提高组动态规划（上）

Basic skills in [Dynamic Programming]

Ruan Xingzhi

洛谷网校 2021 夏

2021 年 7 月 26 日

intro

- 我是阮行止，曾经是 OIer，打过半年 ACM，现在是 CTFer

intro

- 我是阮行止，曾经是 OIer，打过半年 ACM，现在是 CTFer
- 直播打代码，翻车可能性微存

从 LIS 讲起

我们都知道，最长不下降子序列问题可以这样解决：

设计状态

以 $dp[x]$ 表示序列 a 中以 a_x 结尾的 LIS 长度。

设计转移

$$dp[x] = \max_{i<x, a_i \leq a_x} \{dp[i] + 1\}$$

DP 的状态和转移

一个问题可以 DP，是因为这个问题可以**从小问题的解，推断出大问题的解**。

我们可以从初始状态的解，推出最终状态的解，从而解决问题。

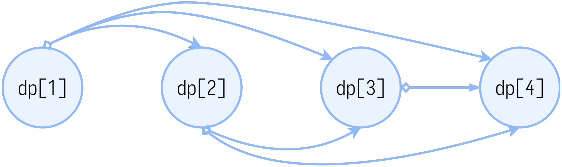
状态和转移的本质

首先，我们把 LIS 问题中的每个状态作为点，放在图上：

我们知道 $dp[1] = 1$，因为单个字符的 LIS 长度只有 1。现在想知道 $dp[4]$，这就需要通过转移来实现。

状态和转移的本质

如果状态 x 可以直接抵达状态 y，我们就连上 $x \to y$ 的有向边：



DP 的状态图

如果我们按以上方法绘图，那么立即就有几条性质：

1. DP 的每一个状态都对应着一个点
2. 每种可能的转移方式，都对应着一条有向边
3. DP 的求解顺序，等同于这张图上的拓扑排序
4. 整张图必须是 DAG，否则不可能找到合适的求解顺序

两种转移方式

上过普及组课的同学都知道，DP 有两种转移方法：

- pull 型的转移，主要考察「一个状态如何从其他状态推出结果」
- push 型的转移，主要考察「一个状态得到解之后，如何去更新其他状态」

本质上对应着拓扑排序的自顶向下方法和自底向上方法。无论采取哪一种转移，「每个状态最终求得解」的时机都是一样的，即拓扑序。

关于计数 DP

事实上，计数类 DP 不算动态规划。但由于其求值方式和 DP 很类似，我们可以把对普通 DP 的认识，直接应用到计数类 DP。

二维 DP：以过河卒为例

https://www.luogu.com.cn/problem/P1002

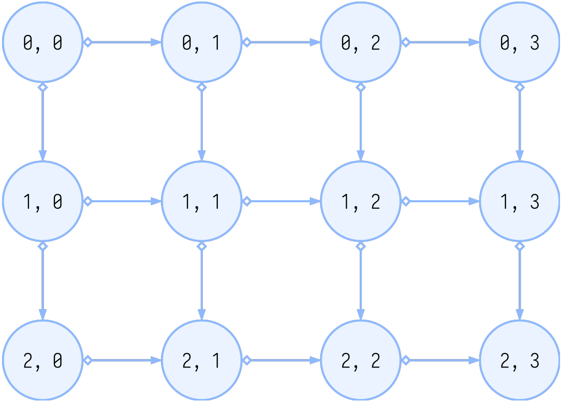
二维 DP：以过河卒为例

https://www.luogu.com.cn/problem/P1002

设计状态：$dp[x][y]$ 表示卒从 $(0,0)$ 走到目标点的方案数。
立即可以找到转移：

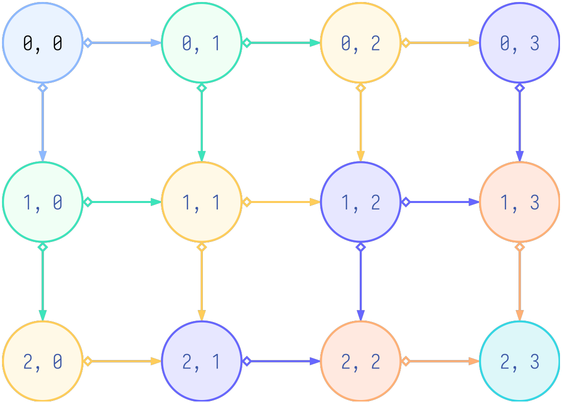
$$dp[x][y] = \begin{cases} 0, & marked[x][y] \\ dp[x][y-1] + dp[x-1][y], & otherwise \end{cases}$$

状态图



滚动数组技巧

一个 DP 可以通过滚动数组来优化，当且仅当其状态图是分层的，下 k 层的结果由上 d 层结果唯一确定。



过河卒 - 滚动数组优化

尽管过河卒的原始 DP 式并非「可以把数组滚掉一维」，但它的状态图是分层的，所以仍然可以滚。

我们按斜线来 DP，将右上角记为第 0 个元素。则不难有滚动后的 DP 式：

$$dp[x] = \begin{cases} 0, & marked \\ dp[x] + dp[x-1], & otherwise \end{cases}$$

注意循环顺序！如果我们从小到大枚举 x，则本层的 x 会更新本层的 $x+1$，本层的 $x+1$ 更新本层的 $x+2...$ 事实上本层的值应当由上一层来确定。

单层滚动数组的通用写法

起两个数组 `arr_old,arr_new`，分别保存旧层和新层。通过旧层计算出新层的值，然后覆盖掉旧层。

覆盖可以通过 memcpy 或者交换指针实现。

通用写法的优势：无需考虑特殊的求值顺序。

关于记忆化搜索

记忆化搜索有如下好处：

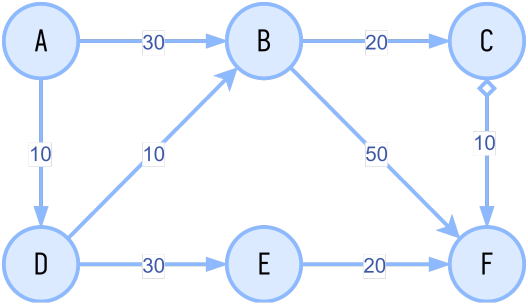
- 只需要直接实现 pull 型转移，无需在代码中考虑循环顺序
- 只经历「可能达到的状态」，不会经过无效状态，节省时间

所以很多时候我们是喜欢记忆化搜索的，尤其是区间 DP 问题。

写法上，判断「状态是否已经有记忆」可以采用 `visit` 数组或初值判断。如果 DP 结果可能是 0，一定要把初值设为其他值。

DAG 最短路

DAG（有向无环图）上的最短路问题：给定一个起点 S 和一个 T，求 S 到 T 之间的最短距离。



DAG 最短路

显然，问题可以通过 DP 来解决。我们记 $f(x)$ 表示从起点到 x 的距离，那么显然有

$$f(x) = \min_{\text{edge}:u\to x} \{f(u) + dis(u \to x)\}$$

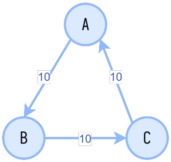
解释：起点到 x 的距离，由那些能直接到达 x 的点决定。经由 u 抵达 x 的代价是 $f(u)$ 加上最后这条边的长度。

推广

这个方法能否推广到任意图的最短路?

推广

「我们一直以来的努力，全部木大」



问题的根源是「循环依赖」。$f(B)$ 依赖于 $f(A)$，$f(A)$ 依赖于 $f(C)$，而 $f(C)$ 又依赖于 $f(B)$。

解决循环依赖

那我们就换一种设计状态的方法：记 `dp[k][x]` 为从起点开始，经历不超过 k 个点，到达 x 的最短路径长度。
我们一举解决了循环依赖问题，`dp[k][x]` 只会依赖于 `dp[k-1][.]`。

状态转移如下：

$$dp[k][x] = \min_{\texttt{edge}:u \to x} \{dp[k-1][u] + dis(u \to x)\}$$

解决循环依赖

那我们就换一种设计状态的方法：记 dp[k][x] 为从起点开始，经历不超过 k 个点，到达 x 的最短路径长度。
我们一举解决了循环依赖问题，dp[k][x] 只会依赖于 dp[k-1][.]。

状态转移如下：

$$dp[k][x] = \min_{\texttt{edge:}\, u \to x} \{dp[k-1][u] + dis(u \to x)\}$$

注意到这是分层的。滚动数组优化之后，即是 Bellman-Ford 单源最算路算法。

以 DP 的视角看 Floyd

我们刚刚解决了单源最短路问题。今要求出图中任意两点之间的距离，我们可以如何 DP?

第一时间想到 `dp[u][v]` 表示从 u 到 v 的距离。但由于循环依赖问题，没有合适的转移。

以 DP 的视角看 Floyd

我们刚刚解决了单源最短路问题。今要求出图中任意两点之间的距离，我们可以如何 DP?

第一时间想到 `dp[u][v]` 表示从 u 到 v 的距离。但由于循环依赖问题，没有合适的转移。

于是考虑记 `dp[k][u][v]` 表示从 u 开始经历不超过 k 个点到达 v 的最短路。显然有

$$dp[k][u][v] = \min_{\texttt{edge}: x \to v} \{dp[k-1][u][x] + dis(x \to v)\}$$

注意到是分层的。滚动数组滚掉第一维，我们得到 Floyd 算法。

乘积最大

https://www.luogu.com.cn/problem/P1018

乘积最大

https://www.luogu.com.cn/problem/P1018

令 $dp[pos][d]$ 表示把数字串的 $[0,pos)$ 位划分成 d 段，得到的收益。则转移是显然的。

$$dp[pos][d]=\max_{x<pos}\{dp[x][d-1]*a_{[x,pos)}\}$$

技巧：处理序列问题时，往往考虑将此序列的前缀作为子问题。

技巧：处理字符串时采用左闭右开区间，常常可以省代码。

Multiplicity

https://codeforces.com/problemset/problem/1061/C

Multiplicity

https://codeforces.com/problemset/problem/1061/C

设计状态：`dp[x][y]` 表示从序列 $a_{1\dots x}$ 中选取 y 个元素组成「好序列」的方案数。

显然

$$dp[x][y] = \begin{cases} dp[x-1][y] + dp[x-1][y-1], & y \mid a[x] \\ dp[x-1][y] & otherwise \end{cases}$$

注意到 n 是 1e5，a_i 是 1e6，直接转移的话空间和时间都是要木大的。

Multiplicity

所以我们考虑优化。注意到状态图是分层的，所以空间可以滚动数组。

于是代码框架是：外层枚举 x，内层枚举 y 进行转移。

现在来优化时间，考虑减少内层循环次数。我们知道，只有当 $y \mid a[x]$ 的那些 y 才需要特殊考虑，于是因数分解 $a[x]$，耗时 $\mathcal{O}(\sqrt{n})$，总复杂度 $\mathcal{O}(n\sqrt{n})$。

Multiplicity

所以我们考虑优化。注意到状态图是分层的，所以空间可以滚动数组。
于是代码框架是：外层枚举 x，内层枚举 y 进行转移。
现在来优化时间，考虑减少内层循环次数。我们知道，只有当 $y \mid a[x]$ 的那些 y 才需要特殊考虑，于是因数分解 $a[x]$，耗时 $\mathcal{O}(\sqrt{n})$，总复杂度 $\mathcal{O}(n\sqrt{n})$。

顺便提一句密码学常识。n 的因数个数你可以期望是 $\ln n$ 的级别，但你分解掉 n 需要付出 $\mathcal{O}(\sqrt{n})$ 的复杂度。目前质因数分解仍然没有找到多项式级别的算法，所以大量密码体系的安全建立在「因数分解的困难性」上。

方格取数

https://www.luogu.com.cn/problem/P1004

方格取数

https://www.luogu.com.cn/problem/P1004

显然两个人「同时走」和「先后走」是一样的结果，只要保证每个数只被取一次。

设计 $dp[step][a][b][x][y]$ 表示「两个人各走 $step$ 步，第一个人走到 (a,b)，第二个人走到 (x,y)，能获取的最大价值」。

$$dp[step][a][b][x][y] = benifit + \max\begin{cases} dp[step-1][a-1][b][x-1][y] \\ dp[step-1][a-1][b][x][y-1] \\ dp[step-1][a][b-1][x-1][y] \\ dp[step-1][a][b-1][x][y-1] \end{cases}$$

发现 $step$ 这层可以滚掉。不难写出代码。

方格取数

有个提醒。本题的题解中，大部分直接说「设计 $dp[x][y][a][b]$ 表示第一个人走到 (x, y)，第二个人走到 (a, b) 的最优解」，这个是没道理的。

我们考虑了「两个人一起走」，这样就不会出现一个人预先走路把另一个人的数抢了的情况，也就解决了后效性。然后通过滚动数组把五维变成四维。
不过事实上，按他们的搞法，完全不考虑这个后效性（也就是把我们代码中的那个 `if` 删去），也能把这题通过。咋过的我就不知道了。

Queries for Number of Palindromes

https://codeforces.com/problemset/problem/245/H

Queries for Number of Palindromes

https://codeforces.com/problemset/problem/245/H

题意就是要统计 $s_{l\dots r}$ 范围内的回文子串个数。假设我们记答案为 $dp[l][r]$，由容斥原理不难有

$$dp[l][r] = dp[l][r-1] + dp[l+1][r] - dp[l+1][r-1] + \texttt{is\_pal}(s_{l\dots r})$$

其中 `is_pal[][]` 可以 $\mathcal{O}(|s|^2)$ 预处理，故整个问题是 $\mathcal{O}(|s|^2)$ 解决的。

Queries for Number of Palindromes

https://codeforces.com/problemset/problem/245/H

题意就是要统计 $s_{l\dots r}$ 范围内的回文子串个数。假设我们记答案为 $dp[l][r]$，由容斥原理不难有

$$dp[l][r] = dp[l][r-1] + dp[l+1][r] - dp[l+1][r-1] + \texttt{is\_pal}(s_{l\dots r})$$

其中 `is_pal[][]` 可以 $\mathcal{O}(|s|^2)$ 预处理，故整个问题是 $\mathcal{O}(|s|^2)$ 解决的。

事实上做这题不需要以 DP 的视角来考虑。本质上此题是在回答 `is_pal` 矩阵的子矩阵之和，一个二维前缀和就做完了。

密令

https://www.luogu.com.cn/problem/P1385

密令

https://www.luogu.com.cn/problem/P1385

首先就要发现一条性质：我们设字符串的总和是 sum，则任何字母总和为 sum 的等长的串都是可以达到的。

证明：采用构造方法。我们从前往后构造新的字符串，考虑第一个字符，如果比目标大就用 $(-1,+1)$ 变换，比目标小就用 $(+1,-1)$ 变换。然后去搞第二个字符……以此类推，直到搞完前 $n-1$ 位。

至于最后一位，我们断言它此时必定等于目标串的最后一位。这是因为两种变换均不会改变字母和 sum。

于是整个问题被简化为：给定 sum，有多少种长度为 n 的序列满足：

- 每个元素在 $[1,26]$ 之间
- 序列和为 sum

密令

于是开始 DP。设 $dp[k][x]$ 表示长度为 k 的序列之和为 x 的方案数，答案显然是 $dp[n][sum]$。转移是显然的。

$$dp[k][x] = \sum_{1 \le i \le \min\{26, x\}} dp[k-1][x-i]$$

密令

于是开始 DP。设 $dp[k][x]$ 表示长度为 k 的序列之和为 x 的方案数，答案显然是 $dp[n][sum]$。转移是显然的。

$$dp[k][x] = \sum_{1 \le i \le \min\{26, x\}} dp[k-1][x-i]$$

卡一下常数。

Ahmad and Spells

https://codeforces.com/gym/101502/problem/C

Ahmad and Spells

https://codeforces.com/gym/101502/problem/C

题目大意：Ahmad 想要放 n 次技能，初始情况下每个技能要 x 的时间来放。他可以买天赋，第 i 个天赋价格是 b_i，可以把放技能的时间缩短为 a_i。还可以雇凶，第 i 个帮手的价格是 d_i，可以直接帮他直接放出 c_i 次技能。

Ahmad 顶多学习一个天赋，顶多雇佣一个帮手。Ahmad 手上的金币有限，希望求出放完技能的最短时间。

Ahmad and Spells

这里有两个参数待优化（买哪个天赋、买哪个帮手）。
我们可以直接枚举买哪个天赋，求出该情况下的最优解。最后统计一下全局最优解。

Ahmad and Spells

这里有两个参数待优化（买哪个天赋、买哪个帮手）。
我们可以直接枚举买哪个天赋，求出该情况下的最优解。最后统计一下全局最优解。

假设我决定了买某个天赋，接下来该如何选择买哪个帮手？显然是价格承受范围内干活最多的。
这里可以写一个数据结构来维护，但事实上我们不需要这样做。

Ahmad and Spells

这里有两个参数待优化（买哪个天赋、买哪个帮手）。
我们可以直接枚举买哪个天赋，求出该情况下的最优解。最后统计一下全局最优解。

假设我决定了买某个天赋，接下来该如何选择买哪个帮手？显然是价格承受范围内干活最多的。
这里可以写一个数据结构来维护，但事实上我们不需要这样做。
首先把帮手按价格排序。剔除掉所有「存在另一个人既比他便宜，干的活又更多」的帮手。于是剩下的人里面严格满足「越贵越好」，可以直接贪心。

宝物筛选

https://www.luogu.com.cn/problem/P1776

宝物筛选

https://www.luogu.com.cn/problem/P1776

没啥说的，二进制分组优化背包。

弹珠

https://www.luogu.com.cn/problem/P1537

弹珠

https://www.luogu.com.cn/problem/P1537

首先我们考虑每一颗弹珠。记 `bool dp[k][v]` 表示使用前 k 颗弹珠能否恰好凑出 v 的价值，则转移是显然的。

$$dp[k][v] = dp[k-1][v] || dp[k-1][v - value_k]$$

分层，可以滚掉，所以空间没问题。但弹珠总数是 20000，最大价值可能是 120000，造成超时。

弹珠

https://www.luogu.com.cn/problem/P1537

首先我们考虑每一颗弹珠。记 `bool dp[k][v]` 表示使用前 k 颗弹珠能否恰好凑出 v 的价值，则转移是显然的。

$$dp[k][v] = dp[k-1][v] || dp[k-1][v - value_k]$$

分层，可以滚掉，所以空间没问题。但弹珠总数是 20000，最大价值可能是 120000，造成超时。

注意到可以使用二进制分组的思想来优化。

Boxes Game

https://codeforces.com/gym/101502/problem/J

博弈论。今有一个序列 w，Alice 和 Bob 轮流从两端取数。取到的数加进自己分数，目标是使得分数最大化。问游戏结束后分数之差。

Boxes Game

https://codeforces.com/gym/101502/problem/J

博弈论。今有一个序列 w，Alice 和 Bob 轮流从两端取数。取到的数加进自己分数，目标是使得分数最大化。问游戏结束后分数之差。

取数是从原序列两端取，故游戏的每一时刻的序列状态，都是 w 的一个区间。

记 $A[l][r]$ 为对子区间 $w_{l...r}$ 进行游戏，先手取得的分数，$B[l][r]$ 为后手取得的分数。转移是显然的：

$$A[l][r] = \max\{w[l] + B[l+1][r], w[r] + B[l][r-1]\}$$

又 $B[l][r] = \sum w_{l...r} - A[l][r]$，记忆化搜索即可。

释放囚犯

https://www.luogu.com.cn/problem/P1622

释放囚犯

https://www.luogu.com.cn/problem/P1622

典型区间 DP。首先，把待释放的囚犯按编号从小到大排序为数组 s。

我们考虑 $dp[l][r]$ 表示释放 $s_l ... s_r$ 这些目标的代价。注意这里的 s_i 是待释放人员的编号而不是牢房编号。

假设我们在考虑 $dp[l][r]$ 的时候，选择首先释放 s_k 这个囚犯。那么当天会有哪些人乱叫：显然是 $[s_{l-1}, s_k), (s_k, s_{r+1}]$ 这群人。人数一共是 $s_{r+1} - s_{l-1} - 2$ 个。

$$dp[l][r] = \min_k \{dp[l][k-1] + dp[k+1][r] + s[r+1] - s[l-1] - 2\}$$

Caesar's Legions

https://codeforces.com/contest/118/problem/D

Caesar's Legions

https://codeforces.com/contest/118/problem/D

我们记 $dp[a][b][x][y]$ 表示队伍里面一共有 a 个步兵、b 个骑兵、以 x 个步兵结尾、以 y 个骑兵结尾。当然这里面 x, y 之间必然有一个是 0，会浪费一点空间，不过这题空间是足够的。

转移显然。要么往尾部添加一个步兵，要么往尾部添加一个骑兵。

数列

https://www.luogu.com.cn/problem/P1799

数列

https://www.luogu.com.cn/problem/P1799

设计状态 $dp[k][x]$ 表示从序列的前 k 个元素里面选择 x 个，得到的最大收益。

显然

$$dp[k][x] = \max\{dp[k-1][x], dp[k-1][x-1] + (a[k] == x)\}$$

False Mirrors

https://vjudge.net/problem/URAL-1152

False Mirrors

https://vjudge.net/problem/URAL-1152

首先，我们可以通过一个 01 串来唯一地描述怪物巢穴有没有被扬掉。0 表示还留着，1 表示扬掉了。一个长度为 n 的 01 串对应着一个状态。

考虑这样的 DP 方式：令 dp[state] 表示「从 state 状态开始，打掉所有巢穴所需要的代价」。显然答案是 dp[000...0]，考虑如何转移。

显然，从 state 状态开始解决问题的代价，取决于 state 的后序状态代价 + 这一步的代价。

这题的奇妙之处在于：尽管时序上 00000 在 11100 之前，但是求解答案时，00000 反而依赖 11100 的结果。

False Mirrors

至于代码上如何实现。一个长度为 20 的 01 串，可以用 int 来表示。
通过一些位运算技巧，可以写出优雅的代码。

这类问题有一个比较通用的代码方法：预处理出一个掩码数组，然后对掩码做运算，而不是一个一个位去手动运算。

Buns

https://codeforces.com/contest/106/problem/C

Buns

https://codeforces.com/contest/106/problem/C

设计状态：$dp[a][b][c][d][e]\ldots$ 表示还剩 a 克甲包子馅、b 克乙包子馅……

Buns

https://codeforces.com/contest/106/problem/C

设计状态：$dp[a][b][c][d][e]\ldots$ 表示还剩 a 克甲包子馅、b 克乙包子馅……

开玩笑的。显然不行辣。DP 的优化里面，有一个技巧叫做「简化状态」。当你的状态数量变少，不仅空间更容易满足需求，时间也可能会降低。

我们注意到，做包子的先后顺序对最后卖出多少钱毫无影响。那我们就指定先做甲包子、再做乙包子……以此类推。

Buns

`https://codeforces.com/contest/106/problem/C`

设计状态：$dp[a][b][c][d][e]...$ 表示还剩 a 克甲包子馅、b 克乙包子馅……

开玩笑的。显然不行辣。DP 的优化里面，有一个技巧叫做「简化状态」。当你的状态数量变少，不仅空间更容易满足需求，时间也可能会降低。

我们注意到，做包子的先后顺序对最后卖出多少钱毫无影响。那我们就指定先做甲包子、再做乙包子……以此类推。

这就带给我们一个性质：甲包子一旦做完，甲馅料还剩多少就不影响后序决策了。因此可以在状态里面直接删掉馅料余量。

最终的状态设计：`dp[k][r]` 表示只做前 k 种包子，使用了 r 克面团，获得的收益。

田忌赛马

https://www.luogu.com.cn/problem/solution/P1650

01 分数规划

严格来讲 01 分数规划的求解方法一般不是 DP。

不过鉴于 01 分数规划也是一个最优化问题，所以在这节课上讲一讲。

缓考策略

众所周知，哈工大的「平均学分绩」是对课程成绩的加权平均数，权重即为学分。具体而言：

$$\frac{\sum s_i \times w_i}{\sum w_i}$$

阮某某马上就要期末考试了，他知道自己每科会考多少分。
他可以缓考至多 k 个科目，这样这些科目就暂且不计入学分绩，让本学期的平均学分绩好看一点。
问其本学期平均学分绩最高能有多少。

01 分数规划

01 分数规划的数学表述如下：求一个布尔数组 x，最大化

$$\frac{\sum a_i \times x_i}{\sum b_i \times x_i}$$

显然向量 x 的意义是「有没有选择某项」。

在刚刚的学分绩问题中，$b_i = w_i, a_i = s_i \times w_i$

二分方法

首先，我们面临一个最优化问题。通过二分答案，可以把最优化问题转变为可行性问题。

那么我们来看一看 01 分数规划如何用二分答案来做。我们现在的任务是：判断是否存在向量 x 满足

$$\frac{\sum a_i \times x_i}{\sum b_i \times x_i} > mid$$

等价变换这个式子，得到

$$\sum x_i \times (a_i - mid * b_i) > 0$$

注意到 $a_i - mid * b_i$ 是常量，故立即可以解决。注意向量 x 需要满足题目的约束。

Dropping Tests

https://vjudge.net/problem/POJ-2976