2020 牛客 NOIP 赛前集训营-提高组（第一场）（题解）

T1 牛牛的方程式

签到题，主要考察对于扩展欧几里得公式的理解，但是不要求把扩欧写出来。

30pt

暴力枚举x, y, z。

100pt

扩展欧几里得算法是用于求解不定方程$ax + by = k \cdot \gcd(a, b)$的一种算法。大家在记忆扩展欧几里得公式的时候不建议简单的记成$ax + by = c$，不然关键点容易忘，一定要背这个完整版的$ax + by = k \cdot \gcd(a, b)$这个公式如何理解？

最浅层次的理解就是把它当成方程式，也就是$ax + by = c$，当c满足它是k倍的$\gcd(a, b)$的时候方程式有解，否则无解。

更深入的理解是说，如果把等号左边看成是一个二元函数$f(x, y) = ax + by$，把等号右侧看成是一个函数$g(k) = k \cdot \gcd(a, b)$。其中$a$和$b$是给定的常数（此时$\gcd(a, b)$也是常数了），$x, y, k$是因变量。

则有$f(x, y) = g(k)$。此时每给定一对任意的x, y，总是能够找到一个唯一的k使得等式成立。每给定一个任意的k，也总能找到至少一对x, y使得等式成立。

这式子是干嘛用的，做换元用的。

不是要求$ax + by + cz = d$有没有解么，看见$ax + by$，也就是$f(x, y)$直接换元成$g(k)$。大家作为中学生可能一元的换元更常见一点，不过这里应该不难理解。

问题转化成求$k \cdot \gcd(a,b) + cz = d$是否有整数解。

看到这里肯定大家都会做了，不过我们把它做到底。

把 k 当成因变量，$\gcd(a,b)$是个常数，这又是个$f(k,z)$，再次换元，这次我们换成大写的 X 吧。

问题转化成求$\gcd(a,b,c)\ X = d$是否有整数解。

所以那就让d除以$\gcd(a,b,c)$咯，除得尽就是整数，除不尽就不是整数。

（这个说法其实不够严谨，因为在换元的时候,x,y是整数是k是整数的一个充分不必要条件）

细节处理上注意特判全 0 的特殊情况，不然会导致失分。

题外话

如果本题需要求出x,y,z的具体值的话，换元一次变成$k \cdot \gcd(a,b) + cz = d$套扩欧先把$z$搞出来，然后因为同时求得了$k$，所以再用一次扩欧$ax + by = k \cdot \gcd(a,b)$这个时候已经有值了，直接代入，求的$a,b$。

反正都说道不定方程了，顺便提一下它的好兄弟，同余方程。

同余方程和不定方程在一定程度上可以相互转化。

例如$ax \equiv c(mod\ b) \Leftrightarrow ax + by = c$这个式子应该不难理解。有了这种转化的思想，就可以做一些同时使用不定方程和同余方程的小综合题。

例如给定a,b,c,p求一组整数解满足$ax + by \equiv c(mod\ p)$，这就又是另一道题了。

T2 牛牛的猜球游戏

签到+，小思维题,主要考察对前缀和的理解。

30pt

直接按题意进行模拟。

60-70pt

如果你写的很复杂，又是线段树又是矩阵乘那就是这个得分。

只用一个的话牛客评测机估计冲过去了，两个都用变成线段树维护矩阵乘肯定凉。

但是不管怎样如果 T2 上了复杂结构或者算法的话，从比赛用时的角度讲已经血亏了。

100pt

对于前缀和，如果仅将其理解成一段区间的数字求和。这个理解是远不够深入的。

实际上除了前缀和，前缀亦或和，乃至前缀积。

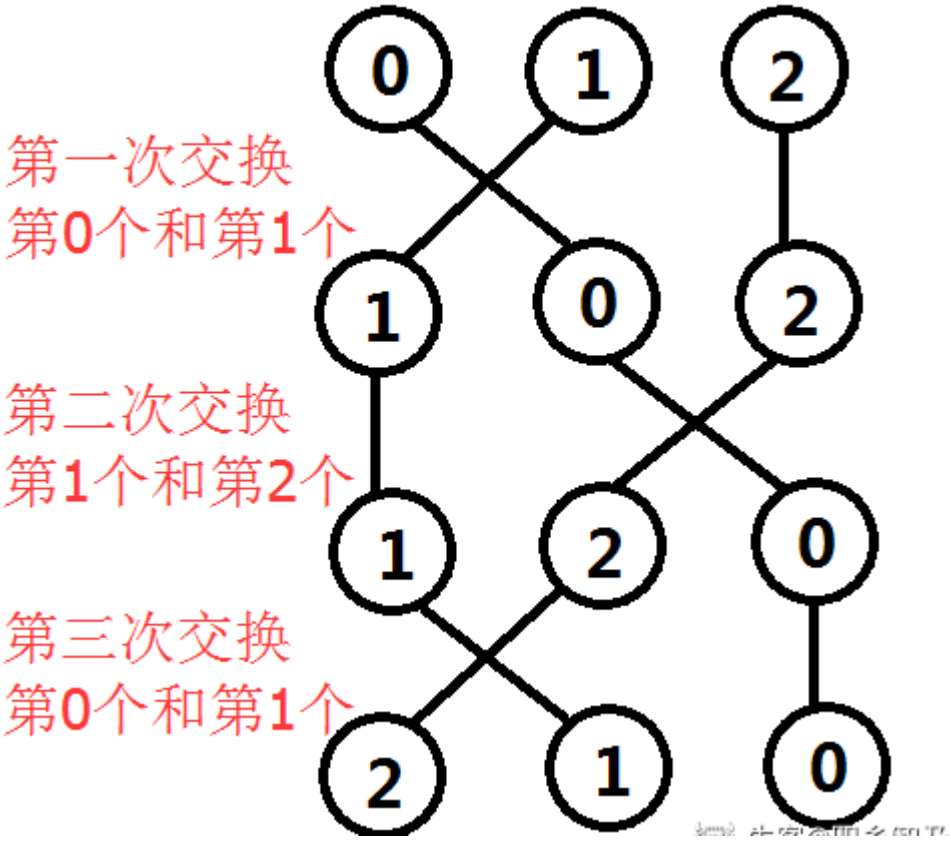
可以把前缀和算法抽象成，如果可以储存和查询某种前缀的影响，并且能够快速消除前缀影响，那么就可以知道任意区间的影响。

如果这样去理解，那么主席树就是对于修改操作影响的“前缀和”，带修的主席树不过是用树状数组维护“前缀和”罢了。

对于本题，显然可以用一个二维数组存储每一步交换的“前缀影响”。

然后想怎么去消除“前缀影响”。

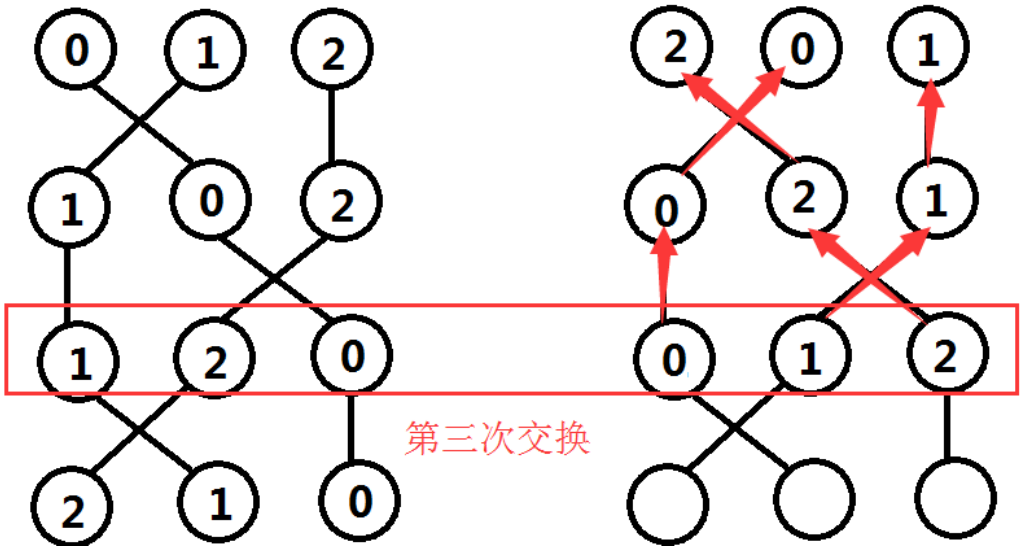
这部分画个图来讲一下，首先简化一下问题，假设只有 3 个球，进行了 3 次连续操作，交换 0,1，交换 1,2，交换 0,1。



假设现在要消除前 2 次的影响，可以在初始状态上做手脚。

也就是一开始球不是0~9按顺序排列好的，而是某种顺序。

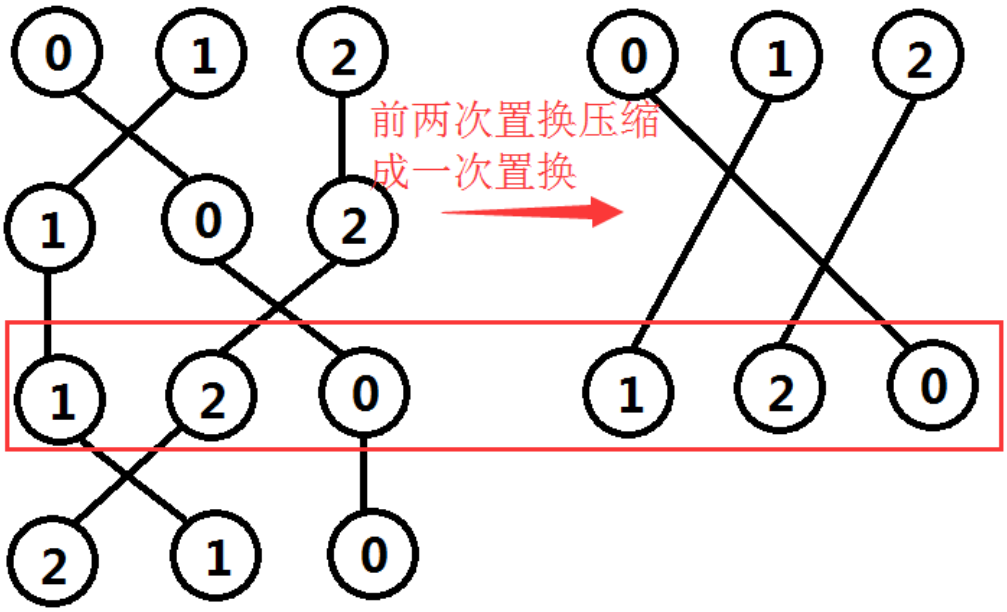
这种顺序在进行完前 2 次换球之后恰好变成了0~9按照顺序排列。



为了求出能够抵消前 2 次顺序，最暴力最笨的方法是倒着爬回去。

那么不管怎么，现在起码有了$O(n^2)$预处理，$O(1)$查询的算法了。

然后继续优化，我们想，其实没必要每次都一步一步爬，可以用路径压缩的思路，只保留从初始状态到当前操作时球被换到哪里就可以了。



根据这个“从初始状态到当前操作时球被换到哪里”的定义，发现这不就是前缀和么。

在算法实现上分为两步

1、先求出能够抵消前$l-1$次操作影响的初始序列

2、在该初始序列下进行前r次换球操作的前缀影响。

std 进行了一些封装和重载，这样算法的前缀和部分看的更清楚。

T3 牛牛的凑数游戏

中等题，思维+数据结构

从 T3 开始上难度了

本题的原题灵感是我早期在牛客出的这个题，本题是原题的加强版。所以大家先

把原题做一遍。

little w and Exchange：https://ac.nowcoder.com/acm/contest/297/D

题解：https://ac.nowcoder.com/discuss/150260

10pt

瞎搞，二进制枚举，dfs 都可以。

20pt

根据原题中的方法，每次把这一段子数组拿出来 sort。

另 10pt（2 的幂）

因为输入的数字保证是2的幂，相当于输入的种类数最多 30 种，所以用 30 种前缀和储存。

坑点是可能有人会想当然，觉得只要有 1，有 2，有 4...连续不断掉就行了，实际上可以没有 4，但是有 4 个1这样。

另 10pt（递增）

这个点没什么用，如果能想到的话基本就写出正解了。这个测试点是给选手提示信息的。

因为数字单调递增，想到可以在数组中二分查找当前的sum，然后每一次都将当前查找到的位置求和更新sum，直到不能更新为止。

实际上加个树状数组就不依赖单调性了，而树状数组又是个比较好想到的结构。

100pt

根据原题，如果想要知道若干个数组成集合的最小不能表示数，需要维护一个sum，然后从小到大遍历过去把小于等于$sum+1$的数字加进去。如果中途出现某个数字大于了$sum+1$，那么$sum+1$就是当前集合的最小不能表示数。

我们想，如果现在不进行排序，我还是扫描整个区间维护sum，这样求出来的答案虽然不对，但是它也确实是一个答案（偏小）。

有一种设计算法的思路叫做迭代法，什么叫迭代呢，就是说我在执行算法的过程中一次无法计算出正确答案，但是通过不断的重复相同的过程，每次都当前的答案作为边界参数进行下一轮迭代，每一次计算的答案都更加逼近正确的答案，最终答案稳定下来时，就得到了正确答案。

迭代算法比较典型的有，二分，三分法，模拟退火。

假设利用迭代的思路去求区间的最小不能表示数，如何设计算法？

很简单，一开始令$sum=0$，然后进行一轮扫描，把比$sum+1$小的数字加进来更新sum。每当扫描一轮后保留sum的值进行下一轮扫描，当然数字不会重复被加进来。

直到某一轮扫描后 sum 的值不产生变化，答案稳定下来，说明当前的 sum 就是所求的答案。

现在提问，假设区间长度为 n 这个暴力迭代扫描的算法，其时间复杂度是多少？

A：$O(n^2)$，B：$O(nlogn)$，C：$O(nlog^{2n})$，D：$n\sqrt{n}$。

答案是B，$O(nlogn)$，看上去有点反直觉，不过你可以自己出一些数据用代码跑跑看。

为什么呢，因为我们想一下如果上一轮扫描时的sum叫做pre，那么一个数字必

须比pre大，比$sum+1$小，它才是这一轮新增的数字。

这个原因，直接导致sum这个变量要么迭代后值不产生变化，要么在迭代后数值至少翻倍。

显然，在进行log次翻倍后，所有的数字都可以取，而一旦把所有数字全部取到，sum的值就不再发生任何变化。

迭代的过程自带log，不需要优化，所以优化在哪里呢，优化在统计小于等于$sum+1$的数字的和就可以了。

看到这里已经可以主席树解了，不过显然这个问题用树状数组更好写。

我们离线进行若干轮迭代（迭代的终止条件是所有查询的sum均不再更新）

首先将所有查询的sum值记录为 0，pre（上一轮的sum）记录为-1。

然后用树状数组从左到右将数字插入到树状数组中，对于每个查询在$l-1$的位置记录小于等于$sum+1$的数字之和记为 b，在 r 的位置记录小于等于$sum+1$的数字之和记为a。

然后更新该查询的答案为$a-b$，一旦某轮更新中发现当前轮的查询答案和上一轮相同。说明该查询已经得到了稳定的答案，不在参与下一轮迭代。

直到所有查询的答案都稳定下来，$break$结束算法并输出所有询问的答案。

T4 牛牛的 RPG 游戏

这个难度在 noip 里面就算是难题了，实际上我认为每年的 noip 两天 6 道题里面总有那么一个 T3 是用来拉高比赛上限的。

这种题不会就算了，前面全部打满，最后再暴力扣点也是一等。

这个题的灵感其实还是自己以前出过的题魔改

自己家学校的校赛题：https://ac.nowcoder.com/acm/contest/303/L

但是这个题不用看，因为是道n^2暴力转移的水题。

这道题数据范围扩大到10^5就是本题中的$min(n,m)=1$的子问题。

出题的时候一直纠结这个题是不是对于 NOIP 太难了，但是转念一想 18 年不是还考了ddp，虽然他 std 不是这么写的，但是我也没上树套树(不是)。

20pt

暴力转移，dp转移方程如下。

$dp[i][j]=max(dp[p][q]+val[i][j]+k[p][q]*(i+j-p-q),dp[i][j]$;

40pt

裸的动态凸包/李超树。

60pt

在 40pt 的基础上合并两层的结果即可。

100pt

要想做这个题的话首先需要 CDQ 分治，这个全都写到题解里面写不下，首先移步我的博客去看一下。

https://blog.nowcoder.net/n/f44d4aada5a24f619442dd6ddffa7320

重点看一下这两小节

分治求最长上升子序列

分治求动态凸包

然后你会发现 std 就是把这两小节的代码拼起来就没了。

std 用了 CDQ 分治套 CDQ 分治，原因是出 noip 题目 std 想尽量避免复杂数据结构。

内层分治实际上是在处理动态凸包，有一定能力的选手可以用李超树代替。

在讲算法的时候我只讲外层分治，内层是为了处理动态凸包，这个在上边的博客里面提到了。

那么 std 究竟在分治啥，对什么进行分治?

实际上我是对状态转移进行了分治。

根据题意，对于棋盘上面的两个位置, pi(xi,yi), pj(xj,yj)，要想存在状态转移 pi(xi,yi)→pj(xj,yj)就必须满足条件$xi \le xj$，$y_i \le y_j$。

这个地方如果借助线段树，必然会写成一个树套 X 的结构。要说不能写也不是，就是难受。

对于 CDQ 分治有一定了解的同学想必可以认识到，数据结构在这种问题的处理上的思想是数据的存储和维护。而 CDQ 分治的思想是分类和降维。

我现在把 std 中 cdqDivAlgorithmForLIS 这个函数调试用的 cout 打开，随便输入一个 2*3 的样例。带着大家跑一下，看它是怎么“分类”和“降维”的。

T4std:https://ac.nowcoder.com/acm/contest/viewsubmission?submissionId=45244250

input:

2 3

1 1 1

1 1 1

1 1 1

1 1 1

output:

CDQForLIS: deep2 --------- begin

    insert: 1-1

    query: 1-2

CDQForLIS: deep2 --------- end

CDQForLIS: deep1 --------- begin

    insert: 1-1

    insert: 1-2

    query: 1-3

CDQForLIS: deep1 --------- end

CDQForLIS: deep0 --------- begin

    insert: 1-1

    query: 2-1

    insert: 1-2

    query: 2-2

    insert: 1-3

    query: 2-3

CDQForLIS: deep0 --------- end

CDQForLIS: deep2 --------- begin

    insert: 2-1

    query: 2-2

CDQForLIS: deep2 --------- end

CDQForLIS: deep1 --------- begin

    insert: 2-1

    insert: 2-2

    query: 2-3

CDQForLIS: deep1 --------- end

7

可以看到对于这个样例目前是对状态转移分成 5 大类（每一个 begin end 之间嵌套的部分是一个状态转移的分类）

然后对于每一个类里面有两种节点，这个是节点类型是在分治的过程中从左侧出队还是从右侧出队决定的。

CDQ 分治不允许处理同时来自左侧或者同时来自右侧的数据，必须是左侧对右侧产生的影响。

这个很好理解，因为同时来自左侧和右侧的信息肯定在递归的过程中就被处理掉了。

CDQForLIS: deep2 --------- begin

    insert: 1-1

    query: 1-2

CDQForLIS: deep2 --------- end

先看这个第一大类，$(1,1)$节点是 insert 类型，这代表它来自左侧，$(1,2)$节点是 query 类型，这代表它来自右侧。

程序运行到此处的意思就是说目前可以进行$(1,1) \rightarrow (1,2)$的状态转移。

然后结束，表示这一大类处理完了。

CDQForLIS: deep1 --------- begin

insert: 1-1

insert: 1-2

query: 1-3

CDQForLIS: deep1 --------- end

接下来处理第二大类，(1,1)和(1,2)都是 insert 类型，它们都来自左侧，(1,3)节点是 query 类型来自右侧。

程序运行到此处的意思是目前可以进行(1,1)->(1,3)和(1,2)->(1,3)的状态转移。

在分类之后，原本是要满足$xi \le xj$，$y_i \le y_j$这个二维的约束条件才能进行状态转移，现在只要满足 insert 在 query 之前出队的都能转移。

而出队顺序的约束条件，是一个一维约束条件，在分类讨论之后每一类内的限制条件，就从二维降到了一维。

所以说它是“分类”和“降维”

可以把我 std 里面所有的 cout 调试全都打开，然后输入一些数据看调试信息。

对于内层嵌套的 cdq 分治处理动态凸包的部分，其实也是分类讨论。

外面的大类是为了让状态转移在每一个类里面从左到右单调转移，而里面套的小类是为了让状态转移使用的直线斜率单调递增，查询的点单调递增。

细分到每个小类里面实际上处理的问题是：

状态转移从左侧单调转移到右侧，每次插入斜率优化的直线斜率单调递增并且查询的点值也是单调递增的。

那么一共加起来多少个小类呢，一共加起来大概 $nlogn$个小类，这些小类的尺寸

大小加起来数量级大概是$nlog^2n$的。所以允许你暴力枚举过去处理。

当然，本题解法其实很多嘛，无论是比较偏算法型的选手还是偏数据结构型的选手都能玩的比较开心。

从我个人的角度是比较喜欢 CDQ 分治套李超树的这个写法的，因为它是一个“程序=数据结构+算法”的结构，有一种艺术和美感在里面。

当然李超树这个东西 NOIP 大概率是不会考的，就算是能用到话我相信他 std 也不会这么写。如果不止限于 NOIP 的选手可以去学一下，NOIP 选手的话不建议把重点放在这种复杂的算法和结构上面。