警钟敲烂

目录

Preface

本文记录遇到的唐氏错误，以及需要注意的点。可能掺杂了少量 trick。

唉唉，「记录警钟敲烂的频率」也可以体现「一个时期卷的程度」吧。

注： 若无特殊说明，本文第 93 条及以后的均默认开了 O2 优化。~~前面的部分年久失修~~（2024.9.5）

upd 2025.8.5： 使用 typst 替代 markdown，手动一条一条改格式，并对较明显的错误进行了修订。鉴于本文过于臃肿的情况，将本文按时间顺序 divide 成章节。

第一章 (2023 ~ 2024.6)

少用递归。

线段树空间开 4 倍。

当自己的算法复杂度过低时，要警惕。

多组测试数据记得清零，不要用 `memset`。

数学题可以用 dp。

推柿子。

加入快读快写。

变量名不要换错。

非负数的原码，反码，补码相同。

开的下的话，数组开实际需要的两倍以上，1MB 可以开 2.6×10^5 万个 `int`（或 1.3×10^5 个 `long long`）。

离散化要去重，离散化要开两倍数组。

主席树开至少 32 倍空间。

2-SAT 点开至少 2 倍，边要看具体用了多少，开得越大越好越好。

`printf` 中标识符要与定义相符，不能输出 `long long` 用了 `%d`。

计算时一定一定要考虑后效性。

开了 ios 优化不要用 `cin`，`cout`。

当一个 STL 是空的时候，不要用 `.top()` or `.front()` 等。

位运算给每个运算都加上括号。

贪心题目一定要记得排序，如果要排序的话。

不卡常能 A 就不要卡，也不要为了简便而写一些不是特别确定的代码（如用式子代替 `if`），一定要求稳！

用邻接矩阵存图的话一定要判重边，给边权取 `min`！

注意题目中给出的编号是从 1 开始还是从 0 开始！

数位 dp 注意前导 0 的处理！（2023.1.31）

翻转数组注意是循环到 `n / 2`，不是 `n`！（2023.2.2）

不要忘记取模，把每个运算后都进行取模！（2023.2.16）

遇到大模拟或大分讨，一定要把所有能特判掉的东西特判掉！（2023.2.26）

快速幂一定要先把 `x` 模 `p`，避免 `x * x` 爆 long long！但不要给 `y` 模 `p`，如果一定要减小 `y` 的话要用拓展欧拉定理。（2023.3.3）

Floyd 用之前记得把 `dis[i][i]` 都赋值成 0！（2023.3.4）

记忆化搜索一定要加判断是否已经走过！（2023.3.7）

注意哪些地方要用 `long long`，哪些地方要用 `double`，或者是 `long double`。（2023.3.8）

输入数据超过 3×10^5 时建议用快读，超过 10^6 时一定要用快读！（2023.3.8）

在结构体里定义数组，并把结构体作为函数参数，要注意结构体内数组的大小不能过大！我今天在结构体里开了 160000 的 `long long` 数组，结果就 RE 了！ 经检验，`long long` 数组最多只能开到 40000 多一点点，50000 都不行！ （2023.3.9）原因：拷贝结构体需要使用栈空间，栈空间堆满就 RE 了，具体数组开多大要看栈空间。（2024.7.22）

在时限够的情况下，如果不能充分保证正确性，要多增加循环次数等，增加正确的可能性（如 CF1804C，zry 只循环到了 n，正解要循环到 $2n$，这时候如果不能保证自己推导过程一定是正确的，可以通过增加循环次数（如循环到 $10n$），来避免 WA）（2023.3.14）

缩点之后要注意数组名是 `H1` 和 `E1`，不是 `H` 和 `E`！（我总是把这个打错）（2023.3.19）

注意缩点之后去连边可能会出现重边（例如 节点 $1,2$ 同属于强连通分量 1，节点 3 属于强连通分量 2，有两条边 $1\to 2,1\to 3$，缩点之后连边就会出现两条从强连通分量 1 连向强连通分量 2 的边），这时候要求某些东西用拓扑排序出错（例如转移为 `dp[v] += dp[x]` 的拓扑排序），因此要去重边。（2023.3.19）

注意 2-SAT 的点数是 2 倍的！一部分循环要到 `2n`！

最小割割的是这些边：算好最大流后，在跑网络流的图中，去掉长度为 0 的边，把源点能到的点集设为 S，不能到的点集设为 T，所有起点在 S 内，终点在 T 内的边是割边。而不是长度为 0 的边是割边，不然如果这样一张图：**源点** $\to$ (**长度** 1)$1\to$ (**长度** 1)$2\to$ (**长度** 1) **汇点**，那么三条边都是割边，就不对了。（2023.3.27）

线段树每个修改和查询操作，如果无法直接返回当前区间的答案而要分治下去的话，就要 `pushdown`，修改操作还要 `pushup`，一定不要忘了！（2023.4.9）

---

debug 时不要光盯着代码，看不出来的话一定要去调试！（2023.4.11）

期望 dp 可以用玄学做法，用 `rand()` 暴力模拟很多次，然后取平均值，在保留位数较小的情况下很有效。（2023.4.21）

`struct` 一定要写构造函数，如果没写，不要在局部定义新的结构体，否则会导致里面的变量可能是随机的！（2023.5.1）

测大样例一定要用文件输入输出！！！（2023.5.12）

---

manacher 不要忘了数组开 2 倍！（2023.7.16）

莫队的块大小如果取的是 $\frac{n}{\sqrt{m}}$，不要忘了与 1 比较一下取最大值！不然有可能块的大小为 0。

小心重边！在做 floyd 读入边的时候要 `dis[u][v] = min(dis[u][v], w)`，而不是 `dis[u][v] = w`！如果是双向边不要忘了给 `dis[v][u]` 也赋值！

前缀和要减去的是 `s[l - 1]`，不是 `s[l]`！（2023.10.11）

模数一定要复制粘贴进代码！不然你就会像 SB 的 PHX 一样，2023 牛客 Day6 T1 中因题面中模数是 998244853 而浪费 2h 调试！（2023.10.14）

一定不要忘记取模！！！！！！！！最后再进行一次取模，有时可以避免因中间部分忘取模造成的影响！写好代码后最后从头到尾检查一遍，保证没有忘取模！！！！！（2023.11.15）

线段树合并时递归完左子树后，左子树内的信息可能会发生改变，此后如果再调用左子树的信息则会 WA，应先将左子树的信息存储下来。（e.g. 今天写 P5298 犯了这个错误，样例过不去）（2023.11.28）

直接复制类似的代码要谨慎，检查细节！（如我今天把 $\geqslant$ 打成了 $\leqslant$，$60 \to 0$）（2024.1.17）

不要忘记取模！！！！！！！！！！看到题面里面写着要取模就写进 TODO 里面！！！！！！！！！！！！）（2024.1.18）

`1ull > -1ll` 的返回值是 `0`！因为有符号整数和无符号整数运算或比较时有符号整数会强制转换成无符号整数，如果不是比较运算符的话返回值也是无符号整数！因此，有符号整数和无符号整数运算时，要将无符号整数强制转换成有符号整数！（2024.2.28）

第二章 (2024.7 ~ 2024.8)

注意 `/` 是向零取整，如 `(-7) / 3` 的值为 `-2`，而向下取整的话是 $\left\lfloor \frac{-7}{-3} \right\rfloor = -3$。如果要向下取整的话，可以额外写个函数，如：

```
inline ll dv(ll x, ll y) {
  if (x == 0 || y == 0) {
    return 0;
  }
  else if ((x > 0 && y > 0) || (x < 0 && y < 0)) {
    return x / y;
  }
  else {
    x = abs(x); y = abs(y);
    return -((x + y - 1) / y);
  }
}
```

向下取整的时候千万要注意被除数和除数的正负性！！！如果除数和被除数异号的话就不要用 `/`！！！（2024.7.11）

$x \in \mathbb{Z}, x \geq \lfloor y \rfloor$ 不能推导出 $x \geq y$（反例是 $x = y \wedge y \notin \mathbb{Z}$），推导时要小心。（2024.7.12）

$\gcd(x, y, z) = 1$ 不是指 x, y, z 两两互质，只是三者最大公约数为 1！（2024.7.17）

质数筛记得 `i` 从 2 开始，不要从 1 开始，1 不是质数！（2024.7.17）

`double` 存整数的上限是 `9.007e15`，超过就会有误差！更大的整数要用 `long double` 存！（2024.7.22）

输出为小数的题目，如果没有 SPJ 输出位数就与要求的精度相同，有 SPJ 就比要求的精度多一位。（2024.7.20）

有些时候代码调不出来，可能是因为输入部分/输出部分的代码写错了，或者是忘写了输入/输出的代码，检查的时候要去看输入部分/输出部分是否写错。（2024.7.22）

求 x 的逆元时要注意 x 是否为 0！如果 $x=0$ 的话虽然不会 RE，但是在意义上除以 0 本来就是不合法的，代码中出现乘 0 的逆元的时候大概率就是没处理好细节，可能会导致 WA。 如计算两个分数 $\frac{p}{x}$ 和 $\frac{y}{p}$ 的乘积在模 p 意义下的值，若分别计算出 $\frac{p}{x}$ 和 $\frac{y}{p}$ 在模 p 意义下的值（都为 0，因为 $p \bmod p=0$）再相乘，计算出来为 0，但实际上应等于 $\frac{y}{x}$ 在模 p 意义下的值。 故求 x 的逆元时要注意 x 是否为 0！求分数在模 p 意义下的值的时候也要注意分子或者分母是否有一项 $\bmod p=0$。分子 $\bmod p=0$ 一般不会出错，但是也要注意；尤其要注意分母是否 $\bmod p=0$（2024.7.22，ZR2121 【21 ZR 联赛集训 day9】操作数列，这题因为没注意这个细节而挂成了 95 分）

不要忘删调试信息。（2024.7.22）

`x = x * y / z;` 不等同于 `x *= y / z;`！前者等同于 `x = (x * y) / z;`，后者等同于 `x = x * (y / z);`。（2024.7.22）

有时候如果模数较大（如 10^{10}），两个模数范围内的数字乘法也可能会爆 `long long`，这时要用 **龟速乘**。龟速乘可以在 $O(\log x)$ 的时间内计算两个数的乘积，可以取模且不会爆 `long long`。（2024.7.22）

Code:

```
ll smul(ll x, ll y) {
  x = mo(x); y = mo(y); // 这一行非必需，保险起见将 x，y 取模
  ll k = 1;
  while (y > 0) {
    if (y & 1) {
      k += x;
      k %= MO;
    }
    x *= 2;
    x %= MO;
    y /= 2;
  }
  return k;
}
```

要清空！要把所有使用到的东西都清空！（2024.7.25，ZR2115 【2021 提高组十连测 day7】）

要检查是否会 MLE！数组不仅不要开小，还不要无脑开太大！（2024.7.25，ZR2149 【21 ZR 联赛集训 day18】聚会）

根号算法的块长要记得设下限和上限，让块长保持在一个合理的区间里面，如 ZR2149 【21 ZR 联赛集训 day18】聚会，可以这么写：`B = min(250ll, max(100ll, (ll)(sqrt(double(qn) * double(n) / double(n + qn)))));`。这道题块长应为 $\sqrt{\frac{n\times q}{n+q}}$，我因为脑瘫把块长算成了 $\sqrt{\frac{n\times q}{n}}=\sqrt{q}$，导致在其他数规模很大、$q$ 很小的情况下会 T，如果给块长设了下限就不会有这种情况。（2024.7.25，ZR2149 【21 ZR 联赛集训 day18】聚会）

更改数组的下标时，要记得把所有用到该数组的地方都改一下！如 ZR2148 【21 ZR 联赛集训 day18】游戏，我在将 `a[i][j]` 改为 `a[j][i]` 的时候忘了给定义处的下标换顺序，导致 WA 了 2 发。（2024.7.26，ZR2148 【21 ZR 联赛集训 day18】游戏）

在做序列相关的思维题时，不仅要考虑原序列，还要考虑前缀和序列、二阶前缀和序列、差分序列、二阶差分序列。（2024.7.26，ZR2148 【21 ZR 联赛集训 day18】游戏）

在做操作时，不仅要关注操作本身的性质，还要关注操作对象的性质。如 ZR2148 【21 ZR 联赛集训 day18】游戏，令前缀和序列 $s_i \leftarrow s_i + a_{i+1}$，$s_{i+1} \leftarrow s_{i+1} - a_{i+1}$，由于 $s_i = \sum_{j=1}^{i} a_j$，$s_{i+1} = \sum_{j=1}^{i+1} a_j$，故该操作相当于交换 s_i 和 s_{i+1}。（2024.7.26，ZR2148 【21 ZR 联赛集训 day18】游戏）。

`int` 常数比 `long long` 小，如果卡常卡不过且可以只开 `int` 的话可以把 `long long` 换成 `int`！（2024.8.9，ZR2885 24ab-day4 激光阵）

离散化常数并不小，约为 1，在计算常数的时候不能将其忽略了！在 ZR2885 24ab-day4 激光阵 中，离散化甚至跑的跟线段树差不多一样慢。另外，`sort` 的复杂度是玄学，常数也不是特别小。（2024.8.10，ZR2885 24ab-day4 激光阵）

不要只关注最终计算出的结果，有时候计算过程中算出的中间值也可以有用。如 ZR2948 24ab-day8 悠长假期 中，我只关注了 $p = 2^n - 1$ 这一档部分分，只去朝着全局的情况想，只用到了 dp 最后算出来的全局的答案，忽略了 dp 过程的中间值可以用来求出区间的答案。（2024.8.13，ZR2948 24ab-day8 悠长假期）

不要因为部分分已经拿了很多了就不去想正解，有时候从部分分到正解并不难。（2024.8.13，ZR2948 24ab-day8 悠长假期）

调试时要确定程序确实出错了，而不是自己看错样例或脑瘫等导致以为程序错了（而实际上是对的）。（2024.8.21）

写比较函数的时候要注意不能出现 $a < b$ 和 $b < a$ 同时为真（即 `cmp(a, b)` 和 `cmp(b, a)` 都为 1）的情况，否则会 RE。当 $a < b$ 和 $b < a$ 同时为假时，才会认为 $a = b$。写比较函数的时候要注意等于的情况。（2024.8.25）

对于会以函数指针的形式传递出去的函数，加了 `inline` 不会有比较好的优化，反而可能会变慢，因为 `inline` 会将其包装起来，详见我与 PPP 的 QQ 聊天记录（我在 QQ 上发给自己了）。（2024.8.25，ZR2944 24ab-day7 石子游戏）

在二分时，要注意 `r * 2` 不能超过类型能存储的最大值，因为 `l` 最大可以达到 `r`，计算 `mid` 时 `mid = ((l + r) >> 1)`，`l + r` 最大会达到 `r * 2`。（2024.8.29）

如果 通过给结构体写构造函数 来 使得其在定义为局部变量时自动清空，要注意这会导致定义成全局变量的时候除了原本已有的自动的清空以外还会再清空一次，消耗 $O(n)$ 的时间（常数可能会到达 0.1）（在 P3803 【模板】多项式乘法（FFT） 中我的手写复数类共开了 8.4e6 个 `double`，给手写复数类写了构造函数后最慢的点多跑了 135ms）。并且原生的对全局变量的清空已经优化过了，跑的很快，几乎可以忽略不计；而手动清空（即置为 0，不论用 `for` 循环清空还是写结构体的构造函数）都要消耗 $O(n)$ 的时间（常数可能会到达 0.1）。故要卡常的时候记得去掉清空的构造函数 **（别忘了给局部变量手动清空）**。(2024.8.29，P3803 【模板】多项式乘法（FFT）)

> Quote
>
> 在你使用 C/C++ 的 `int` 类型时，如果发生了溢出，比较可能的情况是按照模 2^{32} 同余的前提下，在 `int` 范围内取一个合理的值。例如在计算 $2147483647 + 2$ 时，较有可能会得到 -2147483647。
>
> 然而，C/C++ 标准将这种情况归类为“未定义行为”（UB）。当你的程序试图计算会溢出的 `int` 运算时，除了上述结果外，编译器还可能会让你的程序在此时计算出错误结果、死循环、运行错误等，这也是符合 C/C++ 标准的。
>
> 如果你的程序希望利用 `int` 的自然溢出的特性，请转换为 `unsigned int` 类型运算。例如将 `a + b` 改写为 `(int) ((unsigned int) a + (unsigned int) b)`，以避免出现不预期的错误。

（2024.8.30，从 P5285 [十二省联考 2019] 骗分过样例 的题面中抄过来并稍加修改的）

非递归的函数能加 `inline` 就加 `inline`，尤其是调用次数很多的函数！如 P3803 【模板】多项式乘法（FFT） 中，我没有给手写的复数类的运算添加 `inline`，结果 T 了（时限 2s，跑得都超过了 2.2s），加了 `inline` 后只跑了约 500ms。（2024.8.30，P3803 【模板】多项式乘法（FFT））**（本条有待斟酌（2025.7.27））**

第三章 (2024.9 ~ 2024.9)

每个 `vector` 在初始时都会占用 24Byte 的空间（存放 3 个指针），因此不要开过多的 `vector`！(2024.9.1，ZR2978 【24noip 十连测 day1】好数)

~~任何堆内存的 STL 在新开一段空间时都会附带着多开一个 `unsigned long long`，因此 `vector` 占用的空间大约是内部元素原本空间的 2 倍。（2024.9.1）~~ 在 g++下，`vector` 占用的空间最大可达到内部实际存储元素所占空间的 2 倍。（2024.10.25）

链式前向星和 `vector` 建边的速度比较：如果只跑一遍搜索则链式前向星更快，如果跑多次搜索则可能 `vector` 更快。因为 `vector` 的寻址比较连续，在建边时链式前向星更快，在搜索时 `vector` 更快。（2024.9.1，according to hjh）

全局变量的地址是倒着分配的。如下面这段代码：

```
int c, a[100], b[10];
```

则 `&c` 为 `&a[99] + 1`，`&a[0]` 为 `&b[9] + 1`。（2024.9.1）

树剖中向上跳的部分，要比较 `de[tp[x]]` 与 `de[tp[y]]` 的大小来决定是跳 `x` 还是跳 `y`，而不是比较 `de[x]` 和 `de[y]`！

Code:

```
while (tp[x] != tp[y]) {
  if (de[tp[x]] < de[tp[y]]) swap(x, y);
  // do something...
  x = fa[tp[x]];
}
if (de[x] < de[y]) swap(x, y);
// do something...
```

有时候，`x = (x + y) % MO` 会比 `x += y; if (x >= MO) x -= MO;` 要快。如 ZR1278 【20 省选十联测 day9】Horrible Cycles 中，前者比后者差不多快一倍。原因是 `if` 其实比较慢。（2024.9.3，ZR1278 【20 省选十联测 day9】Horrible Cycles）

树状数组下标不能为 0！（2024.9.3）

点分治中 dfs 时记得要判 `!vis[v]`！（2024.9.3）

写 FFT 或 NTT 的时候不要忘了先预处理 `rev` 并进行蝴蝶变换！也不要忘了如果 `flag` 为 1（即 IDFT 的时候）不要忘了把结果的每一项都除以 `s`！（2024.9.3）

在重载运算符的时候，一般参数列表中写成 `const &`，如：

```
nde operator + (const nde &p, const nde &q) {
  return p + q;
}
```

因为不写 `&` 会将传进来的参数复制一份，导致速度变慢；写了 `&` 而不写 `const` 会使得传进来的必须是变量而不是一个表达式（如 `1`，`'a'` 等）。只有写了 `const &` 才能使得表达式也能被传进去，同时不会复制传进来的参数。（2024.9.4）

小心模数为 1 的情况！（2024.9.5）

建议在求出最后答案后再对答案取一下模。（2024.9.5）

`s += 'a'` 是 $O(1)$ 的，`s = s + 'a'` 是 $O($**字符串长度**$)$ 的，因此千万别用后者。（2024.9.5）

```
vector<nde> tr;
int new_nde() {
  tr.push_back(nde());
  return ll(tr.size()) - 1;
}
tr[x].l = new_nde();
```

这段代码有概率 UB。如果 `tr.push_back()` 新申请了一块内存，会导致之前的迭代器全部失效。如果在执行 `tr[x].l = new_nde();` 这句话时，先执行左边获得了 `tr[x].l` 的迭代器，再执

行右边时 `tr.push_back()` 新申请了内存，就导致了 `tr[x].l` 的迭代器失效，会导致 UB。写成 `int k = new_nde(); tr[x].l = k;` 就不会有问题（2024.9.5，according to PPP）

`int a[很大的数] = {1};` 容易死，因为编译器会将其展开为全 0 的初始化列表（i.e. `{1, 0, 0, 0, /*很多个 0*/}`）并存到编译后的程序里，同时占用了编译器内存，会导致 CMLE（编译时内存超限）以及编译后的程序过大。因此千万不要在定义很大的数组时用大括号初始化。（2024.9.5, modified on 2024.12.18）~~另外，听 PPP 说 `bitset` 的构造函数也是这么写的，因此不要开过大的 `bitset`，一般开 `bitset<5e7>` 就会导致 CMLE。（2024.9.5，according to PPP）~~

Linux 下在终端中运行可执行文件要用 `./可执行文件名`，不能用 `./ 可执行文件名`！Linux 下也不能通过在文件资源管理器中直接打开可执行文件来运行可执行文件！（2024.9.6）

`(complex<double>)(x, y)` 和 `complex<double>(x, y)` 有区别。前者是强制类型转换，会先计算出 `x, y` 的结果（也就是 `y`，因为 `,` 是返回右边的运算数的），再将 `y` 强制类型转换成 `complex<double>`，结果相当于 `complex<double>(y)`；后者是调用 `complex<double>` 的构造函数，结果就是 `complex<double>(x, y)`。因此千万不要用强制类型转换！（虽然对于内建类型来说，强制类型转换和构造函数是同一个东西）（2024.10.25）

要注意树状数组的下标不能为 0！最好在 `for (ll i = x; i ⩽ n; i += lowbit(i))` 前面特判一下 `x ⩽ 0` 的情况。（2024.9.6）

在 dfs 树并用父亲的信息求出儿子的信息时，如果是这么写：

```
void dfs(int x, int fr) {
  // 求出 x 的信息
  for (int i = H[x]; i > 0; i = E[i].nxt) {
    int v = E[i].t;
    if (v == fr) continue;
    dfs(v, x);
  }
}
```

要注意特判 `fr ⩽ 0` 的情况（也就是 `x` 是根节点时），或者检查在 dfs 前数组中下标为 `0` 的数是否赋了正确的值。

如果是这么写：

```
void dfs(int x, int fr) {
  for (int i = H[x]; i > 0; i = E[i].nxt) {
    int v = E[i].t;
    if (v == fr) continue;
    // 求出 v 的信息
    dfs(v, x);
  }
}
```

要注意在 dfs 前先求出根节点的信息。（2024.9.6）

进考场前检查衣服和裤子口袋里面有没有东西！永远不要把手机放进口袋里！（2024.9.7）

一定要对拍，不要因为某些题感觉暴力和正解代码几乎一样而不去对拍，因为暴力和正解有时候有一些细节上的不同，有时候思考怎么写暴力而不是正解时就能发现正解的 bug，或者也可以用暴力拍出来。（2024.9.8，ZR2987 提克塔可头）

看到区间最小值不要认为一定是最值分治，有可能它只是取区间中点的普通的分治。（2024.9.8，ZR2985 【24noip 十连测 day2】最小值 (min)）

`#define` 中的表达式左右两边一定要加括号！要不然就可能出现优先级问题，如：

```
#define add(u, v) u + v
cout << (add(1, 2) * 3); // 等价于 1 + 2 * 3，而不是期望的 (1 + 2) * 3
```

应改为 `#define add(u, v) (u + v)`。（2024.9.8）

不要用 `abs(__int128)`！它在 ISO C++ 下会 CE，在 GNU C++ 下不会 CE！评测时的编译标准是 ISO C++（`-std=c++14`），但是考场的电脑默认编译标准是 GNU C++（`-std=gnu++14`），开了 `-std=c++14` 编译选项才能将编译标准改为 ISO C++。因此编译选项要与题目规定的编译选项完全一致，要开 `-std=c++14`！（2024.9.9，P10217 [省选联考 2024] 季风）

---

dfs 求基环树中的环，要注意记 flag 来保证只会求第一次遇到的环，不然会出现问题。如

```
1 2
2 3
3 1
```

这样一个三元环，假设 dfs 顺序是 1, 2, 3，则应在 3 号点时求出这个环，但是在回溯到 1 号点后枚举到 1-3 这条边，由于 `vis[3] == 1`，会导致再求一次环，但是这时候从 1 开始跳 fa 所求出的环就错了（甚至会死循环），因为 3 不是 1 的祖先。

dfs 求基环树中的环可以这样写：（边是无向的）**（要注意这种写法初始时 `num` 要设为 1！）**

```
void find_cye(int x, int fr, int id) {
  vis[x] = 1;
  fa[x] = fr;
  for (int i = H[x]; i > 0; i = E[i].nxt) {
    int v = E[i].t;
    if ((i ^ id) == 1) continue; // 这里使用了技巧，刚开始的时候 num 要设为 1!

    if (vis[v]) {
      if (!fl) {
        fl = 1;

        int u = x;
        t = 0;
        while (u != fa[v]) {
          cye[++t] = u;
          on_cye[u] = 1;
          u = fa[u];
        }
      }
      continue;
    }
    find_cye(v, x, k);
```

```
  }
}
```

其中使用了一个技巧：`(i ^ id) == 1` 来判断这条边是否是从父亲走到自己的边，**注意这样写初始时 `num` 要设为 1！** 这种写法可以将二元环（也就是重边）也判断为环。如果不想判断二元环就使用 `v == fr` 的写法即可。（2024.9.9）

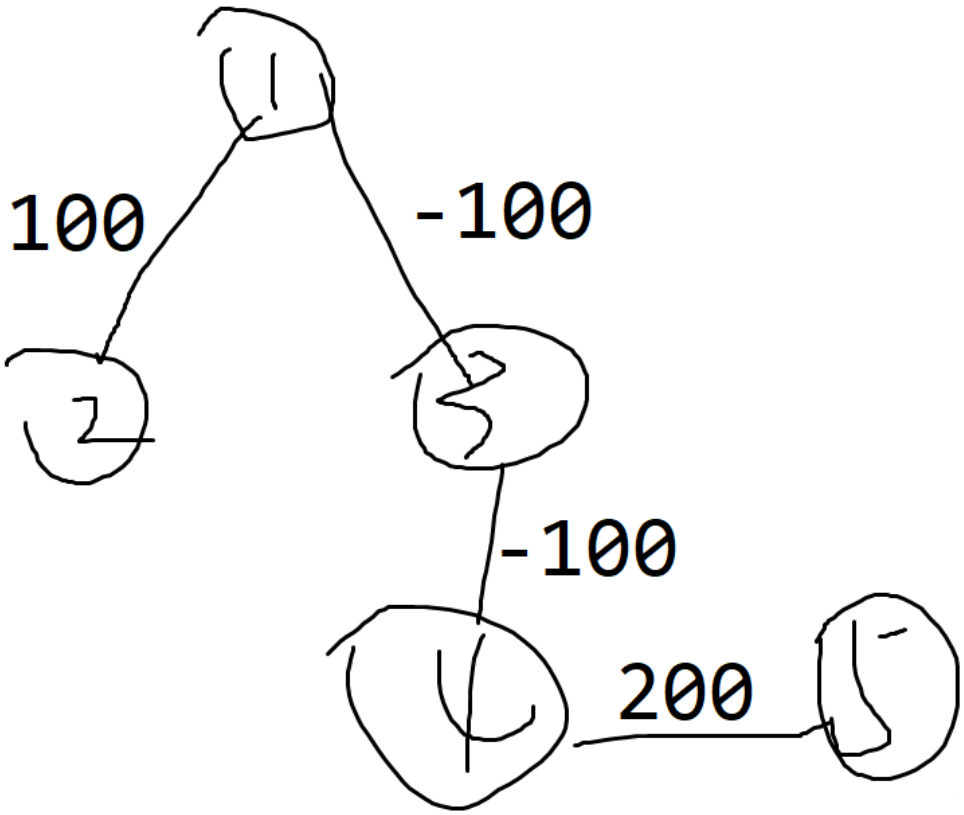
要分清 `n` 和 `m`！也要分清 `n` 和其他的表示总量的变量，不要什么东西都无脑写 `n`！（2024.9.9，ZR2984 【24noip 十连测 day2】翻转 (rev)）

初赛题目中**输出行数**是不算文末回车的，如

```
1
2
```

的输出行数是 2 行。（2024.9.9，S 组初赛模拟卷中的题目）

两遍 dfs 求树的直径的算法在有负权边的图上会错，如图：

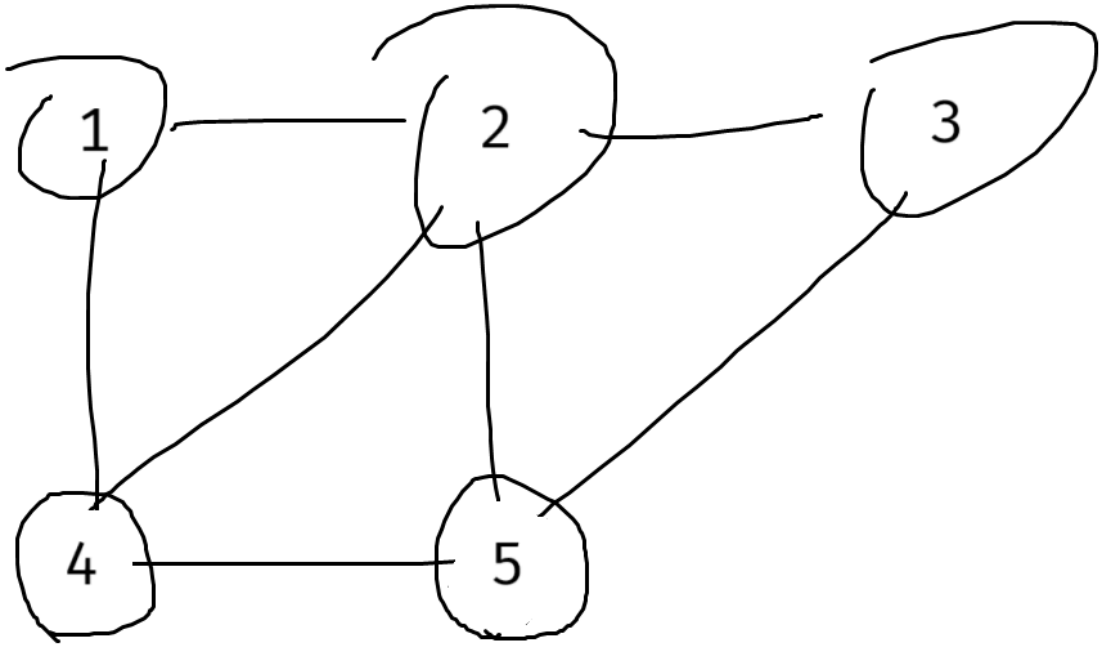


直径明显为 $(4,5)$，但第一遍从 1 开始 dfs，深度最深的点是 2 和 5，深度都是 100。如果取 2 作为最深的点，下一次 dfs 最深的点是 1 和 5，深度都是 100，得出直径为 100，错误。遇到这种情况只能用树形 dp 求直径。（2024.3.10）

在模数较小的时候要注意，有可能正解是基于模数大小的算法。（2024.9.11）

在模数较小或模数是输入的时候要注意，观察 n, m 或者是下标有没有可能大于等于模数，最好在用到它们的时候先取模。（2024.9.11）

无向图 k 染色问题是 NP 困难的。有一个乱搞：dfs，在进入一个新点 x 时枚举所有与 x 相连且已经染色过的点 v，求出所有的 v 的颜色组成的集合的 mex 作为 x 的颜色。下面这个样例可以卡掉这个乱搞：



如果从 1 开始 dfs，给 1 染色成颜色 1，然后走到节点 2 并染色为颜色 2，走到节点 3 并染色为颜色 1，再走到节点 5 并染色为颜色 3，再走到节点 4 的时候由于节点 1,2,5 分别染色为颜色 1,2,3，节点 4 只能染色为颜色 4。但实际上可以这样染色：1:1,2:2,3:3,4:3,5:1，就只用 3 种颜色了。（2024.9.13，CF805E Ice cream coloring）

对于数据中既有 n 又有 m 的，记得分别造 $n > m$，$n = m$，$n < m$ 的数据，以及 n 或 m 中一个特别小的数据来测试代码，以防代码中 `n` 和 `m` 写反了。一定要分清 `n` 和 `m`！（2024.9.14）

注意 `std::set` 和 `std::multiset` 比较元素大小只会用 `<` 运算符，当且仅当 `a < b` 且 `b < a` 时会将 `a` 与 `b` 判定为相等。如果你定义了一个结构体，其中的 `<` 运算符中只比较了结构体存储的多个值中的一个，例如下面的代码：

```
struct nde {
  ll x, y;
  bool operator < (const nde &p) const {
    return y < p.y;
  }
};
```

然后就会导致 `(nde){1, 2}` 和 `(nde){3, 2}` 被 `std::set` 或 `std::multiset` 判定为相等，就会导致 `erase()` 误删元素，或者 `find()` 找到错误的元素。正确写法应当这样写：

```
struct nde {
  ll x, y;
```

```
  bool operator < (const nde &p) const {
    if (y != p.y) return y < p.y;
    else return x < p.x;
  }
};
```

建议对于 `std::set` 和 `std::multiset` 中传入的自定义结构体类型，给它的 `<` 运算符写完整，让 `<` 运算符在比较完关键字后还会比较结构体内的其他每个值！（2024.9.16，ZR2995 [24noip 十连测 day3]御坂美琴）

答案要求保留 k 位小数，则二分答案时设置的精度应为 $10^{-(k+2)}$，也就是二分答案的边界是 $r-l>10^{-(k+2)}$。如果二分答案时设置的精度为 $10^{-(k+1)}$，有可能会被卡精度，比如答案的第 $k+1$ 位小数是 5，而你算出来小了一点点，这一位上变为 4，就会导致第 k 位少了 1。因此二分答案的精度至少要设置为 $10^{-(k+2)}$！（2024.9.22，P3291 [SCOI2016] 妖怪）

在使用 `std::map` 时要小心，即使是调用一个没有使用过的键值 `mp[x]` 作为右值（而不作为左值），也会新建一个键值对 `(x, 初始值)`，占用空间。因此在判断是否插入过键值 `x` 时，应当使用 `mp.count(x) == 0`，而非 `mp[x] == 初始值`！（2024.9.24，P3294 [SCOI2016] 背单词）

注意分数比较大小时必须保证分母为正数，要是分母为负就会导致不等号方向取反。 下面是一个比较分数的实现：

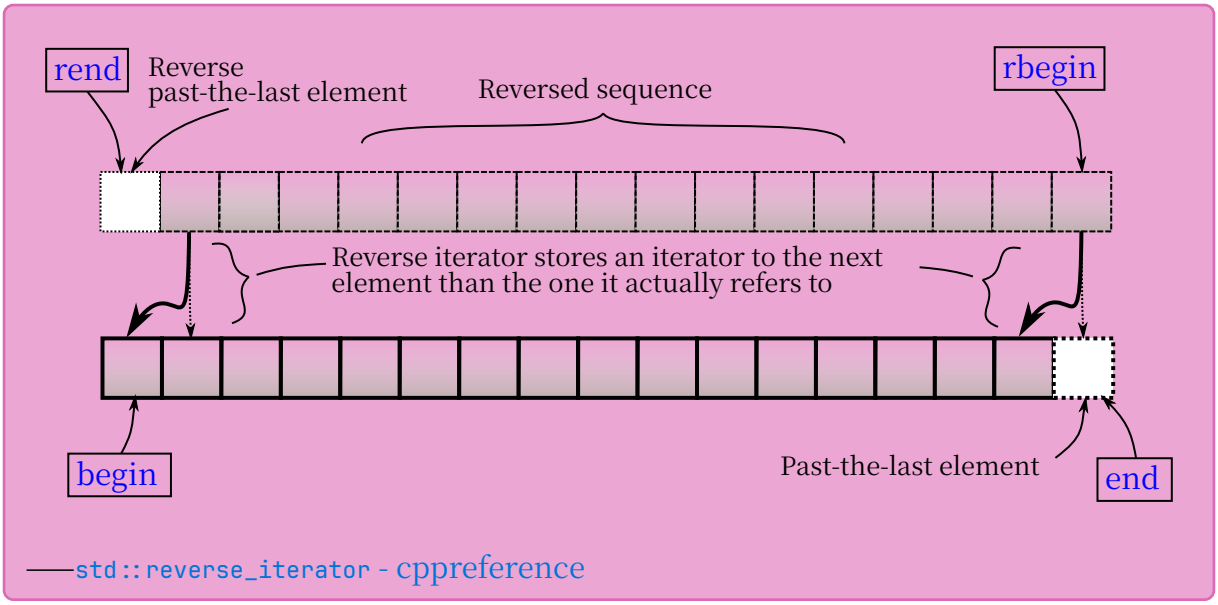
```
bool operator < (const frac &p, const frac &q) {
  if (p.y < 0) p.x *= -1, p.y *= -1;
  if (q.y < 0) q.x *= -1, q.y *= -1;
  return p.x * q.y < p.y * q.x;
}
```

（2024.9.29）

第四章 (2024.10 ~ 2024.11)

可以用 `std::make_reverse_iterator(it)` 获取正向迭代器 `it` 对应的反向迭代器，用 `it.base()` 获取反向迭代器 `it` 对应的正向迭代器。 但是要注意：

> Quote
>
> 对于从迭代器 `i` 构造的逆向迭代器 `r`，关系 `&*r == &*(i - 1)` 始终是 `true`（只要 `r` 可解引用）；因此从末尾后一位置迭代器构造的逆向迭代器解引用为序列的最后元素。



也就是说 `r` 指向的元素是 `i` 的前一个元素，这一点一定要注意；而且，当 `i` 前面紧挨着插入一个元素时，`r` 所指的元素也会相应的发生改变（详见我和 PPP 的聊天记录，合并转发到了 QQ 上我与我自己的聊天上）。因此，最好不要用反向迭代器！（2024.10.3，ZR2932 [24 十一集训] A-训练）

在 STL 的遍历过程中最好不要修改！（2024.10.3，ZR2932 [24 十一集训] A-训练）

`std::queue` 和 `std::stack` 都是给 `std::deque` 再封装来实现的，因此跑的很慢！建议用数组代替这三者；如果不知道最大需要容纳的元素数量，可以使用单个 `std::vector` 来模拟 stack，用两个 `std::vector` 模拟两个栈来模拟 queue，用两个 `std::vector` 实现 deque。

如何用两个栈来模拟 queue?

详见代码。

```
template<typename T>
struct MyQueue {
  vector<T> a, b;

  inline bool empty() {
    return a.empty() && b.empty();
  }
  inline void clear() {
    a.clear(); b.clear();
  }
  inline void release() {
    vector<T>().swap(a); vector<T>().swap(b);
  }

  inline void push(T x) {
    a.push_back(x);
  }

  inline void load_b_from_a() {
    if (b.empty()) {
      while (!a.empty()) {
```

```
        b.push_back(a.back());
        a.pop_back();
      }
    }
  }
  inline T front() {
    load_b_from_a();
    return b.back();
  }
  inline void pop() {
    load_b_from_a();
    b.pop_back();
  }
};
```

如何用两个 vector 来实现 deque?

```
template<typename T>
struct MyDeque {
    vector<T> a, b;

    inline void clear() {
        a.clear(); b.clear();
    }
    inline void release() {
        vector<T>().swap(a); vector<T>().swap(b);
    }
    inline void shrink_to_fit() {
        a.shrink_to_fit(); b.shrink_to_fit();
    }

    inline void push_front(const T &x) {
        a.push_back(x);
    }
    inline void push_back(const T &x) {
        b.push_back(x);
    }
    inline void pop_front() {
        if (a.empty()) {
            int l = max(1, int(b.size()) / 2);
            for (int i = l - 1; i >= 0; --i) {
                a.push_back(b[i]);
            }
            for (int i = 0; i < int(b.size()) - l; ++i) {
                b[i] = b[i + l];
            }
            b.resize(int(b.size()) - l);
        }
        a.pop_back();
    }
    inline void pop_back() {
        if (b.empty()) {
            int l = max(1, int(a.size()) / 2);
            for (int i = l - 1; i >= 0; --i) {
                b.push_back(a[i]);
            }
            for (int i = 0; i < int(a.size()) - l; ++i) {
                a[i] = a[i + l];
```

```
            }
            a.resize(int(a.size()) - l);
        }
        b.pop_back();
    }

    inline bool empty() {
        return a.empty() && b.empty();
    }
    inline int size() {
        return int(a.size()) + int(b.size());
    }

    inline T front() {
        if (!a.empty()) return a.back();
        else return b.front();
    }
    inline T second_front() {
        if (a.empty()) return b[1];
        else if (int(a.size()) == 1) return b[0];
        else return a[int(a.size()) - 2];
    }
    inline T back() {
        if (!b.empty()) return b.back();
        else return a.front();
    }
    inline T second_back() {
        if (b.empty()) return a[1];
        else if (int(b.size()) == 1) return a[0];
        else return b[int(b.size()) - 2];
    }
};
```

（2024.10.6，ZR2932 [24 十一集训] A-训练）

不要把什么数据结构都封装成结构体！（待补）（2024.10.6，牛客 2024 提高组（第一场） 沙漠之旅（travel） & ZR3010 [24noip 十连测 day5]签到题）

`std::vector` 占用的空间最大可达内部存储元素的两倍。（2024.10.10）

一个 `std::string` 初始时就会占用 32Byte 的空间！（2024.10.10）

`std::string` 的底层是 `char` 数组，`s[s.size()]` 处为 `'\0'`，访问此处不会造成越界错误，fsanitize 也不会报错。因此要注意 `std::string` 的下标！（2024.10.18）

如果有向图中存在环，那么当拓扑排序结束时，还没有访问过的点是＜所有在环上的点 以及在环上的点所能到达的点＞。（2024.10.19，ZR2968 Ancient Engine）

不要因为觉得贪心正确性不高就不去写贪心，有时候简单的贪心策略就是对的！（2024.10.20，ZR3047 背包（knapsack））

`std::unordered_map` 并不是很快，很多情况下还是自己手写哈希表比较好。（2024.10.21，ZR3032 对称差 (setxor)）

要注意对称性！（2024.10.21，ZR2976 作曲家）

当操作中有交换两个元素时，注意特判交换的两个元素是同一个位置的情况！（2024.10.22，https://ac.nowcoder.com/acm/contest/90642/A）

注意如果使用倍增求 lca 且将根节点的父亲设为 0 的话，要注意根节点的深度不能为 0！否则根节点的深度就与 0 的深度相同了，会导致查询时若一个点为根节点，则会跳到 0！（2024.10.23，ZR3024 释放魔剑的锻造师）

注意，程序使用的栈空间约为 $\frac{\text{递归层数}}{20000}$ MB （据本地 NOI Linux 2.0 虚拟机测试），计算内存时不要忘记算进栈空间！（2024.10.25）

在使用刷表法进行 dp 转移时，对于没用的位置（如计数 dp 中为 0 的位置，要求 dp 值最小化时为 `INF` 的位置），可以 `continue` 掉。这样在＜有值的位置很少＞的情况下能够极大地提升程序效率。（2024.10.28，ZR2972 冒险家）

`__gcd(__int128, long long)` 会将 `__int128` 隐式类型转换成 `long long`！`__gcd` 有如下三种重载：

```
template<typename T>
T __gcd(T, T);
long long __gcd(long long, long long);
int __gcd(int, int);
```

因为有 `long long __gcd(long long, long long);` 这一种重载，因此 `__int128` 会被隐式类型转换成 `long long`。

不过由于 `template<typename T> T __gcd(T, T);` 的存在，`__gcd(__int128, __int128)` 仍然是对的。

因此，一定要手写 `gcd`!!!（2024.10.28，ZR3003 规规划）

离散化是二分的上界是 `std::unique` 后的数组长度！不要写成原来的数组长度了！（2024.10.28，ZR3028 外星人）

`std::accumulate` 返回值的类型是由它的第三个参数的类型决定的，与它的第三个参数的类型相同。如果你要对一个 `long long` 的数组求和（答案超过 `int`），但是第三个参数传的是 `int`（如 `0`），那么就会爆 `int`，并且 sanitizer 还不会报错（本地测试）！解决办法是要么第三个参数传 `0ll`，要么写清楚模板类型（`std::accumulate<ll*, ll>`）！（2024.10.31，ZR3059 可以了）

杜教筛别忘了特判 $n \leq 0$ 的情况要返回 0！（2024.10.31）

Min_25 筛别忘了加上 $f(1)$！（2024.11.1）

写筛的时候因为数据范围很大，可能会达到 10^{10}，要注意 $n*n$ 会爆 `long long`，要判断几个数乘起来是否 $\le n$ 时建议使用除法而非乘法。不过要注意除法比乘法慢很多。（2024.11.1, , P5325 【模板】Min_25 筛）

写 Powerful Number 筛时，要注意 `dfs` 的写法。下面这种复杂度是错的：

```
inline void dfs_PN(ll i, ll x, mit val) {
  if (i > prn || pr[i] * pr[i] > n) {
    ans += val * calc_sum_g(n / x);
    return;
  }

  dfs_PN(i + 1, x, val); // 不应该把找质数的任务放到下一层去做
  for (ll j = 2, k = pr[i] * pr[i]; x * k ⩽ n; ++j, k *= pr[i]) {
    dfs_PN(i + 1, x * k, val * h[i][j]);
  }
}
```

每次 `dfs` 时必须找到下一个次数 ≥ 2 的质数并乘上去，代码中直接放弃当前质数并 `dfs` 下一层会导致单个 PN 被 `dfs` 多次（尽管只会在最后被统计），导致复杂度退化为 $\sqrt{n} \times$ **层数**，其中层数即为 $\le n$ 的质数个数。

正确的写法如下：

```
inline void dfs_PN(ll pre, ll x, mit val) {
  ans += val * calc_sum_g(n / x);

  for (ll i = pre + 1; i ⩽ tot_pr; ++i) {
    if (x > n / pr[i] / pr[i]) break; // 这一行剪枝不能少！
    for (ll k = 2, u = pr[i] * pr[i]; u ⩽ n / x; ++k, u *= pr[i]) {
      dfs_PN(i, x * u, val * h[i][k]);
    }
  }
}
```

其中 `if (x > n / pr[i] / pr[i]) break;` 的剪枝不能少，不然复杂度也会退化。

另外要注意，不能写 `x * pr[i] * pr[i] > n` 和 `u * x ⩽ n`，因为 `x * pr[i] * pr[i]` 可能会爆 `long long`，要写成除法。

不过 `pr[i] * pr[i]` 是 *len* 的，保证了 `pr[i] * pr[i] * pr[i]` 不会爆 `long long`，也就保证了 `u` 不会爆 `long long`。（2024.11.1，P5325 【模板】Min_25 筛）

Min_25 筛要注意 dp 数组等存储下标为 $\lfloor \frac{n}{i} \rfloor$ 的值的数组要开到 $2\sqrt{n}$！集合 $\{\lfloor \frac{n}{i} \rfloor \mid 1 \le i \le n\}$ 的大小的上限是 $2\sqrt{n}+1$。（2024.11.2）

Min_25 筛遍历 $\lfloor \frac{n}{i} \rfloor$ 的值时要注意不能使用 $\lfloor \frac{n}{i} \rfloor^2 \le n$ 来判断是否有 $\lfloor \frac{n}{i} \rfloor \le \sqrt{n}$，因为 $\lfloor \frac{n}{i} \rfloor$ 最大可达 n，n^2 会爆 `long long`！建议使用除法或预先计算出 $\lfloor \sqrt{n} \rfloor$ 的值。（2024.11.2，P5493 质数前缀统计）

Min_25 筛预处理完全积性函数在质数处的前缀和时不要忘了减去 1 处的贡献！（2024.11.2，P5493 质数前缀统计）

Min_25 筛最后统计答案时不要忘了加上 1 处的贡献！（2024.11.2，P5325 【模板】Min_25 筛）

Min_25 筛如果 WA 了，记得检查是否 1 处的贡献忘减或忘加了！（2024.11.2，P5325 【模板】Min_25 筛）

将一个 `long long` 范围的值赋给 `int` 类型的变量时，sanitizer 不会报错，因为赋值相当于强转，sanitizer 不会对强转报错，只会对运算时的溢出报错！（2024.11.11，ZR2942 [24noip 十连测 day7]水题 Plus）

思维题、结论题要多打表找规律，不要硬想，要有毅力打表找规律；也千万不要陷入思维死角，没有头绪的时候可以先去做别的题或者去上个厕所，重新想时如果之前的思路还是行不通就一定要跳出之前的思路！（2024.11.13，ZR3098 游艇）

不要直接复制粘贴取模的板子，要注意模数是否是题面中的模数！（2024.11.14）

写动态开点线段树和可持久化线段树的时候，要注意树高最大是 $\lceil \log_2 V \rceil + 1$ 的，例如 $V = 10^9$ 时树高会达 31，因此空间最好多开一点！（2024.11.22，P3834 【模板】可持久化线段树 2）

＜扫描线求矩形并的面积的线段树＞空间要开＜离散化后的 x 坐标数组大小＞的 8 倍（也就是矩阵数量的 16 倍），而不是 4 倍！原因是＜扫描线求矩形并的面积的线段树＞比较特殊，会在打完标记后也会进行 `pushup`，也就是说它在叶子结点处也会 `pushup`，而 `pushup` 会访问到叶子结点的两个不存在的子节点的信息（即使这些信息没用），因为叶子结点的下标的上限是 $4n$，其不存在的子节点的下标的上限也就是 $8n$。（2024.11.22，P5490 【模板】扫描线 & 矩形面积并）

注意 SPFA 如果不写循环队列的话，队列数组的大小要开到边数而不是点数！所以建议写 SPFA 时用 `std::queue`。（2024.11.28，P3385【模板】负环）

第五章 (2024.12 ~ 2025.7.14)

结构体写默认构造函数如果加了 `constexpr`，要注意不要开这个结构体的很大的数组，否则会导致 CMLE。如我在 P5012 水の数列 中写了如下代码：

```
constexpr int fn = 1e6;
constexpr int N = fn + 10;
struct frac {
  ll u, v;
  constexpr inline frac() : u(0), v(1) {}
};
frac val[N];
```

然后 NOI Linux 本地编译好好的，但是交到洛谷上后 CMLE 了（显示：`Nothing is compiled: OUTPUT exceeds.`），去掉 `constexpr` 后成功通过编译了。

如果非要写 `constexpr` 的话也可以这样来避免 CMLE：

```
unsigned char pool[N * sizeof(frac)];
frac* val = new(pool) frac[N];
```

（2024.12.17，P5012 水の数列）

在开启 `-std=c++14` 后，`is_integral<__int128_t>::value` 和 `is_integral<__uint128_t>` 是 `false`！（2024.12.25，https://www.cnblogs.com/caijianhong/p/18629812）

注意：`while (cnt--) { /* do something */ }` 后会把 `cnt` 置为 `-1`，所以如果后续还要拿 `cnt` 作为待执行次数的计数器的话，建议写成 `while (cnt > 0) { /* do something */ --cnt; }`。（2025.1.1，CF1540D Inverse Inversions）

有些题目只保证 $n \times m$ 的上限，在使用一维数组模拟二维数组时，要注意如果下标从 1 开始，也就是共需要 $(n+1)(m+1)$ 的空间，那么数组大小要开到 $2nm$（因为总大小最多增加 $n+m+1$）。（205.1.5，ZR3096 25 省选 10 连测 day3-二向皈依）

在代码中进行全局替换时，要注意不要把 `freopen` 的文件名中的某个字母或子串替换掉了！（2025.1.7，ZR2021 [zr 联赛集训 day14]字符串）

在写长链剖分优化 dp 时，要注意如果需要合并自己与儿子的 dp 数组（来贡献到答案）而不是简单的转移，那么不要忘了从重儿子处合并 dp 数组！（P3565 [POI2014] HOT-Hotels）

两个 `short` 类型的数进行算术运算时，结果是 `int`！`bool`、`signed char`、`unsigned char`、`char`、`unsigned short` 同理。（2025.1.12）

在 `vector::size() ⩽ i && i < vector::capacity()` 时，`vector::operator[](i)` 不会使 sanitizer 报错！（2025.2.5）

判断 `x % y` 是否不为 0 不能用 `x % y > 0`，要用 `x % y ≠ 0`！因为余数可能是负数。特别的，判断 `x % 2` 是否为 0 可以用 `x & 1`。（2025.2.14，AT_agc052_e [AGC052E] 3 Letters）

复制字符数组时如果不用 `strcpy`，则要把尾后指针处的字符置为 `'\0'`（也即 `0`），因为字符串的存储格式就是以 `'\0'` 作为结尾。（2025.2.22，P1117 [NOI2016] 优秀的拆分）

使用 Matrix-Tree 定理时，要注意消元求行列式时如果某一列没有非零元素，那么行列式就直接返回 0。（2025.3.5，P6624 [省选联考 2020 A 卷] 作业题）

自然溢出的哈希不要使用 998244353 作为 `BASE`！！！（2025.3.12，P9242 An Easy Geometry Problem，PPP 由于使用 998244353 作为 `BASE` 的奇怪习惯导致被卡哈希了）

三元环计数给边定向时，如果两点度数相同则要按编号大小进行定向！即条件为 `dg[U[i]] < dg[V[i]] || (dg[U[i]] == dg[V[i]] && U[i] > V[i])` 而不是 `dg[U[i]] < dg[V[i]]`！（2025.3.20，P1989 无向图三元环计数）

写线段树的 `build` 时，记得懒标记不仅仅要在叶子结点处清空，在每个节点上都应该清空！因为即使使用了 `pushup` 来求出父节点处的信息，`pushup` 也不会改变懒标记。例如下面是一个规范的 `build` 函数示例：

```
void bld(int x, int l, int r) {
  tr[x].tag = 0;
  if (l + 1 == r) { tr[x].sum = 0; return; }
  int o = ((l + r) >> 1);
  bld(x << 1, l, o);
  bld(x << 1 | 1, o, r);
  pus(x);
}
```

（2025.3.24，CF1830F The Third Grace）

KTT 的叶子结点的 `lim`（也就是最小的能使节点信息改变的 `x` 的增量）一定要设为 `INF`！要不然依照其实现方式的话，叶子结点的信息无法通过 `pushup` 进行更新，也就错了！
（2025.3.25，P5693 EI 的第六分块）

欧拉回路要注意，这种写法是错误的：（以无向图的代码举例子，有向图同理）

```
void dfs(int x) {
  for (int i = C[x]; i > 0; i = E[i].nxt) {
    C[x] = E[i].nxt;
    if (!used[i]) {
      used[i] = used[i ^ 1] = 1;
      int v = E[i].t;
      dfs(v);
    }
  }
  *stk_l++ = x;
}
```

其中 `i` 定义为局部变量，假设第一次 dfs 遍历到了 x 这个节点，在往下递归的过程中，第二次地遍历到了 x 这个节点，在第二层 dfs 处把 `C[x]` 修改为 0。则在回溯到第一层 dfs 后，这一层的 `i` 保持在原有的值（而不是现在的 `C[x]`），而 `C[x]` 已在第二层 dfs 被修改为 0 了，也就是 `i` 的所有 `nxt` 的边都已经被访问过了，不应当再被遍历，但是 `i` 会继续跳 `nxt` 把这些边都遍历一遍。于是会被卡到 $O(n^2)$。

正确写法如下，给 i 加上引用即可（或者直接使用全局的 `C[x]` 替代 `i` 也行）：

```
void dfs(int x) {
  ++cnt[x];
  for (int &i = C[x]; i > 0; i = E[i].nxt) if (!used[i]) {
    used[i] = used[i ^ 1] = 1;
    int v = E[i].t;
    dfs(v);
  }
  *stk_l++ = x;
  assert(stk_l < end(stk));
}
```

（2025.4.8，CF1186F Vus the Cossack and a Graph，我因为这个傻逼错误调了 1h）

对于内建的左移、右移运算符 `x << y` 和 `x >> y`，若 `y` 的值为负或大于等于 `x` 的位数，则行为未定义！虽然底层实现一般会让 `y` 对 `x` 的位数取模，也就是说 `int` 左移或右移 32 位的结果一般与原数相等。（2025.4.10，ZR1945 【21 暑期集训 C 班 day 1】打蚊子，今天写手写 bitset 时被这玩意坑了，第一次发现这个 UB 的过程详见我和 dly 在 2025.1.6 的聊天记录）

当程序没有过某个小样例时，可以试着去测测别的样例，说不定程序恰好会在某个较为极端的情况下出错，而小样例恰好就是这个情况。如 CF2096H Wonderful XOR Problem，我的程序在存在 $r_i = 2^m - 1$ 时会出错，而前小样例的前两个 testcase 都包含这种情况，后两个 testcase 则不包含这种情况。（2025.4.21, CF2096H Wonderful XOR Problem）

假设 `solve()` 是一个会向 stdout 输出一些内容的函数。则 `cout << "! " << solve();` 这行代码，有概率先输出 `！` 再执行 `solve()`，有概率先执行 `solve()` 再输出 `！` 。

原因是该语句等价于 `(cout << "! ") << solve();`，也就是说对于第二个 `<<` 运算符，其左右各有一个表达式，根据 C++ 标准这两个表达式的 **求值顺序** 是不固定的，可以由编译器任意决定。于是如果先求值了左边的 `cout << "! "` 则会先输出 `！` 再执行 `solve()`，反之同理。

（2025.4.27，昨天在 ZJCPC 的热身赛上，PPP 在签到的 B 题（是一道交互）写出了这样的代码导致 WA 了 2 发，还是在~~被迫~~写对拍时 PPP 突然发现这个锅的，只能说 C++ 有些地方太神秘了）

写 Miller Rabin 和 Pollard Rho 时要注意会不会爆 `long long`！（2025.4.29，P4718 【模板】Pollard-Rho）

无论如何，WBLT 都要预先插入一个 ∞ 来避免被删光而导致的错误。（2025.5.12，P6136 【模板】普通平衡树（数据加强版），这题中虽然可以预先 build，但是也有可能初始的数都被删光，故还是要预先插入 ∞）

写凸包相关的时，要注意两个点 x 坐标相同的情况！求上凸壳时只要把点按照 x 第一维从小到大，y 第二维从小到大排序即可避免 x 坐标相同、y 坐标更小的把大的 ban 了。求下凸壳时同理，y 第二维从大到小排序即可。

负数 `<< k` 是未定义行为！（2025.5.25，P4643 [国家集训队] 阿狸和桃子的游戏）

树上，一个点 x 到「树上离它最远点」的距离是：以直径中点为根（如果直径长度为奇数，则在两个中点的中间加一个虚点作为根，使得直径长度变为偶数），则为 x **的深度** $+\frac{1}{2}$ **直径**。（2025.5.28，from PPP）

处理矩阵乘法等不满足交换律的东西时要注意 operand 的顺序！不要想当然按照有交换律的情况做了。（2025.6.24，P4719 【模板】动态 DP）

如果 `a` 是一个数组，则 `int(a)`、`ll(a)` 这种强制类型转换会将 `a` 的地址转换为整数类型，且不会报 warning（即使开了 -Wall, -Wextra），且参与运算就跟普通的数一样。要小心不要把 `int(a[i])` 写成 `int(a)`。（2025.7.3，P1495 【模板】中国剩余定理（CRT）/ 曹冲养猪）

题目给的每个条件都是有用的，不要忽略任何条件。（2025.7.12，CF1355F Guess Divisors Count，做这题是我以为 $|\text{ans}-d|\le 7$ 的条件没用，但正是这个以为让我没能做出来）

第六章 (2025.7.15 ~ ?)

有唐龙对着代码静态看了一晚上无果，结果第二天随便写了个小数据就把自己代码 hack 了，并且这份唐龙代码在 AT 上过了 36/48 个点，怎么回事呢。

教训：不要执着于静态差错！

另外：这样求连通块边数是错的：

```
void dfs_pre(int x, int ek) {
  vis[x] = 1;
  for (auto &pi : E[x]) {
    int v = pi.first, id = pi.second;
    if (id == ek) continue;
    ++cnt_e;
    if (!vis[v]) dfs_pre(v, id);
  }
  return;
}
```

原因是非生成树边（也即返祖边）会被算 2 次。（2025.7.21，ARC045D みんな仲良し高橋君）

在新 ZROJ 上静态空间 MLE 时，如果空间并不是特别大使得其不会在编译期报错，那么会显示如下信息，显示为 RE：

```
main: error while loading shared libraries: libm.so.6: failed to map segment from shared object
```

（2025.7.27，ZR2093【21 ZR 联赛集训 day5】地图，这发提交）

PPP 遇到的编译器自动 inline 做局事件：

在 ZR2094 【21 ZR 联赛集训 day5】毛毛虫 中，

PPP 在一个函数（代码中的 `xb &operator +=`）内值传递一个比较大的结构体（代码中的 `xb b`）作为参数（没有使用引用），虽然他没有把 `xb &operator +=` 定义为 `inline`，但是由于编译器优化把 `xb &operator +=` 变成了 inline 的。

然后他进行了递归层数达到 4×10^5 的递归，每一层都调用了 `xb &operator +=`。因为这个函数是 inline 的，所以它的参数中的那个结构体 `xb b` 被存放在递归函数（即代码中的 `DFSr`）的栈空间里，每递归一层都会在栈空间里申请一个，并且执行完 `xb &operator +=` 的代码后也不会马上释放（因为存放于 `DFSr` 的栈空间）。

解决方案应该是不要在 `DFSr` 中调用 `xb &operator +=`（因为你不知道编译器会不会自动 inline），而是自己手动把 `xb &operator +=` 对应的流程在 `DFSr` 里展开掉。并且要存放参数中的 `xb b` 可以使用全局的一个 `xb` 来存放（不难发现这样做不会影响正确性，但是编译器很笨，非要每次调用时单独在栈空间里开一个 `xb` 存放，我们也没办法 /无奈）

完整事故代码：

```cpp
#include <bits/stdc++.h>
using har = unsigned long long;
char ending;
void *stackbeginning, *stackending;
struct xb {
  private:
    int siz;
    har x[63];
  public:
    void insert(har v) {
      for(int i = 62; i >= 0; i--)
        if(v >> i & 1)
          if(x[i] == 0)
            x[i] = v, siz++, i = 0;
          else
            v ^= x[i];
    }
    har calc() const {
      har res = 0;
      for(int i = 62; i >= 0; i--)
        if((res ^ x[i]) > res)
          res ^= x[i];
      return res;
    }
    xb &operator += (xb b) {
    //  stackbeginning = std::min<void *>(stackbeginning, &b),
    //  stackending = std::max<void *>(stackending, &b);
      if(siz < b.siz)
        std::swap(siz, b.siz), std::swap(x, b.x);
      for(int i = 0; i < 63; i++)
        if(b.x[i])
          insert(b.x[i]);
      return *this;
    }
    xb() : x {}, siz(0) {}
} o[409600];
int n, siz[409600], nt = 0;
har a[409600], ans[409600];
std::vector<int> g[409600];
char beginning;
void DFSo(int x, int F) {
  if(F)
    g[x].erase(std::find(g[x].begin(), g[x].end(), F));
  siz[x] = 1;
  for(int y : g[x])
    DFSo(y, x), siz[x] += siz[y];
  std::sort(g[x].begin(), g[x].end(), [](int a, int b) -> bool {
    return siz[a] > siz[b];
  });
}
xb res, u;
void calc(int x) {
  res.insert(a[x]);
  for(int y : g[x]) {
  //  stackbeginning = std::min<void *>(stackbeginning, &y),
  //  stackending = std::max<void *>(stackending, &y);
    calc(y);
  }
```

```
}
void *lastxbtop;
bool lastxbtopused = false;
void DFSr(int x, int o) {
  ans[x] = ::o[o].calc();
  if(g[x].empty())
    return u = xb(), u.insert(a[x]);
  int pre = ++nt, xbtop = nt;
  if(lastxbtop && !lastxbtopused)
    lastxbtopused = true, printf("%zd\n", (char *)(&xbtop) - (char *)(lastxbtop));
  else
    lastxbtop = &xbtop;
  stackbeginning = std::min<void *>(stackbeginning, &xbtop),
  stackending = std::max<void *>(stackending, &xbtop);
  res = ::o[o];
  for(int i = g[x].size() - 1; i > 0; i--)
    calc(g[x][i]), ::o[++nt] = res;
  if(g[x].size() > 1)
    DFSr(g[x][0], nt), nt--;
  else
    DFSr(g[x][0], o);
  ::o[pre] = u;
  for(int i = 1; i < g[x].size(); i++) {
  //  stackbeginning = std::min<void *>(stackbeginning, &o),
  //  stackending = std::max<void *>(stackending, &o);
    if(i + 1 < g[x].size())
      ::o[nt] += ::o[pre], DFSr(g[x][i], nt), ::o[pre] += u, nt--;
    else
      nt++, ::o[nt] = ::o[pre], ::o[nt] += ::o[o], DFSr(g[x][i], nt), ::o[pre] += u,
nt--;
  }
  u = ::o[pre], nt--, u.insert(a[x]);
}
int main() {
  char o;
  stackbeginning = stackending = &o;
  scanf("%d", &n);
  for(int i = 1; i ⩽ n; i++)
    a[i] = i;
  for(int i = 1, u, v; i < n; i++)
    u = i, v = i + 1, g[u].push_back(v), g[v].push_back(u);
  DFSo(1, 0), DFSr(1, 0);
  for(int i = 1; i ⩽ n; i++)
    printf("%llu\n", ans[i]);
    fprintf(stderr, "Static: %zd\nStack: %zd", &ending - &beginning, (char *)
(stackending) - (char *)(stackbeginning));
}
```

定义 `template` 并会生成多个签名的函数，内部最好不要定义 `static` 数组，否则定义一次就会多一个数组，很占空间！（2025.7.27）

最好把读入读完！否则多测时没读完可能会死。（2025.7.30）

注意，VSCode 里计算列数时，一个 Tab 的所占的列数计算为设置中规定的 Tab 的长度，如我使用的就是 4，但是编译器会将 Tab 所占的列数计算为 1，导致在报错时输出错误行号列号信息时会有偏差。

（2025.7.31，P8987 [北大集训 2021] 简单数据结构）

再次强调：sanitizer 不能查 `long long` 到 `int` 的隐式转换中发生的溢出问题！

记录一下鬼畜事件：

```
inline void upd(int u, int v, const tag_t &t) {
  Max(u, 1), Min(v, n + 1);
  if (u ⩾ v) return;
  return upd(1, 1, n + 1, u, v, t);
}

// 省略若干代码
ll l = max((stk_l - stk ⩾ 2 ? ce_dv(-(stk_l[-1].second - stk_l->second),
stk_l[-1].first - stk_l->first) : 1), 1ll);
sgt.upd(l, n + 1, tag_t(p.second, 0, 0));
```

其中 `l` 类型是 `long long`，可能会 $\geq 2^{\{31\}}$，但是在 `l` 被作为参数传入 `upd` 函数时，参数 `u` 的类型是 `int`，导致发生了从 `long long` 到 `int` 的隐式转换，导致了 WA，并且 sanitizer 没有报错。

本来我以为在 `upd` 函数内 `if (u ⩾ v) return;` 能够解决所有的 `u > n` 的问题，但是现在看来还要注意参数类型！

Tips：如果在调试过程中怀疑是没开 `long long` 导致的问题，可以全文替换 `int` 为 `long long`（除了 `int main()` 这里的 `int`），再测试代码看看输出是否正确。

（2025.8.1，P8987 [北大集训 2021] 简单数据结构，完整代码见 https://www.luogu.com.cn/record/228156438）

如果把信息放到一个 `struct` 内存储，那么由于内存对齐，占用的空间可能比分开存储的更大（特殊地，空结构体被规定必须占用 1Byte 的空间）。可以用 `sizeof` 运算符查看 `struct` 占用的空间大小。

因此，如果要卡空间的话，可以把 `struct` 内的信息分开存储。

（2025.8.20）

空间限制比较小的题要注意，不要因函数递归的栈空间导致 MLE。

（2025.8.30，P4117 [Ynoi2018] 五彩斑斓的世界）

一个节点里面存了多个信息，且一次需要使用里面的两个及以上的信息时，＜将节点存为 `struct` 并开一个 `struct` 数组＞比＜把每个信息分开来单独开一个数组＞要快，原因是内存访问连续。

（2025.8.31，P4119 [Ynoi2018] 未来日记）

使用 `~-(1ull << k)` 来求 $2^k - 1$ 时，要注意 k 必须 < 64，要不然左移时会溢出。

你可以使用 `~0ull >> (64 - k)` 来代替，不过这样在 $k = 0$ 时又会错了。

（2025.9.1，P5354 [Ynoi Easy Round 2017] 由乃的 OJ）

散块不要老想着暴力，有时散块也可以优化到 $< O(B)$ 的复杂度！

（2025.9.2，P5356 [Ynoi Easy Round 2017] 由乃打扑克）

由于内存对齐，

```
struct qry_t {
  int u; ll x; int id;
};
```

相较于

```
struct qry_t {
  int u, id; ll x;
};
```

会占用更多的空间。前者是 24Byte（相当于 3 个 `long long`），后者是 16Byte（相当于 2 个 `int` 和 1 个 `long long`）。

（2025.9.8，P7897 [Ynoi2006] spxmcq）

---

cjr2010 遇到的 g++ 9.3.0 O2 bug（待写）

调试时要对一切东西都抱有怀疑的态度！

（2025.9.11，ZR2543 分割）

写 WBLT 要注意，如果不加特判的话要保证平衡树不会为空，可以通过初始时令根节点为权值为 ∞ 的节点来避免；如果不想这样，也可以在 `ins` 和 `rmv` 函数里面加特判，具体地：

```
void ins(int &x, int val) { // 注意改成 int &x
  if (!x) return x = new_nde(val), void(); // 这一行
  if (tr[x].is_leaf()) {
    tr[x].u = new_nde(min(tr[x].val, val));
    tr[x].v = new_nde(max(tr[x].val, val));
    return pus(x);
  }
  if (val <= tr[tr[x].u].val) ins(tr[x].u, val);
  else ins(tr[x].v, val);
  return pus(x), bal(x);
}
void rmv(int &x, int fr, int val) { // 注意改成 int &x
  if (tr[x].is_leaf()) {
    if (!fr) return x = 0, void(); // 这一行
    tr[fr] = tr[tr[fr].u ^ tr[fr].v ^ x];
    return pus(fr);
  }
  if (val <= tr[tr[x].u].val) rmv(tr[x].u, x, val);
  else rmv(tr[x].v, x, val);
  return pus(x), bal(x);
}
```

另外，`get_rk` 函数也要判平衡树为空的情况。

并且如果不预先往平衡树里插入 ∞，则直接在叶子结点处返回 `1` 的 `get_rk` 函数在这样的情况下会出错：平衡树只有一个数 x，调用 `get_rk(`y`)`，且 $x < y$。这种情况理应返回 2，但是因为根是叶子结点所以返回 1。

正确的写法应当这样写：

```
int get_rk(int x, int val) {
  if (!x) return 1; // 这一行
  if (tr[x].is_leaf()) return int(tr[x].val < val) + 1; // 这一行
  if (val ⩽ tr[tr[x].u].val) return get_rk(tr[x].u, val);
  else return get_rk(tr[x].v, val) + tr[tr[x].u].sz;
}
```

（2025.9.15，P5314 [Ynoi2011] ODT）

有时候在 `vector` 为空时调用 `pop()` 会导致 `size()` 变为 $2^{64}-1$（也就是 `unsigned long` 下的 `-1`），但是此时有概率并不会 RE，而是留到下次调用 `back()` 或 `push()` 等时爆出来。但是如果开了 fsanitize，那么在 `size()` 变为 $2^{64}-1$ 的时候，就会报出来，错误信息类似如下：

```
/usr/include/c++/13/bits/stl_vector.h:1325:2: runtime error: applying non-zero offset
18446744073709551608 to null pointer
```

出错代码如下：（dly 的）

```
#include<bits/stdc++.h>
#define ll long long
#define fi first
#define se second
using namespace std;
#define getchar()(p1==p2&&(p2=(p1=buf)+fread(buf,1,1<<21,stdin),p1==p2)?EOF:*p1++)
char buf[1<<21],*p1=buf,*p2=buf;
void read(){}
template<typename T,typename...T2>
inline void read(T&x,T2&...oth)
{
    x=0;
    int ch=getchar(),f=0;
    while(ch<'0'||ch>'9'){if(ch=='-')f=1;ch=getchar();}
    while(ch⩾'0'&&ch⩽'9')x=(x<<1)+(x<<3)+(ch^48),ch=getchar();
    if(f)x=-x;
    read(oth...);
}
inline void write(ll x)
{
    static int t[25];int tp=0;
    if(x==0)return(void)(putchar('0'));else if(x<0)putchar('-'),x=-x;
    while(x)t[tp++]=x%10,x/=10;
    while(tp--)putchar(t[tp]+48);
}
const int N=5e5+10;
struct node{
    int x,y;
}a[N],q[N];
int n,m,M,tp1,tp2,b1[N],b2[N];
vector<int>vd[N],vq[N];
vector<ll>ans1[N],ans2[N];
```

```
priority_queue<ll>pq[10][N];
pair<ll,ll>st1[N],st2[N],t1[N<<2],t2[N<<2];
void build(int n)
{
    M=1;
    for(;M<n+1;M<<=1);
    for(int i=1;i<(M<<1);i++)t1[i]=t2[i]={-1e10,0};
    M++;
    return;
}
void modify(pair<ll,ll>t[N<<2],int op,int p,int l,int r,int v,ll k)
{
    cout<<123<<" "<<op<<" "<<v<<" "<<k<<endl;
    pq[op][v].push(k);
    cout<<234<<endl;
    // return;
    // cout<<k<<endl;
    p=v+M;
    t[p]={pq[op][v].top(),v};
    for(int i=p>>1;i;i>>=1)t[i]=max(t[i<<1],t[i<<1|1]);
    // cout<<345<<endl;
    return;
}
pair<ll,ll>query(pair<ll,ll>t[N<<2],int p,int lx,int rx,int l,int r)
{
    if(l>r)return {-1e10,0};
    pair<ll,ll>res={-1e10,0};
    for(l=l+M-1,r=r+M+1;l^r^1;l>>=1,r>>=1)
    {
        if(!(l&1))res=max(res,t[l^1]);
        if(r&1)res=max(res,t[r^1]);
    }
    return res;
}
void del(pair<ll,ll>t[N<<2],int op,int p,int l,int r,int v)
{
    pq[op][v].pop(),p=v+M;
    if(pq[op][l].empty())t[p]={-1e10,l};
    else t[p]={pq[op][v].top(),l};
    for(int i=p>>1;i;i>>=1)t[i]=max(t[i<<1],t[i<<1|1]);
    return;
}
void solve(int s)
{
    for(int i=1;i<=n;i++)a[i].x*=s,b1[i]=a[i].x,b2[i]=a[i].y;
    sort(b1+1,b1+1+n),sort(b2+1,b2+1+n);
    int len1=unique(b1+1,b1+1+n)-b1-1,len2=unique(b2+1,b2+1+n)-b2-1;
    for(int i=1;i<=n;i++)a[i].x=lower_bound(b1+1,b1+1+len1,a[i].x)-
b1,a[i].y=lower_bound(b2+1,b2+1+len2,a[i].y)-b2,vd[a[i].x].push_back(a[i].y);
    for(int i=1;i<=m;i++)vq[a[q[i].x].x].push_back(i);
    build(len2);
    cout<<"M:"<<M<<endl;
    for(int i=1;i<=len2;i++)
    {
        while(!pq[1][i].empty())pq[1][i].pop();
        while(!pq[2][i].empty())pq[2][i].pop();
    }
    for(int i=1,x,y,k,cnt=0;i<=len1;i++)
```

```
    {
        if(s>0)
        {
            for(int y:vd[i])cnt+
+,modify(t1,1,1,1,len2,y,b1[i]+b2[y]),modify(t2,2,1,1,len2,y,b1[i]-b2[y]);
            for(int id:vq[i])
            {
                y=a[q[id].x].y,k=q[id].y;
                for(int j=1;j<=min(k,cnt);j++)
                {
                    auto it1=query(t1,1,1,len2,1,y),it2=query(t2,1,1,len2,y+1,len2);
                    if(b1[i]+b2[y]-it1.fi<=b1[i]-b2[y]-
it2.fi)ans1[id].push_back(b1[i]+b2[y]-it1.fi),st1[+
+tp1]=it1,del(t1,1,1,1,len2,it1.se);
                    else ans1[id].push_back(b1[i]-b2[y]-it2.fi),st2[+
+tp2]=it2,del(t2,2,1,1,len2,it2.se);
                }
                while(tp1)modify(t1,1,1,1,len2,st1[tp1].se,st1[tp1].fi),tp1--;
                while(tp2)modify(t2,2,1,1,len2,st2[tp2].se,st2[tp2].fi),tp2--;
            }
        }
        else
        {
            for(int id:vq[i])
            {
                y=a[q[id].x].y,k=q[id].y;
                for(int j=1;j<=min(k,cnt);j++)
                {
                    auto it1=query(t1,1,1,len2,1,y),it2=query(t2,1,1,len2,y+1,len2);
                    if(b1[i]+b2[y]-it1.fi<=b1[i]-b2[y]-
it2.fi)ans2[id].push_back(b1[i]+b2[y]-it1.fi),st1[+
+tp1]=it1,del(t1,1,1,1,len2,it1.se);
                    else ans2[id].push_back(b1[i]-b2[y]-it2.fi),st2[+
+tp2]=it2,del(t2,2,1,1,len2,it2.se);
                }
                while(tp1)modify(t1,1,1,1,len2,st1[tp1].se,st1[tp1].fi),tp1--;
                while(tp2)modify(t2,2,1,1,len2,st2[tp2].se,st2[tp2].fi),tp2--;
            }
            for(int y:vd[i])cnt+
+,modify(t1,1,1,1,len2,y,b1[i]+b2[y]),modify(t2,2,1,1,len2,y,b1[i]-b2[y]);
        }
    }
    for(int i=1;i<=n;i++)a[i].x=b1[a[i].x],a[i].y=b2[a[i].y];
    for(int i=1;i<=len1;i++)vd[i].clear(),vq[i].clear();
    return;
}
int main()
{
    read(n,m);
    for(int i=1,x,y;i<=n;i++)read(x,y),a[i].x=x+y,a[i].y=x-y;
    for(int i=1;i<=m;i++)read(q[i].x,q[i].y),q[i].y++;
    solve(1),solve(-1);
    for(int i=1;i<=m;i++)
    {
        long long sum=0;
        for(int j=1,t1=0,t2=0;j<=q[i].y;j++)
        {
            if(t1==ans1[i].size())sum+=ans2[i][t2++];
```

```
            else if(t2==ans2[i].size())sum+=ans1[i][t1++];
            else if(ans1[i][t1]<ans2[i][t2])sum+=ans1[i][t1++];
            else sum+=ans2[i][t2++];
        }
        write(sum>>1),putchar('\n');
    }
    return 0;
}
```

输入数据为：

```
5 5
1 2
2 3
1 5
3 6
4 1
1 2
3 4
2 1
5 3
4 2
```

（2025.9.16，ZR3335 战舰一定要能开！）

对于决策单调性的题，如果 dp 值是实数，那么不能使用取整后的 dp 值来进行比较！

（2025.9.19，P3515 [POI 2011] Lightning Conductor）

矩阵树定理求的是 L 矩阵的余子式而非行列式！

（2025.9.22，AT_agc051_d [AGC051D] C4）

二分的时候，要注意 `l + r` 会不会爆 `int` 或 `long long`！或者也可以用 `l + ((r - l) >> 1)` 代替 `(l + r) >> 1` 来规避这个问题。

（2025.9.24，帮 dly 调 P5356 [Ynoi Easy Round 2017] 由乃打扑克 的 这发代码，她这个地方写挂了）

用并查集维护「删除序列中的点、查询后继」的操作时，要注意要把 `fa[n + 1]` 设为 $n+1$，否则在没有后继的时候会错！

（2025.9.24，ZR2541 食物）

递推式不仅仅是递推式，还要从方程的视角看待它。

（2025.10.3，P7481 梦现时刻）