

# 搜索、分治与贪心算法



# 搜索

Search

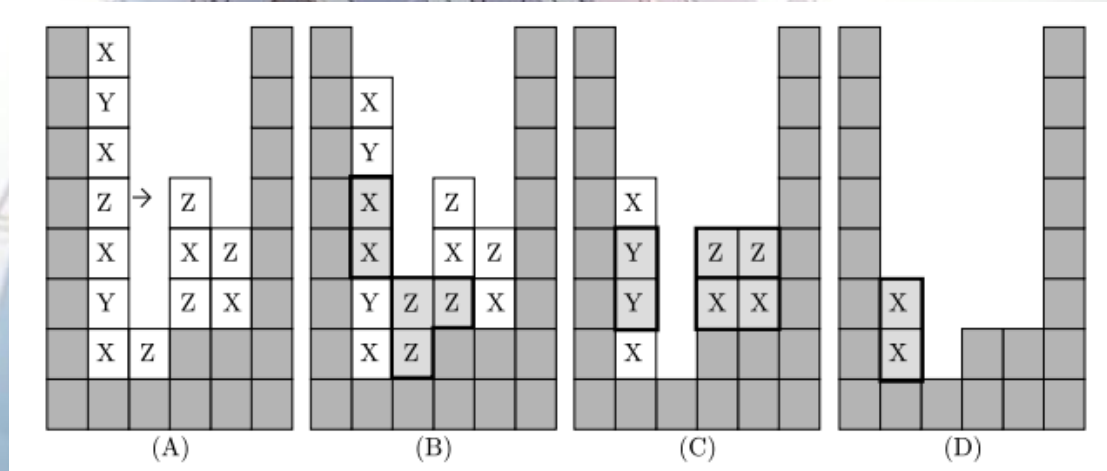
# 搜索入门

- 所谓入门，就是一些基础的东西，比如深度优先搜索(DFS),广度优先搜索(BFS),记忆化搜索,暴力枚举之类的，也就是我们通称的暴力。
- 这是最重要的基础，相信大家都会了，这里就不再赘述。
- 例题：
- 马的走法，倒水问题，八皇后问题，etc...



# A Vexing Problem

- 给一个消除游戏,  $R \times C$  的图中有一些障碍物和石块。每个石块有一个标记。障碍不受重力影响, 石块受影响(即悬空时会坠落)。
- 每次可以左右移动(没有障碍或石块)一格。两个以上相同且相邻的石块连成的一片会被消除, 存在连消。
- 给出初始局面, 求消完所有石块的最小步数。
- Data Constraint
- $4 \leq R, C \leq 9$ . 保证解不超过11。
- Time Limit : 15s



# A Vexing Problem

- 时限居然有15s！
- 所以,这题怎么做都没太大问题。
- 直接记忆化搜索BFS,用Hash来储存状态,每次扩展的时候暴力处理一下坠落和消除的情况。
- 怎么用Hash记录状态？

- 
- 这样就完了？
  - 解法II：数据保证最优解不大于11,这启发我们用迭代加深。在后面的内容我们会讲迭代加深。



# ABCDEF

- 给出一个整数集合S,求有多少组(a,b,c,d,e,f),满足 $(ab + c) / d - e = f$
- Data Constraint
- $|S| \leq 100$

# ABCDEF

- 移项得:  $ab + c = d(e + f)$
- 得到了和前面类似的形式,枚举左边的数,把算出来的答案存在一个散列表里,再枚举右边,判断是否存在。
- 时间复杂度:  $O(|S|^3)$



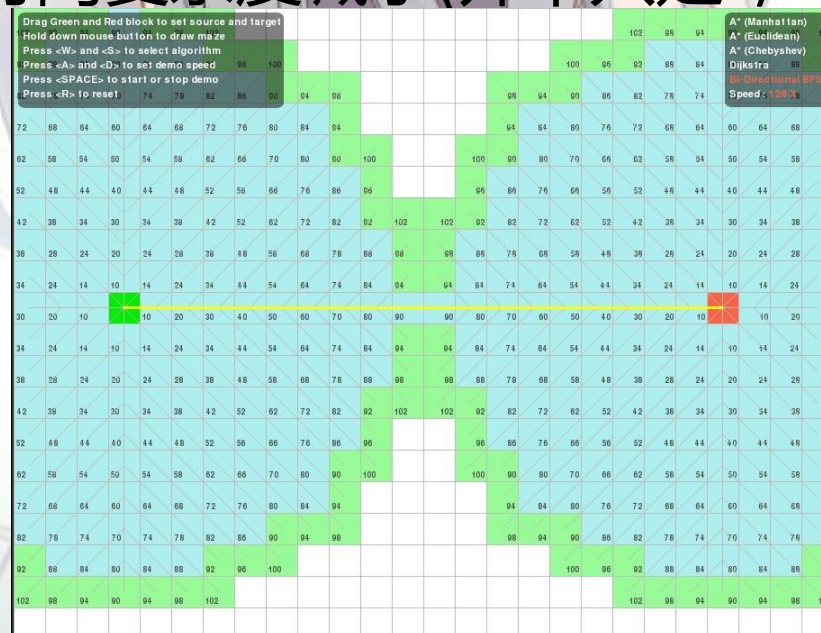
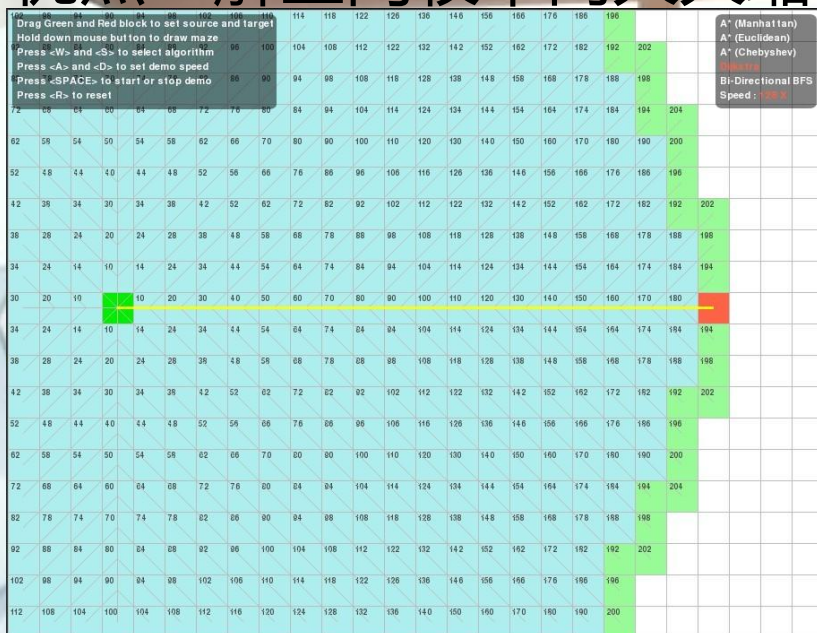


# 双向BFS

Double Breadth-First Search

# 双向BFS

- 双向BFS,简单来说就是同时从起始状态和目标状态开始扩展(交替搜索)。
- 优点：解空间较单向大大缩小(如图),时间复杂度减小(并不只是1/2!)



- 适用范围：已知起始状态和目标状态以及操作满足可逆性。



# 彩球游戏

- $N \times M$ 的方格中有红(R)绿(G)蓝(B)三种颜色的球,每次选择 $2 \times 2$ 四个球进行两种操作:
- 1)四个小球顺时针旋转
- 2)颜色按照以下规则进行变换:
- $R \rightarrow B ; B \rightarrow G ; G \rightarrow R .$
- 求从初始状态到目标状态的最少操作次数。
- Data Constraint
- $2 \leq N , M \leq 4 , N \times M \leq 16$



# 彩球游戏

- 这是一道简单题，相信大家都会了。
- 显然可以双向BFS
- 顺时针旋转的逆操作逆时针旋转。颜色变换逆操作就直接按照规则反过来变换。
- 然后从起始状态和目标状态出发，每次将状态较少的那边拓展一层，然后用一个hash表来记录状态(可以把状态转化为一个整数)，如果在hash表中找到一个出现过的状态，那么当前答案就是最优解。



# 中途相遇法

Meet in The Middle

# 中途相遇思想

- 假如现在有两人甲、乙分别从A,B两地出发,相向而行。当他们都走了 $L/2$ 米相遇时,我们就可以说找到了一条长度为 $L$ 的路径
- 中途相遇法是一种常用的搜索优化技巧。其实质类似双向搜索。





# 中途相遇法在有向图中的应用

The Application of Meet in The Middle in Directed Graph

# 不同路径数

- 求有向图G中点A到点B长度为L的不同路径数目。
- Data Constraint
- 设点数为N,每个点最大度数为D。
- $N \leq 50000$
- $D \leq 3$
- $L \leq 30$

# 不同路径数

- 显然,直接DFS时间复杂度是 $O(D^L)$
- 建立反向图,设  $L_1 = \left\lfloor \frac{L}{2} \right\rfloor$ ,  $L_2 = \left\lceil \frac{L}{2} \right\rceil$  将算法分为三部分:
  - 1.从A出发,沿原图搜索 $L_1$ 层,将第 $L_1$ 层到达点P的次数记为 $\text{Count}(P)$
  - 2.从B出发,沿新图搜索 $L_2$ 层,将第 $L_2$ 层到达点Q的次数记为 $\text{Count}'(Q)$
  - 3.统计答案:枚举一个中间点X,对答案贡献为 $\text{Count}(X) \times \text{Count}'(X)$ 。
- 时间复杂度:  $O(D^{L/2})$



# 方程的解数

- 给出整数M和 $k_1, k_2, \dots, k_N$ ,  $p_1, p_2, \dots, p_N$ .

$$\sum_{i=1}^N k_i x_i^{p_i} = 0$$

- 求满足 $1 \leq x_i \leq M$  的整数解的个数。

- Data Constraint
- $N \leq 10$ ,  $M \leq 14$ ,  $\text{abs}(k_i) \leq 10$ ,  $1 \leq p_i \leq 5$

# 方程的解数

- 我们可以把方程模型转化为有向图模型。
- 研究N=6的情况:

$$k_1x_1^{p_1} + k_2x_2^{p_2} + k_3x_3^{p_3} + k_4x_4^{p_4} + k_5x_5^{p_5} + k_6x_6^{p_6} = 0$$

- 移项得:

$$k_1x_1^{p_1} + k_2x_2^{p_2} + k_3x_3^{p_3} = -k_4x_4^{p_4} - k_5x_5^{p_5} - k_6x_6^{p_6}$$

- 暴力搜出左边的所有情况,存在一个数组里。搜右边的时候,每次查找当前这个和是否出现过,若出现过,则说明找到了一组解。
- 时间复杂度:  $O(M^{N/2})$



# 迭代加深

Iterative Deepening



# 埃及分数问题

- 在古埃及，人们使用单位分数的和表示一切有理数。
- 例如 $2/3 = 1/2 + 1/6$ ，但不允许 $2/3 = 1/3 + 1/3$ ，即加数不能相同。
- 对于一个分数 $a/b$ ，求加数最少的表示方法，加数相同多的，则最小分数越大越好。
- Data Constraint
- $0 < a < b < 500$

# 埃及分数问题

- 埃及分数问题是迭代加深的一个经典问题。
- 如果直接DFS，每一层的状态数量会非常庞大！
- 如果我们枚举了当前搜索的最大深度，那么问题就变得简单了。
- 设当前枚举的深度为 $\text{maxd}$ ，我们每次只搜索到深度不大于 $\text{maxd}$ 的节点。
- 同时我们可以利用这一性质来剪枝(IDA\*)，大大减少了无用的状态数。
- 大部分迭代加深题目都会利用当前枚举的最大深度来剪枝。



# 具体实现

- 我们用dep限制搜索层数，先从2开始，每次深度+1
- 搜索时每一层比上一层的分数小，即分母一次比一次大
- 每次枚举出  $1/a$  后，用当前分数减去，然后递归传递剩余的分数
- 每层搜索枚举的限制条件：
  - 1、保证当前深度分母大于上一深度分母
  - 2、枚举的  $1/a$  小于当前分数，不可能存在等于的状态，因为此种最优解会在限制深度较小的时候出现
  - 3、设当前剩余分数为  $x/y$ ，剩余深度为  $d$ ，则  $x/y < d/a \rightarrow a < d/x * y$ 。
- 不妨先设之后枚举的分母都为  $a$ ，那么最后也就刚好达到  $x/y$ ，但又因分数不能相等，所以  $a$  必须小于该值，即把分数调大。如果分数很小，那么就永远够不着目标分数



# 搜索小结

- 总的来说,搜索的核心就是状态(和DP很像),如何来表示状态,如何扩展状态,如何优化搜索过程都从很大程度上影响了搜索的效率。
- 大部分时候,我们可以这样来优化算法:
  - 1.加一些剪枝(最优性剪枝,连通性剪枝,Alpha-Beta,break/return...)
  - 2.用一些小技巧(枚举二进制,记忆化,结点排序...)
  - 3.杀手锏:换方法
- 如果确信搜索是正解那就大胆上前两条,一般不会被卡。
- 如果不确定,那还是乖乖换方法吧,负责人的出题人是不会让暴力水过的。



# 分治

Divide and Conquer



# 经典问题复习&热身



# 棋盘覆盖问题

- 有一个 $2^k \times 2^k$  的棋盘，恰有一个方格是黑的，其他的为白色。
- 注意黑色方格不能被覆盖，任意一个白色方格不能同时被两个或更多牌覆盖。
- 现在要用3个方格的L形牌覆盖所有白色方格。牌可以旋转对称。求一个覆盖方案。

- 这样就把k规模的问题分成了4个k-1规模的问题

- 递归处理直到k=1为止。

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1																
2																
3																
4																
5																
6						■										
7																■
8								■	■							
9								■	■							
10																
11																
12	■															
13																
14																
15																
16																

[https://blog.csdn.net/dq\\_30263545](https://blog.csdn.net/dq_30263545)

# 循环日程表问题

- $n=2^k$ 个运动员进行循环赛，每个选手必须和其他 $n-1$ 个运动员各赛一次，且每个选手一天只能赛一场。
- 现在要排一个日程表，第 $i$ 行第 $j$ 列是第 $i$ 个选手第 $j$ 天遇到的选手。

1↺	2	3	4↺	5	6	7	8↺
2↺	1	4	3↺	6	5	8	7↺
3↺	4	1	2↺	7	8	5	6↺
4↺	3	2	1↺	8	7	6	5↺
5↺	6	7	8↺	1	2	3	4↺
6↺	5	8	7↺	2	1	4	3↺
7↺	8	5	6↺	3	4	1	2↺
8↺	7	6	5↺	4	3	2	1↺



• 与上题差不多

# 分治

- 分治就是分而治之。
- 分治法往往有三步:
  - 1.划分问题(Divide)：把原问题划分成若干个子问题
  - 2.递归求解(Conquer)：递归解决每一个子问题
  - 3.合并答案(Combine)：按照已有条件合并问题答案,从而解决原问题。
- 本质：缩小问题规模

# 第K大

- 给出两个有序序列。
- 你需要在 $\log N$ 的时间里找出将两个序列合并之后的第K大。



# 第K大

- 先找两序列的中点，假设为 $a_i$ 和 $b_j$ ，且 $a_i \leq b_j$
- 若 $i + j \leq K$ ，则第K大的数一定不在a的前半。
- 否则一定在b的后半。
- 每次至少一个序列长度减半。
- 时间复杂度： $O(\log N)$

# Non-boring sequences

- 如果一个序列的任意连续子序列至少有一个只出现一次的元素，则称这个序列是不无聊(Non-boring)的。
- 给一个长度为N的序列A，现要判断它是否是不无聊的。
- Data Constraint
- $N \leq 200000$
- $A[i] \leq 100000$

# Non-boring sequences

- 对于当前区间 $[L, R]$ ，记只出现一次的元素为 $A[P]$ 。
- 那么当前这个区间必定是不无聊的，所以只需要判断 $A[1...P-1]$ 和 $A[P+1...N]$ 是否满足条件即可。
- 可以开一个数组 $B[i]$ 记录当前区间数字 $i$ 的个数及位置，每次通过扫描分治的小区间来更新分治出来区间的数组 $B$ 。
- 时间复杂度:  $O(N\log N)$



# Xor

- 给出长度为 $n=2^k$ 的序列 $a$ 。
- 给出长度为 $k$ 的序列 $b$ 。
- 令  $c_i = \sum_{j=0}^{n-1} a_j \times b_{\text{Cnt}(i \wedge j)}$  其中 $\text{Cnt}(x)$ 表示 $x$ 二进制中1的个数， $\wedge$ 表示位运算xor。
- 求序列 $c$ 。
- Data Constraint
- $n \leq 10^5$

# Xor

- 不妨考虑按二进制位分治。
- 当前分治区间为 $[L,R]$ ，迭代的深度为 $dep$ ，其中 $[L,mid]$ 第 $dep$ 位均为0， $[mid+1,R]$ 的第 $dep$ 位均为1。
- 令 $f[i][j]$ 表示在当前分治区间内，与 $i$ 有 $j$ 位二进制不同的数之和。
- 很容易由 $[L,mid],[mid+1,R]$ 得到 $[L,R]$ 的 $f$ 值。
- 问题迎刃而解。
- 听说还可以FWT？

# Parking Lot

- 给出一个 $n \times m$ 的网格图。
- 有的位置上有障碍物。
- 给出 $k$ 个操作，每次将一个格子变为障碍物，再询问没有障碍物的最大子正方形。
- Data Constraint
- $n, m, k \leq 2000$
- Time Limit: 3s



# Parking Lot

- 将矩阵分成相等的两部分。不妨设中线为mid。
- 那么我们的答案就是， $[1, \text{mid}]$ ， $[\text{mid}, n]$ ，以及穿过中线的答案中的最大值。前两个可以分治迭代求解。
- 对于穿过中线的正方形：
- 如果没有修改，我们算出中线上的“柱状图”，再用单调队列维护。
- 有修改，当前矩阵中有k个修改，我们就重新维护k次。
- 时间复杂度:  $O(NK\log N)$

# 树中点对距离

- 给出一棵带边权的树和长度限制 $Len$ ，问有多少对点的距离 $\leq Len$
- Data Constraint
- $N \leq 10000$

# 树中点对距离

- 这是一个点剖经典模型。
- 每次找出树的重心，将树分为多个部分进行分治，再用两个指针来合并各个部分的答案。
- 时间复杂度:  $O(N \log N^2)$



# 分治小结

- 分治与其说是算法，我更偏向于认为它是一种思想。
- 一般来说，分治可能会与其他考点一起出题，比较常见的有：几何+分治；DP+分治；数据结构+分治。
- 实际上，分治的本质就是缩小问题规模。
- 关键在于如何划分问题和合并答案，这也是分治的标志。如果一个题目，问题的划分方法比较显然，同时也便于合并子问题答案时，就可以考虑分治了。
- 辅助思考的技巧：考虑中间一个点(线/区间)或者某个特殊位置对划分和合并有什么帮助。



# 贪心

Greedy

# 什么是贪心？

顾名思义，贪心算法总是作出在当前看来最好的选择。也就是说贪心算法并不从整体最优考虑，它所作出的选择只是在某种意义上的局部最优选择。当然，希望贪心算法得到的最终结果也是整体最优的。虽然贪心算法不能对所有问题都得到整体最优解，但对许多问题它能产生整体最优解。如单源最短路径问题，最小生成树问题等。在一些情况下，即使贪心算法不能得到整体最优解，其最终结果却是最优解的很好近似。



# Main Sequence

定义一个“整数括号序列”为每个正整数(Num)都要有一个与之匹配的相反数(-Num)，把正整数视为左括号，负整数视为右括号，合法的条件与括号序列相同。

现在给你N个正整数，将其中给定的M个位置取负，问能否将另一些位置的正整数取负使得这个“整数括号序列”合法，如果可以并输出这个修改后的序列。

$$N \leq 10^6 \quad M \leq N$$

## 题解

**因为某些右括号已经确定了，所以我们从后往前扫这个序列。要使序列合法，我们要求后面的右括号对前面的影响尽量小，所以要贪心的维护一下还未匹配的右括号，每次尽量匹配。**

**对于每个数有两种情况，如果它是给定的M个数之一或者不能与栈顶的右括号匹配，那么肯定要加入未匹配右括号的序列中，否则，我们就把栈顶的数退栈，表示匹配成功。**

# 刺客

一个刺客有把耐久度为 $M$ 的刀，他要杀死 $N$ 个敌人，其中杀死第 $i$ 个敌人需要消耗 $A_i$ 点耐久度，但可以得到一把能杀死 $B_i$ 个敌人的武器（不消耗自己刀的耐久度），问最多可以杀死多少个敌人，在这个前提下，最少需要消耗的耐久度。

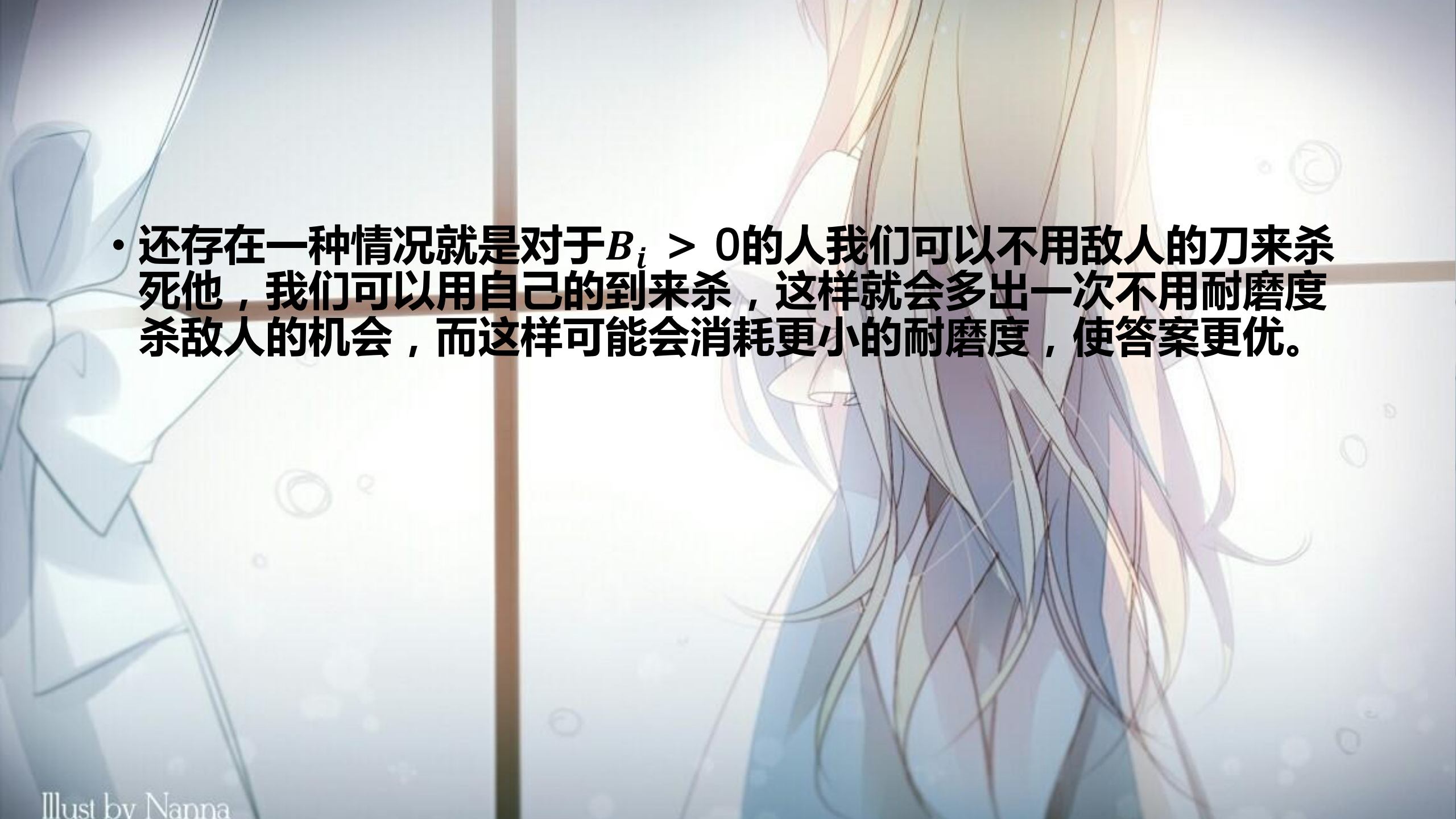
$$N \leq 10^5 \quad M \leq 10^9 \quad A_i \leq 10^9 \quad B_i \leq 10$$



## 题解

容易想到，当我手中有敌人的刀时我们肯定是有先要杀  $B_i > 0$  的人。所以我们只需找一个拥有刀的人消耗耐磨值最小的人，把所有  $B_i > 0$  的先杀掉，再用多余的敌人的刀杀死剩下的人中  $A_i$  最大的。最后剩下的尽量杀就好了。

还用一种可能是不杀  $B_i > 0$ （因为他们的  $A_i$  可能会特别大），直接杀  $B_i = 0$  的人消耗用更小的耐磨值杀相同的人。

- 
- 还存在一种情况就是对于  $B_i > 0$  的人我们可以不用敌人的刀来杀死他，我们可以用自己的到来杀，这样就会多出一次不用耐磨度杀敌人的机会，而这样可能会消耗更小的耐磨度，使答案更优。

# BZOJ 4027 HEOI2015 兔子和樱花

给定一颗 $N$ 的点的有根树，每个点上面有一些樱花，现在要求删掉一些节点，删除节点的樱花会累加到它的父亲上，儿子节点也会接到父亲上。现在要求删除完后，每个点的樱花数加儿子节点数小于 $M$ 。问最多能删多少个点。

$$N \leq 2 * 10^6$$

$$M \leq 10^5$$



## 题解

一个很容易想到的结论就是从叶子到根每个点都尽可能的合并到父亲上，即对于每个点，把儿子节点的权值（樱花数+儿子数）从小到大排序，再从小到大删最多的点。

感性的认识一下为什么这样是对的？因为假如这个节点不合并到父亲上去，那么父亲肯定要合并到祖先，我们的收益都是1。但是我们要对上面的节点影响尽量小，那肯定是合并尽量下的节点。

# DFS Spanning Tree

给出一个没有自环的有向图。这个图的前 $N-1$ 条边构成这个图的一个以节点1为根节点的DFS树。

DFS树：每条非树边都是从一个节点连向它的祖先。

T-Simple环的定义是：至多有一条边不在这棵DFS树上的环。

现在的问题是至少在图上选中多少条边。才使得每个T-simple环都至少有一条边被选中。

$N \leq 2000$

## 题解

我们先来看一个子问题，我们有一个序列，现在有若干个限制条件，即在 $L_i$ 到 $R_i$ 的这段区间内至少有一个位置被打标记。那我们至少要打几个标记。

这个问题，显然可以从左往右扫，对于每个没有标记的限制就在它区间的最右端打一个标记。



## 题解

**那么看看这题，我们发现其实就只是把序列变成了一棵树，把限制放在树上。**

**那么思路就很清晰了，我们只需按树的DFS序从下往上像序列上那样处理就可以了。**

# Shop

现在有 $N$ 个数 $A_i$ 。定义如下三种操作，每种操作有三个参数  
Ord ( 类型 ) ,  $i$  ,  $b$ 。

1. 把第 $i$ 个数变成 $b$
2. 把第 $i$ 个数加 $b$
3. 把第 $i$ 个数乘 $b$

问当给出 $M$ 个操作，最多能从中选出 $K$ 个时，最优情况下操作完后 $N$ 个数的乘积最大是多少，输出最优操作下选取操作的顺序。

$$K \leq M \leq 10^5$$

$$N \leq 10^5$$

$$b \leq 10^6$$

## 提示

如果只有第三种操作是显然我们肯定选前 $K$ 个 $b$ 值最大的操作。  
那么现在我们要考虑一下怎么将第一第二种操作都转化为第三种操作。



## 题解

**我们肯定应该贪心的先做第一种操作，然后做第二种，最后才做第三种来保证值最大。**

**我们先只讨论 $i$ 相同的一组操作。**

**对于第一种操作，只有 $b$ 最大的那个可能被选到。**

**对于第二种操作，最优时肯定是按 $b$ 值从大到小的选取。**

**对于第三种操作，最优时肯定也是按 $b$ 值从大到小的选取。**

## 题解

那么怎么把所有操作都转换成第三种？

对于第一种操作我们很容易想到只需把最大的 $b$ 值减去 $A_i$ 就可以做为新的 $b$ 值并且转化成第二种操作。

而把第二种操作转化成第三种操作我们就要根据上面提到的一个性质：对于第二种操作我们只可能从大到小选，那么这就说明了，在选择以个操作前，当前数的是是确定的，操作完后的数又是确定的，我们除一下得到的就是增加的倍数。

## 题解

现在为止我们就已经把所有操作转换成第三种操作。  
后面要做得只是把全部转换完的操作从大到小选就可以了。并且对于第二种操作我们可以保证肯定是 $b$ 值大的数先被选到。 $B$ 值大的增量肯定是大于 $b$ 值小的增量。

那么最后我们只需要把选定的操作按第一种，第二种，第三种  
的顺序输出就可以了。