动态规划

(dynamic programming)

动态规划

- 利用最优化原理把多阶段过程转化为一系列单阶段问题，利用各阶段之间的关系，逐个求解。

状态表示

- 动态规划过程中，需要存储的量为**状态表示**和**最优化值**
- **状态表示**是对当前子问题的解的局面集合的一种充分的描述
- **最优化值**则是对应的状态集合下的最优化信息，我们最终能通过其直接或间接得到答案。
- 对于状态的表示，要满足两条性质:
  - 具有最优化子结构：即问题的最优解能有效地从问题的子问题的最优解构造而来
  - 具有简洁性：尽可能的简化状态的表示，获得更优的时间复杂度

状态的转移

- 由于具有最优化子结构，所以求当前状态的最优值可以通过其他的状态的最优值加以变化而求出。
- 所以，当一个状态的所有子结构都已经被计算之后，我们就可以通过一个过程计算出他的最优值。这个过程就是**状态的转移。**
- **状态的转移**需要满足要考虑到所有可能性。

状态设计

- 动态规划时间复杂度：状态数×状态转移复杂度
- 一个良好的状态设计是解决许多问题的关键，也是大部分动态规划问题的考察点。
- 所以思考一道问题如何使用动态规划解决时，第一步就是设计状态。
- 下面介绍一些常见的状态设计的技巧。

状态设计

- **增加维数**：当我们发现一个状态表示的并不能充分地表示出所有的特征或不满足最优子结构时，我们可以通过增加状态的维数来满足要求。
- **交换状态与最优值**：由于最优值的规模往往没有太多的限制，所以在某些情况下，我们交换最优值与某一位状态记录的内容，而缩小规模。
- **对部分问题进行Dp**：直接对原问题Dp求出答案可能比较困难。可以仅有Dp求出原问题答案的一部分信息，然后用其他办法或另外的Dp求出答案。

状态设计

- **利用二分减少状态：**对于动态规划过程中某一些大小限制性的状态，我们可以在最开始的时候二分这个参数，而做到利用一个log的代价减少一维状态。
- **分段的状态设计：**在某一些问题中，几维状态的数量互相相关，比如第二位状态随第一维增大而越来越小，所以我能可以将一个Dp拆为几个阶段，从而避免状态规模的浪费。

动态规划

- 背包Dp
- 树形Dp
- 数位Dp
- 概率Dp
- 状态压缩DP
- Dp套Dp

背包Dp

- 一般是给出一些“物品”，每个物品具有一些价值参数和花费参数，要求在满足花费限制下最大化价值，或最大/小化一个给出的式子。

Bzoj 2298

- 一次考试共有n个人参加，第i个人说：“有ai个人分数比我高，bi个人分数比我低。”可以有相同分数，问最多有多少人在讲真话
- $n \leq 100000$

Bzoj 2298

- 每个人的话相当于排名为[ai+1,n-bi]的人分数一样，如果两个人的话所代表的的区间有相交但不完全重合，那么这两个人的话冲突。
- 所以这道题就转变为了选择尽量多的不相交线段。

Bzoj 1190 [HNOI2007]

- 给你N颗宝石，每颗宝石都有重量和价值V(V<=1e9)。要你从这些宝石中选取一些宝石，保证总重量不超过W，且总价值最大为，并输出最大的总价值，每颗宝石的重量符合a*2^b（a<=10;b<=30）。

- N<=100，W<=2^30

Bzoj 1190 [HNOI2007]

- 由于重量和价值都很大，所以不能直接Dp。
- 物品重量很有特点a*2^b，ab都很小。
- b值大的物品不会影响零碎剩余的重量上限。
- 将物品按b值分阶段处理。
- 在阶段内，直接忽略不足2^b的部分，f[i][j]表示前i个物品，剩余的能用重量为j*2^b
- 从上一阶段到下一阶段时，将f[i][j]->f’[i][j*2+零碎部分]，注意到n=100，a<=10，所以剩余重量最多纪录到1000即可。

树形Dp

- 与树或图的生成树相关的动态规划。
- 以每棵子树为子结构，在父亲节点合并。
- 巧妙利用Bfs或Dfs序，可以优化问题，或得到好的解决方法。
- 可以与树分治，以及树上的数据结构相结合。
- 虚树题的显著特征：询问涉及总点数<=多少
- 树形Dp的时间复杂度要认真计算，容易出错。

Cover

- 给定一棵*n* 个点的无根树，节点*i* 的权值为*wi*，如果选择i，则可以将距离这个点wi以内的点全部覆盖，任意一边的长度都为1。
- 问至少选择多少点，能将整棵树上所有点覆盖。

- n<=100

Cover

- f(i,j)表示以i为根的子树，j>=0表示还能覆盖到外面距离为j的点，j<0则表示子树内最深没有覆盖的点深度为-j，此时最小选定的点数。
- 选择i点 f(i,j)<-min(f(son,-wi~wi-1)))
- 如果不选i点，
  - 如果j为非负数，枚举一个向外覆盖最大的孩子
  - f(i,j)<-f(u,j+1)+min(f(son,-(j-2)~j))
  - 如果j为负数，枚举一个没有被覆盖最深的孩子
  - f(i,j)<-f(u,j-1)+min(f(son,(j-1)~ -(j-1)))

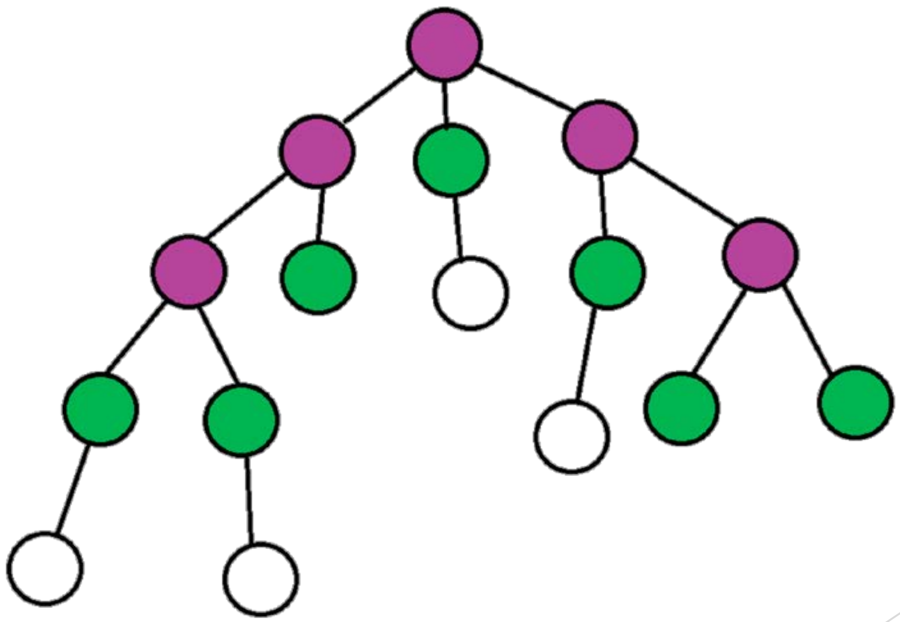
Tree chain problem

- 给定一棵有n 个点的树，以及m 条树链，其中第i 条树链的价值为wi，请选择一些没有公共点的树链，使得价值和最大。

- n,m<=1e5

Tree chain problem

- 考虑树形DP，设f(x) 为以x 为根的子树内选取不相交树链的价值和的最大值，枚举一条LCA 为x 的链u; v; w，那么当前方案的价值为w+ 去除u 到v 路径上的点后深度最小的点的f 的和。



Mine

- 给出一棵树，每个点上都有Wi的矿石，现在可以在一些点上设立仓库，每设立一个仓库，就要花费K的代价。最后每个点的代价如下计算，如果离他最近的点的距离为dis，则代价为$Wi \times f(dis)$。最小化总代价。树边长度都为1。
- N<=200

Mine

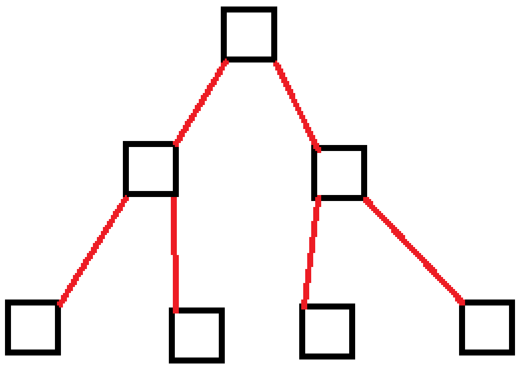
- 设状态f(i,j)表示距离i最近的仓库为j(j还未建立)时，i子树的总代价
- 转移考虑i的孩子的最近仓库
- 如果也是j：f(i,j)<-f(son,j)
- 否则一定在孩子的子树中有一个更近的仓库，枚举那个仓库k，转移时加上费用。

树形依赖背包

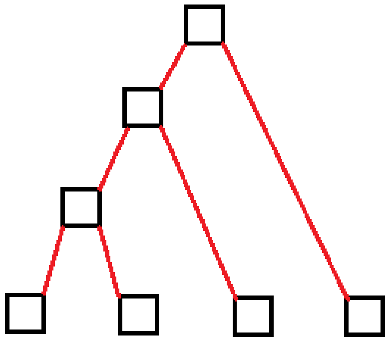
- 给出一棵有根树，每个节点都是一个物品，具有价值和重量，如果一个物品要被选择，那么它的父亲必须被选择。求限制重量内的最大价值和。

树形依赖背包

- 朴素做法O(n^3)：f[i][j]表示在以i为根子树中选择，i强制选择，重量为j的最大价值，转移时每次将一个孩子暴力合并到父亲上。
- 两个一维数组的背包合并是n^2的。
- 但一个一维数组和一个单独的物品合并是线性的。



N^3



N^2

树形依赖背包

- 所以有两种方法；
- 1.在dfs序上Dp，如果不选一个点，则跳过代表他的子树的dfs上连续一段。
- 2.不是每次将孩子与自己合并，然后将数组直接传给孩子，再从孩子递归下去，最后原来自己的Dp数组和孩子的Dp数组直接在对应重量的价值中取max。

数位Dp

- 经典的数位Dp是要求统计符合限制的数字的个数。
- 可以通过记录决定了前多少位以及大小关系来Dp。
- 或者Dp处理不同位数的数的个数，然后Dp统计。
- 善用不同进制来处理。

Codeforces 55 D

- 求[L，R] 中所有可以被它所有非零数位整除的数的个数。
- 1<=L<=R<=91e8

Codeforces 55 D

- 考虑[1,R]-[1,L-1]
- 1~9的lcm只有2520，所以数x等价于x+k*2520
- f(i,a,b,0/1)表示考虑了前i位，此时前i为组成的数mod 2520为a，数位的lcm为b，且小于还是等于边界。

Sdoi2013淘金

- 一个二维坐标。X轴，Y轴坐标范围均为1..N。初始的时候，所有的整数坐标点上均有一块金子，共N*N块。
- 一阵风吹过，初始在(i，j)坐标处的金子会变到(f(i)，f(j))坐标处。其中f(x)表示x各位数字的乘积(去除前导零)，例如f(99)=81，f(12)=2，f(10)=0。如果金子变化后的坐标不在1..N的范围内，我们认为这块金子已经被删除。同时可以发现，对于变化之后的游戏局面，某些坐标上的金子数量可能不止一块，而另外一些坐标上可能已经没有金子。这次变化之后，游戏将不会再对金子的位置和数量进行改变
- 求出风吹过之后金块数量前K大的坐标的金块数量和
- 答案可能很大，输出对10^9+7取模之后的答案。
- N < = 10^12 ,K < = 100000

Sdoi2013 淘金

- 两维分开考虑，考虑一维，最后优先队列即可统计乘积答案。
- 我们发现因为值得质因子只有4中，实际的f可能的取值很少，用f(i,S,0/1)表示从高到低到了第i位，S为当前乘积离散化的值，且考虑过得位置是小于还是等于。

BZOJ 3329

- 求方程$X xor\ 3X = 2X$
- 1.小于等于n的正整数解
- 2.小于等于2^n的正整数解(mod 1e9+7)

- n<=1e18

BZOJ 3329

- $X\ xor\ 3X = 3X \rightarrow X\ xor\ 2X = 3X$
- 又有$X + 2X = 3X$，异或是不进位的加法。
- 所以x和2x中不存在同时为1的二进制位，而2x是x左移一位，即x中无相邻的1。
- 第一问直接f(i,0/1,0/1)表示到了第i位，上一位是0/1，前i位小于还是等于n。
- 第二问可以递推f(i,0/1)表示长度为i的二进制位，最高位为0/1的方案数，矩阵乘法加速。

概率/期望Dp

- 求在一些限制条件下的概率/期望。
- 求概率常常为正推，求期望常常为逆推
- 有拓扑序的问题直接Dp，没有拓扑序的问题高斯消元是常见解法。
- 期望的可加性。
- 概率的全概率公式，贝叶斯公式。
- 思考问题时，考虑将问题取反是否更可行。

$$P(B)=\sum_{i=1}^{n}P(A_i)P(B|A_i) \quad P(B_i|A)=\frac{P(B_i)P(A|B_i)}{\sum_{j=1}^{n}P(B_j)P(A|B_j)}$$

Hnoi 2015 亚瑟王

玩家有一套卡牌，共 n张。游戏时，玩家将 n 张卡牌排列成某种顺序，排列后 将卡牌按从前往后依次编号为 1 ~ n。本题中，顺序已经确定，即为输入的顺序。
每张卡牌都有一个技能。第 i 张卡牌的技能发动概率为 pi，如果成功发动，则会对敌方造成di点伤害。也只有通过发动技能，卡牌才能对敌方造成伤害。基于现实因素以及小K非洲血统的考虑，pi不会为 0，也不会为 1，即 0 < pi < 1。
一局游戏一共有 r 轮。在每一轮中，系统将从第一张卡牌开始，按照顺序依次考虑每张卡牌。在一轮中，对于依次考虑的每一张卡牌：
1 如果这张卡牌在这一局游戏中已经发动过技能，则
    1.1 如果这张卡牌不是最后一张，则跳过之（考虑下一张卡牌）；
    1.2 否则（是最后一张），结束这一轮游戏。
2 否则（这张卡牌在这一局游戏中没有发动过技能），设这张卡牌为第 i 张
    2.1将其以 pi的概率发动技能。
    2.2如果技能发动，则对敌方造成 di点伤害，并结束这一轮。
    2.3如果这张卡牌已经是最后一张（即 i 等于n），则结束这一轮；否则考虑下一张卡牌。

请帮助小 K 求出这一套卡牌在一局游戏中能造成的伤害的期望值。

Hnoi 2015

- 直接的求好像并不好表示状态，考虑期望的可加性，对每张卡牌分开考虑，求出每张卡牌发动的概率。
- Dp[i][j]表示第i张卡牌有j次发动技能的机会的概率，显然后一张卡牌的机会与前一张相关。
  - Dp[i][j] += Dp[i-1][j]*(1-Pi-1)^j
  - Dp[i][j] += Dp[i-1][j+1]*(1-(1-Pi-1)^(j+1))
- 最后求期望即可。

Commutative algebra

- 地面上一共有n个球，一开始有一些是黑色的，有一些是白色的。每次随机选择一个区间（一共有n(n + 1)/2个区间，每个区间有相等的概率被选择），把这个区间的颜色反转，即将该区间中白球变黑球，黑球变白球。
- 期望多少次反转能够使得整个区间都是白色的。
- $n \leq 20$

Commutative algebra

- 一个区间很不好考虑。
- 如果使用Ai=0/1表示第i个球的颜色，我们设置一个Bi=Ai xor Ai-1，(B1=A1，Bn=An)，Ai全部为0的充要条件为Bi全部为0。
- 原来对于一个区间的操作就变为了对两个数的取反操作，而且由于是等概率的，所以Bi之间没有相对位置关系，只需记录有多少个0，多少个1即可。
- 高斯消元。

SRM641 div1 900pts BitToggler

给你一个长度为n(n<=20)的01串，以及一个指针，初始时指针在第x0个字符上。每回合随机一个1到n中的数j，如果指针之前在i上，就花费 | j-i | 的时间把指针从i移动到j上，并且把01串的第j位取反。不停随机，直到01串变成全0或者全1是结束，问到结束前期望花费的时间是多少？

SRM641 div1 900pts BitToggler

- 如果直接状态，由于状态间存在环，不能用类似递推的dp解决，而状态数是高斯消元不能接受的。
- 但可以发现产生代价的移动只有n^2种，由于期望的可加性，我们只要对于每一种代价求出他对期望的贡献即可。
- 那么对于一种移动i->j，我们只需关心i与j的状态，其他位置的颜色数目以及当前指针的位置，这样状态就可以被压缩起来了，状态只有大概n*3*2种，高斯消元求解即可。

Random

- 给出一个n(n<=18)个数的排列A1~An，执行以下操作，直到序列变为升序。
- 统计当前有多少个位置i(i<n)满足Ai>Ai+1，如果有k个，就在这k个中等概率选择一个i，然后交换Ai和Ai+1，另这一次操作花费为i。
- 求期望的操作总花费。

- 题目来源: cls

Random

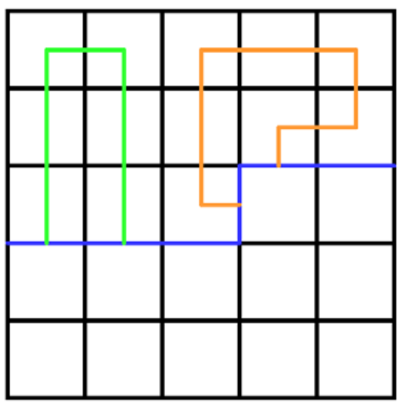
- 虽然状态的转移是不存在环的，但是直接Dp状态太多，为n!。仿照上一题的解法。
- 分别考虑数字i与它后边的数交换所产生的花费的期望。
- 在考虑数字i时，只需考虑i的位置，以及其他位置中每个数是否大于i，状态n*(2^n)。
- 由于是等概率选择，此时虽然这样与原来的k不同，但并不影响结果。

状态压缩Dp

- 当Dp状态不太好写成多维数组的形式，可以将状态压缩为一个数来记录。
- 插头Dp，连通性Dp，斯坦纳树，Dp of Dp。
- 只记录关键信息。
- 选择合适的进制，二的幂次的进制处理更方便，速度跟快。
- 适当的使用STL减少代码复杂度。

插头Dp

- 插头Dp一般是在网格图上要求求出满足特定条件的路径/回路等。
- 逐格Dp，记录轮廓线上路径端点的情况。
- 变化不多，但是写起来比较复杂。



斯坦纳树

- 能较为方便的通过许多需要较为复杂的状态压缩Dp的题(WC2008游览计划)
- 即对于Dp过程中，有明显拓扑序的转移直接转移，如果没有明显拓扑序，则可以通过类似Spfa的做法转移。

Bzoj 3205

- 给定一张h*w地图，一些地方有障碍物，一些地方有转向器，有k<=9个机器人，第i个机器人编号区间初始为[i,i]，现在可以向四个方向推机器人，机器人遇到障碍才会停，遇到转向器会转向，**停在同一个格子**的**两个编号区间相邻的机器人**可以合并，[l,mid]和[mid+1,r]合并后为[l,r]，求最少推多少次可以全部合并。

$h \leq 500$ $w \leq 500$ $k \leq 9$

Bzoj 3205

预处理每个地方向每个方向推动到达哪里

令f[l][r][x][y]表示在点(x,y)将编号在[l,r]区间内的机器人全部合并的最小推动次数。

推动：f[l][r][x][y]=min{f[l][r][tx][ty]+1}

此处转移不太好确定拓扑序，所以可以使用Spfa转移

合并：f[l][r][x][y]=min(f[l][mid][x][y]+f[mid+1][r][x][y])

这里正常转移

Dp套Dp

- 是一种状态压缩Dp。
- 将某一个Dp的过程压缩起来当做外部的大Dp的状态。
- 常用与统计Dp判断性问题的合法方案计数。

Square

给出一个n*n(n<=8)的棋盘，上面有一些格子必须是黑色，其它可以染黑或者染白，求有多少种染色方法使得棋盘上的最大连续白色子正方形边长为k

2014 Asia AnShan Regional Contest

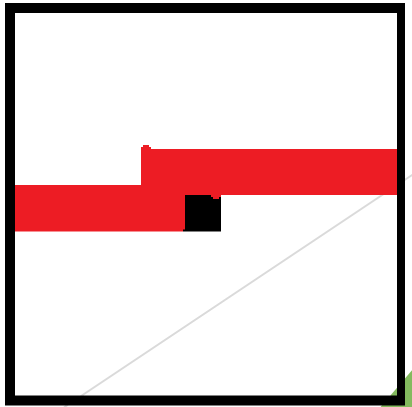
Square

- 如果给出了一个矩阵，如何Dp求最大子正方形

  $f(i,j)=\min(f(i-1,j-1),f(i,j-1),f(i-1,j))+1$ (i,j)白色

  $=0$ (i,j)黑色

- 现在要统计矩阵个数，则要求生成的矩阵f(n, n)=K，而f(n, n)要通过其他f值来求，所以只能讲f[][]当做状态压缩进来。
- Dp过程中答案只与n+1个dp值与已存在的最大值有关，又因为f(i,j-1)>=f(i,j)-1，状态只有几万。



动态规划的优化

- 矩阵快速幂
- 启发式合并
- 斜率优化
- 决策分治优化
- 个数限制转代价

斜率优化

- 方程类似f[i] = min{f[j]+a[i]+b[j]+c[i]*d[j]+e}
- 去掉常数和只与i相关的变量。
- f[i] = min{c[i]*d[j] + (f[j]+b[j])}
- 与直线方程b=kx+y类似。
- 最大/小化截距
- 这样的点一定在凸壳上。
- 维护凸壳，在凸壳上找到切点就是最优决策。

决策分治优化

- 例如IOI2014 Holiday
- 健佳打算参观台湾城市的景点，在台湾共有n(n<=1e5)个城市，它们全部位于一条高速公路上。这些城市连续地编号为0到n-1。对于城市i(0 < i< n-1)而言，与其相邻的城市是i-1和i+1。但是对于城市 0，唯一与其相邻的是城市 1。而对于城市n-1，唯一与其相邻的是城市n-2。
每个城市都有若干景点。健佳有d天假期并且打算要参观尽量多的景点。健佳已经选择了假期开始要到访的第一个城市。在假期的每一天，健佳可以选择去一个相邻的城市，或者参观所在城市的所有景点，但是不能同时进行。即使健佳在同一个城市停留多次，他也不会去重复参观该城市的景点。请帮助健佳策划这个假期，以便能让他参观尽可能多的景点。

很容易发现可能的路线只有几种，一直向右，一直向左，向右走折回再向左走，或者向左走再向右走。

设f[i],g[i],f1[i],g1[i]分别表示向右走i步，向左走i步，向右走最终返回st总共走i步，向左走最终返回st总共走i步，得到的最大愉悦值。

然后答案就是max{max(f1[i]+g[m-i],g1[i]+f[m-i]),i=0……m}

预处理这四个数组，发现四个函数的求法基本是一样的，以f[i]为例

还是枚举向右能到达的最远的点k，取i到k中最大的m-k+i个值

发现随着走的步数增加，到达的最远点只会变远，令最远点为d[i]，所以我们可以直接暴力枚举最远点算出f[mid]和d[mid]，然后对于i<=mid就知道d[i]一定不会大于d[mid]，分治计算下去即可，时间复杂度为O(nlog^2)

凸函数优化(IOI2016 Alien)

- 一个正方形$m \times m$的网格，其中有n个关键点，使用不超过k个小正方形将n个点覆盖，要求每个小正方形的主对角线落在大正方形网格的主对角线上，求最小的总面积。
- 60%: $n \le 50000, m \le 1000000, k \le 100$
- 100%：$n, k \le 100000, m \le 1000000$

凸函数优化(IOI2016 Alien)

- 可以将对角线以下的点翻转到对角线以上。
- 去除被包含的点，剩余的点按照 x 从小到大排序，y 一定也是从小到大的，且一个正方形会覆盖连续一段点。
- 令 $f[i][j]$ 表示使用了 i 个正方形，覆盖了前 j 个点的最小总面积，转移枚举最后一个正方形覆盖的是那一段点。
- $f[i][j] = \min\limits_{0 \le k < j} \{f[i-1][k] + (y_i - x_{k+1} + 1)^2 - max(0, y_k - x_{k+1} + 1)^2\}$
- 斜率优化，用栈维护，复杂度 $O(nk)$，只能通过 60% 的数据。

凸函数优化(IOI2016 Alien)

- 如果我们不限制选取的小正方形个数，而是改为每多选取一个小正方形就需要额外的 C 的面积。
- $g(j)$ 表示覆盖前 j 个点的最小面积
- $g(j) = \min_{k=1}^{n}\{f[k][j] + kC\}$
- $g(j) = \min_{0 \le k < j}\{g(k) + (y_j - x_{k+1} + 1)^2 - max(0, y_k - x_{k+1} + 1)^2 + C\}$
- $g(n)$ 可以通过斜率优化在线性时间内求出，并且可以同时求出此时最少需要的小正方形个数，然后尝试通过 $g(n)$ 求出对应的 $f[x][n]$ 的值。

End

Thanks