物流公司希望能够订一个 n 天的运输计划，使得总成本尽可能地小。

- 任何时间都存在至少一条从码头A到码头B的运输路线。

- 总成本 = n天运输路线长度之和 + K * 改变运输路线的次数（即不同的运输路线数 - 1）

- $1 \le n \le 100, 1 \le m \le 20$

ZJOI2006. 物流运输

- 令 F(i) 为到第 i 天的成本
- F(i) = min(F(j) + cost(j + 1,i) + K)
- 如何求 cost(i, j)?
- 最短路

CF175E.Power Defence

- 怪物以从1米每秒的速度从(-∞,0) 运动到 (+∞,0)。
- 你可以在(x,1), (x,-1)摆放3种塔：火、电、冰
- 其中x为整数
- 每种塔都有攻击半径，并给定火电塔的dps
- 目标在k座冰塔的攻击范围内，速度将被减少至 1/(k+1)
- 给定3种塔的数量na,nb,nc，求对目标造成的最大伤害
- na+nb+nc<=20

CF175E.Power Defence

- 易得最优解中必然有一种方案是所有塔“密集分布”：所有塔的摆放位置确定了，只剩每个位置摆什么塔。

- f[i][x][y]表示前i个位置，已使用x座火塔，y座电塔时造成的最大输出，则冰塔使用了i-x-y座

- 冰塔会影响单位速度，进而影响输出，后效性！

CF175E.Power Defence

- 易得最优解中必然有一种方案是所有塔“密集分布”：所有塔的摆放位置确定了，只剩每个位置摆什么塔。

- f[i][x][y]表示前i个位置，已使用x座火塔，y座电塔时造成的最大输出，则冰塔使用了i-x-y座

- 冰塔会影响单位速度，进而影响输出，后效性！

- dfs求出冰塔的所有可能的摆放方案

CF175E.Power Defence

- 易得最优解中必然有一种方案是所有塔“密集分布”：所有塔的摆放位置确定了，只剩每个位置摆什么塔。

- f[i][x][y]表示前i个位置，已使用x座火塔，y座电塔时造成的最大输出，则冰塔使用了i-x-y座

- 冰塔会影响单位速度，进而影响输出，后效性！

- dfs求出冰塔的所有可能的摆放方案

- 在冰塔的位置确定后，目标在任一段的移动速度也是确定的

- O(n^2)DP求出火塔和电塔的具体摆放方案。

- 时间复杂度：(C(20,10)*(na+nb)^2)。

Q & A