

こちらの過去を

筆者の心情は
を持ちながら



Note

LA 著

邯郸市第一中学

还记得当年和孙某一起做一道题

啊当时我们还是初三呢他还学着数学竞赛

只记得我们证了很长时间很长时间

最后解答里面奇妙的证明让我们相兴而去

也许这就是结束了

唉如果当时我知道

知道有一天自己也可以把它解出来

我要用最酷炫的方法最通俗的证明

让初三的我知道他也可以

如果他能穿越时空也会很欣慰吧

我们都不知不觉的有了自己的进步

头上的天空却看上去还是灰蒙蒙一片

那便走下去吧

走到天空变蓝

站在巨人的肩膀上

你是否能听到我呢

——《Note》序

(一)

那天，月色浸染的栏杆边。
你把这本书递给我。
你的身形，藏匿于黑暗之中，唯有眼睛愈发透亮，清澈如斯。
空气静的，只能听到你我二人早已超出负荷的心跳，风轻轻的与你呼应着。
“你是否能听到我呢... ..”
这便是最浪漫的告白。

(二)

我依然不觉得她只是一本书。
她是一个故事，或者说，是友谊，爱情，理想，热爱等一切有关于“他的青春”的一切故事的结合。那是一个少年在黑暗中发光的心藏，也是一个少年，在静夜中肆意飞舞的灵魂。
这绝不仅是一本关于 OI 的书，透过这本书，我们能听到他，听到作者的青春，他的故事，他的爱与渴望。

(三)

她有一个名字，叫做时间。
... ..

(四)

去吧，用最酷的方法证明。
不只一道道题面，还有你。

2023 年 5 月 13 日
分鱼

自序

「数据删除」

2023 年 10 月 17 日
编者

目录

第一部分 算法与数据结构	1
第 1 章 渐进时间复杂度	2
1.1 基本定义	2
1.2 主定理	2
1.3 势能分析	3
1.4 几种有趣的复杂度分析	3
第 2 章 常见算法及数据结构	4
2.1 哈希算法	4
2.1.1 字符串哈希	4
2.1.2 树哈希	4
2.2 区间数据结构	5
2.2.1 ST 表	5
2.2.2 树状数组	6
2.2.3 分块	7
2.3 平衡树	8
2.3.1 二叉搜索树 (BST)	8
2.3.2 带旋 Treap	8
2.3.3 FHQ-Treap	10
2.3.4 Splay	13
2.4 左偏树	17
第 3 章 抽象数据结构	22
第二部分 动态规划	23
第 4 章 动态 DP	24
第 5 章 DP 优化技巧	26
5.1 长链剖分优化	26
第三部分 图论	28
第 6 章 图论基础	29
6.1 最短路问题	29
6.1.1 Dijkstra	29
6.1.2 Bellman-Ford	29
6.1.3 Floyd	30
第 7 章 点分治	31
7.1 静态点分治	31

第 8 章 虚树	34
第 9 章 支配树	36
第 10 章 网络流	38
10.1 常见建模技巧	38
第四部分 字符串	40
第 11 章 后缀数据结构	41
11.1 后缀自动机	41
11.1.1 endpos 等价类	41
第五部分 数学	44
第 12 章 数论基础	45
12.1 基本定义和一个求和技巧	45
12.1.1 基本定义	45
12.1.2 欧几里得算法	45
12.1.3 类欧几里得算法初探	46
12.1.4 一个求和技巧	47
12.2 素数	48
12.2.1 唯一分解定理	48
12.2.2 素数的个数	49
12.3 同余	50
12.3.1 基本定义	50
12.3.2 费马小定理	50
12.4 Miller-Rabin 素性测试	50
12.5 Pollard's rho 算法	52
12.6 数论函数与 Möbius 反演	53
12.7 容斥原理与二项式反演	57
12.7.1 容斥原理	57
12.7.2 二项式反演	58
第 13 章 组合变换技巧	60
13.1 组合恒等式	60
13.1.1 基本恒等式	60
13.1.2 求和一类组合恒等式	61
13.1.3 二项式定理	61
13.1.4 负数上标下的二项式系数	62
13.1.5 多系数组合恒等式	63
13.2 组合技巧与常用结论	64
13.3 Stirling 数	66
第 14 章 生成函数	70
14.1 普通生成函数及其应用	70

第 15 章 代数结构	74
15.1 二元关系	74
15.1.1 笛卡尔积与二元关系	74
15.1.2 关系的性质	75
15.1.3 等价关系与划分	75
15.1.4 偏序关系	76
15.2 代数系统	77
15.2.1 二元运算	77
15.2.2 代数系统	78
15.3 群论基础	79
15.3.1 基本定义	79
15.3.2 子群	80
15.3.3 群的陪集	81
15.3.4 循环群与置换群	83
15.4 Burnside 引理与 Pólya 定理	84
第 16 章 多项式	87
16.1 前置知识	87
16.2 离散傅里叶变换	87
16.2.1 多项式表示法	87
16.2.2 表示间的转换	87
16.3 分治 FFT	88
第六部分 杂项	90
第 17 章 离线算法	91
17.1 莫队	91
第 18 章 分散层叠算法	92
18.1 简单叙述	92
18.2 一般情况及应用	93
第七部分 从各处搞来的 trick	95
第八部分 杂题选做	97

第一部分

算法与数据结构

第 1 章 渐进时间复杂度

1.1 基本定义

时间复杂度是衡量算法优劣程度的一个重要标准，它意味着算法在**最坏情况**下的运行速度。

假如我们要计算 $\sum_{i=1}^n i$ ，一种简单粗暴的方式是直接写一层循环，注意到，如果这样写，我们一共就要进行 n 次加法，当 n 不断增长的同时，我们的运算次数也会随着线性增长，所以我们说，这个算法的时间复杂度是 $\mathcal{O}(n)$ 的。

这时如果我们的算法进行了两次这样的操作，我们就要进行 $2n$ 次加法，它的时间复杂度是多少？ $\mathcal{O}(2n)$ 吗？没有什么问题，但是仔细想想就会发现，当 n 大到一定程度的时候， $n, 2n, 3n, \dots$ 的区别实际上不大，举个例子，当 n 从 10^8 变到 10^{10} ，常数 2 的影响相对于 n 来说是微乎其微的，事实上，我们在说明时间复杂度的时候，一般省去常数，也就是说 $\mathcal{O}(n), \mathcal{O}(2n), \mathcal{O}(3n)$ 等等都记作 $\mathcal{O}(n)$ 。

还有一种方法是公式计算。小学的时候我们就学过等差数列的求和公式， $\sum_{i=1}^n i = \frac{1}{2}n(n+1)$ ，这次，无论 n 有多大，我们只需要计算三次乘法和一次加法（或者说，一次加法，一次乘法和一次除法）即可，**我们可以认为，单次的加、减、乘、除、取模以及位运算都是 $\mathcal{O}(1)$ 的**，于是上面算法的时间复杂度就是 $\mathcal{O}(1)$ ，这表明程序的运行速度和 n 没有任何关系。

还有一个问题是：假如程序的时间复杂度是 $\mathcal{O}(n^2 + n + 1)$ ，注意到当 n 大到一定程度时， n 相对于 n^2 来说是完全可以忽略不计的，这时，我们可以像省去常数一样，把 n 省去，于是上面的时间复杂度就记作 $\mathcal{O}(n^2)$ 。若读者了解过极限，就可以说是由于

$$\lim_{n \rightarrow +\infty} \frac{n}{n^2} = 0$$

了。类似的，假如时间复杂度中含有对数函数，我们是不写底数的，这是因为我们总可以用换底公式将不同的底数变为常数倍的差距，因此直接记作 $\mathcal{O}(\log n)$ 。

在最开始，我们已经说过这是衡量算法在最坏情况下的运行速度。具体来说，假如现在有一种算法能在平均 $\mathcal{O}(n)$ 的时间内解决某个问题，但是在某些特殊情况下，它的时间复杂度是 $\mathcal{O}(n^2)$ ，我们就不能说它的时间复杂度是 $\mathcal{O}(n)$ 了，尽管这些特殊情况的条件可能很苛刻，但它们确实存在，我们就只能说算法的时间复杂度是 $\mathcal{O}(n^2)$ 。

1.2 主定理

在算法中，主定理 (Master Theorem) 是求解递归时间复杂度的常用方法。在下面的定理中， $T(n)$ 指算法在数据规模为 n 时的运算次数。

定理 1.1 (主定理)

若

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \quad \forall n > b$$

则

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & f(n) = \mathcal{O}(n^{\log_b a - \epsilon}) \\ \Theta(f(n)) & f(n) = \Omega(n^{\log_b a + \epsilon})^a \\ \Theta(n^{\log_b a} \log^{k+1} n) & f(n) = \Theta(n^{\log_b a} \log^k n)^b, k \geq 0 \end{cases}$$

^a Ω 与 \mathcal{O} 恰好相反，它表示程序运行速度的下界

^b Θ 表示程序运行速度的上界 + 下界



例如, $T(n) = 2T(\frac{n}{2}) + n$, 因为 $a = 2, b = 2, \log_2 2 = 1$, 那么 $k = 0$ 可以满足情况 3, 因此 $T(n) = \Theta(n \log n)$

1.3 势能分析

势能分析是求均摊复杂度上界的有力工具, 关键在于“状态”到“数”的正确映射.¹

定义状态 S 为某一时刻的所有数据, S_0 为初始状态. 现在假如我们有一个从状态映射到数的函数 F 使得 $\forall S, F(S) \geq F(S_0)$, 那么我们有如下推论:

设 S_1, S_2, \dots, S_m 为从 S_0 开始连续做 m 次操作所得的操作序列. 设 c_i 为第 i 次操作的时间开销. 再记 $p_i = c_i + F(S_i) - F(S_{i-1})$, 则 m 次操作的总时间花销为

$$\sum_{i=1}^m p_i + F(S_0) - F(S_m)$$

又因为 $F(S) \geq F(S_0)$, 所以

$$\sum_{i=1}^m p_i \geq \sum_{i=1}^m c_i$$

因此, 若 $p_i = \mathcal{O}(T(n))$, 则 $\mathcal{O}(T(n))$ 是均摊复杂度的一个上界.

1.4 几种有趣的复杂度分析

单调队列 (栈) 的复杂度是 $\mathcal{O}(n)$, 因为每个元素最多出栈一次、入栈一次, 因此内层的循环就是 $\mathcal{O}(n)$ 的.

KMP 算法构建 next 数组的时间复杂度是 $\mathcal{O}(n + m)$, 这是因为转移时有三种情况 (下文中的变量意义与本书中对 KMP 的介绍相同):

- 两字符匹配, $i++$, $j++$
- 两字符不匹配且 $j = 0$, $i++$
- 两字符不匹配且 $j \neq 0$, 此时, j 最多回跳 $|j|$ 次

可以发现, 前两种是 $\mathcal{O}(1)$ 转移, 而回跳过程的时间复杂度可以均摊到让 j 自增的那部分 $\mathcal{O}(1)$ 上, 于是总的时间复杂度就是 $\mathcal{O}(n + m)$ 了.

带懒标记的线段树查询是 $\mathcal{O}(\log n)$ 的, 这是因为有 $\mathcal{O}(\log n)$ 层而每层至多访问 4 个结点.

¹<https://oi-wiki.org/basic/complexity/>

第 2 章 常见算法及数据结构

2.1 哈希算法

2.1.1 字符串哈希

哈希 (Hash) 简单来说就是一个映射, 一般是到数的映射, 例如将字符串映射到数.

显然, 假如值域是有限的 (例如, `int` 范围内) 而字符串有无限个, 那么必然会有某些不同的串的哈希值 (也就是映射后的结果) 相同, 我们叫它**冲突**或**哈希碰撞**.

一个优秀的哈希算法应该满足冲突的概率尽可能小, 并且可以在能够接受的时间复杂度内算出某个字符串 (或者是其他东西) 的哈希值.

OI 内最常用的字符串哈希算法是将字符串看作一个 B 进制数字, 其中 B 按照经验取 131 或 13331, 当然, 你也可以取其他数字, 但最好是奇质数.

按照这样的算法, $Hash(xya) = B^2x + By + a$, 乘一个字母的时候直接用它的 `ascii` 码就行, 也就是在 `c++` 里直接将 `char` 转为 `int`, 由于这个数字很大, 所以我们会将其对一个大质数取模, 例如 $10^9 + 7$.

字符串哈希的最大作用 (甚至可以说是唯一作用) 就是快速判断两个字符串是否相等, 显然, 两个串的哈希值相等是两个串相等的必要条件, 当哈希函数设计的比较好 (冲突的概率很小时), 我们就可以认为这是一个充分条件, 也就是说, 我们可以直接通过判断两个串的哈希值是否相等来断定两个串是否相同.

如何简单分析哈希碰撞的概率? 我们可以简单的认为每次算哈希都是在值域范围内均匀随机地取一个值, 那么根据生日悖论, 设 M 是值域, 那么 n 个不同的串遇到哈希碰撞的概率是

$$1 - \prod_{i=0}^{n-1} \frac{M-i}{M}$$

或者说, 在随机数据下, 若取模数在 10^9 范围内, 那么 n 大于 \sqrt{M} 时就已经极容易发生碰撞了, 这是十分危险的.

怎么降低碰撞的概率呢? 我们可以取两个**大质数**分别做哈希, 两个串的两组哈希值均相等才说这两个串相等, 这样的值域可以看做是两个模数的乘积, 于是就可以将值域放大到 10^{18} 级别了.

2.1.2 树哈希

uoj#763. 树哈希

这是一道模板题。

给定一棵以点 1 为根的树, 你需要输出这棵树中最多能选出多少个互不同构的子树。

两棵有根树 T_1 、 T_2 同构当且仅当他们的大小相等, 且存在一个顶点排列 σ 使得在 T_1 中 i 是 j 的祖先当且仅当在 T_2 中 $\sigma(i)$ 是 $\sigma(j)$ 的祖先。

对于所有数据, $1 \leq n \leq 10^6$ 。

Solution: uoj#763. 树哈希

定义以 u 为根的子树的哈希值为 $h(u) = 1 + \sum_{v \in \text{son}_u} f(h(v))$, 其中 f 是一个随机函数 (例如 `xor shift`), 在模大质数意义下或在自然溢出意义下计算¹.

两个子树同构当且仅当它们的哈希值相同, $\mathcal{O}(n)$ dfs 一遍算出来所有 `hash` 值即可, 假如用 `std::set` 去重, 总时间复杂度 $\mathcal{O}(n \log n)$.

¹<https://peehs-moorhsum.blog.uoj.ac/blog/7891>

2.2 区间数据结构

2.2.1 ST 表

普通的 ST 表是用来在 $\mathcal{O}(n \log n) - \mathcal{O}(1)$ 的时间内解决静态区间极值问题的数据结构。

现在给出一个长度为 n 的序列 A 和 m 次询问，每次询问给定在区间 $[1, n]$ 的两个数 l, r ，要求输出 $\max_{i=l}^r A_i$ 。直接暴力做是 $\mathcal{O}(nm)$ 的，考虑通过预处理来加速询问。

ST 表可以成功得出答案的关键在于值的可重复贡献，举个例子来说，

$$\max\{A_1, A_2, A_3\} = \max\{\max\{A_1, A_2\}, \max\{A_2, A_3\}\}$$

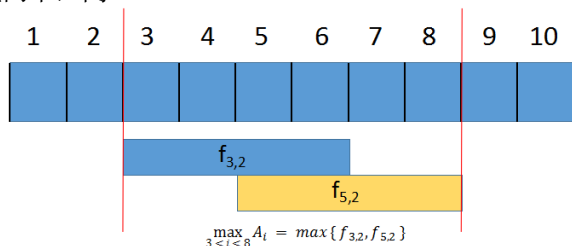
在这里， A_2 被重复计算了两次，但这种重复并不会对答案（最大值）产生影响，于是我们可以将一个大区间拆成两个小区间，而这两个小区间是可以有重叠的部分的。ST 表采取了倍增的思路拆分区，具体来说，我们设 $st_{i,j}$ 表示以序列的第 i 个数为左端点的，长度为 2^j 的区间的最大值，转移如下：

$$st_{i,0} = A_i$$

$$st_{i,j} = \max\{st_{i,j-1}, st_{i+2^{j-1},j-1}\}$$

解释一下就是，一个长度为 2^j 的区间的最大值可以用两个长度为 2^{j-1} 的区间的最大值拼起来。

但是，随便给的一个区间的长度又不一定恰好是 2^j ，这时怎么办呢？用我们刚才说的“可重复贡献”，将它拆成两个区间：



在上面的例子中，我们要知道 A_3, \dots, A_8 的最大值，而我们已经分别知道了 A_3 到 A_6 的最大值和 A_5 到 A_8 的最大值，这两个区间完全覆盖了询问的区间，直接取最大值即可。容易发现，每一个长度不为 2^j 的区间都可以这样拆成两个小区间后合并得到，对于拆成的两个小区间，它们的长度都是 2^j ，其中， $2^j \leq len < 2^{j+1}$ (len 为询问区间的长度)，因为假如 $len < 2^j$ ，一个小区间就超出了询问的长度；假如 $len \geq 2^{j+1}$ ，那么两个长度为 2^j 的区间一定不能完全覆盖询问的区间 ($2^j + 2^j = 2^{j+1} < len$)，因此这里的 j 就等于 $\lfloor \log_2 len \rfloor$ ，我们可以用 c++ 内置的函数 `__lg(len)` 来得到 j 。

```

1 inline void pre() {
2     for(int i = 1; i <= n; i++) st[i][0] = A[i];
3     for(int j = 1; (1 << j) <= n; j++)
4         for(int i = 1; i + (1 << j) - 1 <= n; i++)
5             st[i][j] = max(st[i][j-1], st[i + (1 << (j-1))][j-1]);
6 }
7 inline int ask(int l, int r) {
8     int j = __lg(r - l + 1);
9     return max(st[l][j], st[r - (1 << j) + 1][j]);
10 }

```

区间 min 的实现几乎完全相同（直接把代码中的 `max` 换为 `min`）即可。在上述代码中，预处理的时间复杂度是 $\mathcal{O}(n \log n)$ ，单次询问的时间复杂度为 $\mathcal{O}(1)$ 。

2.2.2 树状数组

树状数组，又称 Fenwick 树或 BIT，是用来解决区间一类问题的另一工具。

给定一个长度为 n 的序列 A ，有 m 次操作，每个操作形如下面的其中一种：

- 给出 p, x ，让 A_p 变为 $A_p + x$ 。
- 给出 l, r ，查询 A_l 到 A_r 的和。

假如没有第一种操作，我们可以用前缀和解决，将 A_l 到 A_r 的和转化为 A_1 到 A_r 的和减去 A_1 到 A_{l-1} 的和。假如没有第二种操作，我们可以直接暴力修改。

但是，假如我们用了前缀和后，修改一个点上的值是 $\mathcal{O}(n)$ 的（我们修改 A_1 的值，那么所有前缀的和都会改变），时间复杂度退化为 $\mathcal{O}(nm)$ 。假如我们不用前缀和，那么第二种操作就是 $\mathcal{O}(n)$ 的，同样不行。

树状数组可以在 $\mathcal{O}(\log n)$ 的时间内实现操作 1 或操作 2，那么总的时间复杂度就是 $\mathcal{O}(m \log n)$ 了，优秀了很多。

和前缀和的查询一样，我们将区间的和变为两个前缀和的差，于是第二种操作可以看成是查询两个前缀和。下面假定我们要查询的前缀和是 $A_1 + A_2 + \dots + A_r$ 。首先将 r 写成二进制，比如 $r = (10110)_2$ ，假如我们可以知道 $A_{10101} + A_{10110}$ ， $A_{10001} + \dots + A_{10100}$ ， $A_{00001} + \dots + A_{10000}$ 的值，那 $A_1 + \dots + A_r$ 的值就是上面三个值的和。

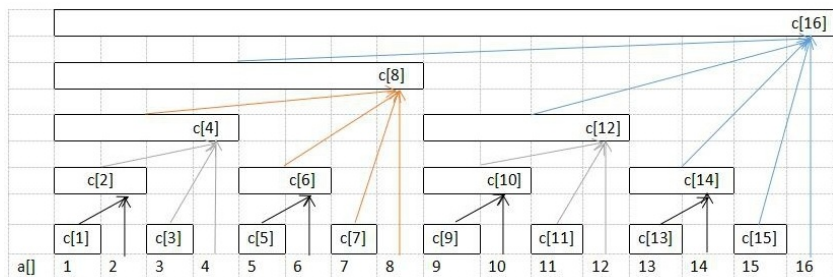
更抽象一点，我们引入一种操作叫做 lowbit ， $\text{lowbit}(x)$ 表示 x 的二进制中最后一位 1 所占的位置表示的数，比如， $\text{lowbit}(6) = 2$ ，因为 $6 = (110)_2$ ，所以 $\text{lowbit}(6) = (010)_2 = 2$ ，再比如 $\text{lowbit}(9) = 1$ ，因为 $9 = (1001)_2$ ，所以 $\text{lowbit}(9) = (0001)_2 = 1$ 。

那我们刚才的拆分原理实际上是这样的，先把 $[1, (10110)_2]$ 拆成 $[1, (10100)_2] + [(10101)_2, (10110)_2]$ ，而 $(10100)_2$ 其实就是 $(10110)_2 - \text{lowbit}((10110)_2)$ 。

换句话说，我们把区间 $[1, r]$ 拆成了 $[1, r - \text{lowbit}(r)] + [r - \text{lowbit}(r) + 1, r]$ ，然后接着用相同的方法拆 $[1, r - \text{lowbit}(r)]$ 。

每次我们将 r 变为 $r - \text{lowbit}(r)$ ，它的二进制的最后一个 1 会变成 0，其他位置不变，那么我们一直这样减下去， r 最终一定会变成一个形如 $(100\dots 00)_2$ 的数，并且最多减 $\log_2 r$ 次（因为 r 的二进制有 $\log_2 r$ 位，也就最多有 $\log_2 r$ 个 1 可以减），同时也就是说，我们最多把区间 $[1, r]$ 拆成 $\mathcal{O}(\log n)$ 个区间。

我们不直接存储每个 A_i 的值，而是维护一个新的数组 C_i ，而其中， $C_i = \sum_{j=i-\text{lowbit}(i)+1}^i A_j$ 。



假如我们已经用某种方法维护好了 C 数组，那么查询 $\sum_{i=1}^r A_i$ 就可以这么写：

```
1 inline int ask(int r) {
2     int res = 0;
3     for(int i = r; i > 0; i -= lowbit(i)) res += C[i];
4     return res;
5 }
```

这是 $\mathcal{O}(\log n)$ 的。再次回顾上面的那张图，假如我们让 A_p 变为了 $A_p + x$ ，有哪些 C_i 需要被改变呢？首先 C_p 是肯定要加上 x 的，其次 $C_{p+\text{lowbit}(p)}$ 也应该加上 x ，然后一直这么加下去，直到 p 大于 n 。假如 $p = 5$ ，我们要改变的实际上就是 C_5, C_6, C_8, C_{16} ，而 $6 = 5 + \text{lowbit}(5)$ ， $8 = 6 + \text{lowbit}(6)$ ， $16 = 8 + \text{lowbit}(8)$ ，和我们刚才的说法吻合。

于是让 A_p 变为 $A_p + x$ 的代码就可以这样写:

```
1 inline void add(int p, int x) {
2     for(int i = p; i <= n; i += lowbit(i)) C[i] += x;
3 }
```

这同样是 $\mathcal{O}(n)$ 的. 两种操作都得到了解决, 时间复杂度均为 $\mathcal{O}(\log n)$.

但其实还有一个问题, 就是怎么快速地求 $\text{lowbit}(x)$. 根据计算机存储的特性, 可以得出这样一个结论:

$$\text{lowbit}(x) = x \ \& \ -x$$

```
1 inline int lowbit(int x) {return x & -x; }
```

2.2.3 分块

相对设计精密的树状数组, 分块是一种更暴力的算法, 有着更简单的思想, 但时间复杂度从 polylog 变成了根号.

先以单点加、区间询问和为例, 我们将整个数组分为 \sqrt{n} 块, 每块的大小为 \sqrt{n} , 例如当 $n = 10$ 时, a_1, a_2, a_3 在第一块, a_4, a_5, a_6 在第二块, a_7, a_8, a_9 在第三块, a_{10} 在第四块.

预处理出每一块的左边界和右边界以及每个位置在哪一块, 在每个块上维护块中元素的和.

单点加法时, 将原数组改动后维护所在块的元素和, 这是 $\mathcal{O}(1)$ 的.

区间查询时, 假如区间的两个端点在同一个块, 直接暴力, 时间复杂度 $\mathcal{O}(\sqrt{n})$, 否则将整个区间拆分为至多 \sqrt{n} 个整块和 $\mathcal{O}(1)$ 个散块, 对于那些整块, 直接用维护好的元素和加起来 ($\mathcal{O}(\sqrt{n})$), 散块暴力 ($\mathcal{O}(\sqrt{n})$). 还以 $n = 10$, 块大小为 3 举例. 假如我们要查询区间 $[2, 7]$ 的和, 将这个区间拆分为 $[2, 3]$, $[4, 6]$ 和 $[7, 7]$, 一个完整的块 ($[4, 6]$) 和两个“块中的一部分” ($[2, 3]$ 和 $[7, 7]$). 对于完整的块, 我们已经维护好了块中元素的和, 直接取出来是 $\mathcal{O}(1)$ 的, 由于至多有 $\mathcal{O}(\sqrt{n})$ 个块, 加起来就是 $\mathcal{O}(\sqrt{n})$ 的. 对于那两个散块, 由于块大小为 \sqrt{n} , 直接暴力遍历这些元素还是 $\mathcal{O}(\sqrt{n})$ 的, 两部分加起来就是 $\mathcal{O}(\sqrt{n})$.

因此我们就得到了 $\mathcal{O}(1)$ 单点修改, $\mathcal{O}(\sqrt{n})$ 区间求和的分块算法.

代码实现是容易的, 我们接着看区间加法.

区间加法和区间查询是类似的, 暴力地在每个整块上打标记, 散块直接加, 这还 $\mathcal{O}(\sqrt{n})$ 的.

类似这样, 我们还可以实现区间推平、区间极值等问题. 分块本身并不困难, 但很多地方都可以用分块的思想解决问题, 例如我们之后要讲解的莫队算法.

【Ynoi2017】由乃打扑克

给定长度为 n 的序列 a 和 m 次操作, 每次操作形如:

1. 给定 l, r, k , 询问区间 $[l, r]$ 的 k 小值.
2. 给定 l, r, k , 让区间 $[l, r]$ 中的每个元素都加上 k .

$1 \leq n, m \leq 10^5$, 操作 2 中的 k 和初始序列 a 中的元素的绝对值不超过 2×10^4

Solution: 【Ynoi2017】由乃打扑克

将每一个元素变为有序对 $\langle a_i, i \rangle$ 后将序列分块, 每个块内按照第一关键字 (a_i) 排序, 预处理的时间复杂度为 $\mathcal{O}(n \log n)$.

对于区间加操作, 整块打标记, 每个散块暴力加之后, 为了维护块内升序, 归并不变的元素和加值的元素这两个有序数列, 总时间复杂度 $\mathcal{O}(\sqrt{n})$.

对于区间询问 k 小值, 将区间两端的散块归并为一个有序数列后二分答案, 每次 check 只需要得到区间小于等于二分值的元素个数.

这可以 $\mathcal{O}(\sqrt{n} \log n)$ 得到，因为我们已经将整个序列变成了 $\mathcal{O}(\sqrt{n})$ 个单调不降的小序列，在每个小序列（长度为 \sqrt{n} ）上二分后将个数加起来即可，于是区间询问 k 小值的时间复杂度为 $\mathcal{O}(\sqrt{n} \log n \log w)$ ，其中 w 为值域范围大小。

总时间复杂度 $\mathcal{O}(m\sqrt{n} \log n \log w)$ ，有点卡常数，实测块大小设为 1500 可过。

2.3 平衡树

平衡树是一类满足 BST(二叉搜索树) 性质的平衡的树状数据结构。

2.3.1 二叉搜索树 (BST)

定义 2.1 (BST)

我们定义满足如下性质的点带权树是 BST:

- 空树是 BST.
- 对于一棵非空树，任取一结点 u ，其左子结点（如果存在）的权值 $w_l < w_u$ ，右子结点（如果存在）的权值 $w_r > w_u$ ，则称该树满足 BST 性质。

BST 有一些比较好的性质，如：

- 中序遍历即排序后结果.
- 最左端的结点即最小值.
- 对称地，最右端的结点即最大值.

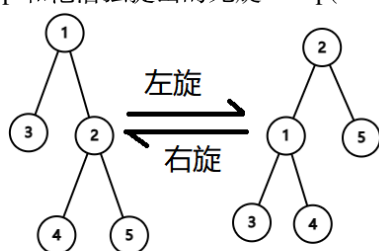
在 BST 中，我们可以很方便地查询某个值是否存在、是第几大、插入、删除结点，这些操作都并不复杂，自顶往下递归即可。设树高是 h ，那么上述操作均是 $\mathcal{O}(h)$ 的，理想情况下， h 是 $\log n$ 级别的（ n 为结点数），但它最坏却达到了 $\mathcal{O}(n)$ ，此时 BST 退化为了——一条链。怎样让树高达到优雅的 $\mathcal{O}(\log n)$ 呢？我们接下来讨论一类平衡树，它们引入了“旋转”操作和另外的权值解决这个问题，名字叫 Treap。

2.3.2 带旋 Treap

Treap，即“Tree”+“Heap”（堆），是一类特殊的平衡树，同时满足堆性质和 BST 性质。没有了解过的读者可能会产生疑惑，堆性质和 BST 在通常情况下是完全矛盾的，怎么能同时满足呢？实际上，我们对于每个结点附加了一个新的信息 rank，它的值是 **完全随机** 的，对于真实的值，Treap 满足 BST 性质，而对于每个结点的 rank，Treap 满足堆性质（小根堆或大根堆均可）。

由于随机的存在，Treap 的期望树高是 $\mathcal{O}(\log n)$ 的，具体证明会比较复杂，感兴趣的读者可以去网上自行查阅资料。

接下来，我们的主要任务就是在插入结点的时候同时维护堆性质和 BST 性质，此处产生了两个分支，传统的带旋转 Treap 和范浩强提出的无旋 Treap(FHQ-Treap)，其中 FHQ-Treap 将在下一小节讲解。



旋转操作

左图对结点 1 左旋, 得到右图, 右图对结点 2 右旋, 得到左图, 由此发现, 旋转分为左旋和右旋, 旋转不改变 BST 性质, 同时可以改变树的形态, 那么我们就可以用旋转维护一个 Treap 了.

和其他树形数据结构类似, 有指针的写法, 也有用数组模拟指针的写法, 这里给出后者. 首先定义结点:

```
1 mt19937 Rand(7629999);
2 struct Node {
3     int ch[2], sz, cnt, val, rank;
4 }tr[maxn];
5 int cnt = 0;
6 inline int New(int v = 0) {
7     tr[++cnt].val = v; tr[cnt].rank = Rand();
8     tr[cnt].ch[0] = tr[cnt].ch[1] = 0, tr[cnt].sz = tr[cnt].cnt = 1;
9     return cnt;
10 }
```

其中, *sz* 维护子树大小, *cnt* 维护值重复的结点个数, *lc, rc* 分别为左右子结点, *val* 为结点存储的真实值, 满足 BST 性质, *rank* 为随机的满足堆性质的值.

旋转操作分为左旋和右旋, 操作比较复杂:

```
1 inline void maintain(int u) {
2     tr[u].sz = tr[tr[u].ch[0]].sz + tr[tr[u].ch[1]].sz + tr[u].cnt;
3 }
4 inline void rotate(int &u, int d) {
5     //d=0:Left Rotate, d=1:Right Rotate
6     int v = tr[u].ch[d ^ 1];
7     tr[u].ch[d ^ 1] = tr[v].ch[d], tr[v].ch[d] = u, u = v;
8     maintain(tr[u].ch[d]), maintain(u);
9 }
```

插入操作是平凡的:

```
1 void insert(int &Root, int val) {
2     if(Root) {
3         tr[Root].sz ++;
4         if(val < tr[Root].val) {
5             insert(tr[Root].ch[0], val);
6             if(tr[Root].rank < tr[tr[Root].ch[0]].rank) rotate(Root, 1);
7         } else if(val > tr[Root].val) {
8             insert(tr[Root].ch[1], val);
9             if(tr[Root].rank < tr[tr[Root].ch[1]].rank) rotate(Root, 0);
10        } else tr[Root].cnt ++;
11    } else Root = New(val);
12 }
```

删除操作比较麻烦, 需要分类讨论子结点的存在情况, 返回值是是否存在值为 *val* 的结点:

```
1 bool delet(int &Root, int val) {
2     if(Root == 0) return false;
3     if(val < tr[Root].val) {
```

```

4   bool flag = delet(tr[Root].ch[0], val);
5   maintain(Root);
6   return flag;
7 } else if(val == tr[Root].val) {
8   if(tr[Root].cnt == 1) {
9     if(tr[Root].ch[0] == 0 && tr[Root].ch[1] == 0) Root = 0;
10    else if(tr[Root].ch[0] == 0) Root = tr[Root].ch[1];
11    else if(tr[Root].ch[1] == 0) Root = tr[Root].ch[0];
12    else if(tr[tr[Root].ch[0]].rank > tr[tr[Root].ch[1]].rank) {rotate(Root, 1); delet(tr[
    Root].ch[1], val);}
13    else {rotate(Root, 0); delet(tr[Root].ch[0], val);}
14  } else tr[Root].cnt --;
15  maintain(Root);
16  return true;
17 } else if(val > tr[Root].val) {
18   bool flag = delet(tr[Root].ch[1], val);
19   maintain(Root);
20   return flag;
21 }
22 }

```

找前驱，后继、值和排名的操作都比较简单，根据 BST 性质向下递归即可，代码实现留给读者练习。

为了处理方便，我们一般在初始时加入两个结点，它们的值分别为 $+\infty$ 和 $-\infty$ ，这样，在一系列操作中均不用再讨论边界情况，相应地，查询到的排名应该减 1，查询排名为 r 的数，实际上应该在树上查询排名为 $r+1$ 的数。

```

1 int root;
2 inline void init_Treap() {
3   root = New(INF);
4   insert(root, -INF);
5 }

```

至此，带旋 Treap 的操作已经被我们实现了。

模板

洛谷 P3369 【模板】普通平衡树

洛谷 P6136 【模板】普通平衡树（数据加强版）

2.3.3 FHQ-Treap

了解了传统的带旋 Treap，接下来我们来看一下另一种强有力的 Treap 实现，名为 FHQ-Treap，它可以很方便地进行传统 Treap 的操作，同时能够持久化，而且码量相对传统 Treap 较小。

无旋 Treap 采用**分裂-合并**的方式同时维护 BST 性质和堆性质，具体如下：

分裂操作

分裂操作有两种，按值分裂和按排名分裂，接下来以按值分裂为例。

分裂接受两个参数，需要分裂的树的根 root 以及分裂的标准值 val，它将一棵树分裂为两棵，其中一棵中的值小于等于 val，另一棵的值大于 val，返回值为两个分裂后的树根。实现时，我们可以传引用，将返回值也作为参数。


```

1 void split(int rt, int val, int &x, int &y) {
2     if(!rt) {x = y = 0; return;}
3     if(tr[rt].val <= val) {
4         x = rt; split(tr[x].rc, val, tr[x].rc, y);
5         maintain(x);
6     } else {
7         y = rt; split(tr[y].lc, val, x, tr[y].lc);
8         maintain(y);
9     }
10 }

```

比较容易理解, 由于原树已经满足了 BST 性质和堆性质, 所以我们只需要考虑一侧.maintain 操作维护的是子树大小, 具体来说, 这里采用的结点定义如下:

```

1 struct Node {
2     int lc, rc, val, rank, sz;
3 }tr[maxn];
4 int root = 0, cnt = 0;
5 inline int New(int val = 0) {
6     tr[++cnt].val = val; tr[cnt].lc = tr[cnt].rc = 0;
7     tr[cnt].rank = Rand(); tr[cnt].sz = 1;
8     return cnt;
9 }
10 inline void maintain(int u) {
11     tr[u].sz = tr[tr[u].lc].sz + tr[tr[u].rc].sz + 1;
12 }

```

和有旋 Treap 并无很大差异.

合并操作

```

1 int merge(int x, int y) {
2     if(!x || !y) return (x | y); // x | y == x + y at here
3     if(tr[x].rank > tr[y].rank) { // x is the root
4         tr[x].rc = merge(tr[x].rc, y);
5         maintain(x);
6         return x;
7     } else {
8         tr[y].lc = merge(x, tr[y].lc);
9         maintain(y);
10        return y;
11    }
12 }

```

返回值是合并后的根.

其他操作

有了分裂和合并两个核心操作, 现在可以用这两个操作十分方便地实现其他操作. 接下来的代码中, 为了方便, 我们在全局定义了三个 int 型变量 x,y,z 作为临时根, 稍后读者会了解到它们的用处.

- 插入值为 val 的结点
- 删除值为 val 的结点
- 查询 val 的排名
- 查询排名为 $rank$ 的值
- 查询值 val 的前驱和后继

将单个结点也看作一棵树，对于插入操作，我们可以依次进行如下操作：

- 将原树按 val 分裂为两棵新树，根为 x 和 y ，其中前者的所有值小于等于 val ，后者的所有值大于 val
- 将 x 合并为 x 和要插入的结点
- 合并新的 x 和 y 作为根

```
1 inline void insert(int val) {
2     split(root, val, x, y);
3     x = merge(x, New(val));
4     root = merge(x, y);
5 }
```

删除、查询排名类似：

```
1 inline void delet(int val) {
2     split(root, val, x, y);
3     split(x, val - 1, x, z);
4     z = merge(tr[z].lc, tr[z].rc); x = merge(x, z);
5     root = merge(x, y);
6 }
7 inline int ask_rank(int val) {
8     split(root, val - 1, x, y);
9     int ans = tr[x].sz + 1;
10    root = merge(x, y);
11    return ans;
12 }
```

简洁而优雅。

查询值、前驱、后继就和普通 BST 完全相同，不再给出。模板题可以和带旋 Treap 共用。

Luogu P3850

Knuth 先生家里有个精致的书架，书架上有 N 本书，如今他想学到更多的知识，于是又买来了 M 本不同的新书。现在他要把新买的书依次插入到书架中，他已经把每本书要插入的位置标记好了，并且相应的将它们放好。由于 Knuth 年龄已大，过几天他已经记不清某些位置上放的到底是什么书了，请问你能帮助他吗？

$1 \leq N \leq 200, 1 \leq M \leq 10^5, 1 \leq Q \leq 10^4$

Solution: Luogu P3850

本题有两种解法，线段树上二分和 FHQ-Treap

先看线段树上二分，我们将所有操作逆序处理，在线段树上二分第 $p+1$ 个空位即可，输出是 $O(Q)$ 的，处理是 $O((N+M) \log(N+M))$ 的。

再看平衡树解法，当然可以用 Splay，但是我们还不会，或者用按排名分裂的 FHQ-Treap，再或者用带懒惰标记的 FHQ-Treap。前者的做法很显然了，分裂后插入即可，我们在这里了解一下带标记的 FHQ-Treap。我们在

位置 p 插入了一个书，那么原本排名大于等于 p 的书的排名就都加了 1，而整体加 1 可以采取经典的方法：打标记 + 下传标记，这种方法我们还会在接下来的 Splay 中看到。

2.3.4 Splay

Splay，中文名为“伸展树”，得名于它的核心操作 splay. 与 Treap 最大的不同是，它不用随机数来时刻保证自己的深度，而是采用了均摊 $O(\log n)$ 的时间复杂度，注意“均摊”二字，这说明 splay 单次操作的时间复杂度是可以到达 $O(n)$ 的。

所谓伸展操作，其实就是将一个结点不断旋转，直到成为根节点，但这样还不够：对于一条链，我们将最底部的点旋转到根后，这棵树还是一条链，这直接导致了我们的复杂度错误。解决方法是很巧妙的，splay 采用了“双旋”的策略，具体来说，在旋转时，我们分以下三种情况。

- 待旋转结点的父结点就是根节点，直接旋转上去
- 待旋转结点的父结点不是根节点，并且待旋转结点和其父结点同为各自父亲的左（或右）儿子，此时先将待旋转结点的父结点旋转上去，再旋转待旋转结点
- 待旋转结点的父结点不是根节点，并且情况与上一种不同，即一左一右，则旋转两次待旋转结点

这里的旋转和我们在 Treap 中给出的示例不太相同，之前我们说的旋转是向下转，这里是向上转，代码如下：

```
1 inline void maintain(int u) {
2     sz[u] = sz[ch[u][0]] + sz[ch[u][1]] + cnt[u];
3 }
4 inline int get(int u) {
5     return (u == ch[fa[u]][1]);
6 }
7 inline void rotate(int u) {
8     int f = fa[u]; int d = get(u), df = get(f);
9     ch[f][d] = ch[u][d ^ 1], fa[ch[u][d ^ 1]] = f;
10    fa[u] = fa[f]; if(fa[f]) ch[fa[f]][df] = u;
11    ch[u][d ^ 1] = f, fa[f] = u;
12    maintain(f), maintain(u);
13 }
```

可以看到我们对于每个结点又储存了它的父结点编号 $fa[u]$ ，上述代码中， $get(u)$ 的含义是得到 u 是左子结点（返回 0）还是右子结点（返回 1）， $maintain$ 维护结点的子树大小。旋转操作的大致思路是：先将 u 另一侧的子结点变为父结点的，再改变自己的父亲，最后改变和原本父结点之间的关系，因为还要考虑 fa 数组，所以略显繁琐，还请读者深刻理解并记忆。

有了这些基本操作，接下来就是 splay 的核心操作：

```
1 inline void splay(int u) {
2     for(int f = fa[u]; f = fa[u], f; rotate(u))
3         if(fa[f]) rotate(get(u) == get(f) ? f : u);
4     root = u;
5 }
```

即伸展操作，将指定结点变为根节点，实现思路之前已经提到。其他扩展操作怎么做呢？对于插入操作，按照 BST 性质找到空位（或某个已有结点），新建结点后**将它旋转到根**。

```
1 inline void insert(int v) {
2     if(!root) {root = New(v); return;}
3 }
```

```

3  int u = root, f = 0;
4  while(true) {
5      if(val[u] == v) {
6          cnt[u] ++; maintain(u); splay(u); return;
7      }
8      int d = (v > val[u]);
9      f = u; u = ch[u][d];
10     if(!u) {
11         ch[f][d] = New(v, f); maintain(f); splay(ch[f][d]); return;
12     }
13 }
14 }

```

注意加粗的字，每次进行插入、删除、查找等一系列操作后，都要执行一次 **splay** 操作，只有这样，我们的均摊复杂度才有保证。给定值，寻找排名或结点编号以及给定排名，寻找值或结点编号和正常的 BST 并无差异，不再占用篇幅，现在假设我们已经写出了 `get_rank` 函数，找到了排名并且**旋转到了根**。那么删除操作也就不难完成。

```

1  inline int get_pre() {
2      int u = ch[root][0];
3      while(ch[u][1]) u = ch[u][1];
4      return u;
5  }
6  inline int get_suf() {
7      int u = ch[root][1];
8      while(ch[u][0]) u = ch[u][0];
9      return u;
10 }
11 inline void delet(int val) {
12     int tmp = get_rank(val);
13     if(cnt[root] > 1) {
14         cnt[root] --; maintain(root); return;
15     }
16     if(!ch[root][0] || !ch[root][1]) {
17         root = ch[root][0] + ch[root][1];
18         fa[root] = 0; return;
19     }
20     int old_root = root; splay(get_pre());
21     ch[root][1] = ch[old_root][1];
22     fa[ch[old_root][1]] = root; maintain(root);
23 }

```

寻找某个值的前驱（后继），我们可以这么写：

```

1 {insert(x); cout << val[get_pre()] << '\n'; delet(x);}
2 {insert(x); cout << val[get_suf()] << '\n'; delet(x);}

```

思路是先旋转到根 (insert)，再顺着往下找，找到后撤销我们不该插入的结点。其他普通平衡树的操作留给读者练习，这里给出一道习题：Luogu P2286 [HNOI2004] 宠物收养场。

splay 还可以进行的操作是维护区间 (FHQ-Treap 同样可以完成)，具体来说，让我们看一道例题：

Luogu P3391

您需要写一种数据结构（可参考题目标题），来维护一个有序数列。

其中需要提供以下操作：翻转一个区间，例如原有序序列是 5 4 3 2 1，翻转区间是 [2, 4] 的话，结果是 5 2 3 4 1。

Solution: Luogu P3391

对于这种维护区间翻转的问题，我们可以用 splay 解决，具体来说，我们将每个元素依次编号为 $1, 2, 3, 4 \dots$ ，然后按照编号建立 BST，什么意思呢？就是我们不管结点中储存的值，BST 所比较的是编号，也就是说在最开始，一个编号为 u 的结点，其左子树的编号都小于 u 。

建树有两种方式，一种是一个一个插入，另一种是每次取中点，递归建完全平衡的树。

之后怎么办呢？容易发现，将结点 u 所在子树（包括 u ）的所有结点的左右子结点互换，作用恰好是反转这个区间，它的确改变了 BST 性质，但是没关系，我们接着往下走。假如我们已经得到了一种方法，能够把任意给定的区间 $[l, r]$ 塞到一棵子树中，且这棵子树中不含其他结点，我们就可以直接在根节点打上标记，于是我们现在的任务是：对于每次操作，找到当前在区间第 l 个位置和第 r 个位置的数在 splay 中的编号，把这个区间割裂出来，提取一下我们的任务就是：找到当前区间上第 i 个数在 splay 上对应的结点编号。

仔细思考一下，我们会发现，这个结点恰好是 splay 中中序遍历第 i 个结点，也就是排名为 i 的结点，而这是很好找到的：

```
1 inline int get_num(int rk) {
2     int u = root;
3     while(true) {
4         pushdown(u);
5         if(sz[ch[u][0]] >= rk) u = ch[u][0];
6         else if(sz[ch[u][0]] + 1 == rk) return u;
7         else rk -= sz[ch[u][0]] + 1, u = ch[u][1];
8     }
9     return -1;
10 }
```

easy.

那么现在，我们只需要考虑怎样将这些结点塞到一起并且割裂开来，问题就完美解决了。

怎么做呢？假如我们要将区间 $[l, r]$ 割出来，我们先把排名为 $l-1$ 的结点伸展到根，再将排名为 $r+1$ 的结点伸展为根的子结点，此时排名为 $r+1$ 的结点的左子树就是区间 $[l, r]$ 的所有结点了（仔细想一想，为什么）。其中，有一个操作是我们没有见过的，就是伸展到某个结点下方，实际上的实现并不困难：

```
1 inline void splay(int u, int goal = 0) {
2     for(int f = fa[u]; f = fa[u], f != goal; rotate(u)) {
3         pushdown(fa[f]), pushdown(f), pushdown(u);
4         if(fa[f] != goal) rotate((get(u) == get(f) ? f : u));
5     }
6     if(goal == 0) root = u;
7 }
```

假如我们要旋转到根，可以执行 `splay(u)` 或 `splay(u, 0)`. `pushdown` 操作是下传标记用的，注意时刻下传标记：

```
1 inline void pushdown(int u) {
2     if(!u || !tag[u]) return;
3     swap(ch[u][0], ch[u][1]);
4     if(ch[u][0]) tag[ch[u][0]] ^= 1;
5     if(ch[u][1]) tag[ch[u][1]] ^= 1;
6     tag[u] = 0;
7 }
```

翻转区间的代码在这里：

```
1 splay(get_num(l-1)), splay(get_num(r-1), root);
2 tag[ch[ch[root][1]][0]] ^= 1;
```

细心的读者可能会发现一个问题，假如我们翻转的区间是 $[1, n]$ 上述的处理会出错. 事实也是这样，解决方法是添加两个虚拟结点加在首尾，或者对于边界问题分类讨论.

至此，我们介绍了三种常用的平衡树，实际上平衡树的种类远多于三种，但在 OI 中，这三种已经基本够用，想要了解更多平衡树的读者可以自行在网上查阅资料.

下面再给出一道例题：

Luogu P4200

每只鸟都有一个编号，都有一个威武值. 每秒钟鸟王都会发一个命令，编号为 v 的鸟飞到 (x, y) 去（坐标系原点是山顶，坐标单位为鸟爪）. 鸟飞得很快，一秒之内就飞到了，可以看作是瞬间移动. 如果编号为 v 的鸟和编号为 u 的鸟某一时刻处在同一位置，它们就会互相鼓励，增加各自的士气值和团结值. 一只鸟的士气值等于此刻与它处在同一位置的鸟中的威武值的最大值，团结值等于此刻与它处在同一位置的鸟的只数. 如果每一时刻都没有鸟与它处在同一位置，则士气值和团结值都为 0. 要注意自己不能鼓励自己，计算士气值和团结值时不能算上自己.

doyouloveme 这样描述战斗力：一只鸟战斗力值等于它在 0 到 t 秒中士气值的最大值与团结值的最大值的乘积. 注意不是乘积的最大值，而是最大值的乘积.

t 秒钟后，求每只鸟的战斗力.

对于 100% 的数据， $1 \leq n \leq 30000, 0 \leq t \leq 300000$ ，坐标为整数，均在 $[-2^{31}, 2^{31})$ 内.

Solution: Luogu P4200

用平衡树解决的话，首先大概率要打 `tag`，因为涉及到了整体的操作（一只鸟不断从所有鸟都在的一个点飞出去又飞回来，那么所有鸟的答案都要更新，而一个一个更新是绝对行不通的）. 坐标的范围很大，但是我们可以离散化到 $O(n + t)$ 级别，之后对每个点开一个平衡树 (FHQ-Treap 或 Splay)，每只鸟对应一个平衡树的结点，结点编号与鸟编号一一对应，BST 性质是对编号排序的. 这里采用 FHQ-Treap，对于每个结点我们维护以下信息：士气值的最大值、团结值的最大值、子树大小，子树威武值最大值、懒惰标记 1 记录士气值，懒惰标记 2 记录团结值（具体用处我们接下来讨论）.

最开始，我们插入 n 个点，假如待插入位置没有鸟，直接塞进去就好，结点只需要记录子树大小、子树威武值最大值还有它本身的威武值，假如那里已经有若干个鸟，我们就需要对这棵树进行一些操作在插入这只鸟了，为了不让这棵树之前的标记影响到待插入的鸟，我们需要先把标记更新（首先更新根节点的答案，再更新 `tag`，最后插入结点），复杂度不会爆掉的原因是不必将所有标记都删掉，只需要保证对新加入的结点不产生影响即可、这在合并、删除的过程中可以顺便完成.

之后，我们开始有了删除操作，其实没有区别，直接删除（合并、分裂的时候会把这个结点的答案顺便更新掉），然后再如同刚开始插入结点即可。

威武值为不超过 $2^{31} - 1$ 的非负整数。

2.4 左偏树

问题引入：

现有 10^6 个单节点小根堆²，顺次编号为 $1, 2, \dots, n$ ，要求支持 $O(\log n)$ 实现以下操作：

- 合并两个小根堆。
- 删除某个小根堆里的最小数（即堆顶）。

首先想一下，假如要暴力地合并两个二叉小根堆，我们应该怎么做？

可以设计流程如下：

- 假如其中一个堆为空，直接返回另一个堆的堆顶。
- 比较两个堆的堆顶，将较小的那个作为堆顶，不妨设为 A ，另一个设为 B 。
- 递归合并 A 的右儿子和 B ，作为 A 的新右儿子。

整个递归过程结束后，返回的即为合并后的堆顶。

上述算法的时间复杂度是多少呢？若两个堆的深度为 d_1 和 d_2 的话，复杂度是 $O(d_A + d_B)$ 的，而 d 最坏是 $O(n)$ ，为了优化这个暴力的合并过程，“左偏树”出现了。

定义 2.2

外结点：没有左儿子或右儿子的结点。

结点的 $dist$ ：外结点的 $dist$ 为 0，空结点的 $dist$ 为 -1 ，否则该结点的 $dist$ 为 $\min\{dist_{lc}, dist_{rc}\} + 1$ 。



性质 对于一棵结点个数为 n 的二叉树，其根节点的 $dist$ 不超过 $\lceil \log(n+1) \rceil$ 。

证明 考虑一棵根节点的 $dist$ 为 d 的左偏树，根据定义，它的前 $d-1$ 层都是满二叉树，含有 $2^d - 1$ 个结点。

假如我们合并两棵左偏树，可以怎么做呢？

- 假如有一棵左偏树为空，直接返回另一个左偏树的根。
- 不妨设堆顶值较小的堆设为 A ，另一个设为 B ，那么合并之后的堆顶即为 A 。
- 递归合并 A 的右儿子和 B ，将返回的新堆顶作为 A 的新右儿子。

上述算法的时间复杂度是多少？ $O(dist_A + dist_B)$ 。而两个堆都是左偏树，所以复杂度为 $O(\log_{szA} + \log_{szB})$ 。

看起来我们的算法非常完美，但是有一个 bug：合并之后的堆不一定是左偏树。不要慌，仔细想想，这个问题不难解决，只需要在每次改变右儿子之后加一个判断：若右儿子的 $dist$ 大于左儿子的 $dist$ ，交换左右儿子，同时，我们还应更新根节点的 $dist$ ，变为新右儿子的 $dist + 1$ 。

怎样删除一个堆的根节点呢？合并根节点的左右儿子作为新根，丢掉原来的根节点即可，类似地，插入一个结点可以看作两个堆合并。

下面给出代码示例，由于在洛谷上，若两个初始结点的值相同，认为先输入的结点更小，所以结点的值为一个 pair，第一维为值，第二维为输入次序。

```
1 struct Node {
2     int lc, rc, dist;
3     pair<int, int> v;
4     Node(const pair<int, int> _v) :
```

²在接下来的讨论中，小根堆和大根堆的流程是几乎完全一样的


```

5     v(_v) {lc = rc = 0, dist = -1;}
6     Node() {}
7 }tr[maxn];
8 int cnt = 0;
9 inline int New(const pair<int, int> v) {
10     tr[++cnt] = Node(v);
11     return cnt;
12 }

1 int merge(int x, int y) {
2     //return the new root.
3     if(!x || !y) return (x | y);
4     if(tr[y].v < tr[x].v) swap(x, y);
5     tr[x].rc = merge(tr[x].rc, y);
6     if(tr[tr[x].rc].dist > tr[tr[x].lc].dist) swap(tr[x].lc, tr[x].rc);
7     tr[x].dist = tr[tr[x].rc].dist + 1;
8     return x;
9 }
10 int del(int x) {
11     //return the new root.
12     return merge(tr[x].lc, tr[x].rc);
13 }

```

上面介绍了左偏树的一些基本操作，下面简单说一些其他操作，具体代码可以参考 [OI_WIKI](#).

删除任意结点

合并左右儿子，之后自底向上更新 `dist`，不满足左偏性质时交换左右儿子，`dist` 无需更新时结束递归。时间复杂度 $O(\log n)$ 。

对整个对加减乘

在根节点上打上标记即可，进行其他操作时上传标记。

Luogu P1552

在这个帮派里，有一名忍者被称之为 Master。除了 Master 以外，每名忍者都有且仅有一个上级。为保密，同时增强忍者们的领导力，所有与他们工作相关的指令总是由上级发送给他的直接下属，而不允许通过其他方式发送。

现在你要招募一批忍者，并把它派遣给顾客。你需要为每个被派遣的忍者支付一定的薪水，同时使得支付的薪水总额不超过你的预算。另外，为了发送指令，你需要选择一名忍者作为管理者，要求这个管理者可以向所有被派遣的忍者发送指令，在发送指令时，任何忍者（不管是否被派遣）都可以作为消息的传递人。管理者自己可以被派遣，也可以不被派遣。当然，如果管理者没有被派遣，你就不需要支付管理者的薪水。

你的目标是在预算内使顾客的满意度最大。这里定义顾客的满意度为派遣的忍者总数乘以管理者的领导力水平，其中每个忍者的领导力水平也是一定的。

给定每一个忍者 i 的上级 B_i ，薪水 C_i ，领导力 L_i ，以及支付给忍者们的薪水总预算 M ，输出在预算内满足上述要求时顾客满意度的最大值。

$1 \leq N \leq 10^5$, $1 \leq M \leq 10^9$, $0 \leq B_i < i$, $1 \leq C_i \leq M$, $1 \leq L_i \leq 10^9$.

Solution: Luogu P1552

还是挺板的，初始时每个点是一个大根堆，自上而下合并，假如堆大小超过了限制，不断弹出堆顶，同时统计答案即可，记得开 long long.

Luogu P3261

小铭铭最近获得了一副新的桌游，游戏中需要用 m 个骑士攻占 n 个城池.

这 n 个城池用 1 到 n 的整数表示. 除 1 号城池外，城池 i 会受到另一座城池 f_i 的管辖，其中 $f_i < i$. 也就是说，所有城池构成了一棵有根树.

这 m 个骑士用 1 到 m 的整数表示，其中第 i 个骑士的初始战斗力为 s_i ，第一个攻击的城池为 c_i .

每个城池有一个防御值 h_i ，如果一个骑士的战斗力大于等于城池的生命值，那么骑士就可以占领这座城池；否则占领失败，骑士将在这座城池牺牲. 占领一个城池以后，骑士的战斗力将发生变化，然后继续攻击管辖这座城池的城池，直到占领 1 号城池，或牺牲为止.

除 1 号城池外，每个城池 i 会给出一个战斗力变化参数 (a_i, v_i) . 若 $a_i = 0$ ，攻占城池 i 以后骑士战斗力会增加 v_i ；若 $a_i = 1$ ，攻占城池 i 以后，战斗力会乘以 v_i .

注意每个骑士是单独计算的. 也就是说一个骑士攻击一座城池，不管结果如何，均不会影响其他骑士攻击这座城池的结果.

现在的问题是，对于每个城池，输出有多少个骑士在这里牺牲；对于每个骑士，输出他攻占的城池数量.

$1 \leq n, m \leq 3 \times 10^5$, $-10^{18} \leq h_i, v_i, s_i \leq 10^{18}$, $1 \leq f_i < i, 1 \leq c_i \leq n, a_i \in \{0, 1\}$, 保证 $a_i = 1$ 时, $v_i > 0$, 保证任何时候骑士战斗力值的绝对值不超过 10^{18} .

Solution: Luogu P3261

跟例题 1 基本一样，无非是加了 tag 的可并堆而已，还是注意 long long 即可.

Luogu P3273

有 N 个节点，标号从 1 到 N ，这 N 个节点一开始相互不连通. 第 i 个节点的初始权值为 a_i ，接下来有如下一些操作：

- U x y: 加一条边，连接第 x 个节点和第 y 个节点
- A1 x v: 将第 x 个节点的权值增加 v
- A2 x v: 将第 x 个节点所在的连通块的所有节点的权值都增加 v
- A3 v: 将所有节点的权值都增加 v
- F1 x: 输出第 x 个节点当前的权值
- F2 x: 输出第 x 个节点所在的连通块中，权值最大的节点的权值
- F3: 输出所有节点中，权值最大的节点的权值

$N \leq 300000, Q \leq 300000$ ，保证输入合法，并且 $-1000 \leq v, a_i \leq 1000$.

Solution: Luogu P3273

这道题就不是裸的模板题了，原因是加入了查询某个非根结点的值，修改某个非根结点的值，对某一个堆整体加减这三个操作.

假如只有修改、查询某个非根结点的值与合并两个堆这样的操作，我们是很容易实现的，方法在上文已经有过介绍，而现在，我们需要在支持这些操作的同时支持维护 tag 和查询某结点的祖先的 tag 和.

显然查询祖先的 tag 和是 $O(n)$ 的，只要解决了这个问题，剩下的操作用左偏树不难实现，接下来我们重点看这个操作的优化.

首先我们要明确一点，**一定不能暴力地跳祖先**，因为无论如何，这样的复杂度一定不会优。那怎么办呢？我们想办法把 `tag` 固定在堆的根节点上。

具体来讲，我们在进行合并操作时，使用 **启发式合并**，暴力地将结点数较小的那个堆的 `tag` 更新到每个结点，然后 $O(1)$ 维护合并后的根的 `tag`。

上述操作的复杂度是什么呢？每个结点被暴力更新，更新后，所在堆的大小至少翻倍，于是每个结点最多下放 $O(\log n)$ 次标记，于是均摊复杂度为 $O(\log n)$ ，下面放代码：

```
1 if(op == "U") {
2     cin >> x >> y;
3     x = find(x), y = find(y);
4     if(x == y) continue;
5     if(sz[x] > sz[y]) swap(x, y);
6     pushdown(x, tag[x] - tag[y]);
7     p[x] = p[y] = merge(x, y);
8     if(p[x] == x) {
9         S.erase(S.find(tr[y].val + tag[y]));//
10        tag[x] = tag[y], sz[x] += sz[y]; tag[y] = sz[y] = 0;
11    } else {
12        S.erase(S.find(tr[x].val + tag[y]));//
13        sz[y] += sz[x]; tag[x] = sz[x] = 0;
14    }
15 }
```

加有注释的行我们稍后讨论，上面的代码中，`p` 是并查集的数组，`find` 是并查集的查询函数，`sz` 维护堆的大小，下标在根上，`tag` 维护堆整体加的值，下标同样在根。

下面是 `pushdown` 函数：

```
1 void pushdown(int x, int y) {
2     if(!x) return;
3     tr[x].val += y;
4     pushdown(tr[x].lc, y), pushdown(tr[x].rc, y);
5 }
```

暴力更新，不再多说。

任意点更新的 `maintain` 函数和修改过的 `merge` 函数在这里：

```
1 void maintain(int x) {
2     if(!x) return;
3     if(tr[x].dist != tr[tr[x].rc].dist + 1) {
4         tr[x].dist = tr[tr[x].rc].dist + 1;
5         maintain(tr[x].fa);
6     }
7 }
8 int merge(int x, int y) {
9     if(!x || !y) return (x | y);
10    if(tr[x].val < tr[y].val) swap(x, y);
11    tr[x].rc = merge(tr[x].rc, y);
12    tr[tr[x].rc].fa = x;
```

```

13  if(tr[tr[x].lc].dist < tr[tr[x].rc].dist)
14      swap(tr[x].lc, tr[x].rc);
15  maintain(x);
16  return x;
17 }

```

现在, 我们来看如何维护全局的最大值, 马上能想到, 开一个 `multiset` 维护所有堆的根节点的值, 询问时直接取最大值即可, 这也是我们刚刚注释掉的话所维护的. 对于全局的加减, 我们直接维护一个全局变量即可.

Luogu P4331

给定一个整数序列 a_1, a_2, \dots, a_n , 求出一个递增序列 $b_1 < b_2 < \dots < b_n$, 使得序列 a_i 和 b_i 的各项之差的绝对值之和 $|a_1 - b_1| + |a_2 - b_2| + \dots + |a_n - b_n|$ 最小. 要求输出方案.
 $n \leq 1000000, 0 \leq a_i \leq 2 \times 10^9$

Solution: Luogu P4331

首先做一个小处理, 把第 a_i 变为 $a_i - i$, 这样, 我们求出一个不降序列 b' , 那么 $b_i = b'_i + i$ 即为所求的 b 序列, 之后, 除非特殊说明, 我们说到的 a 都是指 a' , b 都是指 b' .

先考虑两种特殊情况:

- a 不降, 这是最简单的形式了, 只需要让 $b_i = a_i$ 即可得到一组合法的答案最小的序列了.
- a 严格递减, 考虑所求答案在数轴上的表现形式, 对于所有的位置均取所有数的中位数是最优的, 证明可以用几何意义.

对于一般情况呢? 一般的序列显然是有若干个上述两种情况拼接而成的, 现在的问题转化为了如何求拼接后的答案.

现在不妨设前一段的取值为 c_1 , 后一段为 c_2 , 若 $c_1 \leq c_2$, 什么也不用做, 否则可以证明, 再次取中位数是最优的, 证明类似.

我们要做的就是快速合并, 取中位数, 这样的操作可以用可并堆实现.

第3章 抽象数据结构

内容提要

▣ 该部分内容选自李欣隆《抽象数据结构》

“抽象数据结构”是指，我们不去具体考虑维护的信息是什么，而是用类似群论（或者说就是群论）的想法，考虑怎样的数据结构可以维护有怎样性质的信息。

在阅读本章内容之前，你需要对群论有初步的认识，这在我们之后的数学一章中会讲到。

在学习线段树的时候，我们看到了很多区间问题都可以用线段树维护，那么什么样的信息可以用传统线段树（这里是指可以带有懒标记，单次操作 $\mathcal{O}(\log n)$ 的线段树）维护呢？

考虑我们都曾维护过什么样的信息：区间和、区间 \max ，区间 \min ，区间最大子段和，等等。令线段树上结点维护的所有可能的信息构成的集合为 \mathcal{D} ，tag 维护的所有可能的信息构成的集合为 \mathcal{M} ，那么我们只需要能够把 tag 作用到信息上、把 tag 和 tag 合并为一个 tag，就可以把这两个信息上到线段树上。

于是， $(\mathcal{D}, +)$ 构成一个半群（因为合并多个区间必须满足结合律、并且封闭性是显然的），这个操作的意义是合并两个区间， $(\mathcal{M}, *)$ 也构成一个半群，这个运算可以合并两个 tag，还有一个作用在 \mathcal{D} 和 \mathcal{M} 之间的运算 \times ，即 $\mathcal{D} \times \mathcal{M} \rightarrow \mathcal{D}$ ，作用是将标记作用到信息上，方便起见， \times 这个运算可以和 $*$ 合并，那么 $(\mathcal{D} \cup \mathcal{M}, *)$ 应该构成一个半群（这也同时达到了另一个要求： $(d \times m) \times m = d \times (m * m)$ ）。

还有没有什么遗漏呢？标记应当是可以下放的，即满足分配律 $(d_1 + d_2) * m = d_1 * m + d_2 * m$ ，这就可以了。

想想是不是这样的，当我们维护的是区间和时， $\mathcal{D} = \mathbb{R}$ ， $+$ 就是实数加法， \mathcal{M} 也是 \mathbb{R} ， $*$ 同样是实数加法，对于剩下的三种信息，都是类似的。

这个模型称为“**双半群模型**”，原因是 \mathcal{D} 和 \mathcal{M} 都是两个半群，它们之间的联系是有分配率。

这个模型看起来没什么用，但是却可以辅助线段树中 tag 的构造。以【NOIP2022】比赛一题中的线段树部分为例，我们来讲讲如何更迅速地得到 tag 的构造方法。

在本题中，我们要实现序列 X, Y 的区间推平，令 $T_i = X_i \times Y_i$ ， C_i 为 T_i 的历史版本和，即要实现 C 的区间加对应位置的 T_i 。查询的是 C_i 的区间和。

\mathcal{D} 中元素应该是什么样子的呢？里面首先应该有一个历史版本和 s ，其次，假如只有推平 X 的操作，那么每次的变化形如一个 $\sum Y_i$ 的东西，考虑把它维护进去，设为 y ，表示区间的 Y_i 之和， x 类似，表示区间的 X_i 之和，两个合起来，再维护一个 $\sum X_i \times Y_i$ ，设为 xy ，同时肯定是要有区间长度的（因为我们有区间推平，不维护长度就无法维护 $\sum X_i$ 和 $\sum Y_i$ ），因此 \mathcal{D} 中元素被初步设定为一个五元组 $\langle x, y, xy, s, len \rangle$ ， \mathcal{D} 的合并可以很自然地将五元组的对应位置直接相加得到。

重点在于 tag 的构造，即 \mathcal{M} 中元素维护的信息。首先应该有区间推平的 tag $setx, sety$ ，其次要考虑怎么去维护 \mathcal{D} ：套路化地去考虑维护 $addx, addy, addxy, addc$ 分别表示让 s 加上对应倍的 x, y, xy ，区间的每个位置让 s 加上 $addc$ （一共就是 $len \times addc$ ）。

$\mathcal{D} * \mathcal{M}$ 是比较平凡的， $\mathcal{M} * \mathcal{M}$ 也不难搞：将冲突的都转到 $addc$ 上就行，于是就做完了。

第二部分

动态规划

第 4 章 动态 DP

动态 DP 是一类解决支持树上带修 dp 问题的方法。

Luogu P4719

给定一棵 n 个点的树，点带点权， i 号点的权值为 val_i 。有 m 次操作，每次操作给定 x, y ，表示修改点 x 的权值为 y 。

你需要在每次操作之后求出这棵树的最大权独立集的权值大小。

Solution: Luogu P4719

假如没有修改操作，这个静态问题就是一个我们非常熟悉的经典树上 dp。先让一个点（不妨为 1）作为整棵树的根，设 $f_{i,0}$ 为钦定 i 号点不选，其子树的答案， $f_{i,1}$ 为钦定 i 号点选，其子树的答案，那么整棵树的答案就是 $\max\{f_{1,0}, f_{1,1}\}$ 。

转移是容易的：

$$f_{u,0} = \sum_{v=son_u} \max\{f_{v,0}, f_{v,1}\}$$
$$f_{u,1} = val_u + \sum_{v=son_u} f_{v,0}$$

这样就 $\mathcal{O}(n)$ 解决了静态问题，但是本题要我们解决的是动态问题，这样的 dp 已经不能满足我们的需求。

当改掉一个点（设为 u ）的权值后，哪些点的 f 发生了变化呢？没错，是 u 到根的路径上的点，那么一个显然的思路是每次修改后只维护这些点的 f 值，这样单次修改是 $\mathcal{O}(dep_u)$ 的，其中 dep_u 指的是点 u 的深度。

很可惜，当树退化为一棵链后，深度还是 $\mathcal{O}(n)$ 的，但这种想法给了我们另一个思路：重链剖分。

具体来说，先将整棵树重链剖分，设 $g_{i,0}$ 为钦定 i 号点不选，它的所有轻儿子的子树所得到的答案， $g_{i,1}$ 为钦定 i 号点选，它的所有轻儿子的子树和它自己所得到的答案，得到新的 f 的转移方式：

$$f_{u,0} = g_{u,0} + \max\{f_{hson_u,0}, f_{hson_u,1}\}$$
$$f_{u,1} = g_{u,1} + f_{hson_u,0}$$

其中 $hson_u$ 指的是 u 的重儿子。

写成广义矩阵乘法的形式，这里的矩阵乘法内部的“乘”是“加”，“加”是取 \max ，即若矩阵 A, B, C 满足 $C = A \times B$ ，就意味着。

$$C_{ij} = \max_k \{A_{ik} + B_{kj}\}$$

转移方程就可以写作：

$$\begin{bmatrix} f_{u,0} \\ f_{u,1} \end{bmatrix} = \begin{bmatrix} g_{u,0} & g_{u,1} \\ g_{u,1} & -\infty \end{bmatrix} \times \begin{bmatrix} f_{hson_u,0} \\ f_{hson_u,1} \end{bmatrix}$$

请读者自行验证。

这样有什么用呢？我们只需要维护每个点的转移矩阵（实际上就是 g 数组）， $f_{1,0}$ 和 $f_{1,1}$ 就是根所在重链上所有矩阵从上到下乘积所得 2×2 矩阵的第一行第一列、第二行第一列的两个元素。

这是为什么呢？对于一个重链最上方的结点（设为 u ），它的重链矩阵的乘积就是

$$\begin{bmatrix} f_{u,0} & f_{u,0} \\ f_{u,1} & -\infty \end{bmatrix}$$

（请读者务必想清楚这里的原因，将这个矩阵按照转移方程不断展开，结果就是重链上所有矩阵的广义乘积）

这里的关键之处在于将树上问题转化成了矩阵序列的区间乘积问题，接下来我们只需要想办法在每次修改操作后维护这些矩阵即可。

设修改 val_x 为 y ，先修改掉 $g_{x,1}$ ，它变为了 $g_{x,1} - val_x + y$ ，而 $g_{x,0}$ 是不变的，这里直接在线段树上改掉即可。从 x 出发不断沿着重链向上跳，每次跳到重链头的上方结点时，根据我们刚才的结论，该结点的 f 值是一个区间矩阵的积，线段树求出来后，上方结点的新 g 值也就可求了，直到跳到根所在重链，这里相比平常的重链剖分，还需要额外记录一下每条重链的结尾。

时间复杂度 $\mathcal{O}(n \log^2 n)$ 。

有没有更快的做法呢？可以用 LCT 维护，每次修改时将修改的点 splay 上去直接修，查询时将 1 号点 splay 上去直接查就行，时间复杂度 $\mathcal{O}(n \log n)$ 。

相较于大常数的 LCT，全局平衡二叉树会更快一些，时间复杂度同样是 $\mathcal{O}(n \log n)$ 。具体来说，用 LCT 是很浪费的，因为我们完全没有“加边”，“删边”等复杂操作，整个 LCT 的形态始终没有变。将整棵树重链剖分后，把每个重链用重边组织成二叉树，重链之间用连接它们的边作为轻边连接起来，假如我们能把树高控制在 $\mathcal{O}(\log n)$ ，每次修改直接暴力往上条就行了。这是可以做到的，将每个点的权值设为轻儿子的子树大小和再加一，每次取出链的带权重心作为根，两边递归处理，两边的根分别作为左右儿子即可。这样组织出来的树高是 $\mathcal{O}(\log n)$ 的，原因是父亲子树大小必然乘 2。

第 5 章 DP 优化技巧

很多情况下，我们设出状态后得到了正确的转移方程，但暴力转移的时间复杂度高得惊人，这时除了优化状态、重新设计转移方程之外，还有一种相对暴力的优化方法：对转移的时间复杂度进行优化，接下来我们主要围绕这一点进行讲解。

5.1 长链剖分优化

这个 trick 的实现类似于 dsu on tree，可以线性维护子树中与深度有关的信息。

长链剖分类似重链剖分，不同之处在于，结点 u 的重儿子是其儿子中高度最大的那一个，一个结点的高度定义为它到子树中所有叶子结点的距离的最大值，形式化地，叶子结点的高度为 1，非叶子结点 u 的高度 $height_u$ 为

$$\max_{v \in son_u} \{height_v\} + 1$$

之后我们就可以用一系列重边将树划分为一堆链了。长链剖分的性质为：从某结点走到根结点，经过的轻边个数是 $\mathcal{O}(\sqrt{n})$ 的，其中 n 为结点个数，同时这个上界是可以构造¹得到的。

CF1009F

给定一棵以 1 为根， n 个节点的树。设 $d(u, x)$ 为 u 子树中到 u 距离为 x 的节点数。

对于每个点，求一个最小的 k ，使得 $d(u, k)$ 最大。

$1 \leq n \leq 10^6$

Solution: CF1009F

本题可以暴力地线段树合并，但不是我们的重点，考虑 dp 解决这个问题。

设 $f_{u,i}$ 表示在以 u 为根的子树中距离 u 为 i 的结点的数量，转移是容易的：

$$f_{u,i} = \sum_{v \in son_u} f_{v,i-1}$$

答案可以在转移时顺便维护，时间复杂度 $\mathcal{O}(n^2)$ 。怎样优化呢？做法是对于每个点，直接继承其重儿子的状态，暴力转移轻儿子的状态，转移轻儿子的状态时，只枚举到该儿子的高度（因为剩下的一定全是 0），这样，转移的时间是所有重链的长度和，即 $\mathcal{O}(n)$ 。

长链剖分优化 dp 的代码实现非常精彩，我们还是同重链剖分一样预处理出 dfs 序，对于每一条重链整体分配内存，只不过链上的不同结点的首指针不同。以本题为例，略去对于高度和重儿子的预处理（点 u 的高度为 $height_u$ ，重儿子为 $lson_u$ ），看处理 dfs 序的部分：

```
1 int *f[maxn], g[maxn], dfn[maxn], ans[maxn], dfs_clock = 0;
2 void hld(int u, int fa) {
3     dfn[u] = ++dfs_clock; f[u] = g + dfn[u];
4     if(!lson[u]) return;
5     hld(lson[u], u);
6     for(int i = h[u]; i != -1; i = ne[i]) {
7         int v = e[i]; if(v == fa || v == lson[u]) continue;
8         hld(v, u);
9     }
```

¹<https://oi-wiki.org/graph/hld/>

10 }

f_u 存储了点 u 在 dp 数组 (g) 中的首指针, 由于我们在预处理时优先走到重儿子, 所以一条重链的 f 数组是连续的. 接下来是最终处理答案:

```

1 void solve(int u, int fa) {
2     if(lson[u]) {solve(lson[u], u); ans[u] = ans[lson[u]] + 1; }
3     f[u][0] = 1; if(f[u][ans[u]] <= 1) ans[u] = 0;
4     for(int i = h[u]; i != -1; i = ne[i]) {
5         int v = e[i]; if(v == fa || v == lson[u]) continue;
6         solve(v, u);
7         for(int j = 0; j < height[v]; j++) {
8             f[u][j + 1] += f[v][j];
9             if(f[u][j + 1] > f[u][ans[u]]) ans[u] = j + 1;
10            else if(f[u][j + 1] == f[u][ans[u]] && j + 1 < ans[u]) ans[u] = j + 1;
11        }
12    }
13 }
```

先处理出重儿子的答案并直接继承, 由于我们实际上用了同一个 dp 数组 g , 因此, 继承时由于原本到重儿子距离为 x 的结点, 到 u 的距离变成了 $x + 1$, 我们需要将 dp 数组整体右移动 1, 即 $dp_{u,i} = dp_{hson_u,i-1}$, 为了避免这些开销, 做到 $\mathcal{O}(1)$ 继承状态, 我们选择直接移动首指针, 从 f_{hson_u} 变为 f_u , 实际上就是首指针向左移动了一位, 这就是之前的预处理所要做的, 并且这样我们就不用显式地做继承操作了.

时间复杂度 $\mathcal{O}(n)$, 最后点 u 的答案即为 ans_u .

「POI2014」酒店 Hotel

给定一张图, 共有 n 个点, 且任意两点间有且仅有一条简单路径将它们连通. 每条边的长度相等.

选出三个不同的点, 使得三个点之间距离两两相等. 求方案数.

$1 \leq n \leq 5 \times 10^6$, 原数据范围为 5000.

Solution: 「POI2014」酒店 Hotel

钦定 1 号点为根, 注意到三个点的 lca 一定不可能是其中的某一个, 因此本质上只有一种情况:

设 $f_{u,i}$ 表示以 u 为根的子树中到 u 的距离为 i 的点的数量, $g_{u,i}$ 表示以 u 为根的子树中, 满足还需一个到 u 距离为 i 的点就能构成合法三元组的点对个数.

边算每棵子树边转移, 初始时 $f_{u,0} = 1$, 设当前算完了子树 v , 则依次更新 (一定要先更新 g 再更新 f)

$$g_{u,i} \leftarrow g_{u,i} + g_{v,i+1} + f_{u,i} \times f_{v,i-1}$$

$$f_{u,i} \leftarrow f_{u,i} + f_{v,i-1}$$

在更新之前, 我们先维护答案:

$$ans \leftarrow ans + f_{u,i} \times g_{v,i+1} + f_{v,i} \times g_{u,i-1}$$

遍历完之后, 答案也就更新结束了, 用长链剖分优化, 时间复杂度 $\mathcal{O}(n)$.

第三部分

图论

第 6 章 图论基础

6.1 最短路问题

给定一张边带权的图（有向或无向均可）和起点，问对于所有点，从起点到它的路径的最小权值和是多少。

6.1.1 Dijkstra

Dijkstra 算法只能在所有边的权值都非负的情况下得到正确答案，下面的讨论都默认无负权边。

想象在起点 s 处放一桶水，边的权值看作距离，水会沿着边以一定的速度流动，会最先流到哪个点呢？毫无疑问是离起点最近的点，此时，起点到这个点的最短路就已经确定了，不妨设这个点为 u_0 。转化成计算机风格的语言就是：最开始 s 到其他点的最短路长度都是 $+\infty$ ，枚举起点 s 的所有出边，更新 s 到这些点的最短路长度，然后在这些点中选出一个最短路最短的点，标记最短路。

接下来水会流到哪里去？原本从 s 流出的水会接着流，而流到 u_0 的水会在这张照片上变成一个新起点，向 u_0 的所有出边流水。

维护水流到每个点的剩余时间，每次取出最小的并更新，这个过程就是 Dijkstra 算法。其伪代码如下：

Algorithm 1: Dijkstra Algorithm

Data: Graph G which $G[u]$ contains all the edges that started at u , the starting point s and the number of points n .

Result: Get array dist , $\text{dist}[u]$ means the shortest path from s to u .

```
1 for  $i \leftarrow 1$  to  $n$  do
2    $\text{dist}[i] \leftarrow +\infty, \text{st}[i] \leftarrow \text{False};$ 
3 end
4  $\text{dist}[s] \leftarrow 0;$ 
5 for  $i \leftarrow 1$  to  $n$  do
6   for  $j \leftarrow 1$  to  $n$  do
7     if  $\text{st}[j] == \text{False}$  and  $\text{dist}[j] < \text{dist}[u]$  then
8        $u \leftarrow j;$ 
9     end
10     $\text{st}[u] \leftarrow \text{True};$ 
11    for  $e : G[u]$  do
12       $\text{dist}[e.\text{endpoint}] \leftarrow \max \{ \text{dist}[v], \text{dist}[u] + e.\text{weight} \}$ 
13    end
14  end
15 end
```

时间复杂度 $\mathcal{O}(n^2)$ ，注意到每次“找到 dist 最小的点”这个操作可以用小根堆实现，于是在稀疏图中可以将时间复杂度优化至 $\mathcal{O}(m \log m)$

6.1.2 Bellman-Ford

对边 e 的**松弛操作** (relaxation) 是指，更新 $\text{dist}_v = \min \{ \text{dist}_v, \text{dist}_u + w \}$ ，其中 u 是 e 的起点， v 是 e 的终点， w 是 e 的权值，成功松弛是指，经过本次松弛操作后， dist_v 发生了变化。

一种很自然的想法是：初始时 $\text{dist}_s = 0$ ，其他为 $+\infty$ ，不断对每一条边进行松弛操作，直到每一条边都不能成功松弛，此时我们就得到了起点到每个点的最短路。

最多会松弛多少条边呢？在每一轮松弛中，我们对每条边松弛一次，那么最多进行 n 轮松弛就够了，这是因为最短路一定不会两次经过同一个点。

但真的是这样吗？在权值非负的情况下这是成立的，但假如权值出现了负数就有所改变了：最短路甚至可能不存在！例如，起点有一条到自己的边，权值是 -1 ，那么我们就可以不断走这条边，让最短路变成 $-\infty$ 。由此得出，存在最短路当且仅当从起点到该点的所有路径中，不存在一个权值和为负数的环，否则该点的最短路为 $-\infty$ 。

Algorithm 2: Bellman-Ford Algorithm

Data: All edges and the number of points n .

Result: Get array dist , $\text{dist}[u]$ means the shortest path from s to u .

```

1 for  $i \leftarrow 1$  to  $n$  do
2    $\text{dist}[i] \leftarrow +\infty$ ;
3 end
4  $\text{dist}[s] \leftarrow 0$ ;
5 for  $i \leftarrow 1$  to  $n$  do
6   foreach edge  $e$  do Relaxation( $e$ );
7 end
```

时间复杂度为 $\mathcal{O}(nm)$ 。当 n 轮松弛后仍可松弛，说明图中存在负环。注意到只有当边的起点的 dist 发生变化时，这条边才需要再次松弛，所以我们可以用队列维护所有起点，每次取出一个，松弛其出边，直到队列为空，这种优化后的 Bellman-Ford 算法在国内被称为 spfa。

6.1.3 Floyd

Floyd 算法可以在 $\mathcal{O}(n^3)$ 的时间复杂度内求**全源最短路**，即任给两点 u, v ，求出以 u 为起点到 v 的最短路长度，它相较于 n 次 Dijkstra 的优势在于核心代码只有一个语句，十分好写。

Floyd 基于 dp，设 $dp_{u,v,k}$ 指在中途经过的点的编号不超过 k 的情况下 u 到 v 的最短路，初始时， u 到 v 若有边，则 $dp_{u,v,0}$ 即为这些边的最小权，否则为 $+\infty$ 。

转移是容易的： $dp_{u,v,k} = \min \{dp_{u,v,k-1}, dp_{u,k,k-1} + dp_{k,v,k-1}\}$ ，可以证明：去掉 k 这一维直接转移仍是可行的。

Algorithm 3: Floyd Algorithm

Data: The whole graph in a 2d array G and number of the points n .

Result: Get array dist , $\text{dist}[u][v]$ means length of the shortest path from u to v .

```

1 for  $i \leftarrow 1$  to  $n$  do
2   for  $j \leftarrow 1$  to  $n$  do
3     if there is an edge from  $i$  to  $j$  in graph  $G$  then
4        $\text{dist}[i][j] \leftarrow G[i][j]$ 
5     else
6        $\text{dist}[i][j] \leftarrow +\infty$ 
7     end
8   end
9 end
10 for  $k \leftarrow 1$  to  $n$  do
11   for  $i \leftarrow 1$  to  $n$  do
12     for  $j \leftarrow 1$  to  $n$  do
13        $\text{dist}[i][j] \leftarrow \min \{\text{dist}[i][j], \text{dist}[i][k] + \text{dist}[k][j]\}$ 
14     end
15   end
16 end
```

第 7 章 点分治

7.1 静态点分治

Luogu P3806

给定一个 n 个点的带边权树, 询问 m 次, 每次询问树上是否存在一个距离为 k 的点对.

Solution: Luogu P3806

什么是点分治呢? 我们以这道题为例, 向读者介绍这一树上问题的优美算法.

对于根 $Root$, 所有经过 $Root$ 的路径可以被分为两类: 一端点为 $Root$ 的路径和两端点均不为 $Root$ 的路径, 而后者可以由前者拼接而成——这是显而易见的.

受此启发, 我们可以设计一个朴素算法, 流程如下:

首先任意选定一个根, 设为 u , 在 dfs 的过程中不断统计以 u 为一个端点的路径的权值, 每搜完一个子节点, 遍历所有询问, 假如可以拼接成功, 将该询问的答案标记为 $True$.

复杂度分析: 点分治过程中, 每一层的所有递归过程合计对每个点处理一次, 假设递归层数为 h , 那么总的时间复杂度为 $O(nh)$.

显然 h 是可以劣到 $O(n)$ 的, 于是我们考虑将树重构, 具体来说, 我们每次**选择子树的重心作为根节点**, 可以保证递归层数为 $O(\log n)$, 总复杂度 $O(n \log n)$.

```
1 bool vis[maxv];
2 int Root = 0, Sum = 0, sz[maxn], maxsz[maxn];
3 void Calc_Size(int u, int fa) {
4     sz[u] = 1, maxsz[u] = 0;
5     for(int i = h[u]; i != -1; i = ne[i]) {
6         int v = e[i];
7         if(v == fa || vis[v]) continue;
8         Calc_Size(v, u);
9         sz[u] += sz[v], maxsz[u] = max(maxsz[u], sz[v]);
10    }
11    maxsz[u] = max(maxsz[u], Sum - sz[u]);
12    if(maxsz[u] < maxsz[Root]) Root = u;
13 }
```

想必读者已经知道这个函数的作用了: sz_u 记录以 u 为根的子树大小, $maxsz_u$ 记录删去 u 后, 所得图的最大连通块的大小, 是求树的重心的核心数组. 与正常的求树的重心不同的地方在于, 判断是否搜某个邻点的时候, 多了一个判断: vis_v 是否为真. vis_v 的含义是什么呢? 实际上, 正因为我们在点分治的过程中不断地选根, 这些根又无法真正地删去, 是为了不重复遍历已经确定了的根, 我们选择用 vis 数组来记录节点是否已经被确定为某个根.

这样的判断在之后的函数中会频繁出现, 我们不再重复强调.

$Root$ 是全局变量, 记录当前计算的子树的重心, Sum 则是当前正在计算的子树的总大小, 初始为 n .

我们不再对上述代码做过多的介绍, 假如读者对某些地方有疑问, 请回看“树的重心”一节.

与之十分类似的, 是这样一个函数:

```
1 int dist[maxn];
```

```

2 vector<int> dists;
3 void Calc_Dist(int u, int fa) {
4     dists.push_back(dist[u]);
5     for(int i = h[u]; i != -1; i = ne[i]) {
6         int v = e[i];
7         if(v == fa || vis[v]) continue;
8         dist[v] = dist[u] + w[i];
9         Calc_Dist(v, u);
10    }
11 }

```

就像它的函数名一样，该函数的作用是计算当前正在考虑的子树的每个节点到根的距离，同时记录在 *dists* 中。

接下来才是分治的核心：

```

1 void Dfs(int u, int fa) {
2     vis[u] = true;
3     tf[0] = true; tags.push(0);
4     for(int i = h[u]; i != -1; i = ne[i]) {
5         int v = e[i];
6         if(v == fa || vis[v]) continue;
7         dist[v] = w[i]; dists.clear();
8         Calc_Dist(v, u);
9
10        //q[i] is the i_th query
11        for(int j = 0; j < dists.size(); j++)
12            for(int k = 1; k <= m; k++)
13                if(q[k] - dists[j] >= 0) ans[k] = (ans[k] || tf[q[k] - dists[j]]);
14        for(int j = 0; j < dists.size(); j++)
15            if(dists[j] < maxv) tf[dists[j]] = true, tags.push(dists[j]);
16    }
17    while(!tags.empty()) {
18        tf[tags.front()] = false; tags.pop();
19    }
20
21    for(int i = h[u]; i != -1; i = ne[i]) {
22        int v = e[i];
23        if(v == fa || vis[v]) continue;
24        Sum = sz[v]; Root = 0;
25        Calc_Size(v, u);
26        Calc_Size(Root, 0);
27        Dfs(Root, u);
28    }
29 }

```

看起来比较复杂，一点点分析。

首先, 我们将 u 确定为根, 代码中体现为 $\text{vis}[u] = \text{true}$. tf 数组的含义是什么呢? tf_x 为真, 意味着在 Dfs 到 v 之前, 已经有至少一条端点为 u , 长度为 x 的路径. 显然, 在我们递归找下一层的根并且继续下去之前, 我们应该将 tf 清空, 但是这里为了保证复杂度正确, 我们 **不应直接执行 `memset`**, 而应维护一个队列, 遍历完之后对于每个队列中的元素, 执行回溯, 即 $\text{tf}[\text{tags.front()}] = \text{false}$; 还有一个细节是 **一定要及时更新 `Sum`**, 否则找到的重心不对.

剩下的部分逻辑较为清晰, 可以简单看作: 计算距离, 维护答案, 找到下一层的根, 递归进行点分治. 主函数中是这样的:

```
1 Sum = n;
2 Calc_Size(1, 0);
3 Calc_Size(Root, 0);
4
5 Dfs(Root, 0);
6
7 for(int i = 1; i <= m; i++)
8     if(ans[i]) cout << "AYE\n";
9     else cout << "NAY\n";
```

Luogu P4178

本题和模板的区别在于查询为给出一个 k , 求距离 $\leq k$ 的点对数量.

Solution: Luogu P4178

最暴力的方法就是直接用树状数组统计小于等于每个值的点对数量, 代替 tf 数组即可.

时间复杂度多一个 \log .

本题还可以通过容斥或染色, 再套用点分治解决, 这里不再展开, 可以到洛谷的题解区查看.

Luogu P2634

询问长度为 3 的倍数的路径个数.

Solution: Luogu P2634

将 tf 改为记录权值模 3 为 0, 1, 2 的路径个数, 维护答案时用乘法原理 + 加法原理即可.

第 8 章 虚树

有时我们会被要求多次处理树上不同点集, 并且点集大小的和在某个范围内的相关问题. 由于多次询问, 假如每次都按照整棵树的操作进行就太浪费了, 这时我们可以用虚树.

【SDOI2011】消耗战

给定一棵 n 个点的树和 q 次询问, 边有边权, 割掉一个边的代价就是其边权. 第 i 次询问给出点集 S_i , 保证 $1 \notin S_i$, 输出割掉某些边后使 S_i 中的点与 1 号点均不连通的最小代价. $2 \leq n \leq 2.5 \times 10^5, 1 \leq m \leq 5 \times 10^5, \sum |S_i| \leq 5 \times 10^5$

Solution: 【SDOI2011】消耗战

下面设树根为 1, 假如单次询问, 这就变成了一个经典的树上 dp 问题: 设 f_i 为只考虑 i 的子树, 它们和 1 号点不连通的代价, 转移平凡.

每次都这样做, 时间复杂度是 $\mathcal{O}(qn)$ 的, 注意到 $\sum |S_i| \leq 5 \times 5 \times 10^5$, 如果我们能将第 i 次查询的时间复杂度从 $\mathcal{O}(n)$ 变成 $\mathcal{O}(|S_i|)$ 或 $\mathcal{O}(|S_i| \log |S_i|)$, 本题就能得到解决了. 我们不改变 dp 策略而去尝试减少整棵树的点数.

只考虑在集合 S_i 中的点以及它们两两之间的 lca 还有 1 号结点, 建出一棵新树, 设 $cost_u$ 为在原树上 u 到根的边的最小费用 (特别地, 根的 $cost$ 为 $+\infty$), 那么在新树上就有

$$f_u = \min\left\{\sum_{v \in son_u} f_v, cost_u\right\}$$

正确地得出了 f_1 , 即答案. 这棵新树就叫**虚树**, 它只包含原树的部分点, 边也不一定是原树中的边, 但在这棵树上我们却能够得出正确的答案.

这棵新树的点数是多少呢? $\mathcal{O}(|S| \log |S|)$, 这是因为每往上多一个 lca, 都需要消耗下面的至少 2 个点, 问题化为怎样迅速地建出虚树 (时间复杂度不超过 $\mathcal{O}(|S| \log |S|)$).

我们用增量构造的方式建出虚树, 在最开始预处理出每个点的 dfs 序 dfn_u , 将 S 中的点按照 dfn 升序排列, 维护一个栈, 表示根到栈顶元素这条链上有用的结点.

最开始虚树中只有根结点 1, 依次往虚树里加入 $S_1, S_2 \dots$ (注意这里的 S 已经排好序), 若栈中元素只有一个 (根), 将 S_i 放入栈顶后返回, 否则栈中有至少两个元素.

设 l 为栈顶结点与当前待加入点的 lca. 若 l 就是栈顶元素, 直接返回 (只在本题, 在别的情况可能是入栈后返回), 否则不断连从栈顶元素的上一个元素到栈顶元素的边并弹出栈顶, 直到栈顶元素的上一个元素在原树上的深度小于 l 的深度或栈中只有一个元素.

这时, 如果栈顶不是 l , 就从 l 向栈顶连一条边并将栈顶设为 l .

最后将 u 入栈.

对每个 S_i 这样操作后, 再将栈中维护的那条链在虚树上连边, 虚树就建好了.

```
1 inline int lca(int u, int v);
2 int dfn[maxn]; //dfs order
3 int stk[maxn], top; //stack and the position of its top
4 inline void virtual_tree_insert(int u) {
5     if(top == 1) {stk[++top] = u; return;}
6     int l = lca(u, stk[top]); if(l == stk[top]) return;
7     while(top > 1 && dfn[stk[top - 1]] >= dfn[l]) AddEdge(stk[top - 1], stk[top]), top--;
8     if(stk[top] != l) AddEdge(l, stk[top]), stk[top] = l;
9     stk[++top] = u;
```



```
10 }  
11  
12 //in main()  
13 sort(S.begin(), S.end(), [](int a, int b) {return dfn[a] < dfn[b]; });  
14 stk[top = 1] = 1;  
15 for(auto x : S) virtual_tree_insert(x);  
16 for(int i = 2; i <= top; i++) AddEdge(stk[i - 1], stk[i]);
```

时间复杂度 $\mathcal{O}(\sum_i S_i \log S_i)$.

习题: CF613D, Luogu P4103, Luogu P7409

第9章 支配树

Luogu P5180 【模板】支配树

给定一张有向图，求从 1 号点出发，每个点能支配的点的个数（包括自己）。
对于 100% 的数据，保证 $n \leq 2 \times 10^5, m \leq 3 \times 10^5$ 。

Solution: Luogu P5180 【模板】支配树

先来解释一下基本的概念：

给定一个有向图 G 和唯一起点 r ，若每一条从 r 到 u 的路径都必须经过点 v ，称 u 被 v 支配或 v 支配 u 。

为了方便，我们定义 $u \rightarrow v$ 表示从 u 到 v 的边，用 $u \rightsquigarrow v$ 表示一条从 u 到 v 的路径。

显然，对于每个点 $u \neq r$ ，它都有两个平凡的支配点 r 和 u ，但也可能有其它的非平凡支配点，假如某个 u 的支配点 $v \neq u$ 满足它被 u 剩下的所有非平凡支配点支配，称 v 为 u 的最近支配点，记做 $\text{idom}(u)$ 。

引理 1：除 r 外的所有点都有一个唯一的最近支配点。

证明：反证法，假如 u 有两个最近支配点，不妨设为 v, w 。由于 v 是支配点，因此所有从 r 到 u 的路径都可以写成 $r \rightsquigarrow v \rightsquigarrow w \rightsquigarrow u$ 或 $r \rightsquigarrow w \rightsquigarrow v \rightsquigarrow u$ 。假如两种路径同时存在，根据第一种路径存在，我们得知存在一条 w 到 u 的路径，同时不经过 v ，又根据第二种路径存在，我们得知存在一条 r 到 w 的路径，同时不经过 v ，因此我们得到了 $r \rightsquigarrow w \rightsquigarrow u$ 的路径，它没有经过 v ，因此 v 不支配 u ，矛盾！因此若存在若干个支配点，它们一定是支配与被支配的关系（即上述两种路径只会存在一种），根据支配的传递性质可以得知，若存在支配点，则一定存在唯一的最近支配点。另一方面，由于 r 一定支配 u ，所以 u 至少存在一个支配点，因此对于任意 $u \neq r$ ， u 都有唯一的最近支配点，证毕！

对每个 $u \neq r$ ，在新图上连边 $\text{idom}(u) \rightarrow u$ ，那么新图就是一棵树，指定 r 为根，那么树边就是父亲指向儿子，意义为每个点会支配其子树中的点，这棵树就叫做支配树。我们的目标就是在 $\mathcal{O}((n+m)\alpha(n,m))$ 或 $\mathcal{O}((n+m)\log n)$ 的时间复杂度内用 Lengauer-Tarjan 算法求出支配树（两种复杂度取决于并查集的实现）。

首先，我们以 r 为根对图跑出一棵 dfs 树，设点 u 的 dfs 序为 dfn_u ，设 dfs 序为 x 的点的编号为 idfn_x ，称从祖先指向后代的非树边为前向边，后代指向祖先的边为后向边，从一棵子树指向另一棵子树的为横叉边。

引理 2 (路径引理)：若点 u, v 满足 $\text{dfn}_u \leq \text{dfn}_v$ ，那么任意 $u \rightsquigarrow v$ 必然经过它们的公共祖先（不一定是 LCA）。

证明：删去所有公共祖先，若 u 能到 v ，那么 u 就应该在 dfs 的时候成为 v 的祖先，而此时 $u \rightsquigarrow v$ 显然经过了 u 。

定义点 u 的半支配点 $\text{sdom}(u)$ 为 dfn 最小的，只经过 dfn 比 u 大的点（不包含起点、终点）可以到达点 u 的点，形式化地有

$$\text{sdom}(u) = \text{idfn}_{\min(\text{dfn}_v)}, \exists v \rightarrow v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_k, v_k = u, \forall i \in [1, k-1], \text{dfn}_{v_i} > \text{dfn}_u$$

引理 3：对于任意 $u \neq r$ ， $\text{idom}(u)$ 是 u 在 dfs 树上的祖先。

证明：反证，假如 $\text{idom}(u)$ 不是 u 的祖先，这时候显然有一条路径 $r \rightsquigarrow \text{LCA}(u, \text{idom}(u)) \rightsquigarrow u$ 的路径，不经过 $\text{idom}(u)$ ，与 $\text{idom}(u)$ 支配 u 矛盾！

引理 4：对于任意 $u \neq r$ ，必然有 $\text{dfn}_{\text{sdom}(u)} \leq \text{dfn}_u$ 。

证明：当 $\text{dfn}_{\text{sdom}(u)} > \text{dfn}_u$ 时，选取 $\text{sdom}(u)$ 的父亲，它可以作为更优的 $\text{sdom}(u)$ （第一步走到原先的 sdom ，接着按照原先的路径走）。

引理 5：对于任意 $u \neq r$ ， $\text{sdom}(u)$ 是 u 在 dfs 树上的祖先。

证明：反证法，根据引理 4， $\text{dfn}_{\text{sdom}(u)} \leq \text{dfn}_u$ ，此时 $\text{sdom}(u)$ 只经过 dfn 不小于 dfn_u 的点到达 u 的路径的第一步必然不能是走树边（ dfn 比 u 小），也必然不能是前向边、后向边、横叉边（原因一样），因此 $\text{sdom}(u)$

不能是 u 的 sdom , 矛盾!

引理 6: 对于任意 $u \neq r$, 必然有 $\text{idom}(u) \neq u$, $\text{sdom}(u) \neq u$.

证明: 若 $\text{sdom}(u) = u$, 取新的 $\text{sdom}(u)$ 为原来的 $\text{sdom}(u)$ 的父亲会更优; 至于 $\text{idom}(u) \neq u$, 已经在定义中说明.

引理 7: 对于任意 $u \neq r$, $\text{idom}(u)$ 是 $\text{sdom}(u)$ 在 dfs 树上的祖先.

证明: 根据引理 3,5, $\text{idom}(u)$ 和 $\text{sdom}(u)$ 互为祖先后代关系, 若 $\text{sdom}(u)$ 是 $\text{idom}(u)$ 的祖先, 那么就存在一条不经过 $\text{idom}(u)$ 从 r 到 $\text{sdom}(u)$ (通过树边) 再到达 u (通过 dfn 比 $\text{idom}(u)$ 更大的点) 到达 u 的方案, 与 $\text{idom}(u)$ 支配 u 矛盾!

上述引理 3,5,7 共同说明了:

定理 1: 对于任意 $u \neq r$, $\text{idom}(u)$ 是 $\text{sdom}(u)$ 的祖先, $\text{sdom}(u)$ 是 u 的祖先.

引理 8: 若 $u \neq v$ 且 u 是 v 在 dfs 树上的祖先, 那么必然有 u 是 $\text{idom}(v)$ 的祖先或者 $\text{idom}(v)$ 是 $\text{idom}(u)$ 的祖先.

证明: 否则的话, 这条链形如 $r \rightsquigarrow \text{idom}(u) \rightsquigarrow \text{idom}(v) \rightsquigarrow u \rightsquigarrow v$, 其中走的边都是树边, 那么我们就找到了一条路径 $r \rightsquigarrow \text{idom}(u) \rightsquigarrow u \rightsquigarrow v$ 绕过了 $\text{idom}(v)$ 到达 v (因为 $\text{idom}(v)$ 不支配 u , 所以可以从 $\text{idom}(u)$ 不经过 $\text{idom}(v)$ 到达 u), 矛盾!

为了可读性, 定义 $u \Rightarrow v$ 表示 u 可以通过树边走到 v , 即 u 在 dfs 树上 v 的祖先.

定理 2: 对于任意 $u \neq r$, 若所有满足 $\text{sdom}(v) \Rightarrow v \Rightarrow u$ 的点 v 都满足 $\text{dfn}_{\text{sdom}(v)} \geq \text{dfn}_{\text{sdom}(u)}$, 那么 $\text{idom}(u) = \text{sdom}(u)$.

证明: 其实 $\text{dfn}_{\text{sdom}(v)} \geq \text{dfn}_{\text{sdom}(u)}$ 就是 $\text{sdom}(u) \Rightarrow \text{sdom}(v)$, 我们只需要证明 $\text{sdom}(u)$ 可以支配点 u , 又由于 $\text{idom}(u) \Rightarrow \text{sdom}(u)$, 就可以得到 $\text{idom}(u) = \text{sdom}(u)$ 了.

「未完成」

第 10 章 网络流

10.1 常见建模技巧

多源多汇最大流

很套路，建一个超级源，向所有源点连容量为 $+\infty$ 的边，再建一个超级汇，从所有汇点向超级汇连容量为 $+\infty$ 的边。

结点有容量限制的最大流

常见套路，简单拆点就行，具体来说，把每个点 u 拆为两个点 u_1 和 u_2 ，从 u_1 向 u_2 连一条容量为 u 的容量的边，对于所有 u 的出边，改为从 u_2 出发，对于所有 u 的入边，改为 u_1 的入边。

无源无汇有容量下界的可行流

由于无源无汇 + 每条边有容量上界 (c)、下界 (b)，仅是求出可行流就已经有难度了（若无下界，零流就是可行流，没有任何用处）。

做法同样是改造弧，可以发现，每条弧的实际可操控的流量其实是 $c - b$ ，我们必须跑满下界，这启发我们在满足下界的情况下找到流量平衡的方法。

如果建出**下界网络**¹，这时还未满足流量平衡，我们决定把锅甩给源汇点，因为他们的流量是可以不平衡的。

考虑新建源点 s 和汇点 t ，计算出每个点 u 的流入量 $\text{in}(u)$ 和流出量 $\text{out}(u)$ ，其中流入量指所有指向 u 的弧的容量和，流出量指所有从 u 出发的弧的容量和，并求出 $\delta(u) = \text{in}(u) - \text{out}(u)$ 。

重新梳理一下思路，现在，对于每个点 u ，我们知道了它积蓄（或欠缺）的流量 $\delta(u)$ ，并且每条弧有容量限制 $c - b$ ，现在要找到一种方案，让流量平衡。

找到我们附加的源点 s 和汇点 t ，若 $\delta(u) > 0$ ，则从 s 到 u 附加一条容量为 $\delta(u)$ 的弧，若 $\delta(u) < 0$ ，则从 u 到 t 附加一条容量为 $-\delta(u)$ 的弧。

对于原图中上界为 c ，下界为 b 的边，在新图中，容量为 $c - b$ 。

最后，在新图中跑 s 到 t 的最大流，当且仅当所有附加弧都流满时，原图有可行流。其中，每条边的可行流量为下界 + 新图中的流量。

有源汇有上下界可行流

在无源汇上下界可行流的基础上，我们来讨论有源汇的情况。对于这种情况，我们只需要建一条新边 (t, s) ，容量下界是 0，上界是 $+\infty$ ，然后跑一遍无源汇可行流即可，最终， t 到 s 那条边上的流量即为一个可行流。

注意我们在跑无源汇可行流时附加的源点和汇点并非给定的源汇，而是我们又附加上去的新点，不要混淆。

有源汇有上下界最大流

不要慌，我们已经找到了一个可行解了，接下来只需要不断扩大流量。方法很简单，在新图中跑 s 到 t 的增广路即可，注意这里的 s 和 t 是给定的源点和汇点，并非我们为了跑无源汇可行流而新建的点。

最大权闭合子图

首先了解一下什么是闭合子图，实际上就是图 (V', E') ，同时满足：

- $V' \subseteq V, E' \subseteq E$
- $u \in V', (u, v) \in E \implies v \in V' \text{ and } (u, v) \in E'$

最大权闭合子图就是所有点的权值和最大的闭合子图。

建模方法：

- 将所有点分为两类，集合 X 中的点的权值大于 0， Y 中的权值小于 0，权值恰好为 0 的点在哪个里都可以。
- 新建一个图，附加源点 s 和汇点 t ，从 s 向 X 中的点建弧，权值为点权，从 Y 中的点向 t 建弧，权值为点权的绝对值。
- 对于原图中的弧，在新图中的容量为 $+\infty$ ，入点和出点不变。

¹下界网络：将下界为 b 的弧改造为容量为 b 的弧。

- 新图的每一个割都对应了原图的一个闭合子图, 具体来说, 由于割去的只能是容量不为 $+\infty$ 即非原图中的弧, 我们说割掉一个点, 即为割掉了这个点与 s 或 t 之间的弧. 那么 X 中没有被割掉的点与 Y 中被割掉的点构成了原图的一个闭合子图.

原理不难理解, X 中没有被割掉的点经过原图中的边只能到达 Y 中被割掉的点 (否则这就不是一个合法的割).

怎样让权值最大呢? 只需要找到最小割, 用所有权值为正的点的权值和减去最小割就是最大权了. 原理很简单, 把所有点分为四类: 在子图里的 X 点 (权和绝对值为 A), 不在子图里的 X 点 (权和绝对值为 B), 在子图里的 Y 点 (权和绝对值为 C), 不在子图里的 Y 点 (权和绝对值为 D), 我们需要的就是让 $A - C$ 最大化, 而 $A + B$ 是个常数, 最小割实际上就是最小的 $B + C$, 那么 $A - C$ 最大实际上就是 $(A + B) - (B + C)$ 最大, 即 $B + C$ 最小, 那么最大权显然就是 X 中点的权值和 $(A + B)$ 减去最小割了.

第四部分

字符串

第 11 章 后缀数据结构

11.1 后缀自动机

处理字符串问题时，很多时候假如我们能够将这个字符串设计成一张 DAG，边代表字符，每条路径都对应原串的一个子串，问题就可以在图上得到很好的解决。

怎么做呢？把每个后缀插入到 Trie 不就行了吗？而且这还是一棵有向树的结构，设一个点代表的字符串就是从根结点到它的路径上的边表示的字符顺次连接构成的字符串，那么两个点的 lca 代表的就是这两个点代表的字符串的 LCP（最长公共前缀）。

看起来确实很好，但这个 Trie 会有多少个点呢？ $\mathcal{O}(n^2)$ ，这样一看这个结构就很没用了，因为还不如我们在字符串上直接处理来的方便，那有没有一种方式来减少点数呢？这就是**后缀自动机** (Suffix Automaton, SAM)，它用 $\mathcal{O}(n)$ 的空间复杂度达到了和上面所说的 Trie 同样的效果。

先看一下 SAM 的结构：SAM 是一张 DAG，结点被称作**状态**，边被称作**转移**，转移标有字符，并且从某个点出发的所有转移上的字符均不同。SAM 存在一个特殊状态：源点 t_0 ，保证其它结点均可从 t_0 到达，存在若干个终止状态，使得字符串的每个后缀和从 t_0 到某个终止状态的路径一一对应。

在保证了上述性质的前提下，SAM 是点数最少的。

在下面的讨论中，我们设字符集为 Σ ，源点为 t_0 ，将要构造的是字符串 s 的后缀自动机且 s 的下标从 0 开始，字符串的子串是原串连续的一部分。

11.1.1 endpos 等价类

定义一个子串的 endpos 是一个集合，集合中元素时其在原串中的每个出现位置（以末尾元素为基准），例如对于串“aababaa”，子串“aa”的 endpos 就是 $\{1, 6\}$ ，子串“ab”的 endpos 就是 $\{2, 4\}$ ，子串“a”的 endpos 就是 $\{0, 1, 3, 5, 6\}$ ，特别地，定义空串的 endpos 是每个位置，在这个例子中为 $\{0, 1, 2, 3, 4, 5, 6\}$ 。

在标题中，我们说了“endpos 等价类”，在数学一章的代数结构中，我们讲解了等价关系与等价类的概念，那么对于两个子串 s_1, s_2 ，定义 $s_1 R s_2$ 当且仅当它们的 endpos 相同， R 是否是一个等价关系呢？答案是肯定的，其自反性、对称性和传递性都是显然的。

能不能再给力一点啊？endpos 好像什么用都没有啊！

性质 对于两个子串 s_1, s_2 ，它们的 endpos 之间只可能为以下关系中的两种：

1. $\text{endpos}(s_1) \subseteq \text{endpos}(s_2)$ 或 $\text{endpos}(s_2) \subseteq \text{endpos}(s_1)$ 。
2. $\text{endpos}(s_1) \cap \text{endpos}(s_2) = \emptyset$ 。

这个性质告诉我们，两个子串的 endpos 只可能互相包含或者无交。

证明 若 $\text{endpos}(s_1) \cap \text{endpos}(s_2) \neq \emptyset$ ，说明两个串都在某个位置出现了，那么较短的那个串（不妨让它是 s_1 ）一定是较长的那个串（ s_2 ）的后缀。

因此，对于每个 s_2 出现过的位置， s_1 必然出现，故 $\text{endpos}(s_2) \subseteq \text{endpos}(s_1)$ 。

性质 两个非空子串 s_1, s_2 若满足 $\text{endpos}(s_1) = \text{endpos}(s_2)$ （假设 $|s_1| \leq |s_2|$ ），当且仅当 s_1 每次在 s 中出现都是以 s_2 的后缀的形式。

性质显然。

性质 将所有 endpos 相同的串按照长度大小升序排列，所有长度一定恰好构成一个区间 $[l, r]$ ，即这些串的长度分别是 $l, l+1, l+2, \dots, r$ 且每个长度更小的串都是长度更大的串的后缀。

证明 注意到串的长度一定不会相同，因为长度相同且 endpos 相同等价于两个串相同。设在该等价类中长度为 x 的串为 s_x ，那么：

当 endpos 等价类中只有一个元素时，命题显然成立，否则设所有长度构成的集合为 P ，若有 $x \notin P$ 且 $l < x < r$ ，取长度为 r 的串的长度为 x 的后缀（设为 s'_x ），由于 $x > l$ ，故 s_l 是 s'_x 的后缀，那么 $\text{endpos}(s'_x) \subseteq$

$\text{endpos}(s_l)$, 又由于它是 s_r 的后缀, 因此 $\text{endpos}(s_r) \subseteq \text{endpos}(s'_x)$, 两式联立, 因为 $\text{endpos}(s_l) = \text{endpos}(s_r)$, 于是 $\text{endpos}(s'_x) = \text{endpos}(s_l) = \text{endpos}(s_r)$.

例如对于串 "abcababc", 有 $\text{endpos}("abc") = \text{endpos}("bc") = \{2, 7\}$, 但它不等于 $\text{endpos}("c")$, 还有 $\text{endpos}("ab") = \text{endpos}("b") = \{1, 4, 6\}$.

现在, endpos 等价类就可以发挥它的作用了, 我们让 SAM 的每个结点表示一个 endpos 等价类, 即 endpos 相同的那些串, 定义状态 u 中的最短串长度为 $\text{minlen}(u)$, 最长串长度为 $\text{len}(u)$ 状态 u 代表的串中长度为 $\text{minlen}(u) - 1$ 的后缀所在的结点称为 $\text{link}(u)$. 例如在上面的例子中, 设代表 "abc" 和 "bc" 这个 endpos 等价类的状态为 u , 那么 $\text{link}(u)$ 就是代表 "c" 这个串的 endpos 等价类的状态, 设代表 "ab" 和 "b" 这个 endpos 等价类的状态为 v , 那么 $\text{link}(v)$ 就是代表空串的 endpos 等价类的那个状态, 即 t_0 .

称 link 为**后缀链接**, 那么所有后缀链接构成了一棵以 t_0 为根的有向树 (称作 **parent Tree**), 树边是从儿子指向父亲的, 因为 link 只会从长度大的串指向长度小的串. 沿着后缀链接从 t_0 向下走到叶子, 这些状态代表的串的长度区间的并恰好就是 $[0, |s|]$, 这是显然的, 因为我们有 $\text{minlen}(u) - 1 = \text{len}(\text{link}(u))$.

现在我们讲解 SAM 的构造方法, 这是一个在线算法, 具体来说, 我们是依次往 SAM 中加入每个字符的.

定义每个状态为一个结构体:

```
1 struct state {
2     int len, link;
3     std::map<char, int> next;
4 };
5 const int maxs = 1e5 + 10;
6 state st[maxs * 2];
7 int sz = 0, last = 0;
```

其中, len 和 link 的含义和我们之前的内容是完全一致的, next 存储了该状态的所有出边, 当字符集 Σ 很小时, 我们也可以直接开成数组. maxs 指的是字符串的最大长度, 注意, 我们之前说 SAM 的结点数是 $\mathcal{O}(n)$ 的, 但在这里我们不能将状态个数开成 n , 因为准确来说, SAM 的结点数不会超过 $2n - 1$, 因此我们应该开 2 倍, 这个性质在代码中可以很清晰地体现出来.

sz 指的是当前结点最大编号, 新开结点时要用到, last 指我们上次加入字符时末尾结点的编号, 即之前的整个串所在状态的编号.

SAM 的初始化写成函数时这样的:

```
1 inline void init() {st[0].len = 0, st[0].link = -1, sz = 1, last = 0; }
```

$\text{st}[0]$ 代表空字符串, 即 t_0 , 空字符串的 link 是特别定义的. 接下来给出往字符串末尾加入字符, 维护 SAM 的函数:

```
1 inline void insert(char c) {
2     int cur = sz++;
3     st[cur].len = st[last].len + 1;
4     int p = last;
5     while(p != -1 && !st[p].next.count(c)) {
6         st[p].next[c] = cur;
7         p = st[p].link;
8     }
9     if(p != -1) {
10        int q = st[p].next[c];
11        if(st[p].len + 1 != st[q].len) {
12            int clone = sz++; st[clone] = st[q], st[clone].len = st[p].len + 1;
```



```

13     while (p != -1 && st[p].next[c] == q) {
14         st[p].next[c] = clone;
15         p = st[p].link;
16     }
17     st[q].link = st[cur].link = clone;
18 } else st[cur].link = q;
19 } else st[cur].link = 0;
20 last = cur;
21 }

```

首先，由于我们往原串的末尾加入了一个字符 c ，现在的整个新串在原 SAM 中是一定没有状态表示的，因此我们一定要新开一个状态 cur 表示现在的整个新串，它的 len 是容易得到的，就是原串长度 $+1$ 而已， $next$ 也很容易：就是空，关键在于 $link$ 。

怎么得到 $link$ 呢？回想一下 $link$ 的定义，我们要找到一个新串的后缀使得它的 $endpos$ 和整个新串不同，注意到新串的后缀一定是原串的后缀再加上一个字符 c ，因此我们从整个原串，即 $last$ 开始，不断向上跳 $link$ ，直到该状态有一个字符为 c 的出边，在这同时，将那个没有字符 c 的边的原串后缀加一条字符 c 的边到新串所在状态，在代码中对应第 4 到第 8 行。

如果没有找到，说明原串根本就没有一个符合条件的后缀，于是 $link(cur)$ 就是空串，即 t_0 ，在代码中为 0 号结点，更新 $last$ 为 cur 后就结束了。

否则我们就找到了一个状态 p 满足 p 代表的字符串均为原串的后缀，且有一条字符为 c 的出边，并且在这样满足条件的状态中， $len(p)$ 是最大的。设状态 p 代表的字符串中长度最大的那个串为 s_p ，状态 p 通过字符为 c 的边指向的结点为 q 。

可以直接将 $link(cur)$ 设为 q 吗？看起来不错，但是仔细想想就会发现出问题了！要知道，我们从 t_0 出发沿着 SAM 边走，得到的是原串的一个子串，这也就是说， $s_p + c$ （这里的字符串加法指的是直接将两个字符串拼接起来）可能仅仅是 q 的最长串的一个后缀而已， q 的最长串并不一定是新串的后缀。

这时我们再分类讨论，假如 $len(q) = len(p) + 1$ ，说明 $s_p + c$ 就是 q 的最长串，因此 q 代表的字符串同样就是新串的后缀并且在之前出现过，因此它的 $endpos$ 不仅仅是新串长度，所以 $link(cur) = q$ 是正确的，执行这个操作后直接返回即可，这对应了代码中的第 18 行。

否则， q 代表的最长串一定不是新串的后缀，我们的 $link(cur)$ 一定是一个满足其状态最长串为 $s_p + c$ 的结点，但 SAM 中是不存在这样的结点的，因此需要新建一个结点 $clone$ 并复制 q 的信息到 $clone$ ($link$ 和 $next$)，同时 $clone$ 的 len 应该为 $len(p) + 1$ ，即 $|s_p + c|$ 。

此时 $link(q) = clone$ ，这是因为 q 的最长串的后缀 $s_p + c$ 的 $endpos$ 多出了一个值； $link(cur) = clone$ ，这也是显然的。接下来还应该维护什么东西呢？对于原串的更小的后缀，若原本它的 $next[c]$ 指向了 q ，我们就将它重定向到 $clone$ ，然后就做完了。

很令人惊讶的是，这个 SAM 的在线构造方式是 $\mathcal{O}(n)$ 的！线性（这里假设 $std::map$ 的是 $\mathcal{O}(1)$ 的，因为 $\log \Sigma$ 一般很小）！下面给出证明（在证明中，同样认为 $|\Sigma|$ 是常数级别）。

首先我们证明，SAM 的转移数不超过 $3n - 4$ （假设 $n \geq 3$ ）。

证明 鸽了，看心情补。如果忘了大概就不补了。

一个字符串的反串建成 SAM 后的 parent tree 就是原串的后缀树 (Suffix Tree)。

第五部分

数学

第 12 章 数论基础

在数学中，整数是很重要的一个研究对象，数论终点研究的就是整数的性质。接下来，假如没有特殊说明，我们默认所有数均为整数。

12.1 基本定义和一个求和技巧

12.1.1 基本定义

在小学的时候，我们就学过“素数”，“因数（约数）”，“倍数”等一些概念，在这里，我们用数学语言来表达它们，定义整除符号：

若 $m > 0$ 并且 $\frac{m}{n} \in \mathbb{Z}$ ，我们就说 n 整除 m ，记作 $n \mid m$ 。

若对于两个数 a, b ，同时有 $k \mid a$ 并且 $k \mid b$ ，那么我们说 k 是 a 和 b 的公因数，公因数中最大的那个称为 **最大公因数 (gcd)**，即

$$\gcd(a, b) = \max_{k \mid a \wedge k \mid b} k$$

显然 1 是任意两个数的公因数， $\gcd(1, n) = 1$ ， $\gcd(n, n) = n$ ，若 $\gcd(a, b) = 1$ ，我们称 a, b **互质**。

类似的，我们还可以定义 **最小公倍数 (lcm)**：

$$\text{lcm}(a, b) = \min_{k > 0 \wedge a \mid k \wedge b \mid k} k$$

当 $a \leq 0$ 或者 $b \leq 0$ 时， $\text{lcm}(a, b)$ 是无定义的，数学中常用 (a, b) 直接表示 $\gcd(a, b)$ 。

两者之间的关系为

$$\gcd(a, b) \times \text{lcm}(a, b) = a \times b$$

证明考虑 $\gcd(a, b) = d$ ，则 $\text{lcm}(a, b) = \frac{a}{d} \times \frac{b}{d} \times d$ 。

相较于 \gcd 而言， lcm 就没有很好的性质，因此在下面的讨论中不会过多涉及 lcm 。

12.1.2 欧几里得算法

辗转相除法，又名欧几里得算法，可以解决求 (a, b) 的问题，具体来说有定理

$$(a, b) = (b, a \bmod b)$$

例如求 $(12, 18)$ ，我们有 $(12, 18) = (18, 12) = (12, 6) = (6, 0) = 6$ ，时间复杂度 $\mathcal{O}(\log n)$ 。定理的证明如下：

若 $d \mid (a, b)$ ，那么有 $d \mid a, d \mid b \implies d \mid a - b$ ，于是 $d \mid (b, a - b)$ 。

同理，若 $d \mid (b, a - b)$ ，那么有 $d \mid b, d \mid a - b \implies d \mid a$ ，于是 $d \mid (a, b)$ 。

于是， a, b 的公因数和 $(b, a - b)$ 的公因数完全相同，因此最大公因数也相等。

利用欧几里得算法，还可以求出不定方程 $ax + by = \gcd(a, b)$ 的所有解，原理如下：

- 若 $b = 0$ ，直接令 $x = 1, y = 0$ 。
- 否则，进行下一步欧几里得算法的流程，称 $a' = b, b' = a \bmod b$ ，假如已经得到 x', y' 使得 $a'x' + b'y' = (a, b)$ ，那么我们就知道了不定方程 $bx + (a \bmod b)y = (a, b)$ 的一组解 x', y' ，注意到 $a \bmod b = a - \lfloor \frac{a}{b} \rfloor b$ ，所以我们知道的其实就是不定方程 $bx + (a - \lfloor \frac{a}{b} \rfloor b)y = (a, b)$ 的一组解，变形得 $ay + \left((x - \lfloor \frac{a}{b} \rfloor b)y \right) = (a, b)$ 。
- 观察上式，可以发现我们要求的 $x = y', y = x' - \lfloor \frac{a}{b} \rfloor y'$

上述算法称为 **扩展欧几里得算法**.

得到了一组特解, 这个不定方程的通解是什么呢? 可以证明它的特解一定形如 $x = x' + k \times \frac{b}{\gcd(a, b)}, y = y' - k \times \frac{a}{\gcd(a, b)}$, 其中, x', y' 是一组特解.

我们得到了裴蜀定理的一部分, 下面给出裴蜀定理:

定理 12.1 (裴蜀定理)

对于不定方程 $ax + by = m$, 其有解的充要条件为 $\gcd(a, b) \mid m$.



另一部分 (必要性) 的证明是显然的, 因为 $\gcd(a, b) \mid ax + by$.

12.1.3 类欧几里得算法初探

通过一种形式类似于欧几里得算法的递归, 我们可以在 $O(\log n)$ 的时间内求出 $\sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor$, 记为 $f(n, a, b, c)$. 既然是递归形式, 我们就把这个式子换一换样子:

$$\sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor = \sum_{i=0}^n \sum_{j=0}^{\lfloor \frac{ai+b}{c} \rfloor - 1} 1 = \sum_{j=0}^{\lfloor \frac{an+b}{c} \rfloor - 1} \sum_{i=0}^n \left[\lfloor \frac{ai+b}{c} \rfloor - 1 \geq j \right]$$

记 $m = \lfloor \frac{an+b}{c} \rfloor$, 根据下取整函数的性质, $\lfloor \frac{ai+b}{c} \rfloor - 1 \geq j$ 等价于 $j+1 \leq \frac{ai+b}{c}$, 移项变一下形式得到

$$\sum_{j=0}^{m-1} \sum_{i=0}^n [ai \geq cj + c - b]$$

逆用取整函数的性质, 艾弗森括号中的不等式又等价于 $ai > cj + c - b - 1$, 等价于 $i > \frac{cj+c-b-1}{a}$, 等价于 $i > \lfloor \frac{cj+c-b-1}{a} \rfloor$, 那么原式化为

$$\sum_{j=0}^{m-1} \sum_{i=0}^n \left[i > \lfloor \frac{cj+c-b-1}{a} \rfloor \right] = \sum_{j=0}^{m-1} \left(n - \lfloor \frac{cj+c-b-1}{a} \rfloor \right) = nm - \sum_{j=0}^{m-1} \lfloor \frac{cj+c-b-1}{a} \rfloor$$

注意到这可以用我们定义的函数表达, 即 $nm - f(m-1, c, c-b-1, a)$.

每次递归, 我们将 n 变为了 $m-1$, 即 $\lfloor \frac{an+b}{c} \rfloor - 1$, 是否有点不对劲? 对, 这个式子的确有可能无限递归下去, 依照欧几里得算法的思路, 我们应当在每次递归的时候加上一个取模操作, 并且设定一个递归终点, 显然 $a=0$ 时原式等于 $(n+1)\lfloor \frac{b}{c} \rfloor$, 是理想的递归终点.

也就是说, 我们如果能让所求变为 $f(n, a \bmod c, b \bmod c, c)$ 这样的形式, 那么这个递归的时间复杂度就和欧几里得算法一致了 (根据 a 和 c 这两个位置, 它们不断互相取模), 怎么变呢? 其实很容易:

$$\begin{aligned} f(n, a, b, c) &= \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor = \sum_{i=0}^n \lfloor \frac{(c\lfloor \frac{a}{c} \rfloor + a \bmod c)i + (c\lfloor \frac{b}{c} \rfloor + b \bmod c)}{c} \rfloor \\ &= \sum_{i=0}^n \left[\lfloor \frac{a}{c} \rfloor i + \lfloor \frac{b}{c} \rfloor + \frac{(a \bmod c)i + b \bmod c}{c} \right] \\ &= \frac{n(n+1)}{2} \lfloor \frac{a}{c} \rfloor + (n+1) \lfloor \frac{b}{c} \rfloor + \sum_{i=0}^n \lfloor \frac{(a \bmod c)i + (b \bmod c)}{c} \rfloor \\ &= \frac{n(n+1)}{2} \lfloor \frac{a}{c} \rfloor + (n+1) \lfloor \frac{b}{c} \rfloor + f(n, a \bmod c, b \bmod c, c) \end{aligned}$$

代码如下, 这里直接用了 `int`, 但很多情况下需要开 `long long`, 改一下即可.

```
1 int f(int n, int a, int b, int c) {
2     if(a == 0) return (n + 1) * (b / c);
3     if(a >= c || b >= c)
```

```

4   return (n) * (n + 1) / 2 * (a / c) + (n + 1) * (b / c) + f(n, a % c, b % c, c);
5   int m = (a * n + b) / c;
6   return n * m - f(m - 1, c, c - b - 1, a);
7 }

```

实际上我们还可以做更多的事, 例如求 $g(n, a, b, c) = \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor i$ 和 $h(n, a, b, c) = \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor^2$, 利用类似的变换, 我们可以得到

$$g(n, a, b, c) = \frac{n(n+1)(2n+1)}{6} \lfloor \frac{a}{c} \rfloor + \frac{n(n+1)}{2} \lfloor \frac{b}{c} \rfloor + g(n, a \bmod c, b \bmod c, c)$$

$$h(n, a, b, c) = h(n, a \bmod c, b \bmod c, c) + 2 \lfloor \frac{b}{c} \rfloor f(n, a \bmod c, b \bmod c, c) + 2 \lfloor \frac{a}{c} \rfloor g(n, a \bmod c, b \bmod c, c)$$

$$+ \lfloor \frac{a}{c} \rfloor^2 \frac{n(n+1)(2n+1)}{6} + \lfloor \frac{b}{c} \rfloor^2 (n+1) + \lfloor \frac{a}{c} \rfloor \lfloor \frac{b}{c} \rfloor n(n+1)$$

这是取模的部分, 另一部分为

$$g(n, a, b, c) = \frac{1}{2} (nm(n+1) - f(m-1, c, c-b-1, a))$$

$$h(n, a, b, c) = \frac{1}{2} nm(m+1) - g(m-1, c, c-b-1, a) - f(m-1, c, c-b-1, a)$$

虽然这三个函数互相调用, 但是它们递归的形式都是一样的, 所以可以对同一组参数一起处理, 求出最终的答案, 时间复杂度仍然是优秀的 $\mathcal{O}(\log n)$.

这里多说一句, 在算 h 的递归式子时, 为了避免两个求和相乘, 可以将 n^2 拆成 $n(n+1) - n = 2(\sum_{i=1}^n i) - n$.

Luogu P5171 Earthquake

给定 a, b, c , 求满足方程 $ax + by \leq c$ 的非负整数解个数。
对于所有数据, $1 \leq a, b \leq 10^9, 0 \leq c \leq \min(a, b) \times 10^9$ 。

Solution: Luogu P5171 Earthquake

变一下形式, 固定 x 后求合法的 y 的数量, 原式即为 $y \leq \lfloor \frac{c-ax}{b} \rfloor$, 由于 y 应当大于等于 0, 那么合法的 x 应当满足 $\lfloor \frac{c-ax}{b} \rfloor \geq 0$, 即 $x \leq \lfloor \frac{c}{a} \rfloor$, 枚举 x , 答案也就是

$$\sum_{x=0}^{\lfloor \frac{c}{a} \rfloor} \lfloor \frac{c-ax}{b} \rfloor + 1$$

能直接套类欧几里得板子吗? 不行, 因为 $-a < 0$, 怎么办呢? 考虑让式加一个 x 再减一个 x , 化为

$$\sum_{x=0}^{\lfloor \frac{c}{a} \rfloor} \lfloor \frac{c+(b-a)x}{b} \rfloor - x + 1$$

如果 $b-a < 0$ 怎么办呢? 交换一下 a, b 就行了, 因为这相当于我们去固定 y 求合法的 x 的数量, 换句话说, 这个约束对于 a, b 是对称的, 接下来就是套板子了。

12.1.4 一个求和技巧

首先是一个简单的求和变换:

$$\sum_{m|n} a_m = \sum_{m|n} a_{n/m}$$

这个等式的正确性是显然的, 因为因数总是成对出现。

还有一个更具一般性的等式，虽然它并没有什么用处：

$$\sum_{m|n} a_m = \sum_k \sum_{m>0} a_m [n = mk]$$

接下来的一个等式才是我们的重点，这个等式会在之后介绍莫比乌斯反演时用到，还请读者掌握：

$$\sum_{m|n} \sum_{k|m} a_{k,m} = \sum_{k|n} \sum_{l|(n/k)} a_{k,kl}$$

证明

$$\begin{aligned} \text{leftside} &= \sum_{j,l} \sum_{k,m>0} a_{k,m} [n = jm] [m = kl] = \sum_j \sum_{k,l>0} a_{k,kl} [n = jkl] \\ \text{rightside} &= \sum_{j,m} \sum_{k,l>0} a_{k,kl} [n = jk] [n/k = ml] = \sum_m \sum_{k,l>0} a_{k,kl} [n = mlk] \end{aligned}$$

12.2 素数

关于素数，我们先前已经有过一定了解，即除了1和它本身，没有其他因数的数，它是数论中的一个重点研究对象，下文中我们一般用 p 来代表一个素数。

称 a, b **互质**（或互素），当且仅当 $(a, b) = 1$ ，可以记作 $a \perp b$ 。

接下来是几个结论：

$$\frac{n}{(n, m)} \perp \frac{m}{(n, m)}$$

$$k \perp n, k \perp m \iff k \perp nm$$

12.2.1 唯一分解定理

定理 12.2 (唯一分解定理)

对于所有 $n > 1$ ， n 一定可以被唯一分解为若干个素数相乘的形式，即

$$n = \prod_{k=1}^m p_k^{\alpha_k}$$



例如， $60 = 2 * 2 * 3 * 5$ 。

分解的“唯一性”从何而来呢？它看起来很显然，事实上并不是这样的，“唯一”并不可以从素数定义直接得到，我们用一个《具体数学》的例子来说明这个问题。

我们定义一个新的数集 $S = \{x \mid x = m + n\sqrt{10}, m, n \in \mathbb{Z}\}$ ，所谓“分解”可以怎么做呢？ $6 = 2 \times 3 = (4 + \sqrt{10})(4 - \sqrt{10})$ ，是不唯一的，我们类似地定义“素数”，即不可再分解的数。现在有一个问题：在这样的定义下，“素数”是否存在。事实上是存在的， $2, 3, 4 + \sqrt{10}, 4 - \sqrt{10}$ 都是这个定义下的“素数”。

我们来证明唯一分解定理的“唯一”：

证明 设

$$n = p_1 p_2 \cdots p_m = q_1 q_2 \cdots q_k, p_1 \leq p_2 \leq \cdots \leq p_m, q_1 \leq q_2 \leq \cdots \leq q_k$$

假设 $p_1 < q_1$ ，则 $\gcd(p_1, q_1) = 1$ ，由裴蜀定理，我们可以找到一组 a, b 满足 $ap_1 + bq_1 = 1$ ，于是 $ap_1 q_2 \cdots q_k + bq_1 q_2 \cdots q_k = q_2 \cdots q_k$ 。

由于 $p_1 \mid ap_1q_2 \cdots q_k$ 并且 $p_1 \mid bn = bq_1q_2 \cdots q_k$, 所以 $p_1 \mid q_2 \cdots q_k$, 于是 $q_2, q_3 \cdots q_k$ 中必然有一个是 p_1 , 这与 $q_1 \leq q_2 \leq \cdots$ 矛盾, 所以 $p_1 \geq q_1$.

类似的, 我们有 $q_1 \geq p_1$, 于是 $p_1 = q_1$.

顺次推下去, 我们有 $p_1 = q_1, p_2 = q_2, \cdots$

证毕.

唯一分解定理是一个基础但有用的定理, 将 a, b 唯一分解得到

$$a = \prod_i p_i^{\alpha_i} \quad b = \prod_i p_i^{\beta_i}$$

那么就有

$$\gcd(a, b) = \prod_i p_i^{\min\{\alpha_i, \beta_i\}}$$

$$\text{lcm}(a, b) = \prod_i p_i^{\max\{\alpha_i, \beta_i\}}$$

12.2.2 素数的个数

首先我们应该说明的是, 素数是有无穷个的, 证明可以用反证法, 将所有的素数乘起来再加 1 一定是一个新的素数.

那在一定的范围内, 素数有多少个呢? 令 $\pi(n)$ 表示区间 $[1, n]$ 之内的素数个数, 我们有 $\lim_{n \rightarrow +\infty} \pi(n) = \frac{n}{\ln n}$

埃式筛:

怎样快速地求出 1 到 n 的所有素数呢? 一种 $O(n \log \log n)$ 的方法叫做**埃拉托斯特尼筛法**, 它的原理是这样的:

对于一个质数, 它的倍数中不是它本身的数都是合数, 于是我们可以用枚举倍数的方式筛出所有的素数. 具体来说, 我们从 2 到 n 枚举, 假如枚举到 p 时, p 没有被筛过, 我们就可以确定 p 是一个素数, 之后, 我们再用 p 来筛掉其他的倍数.

```
1 vector<int> primes;
2 bool vis[maxn];
3 void sieve(int n) {
4     for(int i = 2; i <= n; i++) if(!vis[i]) {
5         primes.push_back(i);
6         for(int j = i; j <= n; j += i)
7             vis[j] = true;
8     }
9 }
```

可以证明, 上述做法的时间复杂度为 $O(n \log \log n)$.

线性筛:

埃式筛没有做到线性的原因是每个数可能会被删去多次, 怎样保证每个数只被删去一次呢? 方法是用每个数的最小质因子来删去它, 这就是**欧拉筛**, 也叫**线性筛**.

```
1 vector<int> primes;
2 bool vis[maxn];
3 void sieve(int n) {
4     for(int i = 2; i <= n; i++) {
5         if(!vis[i]) primes.push_back(i);
6         for(int j = 0; j < primes.size(); j++) {
7             if(i * primes[j] > n) break;
```

```

8     vis[i * primes[j]] = true;
9     if(i % primes[j] == 0) break;
10  }
11  }
12 }

```

保证复杂度线性的关键之处在于 `if(i % primes[j] == 0) break;` 一句, 若条件成立, 说明 `primes[j]` 之后的 `i × primes[k]` 一定有一个质因子 `primes[j]` (来自 `i`), 直接 `break`.

复杂度是 $\mathcal{O}(n)$.

12.3 同余

12.3.1 基本定义

定义 12.1 (同余)

$$a \equiv b \pmod{m} \iff a \bmod m = b \bmod m$$



在模意义下, 两个同余的数加、减、乘一个相同的数是可行的, 但是除法呢? 我们有

$$ad \equiv b \pmod{m} \iff a \equiv b \pmod{\frac{m}{\gcd(d, m)}}$$

乘法逆元: 在模 p 意义下, 定义数 a 的乘法逆元 a^{-1} 满足 $a \times a^{-1} \equiv 1 \pmod{p}$.

可以发现, 在模意义下除以 a 就等价于乘 a^{-1} , 前提是 a^{-1} 存在, 而由扩展欧几里得算法, 当 p 为质数时, a^{-1} 是一定存在的, 原因是我们可以将 $a \times a^{-1} \equiv 1 \pmod{p}$ 转换为 $a \times a^{-1} - 1 = k \times p$, 即 $a \times a^{-1} - k \times p = 1$, 解出 a^{-1} 即可.

模 n 的剩余系是指将模 n 同余的数看作本质相同的数得到的集合, 例如模 6 的剩余系包含 0, 1, 2, 3, 4, 5, 并且 $3 + 4 = 1, 3 \times 4 = 0$.

模 n 的缩系是指将 $[1, n]$ 中满足 $(m, n) = 1$ 的数 m 提出来构成的集合, 例如模 6 的缩系只包含 1, 5, 模 14 的缩系包含 1, 3, 5, 9, 11, 13, 缩系下的加法不封闭而乘法封闭.

12.3.2 费马小定理

定理 12.3 (费马小定理)

若 p 是素数, $1 \leq a < p$, 则 $a^{p-1} \equiv 1 \pmod{p}$



证明 我们知道 $A_i = i \times a, i < p$ 是 1 到 $p-1$ 的一个排列, 于是:

$$\prod_{i=1}^{p-1} A_i \equiv n^{p-1} (p-1)! \equiv (p-1)! \pmod{p}$$

由于 $(p-1)!$ 不被 p 整除, 所以两边同时消去, 证毕.

于是, 乘法逆元还可以通过费马小定理, 用快速幂求出, 即 $a^{-1} \equiv a^{p-2} \pmod{p}$.

12.4 Miller-Rabin 素性测试

我们在之前已经得到了一个 $\mathcal{O}(\sqrt{n})$ 的判断一个数是否是素数的算法, 但是这个复杂度实在是有点高, 对于 $n = 10^{18}$ 的大型数据无能为力, 这时候我们就可以用 **Miller-Rabin 素性测试**.

这个算法不是一个确定性算法，也就是说，它存在误判的可能（尽管这个可能非常小），但在 OI 范围内，我们可以将他改造为确定性算法，首先介绍一下传统的 Miller-Rabin 算法。

由费马小定理的逆否命题可知：

- 若存在 $1 \leq a < p$ 使得 $a^{p-1} \not\equiv 1 \pmod{p}$ ，则 p 不是素数。

于是，我们可以随机选取若干个 a 来进行测试，假如结果模 p 不为 1，那么它一定是合数，假如我们没有找到这样的 a ，那么 p 极大概率是一个素数。

为什么说是“极大概率”呢？因为的确是有合数能通过这样所有 a 的测试的，这样的数叫做卡迈克尔数 (Carmichael Number)，例如， $561 = 3 \times 11 \times 17$ 就是一个卡迈克尔数。不幸的是，在 10^9 内，这样的数就有足足 646 个。

我们尝试加强测试的强度，具体地，我们有二次探测定理：

定理 12.4 (二次探测定理)

若 p 为奇素数且 $x^2 \equiv 1 \pmod{p}$ ，则 x 的解为 $x \equiv \pm 1 \pmod{p}$ 。



所以我们可以这样设计我们的算法，对于每次随机出的 a ：

- 初始化 $d = p - 1$ 假如 $a^d \not\equiv 1 \pmod{p}$ ，返回 False。
- 不断将 d 变为 $\frac{d}{2}$ 直到 d 为奇数或 $d \not\equiv 1 \pmod{p}$
- 若最终 $a^d \not\equiv \pm 1 \pmod{p}$ ，则返回 False，否则通过本轮测试。

若随机选取 k 个底数，则算法的错误率为 4^{-k} ，也就是说，选取 20 到 40 个底数时，错误率是可以接受的。

在最开始我们说过，在 OI 范围内，可以将 Miller-Rabin 改造为一个确定性算法，具体来说，我们更改的是随机底数的部分，我们将随机的那些数改为确定的 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37（前 12 个质数），就能保证在 $n \leq 2^{78}$ 时，算法的正确性了。

在 OEIS 上，我们可以找到选取前 n 个质数能够判断的数值的范围¹。

```
1 inline bool Miller_Rabin(ll p) {
2     static const int Table[12] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
3     if(p < 3 || p % 2 == 0) return p == 2;
4     if(p < 40) {
5         for(int i = 0; i < 12; i++) if(Table[i] == p) return true;
6         return false;
7     }
8
9     ll tmp = p - 1; int b = 0;
10    while((tmp & 1) ^ 1) tmp >>= 1, b++;
11
12    for(int i = 0, j; i < 12; i++) {
13        ll x = ksm(Table[i], tmp, p);
14        if(x == 1) continue;
15        for(j = 0; j < b; j++) {
16            if(x == p - 1) break;
17            x = x * x % p;
18        }
19        if(j >= b) return false;
20    }
21    return true;
```

¹<https://oeis.org/A014233>

22 }

这里给出的代码通过将 $p-1$ 分解为 $2^b \times tmp$, 逆过来进行二次探测, 将复杂度去掉了一个 \log (快速幂的复杂度), 但整体思路是完全一样的.

12.5 Pollard's rho 算法

Pollard's rho 算法是在期望 $\mathcal{O}(n^{1/4})$ 的时间复杂度内求出合数 n 的一个非平凡因子的算法.

最开始的随机化算法

假如我们要求出 1000 的一个非平凡因子, 随机抽取一个数, 它是想要的那个数的概率显然是 $\frac{1}{1000}$, 这样实在有点小, 那就选两个数 i, j , 让 $|i-j|$ 是想要的那个数的概率是多少呢? 是大约 $\frac{1}{500}$, 因为选出第一个数的概率是 1, 让两数差的绝对值是某个给定值的概率就是 $2/n$, 因为 $i-j=x$ 和 $j-i=x$ 都是可行的, 再根据生日悖论, 我们可以生成一堆数, 每次用相邻两项作差, 和 n 求 gcd, 这就是 Pollard's rho 算法.

Pollard's rho 算法的特别之处在于它的随机函数是这样的: $f(x) = x^2 + c$, 其中 c 是一个给定的常数, 怎样生成我们的随机序列呢? 首先, x_1 随便取, 然后让 $x_i = f(x_{i-1})$ 即可. 不难发现这样生成的数列必然会有一个循环节, 如果将它们依次排开, 这些数会形成一个 ρ 形状, 那个圈就是循环的部分, ρ 的尾是 x_1 , 因此算法名中有一个形象的 “rho”.

现在的问题是怎样找到这个循环以结束本次的随机 (当选取的常数 c 不当时, 我们可能找不到这个合数的一个非平凡因子, 这时, 我们需要重新调整 c 的值, 再次进行 Pollard's rho 算法), 这里就要用到 Floyd 判圈算法: 它的原理是这样的, 两个人在一个圈上走, 让一个人以一倍速跑, 另一个人以二倍速跑, 一定存在一个时刻, 使得它们在环上的某一处相遇, 这时, 我们就结束算法.

```
1 inline int f(int x, int c, const int n) {
2     return (x * x + c) % n;
3 }
4
5 inline int Pollard_rho(int N) {
6     int c = rand() % (N - 1) + 1;
7     int t = f(0, c, N), r = f(f(0, c, N), c, N);
8     while(t != r) {
9         int d = __gcd(abs(t - r), N);
10        if(d > 1) return d;
11        t = f(t, c, N), r = f(f(r, c, N), c, N);
12    }
13    return N; //not found, re pollard_rho
14 }
```

基于路径的倍增和常数优化

频繁调用 $\mathcal{O}(\log n)$ 的 gcd 函数严重拖慢了算法的运行速度, 若 $\gcd(a, b) > 1$, 则 $\gcd(ac, b) > 1$, 等价于 $\gcd(ac \bmod b, b) > 1$, 因此, 我们可以先不断地用乘法, 最后达到一个阈值时, 进行一次总的 gcd, 一般情况下, 我们每次在这个 ρ 上取一段 $[2^{k-1}, 2^k]$ 的区间, 同时, 倍增的上界是 127.

```
1 inline ll Pollards_Rho(ll n) {
2     if(n == 4) return 2;
3     if(Miller_Rabin(n)) return n;
4
5     while(true) {
```

```

6   c = rand() % (n - 1) + 1;
7   ll t = 0, r = 0, p = 1, q, d;
8   for(int lim = 1; lim == 1 || t != r; lim = min(128, lim << 1)) {
9       for(int i = 0; i < lim; i++) {
10          t = f(t, n), r = f(f(r, n), n);
11          if(t == r || (q = p * abs(t - r) % n) == 0)
12              break;
13          p = q;
14      }
15      d = __gcd(p, n);
16      if(d > 1) return d;
17  }
18 }
19 }

```

12.6 数论函数与 Möbius 反演

看起来很高大上，实际上数论函数就是指定义域为正整数的函数。在研究具体的数论函数之前，我们先讨论一些抽象内容：

积性函数：若数论函数 f 满足 $f(1) = 1$ 且对任意 $a, b \in \mathbb{N}^*$, $(a, b) = 1$ 有 $f(ab) = f(a)f(b)$ ，称 f 积性。

完全积性函数：若数论函数 f 满足 $f(1) = 1$ 且对任意 $a, b \in \mathbb{N}^*$ 有 $f(ab) = f(a)f(b)$ ，称 f 完全积性。

我们稍后再讨论知道一个数论函数积性后有什么用以及有哪些有用的数论函数，还请读者先抱有一些疑问地做一些抽象的工作。

显然的，若 $x = \prod_i p_i^{\alpha_i}$ ，并且 f 为积性函数，那么 $f(x) = \prod_i f(p_i^{\alpha_i})$ ，这可以由 f 的定义不加推导地得出。

并且，若 f 和 g 均为积性函数，那么 $h(x) = f(x)g(x)$ 也是积性函数（定义可证）， $h(x) = f(x^p)$, $h(x) = f^p(x)$ 也为积性函数，甚至 $h(x) = \sum_{d|x} f(d)g(\frac{n}{d})$ 也是积性函数。

最后一个结论的简单证明：

设 $(a, b) = 1$ ，那么

$$\begin{aligned}
 h(a)h(b) &= \sum_{d_1|a} \sum_{d_2|b} f(d_1)f(d_2)g\left(\frac{a}{d_1}\right)g\left(\frac{b}{d_2}\right) \\
 &= \sum_{d|ab} f(d)g\left(\frac{ab}{d}\right) = h(ab)
 \end{aligned}$$

这个运算的形式非常特别，初次接触的时候或许会觉得不知所云，它的形式看起来像是卷积（ d 和 n/d 贡献到 n ，读者不知道什么是卷积也没关系，只需要知道是两个东西按照某种规则做运算即可），实际上这就是著名的**狄利克雷卷积**（Dirichlet convolution），定义为

$$(f * g)(n) = \sum_{d|n} f(d)g\left(\frac{n}{d}\right)$$

于是我们知道：两个积性函数的 Dirichlet 卷积也是积性的。显然，Dirichlet 满足很多常见的其他性质，例如交换律、结合律、分配律（对加减）即 $(f + g) * h = f * h + g * h$ 。

特别地，有 $f = g$ 的充要条件为 $f * h = g * h$ ，其中 h 为一个满足 $h(1) \neq 0$ 的数论函数。

证明：充分性显然，下证必要性：设 $r = (f - g) * h$ ，设 t 是最小的满足 $f(t) \neq g(t)$ 的数，那么

$$r(t) = \sum_{d|t} (f - g)(d)h\left(\frac{t}{d}\right) = (f - g)(t)h(1) \neq 0$$

因此在 t 处有 $(f * h)(t) \neq (g * h)(t)$, 必要性得证.

Dirichlet 卷积有单位元吗?

设 ϵ 是 Dirichlet 卷积的单位元, 即对 f 恒有 $f * \epsilon = f$, 我们来解出这个 ϵ :

$(f * \epsilon)(n) = \sum_{d \mid n} f(d) \epsilon(\frac{n}{d}) = f(n)$, 一个显然的想法是直接取 $\epsilon(n) = [n = 1]$, 根据上面的 $f = g$ 的充要条件, 设有两个单位元, 那么这两个单位元一定相等, 因此 $\epsilon(n) = [n = 1]$ 就是 Dirichlet 卷积的单位元.

有了单位元, 逆元还远吗? (划掉) 设 f^{-1} 是 f 的逆元当且仅当 $f * f^{-1} = \epsilon$, 由等式的性质, 若存在逆元则一定唯一 (若 $f(x) \neq 0$).

构造?

$$f^{-1}(n) = \frac{\epsilon(n) - \sum_{d \mid n, d \neq 1} f(d) g(\frac{n}{d})}{f(1)}$$

逆元的性质? $(f * g)^{-1} = f^{-1} * g^{-1}$, 积性函数的逆元也是积性函数.

来点有用的吧!

先说一个弱智 (强者) 函数 $1(n)$ 满足 $1(n) = 1$. 没错就是函数值恒为 1...

等下它好像没什么用啊? 找找逆元试试?

设有数论函数 μ 满足 $\mu * 1 = \epsilon$, 根据 “积性函数的逆元仍然是积性函数” 以及 1 显然是一个完全积性函数, 那么 μ 也应当是一个积性函数, 并且 $\mu(1) = 1$.

既然它是积性函数, 我们就只需要考虑它在质数的幂处的取值, 剩下的取值都可以唯一分解后得到.

对数 n , 有

$$\epsilon(n) = [n = 1] = \sum_{d \mid n} \mu(d) 1(\frac{n}{d}) = \sum_{d \mid n} \mu(d)$$

因此, $\sum_{d \mid n} \mu(d) = [n = 1]$, 这是一个重要结论.

我们已经有了 $\mu(1) = 1$, 对质数 p , 我们应当满足 $\mu(p) + \mu(1) = [p = 1] = 0$, 因此必然有 $\mu(p) = -1$, 同理, 对于 $p^k, k > 1$ 必然有 $\sum_{d \mid p^k} \mu(d) = 0$, 取 $k = 2$ 得到 $\mu(p^2) = 0$, 再取 $k = 3$ 得到 $\mu(p^3) = 0$, 一直取下去有 $\forall k \geq 2$, 有 $\mu(p^k) = 0$.

对于更一般的数 n , 将 n 唯一分解后根据 μ 的积性, 有

$$\mu(n) = \begin{cases} 1, & n = 1 \\ (-1)^k, & n = \prod_i p_i \\ 0, & \text{存在 } i \text{ 使得 } p_i^2 \mid n \end{cases}$$

很好! 我们将这个神秘函数 μ 解出来了, 并且能够通过分解质因数的时间复杂度将某个 $\mu(n)$ 算出来, 朴素为 $\mathcal{O}(\sqrt{n})$, 并且能够用线性筛 (欧拉筛) 在 $\mathcal{O}(n)$ 时间内求出 $\mu(1..n)$ (事实上, 对每个积性函数都可以用线性筛做), 代码如下:

```
1 inline void Sieve(int n) {
2     mu[1] = 1;
3     for(int i = 2; i <= n; i++) {
4         if(!d[i]) {d[i] = i, mu[i] = -1, primes.emplace_back(i); }
5         for(auto p : primes) {
6             if(1ll * p * i > n) break;
7             d[p * i] = p;
8             if(i % p == 0) {
9                 mu[p * i] = 0;
10                break;
11            } else mu[p * i] = mu[p] * mu[i];
```

12 }
 13 }
 14 }

魔术开始!

假如我们知道 $f = g * 1$, 两边卷 μ , 得到 $f * \mu = g * 1 * \mu$, 又因为 $1 * \mu = \varepsilon$, 因此

$$g = \mu * f$$

反过来同样成立, 即

$$f(n) = \sum_{d|n} g(d) \iff g(n) = \sum_{d|n} f(d) \mu\left(\frac{n}{d}\right)$$

上式称为**莫比乌斯反演** (Möbius Inversion), 下面以欧拉函数 φ 为例展示一下其威力.

欧拉函数 $\varphi(n)$ 表示 1 到 n 中与 n 满足 $(m, n) = 1$ 的数 m 的个数, 即 $\varphi(n) = \sum_{i=1}^n [(i, n) = 1]$.

结论: $\sum_{d|n} \varphi(d) = n$.

证明: 考虑将 $\frac{0}{n}, \frac{1}{n}, \frac{2}{n}, \dots, \frac{n-1}{n}$ 写成一排, 化简成最简分数 ($\frac{0}{n}$ 化为 $\frac{0}{1}$), 那么分母为 $d (d|n)$ 的分子有 $\varphi(d)$ 个, 一共有 n 个, 等式成立.

根据 Möbius 反演,

$$n = \sum_{d|n} \varphi(d) \iff \varphi(n) = \sum_{d|n} \mu(d) \frac{n}{d}$$

由于 μ 只在很少的位置取到非 0 值, 具体的, 设 $n = \prod_{i=1}^m p_i^{\alpha_i}$, 那么

$$\begin{aligned} \varphi(n) &= \sum_{d|n} \mu(d) \frac{n}{d} = n \sum_{d|n} \mu(d) \frac{1}{d} \\ &= n \prod_{p_{i_1} p_{i_2} \cdots p_{i_k} | n} (-1)^k \frac{1}{p_{i_1} p_{i_2} \cdots p_{i_k}} \\ &= n \prod_{i=1}^m \left(1 - \frac{1}{p_i}\right) \end{aligned}$$

最后一个等号并不是那么显然, 但是逆过去很容易证明其正确性, 同样的结论可以用容斥原理得到: $\varphi(n)$ 等于 n 减去能被 p_i 整除的, 再加上能被 $p_i p_j$ 整除的, 再减去能被 $p_i p_j p_k$ 整除的, 一直做下去, 求和即证.

实际上, 上式中的 n 也是一个特殊数论函数, 称为 id , 满足 $id(n) = n$, 并且有幂函数 $id^k(n) = n^k$.

无论通过 Dirichlet 卷积的性质还是 φ 的解析式, 都可以得到同一个结论: φ 是积性函数.

φ 在质数的幂上有特殊的取值, 具体地有 $\varphi(p^k) = p^k - p^{k-1}$, 当 $k = 1$ 时有 $\varphi(p) = p - 1$.

φ 被称为欧拉函数是因为欧拉是研究其性质的第一人, 具体地有**欧拉定理**²: 若 $(n, m) = 1$, 则 $n^{\varphi(m)} \equiv 1 \pmod{m}$.

注意到当 m 是质数的时候, 欧拉定理就给出了费马小定理 $n^{p-1} \equiv 1 \pmod{p}$.

证明: 考虑将模 m 的缩系抽出来记做 $S_{1.. \varphi(m)}$, 和费马小定理的证明类似, nS_i 构成了 S 的一个排列, 用 $\prod S_i$ 去除, 命题得证.

实际上, 假如读者会一些群论, 由群论中的 Lagrange 定理, 数论中的欧拉定理是显然的.

当 $(n, m) \neq 1$ 的时候, 我们有扩展欧拉定理:

² “如果 N 与 x 互素, n 是与 N 互素且不超过它的数的个数, 那么 $x^n - 1$ 总能被 N 整除.” ——欧拉

$$n^k \equiv \begin{cases} n^{k \bmod \varphi(m)}, & (n, m) = 1 \\ n^k, & (n, m) \neq 1, k \leq \varphi(m) \\ n^{(k \bmod \varphi(m)) + \varphi(m)}, & k > \varphi(m) \end{cases} \pmod{m}$$

证明略去，编者没学过。

Luogu P1390 公约数的和

给定 n ，求

$$\sum_{i=1}^n \sum_{j=i+1}^n (i, j)$$

对于 100% 的数据满足 $2 \leq n \leq 2 \times 10^6$ 。

Solution: Luogu P1390 公约数的和

纯板子，只需要算 $\sum_{i=1}^n \sum_{j=1}^n (i, j)$ 再减去 $\sum_{i=1}^n i$ 后除以二即可，后面的式子是容易算的，考虑前一个式子，枚举 $d = (i, j)$ 有：

$$\sum_{i=1}^n \sum_{j=1}^n (i, j) = \sum_{d=1}^n d \sum_{i=1}^{n/d} \sum_{j=1}^{n/d} [(i, j) = 1]$$

注意到 $[(i, j) = 1] = \sum_{d|(i, j)} \mu(d)$ ，带进去，

$$\sum_{d=1}^n d \sum_{i=1}^{n/d} \sum_{j=1}^{n/d} [(i, j) = 1] = \sum_{d=1}^n d \sum_{i=1}^{n/d} \sum_{j=1}^{n/d} \sum_{k|(i, j)} \mu(k)$$

交换求和顺序，

$$\sum_{d=1}^n d \sum_{i=1}^{n/d} \sum_{j=1}^{n/d} \sum_{k|(i, j)} \mu(k) = \sum_{d=1}^n d \sum_{D=1}^{n/d} \mu(D) \left\lfloor \frac{n}{dD} \right\rfloor^2$$

解释一下就是先枚举 (i, j) 的因子 D ，有多少对 (i, j) 满足 $1 \leq i, j \leq \lfloor \frac{n}{d} \rfloor$ 并且 $D | (i, j)$ 呢？这只需要 i, j 都是 D 的倍数而 $[1, x]$ 中 D 的倍数有 $\lfloor \frac{x}{D} \rfloor$ 个。

线性筛出来 μ ，暴力算上面的式子就是 $\mathcal{O}(n \log n)$ 的。

有没有更给力的复杂度？考虑另一个等式

$$n = \sum_{d|n} \varphi(d)$$

用它做反演称之为**欧拉反演**，具体地，套到最开始的式子有

$$\sum_{i=1}^n \sum_{j=1}^n (i, j) = \sum_{i=1}^n \sum_{j=1}^n \sum_{d|(i, j)} \varphi(d)$$

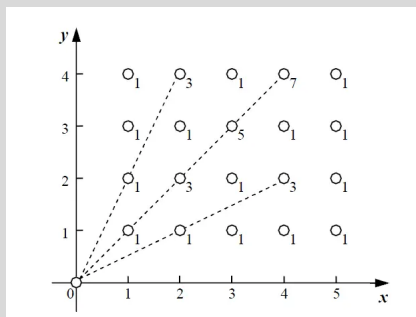
交换求和顺序：

$$\sum_{i=1}^n \sum_{j=1}^n \sum_{d|(i, j)} \varphi(d) = \sum_{d=1}^n \varphi(d) \left\lfloor \frac{n}{d} \right\rfloor^2$$

线性筛出来 φ 后暴力算是 $\mathcal{O}(n)$ ，注意到可以整除分块，于是预处理 φ 的前缀和后整除分块可以做到 $\mathcal{O}(n + \sqrt{n})$

【NOI2010】能量采集

$n \times m$ 平面直角坐标系上, x 为 1 到 n , y 为 1 到 m , 对每个点 (i, j) , 其值为 $(0, 0)$ 到 (i, j) 这条线段上点数的二倍加一 (不包含左右端点), 求所有点的值之和。 $n = 5, m = 4$ 的情况如下图所示, 每个点旁边的数字即其值, 答案为 36。



对于 100% 的数据, 保证 $1 \leq n, m \leq 10^5$ 。

Solution: 【NOI2010】能量采集

$(0, 0)$ 到 (i, j) 这条线段上 (不包含 $(0, 0)$ 但包含 (i, j)) 有多少个点? 实际上就是有多少个正整数 d 满足 $d \mid i$ 并且 $d \mid j$, 这样, $(\frac{i}{d}, \frac{j}{d})$ 和 (i, j) 与 $(0, 0)$ 连线就一致了, 因此点 (i, j) 的值实际上就是 $2 \gcd(i, j) - 1$ 。

将 (i, j) 的值变为 $2 \gcd(i, j)$, 最终答案再减去 nm 即可, 也就是说我们要计算

$$2 \sum_{i=1}^n \sum_{j=1}^m \gcd(i, j)$$

和上一道题一模一样啊有没有! 式子的 $\frac{1}{2}$ 就等于

$$\sum_{D=1}^{\min(n, m)} \mu(D) \sum_{d=1}^{\min(n, m)/D} d \lfloor \frac{n}{Dd} \rfloor \lfloor \frac{m}{Dd} \rfloor$$

筛完 μ 暴力算, 时间复杂度 $\mathcal{O}(n \log n)$ 。

实际上同样可以用欧拉反演得到

$$\sum_{d=1}^{\min(n, m)} \varphi(d) \lfloor \frac{n}{d} \rfloor \lfloor \frac{m}{d} \rfloor$$

预处理 φ 前缀和, 对 n, m 同时整除分块, 时间复杂度 $\mathcal{O}(n + \sqrt{n})$ 。

12.7 容斥原理与二项式反演

12.7.1 容斥原理

容斥原理的基本形式是这样的:

$$\left| \bigcup_{i=1}^n S_i \right| = \sum_{P \subseteq S, P \neq \emptyset} (-1)^{|P|-1} \left| \bigcap_{i=1}^{|P|} P_i \right|$$

其中, S 为集合 S_i 构成的集族。

证明 考虑每个元素 e 产生的贡献, 假设它在集合 T_1, T_2, \dots, T_m 中出现, 那么它在右式中的贡献为

$$\sum_{i=1}^m (-1)^{i-1} \binom{m}{i} = \binom{m}{0} - \sum_{i=0}^m (-1)^i \binom{m}{i} = 1$$

最后一个等号逆用了二项式定理.

为什么是这个形式？

考虑简单的三集合形式，容斥原理给出了

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |A \cap C| + |A \cap B \cap C|$$

当我们在右侧写下 $|A| + |B| + |C|$ 的时候，某两个集合重叠的部分会被多算一遍，多算了就再减掉就好，于是接着写下 $-|A \cap B| - |B \cap C| - |A \cap C|$ ，结束了吗？好像还没有，此时三个集合重叠的部分又被少算了一次（证明在下方），再加回来 $+|A \cap B \cap C|$ 就不重不漏了。

更抽象地，假如我们写好了 $\leq k$ 个集合重叠的部分，那么恰好 $k+1$ 个集合重叠的部分会被算多少次呢？

$$\sum_{i=1}^k (-1)^{i-1} \binom{k+1}{i} = \sum_{i=0}^{k+1} (-1)^{i-1} \binom{k+1}{i} + 1 - (-1)^k = 1 - (-1)^k$$

我们应该让被算的次数等于 1！于是我们应该再算上 $(-1)^k$ 次，即让答案加上

$$\sum_{P \subseteq S, |P|=k+1} (-1)^k \left| \bigcap_{i=1}^{|P|} P_i \right|$$

容斥定理的形式就被我们很好地解释了。

对于并集的形式，可以用补集转化的方法，设 U 为全集，于是有

$$\left| \bigcap_{i=1}^n S_i \right| = |U| - \left| \bigcup_{i=1}^n \overline{S_i} \right|$$

我们可以将容斥原理一般化，对于两个关于集合的函数 $f(S), g(S)$ ，有

$$f(S) = \sum_{T \subseteq S} g(T) \iff g(S) = \sum_{T \subseteq S} (-1)^{|S|-|T|} f(T)$$

证明可以直接将左式代入右式或将右式代入左式。

这个式子同样有补集形式：

$$f(S) = \sum_{S \subseteq T} g(T) \iff g(S) = \sum_{S \subseteq T} (-1)^{|T|-|S|} f(T)$$

12.7.2 二项式反演

考虑之前容斥原理的基本形式，用补集转换不难得到：

$$|\overline{S_1} \cap \overline{S_2} \cap \cdots \cap \overline{S_n}| = |S| - \sum_{1 \leq i \leq n} |S_i| + \sum_{1 \leq i < j \leq n} |S_i \cap S_j| + \cdots + (-1)^n |S_1 \cap S_2 \cap \cdots \cap S_n|$$

注意到补集的补集为原集，于是又有

$$|S_1 \cap S_2 \cap \cdots \cap S_n| = |S| - \sum_{1 \leq i \leq n} |\overline{S_i}| + \sum_{1 \leq i < j \leq n} |\overline{S_i} \cap \overline{S_j}| + \cdots + (-1)^n |\overline{S_1} \cap \overline{S_2} \cap \cdots \cap \overline{S_n}|$$

考虑一种特殊情况：多个集合的交集大小只和集合的数目有关，令 $f(n)$ 表示 n 个补集的交集大小， $g(n)$ 表示 n 个原集的交集大小，于是有**二项式反演形式一**：

$$f(n) = \sum_{i=0}^n (-1)^i \binom{n}{i} g(i) \iff g(n) = \sum_{i=0}^n (-1)^i \binom{n}{i} f(i)$$

这其实是一种特殊情况，更一般的形式是这样的：

$$f(n) = \sum_{i=m}^n \binom{n}{i} g(i) \iff g(n) = \sum_{i=m}^n (-1)^{n-i} \binom{n}{i} f(i)$$

证明是套路地左代到右或右代到左。

最常用的是另一种，二项式反演形式二：

$$f(n) = \sum_{i=n}^m \binom{i}{n} g(i) \iff g(n) = \sum_{i=n}^m (-1)^{i-n} \binom{i}{n} f(i)$$

组合意义： $f(n)$ 表示“钦定选 n 个”的方案数， $g(n)$ 表示“恰好选 n 个”的方案数。

例题 1：BZOJ2839 集合计数

一个 n 元素集合有 2^n 个子集，现在要求选出至少一个子集，使它们交集的元素数量恰好为 k ，求取法方案数模 $10^9 + 7$ 。 $n \leq 10^6, 0 \leq k \leq n$

Solution: 令 $f(m)$ 表示钦定交集元素数量至少为 m 的方案数，显然有 $f(m) = \binom{n}{m} (2^{2^{n-m}} - 1)$ ，接下来就很容易了，令 $g(m)$ 为交集元素数量恰好为 m 的方案数，有 $f(m) = \sum_{i=m}^n \binom{i}{m} g(i)$ ，由二项式反演得到 $g(m) = \sum_{i=m}^n (-1)^{i-m} \binom{i}{m} f(i)$ ，通过一些预处理，我们可以 $\mathcal{O}(n)$ 求出 $g(k)$ ，即最终答案。

第 13 章 组合变换技巧

13.1 组合恒等式

定义 13.1

$\binom{n}{m}$ 表示从 n 个不同的数中选取 m 个数的方案数，两个方案不同当且仅当两种方案中的数组成的集合不同.

如何计算呢？我们先考虑排列数：

定义 13.2

定义从 n 个不同的数中选出 m 个数的方案数，两个方案 A, B 不同当且仅当 $\exists i \in [1, m], A_i \neq B_i$

$$A_n^m = \frac{n!}{(n-m)!} = n^{\overline{m}} \quad (13.1)$$

这是怎样得出的呢？考虑往 m 个空位依次填数，第一个空位有 n 种填法，第二个空位有 $n-1$ 种填法，依此类推，就得到了计算排列的公式，从而，我们有第一个组合公式，也就是计算公式：

$$\binom{n}{m} = \frac{n!}{m!(n-m)!} = \frac{n^{\overline{m}}}{m!} \quad (13.2)$$

注意到下降幂的定义域是在 \mathbb{R} 的，于是我们可以将上标也即公式 13.2 中的 n 的定义域替换为 \mathbb{R} 。

显然 m 应该是一个整数，那么当 $m < 0$ 时，我们如何定义呢？《具体数学》中给出了答案：

$$\binom{r}{k} = \begin{cases} \frac{r^{\overline{k}}}{k!} & \text{integer } k \geq 0 \\ 0 & \text{integer } k < 0 \end{cases} \quad (13.3)$$

接下来就到了重头戏：组合恒等式，这些恒等式的证明并不困难，这里不再占据篇幅来证明。

13.1.1 基本恒等式

$$\binom{n}{k} = \binom{n}{n-k} \quad (13.4)$$

$$\binom{r}{k} = \frac{r}{k} \binom{r-1}{k-1}, \quad \text{integer } k \neq 0 \quad (13.5)$$

公式 13.5 中 $k \neq 0$ 的限制十分烦人，于是我们考虑把 k 移项到左边，得到：

$$k \binom{r}{k} = r \binom{r-1}{k-1} \quad (13.6)$$

在这同时，我们得到了另两个恒等式：

$$(r-k) \binom{r}{k} = r \binom{r-1}{k}, \quad \text{integer } k \quad (13.7)$$

$$\binom{r}{k} = \binom{r-1}{k} + \binom{r-1}{k-1}, \quad \text{integer } k \quad (13.8)$$

式 13.8 是我们递推求组合数的常用公式，请读者务必牢记。

上面的 8 个式子是所有组合恒等式中的基石，掌握了它们，我们就可以进行下一步了，不必被繁杂的公式所困扰，而应更多地注意它们背后的组合意义。

以式 13.8 为例，假如我们钦定 r 为整数，我们可以将它看作：从 r 中选出 k 个数，总共的方案数等于从 $r-1$ 个数中选出 k 个的方案数（不选第一个数）与从 $r-1$ 个数中选出 $k-1$ 个数的方案数（选择第一个数）的和，这样来看，这个公式的正确性就十分明朗了，实际上，这也是我们证明一些组合恒等式的惯用套路：**给它一个实际意义**。

我们看看是否可以从这 8 个恒等式中推出一些其他优美的等式出来。

13.1.2 求和一类组合恒等式

先用式 13.8：

$$\begin{aligned} \binom{5}{3} &= \binom{4}{3} + \binom{4}{2} \\ &= \binom{4}{3} + \binom{3}{2} + \binom{3}{1} \\ &= \binom{4}{3} + \binom{3}{2} + \binom{2}{1} + \binom{2}{0} \\ &= \binom{4}{3} + \binom{3}{2} + \binom{2}{1} + \binom{1}{0} + \binom{1}{-1} \end{aligned}$$

而由于 $\binom{1}{-1} = 0$ ，于是我们不必再展开下去。

从上述过程中，我们可以提取出一个重要的恒等式：

$$\begin{aligned} \sum_{k \leq n} \binom{r+k}{k} &= \binom{r}{0} + \binom{r+1}{1} + \binom{r+2}{2} + \cdots + \binom{r+n}{n} \\ &= \binom{r+n+1}{n}, \quad \text{integer } n \end{aligned} \quad (13.9)$$

类似地，我们可以得到以下恒等式：

$$\begin{aligned} \sum_{0 \leq k \leq n} \binom{k}{m} &= \binom{0}{m} + \binom{1}{m} + \binom{2}{m} + \cdots + \binom{n}{m} \\ &= \binom{n+1}{m+1}, \quad \text{integers } n, m \geq 0 \end{aligned} \quad (13.10)$$

如果我们把组合数和差分结合起来，会有什么神奇的性质呢？

$$\Delta \left(\binom{x}{m} \right) = \binom{x+1}{m} - \binom{x}{m} = \binom{x}{m-1}$$

那么由有限微积分的相关知识，我们得到了：

$$\sum \binom{x}{m} \delta x = \binom{x}{m+1} + C$$

那么式 13.10 就是显而易见的了。

13.1.3 二项式定理

组合数的一个重要应用是**二项式定理**：

定理 13.1 (二项式定理)

$$(x+y)^r = \sum_k \binom{r}{k} x^k y^{r-k}, \quad \text{integer } r \geq 0 \text{ or } |x/y| < 1$$



由二项式定理，我们可以推出很多重要等式，下面我们一一介绍。

$$2^n = \binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{n}, \quad \text{integer } n \geq 0$$

证明 在二项式定理中直接令 $x=y=1$ 即可。

$$0 = \binom{n}{0} - \binom{n}{1} + \cdots + (-1)^n \binom{n}{n}, \quad \text{integer } n \geq 0$$

$$(1+z)^r = \sum_k \binom{r}{k} z^k, \quad |z| < 1 \quad (13.11)$$

当 r 是非负整数时，二项式定理的正确性是容易得到的，而当 r 是任意复数的时候，我们可以用泰勒展开：

$$\begin{aligned} f(z) &= \frac{f(0)}{0!} z^0 + \frac{f'(0)}{1!} z^1 + \frac{f''(0)}{2!} z^2 + \cdots \\ &= \sum_{k \geq 0} \frac{f^{(k)}(0)}{k!} z^k \end{aligned}$$

令 $f(z) = (1+z)^r$ ，我们有 $f^{(k)}(z) = r^{\underline{k}}(1+z)^{r-k}$ 。令 $z=0$ ，我们就得到了式 13.11。

13.1.4 负数上标下的二项式系数

现在，让我们来看看当组合数的上标为负数时，组合数的取值。

由式 13.8，我们有 $\binom{0}{0} = \binom{-1}{0} + \binom{-1}{-1}$ ，而 $\binom{-1}{-1} = 0$ ，于是我们必须让 $\binom{-1}{0} = 1$ ，类似地，我们可以得到下表：

n	$\binom{n}{0}$	$\binom{n}{1}$	$\binom{n}{2}$	$\binom{n}{3}$	$\binom{n}{4}$	$\binom{n}{5}$	$\binom{n}{6}$	$\binom{n}{7}$	$\binom{n}{8}$
-4	1	-4	10	-20	35	-56	84	-120	165
-3	1	-3	6	-10	15	-21	28	-36	45
-2	1	-2	3	-4	5	-6	7	-8	9
-1	1	-1	1	-1	1	-1	1	-1	1
0	1	0	0	-0	0	0	0	0	0

事实上，我们可以得到以下等式：

$$\binom{r}{k} = (-1)^k \binom{k-r-1}{k}, \quad \text{integer } k \quad (13.12)$$

证明并不困难，因为我们知道：

$$\begin{aligned} r^{\underline{k}} &= r(r-1) \cdots (r-k+1) \\ &= (-1)^k (-r)(-r-1) \cdots (-r-k+1) \\ &= (-1)^k (k-r-1)^{\underline{k}} \end{aligned}$$

式 13.12 的一些变形是非常优美且有用的，例如：

$$(-1)^m \binom{-n-1}{m} = (-1)^n \binom{-m-1}{n} \quad (13.13)$$

原因是等式两边都等于 $\binom{n+m}{n}$.

$$\begin{aligned}\sum_{k \leq m} \binom{r}{k} (-1)^k &= \sum_{k \leq m} \binom{k-r-1}{k} \\ &= \binom{-r+m}{m} \\ &= (-1)^m \binom{r-1}{m}\end{aligned}\quad (13.14)$$

再介绍一个恒等式:

$$\sum_{k \leq m} \binom{m+r}{k} x^k y^{m-k} = \sum_{k \leq m} \binom{-r}{k} (-x)^k (x+y)^{m-k} \quad (13.15)$$

证明 首先, 当 $m < 0$ 时, 等式两侧都等于 0, 当 $m = 0$ 时, 等式两侧都等于 1.

接下来, 我们令 S_m 为等式左边, 可以得到:

$$S_m = \sum_{k \leq m} \binom{m-1+r}{k} x^k y^{m-k} + \sum_{k \leq m} \binom{m-r+r}{k-1} x^k y^{m-k}$$

而我们已经知道

$$\sum_{k \leq m} \binom{m-1+r}{k} x^k y^{m-k} = y S_{m-1} + \binom{m-1+r}{m} x^m \sum_{k \leq m} \binom{m-1+r}{k-1} x^k y^{m-k} = x S_{m-1}$$

于是, 当 $m > 0$ 时, 有:

$$S_m = (x+y) S_{m-1} + \binom{-r}{m} (-x)^m$$

而对于等式右侧, 显然有相同的结论, 证毕.

将某些 x, y 的取值代入式 13.15, 可以得到一些结论:

当 $x = -1, y = 1$ 时,

$$\sum_{k \leq m} \binom{m+r}{k} (-1)^k = \binom{-r}{m}, \quad \text{integer } m \geq 0$$

当 $x = y = 1, r = m+1$ 时:

$$\sum_{k \leq m} \binom{2m+1}{k} = \sum_{k \leq m} \binom{m+k}{k} 2^{m-k}$$

注意到等式左边实际上是杨辉三角中第 $2m+2$ 行的前 $m+1$ 项和, 而由二项式定理, 该行的和为 2^{2m+1} , 又因为杨辉三角的对称性, 我们得到等式左边实际上就是 $\frac{1}{2} \times 2^{2m+1} = 2^{2m}$, 于是:

$$\sum_{k \leq m} \binom{m+k}{k} 2^{-k} = 2^m, \quad \text{integer } m \geq 0 \quad (13.16)$$

13.1.5 多系数组合恒等式

紧接着, 我们介绍另一类重要的组合恒等式, 其中含有多于两个二项式系数, 更加复杂.

$$\binom{r}{m} \binom{m}{k} = \binom{r}{k} \binom{r-k}{m-k}, \quad \text{integer } m, k \quad (13.17)$$

当 $k = 1$ 时, 式 13.17 对于我们就十分熟悉了, 它就是我们所熟知的吸收律.

上面公式的一个应用就是将 m 从两个组合数中均存在变为一个含 m , 一个不含 m 的组合数.

证明可以直接套用公式 13.2.

类似的恒等式在这里一并给出:

$$\sum_k \binom{r}{m+k} \binom{s}{n-k} = \binom{r+s}{m+n}, \quad \text{integers } m, n \quad (13.18)$$

$$\sum_k \binom{l}{m+k} \binom{s}{n+k} = \binom{l+s}{l-m+n}, \quad \text{integers } m, n, \text{ integer } l \geq 0 \quad (13.19)$$

$$\sum_k \binom{l}{m+k} \binom{s+k}{n} (-1)^k = (-1)^{l+m} \binom{s-m}{n-1}, \quad \text{integers } m, n, \text{ integer } l \geq 0 \quad (13.20)$$

$$\sum_{k \leq l} \binom{l-k}{m} \binom{s}{k-n} (-1)^k = (-1)^{l+m} \binom{s-m-1}{l-m-n}, \quad \text{integers } m, n, l \geq 0 \quad (13.21)$$

$$\sum_{0 \leq k \leq l} \binom{l-k}{m} \binom{q+k}{n} = \binom{l+q+1}{m+n+1} \quad \text{integers } l, m \geq 0, \quad \text{integers } n \geq q \geq 0 \quad (13.22)$$

13.2 组合技巧与常用结论

Catalan 数列满足这样的性质:

$$H_0 = H_1 = 1$$

$$H_n = \sum_{i=0}^{n-1} H_i H_{n-i-1}, \quad n \geq 2$$

它有很多组合意义, 其中比较著名的三个分别为:

- H_n 即为将 $1, 2, \dots, n$ 按顺序出、入栈的方案数, 例如负号表示出栈, $1, 2, 3$ 的一个出入栈顺序为 $1, 2, -2, -1, 3, -3$.
- H_n 为在二维笛卡尔坐标系中规定每次只能向右或向上走 1, 从 $(0, 0)$ 走到 (n, n) 且不能穿过直线 $y = x$ (一个点接触到是可以的) 的方案数, 例如 $(0, 0) \rightarrow (0, 1) \rightarrow (1, 1)$.
- H_n 为长度为 $2n$ 的合法括号序列的数量.

对于第一个、第三个意义, 递推式可以由枚举 1 的出栈位置或第一个括号的匹配位置得到, 第二个意义的递推则是枚举除终点外走到的最后一个 x 和 y 相等的点的位置. 接下来我们先用即将向读者讲解的生成函数求出 H_n 的封闭形式, 再从意义二 (在笛卡尔坐标系中的解释) 用一个更为精巧的组合意义得到 Catalan 数的封闭形式.

生成函数的解法很暴力, 设 $H(z)$ 为 H 的生成函数, 那么有

$$\begin{aligned} H(z) &= \sum_{n \geq 0} H_n z^n = 1 + \sum_{n \geq 1} H_n z^n = 1 + \sum_{n \geq 1} \sum_{i=0}^{n-1} H_i H_{n-i-1} z^n \\ &= 1 + \sum_{n \geq 0} \sum_{i=0}^n H_i H_{n-i} z^{n+1} = 1 + z \sum_{n \geq 0} \sum_{i=0}^n H_i H_{n-i} z^n \\ &= 1 + z \sum_{n \geq 0} \sum_{i=0}^n (H_i z^i) (H_{n-i} z^{n-i}) = 1 + z H(z)^2 \end{aligned}$$

解得 $H(z) = \frac{1 \pm \sqrt{1-4z}}{2z}$.

这里出现了问题, 分子上出现了两种情况, 到底应该选哪一个呢? $H(0)$ 应该等于 $H_0 = 1$, 将 $z = 0$ 代到上面的式子中, 我们希望得到一个 $\frac{0}{0}$ 或 1 (因为 $H(0) = \frac{0}{0}$ 本质上相当于 $0H(0) = 0$), 发现应该取 $H(z) = \frac{1 - \sqrt{1-4z}}{2z}$.

这个生成函数的封闭形式和传统的线性递推不同, 因此考虑将 $\sqrt{1-4z}$ 用二项式定理展开:

$$(1-4z)^{1/2} = \sum_{i \geq 0} \binom{\frac{1}{2}}{i} (-4z)^i = 1 + \sum_{i \geq 1} \frac{(1/2)_i}{i!} (-4z)^i$$

先考虑 $\frac{1}{2}$ 的 i 阶下降幂, 它等于 $(\frac{1}{2})(\frac{-1}{2})(\frac{-3}{2}) \cdots (\frac{3-2i}{2})$. 分母乘起来, 分子用双阶乘表示, 上式等于 $\frac{(-1)^{i-1} (2i-3)!!}{2^i}$.

带回到原式, 有 $(1-4z)^{1/2}$ 等于

$$1 + \sum_{i \geq 1} \frac{(-1)^{i-1}(2i-3)!!}{i!2^i} (-4z)^i$$

用双阶乘的化简技巧, 即 $(2n-1)!! = \frac{(2n)!}{(2n)!!} = \frac{(2n)!}{(n)!2^n}$, 先用前一个等号, 将上式化为

$$1 + \sum_{i \geq 1} \frac{(-1)^{i-1}(2i-2)!}{i!2^i(2i-2)!!} (-4z)^i$$

再用 $(2n)!! = 2^n n!$, 再化为

$$1 - \sum_{i \geq 1} \frac{(2i-2)!}{i!(i-1)!} 2z^i$$

这就是 $\sqrt{1-4z}$, 将这个展开式带回到 $H(z)$, 得 $H(z) =$

$$\frac{\sum_{i \geq 1} \frac{(2i-2)!}{i!(i-1)!} 2z^i}{2z} = \sum_{i \geq 1} \frac{(2i-2)!}{i!(i-1)!} z^{i-1} = \sum_{i \geq 0} \frac{(2i)!}{(i+1)!i!} z^i$$

于是 $H_n = [z^n]H(z) = \frac{(2n)!}{n!(n+1)!} = \frac{1}{n+1} \binom{2n}{n}$.

考虑 Catalan 数在笛卡尔坐标系下的意义, 从 $(0,0)$ 到 (n,n) 的非降路径个数为 $\binom{2n}{n}$, 这是因为总共要向右 n 步, 向上 n 步, 在这 $2n$ 步里选出 n 个位置向上走即可.

怎样算出穿过 $y=x$ 的非降路径的方案数呢? 若路径穿过了直线 $y=x$, 那么它一定和 $y=x+1$ 这条直线有交点, 考虑最后一个这样的交点, 设为 $(p,p+1)$, 将路径在 $(0,0)$ 到 $(p,p+1)$ 之间的部分沿着直线 $y=x+1$ 翻折, 翻折后的路径和这之后没有被翻折的路径组成了一个从 $(-1,1)$ 到 (n,n) 的非降路径, 不难得出每一个 $(-1,1)$ 到 (n,n) 的非降路径和每一个从 $(0,0)$ 到 (n,n) 且经过了直线 $y=x+1$ 的非降路径可以构成双射, 因此后者的数量等于前者的数量等于 $\binom{2n}{n-1}$, 于是我们要求的方案数即为 $\binom{2n}{n} - \binom{2n}{n-1}$, 化简得 $\frac{1}{n+1} \binom{2n}{n}$.

Catalan 数还有一个用处并不是很大的递推公式:

$$H_n = \frac{4n-2}{n+1} H_{n-1}$$

当我们将问题转化为网格图上固定起点和终点, 不能经过某一条直线时的方案数, 我们就可以用类似的方法翻折后构造双射计算, 但是还有一种类似的问题不能直接这样算:

在平面直角坐标系上, 给定两条直线 $y=x+a$ 和 $y=x-b$, 其中 $a, b > 0$, 求从 $(0,0)$ 出发, 每次向右或向上走 1, 不触碰任一直线到达 (n,m) 的方案数.

假如我们用 Catalan 类似的容斥, 会发现翻折完之后是没办法处理的, 这时候怎么办呢? 其实思路是类似的. 考虑不管不能触碰直线的限制, 方案数有 $\binom{n+m}{n}$, 最终答案应该再减去碰到了某条直线的方案.

将 $y=x+a$ 看作直线 A , $y=x-b$ 看作直线 B , 那么所有的方案都可以写成 $AABABBA..$ 的形式 (每碰到一次直线 A 就写下一个 A), 将连续多次碰到同一条直线缩起来, 形式就是 $ABABA..$ 或 $BABAB...$

于是我们应该减去的就是: 第一个碰到的直线是 A 的方案数加上第一个碰到的直线是 B 的方案数.

考虑算前者, 也就是说我们要减去 $A \ AB \ ABA \ ABAB \ ABABA..$ 的方案数.

自然想到翻折, 将 (n,m) 沿着 A 翻折得到 $(m-a, n-a)$, 从 $(0,0)$ 到 $(m-a, n-a)$ 的路径个数代表什么? 假如将所有路径最后一个与 A 的交点到终点处的路径翻折, 我们就和上面的翻折终点的路径一一对应了, 但是实际上最后一次经过 A 后还可能经过 B , 于是 $(0,0)$ 到 $(m-a, n-a)$ 的方案数实际上就是 $..A \ ..AB$ 的方案数和.

将 $(m-a, n-a)$ 再沿着 B 翻一遍求路径个数, 算出来的就是以 $BA \ BAB$ 为结尾的方案数, 类似的再沿着 A 折, 算出来以 $ABA \ ABAB$ 为结尾的方案数, 直到方案数为 0.

以 A 为第一个被碰到的直线的路径数就是以 $A \ AB$ 结尾减去以 $BA \ BAB$ 结尾加上 $ABA \ ABAB$ 结尾... 的和, 不断递归下去求, 当终点不在第一象限时停止, 递归深度为 $\mathcal{O}(n+m)$.

以 B 开头的方案数类似求，不同之处只在于第一次将 (n, m) 沿 B 对称而不是 A .
总时间复杂度 $\mathcal{O}(n + m)$.

13.3 Stirling 数

第一类 Stirling 数 $\begin{bmatrix} n \\ k \end{bmatrix}$ 指的是将 n 个元素排成 k 个轮换的方案数，例如 $[2, 3, 1]$ 和 $[3, 1, 2], [1, 2, 3]$ 本质相同，有 $\begin{bmatrix} 0 \\ 0 \end{bmatrix} = 1$ 以及递推式

$$\begin{bmatrix} n \\ k \end{bmatrix} = \begin{bmatrix} n-1 \\ k-1 \end{bmatrix} + (n-1) \begin{bmatrix} n-1 \\ k \end{bmatrix}$$

考虑组合意义的证明，第 n 个元素单独成一个轮换的方案数加上并入原本轮换的方案数即得上式.

第二类 Stirling 数 $\begin{Bmatrix} n \\ k \end{Bmatrix}$ 指的是将 n 个物品分成 k 个非空集合的方案数，有 $\begin{Bmatrix} 0 \\ 0 \end{Bmatrix} = 1$ 以及递推式

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix} + k \begin{Bmatrix} n-1 \\ k \end{Bmatrix}$$

组合意义的证明：考虑最后一个物品，它单独放在一个集合或者并入本来就有的集合中，两者相加即得.
说清楚了定义，接下来我们来研究 Stirling 数的一些性质，以及它可以用来解决什么样的问题.

对于一个排列 p ，将 i 与 p_i 连边可以发现， p 可以写成若干个环的形式，也就是说对于任意一个排列，它都与若干个轮换等价，这引出第一个变换：

$$\sum_{k=0}^n \begin{bmatrix} n \\ k \end{bmatrix} = n!$$

右侧指的是长度为 n 的排列数量，实际上我们也可以归纳证明.

还记得有限微积分吗？我们总是要将指数换为下降幂的形式，例如

$$x^4 = x^{\underline{4}} + 6x^{\underline{3}} + 7x^{\underline{2}} + x^{\underline{1}}$$

为什么要在这里提这个问题呢？因为普通幂和下降幂相互转化的系数正是 Stirling 数！具体的有

$$x^n = \sum_{k=0}^n \begin{Bmatrix} n \\ k \end{Bmatrix} x^{\underline{k}}, \quad n \geq 0$$

证明可以做归纳：

$$x \cdot x^n = \sum_{k=0}^n \begin{Bmatrix} n \\ k \end{Bmatrix} x \cdot x^{\underline{k}}$$

由于 $x \cdot x^{\underline{k}} = x^{\underline{k+1}} + kx^{\underline{k}}$ (这是由于 $x^{\underline{k+1}} = (x-k)x^{\underline{k}}$ ，展开即得)，因此当 $n > 0$ 的时候有

$$\begin{aligned}
\sum_{k=0}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\} x \cdot x^k &= \sum_{k=0}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\} x^{k+1} + \sum_{k=0}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\} k x^k \\
&= \sum_{k=1}^{n+1} \left\{ \begin{matrix} n \\ k-1 \end{matrix} \right\} x^k + \sum_{k=1}^n k \left\{ \begin{matrix} n \\ k \end{matrix} \right\} x^k \\
&= \sum_{k=1}^{n+1} \left(\left\{ \begin{matrix} n \\ k-1 \end{matrix} \right\} + k \left\{ \begin{matrix} n \\ k \end{matrix} \right\} \right) x^k = \sum_{k=0}^{n+1} \left\{ \begin{matrix} n+1 \\ k \end{matrix} \right\} x^k
\end{aligned}$$

证毕.

同样的, 可以用第一类 Stirling 数在普通幂和上升幂之间转换:

$$x^{\overline{n}} = \sum_{k=0}^n \left[\begin{matrix} n \\ k \end{matrix} \right] x^k$$

至于另外两个方向的转化, 只需要做一个类似容斥的东西:

$$x^n = \sum_{k=0}^n \left\{ \begin{matrix} n \\ k \end{matrix} \right\} (-1)^{n-k} x^{\overline{k}} \quad x^{\overline{n}} = \sum_{k=0}^n \left[\begin{matrix} n \\ k \end{matrix} \right] (-1)^{n-k} x^k$$

【统一省选 2020】组合数问题

众所周知, 小葱同学擅长计算, 尤其擅长计算组合数。小葱现在希望你计算

$$\left(\sum_{k=0}^n f(k) \times x^k \times \binom{n}{k} \right) \bmod p$$

的值。其中 n, x, p 为给定的整数, $f(k)$ 为给定的一个 m 次多项式 $f(k) = a_0 + a_1 k + a_2 k^2 + \dots + a_m k^m$ 。 $\binom{n}{k}$ 为组合数, 其值为 $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ 。

对于所有测试数据: $1 \leq n, x, p \leq 10^9, 0 \leq a_i \leq 10^9, 0 \leq m \leq \min(n, 1000)$ 。

Solution: 【统一省选 2020】组合数问题

首先把 $f(k)$ 展开, 然后用第二类 Stirling 数展开普通幂得到

$$\sum_{k=0}^n \sum_{i=0}^m a_i k^i x^k \binom{n}{k} = \sum_{k=0}^n \sum_{i=0}^m a_i x^k \sum_{j=0}^i \left\{ \begin{matrix} i \\ j \end{matrix} \right\} k^{\underline{j}} \binom{n}{k}$$

由 $k \binom{n}{k} = n \binom{n-1}{k-1}$ 可得

$$k^{\underline{j}} \binom{n}{k} = n^{\underline{j}} \binom{n-j}{k-j}$$

同时交换求和顺序, 原式变为

$$\sum_{i=0}^m \sum_{j=0}^i a_i \left\{ \begin{matrix} i \\ j \end{matrix} \right\} n^{\underline{j}} \sum_{k=0}^n x^k \binom{n-j}{k-j}$$

关注后面对于 k 的求和 $\sum_{k=0}^n x^k \binom{n-j}{k-j}$, 它等于

$$\sum_{k=0}^n x^k \binom{n-j}{k-j} = \sum_{k=0}^n x^{n-k} \binom{n-j}{k} = x^j \sum_{k=0}^{n-j} x^{(n-j)-k} \binom{n-j}{k} = x^j (1+x)^{n-j}$$

于是原式等于

$$\sum_{i=0}^m \sum_{j=0}^i a_i \begin{Bmatrix} i \\ j \end{Bmatrix} n^j x^j (1+x)^{n-j} = \sum_{j=0}^m n^j x^j (1+x)^{n-j} \sum_{i=j}^m a_i \begin{Bmatrix} i \\ j \end{Bmatrix}$$

预处理第二类 Stirling 数即可做到 $\mathcal{O}(m^2)$ 。

【清华集训 2016】如何优雅地求和

有一个多项式函数 $f(x)$ ，最高次幂为 x^m ，定义变换 Q ：

$$Q(f, n, x) = \sum_{k=0}^n f(k) \binom{n}{k} x^k (1-x)^{n-k}$$

现在给定函数 f 和 n, x ，求 $Q(f, n, x) \bmod 998244353$ 。

出于某种原因，函数 f 由点值形式给出，即给定 a_0, a_1, \dots, a_m 共 $m+1$ 个数， $f(x) = a_x$ 。
可以证明该函数唯一。

对于所有的测试点，保证 $1 \leq n \leq 10^9$ ， $1 \leq m \leq 2 \times 10^4$ ， $0 \leq a_i, x < 998244353$ 。

Solution: 【清华集训 2016】如何优雅地求和

先讲一个常规的 Stirling 数解法，根据上面那道题的道路，我们不妨先假设自己得知了 f 的下降幂表示，即得到了系数 f_i 使得

$$f(x) = \sum_{i=0}^m g_i x^{\underline{i}}$$

用这个展开 f 并且推式子可以得到

$$Q(f, n, x) = \sum_{i=0}^m g_i n^{\underline{i}} x^i$$

那么我们只需要知道 g_i 就可以 $\mathcal{O}(m)$ 求答案了，但是这个东西看上去比较困难。
思路是把下降幂用阶乘表示，即

$$f(x) = \sum_{i=0}^m g_i \frac{x!}{(x-i)!}$$

发现了什么？把 $x!$ 移到左边就有

$$\frac{f(x)}{x!} = \sum_{i=0}^m g_i \frac{1}{(x-i)!}$$

因此 $\text{EGF}(F) = \text{OGF}(G) * e^x$ ，也就是 $\text{OGF}(G) = \text{EGF}(F) * e^{-x}$ 。

NTT 碾过去，时间复杂度 $\mathcal{O}(m \log m)$ 。

第二种解法来自 [luogu](#) 题解的 [zky](#) 神仙，主要思路是不将 f 拆成下降幂，而是令

$$f(x) = \sum_{i=0}^m \binom{x}{i} s_i$$

然后二项式反演得到

$$s_x = \sum_{i=0}^m \binom{x}{i} (-1)^{x-i} f(i)$$

同样是 NTT 得到 s_i ，然后原式就等于

$$\begin{aligned} \sum_{k=0}^n \sum_{i=0}^m \binom{n}{k} \binom{k}{i} s_i x^k (1-x)^{n-k} &= \sum_{k=0}^n \sum_{i=0}^m \binom{n}{i} \binom{n-i}{k-i} s_i x^k (1-x)^{n-k} \\ &= \sum_{i=0}^m s_i \binom{n}{i} \sum_{k=0}^n \binom{n-i}{n-k} x^k (1-x)^{n-k} \\ &= \sum_{i=0}^m s_i \binom{n}{i} x^i \end{aligned}$$

同样是 $\mathcal{O}(m \log m)$ 的。

还有来自 EI 神仙的线性做法，具体来说，我们沿用最开始的思路，答案是 $\sum_{i=0}^m g_i n^i x^i$ 并且 $f(x) = \sum_{i=0}^m g_i x^i$ 。

考虑每个 $f(x)$ 对 g_i 的贡献， $f(i)$ 会对 g_j 产生的贡献为 $\frac{f(i)}{i!} \times [x^{j-i}]e^{-x}$ ，带回到答案的式子， $f(i)$ 对 g_j 产生贡献会让答案加上

$$\frac{f(i)}{i!} \times \frac{(-1)^{j-i}}{(j-i)!} n^j x^j$$

于是 $f(i)$ 对答案的贡献就是（枚举 j ）：

$$\frac{f(i)}{i!} \sum_{j=i}^m \frac{(-1)^{j-i}}{(j-i)!} n^j x^j = \frac{f(i)}{i!} \sum_{j=0}^{m-i} (-1)^j \frac{n^{i+j}}{j!} x^{i+j}$$

\sum 里除以一个 n^i ， \sum 外乘一个 n^i ，原式变为

$$f(i) \frac{n^i}{i!} \sum_{j=0}^{m-i} (-1)^j \binom{n-i}{j} x^{i+j}$$

设后面的 \sum 里面是 $h_i = \sum_{j=0}^{m-i} (-1)^j \binom{n-i}{j} x^{i+j}$

发现 h 可以递推求，具体的，

$$\begin{aligned} h_i &= \sum_{j=0}^{m-i} (-1)^j \binom{n-i}{j} x^{i+j} = \sum_{j=0}^{m-i} (-1)^j \binom{n-i-1}{j} x^{i+j} + \sum_{j=0}^{m-i} (-1)^j \binom{n-i-1}{j-1} x^{i+j} \\ &= (-1)^{m-i} \binom{n-i-1}{m-i} x^m + \frac{1}{x} h_{i+1} - h_{i+1} \end{aligned}$$

第 14 章 生成函数

生成函数 (GF) 在组合领域有着重要的应用, 它将多项式和数列联系到了一起, 下面我们从普通生成函数开始 (OGF), 讲到指数型生成函数 (EGF), 结束于狄利克雷生成函数 (DGF)。

在下面的讨论中, 数列从第 0 项开始定义, 若数列 a 的长度为 n , 则让其第 i 项 ($i > n$) 均为 0, 即我们讨论的都是项数无穷的数列。

14.1 普通生成函数及其应用

对于每个数列 $\langle a_i \rangle$, 我们希望找到一个能够优雅地将它表示出来, 并且有某种方式让这种表示和每一个无穷数列建立双射, 由此想到多项式。

定义 14.1 (普通生成函数)

定义数列 $\langle g_i \rangle$ 的普通生成函数为下面的形式幂级数 $G(z)$:

$$G(z) = \sum_{n=0}^{+\infty} g_n z^n$$

同时, 令 $[z^n]G(z)$ 表示 g_n , 即 z^n 的系数。



定义中的形式幂级数是指, 我们不关心 z 的取值, z 在式子中只起到一个类似占位符的作用, 比起 z 来, 应该关注的是 z^n 的系数, 这对应了数列的第 n 项。

例如, 数列 $\langle 1, 1, 1, \dots \rangle$ 的 OGF 为

$$\sum_i z^i$$

但这也太奇怪了, 我们用一个无穷项的多项式去表示了一个无穷项的数列, 系数还是一一对应的, 这完全没有对我们的分析起到任何影响, 但是数列不能化简, 多项式却可以. 根据等比数列求和公式:

$$\sum_i z^i = \frac{1}{1-z}$$

先别着急否定这个式子, 或许你会说, 将 $z = 2$ 带进这个式子, 你就得出了 $2^0 + 2^1 + 2^2 + \dots = -1$ 的荒谬结果: 这个我们在将等比数列求和公式中的 z^n 强硬地变成 0 的时候, 完全没有考虑 $|z|$ 应不大于 1。

这并不影响我们的工作, 因为我们已经强调过, 这是一个形式幂级数: z 的取值怎样, 敛散性如何, 统统不是我们应该考虑的: 我们只是将数列到多项式做了一个双射, 仅此而已. 也正因如此, $\frac{1}{1-z}$ 的表示也并非毫无理由. 并且, 将 $\frac{1}{1-z}$ Taylor 展开, 同样可以得到 $\sum_i z^i$ 的结果。

对 $\frac{1}{1-z} = \sum_n z^n$ 的两边分别对 z 求导, 可以得到

$$\frac{1}{(1-z)^2} = \sum_n (n+1)z^n$$

这也就是说数列 $\langle 1, 2, 3, \dots \rangle$ 的普通生成函数为 $\frac{1}{(1-z)^2}$ 。

生成函数是一个形式幂级数, 这也就是说它同样可以进行多项式的运算, 若 $G(z)$ 和 $H(z)$ 是两个普通生成函数, 它们代表的数列分别为 $\langle g_i \rangle$ 和 $\langle h_i \rangle$, 定义其运算如下:

$$\begin{aligned}
G(z) \pm H(z) &= \sum_n (g_n \pm h_n) z^n \\
cG(z) &= \sum_n (cg_n) z^n \\
z^k G(z) &= \sum_n (g_n) z^{n+k} \\
G(z)H(z) &= \sum_n \sum_{0 \leq i \leq n} (g_i \times h_{n-i}) z^n
\end{aligned}$$

新的结果又对应了一个新的数列，最后一个运算中，两个多项式的乘法一般称作**卷积**。

回顾我们之前求导得到的结果： $\frac{1}{(1-z)^2} = \sum_n (n+1)z^n$ ，通过卷积同样可以得到，方法是将 $\frac{1}{1-z}$ 和 $\frac{1}{1-z}$ 乘一下得到 $\frac{1}{(1-z)^2}$ ，将他们展开得到的多项式卷起来得到

$$\sum_n \sum_{0 \leq i \leq n} (1 \times 1) z^n = \sum_n (n+1) z^n$$

不难从卷积的定义中得出：将某数列对应的生成函数卷上 $\frac{1}{1-z}$ 就能得到它的前缀和数列对应的生成函数。接下来介绍几个在具体问题中非常常用的生成函数封闭形式：

$$\frac{1}{(1-z)^m} = \sum_n \binom{m+n-1}{m-1} z^n$$

证明可以通过数学归纳法，这里给出一个正常思路下的证明。

考虑 $\left(\frac{1}{1-z}\right)^m = (z^0 + z^1 + z^2 + \cdots)^m$ ，它的第 n 项系数就是从 $a_1 + a_2 + \cdots + a_m = n$ 的非负整数解的个数，通过隔板法求得其为 $\binom{m+n-1}{m-1}$ 。

再看另一个生成函数：

$$\frac{1}{1-\alpha z} = \sum_n (\alpha^n) z^n$$

证明是将 $\frac{1}{1-z} = \sum_n z^n$ 中的 z 替换为 αz 。结合上一个结论，我们有

$$\frac{c}{(1-\alpha z)^m} = \sum_n c \alpha^n \binom{m+n-1}{m-1} z^n$$

类似地，将 z 替换为 $-z$ 得到

$$\frac{1}{1+z} = \sum_n (-1)^n z^n$$

两边积分得

$$\int \frac{1}{1+z} dz = \int \sum_n (-1)^n z^n dz$$

左边等于 $\ln\left(\frac{1}{1+z}\right)$ ，右边等于 $\sum_{n \geq 1} \frac{(-1)^{n-1}}{n} z^n$

我们没有考虑不定积分的常数 C ，这可以将它在 0 到 z 处定积分解决，例如在上面

$$\int_0^z \frac{1}{1+t} dt = \ln\left(\frac{1}{1+z}\right)$$

右边类似，之后的积分我们就直接写作不定积分，但这并不意味着我们可以忽略其常数 C ，而是我们可以通过这样的定积分将 C 消去。

对 $\frac{1}{1-z} = \sum_n z^n$ 两边积分得

$$\ln\left(\frac{1}{1-z}\right) = \sum_{n \geq 1} \frac{1}{n} z^n$$

还有由二项式定理直接得来的生成函数封闭形式：

$$(1+z)^r = \sum_n \binom{r}{n} z^n$$

讲了这么多，生成函数到底有什么用呢？它的一个主要用途就是求解数列通项，下面我们以 Fibonacci 数列的通项公式求法为例。

下文中 Fibonacci 数列的第 n 项记作 f_n ，它的定义为 $f_0 = 0, f_1 = 1, f_n = f_{n-1} + f_{n-2}$ 。这是一个线性递推，特征根是可以直接上的，但不是这里的重点。

将 Fibonacci 数列的普通生成函数记作 $F(z)$ ，那么根据递推式，我们有

$$\begin{aligned} F(z) &= \sum_n f_n z^n = 0 + z + \sum_{n \geq 2} (f_{n-2} + f_{n-1}) z^n = z + \sum_n f_n z^{n+2} + \sum_{n \geq 1} f_n z^{n+1} \\ &= z + z^2 F(z) + z F(z) \end{aligned}$$

移项解得 $F(z) = \frac{z}{1-z-z^2}$ ，我们得到了 $F(z)$ 的一个封闭形式，但这对我们来说没什么用处，因为它和通项没什么直接联系。

这里要用到部分分式定理：

定理 14.1

对于两个多项式 $G(x)$ 和 $H(x)$ ， $G(x)$ 的次数小于 $H(x)$ ， $H(x) = 0$ 的所有根为实数 x_1, \dots, x_m ，根 x_i 的重数为 α_i ，那么 $\frac{G(x)}{H(x)}$ 必然可以写成

$$\sum_{1 \leq i \leq m} \sum_{1 \leq j \leq \alpha_i} \frac{K_{ij}}{(x - x_i)^j}$$

的形式，其中 K_{ij} 为常数。



证明暂时略去，我们先将目标放在 Fibonacci 数列上。

根据这个定理，我们先求出 $1 - z - z^2$ 的两个根为 $\phi = \frac{-1+\sqrt{5}}{2}$ 和 $\hat{\phi} = \frac{-1-\sqrt{5}}{2}$ ，用这个定理将 $\frac{z}{1-z-z^2}$ 部分分式展开得

$$\frac{z}{1-z-z^2} = \frac{\frac{-\phi}{2\phi+1}}{z-\phi} + \frac{\frac{-\hat{\phi}}{2\hat{\phi}+1}}{z-\hat{\phi}}$$

变成我们熟悉的形式：

$$\begin{aligned} \frac{1}{1-z-z^2} &= \frac{\frac{\phi}{2\phi+1}}{\phi-z} + \frac{\frac{\hat{\phi}}{2\hat{\phi}+1}}{\hat{\phi}-z} \\ &= \frac{\frac{1}{2\phi+1}}{1-\frac{1}{\phi}z} + \frac{\frac{1}{2\hat{\phi}+1}}{1-\frac{1}{\hat{\phi}}z} \end{aligned}$$

那么， $\frac{1}{1-\frac{1}{\phi}z}$ 对应的数列可以用之前的结论得到，其第 n 项为 $\frac{1}{\phi^n}$ ，类似地， $\frac{1}{1-\frac{1}{\hat{\phi}}z}$ 的第 n 项为 $\frac{1}{\hat{\phi}^n}$ 。于是 f_n 就是这两项的和，化简得

$$\begin{aligned}
f_n &= \frac{1}{\phi^n} + \frac{1}{\hat{\phi}^n} \\
&= \frac{1}{\sqrt{5}} \left(\frac{1}{\phi^n} - \frac{1}{\hat{\phi}^n} \right) \\
&= \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right)
\end{aligned}$$

除了求解数列的通项，生成函数还可以证明一些恒等式，例如把 $(1+z)^r$ 和 $(1+z)^s$ 卷起来得到

$$(1+z)^{r+s} = \sum_n \sum_{0 \leq m \leq n} \binom{r}{m} \binom{s}{n-m} z^n$$

即 $[z^n](1+z)^{r+s} = \sum_{0 \leq m \leq n} \binom{r}{m} \binom{s}{n-m}$ ，在这同时，我们还可以直接对结果展开而不通过卷积，得到 $[z^n](1+z)^{r+s} = \binom{r+s}{n}$ ，对比两式，就得到了

$$\sum_{0 \leq k \leq n} \binom{r}{k} \binom{s}{n-k} = \binom{r+s}{n}$$

再来一个有意思的：用 $-z$ 替换 z 得到

$$(1-z)^r = \sum_n (-1)^n \binom{r}{n} z^n$$

将它和 $(1+z)^r$ 卷起来就可以得到

$$(1-z^2)^r = \sum_n \sum_{0 \leq k \leq n} (-1)^k \binom{r}{k} \binom{r}{n-k} z^n$$

于是有以下两式

$$\begin{aligned}
[z^n](1-z^2)^r &= \sum_{0 \leq k \leq n} (-1)^k \binom{r}{k} \binom{r}{n-k} \\
[z^{2n}](1-z^2)^r &= (-1)^n \binom{r}{n}, \quad [z^{2n+1}](1-z^2)^r = 0
\end{aligned}$$

于是就有

$$\sum_{0 \leq k \leq n} (-1)^k \binom{r}{k} \binom{r}{n-k} = [n \text{ even}] \binom{r}{n/2} (-1)^{n/2}$$

非常有意思.

下面我们从 Taylor 展开入手，看看自然对数 e 在生成函数中的表现：将 e^z Taylor 展开得

$$e^z = \sum_n \frac{1}{n!} z^n$$

e 和阶乘产生了联系，正因如此，在我们之后将要了解的指数生成函数中， e 的出现会十分频繁.

《具体数学》一书的作者在该书中用多米诺骨牌和凑零钱很生动地应用了生成函数，可惜篇幅过长，编者无意在此重复展示，有兴趣的读者可以在《具体数学》第七章中一看.

第 15 章 代数结构

15.1 二元关系

内容提要

□ 本章内容参考《离散数学》

(第二版)

[屈婉玲 2015 离散数学]

15.1.1 笛卡尔积与二元关系

由两个元素 x, y 有序排列形成的二元组称作一个**有序对**或**序偶**, 记作 $\langle x, y \rangle$, 其中 x 是第一元素, y 是第二元素.

由于这个二元组有序, 所以当 $x \neq y$ 时 $\langle x, y \rangle$ 不等于 $\langle y, x \rangle$.

设 A, B 是两个集合, 用 A 中元素作为第一元素, B 中元素作为第二元素构成有序对, 这些有序对组成的集合称作 A 和 B 的**笛卡尔积**, 记作 $A \times B$, 即

$$A \times B = \{\langle a, b \rangle \mid a \in A \wedge b \in B\}$$

不难证明, 笛卡尔积不满足交换律, 但它对并和交运算满足分配律.

定义 15.1 (二元关系)

若某集合是空集或其所有元素都是有序对, 称它是一个二元关系, 简称关系, 记作 R .

对于二元关系 R , 若有 $\langle x, y \rangle \in R$, 则称 xRy , 否则称 $x \not R y$.

设 A, B 为集合, $A \times B$ 的任何子集所定义的二元关系称作从 A 到 B 的二元关系, 特别地, 当 $A = B$ 时, 称作 A 上的二元关系.

如可以在 \mathbb{Z} 上定义小于等于关系: xRy 当且仅当 $x \leq y$. 给出一个关系的方式有三种: 直接给出集合的表达式、给出关系矩阵和给出关系图, 其中关系矩阵是指: 设 $A = \{x_1, x_2, \dots, x_n\}$, R 是 A 上的关系, 则矩阵中元素 $r_{ij} = [x_i R x_j]$, 这个矩阵就叫 R 的**关系矩阵**. 关系图是指, 若 $x_i R x_j$, 就从 i 向 j 连一条有向边构成的图.

关系同函数一样, 存在定义域 (记作 $\text{dom } R$)、值域 (记作 $\text{ran } R$), 它们的并称为关系的**域** (记作 $\text{fld } R$), 也存在逆关系, 定义 R 的逆关系为

$$R^{-1} = \{\langle x, y \rangle \mid \langle y, x \rangle \in R\}$$

关系可以复合, 定义为

$$F \circ G = \{\langle x, y \rangle \mid \exists t \text{ s.t. } \langle x, t \rangle \in F \wedge \langle t, y \rangle \in G\}$$

设 R 是关系, A 是集合, 称 R 在 A 上的**限制**为

$$R \upharpoonright A = \{\langle x, y \rangle \mid xRy \wedge x \in A\}$$

A 在 R 下的**像**为

$$R[A] = \text{ran}(R \upharpoonright A)$$

这些都和函数的定义类似或一致, 因此也有相同的性质, 如复合满足结合律、两个关系复合的逆 $(F \circ G)^{-1}$ 等于 $G^{-1} \circ F^{-1}$.

一个重要的关系是**恒等关系** I_A , 定义为

$$I_A = \{\langle x, y \rangle \mid x, y \in A \wedge x = y\} = \{\langle x, x \rangle \mid x \in A\}$$

还有**全域关系** E_A , 定义为 $E_A = A \times A$. 恒等关系的性质为: 设 R 是 A 上的关系, 则 $R \circ I_A = I_A \circ R = R$, 顺着定义即得.

通过恒等关系，我们可以定义关系 R 的 n 次幂 R^n 为

$$R^0 = I_A$$

$$R^n = R^{n-1} \circ R$$

通过两个关系的关系矩阵乘法（矩阵乘法中的加为逻辑加，即特别定义 $1 + 1 = 1$ ）得到的关系矩阵就是复合结果的关系矩阵。

15.1.2 关系的性质

这部分的内容是为了接下来的等价关系、划分以及群的陪集做铺垫，概念较多。

定义 15.2 (自反、对称、传递)

设 R 是 A 上的关系，

若 $\forall x \in A$, 有 $\langle x, x \rangle \in R$, 称 R 在 A 上是自反的. 若 $\forall x \in A$, 有 $\langle x, x \rangle \notin R$, 称 R 在 A 上是反自反的.

若 $\forall x, y \in A$ 有 $\langle x, y \rangle \in R \rightarrow \langle y, x \rangle \in R$, 称 R 为 A 上对称的关系. 若 $\forall x, y \in A$ 有 $\langle x, y \rangle \in R \rightarrow \langle y, x \rangle \notin R$, 称 R 为 A 上反对称的关系.

若 $\forall x, y, z \in A$ 有 $\langle x, y \rangle \in R \wedge \langle y, z \rangle \in R \rightarrow \langle x, z \rangle \in R$, 称 R 为 A 上传递的关系.



容易发现，关系 R 自反当且仅当 $I_A \subseteq R$, 传递当且仅当 $R \circ R \subseteq R$, 对称当且仅当 $R = R^{-1}$, 反自反当且仅当 $R \cap I_A = \emptyset$, 反对称当且仅当 $R \cap R^{-1} \subseteq I_A$.

当 R 没有某些性质如自反性时，我们有时希望向集合中添加一些元素让它有这个性质，通过添加最少的有序对使 R 变为自反关系 R' 后， R' 就叫做 R 的**自反闭包** $r(R)$ ，类似地，可以定义**对称闭包** $s(R)$ 和**传递闭包** $t(R)$ 。

可以证明， $r(R) = R \cup R^0$, $s(R) = R \cup R^{-1}$, $t(R) = R \cup R^2 \cup R^3 \cup \dots$ ，这个并虽然形式上是无穷个元素进行并，但事实上一定存在一个元素 r 使得 $t(R) = R \cup R^2 \cup \dots \cup R^r$ ，也就是说，在这之后再并是无效的了。

在计算机中求传递闭包可以用 Floyd-Warshall 算法，没错，就是图论中求全源最短路所用的算法。（它们之间有什么联系？）

15.1.3 等价关系与划分

定义 15.3

设 R 是非空集合 A 上的关系，若 R 具有自反性、对称性和传递性，称 R 是 A 上的**等价关系**。设 R 是一个等价关系，若 $\langle x, y \rangle \in R$ ，称 x 等价于 y ，记作 $x \sim y$ 。

若 R 是等价关系， $\forall x \in A$ ，令

$$[x]_R = \{y \mid y \in A \wedge xRy\}$$

称为 x 关于 R 的**等价类**，简称 x 的**等价类**，简记为 $[x]$ 或 \bar{x} 。



一个等价关系的例子是：设 $R = \{\langle x, y \rangle \mid x \equiv y \pmod{5}\}$ ，这个等价关系将所有整数按照模 5 的结果划分为了五个等价类（实际上就是数论中所说剩余系），在这个关系中，所有模 5 相同的值本质上都是等价的，如 1 和 6（在模 5 剩余系下加减乘除 1 和 6 是等效的）。

定理 15.1

设 R 是非空集合 A 上的等价关系，则

1. $\forall x \in A$, $[x]$ 是 A 的非空子集.
2. $\forall x, y \in A$, xRy 当且仅当 $[x] = [y]$.
3. $\forall x, y \in A$, $x \not R y$ 当且仅当 $[x] \cap [y] = \emptyset$.
4. $\cup\{[x] \mid x \in A\} = A$.



证明 只证 3, 若存在 z 使得 $z \in [x] \wedge z \in [y]$, 下证 $[x] = [y]$, 就可由 2 知 xRy , 矛盾. 由 $z \in [x]$ 可知 zRx , 于是 $[z] = [x]$, 同理 $[z] = [y]$, 于是 $[x] = [y]$, 证毕. 另一侧是显然的.

结合关系图理解就是: 等价关系将图划分为若干个互相独立的连通块, 其中每个连通块都是一个完全图, 将所有连通块合起来就是整张图.

定义 15.4 (商集)

设 R 为非空集合 A 上的等价关系, 以 R 的所有等价类作为元素的集合称为 A 关于 R 的商集, 记作 A/R , 即

$$A/R = \{[x]_R \mid x \in A\}$$

例如, 设 $A = \{1, 2, 3, 4, 5, 6, 7\}$, $R = \{(x, y) \mid x \equiv y \pmod{3}\}$, 那么就有

$$[1] = [4] = [7] = \{1, 4, 7\}, \quad [2] = [5] = \{2, 5\}, \quad [3] = [6] = \{3, 6\}$$

$$A/R = \{\{1, 4, 7\}, \{2, 5\}, \{3, 6\}\}$$

定义 15.5 (划分)

设 A 为非空集合, 若 A 的子集族 π ($\pi \subseteq P(A)$) 满足

1. $\phi \notin \pi$
2. $\forall x, y \in \pi, x \neq y \rightarrow x \cap y = \phi$
3. $\cup \pi = A$

则称 π 是 A 的一个划分, π 中的元素为 A 的划分块.

可以发现, 商集就是一个划分, 并且不同的商集对应不同的划分, A 上的等价关系和划分是一一对应的.

15.1.4 偏序关系

定义 15.6 (偏序关系)

设 R 为非空集合 A 上的关系, 如果 R 满足自反性、反对称性和传递性, 则称 R 为 A 上的偏序关系, 记作 \preceq , 如果 $\langle x, y \rangle \in \preceq$, 则记作 $x \preceq y$, 读作 x “小于等于” y .

这里的“小于等于”不是对数字大小的比较, 而是对集合中元素顺序的描述, 例如可以在 \mathbb{N}^* 上定义偏序关系 $x \preceq y \iff x \mid y$, 那么我们就可以说 $3 \preceq 6 \preceq 18$. 假如定义 $x \preceq y \iff y \mid x$, 那么就有 $18 \preceq 6 \preceq 3$ 了.

我们平常所说的数字的小于等于关系 \leq 和大于等于关系 \geq 就是定义在 \mathbb{R} 上的偏序关系.

定义 15.7 (可比、全序关系、偏序集与覆盖)

设 \preceq 是定义在非空集合 A 上的偏序关系, 定义

1. $\forall x, y \in A, x \prec y \iff x \preceq y \wedge x \neq y$, 读作 x “小于” y .
2. $\forall x, y \in A, x$ 与 y 可比当且仅当 $x \preceq y \vee y \preceq x$

如果 $\forall x, y \in A, x$ 与 y 都是可比的, 那么称 R 为 A 上的全序关系或线序关系.

集合 A 和集合 A 上的偏序关系 \preceq 一起称作偏序集, 记作 $\langle A, \preceq \rangle$.

设 $\langle A, \preceq \rangle$ 为偏序集, $\forall x, y \in A$, 如果 $x \prec y$ 并且不存在 $z \in A$ 使得 $x \prec z \prec y$, 则称 y 覆盖 x .

可以发现, 当我们尝试比较两个元素 x, y 时可能会出现三种情况: $x = y$ 、 x 与 y 不可比, $x \prec y$ 或 $y \prec x$. 若这是一个全序关系, 则不会出现不可比的情况.

在使用 `std::sort` 时, 我们有时候会传进一个比较函数, 这个函数实际上就是一个序关系 \prec , 当 $x \prec y$ 时返回 `True`, 否则返回假, 不能同时有 $x \prec y$ 并且 $y \prec x$, 否则可能出现错误.

定义 15.8 (最(极)小(大)元)

设 $\langle A, \preceq \rangle$ 为偏序集, $B \subseteq A, y \in B$

1. 若对于任意 $x \in B$ 有 $y \preceq x$, 称 y 为 B 的最小元.
2. 若对于任意 $x \in B$ 有 $x \preceq y$, 称 y 为 B 的最大元.
3. 若对于任意 $x \in B$ 有 $x \preceq y \rightarrow x = y$, 称 y 为 B 的极小元.
4. 若对于任意 $x \in B$ 有 $y \preceq x \rightarrow x = y$, 称 y 为 B 的极大元.



最小元和极小元的区别是: 最小元和其它元素都可比, 但极小元不一定 (注意我们的偏序关系可以不是全序的), 最小元如果存在, 它一定唯一, 但极小元一定存在却可能不唯一.

定义了这些之后我们就可以定义上界与下界了:

定义 15.9 (上界与下界)

设 $\langle A, \preceq \rangle$ 为偏序集, $B \subseteq A, y \in A$

1. 若对于任意 $x \in B$ 有 $x \preceq y$, 称 y 为 B 的上界.
2. 若对于任意 $x \in B$ 有 $y \preceq x$, 称 y 为 B 的下界.
3. 令 $C = \{y \mid y \text{ 为 } B \text{ 的上界}\}$, 则称 C 的最小元为 B 的最小上界或上确界.
4. 令 $C = \{y \mid y \text{ 为 } B \text{ 的下界}\}$, 则称 C 的最大元为 B 的最大下界或下确界.



例如在 $A = \mathbb{R}, B = \{x \mid 1 \leq x \leq 10\}$ 中, 偏序关系为 \leq 时, 任意小于等于 1 的数都是 B 的下界, 但只有一个下确界 1; 任意大于等于 10 的数都是 B 的上界, 但只有一个上确界 10.

上界、下界、上确界、下确界都可能不存在, 假如存在上确界或下确界, 它们一定唯一.

15.2 代数系统

内容提要

□ 本章内容参考《离散数学》 (第二版) [屈婉玲 2015 离散数学]

15.2.1 二元运算

定义 15.10 (二元运算)

S 是一个集合, 函数 $f: S \times S \rightarrow S$ 称为 S 上的二元运算, 简称二元运算.



例如在自然数集 \mathbb{N} 上的二元运算 $f(\langle x, y \rangle) = x + y$, 即普通的加法运算, 注意 \mathbb{N} 上的减法运算不能算是二元运算, 因为它不封闭, 减出来的数可能是负数, 并不在 \mathbb{N} 上.

相应的我们可以定义一元运算 $f: S \rightarrow S$, 例如求相反数、求绝对补集、逻辑运算中的非, 等等.

下面设 \circ 是作用在集合 S 上的二元运算, 用到时不再过多说明.

若 $\forall x, y \in S$ 有 $x \circ y = y \circ x$, 称运算 \circ 在 S 上适合 **交换律**.

若 $\forall x, y, z \in S$ 有 $(x \circ y) \circ z = x \circ (y \circ z)$, 称运算 \circ 在 S 上适合 **结合律**.

若 $\forall x \in S$ 有 $x \circ x = x$, 称运算 \circ 在 S 上适合 **幂等律**, 若只对某些特定的 x 成立, 称这些 x 为 **幂等元**. 例如集合的交、并操作就适合幂等律.

设 \circ 和 $*$ 是集合 S 上的两个二元运算, 若对任意 x, y, z 有

$$x * (y \circ z) = (x * y) \circ (x * z)$$

$$(y \circ z) * x = (y * x) \circ (z * x)$$

则称运算 $*$ 对 \circ 是可分配的, 或说 $*$ 对 \circ 适合 **分配律**.

使用归纳法可以证明, 若 $*$ 对 \circ 适合分配律, 则同样适合广义分配律, 即

$$\begin{aligned}x * (y_1 \circ y_2 \circ \cdots \circ y_n) &= (x * y_1) \circ (x * y_2) \circ \cdots \circ (x * y_n) \\(y_1 \circ y_2 \circ \cdots \circ y_n) * x &= (y_1 * x) \circ (y_2 * x) \circ \cdots \circ (y_n * x)\end{aligned}$$

设 $*$ 和 \circ 是集合 S 上的两个可交换二元运算, 若对任意 $x, y \in S$ 有

$$x * (x \circ y) = x$$

$$x \circ (x * y) = x$$

则称 $*$ 和 \circ 满足吸收律, 如对幂集 $P(S)$ 上的 \cap 和 \cup 运算满足吸收律.

若存在 $e_l \in S$ 使得对于任意 $a \in S$ 有 $e_l \circ a = a$, 则称 e_l 为一个左单位元, 相应的, 若存在 $e_r \in S$ 使得对于任意 $a \in S$ 有 $a \circ e_r = a$, 则称 e_r 为一个右单位元, 若 e 既是左单位元, 又是右单位元, 则称其为单位元或么元.

例如在 \mathbb{N} 上的加法中, 0 就是单位元, 在 \mathbb{R} 上的乘法中, 1 就是单位元.

性质 若 e_l 和 e_r 分别为左单位元和右单位元, 那么一定有 $e_l = e_r = e$ 且 e 是唯一的单位元.

证明 由于 e_r 是右单位元, 所以 $e_l = e_l \circ e_r$.

由于 e_l 是左单位元, 所以 $e_r = e_l \circ e_r$.

因此 $e_l = e_r$.

若 e 和 e' 都是单位元, 同上可得 $e = e'$.

若存在 θ_l (或 θ_r) 使得对于任意 $a \in S$ 有 $\theta_l \circ a = \theta_l$ (或 $a \circ \theta_r = \theta_r$), 则称 θ_l (或 θ_r) 为左零元 (或右零元). 若 θ 既是左零元又是右零元, 称其为零元.

性质 $\theta_l = \theta_r = \theta$ 且 θ 唯一.

证明略.

设 e 为单位元, 若对于 $x \in S$, 存在 $y_l \in S$ (或 $y_r \in S$) 使得 $y_l \circ x = e$ (或 $x \circ y_r = e$), 则称 y_l (或 y_r) 为 x 的左逆元 (或右逆元). 若 y 既是 x 的左逆元又是 x 的右逆元, 称其为 x 的逆元, 若 x 存在逆元, 则称 x 可逆.

例如在 \mathbb{R}^* 上, x 的乘法逆元为 $\frac{1}{x}$.

性质 若运算 \circ 可结合, x 存在左逆元 y_l , 右逆元 y_r , 则 $y_l = y_r = y$ 且 y 唯一.

证明

$$y_l = y_l \circ e = y_l \circ (x \circ y_r) = (y_l \circ x) \circ y_r = e \circ y_r = y_r$$

同上可得逆元唯一.

若对任意 $x, y, z \in S$ 满足

1. 若 $x \circ y = x \circ z$ 且 $x \neq \theta$, 则 $y = z$.
2. 若 $y \circ x = z \circ x$ 且 $x \neq \theta$, 则 $y = z$.

称运算 \circ 满足消去律, 其中第一条为左消去律, 第二条为右消去律.

15.2.2 代数系统

定义 15.11 (代数系统)

非空集合 S 和 S 上 k 个一元或二元运算 f_1, f_2, \dots, f_k 组成的系统称为一个代数系统, 简称代数, 记作 $\langle S, f_1, f_2, \dots, f_k \rangle$.



在代数系统中可能有一些特殊的元素, 如单位元、零元, 有时候为了突出强调, 可以将它们一并写进表示中. 定义代数系统时, 存在某种特殊元素可能是系统的性质, 例如规定必须有单位元, 这时称这些元素为该代数系统的特异元素或代数常数. 在表示一个代数系统时, 有时为了简便, 用集合名字直接代指, 如 $\mathbb{R}, \mathbb{Z}, P(S)$.

若两个代数系统中的运算个数相同、对应运算的元数相同且代数常数的个数也相同, 称它们是同类型的代数系统. 如 $\langle \mathbb{Z}, +, \times, 0, 1 \rangle$ 和 $\langle \mathbb{R}, +, \times, 0, 1 \rangle$ 同类型, $\langle \mathbb{R}, +, \times, -, 0, 1 \rangle$ 和 $\langle P(B), \cup, \cap, \sim, \phi, B \rangle$ 同类型.

定义 15.12 (代数系统的同态与同构)

设 $V_1 = \langle A, \circ \rangle$, $V_2 = \langle B, * \rangle$ 是同类型的代数系统, 函数 $f: A \rightarrow B$ 且对于任意 $x, y \in A$ 有

$$f(x \circ y) = f(x) * f(y)$$

就称 f 是 V_1 到 V_2 的同态映射, 简称同态.



根据 f 的性质可以将同态分为单同态、满同态和同构, 其中同构指 f 是一个双射, 记作 $V_1 \cong V_2$, 满射时称作满同态, 记作 $V_1 \sim V_2$, 此时称 V_2 为 V_1 的同态像.

若 f 是从 V 到 V 的, 则称 f 为自同态.

15.3 群论基础

15.3.1 基本定义

群是群论中的一个重要研究对象, 实际上就是一个代数系统 (S, \circ) , 其中 S 是一个集合, \circ 是作用在这个集合中元素的二元运算, 只不过这个二元组满足一些好的性质.

定义 15.13 (群)

对于一个集合 G 和一个作用在 G 上的二元运算 \circ , 假如它们满足

- 封闭性: $\forall a, b \in G, a \circ b \in G$
- 结合律: $\forall a, b, c \in G, (a \circ b) \circ c = a \circ (b \circ c)$
- 存在单位元: $\exists e \in G, \text{ s.t. } \forall a \in G, a \circ e = e \circ a = a$
- 有逆元: $\forall a \in G, \exists a^{-1} \in G, \text{ s.t. } a \circ a^{-1} = a^{-1} \circ a = e$

我们就可以说, (G, \circ) 构成了一个群.

若集合 G 是有穷集, 则称群 G 是有限群, 否则为无限群, 集合 G 的基数称为群 G 的阶.

只含单位元的群称为平凡群.



练习 15.1 证明整数加法 $(\mathbb{Z}, +)$ 构成群.

证明 封闭性: $\forall a, b \in \mathbb{Z}, a + b \in \mathbb{Z}$, 结合律显然.

单位元为 0: $\forall a \in \mathbb{Z}, a + 0 = 0 + a = a$.

a 存在逆元 $-a$: $\forall a \in \mathbb{Z}, a + (-a) = (-a) + a = 0$.

类似地可以证明, $(\mathbb{R}, +)$, (\mathbb{R}^*, \times) 也都构成群.

在不引起歧义的情况下, 我们可以将 $a \circ b$ 简记为 ab , 将 n 个 a 的连续运算记为 a^n , 如 $a^3 = a \circ a \circ a$.

由于群不一定满足交换律, 所以 ab 不一定等于 ba . 特别的, 满足交换律的群称作交换群或阿贝尔群.

接下来我们来简单介绍一下群的良好性质.

性质 对于一个群 (G, \circ) , 它的单位元唯一.

证明 设有两个单位元 $e_1 \neq e_2$, 那么有 $e_1 \circ a = a$, 取 $a = e_2$, 于是 $e_1 \circ e_2 = e_2$, 对于 e_2 , 同样有 $a \circ e_2 = a$, 取 $a = e_1$, 得到 $e_1 \circ e_2 = e_1$, 结合两式即得 $e_1 = e_1 \circ e_2 = e_2$.

性质 对于一个群 (G, \circ) 的每个元素 a , 其逆元 a^{-1} 唯一.

证明 反证法, 假设 a 有两个逆元 b, c , 于是 $b = be = b(ac) = (ba)c = ec = c$.

性质 $ax = b$ 当且仅当 $x = a^{-1}b$

证明 从右侧推到左侧是容易的, 直接代入即可.

如何从左侧推到右侧呢? 我们对等式两边同时左乘 a^{-1} 就得到 $x = a^{-1}b$ 了.

从这里我们也可以发现, 群满足消去律.

还有几个比较显然的性质, 留作练习:

- $(a^{-1})^{-1} = a$

- $(ab)^{-1} = b^{-1}a^{-1}$

由于很多类似的结构有封闭性和结合律，但没有单位元或有的元素没有逆元，我们把这样的代数结构叫做半群，例如在模 4 乘法群中，2 就没有逆元， (\mathbb{Z}_4, \times) 构成半群。若半群存在单位元，我们就叫它么半群，这是因为单位元也可以称作么元。

定义 15.14 (元素的阶)

设 G 是群， $a \in G$ ，那么使得 $a^k = e$ 成立的最小正整数 k 叫做 a 的阶，记作 $|a| = k$ ， a 叫做 k 阶元。若不存在这样的 k ，就说 a 是无限阶元。



15.3.2 子群

定义 15.15 (子群)

设 (G, \circ) 是一个群，假如对于非空子集 $H \subseteq G$ ， (H, \circ) 也是一个群，那么称 (H, \circ) 是 (G, \circ) 的一个子群，记作 $(H, \circ) \leq (G, \circ)$ (也可以直接记作 $H \leq G$)。

特别地，若集合 H, G 满足 $H \subset G$ ，称 H 是 G 的真子群，记作 $H < G$ 。

对 G 而言， G 和 e 显然都是 G 的子群，称作 G 的平凡子群。



例如， $(\mathbb{Z}, +) \leq (\mathbb{Q}, +) \leq (\mathbb{R}, +) \leq (\mathbb{C}, +)$ 。

性质 H 的单位元即为 G 的单位元，同时 $\forall a \in H, a^{-1} \in H$ ，这里的逆元是在 G 中的逆元。

证明 若对于 $a, b \in G$ 有 $ab = a$ ，那么左乘 a^{-1} 即得 $b = e$ ，于是一定不存在 $e' \neq e$ 使得 $\forall a \in H, ae' = a$ ，但又由于 H 是群，所以 $e \in H$ 且是 H 的逆元。

设 a 在 G 中的逆元是 a_G^{-1} ，在 H 中的逆元是 a_H^{-1} ，那么就有 $a \circ a_G^{-1} = a \circ a_H^{-1} = e$ ，同时左乘 a_H^{-1} (它一定在 G 中) 就可以得到二者相等。

那么 $H \leq G$ 的充要条件是：

- $H \subseteq G$ 且 $H \neq \emptyset$.
- $e \in H, \forall a \in H, a^{-1} \in H$

上面我们已经证明过了必要性，而充分性是容易证明的。

除了定义之外，我们还有三个子群的判定定理：

定理 15.2 (子群判定定理一)

H 是 G 的子群当且仅当 $H \neq \emptyset$ 且

- $\forall a, b \in H$ 有 $ab \in H$
- $\forall a \in H$ 有 $a^{-1} \in H$



证明 只需证明 $e \in H$ 即可，这是显然的，因为总能找出 a 和 a^{-1} 。

定理 15.3 (子群判定定理二)

H 是 G 的子群当且仅当 $H \neq \emptyset$ 且 $\forall a, b \in H$ 有 $ab^{-1} \in H$ 。



证明 首先取 $b = a$ 得到 $e \in H$ ，之后取 $a = e$ 就有 $\forall b \in H, b^{-1} \in H$ 。

定理 15.4 (子群判定定理三)

G 为群， H 是 G 的非空子集，若 H 是有穷集，则 $H \leq G$ 当且仅当 $\forall a, b \in H$ 有 $ab \in H$ 。



证明 只需证明 $\forall a \in H, a^{-1} \in H$ 。

任取 $a \in H$ ，若 $a = e$ 则 $a^{-1} = e^{-1} = e \in H$ 。否则令 $S = \{a, a^2, a^3, \dots\}$ ，有 $S \subseteq H$ ，又由于 H 是有穷集，必然存在 $i < j$ 使 $a^j = a^i$ ，于是 $a^{j-i} = e$ 。因为 $a \neq e$ ，所以 $j - i > 1$ ，则 $a^{j-i-1} = a^{-1} \in S \subseteq H$ 。

使用这些判定定理可以得出一些有用的结论.

性质 G 为群, $a \in G$, 令 $H = \{a^k \mid k \in \mathbb{Z}\}$, 则 H 是 G 的子群, 称作由 a 生成的子群, 记作 $\langle a \rangle$.

证明 任取 $a^x, a^y \in \langle a \rangle$ 有 $a^x(a^y)^{-1} = a^{x-y} \in \langle a \rangle$, 由子群判定定理二 15.3 可知 $\langle a \rangle \leq G$.

性质 G 是群, 令 C 为其所有可交换的元素构成的集合, 即

$$C = \{a \mid a \in G \wedge \forall x \in G, ax = xa\}$$

则 $C \leq G$, 称 C 为 G 的中心.

证明 首先 C 非空, 这是由于 $e \in C$, 考虑用判定定理二 15.3, 只需证明 ab^{-1} 可交换.

$$(ab^{-1})x = ab^{-1}(x^{-1})^{-1} = a(x^{-1}b)^{-1}$$

由于 b 可交换, $x^{-1}b = bx^{-1}$, 于是

$$a(x^{-1}b)^{-1} = a(bx^{-1})^{-1} = axb^{-1}$$

又由于 a 可交换:

$$axb^{-1} = xab^{-1} = x(ab^{-1})$$

性质 设 G 是群, H, K 是 G 的子群, 那么

- $H \cap K$ 是 G 的子群
- $H \cup K$ 是 G 的子群当且仅当 $H \subseteq K$ 或 $K \subseteq H$.

证明略.

设 B 是 G 的子集且 G 是群, 将 G 的所有子群中包含 B 的子群的交称作由 B 生成的子群, 记作 $\langle B \rangle$, 即

$$\langle B \rangle = \bigcap \{H \mid B \subseteq H \wedge H \leq G\}$$

由上面的性质, $\langle B \rangle$ 显然是 G 的子群, 其中的元素均为 B 中元素或其逆元.

15.3.3 群的陪集

定义 15.16 (群的陪集)

设 H 是群 G 的子群, $a \in G$, 令 $Ha = \{ha \mid h \in H\}$, 称它是子群 H 在 G 中的右陪集, a 为 Ha 的代表元素. 类似地定义左陪集 $aH = \{ah \mid h \in H\}$

左陪集和右陪集的性质是一致的, 下面我们讨论右陪集的性质.

定理 15.5

G 是群, $H \leq G$, 则 $\forall a, b \in G$ 有

$$a \in Hb \iff ab^{-1} \in H \iff Ha = Hb$$

证明 先证前半部分, 即 $a \in Hb \iff ab^{-1} \in H$:

$$a \in Hb \iff \exists h \in H \text{ s.t. } a = hb \iff ab^{-1} \in H$$

再证另一部分, 即 $a \in Hb \iff Ha = Hb$:

$Ha = Hb \implies a \in Hb$ 是显然的, 这是因为 $a \in Ha$, 只需证 $a \in Hb \implies Ha = Hb$.

任取 ha , 只要证明存在 $h' \in H$ 使得 $h'b = ha$, 我们就证明了 $Ha \subseteq Hb$. 接下来取 $h' = hab^{-1}$, 由于 $ab^{-1} \in H$ 且 $h \in H$, 由群的封闭性, $h' \in H$, 命题得证. 接下来证明 $Hb \subseteq Ha$, 只需证存在 h' 使得 $h'a = hb$, 此时 $h' = hba^{-1}$, 同样的, 因为 $ba^{-1} = (ab^{-1})^{-1} \in H$, 所以 $h' \in H$, 命题得证.

因此 $Ha = Hb$, 证毕.

定理 15.6

设 H 是 G 的子群, 在 G 上定义二元关系 $R: \forall a, b \in G$,

$$\langle a, b \rangle \in R \iff ab^{-1} \in H$$

则 R 是 G 上的等价关系, 且 $[a]_R = Ha$.



证明 自反性、传递性显然, 对称性: $ab^{-1} = (ba^{-1})^{-1} \in H$, 由于 H 是群, 所以 $ba^{-1} \in H$.

于是 R 是 G 上的等价关系, 下证 $[a]_R = Ha$.

任取 $b \in G$, 有

$$b \in [a]_R \iff \langle a, b \rangle \in R \iff ab^{-1} \in H$$

根据之前的定理15.5可知

$$ab^{-1} \in H \iff Ha = Hb \iff b \in Ha$$

于是就证明了 $b \in [a]_R \iff b \in Ha$.

推论: 设 H 是 G 的子群, 则

1. $\forall a, b \in G, Ha = Hb$ 或 $Ha \cap Hb = \phi$
2. $\cup\{Ha \mid a \in G\} = G$

因此, H 的所有右陪集的集合恰为 G 的一个划分, 并且在接下来我们会证明, **这个划分的所有划分块都与 H 等势**.

一般来说是不能得到 $Ha = aH$ 的, 假如对所有的 $a \in G$ 都有 $aH = Ha$, 那么称 H 为 G 的正规子群或不变子群, 记作 $H \trianglelefteq G$. 任何群 G 都至少有两个正规子群 $\{e\}$ 和 G .

尽管 Ha 和 aH 可能不同, 但 H 在 G 中的右陪集个数和左陪集个数却是相等的:

性质 令

$$S = \{Ha \mid a \in G\} \quad T = \{aH \mid a \in G\}$$

必然有 $|S| = |T|$

证明 设计一个函数 $f: S \rightarrow T, f(Ha) = a^{-1}H, \forall a \in G$, 容易证明 f 是一个双射函数.

因此不再区分左陪集数和右陪集数, 统称为 H 在 G 中的陪集数, 也称作 H 在 G 中的**指数**, 记作 $[G:H]$, 由此引出著名的**拉格朗日定理**:

定理 15.7 (Lagrange 定理)

设 G 为有限群, H 为 G 的子群, 则

$$|G| = |H| \cdot [G:H]$$



证明 只需证明上文中提到的“任意划分块都与 H 等势”, 即 $\forall a \in G, |Ha| = |H|$ 即可.

$|Ha| \leq |H|$ 是显然的, 因为 $|Ha|$ 中的每个元素都是 H 中的一个元素和 a 运算得到的, 只需证明 $\forall h_1, h_2 \in H$, 若 $h_1 \neq h_2$ 则 $h_1a \neq h_2a$.

反证法, 右乘 a^{-1} , 结论显然.

推论 1: 设 G 是 n 阶群, 则 $\forall a \in G, |a|$ 是 n 的因子, 且 $a^n = e$.

证明 任取 $a \in G$, 由于 $\langle a \rangle$ 是 G 的子群, 由 Lagrange 定理可得 $\langle a \rangle$ 的阶是 n 的因子. 另一方面, 设 $|a| = r$, 则

$$\langle a \rangle = \{a^0, a^1, \dots, a^{r-1}\}$$

也就是说 $\langle a \rangle$ 的阶就是 $|a|$, 因此 $|a|$ 是 n 的因子, $a^n = e$ 显然.


推论 2: 设 G 是素数阶的群 (设 $|G| = p$), 则存在 $a \in G$ 使得 $G = \langle a \rangle$

证明 由 $p \geq 2$ 得一定存在一个非单位元 $a \in G$, 则 $\langle a \rangle$ 是 G 的子群, 由 Lagrange 定理, $\langle a \rangle$ 的阶是 p 的因子, 又由于 $a \neq e$ 所以阶不为 1, 因此阶为 p , 于是 $G = \langle a \rangle$.

15.3.4 循环群与置换群

定义 15.17 (循环群)

若存在 $a \in G$ 使得 $G = \langle a \rangle$, 则称 G 为循环群, a 为 G 的生成元. $G = \langle a \rangle$ 可以根据 a 的阶分为 n 阶循环群和无限循环群.

例如整数加群 $\langle \mathbb{Z}, + \rangle$ 是无限循环群, 生成元为 1 和 -1 , 模 6 整数加群是 6 阶循环群, 生成元是 1 和 5. 

性质 设 $G = \langle a \rangle$ 是循环群.

1. 若 G 是无限循环群, 则 G 只有两个生成元 a 和 a^{-1} .
2. 若 G 是 n 阶循环群, 则 G 含有 $\varphi(n)$ ¹ 个生成元, 对于任意小于 n 且与 n 互质的自然数 r , a^r 是 G 的生成元.

证明 对于 1, a^{-1} 显然是 G 的生成元, 只需证明没有另外的生成元 b . 由于 a 在 $\langle b \rangle$ 中, 所以必然有 t 使得 $a = b^t$, 又 b 在 $\langle a \rangle$ 中, 所以有 m 使得 $b = a^m$, 于是

$$a = b^t = (a^m)^t = a^{mt}$$

于是 $a^{mt-1} = e$, 由于 G 是无限循环群, 所以 $mt = 1$, 只有 $m = t = 1$ 或 $m = t = -1$.

对于 2, 当 $n = 1$ 时显然成立, 只需证明 $n > 1$ 的情况. 先证充分性: 设 $\gcd(r, n) = 1$ 且 $r < n$, 由裴蜀定理 12.1, 必然有整数 u, v 使 $ur + vn = 1$, 因此 $a = a^{ur+vn} = (a^r)^u \circ (a^n)^v = (a^r)^u$, 得证. 再证必要性, 设 a^r 是 G 的生成元, 则 $|a^r| = n$, 令 $d = \gcd(r, n)$, 则存在整数 t 使得 $r = dt$, 则

$$(a^r)^{\frac{n}{d}} = (a^n)^t = e$$

于是 $d = 1$, 得证.

性质

1. 循环群的子群仍是循环群.
2. 无限循环群的子群除 $\{e\}$ 外均是无限循环群.
3. 若 $G = \langle a \rangle$ 是 n 阶循环群, 则对 n 的每个正因子 d , G 恰有一个 d 阶子群.

证明略.

定义 15.18 (置换)


设 $S = \{1, 2, 3, \dots, n\}$, S 上的任意双射函数 $\sigma: S \rightarrow S$ 称为 S 上的 n 元置换, 一般记作

$$\sigma = \begin{pmatrix} 1 & 2 & \cdots & n \\ \sigma(1) & \sigma(2) & \cdots & \sigma(n) \end{pmatrix}$$

实际上 σ 就是一个 1 到 n 的排列, 置换的复合称为两个置换的乘积.

设 σ 是 $S = \{1, 2, 3, \dots, n\}$ 上的 n 元置换, 若

$$\sigma(i_1) = i_2, \sigma(i_2) = i_3, \dots, \sigma(i_{k-1}) = i_k, \sigma(i_k) = i_1$$

并且保持 S 中的其它元素不变, 则称 σ 是 S 上的 k 阶轮换, 记作 $(i_1 i_2 \cdots i_k)$. 特别地, 若 $k = 2$, 称 σ 为 S 上的对换. 

例如,

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 1 & 5 \end{pmatrix} \quad \tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 2 & 1 & 4 & 5 \end{pmatrix}$$

是两个 5 元置换, 两个置换的复合为

$$\sigma\tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 4 & 3 & 5 \end{pmatrix} \quad \tau\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 3 & 2 & 1 & 5 \end{pmatrix}$$

¹这里指数论中的欧拉函数

根据轮换的定义和置换复合的定义, 我们总可以将一个置换拆分为若干个轮换的积, 例如上文中的两个置换

$$\sigma = (1\ 2\ 3\ 4) \quad \tau = (1\ 3)$$

称为置换的分解, 分解结果称为轮换分解式. 又由于轮换可以被拆为对换的积, 因此同样可以将置换拆分成若干对换的积.

例如上面的 $\sigma = (1\ 2\ 3\ 4)$ 可以被写成 $(1\ 2)(1\ 3)(1\ 4)$, 还可以被写成 $(1\ 2)(3\ 4)(1\ 3)$, 因此对换的分解可能不唯一, 但可以证明, 置换被分解为对换的形式后, 对换数量的奇偶性一定.

如果 n 元置换 σ 分解为对换之积后有奇数个对换, 称其为**奇置换**, 否则为**偶置换**. n 元奇置换和偶置换是一一对应的, 因此它们各有 $\frac{n!}{2}$ 个.

对于一个恒等置换 $\sigma(i) = i$, 我们将其特别地记作一个类似轮换的形式 (1) , 它同样可以写成 $(1)(2)(3) \cdots (n)$.

考虑所有的 n 元置换构成的集合 S_n , 满足封闭性、结合律, 单位元为恒等置换, 同时任意置换存在逆置换, 因此 S_n 对置换乘法构成群, 称为 n 元**对称群**. 例如, 设 $S = \{1, 2, 3\}$, 则三元对称群

$$S_3 = \{(1), (1\ 2), (1\ 3), (2\ 3), (1\ 2\ 3), (1\ 3\ 2)\}$$

接下来考虑 S_n 的子群. 设 A_n 是所有 n 元偶置换构成的集合, 可以证明 A_n 是 S_n 的子群, 称作 n 元**交错群**, 如 $A_3 = \{(1), (1\ 2\ 3), (1\ 3\ 2)\}$. S_n 的所有子群都称作 n 元**置换群**.

接下来的计数原理是群论在 OI 中的重要应用.

15.4 Burnside 引理与 Pólya 定理

定理 15.8 (Burnside 引理)

设 A 和 B 为两个集合, G 为 A 上的置换群, X 为一些从 A 到 B 的映射且这些映射在群 G 的作用下封闭, 则

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$$

其中

$$X^g = \{x \mid x \in X \wedge g(x) = x\}$$

X/G 表示 G 作用在 X 上产生的所有等价类构成的集合.

为了证明 Burnside 引理, 我们先引入**轨道稳定子定理**, 下面的 G, X 的定义同 Burnside 引理.

定理 15.9 (轨道稳定子定理)

$\forall x \in X$, 定义其稳定子为

$$G^x = \{g \mid g(x) = x \wedge g \in G\}$$

其轨道为

$$G(x) = \{g(x) \mid g \in G\}$$

那么

$$|G| = |G^x| \cdot |G(x)|$$

证明 先证 G^x 是 G 的子群.

封闭性: 若 $f(x) = x$ 且 $g(x) = x$, 那么 $(f \circ g)(x) = x$, 于是 $(f \circ g) \in G^x$.

结合律: 置换满足结合律.

单位元: 显然恒等置换 I 满足 $I(x) = x$, 因此 $I \in G^x$.

逆元: 对于任意 $g(x) = x$, $I(x) = (g^{-1} \circ g)(x) = g^{-1}(g(x)) = g^{-1}(x) = x$

由 Lagrange 定理 15.7 可得

$$|G| = |G^x| \cdot [G : G^x]$$

于是只需证 $[G : G^x] = |G(x)|$ ，为此我们构造一个映射 $\varphi(g(x)) = gG^x$ ，下证 φ 是一个双射。

若 $fG^x = gG^x$ ，则由 15.5 可知 $g^{-1} \circ f \in G^x$ ，也就是说 $g^{-1} \circ f = I$ ，则 $f = g$ ，因此这是单射。

另一方面，满射是显然的，因此 φ 是一个双射，证毕。

接下来证明 Burnside 引理：

证明

$$\sum_{g \in G} |X^g| = \sum_{g \in G} \left| \{x \mid g(x) = x\} \right| = \left| \{ \langle g, x \rangle \mid \langle g, x \rangle \in G \times X, g(x) = x \} \right| = \sum_{x \in X} |G^x|$$

根据轨道稳定子定理，有

$$\sum_{x \in X} |G^x| = \sum_{x \in X} \frac{|G|}{|G(x)|} = |G| \sum_{x \in X} \frac{1}{|G(x)|}$$

枚举在 G 的置换作用下的等价映射集合 Y ，

$$|G| \sum_{x \in X} \frac{1}{|G(x)|} = |G| \sum_{Y \in X/G} \sum_{x \in Y} \frac{1}{|G(x)|}$$

此时 x 经过群 G 作用后得到的不同映射个数为 $|G(x)| = |Y|$ ，于是

$$|G| \sum_{Y \in X/G} \sum_{x \in Y} \frac{1}{|G(x)|} = |G| \sum_{Y \in X/G} \sum_{x \in Y} \frac{1}{|Y|} = |G| \sum_{Y \in X/G} 1 = |G| \cdot |X/G|$$

整理即可得到 Burnside 引理。

Burnside 引理本质上只说了一件事情：本质不同的染色方案个数等于每个置换不动点个数的平均值，请读者停下来，仔细想一想这句话在说什么。

有了 Burnside 引理后，Pólya 定理是容易的：

定理 15.10 (Pólya 定理)

前置条件相同，设 X 为所有从 A 到 B 的映射构成的集合，有

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |B|^{c(g)}$$

其中 $c(g)$ 指的是将置换 g 分解为若干轮换的乘积后，轮换的个数。



证明 在 Burnside 引理中， $g(x) = x$ 当且仅当映射 x 将 A 中在同一个轮换中的元素映射到了同一个 B 中的元素，方案数是 $|B|^{c(g)}$ 。

【HNOI2008】Cards

小春现在很清闲，面对书桌上的 n 张牌，他决定给每张牌染色，目前小春拥有 3 种颜色：红色，蓝色，绿色。他询问 Sun 有多少种染色方案，Sun 很快就给出了答案。

进一步，小春要求染出 S_r 张红色， S_b 张蓝色， S_g 张绿色。他又询问有多少种方案，Sun 想了一下，又给出了正确答案。最后小春发明了 m 种不同的洗牌法，这里他又问 Sun 有多少种不同的染色方案。两种染色方法相同当且仅当其中一种可以通过任意的洗牌法（即可以使用多种洗牌法，而每种方法可以使用多次）洗成另一种。

Sun 发现这个问题有点难度，决定交给你，由于答案可能很大，你只需要求出答案对于 P 取模的结果。

输入数据保证任意多次洗牌都可用这 m 种洗牌法中的一种代替，且对每种洗牌法，都存在一种洗牌法使得能回到原状态。同时，对于 100% 的数据，保证 P 为一个质数， $\max\{S_r, S_b, S_g\} \leq 20$ ， $m \leq 60$ 。

Solution: 【HNOI2008】Cards

输入数据保证的其实就是给出的置换再加上恒等置换构成一个置换群（具有封闭性、单位元），因此本质不同的染色数就是对每个置换的不动点个数的平均值，考虑对每个置换求其不动点个数.

对一个置换 p ，考虑经典 trick i 向 p_i 连边，构成了若干个环，一个染色方案是 p 上的不动点当且仅当每个环上元素的颜色相同，做一个背包就好了，之后求出置换数量的逆元乘一下就好.

第 16 章 多项式

16.1 前置知识

16.2 离散傅里叶变换

试想我们需要求出两个多项式 A, B 的乘积, 记作 C , 其中

$$A = \sum_{i=0}^n a_i x^i, \quad B = \sum_{i=0}^m b_i x^i$$
$$C = A \times B = \sum_{i=0}^{n+m} c_i x^i, \quad \text{其中 } c_i = \sum_{j=0}^i a_j \times b_{i-j}$$

暴力做是 $\mathcal{O}(nm)$ 的, 若 n, m 同阶, 那么我们可以通过快速傅里叶变换 (FFT) 的离散形式做到 $\mathcal{O}(n \log n)$ 求出 C 的每一项系数, 下面展开讨论。

16.2.1 多项式表示法

常用的多项式一般是形如 $\sum_{i=0}^n a_i x^i$ 的形式的, 这叫做**系数表示法**。可不可以通过另一种方式唯一确定一个多项式呢? 答案是存在。根据多项式插值的相关知识, 我们可以知道一个 n 次多项式可以由 $n+1$ 个点确定, 例如两点确定一个一次函数, 也就是一个一次多项式, 三点确定一个二次函数, 也就是一个二次多项式, 以此类推。例如, 两个点 $(0, 1), (1, 2)$ 可以唯一确定一个一次多项式 $1+x$, 当 $x=0$ 时, $1+x=1$, 对应了点 $(0, 1)$; 当 $x=1$ 时, $1+x=2$, 对应了点 $(1, 2)$, 这种多项式的表示方法叫做**点值表示法**。

多项式的点值表示法非常的显然, 但是离散傅里叶变换和其逆变换所作的工作恰是在这两种表示的方法之间相互转换。先不管其具体原理, 假设我们已经拥有了一种算法, 他能够快速地将一个多项式的系数表示法变为点值表示法, 或是将点值表示法变为系数表示法, 对我们所做的多项式乘法有什么帮助呢?

想一想对两个点值表示的多项式进行多项式乘法, 我们是可以直接将对应的 y 乘起来作为最终点值的结果的! 例如, $x+3$ 有一个点值 $(3, 6)$, (x^2+1) 有点值 $(3, 10)$, 那么 $(x+3)(x^2+1)$ 在 $x=3$ 时的取值应该是什么呢? 是两个取值乘起来, 即 $6 \times 10 = 60$, 可以 $\mathcal{O}(1)$ 得到。

16.2.2 表示间的转换

我们要求两个系数多项式 A, B 的乘积 C 对应的系数多项式, 可以先通过这个神奇的算法将 A, B 换为点值多项式的形式, 再 $\mathcal{O}(n)$ 求出 C 的某一个点值多项式, 再通过某个神奇的逆操作将答案从点值多项式变回系数多项式。

现在我们只需要找到这个神奇的算法了, 其实这个算法就是离散傅里叶变换和其逆变换。尝试找出一个暴力的系数-点值转换方法: 对于系数表示变为点值表示, 可以每次选取一个 x , $\mathcal{O}(n)$ 算出对应的值, 这是 $\mathcal{O}(n^2)$ 的; 对于逆变换, 显然可以直接插值, 也是 $\mathcal{O}(n^2)$ 的。如果我们采取这种暴力的转换方式, 最终得到的时间复杂度还是 $\mathcal{O}(n^2)$ 的, 与直接算两个系数表示法的乘积是一样的。

傅里叶变换可以在 $\mathcal{O}(n \log n)$ 的时间复杂度内做到系数表示和点值表示之间的转换, 它的成功之处在于选取了很特殊的 n 个 x 值, 下面我们详细展开。

首先, 设 A 的次数为 n , B 的次数为 m , 那么 C 的次数是 $n+m$, 首先找到一个最小的 t 使得 $t = 2^k$, $k \in \mathbb{N}$, 同时 $t > n+m$, 将 A 和 B 都变为 $t-1$ 次多项式, 例如当 A 为三次, B 为 4 次时, 我们将它们都变为 7 次的, 下面假设我们已经做完了上述操作, 现在 A 和 B 都是 n 次多项式。

我们选取 $n+1$ 个点值 (注意 $n+1=2^k$)，它们是 $n+1$ 次单位根，即 $w_{n+1}^0, w_{n+1}^1, \dots, w_{n+1}^n$ ，下面考虑将 A 变为这些单位根的点值表示：

设 $A = F(x) = \sum_{i=0}^n a_i x^i$ ，那么

$$\begin{aligned} F(x) &= \sum_{i=0}^n a_i x^i = (a_0 x^0 + a_2 x^2 + \dots + a_{n-1} x^{n-1}) + (a_1 x^1 + a_3 x^3 + \dots + a_n x^n) \\ &= G(x^2) + x \cdot H(x^2) \end{aligned}$$

其中

$$\begin{aligned} G(x) &= a_0 x^0 + a_2 x^1 + \dots + a_{n-1} x^{\frac{n-1}{2}} \\ H(x) &= a_1 x^0 + a_3 x^1 + \dots + a_n x^{\frac{n-1}{2}} \end{aligned}$$

试试带入一个单位根 w_{n+1}^k ？

$$F(w_{n+1}^k) = G(w_{n+1}^{2k}) + w_{n+1}^k \times H(w_{n+1}^{2k})$$

有什么特别的吗？注意 $n+1$ 是 2 的幂，因此有 $w_{n+1}^{2k} = w_{\frac{n+1}{2}}^k$ ，when $0 \leq 2k \leq n+1$ ，于是

$$\begin{aligned} F(w_{n+1}^k) &= G(w_{\frac{n+1}{2}}^k) + w_{n+1}^k \times H(w_{\frac{n+1}{2}}^k) \\ F(w_{n+1}^{k+\frac{n+1}{2}}) &= G(w_{\frac{n+1}{2}}^k) - w_{n+1}^k \times H(w_{\frac{n+1}{2}}^k) \end{aligned}$$

而 G 和 H 的次数都被除了 2，不断递归下去，时间复杂度为 $\mathcal{O}(n \log n)$ 。

至于单位根的表示，用欧拉公式，有 $e^{i\theta} = \cos(\theta) + i \sin(\theta)$ 。

现在我们解决了系数到点值的变换，还剩下点值到系数的逆变换，对于这部分，可以证明：只需要将原来的单位根变为单位根的倒数跑傅里叶变换后，将每个点值除以 $n+1$ 即可得到其系数表示，下面给出证明。

设 $p_i = w_{n+1}^i, q_i = w_{n+1}^{-i}$ ，则多项式 A 的点值表示为 $A(p_0), A(p_1), \dots, A(p_n)$ ，跑一次单位根倒数的傅里叶变换后，有其表示的第 k 项（从 0 开始）为

$$\begin{aligned} \sum_{i=0}^n A(p_i) \times q_k^i &= \sum_{i=0}^n q_k^i \sum_{j=0}^n a_j \times p_i^j \\ &= \sum_{j=0}^n a_j \times \left(\sum_{i=0}^n (w_{n+1}^{j-k})^i \right) \end{aligned}$$

后面的括号里里面是一个等比数列求和，当且仅当 $j-k=0$ 时为 $n+1$ ，否则为 0，于是原式等于 $(n+1)a_k$ ，证毕！

至于怎样得到这个结果，不在本文的讨论范围之内，怎样将分治形式转化为倍增的实现以及进一步的常数优化，本文也不再展开。

在本节的最后，我们来说一下模意义下的**快速数论变换**，根据原根的性质，将单位根 w 换为原根后性质是不会变的，因此对于一些 $M-1$ 含有较多 2 的因子的质数 M 例如 998244353，就可以根据该性质做到模意义下没有精度误差的变换，称作快速数论变换 (NTT)。

16.3 分治 FFT

设 $f_0 = 1, f_i = \sum_{j=0}^{i-1} f_j g_{i-j}$ ，要求在 $\mathcal{O}(n \log^2 n)$ 的时间复杂度内求出 f ，该问题可以用分治 NTT 解决。

顾名思义，分治 NTT 就是 NTT 套一层分治，具体地可以说是 CDQ 分治。考虑当前分治到区间 $[l, r]$ ，先递归到 $[l, mid]$ 求出这部分的 f 值后考虑 $[l, mid]$ 对 $[mid, r]$ 的贡献：

$$f_i = \sum_{j=l}^{mid} f_j g_{i-j}, \quad i \in [mid+1, r]$$

这部分是一个卷积的形式，NTT 碾过去，每一层的时间复杂度是 $\mathcal{O}(n \log n)$ ，总时间复杂度 $\mathcal{O}(n \log^2 n)$ 。

设 $f_0 = 1, f_i = \sum_{j=0}^{i-1} f_j f_{i-1-j}$ ，时间复杂度还是 $\mathcal{O}(n \log^2 n)$ 。这次不能直接分治 NTT 了，因为我们在算贡献的时候可能会超过 mid ，这部分的贡献就算不对了。

怎样避免呢？算贡献时候，假如 $l = 0$ ，说明遇到了算不对贡献的情况，此时我们只算一部分贡献，即 $[0, mid]$ 卷上自己对右半边区间的贡献，剩下的那部分在之后的分治中补上，具体的，假如 $l \neq 0$ ，贡献变为原来的 2 倍即可，时间复杂度还是 $\mathcal{O}(n \log^2 n)$ 。

例题：ATcoder ABC315 Ex。

第六部分

杂项

第 17 章 离线算法

17.1 莫队

普通莫队算法可以解决离线区间询问一类问题.

Luogu P1903

墨墨购买了一套 N 支彩色画笔 (其中有些颜色可能相同), 摆成一行, 你需要回答墨墨的提问. 墨墨会向你发布如下 M 条指令:

1. $Q\ L\ R$ 代表询问你从第 L 支画笔到第 R 支画笔中共有几种不同颜色的画笔.

2. $R\ P\ Col$ 把第 P 支画笔替换为颜色 Col .

$1 \leq N, M \leq 10^5$, 输入的所有值均为不超过 10^6 的正整数.

Solution: Luogu P1903

我们以这道题作为莫队的例题讲解普通的莫队算法.

首先将这 N 支画笔分块, 将所有询问离线下来后, 以左端点所在的块的编号升序为第一关键字, 右端点升序为第二关键字排序.

依次处理这些询问, 维护两个指针 l, r 表示当前我们所处的区间, 用 ans 来维护当前所处区间的答案, 初始值可以设置为 $l = r = 1, ans = 1$.

为了能够维护 ans , 我们利用值域不大的特点 (不超过 10^6) 开一个桶 s , s_i 维护了当前区间中颜色为 i 的画笔的数量.

接下来是求答案的过程, 我们不断移动 l 和 r (每次只左移 1 或右移 1) 使得它和我们下一个要询问的区间相同, 假如我们成功地维护了 ans , 这个询问的答案就是 ans 了.

维护是容易的, 以将 l 移动到 $l - 1$ 为例, 假如 $l - 1$ 这支画笔的颜色还没有出现过, 我们就将答案加一即可; 无论有没有出现过, 我们都将 s_k 加一, 其中 k 指这支画笔的颜色. 假如是将 l 移动到 $l + 1$, 意味着我们维护的区间不再有 l 这支画笔, 用类似的方法维护 s 这个桶和当前答案 ans 即可.

每次维护是 $\mathcal{O}(1)$ 的, 于是我们的时间复杂度就是左右指针的总移动次数 (长度), 接下来证明这是 $\mathcal{O}(n\sqrt{n})$ 的 (假设 n, m 同阶).

先看左指针, 由于我们将询问按照左端点的块编号升序排序了, 所以左指针在两个左端点所在块相同的询问之间的移动距离至多是 $\mathcal{O}(B)$ 的, 其中 B 是块大小, 这里是 \sqrt{n} , 由于有 m 次询问, 于是这部分是 $\mathcal{O}(m\sqrt{n})$ 的, 而对于那些左端点所在块不同的询问之间的移动, 由于左端点所在块升序, 所以这部分最多移动 $\mathcal{O}(n)$, 所以总共就是 $\mathcal{O}(m\sqrt{n} + n) = \mathcal{O}(n\sqrt{n})$.

再看右指针, 按照左指针所在块分成 \sqrt{n} 类, 每一类内部的移动至多是 $\mathcal{O}(n)$ 的, 因为每一类内部按照右端点升序, 复杂度 $\mathcal{O}(n\sqrt{n})$. 对于不同类之间的移动, 这部分同样是 $\mathcal{O}(n\sqrt{n})$ 的, 因为最多有 $\mathcal{O}(\sqrt{n})$ 次这样的移动, 每次至多 $\mathcal{O}(n)$.

因此, 左右指针移动次数是 $\mathcal{O}(n\sqrt{n})$ 的, 这样看似暴力的做法的复杂度实际上是 $\mathcal{O}(n\sqrt{n})$.

莫队的精髓在于, 我们将询问离线后按照某种特定的方式排序后用暴力的方法解决, 经过分析, 优化可以上升到时间复杂度的层面而不只是常数.

第 18 章 分散层叠算法

18.1 简单叙述

分散层叠 (Fractional Cascading) 算法是一种并不常见的算法, 它可以对一些经典的分块问题进行很大的优化.

Luogu P6466

给出 k 个长度为 n 的有序数组.

现在有 q 个查询: 给出数 x , 分别求出每个数组中大于等于 x 的最小的数 (非严格后继), 若后继不存在, 则定义为 0.

每个查询的答案定义为 k 个后继的异或和, 你需要在线地回答这些询问.

$1 \leq k \leq 100$, $2 \leq n \leq 10^4$, $q \leq 5 \times 10^5$, $1 \leq d \leq 10$, 输入中出现的数均在 $[1, 5 \times 10^8)$ 范围内, 保证所有数组中出现的数字不重复, 每个数组严格递增.

空间复杂度限制在 $\mathcal{O}(kn)$

Solution: Luogu P6466

设这些数组分别为 a_1, a_2, \dots, a_k , 那么对于单次询问, 我们可以对每个 a_i 二分找到其答案, 时间复杂度为 $\mathcal{O}(k \log n)$, 不能承受.

假如放宽空间的限制, 可以将 a_1, a_2, \dots, a_k 合并为一个长度为 nk 的有序数组 A , 并且对其中的每个元素记录其在每个 a_i 中非严格的后继 (可以等于自己), 那么对于单次询问, 我们就可以先在 A 中二分找到大于等于 x 的最小数, 那么 a_i 中大于等于 x 的最小的数就是二分找到的数在 a_i 中的非严格后继, 空间复杂度为 $\mathcal{O}(k^2 n)$, 但单次查询的时间复杂度为 $\mathcal{O}(k + \log n)$, 预处理时间为 $\mathcal{O}(nk \log k)$ (类似归并, 用一个小根堆存下每个数组中当前最小的数, 并且对每个数组记录一个 pos 来处理后继数组).

分散层叠算法可以用 $\mathcal{O}(nk)$ 的预处理时间达到在线的单次查询 $\mathcal{O}(k + \log n)$, 并且空间复杂度仅为 $\mathcal{O}(nk)$, 接下来我们来实现这个算法.

首先再设 k 个序列 b_1, b_2, \dots, b_k , 初始时有 $b_1 = a_1$, 对于 $b_i (i > 1)$, 它的构造方式如下:

- 取出 b_{i-1} 中所有偶数位的数, 即 $b_{i-1,2}, b_{i-1,4} \dots$ 构成序列 b'_{i-1} .
- 归并 b'_{i-1} 和 a_i 得到 b_i .
- 在归并的同时, 我们维护每一个元素在 a_i 和 b'_{i-1} 中的非严格后继的下标.

用 2020 候选队论文中的例子, 若

$$a_1 = 11, 35, 46, 79, 81$$

$$a_2 = 13, 44, 62, 66$$

$$a_3 = 23, 25, 26$$

$$a_4 = 24, 64, 65, 80, 93$$

那么, 用 $t[x, y]$ 来表示一个 b 数组中的元素, 其中 t 是其值, x 和 y 是它在两个原序列中的非严格后继的下标, 有

$$b_1 = 11[1, 1], 35[2, 1], 46[3, 1], 79[4, 1], 81[5, 1]$$

$$b_2 = 13[1, 2], 35[2, 2], 44[2, 3], 62[3, 4], 66[4, 4], 79[5, 4]$$

$$b_3 = 23[1, 2], 25[2, 2], 26[3, 2], 35[4, 2], 62[4, 4], 79[4, 6]$$

$$b_4 = 24[1, 2], 25[2, 2], 35[2, 4], 64[2, 6], 65[3, 6], 79[4, 6], 80[4, 7], 93[5, 7]$$

在给定一个询问，例如 $q = 50$ 时，我们先在 b_4 中二分找到 $64[2, 6]$ ，其中 2 是指在 a_4 中二分的结果应该是 $a_4[2] = 64$ ，6 则是指在 b_3 中二分到的结果大致在下标 6，准确来说是在下标 6 即 $79[4, 6]$ 或前一个（下标 5）即 $62[4, 4]$ ，比较可得应该是后者。类似地一直往前推回去，就可以得到所有 a_i 的答案。

原候选队论文中给出的 b 数组的构造方法是逆序的，即令 $b_k = a_k$ ， b_{k-1} 为 a_{k-1} 和 b'_k 归并的结果，这样的话应该在 b_1 二分找到 a_1 的答案，不断向下找 $a_2, a_3 \dots$ 的答案，更舒服一些。在之后的分析中我们同样采用这样的构造。

单次查询的时间复杂度是 $\mathcal{O}(k + \log n)$ 的，空间复杂度是 $\mathcal{O}(nk)$ ，证明考虑每个 b_i 的空间为

$$\begin{aligned} |b_i| &\leq |a_i| + \frac{1}{2}|b_{i+1}| \\ |b_k| &= |a_k| \end{aligned}$$

故有

$$|b_i| \leq |a_i| + \frac{1}{2}|a_{i+1}| + \frac{1}{4}|a_{i+2}| + \dots$$

那么

$$\sum b_i \leq \sum a_i \left(1 + \frac{1}{2} + \frac{1}{4} + \dots\right) \leq 2 \sum a_i$$

于是 b 的空间复杂度和 a 只差了一个常数 2。

需要特别注意边界的处理，若 b_1 中所有数都小于询问值，我们需要手动把二分到的位置减一，否则之后的迭代就都挂了，之后的边界处理也是类似的。

18.2 一般情况及应用

考虑一个 DAG，每个结点的出度和入度均不超过一个常数 d ，点 u 上有一个有序序列 L_u ， q 次查询，每次询问对一条链上所有点对应的序列进行二分查找，前面的模板题本质上就是一条长度为 k 的链。

仍然沿用前面的做法，对每个点新建序列 M_u ，它是 L_u 及点 u 的所有后继 v 的 M_v 。与一般的分散层叠不同的是，对于每个后继，我们从其中均匀挑出 $\frac{1}{d}$ 归并进来（原本是挑出 $\frac{1}{2}$ ），这样做会导致分散层叠得到的答案与正确答案之间的距离不超过 d ，由于 d 是一个常数，所以还是可以做到单次 $\mathcal{O}(\log n + k)$ 查询的。

Bonus:

在之前的讨论中，我们都是将 a_i 的所有元素全塞进 b_i 中的，那可不可以只塞进去一部分，但同样做到“找到的位置和正确答案所在的位置不超过某个值”呢？显然是可以的：将 a_i 中的元素按照一定的比例系数 $\frac{1}{p}$ 均匀选取（这意味这每 p 个数就会有一个被选到）塞进 b_i 里，那么我们得到的答案位置与正确答案所在位置至多相差 p ，可以再套一层二分， $\mathcal{O}(\log p)$ 得到正确答案。

这样做有什么好处呢？单次查询的时间复杂度变为了 $\mathcal{O}(\log n + k \log p)$ ，但是建立分散层叠的时空复杂度均降为 $\mathcal{O}\left(\frac{nk}{p}\right)$ 。

区间加法，区间 Rank

给定序列 A ，多次询问，每次询问形如实现区间加 x 或区间查询不超过 x 的数的个数。

Solution: 区间加法，区间 Rank

这是一个经典的分块问题，我们先来回顾一下基本解法：

对序列分块，设块长为 B ，让每个块中的元素分别升序排列，预处理的时间复杂度是 $\mathcal{O}(n \log n)$ 。

对于区间加操作，整块打标记，这不会影响块中元素的顺序，散块暴力加然后重构：将同一个块中那些加了 x 的元素看作一个序列，没有加 x 的看作另一个序列，那么重构这个块可以通过归并这两个序列 $\mathcal{O}(B)$ 实现，总时间复杂度为 $\mathcal{O}\left(\frac{n}{B} + B\right)$ 。

对于查询操作，整块二分，散块暴力，时间复杂度为 $\mathcal{O}(B + \frac{n}{B} \log B)$ 。

询问的两个部分合起来，时间复杂度为 $\mathcal{O}(B + \frac{n}{B} \log B)$ ，取 $B = \sqrt{n \log n}$ 即可做到 $\mathcal{O}(n\sqrt{n \log n})$ 。

现在，我们用分散层叠算法将它优化至 $\mathcal{O}(n\sqrt{n \log \log n})$ 。

还记得我们在上文的 Bonus 中提到的“按一定的比例系数均匀选取”吗？对原序列分块、块内排序后，我们选取 $\frac{1}{D}$ 为比例系数构造分散层叠，这里的策略有所改变：建立多组分散层叠，具体来说，将连续的 D 个块建成一个分散层叠，那么分散层叠的组数就是 $\mathcal{O}(\frac{n}{DB})$ 。

建好分散层叠后，对于在同一组分散层叠中的块，我们可以对每个块在 $\mathcal{O}(\log D)$ 的时间复杂度内查到答案（这是因为我们选取的比例系数是 $\frac{1}{D}$ ，因此我们每次可以 $\mathcal{O}(1)$ 确定一个与正确答案相距不超过 D 的答案，在这个范围内二分即可）。

这样，对于查询操作，单组分散层叠的时间复杂度是 $\mathcal{O}(\log B + D \log D)$ ，一共有 $\mathcal{O}(\frac{n}{BD})$ 组分散层叠，并且散块还有 $\mathcal{O}(B)$ 的贡献，前两个相乘再加上第三个，得到查询操作的时间复杂度为 $\mathcal{O}(\frac{n}{BD} \log B + \frac{n}{B} \log D + B)$ 。

修改操作怎么办呢？假如区间加覆盖了整组分散层叠，直接打 tag 就行。因为这并不会影响分散层叠的结构。多少组分散层叠的结构会受到影响呢？就是边缘的那两组，按照套路，我们应该对这两组分散层叠重构，由于我们选取的比例系数为 $\frac{1}{D}$ ，而一共有 D 个长度为 B 的块，暴力重构一组分散层叠的时间复杂度为 $\mathcal{O}(B)$ ，于是修改操作的总时间复杂度为 $\mathcal{O}(B + \frac{n}{BD})$ 。

两部分加起来，操作的单次时间复杂度为 $\mathcal{O}(\frac{n}{BD} \log B + \frac{n}{B} \log D + B)$ ，不妨设操作次数和 n 同阶，那么 n 次操作的总时间复杂度为 $\mathcal{O}(\frac{n^2}{BD} \log B + \frac{n^2}{B} \log D + nB)$ 。

平衡一下复杂度，取 $B = \sqrt{n \log \log n}$ ， $D = \log n$ 就得到了 $\mathcal{O}(n\sqrt{n \log \log n})$ 的时间复杂度。

能不能再给力一点啊？这 $\mathcal{O}(n\sqrt{n \log \log n})$ 也太垃圾了吧？分散层叠就这？

实际上我们是可以做到 $\mathcal{O}(n\sqrt{n})$ 的！具体来说，建立一棵线段树，它有 $\frac{n}{B}$ 个叶子，每个叶子对应了一个块，其中存放了这个块对应的序列排序后的结果，非叶子结点则存放了它的两个儿子对应的序列归并后的结果，只不过不是完全归并，而是将每个序列挑出来一部分后归并，具体来说是选取 $\frac{1}{p}$ 的元素（左儿子选 $\frac{1}{p}$ ），右儿子选 $\frac{1}{p}$ 。

记得我们现在做的不仅是线段树，还是分散层叠，故在归并两个儿子时还要维护每个元素在两个儿子中二分的结果，实际上就是我们之前说的 DAG 上分散层叠的特殊情况（线段树的父亲向儿子连有向边）。

建出这样的结构并不困难，重要的是维护及处理答案。

首先看修改操作，我们希望动态地维护好这个线段树结构。线段树的深度是 $\log \frac{n}{B}$ 的，因此被不完全覆盖的线段个数为 $\mathcal{O}(\log \frac{n}{B})$ 。若一个区间被完全覆盖，那就意味着它及它的儿子是没有必要重构的（这很显然，因为让全部的数都加一个值不会改变分散层叠的结构）。

假如我们的比例系数 $p = 2$ ，这意味这线段树上的每个结点上的序列长度都是 B ，那么重构这些结点的时间复杂度是 $\mathcal{O}(B \log \frac{n}{B})$ 。

这看起来还是有点烂，不如试试别的比例系数？当比例系数为 $\frac{1}{3}$ 时，由于每一层需要重构的结点个数是 $\mathcal{O}(1)$ ，倒数第 i 层的结点所存序列的长度为 $(\frac{2}{3})^{i-1} B$ ，所以重构的时间复杂度就是

$$\mathcal{O}\left(B\left(1 + \frac{2}{3} + \frac{4}{9} + \frac{8}{27} + \dots\right)\right) = \mathcal{O}(B)$$

时间复杂度瞬间变得优美了起来。

修改的时间复杂度这么好，那查询呢？对于一个区间的查询，先在根结点上二分查找，之后就可以 $\mathcal{O}(1)$ 确定两个儿子中二分的结果，不断递归下去即可，这部分的时间复杂度是 $\mathcal{O}(\log n + \frac{n}{B})$ ，再加上对散块的暴力，时间复杂度为 $\mathcal{O}(B + \log n + \frac{n}{B})$ 。

查询和修改的时间复杂度加起来就是 $\mathcal{O}(B + \log n + \frac{n}{B})$ ，取 $B = \sqrt{n}$ 即可做到单次操作 $\mathcal{O}(\sqrt{n})$ 的优秀复杂度， n 次操作的时间复杂度为 $\mathcal{O}(n\sqrt{n})$ 。

分散层叠算法可以做到的还有很多，例如加强第十分块（Luogu P6578），感兴趣可以看 2020 国家候选队论文。

第七部分

从各处搞来的 trick

高维前缀和 若 $g(S) = \min_{S \subseteq T} f(T)$, S, T 都是集合, 而我们已经对于每个 S 知道了 $f(S)$, 怎样求出 $g(S)$ 呢?

一种很暴力的想法是子集枚举, 时间复杂度 $\mathcal{O}(3^n)$, 其中 n 是全集的势, 但实际上我们可以做到 $\mathcal{O}(n2^n)$, 代码如下:

```
1 for(int mask = 0; mask < (1 << n); mask ++)  
2   g[mask] = f[mask];  
3 for(int mask = (1 << n) - 1; mask >= 0; mask --)  
4   for(int i = 0; i < n; i ++)  
5     if(mask >> i & 1) g[mask ^ (1 << i)] = min(g[mask ^ (1 << i)], g[mask]);
```

实际上用类似的想法还可以求 $g(S) = \sum_{S \subseteq T} f(T)$ 或 $g(S) = \max_{S \subseteq T} f(T)$.

树的带权重心 树的带权重心是 dfs 序中位数对应结点的祖先, 具体来说, 祖先中最深的满足子树权值和不小于 $\frac{sum}{2}$ 的那个点, 其中 sum 为所有点的权值和.

xor shift 是一个伪随机数生成算法, 也可以用作 hash 函数, 实现如下:

```
1 unsigned long long Rand() {  
2   static unsigned long long seed = 'x';  
3   seed ^= seed << 13;  
4   seed ^= seed >> 17;  
5   seed ^= seed << 5;  
6   return seed;  
7 }
```

seed 的初始值是随机种子, 这里选用了定值 'x'.

作为 hash 函数的时候, 为了防止出题人对着卡, 可以在 xor shift 前后异或一个随机常数, 变成这样:

```
1 unsigned long long h(long long x) {  
2   static unsigned long long mask = 20070810;  
3   x ^= mask;  
4   x ^= x << 13;  
5   x ^= x >> 17;  
6   x ^= x << 5;  
7   x ^= mask;  
8   return x;  
9 }
```

第八部分

杂题选做

dalao

给定三个长度为 n 的排列 a, b, c , 求

$$\sum_{1 \leq x, y \leq n} [a_x < a_y] [b_x < b_y] [c_x < c_y]$$

其中, $n \leq 2 \times 10^6$.

Solution: dalao

很容易看到这是一个摆在明面上的三维偏序, 当 $n \leq 2 \times 10^5$ 时可以用 CDQ 分治等多种经典方法解决, 但是本题 $n \leq 2 \times 10^6$ 的数据范围迫使我们找到一个单 \log 复杂度的做法.

一个题目中给出的显然的性质是 a, b, c 均为长度为 n 的排列, 我们尝试从这个性质入手解决这个问题.

如果我们还是直接用三维偏序解决问题, 复杂度是降不下来的, 因为我们无法避免地要在分治过程中加一个带 \log 的数据结构 (BIT 等), 于是我们换一种思路, 把三维偏序换为多个二维偏序问题, 具体如下:

令 $T_{x,y} = [a_x < a_y] + [b_x < b_y] + [c_x < c_y]$, $S_{x,y} = \max\{T_{x,y}, T_{y,x}\}$, 不难发现 $S_{x,y}$ 的值域为 $\{2, 3\}$, 我们所求即为 $A = \sum_{1 \leq x < y \leq n} [S_{x,y} = 3]$.

再设 $B = \sum_{1 \leq x < y \leq n} [S_{x,y} = 2]$, 那么 $A + B = \binom{n}{2}$, 假如我们能再得到一个关于 A, B 的方程, 我们就可以解出 A 了.

接下来是最重要的一步, 我们再令 $P_{a,b} = \sum_{1 \leq x < y \leq n} [a_x < a_y] [b_x < b_y]$, 那么我们就有 $P_{a,b} + P_{b,c} + P_{a,c} = 3A + B$ (想一想, 为什么), 而对于这一步, 我们只需求三次二维偏序, 时间复杂度是 $\mathcal{O}(n \log n)$ 的, 答案即为两式相减再除以二.

bzoj3812 主旋律

给定一个 n 个点 m 条边的有向强连通图, 求有多少种删边的方案使得删后的图仍然强联通, 方案数模 $10^9 + 7$.

$n \leq 15, 0 \leq m \leq n(n-1)$

Solution: bzoj3812 主旋律

首先, $n \leq 15$ 的范围让我们很自然地想到状态压缩, 总的方案数是容易计算的, 我们只需要再减去不强联通的方案数即可. 一个图不强联通, 等价于将强联通分量缩点后, 原图变为一个点数大于 1 的拓扑图, 我们就按照这个等价条件进行 dp.

在做这个之前, 我们先考虑一种简单情况, 求让一个图是拓扑图的方案数, 设在集合 S 中的点的答案为 $f(S)$. 如何转移呢? 我们只需要枚举那些出度为 0 的点的集合 T , 这样, $S - T$ 到 T 的边可以随便选. 可是这样枚举是有重复计数的, 因为 T 不一定找到了所有出度为 0 的点, 解决方法是容斥, 注意到对于恰好有 a 个出度为 0 的点的方案, 它会在 $|T| = b$ 的时候被重复计算 $\binom{b}{a}$ 次, 这恰好的容斥的形式, 于是有 $f(S) = \sum_{T \subseteq S, T \neq \emptyset} (-1)^{|T|-1} f(S - T) \times 2^{\text{ways}(S-T, T)}$, 其中 $\text{ways}(S, T)$ 表示从集合 S 中的点出发, 到集合 T 中的点的边的数量.

回到本题, 怎么处理缩点的问题呢? 一种暴力的想法是直接枚举每个强联通分量有哪些点, 这样是很慢的, 只能过掉 $n \leq 8$ 的数据.

实际上我们完全没有必要知道哪个点在哪个强联通分量当中, 我们容斥需要的只有奇偶性. 令 $f(S)$ 为只考虑集合 S 中的点和它们之间的边, S 为一个强联通分量的方案数, $g_0(S)$ 为只考虑集合 S 中的点和它们之间的边, 分成偶数个强联通分量, 并且这些强联通分量互相之间没有边相连的方案数, $g_1(S)$ 是类似的, 只不过从偶数变为了奇数.

那么我们就可以很好地转移到 f 了，具体来说，

$$f(S) = 2^{\text{ways}(S,S)} - \sum_{T \subseteq S, T \neq \phi} (g_1(T) - g_0(T)) \times 2^{\text{ways}(S-T, S-T) + \text{ways}(T, S-T)}$$

现在我们需要做的就是找到 g 的转移方法， g 的转移更容易，枚举其中的一个强联通分量即可，如下：

$$g_0(S) = \sum_{T \subseteq S, T \neq \phi} f(T) \times g_1(S - T)$$

$$g_1(S) = \sum_{T \subseteq S, T \neq \phi} f(T) \times g_0(S - T)$$

需要注意的一点细节是，我们在转移 f 的时候，当 $T = S$ ，用到的 $g_1(S)$ 中不应包含 S 本身强联通的情况，所以我们可以先转移 f ，再转移 g_0 和 g_1 ，这里给出一份参考代码：

```
1 g[0][0] = 1;
2 for(int st = 1; st <= S; st++) {
3     f[st] = ksm(2, ways(st, st));
4     for(int subst = (st - 1) & st; subst; subst = (subst - 1) & st) {
5         f[st] = (f[st] - (g[1][subst] - g[0][subst]) % MOD
6             * ksm(2, ways(st ^ subst, st ^ subst) + ways(subst, st ^ subst)) % MOD) % MOD;
7         if(!(subst & lowbit(st))) continue;
8         g[1][st] = (g[1][st] + f[subst] * g[0][st ^ subst] % MOD) % MOD;
9         g[0][st] = (g[0][st] + f[subst] * g[1][st ^ subst] % MOD) % MOD;
10    }
11    f[st] = ((f[st] - g[1][st] + g[0][st]) % MOD + MOD) % MOD;
12    g[1][st] = (g[1][st] + f[st]) % MOD;
13 }
```

meal

给定一棵根结点为 1 的树，每个结点有两个权值 a_i, b_i ，两人轮流操作，每次操作任意选定一个结点 u 和 $k \leq a_u$ ，将 a_u 变为 $a_u - k$ ，同时，每次操作过后，应当满足对于任意结点 i ， $a_i \geq \sum_{j \in \text{child}_i} a_j \times b_j$ ，这里的 child_i 指的是与 i 直接相连的儿子。

询问对于每棵给定的树，是先手必胜还是后手必胜。

$n \leq 5 \times 10^4, a_i \leq 5 \times 10^4, b_i \leq 10^9$ ，数据保证初始状态符合 $a_i \geq \sum_{j \in \text{child}_i} a_j \times b_j$ 。

Solution: meal

先考虑几种特殊情况，假如 $b_i = 0$ 问题转化为一个 n 堆石子的 nim 游戏，这启发我们去考虑用 nim 游戏解决这个问题。假如 $b_i = 1$ ，我们设 $\text{num}_i = a_i - \sum_{j \in \text{child}_i} a_j \times b_j$ ，那么每次操作实际上就是选定一个结点 u ，让 num_u 变为 $\text{num}_u - k$ ，让 $\text{num}_{\text{father}}$ 变为 $\text{num}_{\text{father}} + b_u \times k$ 。

接下来就是套路的问题了，我们将所有深度为奇数的点分到 A 集合中（包括根节点），深度为偶数的点分到 B 集合中，问题相当于在 A 中直接做 nim 游戏，原因是对于每一次操作，假设上一手操作的是 B 集合中的元素，实质上就是把若干个石子塞到了 A 集合的某一堆中，那么我们就可以把他们原封不动地塞回到 B 集合中（想一想，为什么），假设上一手操作的是 A 集合中的石子，我们只需要按照 nim 游戏的策略继续即可。

回到一般的情况，注意到假如一个结点 u 满足 $b_u = 0$ ，那么它对它的父亲完全不会产生任何影响，于是我们可以断开 u 和它父亲的边，把原树分裂成森林。现在，所有 b_i 都不为 0 了（根节点不影响），那么实际上问题是没有变的，我们从 B 中删了若干个石子，加到 A 的某一堆上，同样还是可以原封不动地还回去的，于是计算出所有深度为奇数的结点的 num 值的异或即可。

toll

给定一个 n 个点的有向图，起点为 1，终点为 n ，边有边权，记第 i 条边的权值为 w_i 。现要求标记若干条边（标记一条边的费用即为这条边的权值），要求标记后对于任意一条从 1 到 n 的路径上，有且仅有一条边被标记，求最小费用。
 $2 \leq n \leq 100, 1 \leq m \leq 2500, 1 \leq w_i \leq 10^9$

Solution: toll

假如没有“仅有一边被标记”的限制，即路径上标记的边数不为 0 即可，那么本题转化为经典的最小割问题，现在考虑怎样建模能够避免路径上多条边被标记的情况。对于一种非法情况，当且仅当存在一点 u ，使得 1 到 u 的路径上有边被标记的同时 u 到 n 的路径上有边被标记，注意到，假如我们添加一条 a 到 b ，容量为 $+\infty$ 的边，我们要让 1 与 n 不连通，必定有 1 到 a 被割或 b 到 n 被割，并且 **由于割最小，两种情况不会同时成立**，那么，我们只需要对于原图的每一条边 (u, v) ，添加一条从 v 到 u ，容量为 $+\infty$ 的边，然后跑最小割即可。

JSOI2015 子集选取

给定 n 个元素的集合 $S = \{1, 2, \dots, n\}$ 和整数 k ，现在要从 S 中选出若干子集 $A_{i,j}$ ($A \subseteq S, 1 \leq j \leq i \leq k$) 排成下面所示边长为 k 的三角形（因此总共选出了 $\frac{1}{2}k(k+1)$ 个子集）。

$$\begin{array}{ccccccc} & & & & & & A_{1,1} \\ & & & & & & \\ & & & & & & A_{2,1} & A_{2,2} \\ & & & & & & A_{3,1} & A_{3,2} & A_{3,3} \\ & & & & & & \vdots & \vdots & \vdots & \ddots \\ & & & & & & A_{k,1} & A_{k,2} & A_{k,3} & \cdots & A_{k,k} \end{array}$$

此外，JYY 对选出的子集之间还有额外的要求：选出的这些子集必须满足 $A_{i,j} \subseteq A_{i,j-1}$ 且 $A_{i,j} \subseteq A_{i-1,j}$ 。JYY 想知道，求有多少种不同的选取这些子集的方法。因为答案很大，JYY 只关心输出答案模 1,000,000,007 的值。

对于两种选取方案 $A = \{A_{1,1}, A_{2,1}, \dots, A_{k,k}\}$ 和 $B = \{B_{1,1}, B_{2,1}, \dots, B_{k,k}\}$ 只要存在 i, j 满足 $A_{i,j} \neq B_{i,j}$ ，我们就认为 A 和 B 是不同的方案。

Solution: JSOI2015 子集选取

注意到，对于集合中的每个元素，我们是单独考虑的，即：对于元素 $k \in S$ ，最终方案里含有 k 的 $A_{i,j}$ 一定是堆在三角形的左上角的，设 x 为这样的方案数，那么最终结果就是 x^n 。

怎样求 x 呢？我们令 $f(k)$ 表示考虑 k 行的三角形时的方案数，对最后一行的填充方案进行分类讨论（全不含 k ，前 m 个 A 含 k ），得 $f(k) = f(k-1) + f(k-2) + \dots + f(0) = 2f(k-1)$ 。

所以最终答案就是 2^{n^k} 了，快速幂计算即可。

SP1557 GSS2 - Can you answer these queries II

n 个数和 q 次询问，每次查询给定区间 $[A_l, A_r]$ ，询问区间的最大子段和，**在这里的子段和中，重复的元素只计算一次**，子段可以为空。
 $n, q \leq 10^5$ 。

Solution: SP1557 GSS2 - Can you answer these queries II

这种“重复元素只计算一次”的题目有一个一般情况下很通用的套路，就是将询问离线，再考虑前驱，计算贡献。

在本题中，我们的离线策略是这样的：将所有询问按照右端点升序排列（在这样的前提下，左端点怎样都可以），我们从 1 到 n 依次加入这些数。什么意思呢？因为询问的右端点已经升序了，我们在依次考虑每个询问时，让 B_i 表示 $S_{i,i}, S_{i,i+1}, \dots, S_{i,r}$ 的最大值（ $S_{i,r} = \sum_{i=l}^r A_i$ ），其中， r 是当前的右端点，于是这个询问的答案即为 $\max_{i=l}^r \{B_i\}$ ，现在的任务是快速地维护 B 数组，计算 B 的区间最大值。

这时，我们对询问的排序就起到了很大的效果，在 B 数组上的表现为：当我们不断右移右端点时，对每个 i 来说， B_i 只会保持不变或者变大——这是显然的，因为之前的区间 $[i, r'], r' < r$ 没有被删去。

加入数 A_i 时， B 数组会有什么变化呢？我们令 Pre_i 为上一个使得 $A_j = A_i$ 的 j ，如果不存在，则 $Pre_i = 0$ 。容易发现，我们改变的只会是下标在 $[Pre_i + 1, i]$ 的 B 值，因为重复的元素只计算一次，所以 A_i 对于其他的 $B_j, j \leq Pre_i$ 是没有影响的（ $S_{j,i} = S_{j,i-1}$ ，而 $S_{j,i-1}$ 已经被算过了，再取一次最大值是没有意义的），影响是什么呢？对每一个 B_j ，取 $\max\{B_j, S_{j,r-1} + A_r\}$ 作为新的 B_j 。

为了更加清晰，我们将数组 B 拆开来，即让 C_i 表示 $\max_{j=i}^r \{S_{i,j}\}$ ， D_i 表示 $S_{i,r}$ ，其中 r 仍然是当前的右端点，那么，对于每次加入 A_{r+1} ，我们要进行的操作实际上是：

- 将 D_i 变为 $D_i + A_{r+1}$ ，其中 $i \in [Pre_{A_{r+1}} + 1, r + 1]$
- 将 C_i 变为 $\max\{C_i, D_i\}$ ，其中 $i \in [Pre_{A_{r+1}} + 1, r + 1]$

总结一下，我们需要用某个数据结构维护这样的操作：

- 区间历史最大值（询问）
- 区间加法后维护历史最大值（每次的修改）

这是容易维护的，我们只需要对每个结点维护：历史最大值，当前结点的所有子结点的最大值，区间加法 tag ，区间加法 tag 的最大值。合并两个结点的操作是这样的（带 $h_$ 前缀的即历史最大值，否则为当前值）：

```
1 struct Node {
2     ll mx = 0, h_mx = 0, tag = 0, h_tag = 0;
3     Node operator + (const Node &rhs) {
4         Node res;
5         res.mx = max(mx, rhs.mx), res.h_mx = max(h_mx, rhs.h_mx);
6         res.tag = res.h_tag = 0;
7         return res;
8     }
9 };
```

pushdown 函数：

```
1 inline void pushdown(int u) {
2     tr[lc].h_tag = max(tr[lc].h_tag, tr[lc].tag + tr[u].h_tag);
3     tr[rc].h_tag = max(tr[rc].h_tag, tr[rc].tag + tr[u].h_tag);
4     tr[lc].tag += tr[u].tag, tr[rc].tag += tr[u].tag;
5     tr[lc].h_mx = max(tr[lc].h_mx, tr[lc].mx + tr[u].h_tag);
6     tr[rc].h_mx = max(tr[rc].h_mx, tr[rc].mx + tr[u].h_tag);
7     tr[lc].mx += tr[u].tag, tr[rc].mx += tr[u].tag;
8     tr[u].tag = tr[u].h_tag = 0;
9 }
```

AT_agc060_c

考虑 $(1, 2, \dots, 2^N - 1)$ 的一个排列 $P = (P_1, P_2, \dots, P_{2^N - 1})$. 称 P 像堆当且仅当 $P_i < P_{2i}$ 和 $P_i < P_{2i+1}$ 对 $1 \leq i \leq 2^{N-1} - 1$ 成立.

给定正整数 A 和 B . 令 $U = 2^A$, $V = 2^{B+1} - 1$. 在所有像堆的排列中任取一个, 求 $P_U < P_V$ 的概率模 998244353.

$2 \leq N \leq 5000$, $1 \leq A, B \leq N - 1$.

Solution: AT_agc060_c

看到 $2i$ 和 $2i + 1$, 我们应该立刻想到二叉树, 实际上也的确如此, 我们按照 “结点 i 的左儿子是 $2i$, 右儿子是 $2i + 1$, 根节点是 1 来建一棵二叉树”, 让 i 结点的权值为 P_i .

可以发现, 这棵二叉树实际上也是一棵二叉堆, 我们将偏序关系反映到树上, 即: 每个结点到它儿子的边为有向边, 方向是从自己到儿子. 我们对 $< P_i, i >$ 进行一个升序排列, 下标的第二维对应到树上实际上就是图的一个拓扑序, 同样的, 对于每一种拓扑序, 都可以还原出原先的 P .

我们做了什么事呢? 就是把一个序列上的偏序问题转化为了给定的一棵特殊满二叉树的拓扑序个数问题. 令 S_i 表示 i 层的这样的满二叉树的拓扑序的数量, 则有转移方程 (先把根拿出来, 它一定在第一位, 之后树被拆分为两个 $i - 1$ 层的树, 分别拓扑排序后合并即可):

$$S_i = S_{i-1}^2 \times \binom{2^i - 2}{2^{i-1} - 1}$$

回到本题中, U 所对应的就是第 $A + 1$ 层最左边的结点, V 所对应的就是第 $B + 1$ 层最右边的结点, 现在我们已经求出了 P 的总方案数 (S_N), 只需要再求出满足 $P_U < P_V$ 的方案数, 做商即可.

先把根节点 (1) 去掉, 它现在应该在拓扑序的第一位, 之后, 整棵树又被拆为了两棵子树, U 在左侧, V 在右侧, 很自然地, 我们定义 $f_{i,j}$ 表示剩下一棵深度为 i 的满二叉树和一棵深度为 j 的满二叉树的满足约束条件的方案数, 显然 $f_{i,j}$ 能从 $f_{i-1,j}$ 和 $f_{i,j-1}$ 转移来, 具体地说:

$$f_{i,j} = f_{i-1,j} \times S_{i-1} \times \binom{2^i + 2^j - 3}{2^{i-1} - 1} + f_{i,j-1} \times S_{j-1} \times \binom{2^i + 2^j - 3}{2^{j-1} - 1}$$

首先是看两棵子树中, 哪个根在序列的第一位, 之后, 将那棵树拆开, 其中一棵含 U (或 V), 另一棵不含. 不含 U, V 的那棵子树的贡献是 $S_{i-1} (S_{j-1})$, 剩下的部分就是 $f_{i-1,j} \times \binom{2^i + 2^j - 3}{2^{i-1} - 1} (f_{i,j-1} \times \binom{2^i + 2^j - 3}{2^{j-1} - 1})$ 了.

令 $ans_{i,j}$ 为 $\frac{f_{i,j}}{S_i \times S_j \times \binom{2^i + 2^j - 2}{2^i - 1}}$, 那么答案就是 $ans_{N-1, N-1}$.

这个式子很丑, 而且由于指数很大, 我们没办法算组合数, 于是我们可以化简一下:

$$\begin{aligned} ans_{i,j} &= \frac{f_{i,j}}{S_i S_j \binom{2^i + 2^j - 2}{2^i - 1}} \\ &= \frac{f_{i-1,j} S_{i-1} \binom{2^i + 2^j - 3}{2^{i-1} - 1} + f_{i,j-1} S_{j-1} \binom{2^i + 2^j - 3}{2^{j-1} - 1}}{S_i S_j \binom{2^i + 2^j - 2}{2^i - 1}} \\ &= \frac{f_{i-1,j} S_{i-1} \binom{2^i + 2^j - 3}{2^{i-1} - 1}}{S_i^2 \binom{2^i - 2}{2^{i-1} - 1} S_j \binom{2^i + 2^j - 2}{2^i - 1}} + \frac{f_{i,j-1} S_{j-1} \binom{2^i + 2^j - 3}{2^{j-1} - 1}}{S_{j-1}^2 \binom{2^j - 2}{2^{j-1} - 1} S_i \binom{2^i + 2^j - 2}{2^i - 1}} \\ &= ans_{i-1,j} \times \frac{\binom{2^{i-1} + 2^j - 2}{2^{i-1} - 1} \binom{2^i + 2^j - 3}{2^{i-1} - 1}}{\binom{2^i - 2}{2^{i-1} - 1} \binom{2^i + 2^j - 2}{2^i - 1}} + ans_{i,j-1} \times \frac{\binom{2^i + 2^{j-1} - 2}{2^i - 1} \binom{2^i + 2^j - 3}{2^{j-1} - 1}}{\binom{2^j - 2}{2^{j-1} - 1} \binom{2^i + 2^j - 2}{2^i - 1}} \end{aligned}$$

把组合数展开成阶乘后玩一遍消消乐就会得到一个优美的式子:

$$ans_{i,j} = ans_{i-1,j} \times \frac{2^i - 1}{2^i + 2^j - 2} + ans_{i,j-1} \times \frac{2^j - 1}{2^i + 2^j - 2}$$

初始值为 $ans_{n-A-1,j} = 1$, 其中 $j \geq n - B$. 时间复杂度为 $\mathcal{O}(n^2 \log n)$

CF506E Mr. Kitayuta's Gift

给定只含小写字母的字符串 s 和正整数 n ，往 s 中任意位置插入任意 恰好 n 个小写字母，求最终字符串是回文串的方案数，两种方案相同当且仅当最终的字符串相同，与字符的插入位置和顺序无关。

$$|s| \leq 200, n \leq 10^9$$

Solution: CF506E Mr. Kitayuta's Gift

10^9 的 n 很吓人，根据经验来看，最终的复杂度应该有一个 $\log n$ 。

先考虑 $|s| + n$ 是偶数的情况。

这种计数问题肯定是 dp 了，考虑设计状态，设 $f_{x,l,r}$ 为已经填了左边 x 个字符和右边 x 个字符（因为最终是回文串，所以肯定是对称着填的），尽可能地匹配 s 后，还剩下 $s_l..s_r$ 未被匹配的方案数， g_x 为已经填了左边 x 个字符和右边 x 个字符，同时 s 已经被完全包含了的方案数。转移就考虑 s_l 和 s_r 是否相等，具体来说，有：

$$\left. \begin{aligned} f_{x+1,l+1,r} &\leftarrow f_{x,l,r} \\ f_{x+1,l,r-1} &\leftarrow f_{x,l,r} \\ f_{x+1,l+1,r} &\leftarrow 24f_{x,l,r} \end{aligned} \right\} s_l \neq s_r$$

$$\left. \begin{aligned} f_{x+1,l+1,r-1} &\leftarrow f_{x,l,r} \\ f_{x+1,l,r} &\leftarrow 25f_{x,l,r} \end{aligned} \right\} s_l = s_r, r - l > 1$$

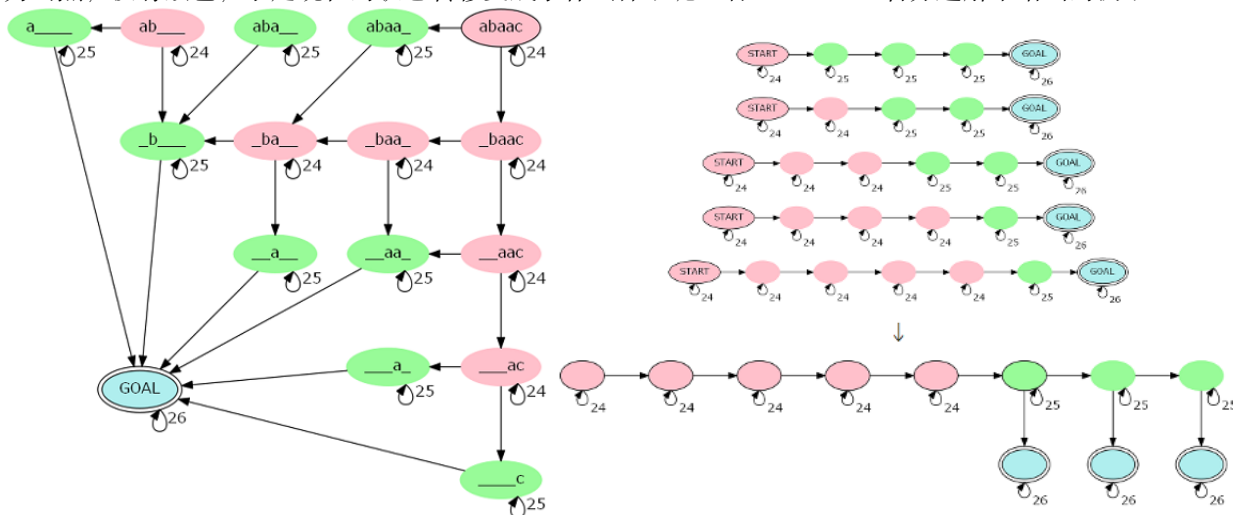
$$\left. \begin{aligned} g_{x+1} &\leftarrow f_{x,l,r} \\ f_{x+1,l,r} &\leftarrow 25f_{x,l,r} \end{aligned} \right\} s_l = s_r, r - l \leq 1$$

$$g_{x+1} \leftarrow 26g_x$$

解释一下的话就是，假如 $s_l \neq s_r$ ，那么我们在这两位上（左右分别一位）填 s_l 就可以让 s_l 被匹配，转移到 $f_{x+1,l+1,r}$ ，填 s_r 同理，剩下的 24 种填法就是无法匹配的情况，以 24 的系数转移到 $f_{x+1,l,r}$ ，表示没能多匹配上 s 。其他两类情况同理，这里不多赘述。

这样的时间复杂度实在是太垃圾了，对于本题的数据规模实在无能为力，我们考虑优化这个转移过程。

具体来说，我们将 s 的所有连续都抽象成点，其中满足 $s_l = s_r$ 的为绿点， $s_l \neq s_r$ 的则为红点，最终的结点为终点，没有颜色，于是现在的状态转移变成了什么样子呢？看 Codeforces 官方题解中给出的例子 $s = abaac$ ：



其中，自环旁的数字表示自环的数量。

我们最终要求的是什么呢？就是从起点 $s_{1..|s|}$ 经过 $k = \frac{|s|+n}{2}$ 条边走到终点 $GOAL$ 的方案数。注意到，当我们这样直接转移的时候，实际上很大一部分时间是浪费在走自环上的（因为点数远小于 k ），假如能够省去这部分开销就很好了。

对于两段不同的路径，假如它们经过的红点个数和绿点个数均相同，那么这两个路径其实是没有本质区别的，即：不妨设两段路径分别为 L_1 和 L_2 ，那么，我们从 L_1 走 k 步到终点的方案数是一定等于从 L_2 走 k 步到终点的方案数的，因为总可以建立一个双射，同时我们还得到，先走 x 个红点，再走 y 个绿点，共走 k 步后到达终点的方案数和打乱红绿点的顺序后走 k 步到达终点的方案数也是相同的，于是我们其实可以把这个图建成上方三图中右上方这个样子，即把红绿点数目相同的路径变为同一类路径，计算这一类路径走 k 步到达终点的方案数 \times 本字符串中这一类路径的个数作为这一类路径的贡献加入到答案。

我们设 $\sharp(a)$ 表示从起点开始，经过 a 个红点和 $\lceil \frac{|s|-a}{2} \rceil$ 个绿点后走到终点的方案数， $F(a)$ 表示走 a 个红点和 $\lceil \frac{|s|-a}{2} \rceil$ 个绿点，走 k 步的方案数，那么答案即为

$$\sum_a \sharp(a) F(a)$$

先看怎么算 $F(x)$ ，因为红点的最大个数一定不超过 $|s| - 1$ （因为最终走到终点的点一定是绿点，同时，每走过一个红点，剩下的字符串的长度就减一），于是我们需要对这 $|s| - 1$ 个点数 $\mathcal{O}(|s|)$ 的图均跑一次矩阵快速幂，这同样也是无法接受的（ $\mathcal{O}(|s|^4 \log n)$ ），我们必须再缩小图的规模，经过一番思考后，我们可以得出上方三图中右下方的图，我们将终点分裂了，但是图的总规模由 $\mathcal{O}(|s|^2)$ 变为了 $\mathcal{O}(|s|)$ 。对这个图跑矩阵快速幂后，经过 a 个红点和 b 个绿点，走 k 步后到达终点的方案数就是起点为从右往左数第 a 个红点，终点为从左往右数第 b 个绿点下方对应的终点的长度为 k 的路径个数。

再来考虑算 $\sharp(x)$ ，我们可以直接在不做任何修改的原图中解决，设 $g_{x,l,r}$ 为从 $s_{l..r}$ 对应的节点出发，经过 x 个红点后到达终点的方案数，转移是平凡的，同样是考虑 s_l 是否等于 s_r 即可。于是我们就可以 $\mathcal{O}(|s|^3)$ 求出所有的 $\sharp(x)$ 了，至此， $|s| + n$ 为偶数的情况已经被我们完全解决，时间复杂度为 $\mathcal{O}(|s|^3 \log n)$ 。

当 $|s| + n$ 为奇数时，哪些方案是不合法的？答案是那些 **k 步中的最后一步是从一个长度为 2 的绿点转移到终点的方案**，剔除掉这些方案并不困难，在走 $k - 1$ 步的矩阵中，我们把每个这样的绿点设为合法的终点，对于不同的路径中红点个数统计答案即可。

本题稍微有点卡常，第一是，注意到我们的矩阵是一个上三角矩阵，乘法可以优化 6 倍的常数，第二是，本题的模数很小 10007，在矩阵乘法的时候，我们可以把数组开成 long long，乘法过程中不用取模，最后一起取模就可以，第三是，我们可以先判断奇数的情况，算出负贡献后再去算正贡献，这样就可以只做一次矩阵快速幂（从 $k - 1$ 步矩阵到 k 步矩阵只需一次乘法）。

CF516D: Drazil and Morning Exercise

给定一棵 n 个点的边带权的树， $\text{dist}(u, v)$ 表示 u, v 间唯一路径的长度，定义 $f(u) = \max_{v=1}^n \{\text{dist}(u, v)\}$ ， q 次询问，每次给定数 l 求满足 $\max_{s \in S} f(s) - \min_{s \in S} f(s) \leq l$ 的联通块 S 的最大大小。
 $n \leq 10^5, q \leq 50$.

Solution: CF516D: Drazil and Morning Exercise

可以证明，使得 $f(u) = \text{dist}(u, v)$ 的 v 一定可以是任意一条直径的两端之一，有了这个性质，我们可以进一步地发现，选定一条直径 d ，在 d 上的点的 f 值是从两端想中间递减的，而对于不在 d 上的点，我们把在直径上且 f 值最小的点（称为 *root*）作为整棵树的树根，每个点的 f 值是一定大于其祖先的 f 值的，证明并不困难。

想到这个根结点的选取方案之后就并不困难了，我们将一个联通块中深度最小的结点作为这个联通块的代表元，它的 f 是整个联通块中最大的，考虑每个结点会对哪些联通块做出 1 的贡献（可以被加到那个联通块里），显然符合这些条件的联通块的代表元是从这个结点出发到根的一条链（因为链上的结点的 f 值是递增的），所以我们要实现的操作就是链加 1，子树求和，这可以用树上差分解决。

树上差分的复杂度是 $\mathcal{O}(n)$ 的，找到每个点对应的链头可以倍增解决，时间复杂度为 $\mathcal{O}(\log n)$ ，总时间复杂度为 $\mathcal{O}(qn \log n)$ 。

CF516E: Drazil and His Happy Friends

有 n 个男生 m 个女生, 编号分别为 $0 \sim n-1$ 和 $0 \sim m-1$, 其中有 b 个男生和 g 个女生是快乐的, 其他人是不快乐的.

在第 i 天, 编号为 $i \bmod n$ 的男生和编号为 $i \bmod m$ 的女生会一起玩. 如果他们俩中有一个人是快乐的, 则另一个人也会变快乐.

求至少要多少天所有人都会变快乐, 或者判断不可能所有人都变快乐.

$n, m \leq 10^9, b, g \leq 10^5$.

Solution: CF516E: Drazil and His Happy Friends

首先设 $\gcd(n, m) = d$, 那么, 男生 i 和女生 j 可能一起玩等价于 $x = k_1 \times n + i = k_2 \times m + j$, 移项可得原式等价于 $i \equiv j \pmod{d}$, 于是, 我们将所有男生和女生按照模 d 分类, 模 d 相同的在同一类中, 这样, 不同类之间的人之间不会产生影响.

若在某一类中没有一个人初始时是开心的, 那么这一类中的人必然不会有人能开心, 此时无解. 反之, 根据裴蜀定理, 一定可以让所有人开心, 这样, 当 $d \geq b + g$ 时结果一定为 -1 , 我们将不同的类的个数缩小到了 2×10^5 .

把模 d 余 i 类中的数 x 变为 $\lfloor \frac{x}{d} \rfloor$, 此时每一类中的男生个数为 $n' = \frac{n}{d}$, 女生个数为 $m' = \frac{m}{d}$, 有 $\gcd(n', m') = 1$, 我们对这样的子情况求解, 假如最开始时所有人都开心, 子情况的结果为 1 , 否则设子情况的结果为 ans , 将其中的数 x 变为原样后, 应该的结果即为 $ans \times d + i$, 表示子情况中的过一天实际上是过了 d 天, 起始点是第 i 天而不是第 0 天. 最终结果即为所有子情况的结果取 \max .

下面考虑每一种子情况, 我们将问题转化为图论问题, 对“所有女生均开心需要的天数”和“所有男生均开心需要的天数”分开, 结果又化为两部分的 \max , 下面以“所有女生均开心需要的天数”为例.

一个女生 i 在第 Day 天变得开心了, 那么使他开心的那个男生又会在第 $Day + n'$ 天让女生 $(Day + n') \bmod m'$ 变得开心, 这实际上可以看成女生 i 开心后, 过了 n' 天, 让女生 $(Day + n') \bmod m'$ 变得开心, 把男生直接忽略. 若一个女生 i 在开始就开心, 那么她会在第 i 天让 $(i \bmod n')$ 号男生变开心, 这个男生又会在 n' 天后让第 $(i + n') \bmod m'$ 个女生变开心, 同样地, 我们把男生踢开, 将这个转移化为两个女生之间的转移.

对于每个女生 i , 她向 $(i + n') \bmod m'$ 号女生连边, 边权为 n' , 表示她开心后, 能在 n' 天后让 $(i + n') \bmod m'$ 号女生也开心, 我们称这些边为一类边.

假如一个女生 i 本来就开心, 从起点 S 向 i 连一条边权为 i 的边, 表示到了第 i 天之后, 她可以让其他女生也变得开心, 假如一个男生 i 本来就开心, 那么从起点 S 向 $i \bmod m'$ 连一条权值为 i 的边, 原理类似, 这些边为二类边.

建好图之后, 从 S 出发跑最短路, 那么这个子情况的子情况的答案即为从 S 到所有最开始不开心的女生的最短路长度的最大值.

但是边数实在是太多了, 有整整 10^9 ! 我们能不能避免最短路和建图呢? 答案是可以的.

注意到, 所有第一类边将整个图连成了一个大环, 并且环上边的长度都相等, 我们设起点能到达的点为关键点, 那么关键点对答案的贡献是可以直接算的, 对于两个相邻关键点之间的点, 显然是这条链上末尾关键点的前一个结点的最短路长度最大, 我们只需要考虑这些点即可, 而这些点是 10^5 级别的.

怎么将所有关键点按照顺序排列呢? 整个环实际上是

$$0 \rightarrow n' \bmod m' \rightarrow 2n' \bmod m' \rightarrow 3n' \bmod m' \cdots \rightarrow m'n' \bmod m' = 0$$

只需要求出这个点同余于几倍的 m' 即可, 这可以用扩展欧几里得解决.

AT_agc029_f

给定 n 个点和 $n - 1$ 个集合 S_i ，每个集合包含至少两个点，现在要在每个集合中选取两个点并在这两个点之间连边，询问是否可以让这些点和边构成一棵树。如果可行，输出方案。

$$n \leq 10^5, \sum |S_i| \leq 2 \times 10^5.$$

Solution: AT_agc029_f

考虑有解的必要条件：任选一个点 u 删去，那么剩下的 $n - 1$ 个点与这 $n - 1$ 个集合构成双射，集合 S_i 中含有点 p_i ，这个性质很显然，可以考虑将 u 作为树根，那么每个结点 p_i 与它连向父亲的边 S_i 就一一对应了。

这个东西怎么判断呢？建一个二分图，左部点是 $n - 1$ 个集合，右部点是 n 个点，每一个集合和它里面的点之间连边，跑一个最大匹配，假如最大匹配小于 $n - 1$ ，那么一定是不符合条件的。

那如果最大匹配等于 $n - 1$ 呢？我们从那个没有被匹配的点出发，重复以下步骤：

- 当前点为 u ，考虑所有含有 u 且未被访问过的集合 S
- 标记 S 被访问过，将 S 选择的两个点确定为 S 在匹配中连接的点和 u
- 接着对每个 S 在匹配中连接的那个点，回到第一步扩展状态
- 假如没有找到这样的未被访问过的集合，那么一定无解

假如 $n - 1$ 个集合都被标记过，现在就已经构造出了一个解，但是当到某个点时卡住了，怎么证明一定无解呢？

- 此时，没有访问过的集合数量必然和没有访问过的点的数量相同
- 并且，已经访问过的点和这些集合完全没有任何关系（一定不在这些集合中）
- 那么，这些集合中（不妨设为 m 个）最多有 m 个点
- 这 m 条边和这 m 个点一定会连成环，于是一定构不成树

CF1225F: Tree Factory

给定一棵 n 个点的带编号有根树，你可以生成一条有编号的链，并构造一种方案以最小的操作次数将链变为给定的树（这里的最小指这条链的最小操作次数是所有链的最小操作次数的最小值）。

$$n \leq 10^5.$$

Solution: CF1225F: Tree Factory

反向考虑怎么将给定的树以最小的操作次数变为链，这个次数是 $n - d$ ，其中 d 是树的最大深度，到达这个下界是简单的，只需要每次选一个不在最长的从根出发的链上的点，将它在链上的兄弟结点接在它下面即可，每次操作最多使得 d 加一。

AT_agc_030_c

给定 $K \in [1, 1000]$ ，要求填一个 $N \times N$ 个格子的图 $N \leq 500$ ，其中 N 是自己选定的，要求满足：

- 对于每个 $i \in [1, K]$ ，至少有一个格子中填的是 i
- 每个格子都必须填数，并且填的数必须在 $[1, K]$
- 与一个格子相邻的格子为它上下左右四个方向的四个格子（若格子在左边界，则它左边的格子为整个图最右边的那个格子，上下和右方向类似，即在模意义下的加减一）若两个格子 (A, B) 中填的数字相同，那么对每个值 $V \in [1, K]$ ， v 在 A 相邻的格子中的出现次数与在 B 相邻的格子中的出现次数相同。

Solution: AT_agc_030_c

首先, 当 $K \leq 500$ 时, 显然可以让第 i 行都填数字 i , 即填成这个样子:

1	1	1	1	...	1
2	2	2	2	...	2
\vdots	\vdots	\vdots	\vdots	\ddots	\vdots
K	K	K	K	...	K

看起来就很合法. 那如果 $K > 500$ 呢? 这样的排布实在太浪费了 (一行就只有一种数), 那有没有一种方式能够一行 (列、斜) 填两个数呢?

我们斜着来, 这样也是合法的:

1	2	3	4
2	3	4	1
3	4	1	2
4	1	2	3

容易验证, 斜着来, 隔一个数把一种数换成另一种数也是合法的:

5	2	3	4
2	3	4	1
3	4	5	2
4	1	2	3

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

5	2	7	4
6	3	4	1
7	4	5	2
4	1	6	3

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

这样, 一个 $N \times N$ 的网格就能有 $2N$ 种数字了, 按这种方案构造即可.

AT_agc030_d

给定一个长度为 N 的序列 A 和 Q 个操作, 每个操作形如交换 A_l 和 A_r , 对于每个操作, 你可以依次选择是否执行. 询问对于所有 2^Q 中执行方案, 最终形成的序列的逆序对数目之和模大质数的结果.

$N, Q \leq 5000$.

Solution: AT_agc030_d

应该算是常见套路了吧, 我们只需要算出最终逆序对数目的期望 E , 再乘 2^Q 就是逆序对数目的和了, 又由期望的线性性, 令 $f_{l,r}$ 表示 $A_l > A_r$ 的概率, 考虑完这 Q 次操作后, E 就等于

$$\sum_{l=1}^n \sum_{r=l+1}^n f_{l,r}$$

对于交换 A_l 和 A_r 的操作, 实际上就是形如

$$f_{i,l} = f_{i,r} = \frac{f_{i,l} + f_{i,r}}{2}, i \neq l, i \neq r$$

$$f_{l,i} = f_{r,i} = \frac{f_{l,i} + f_{r,i}}{2}, i \neq l, i \neq r$$

$$f_{l,r} = f_{r,l} = \frac{f_{l,r} + f_{r,l}}{2}$$

初始化 f 是 $\mathcal{O}(n^2)$ 的, 每次转移是 $\mathcal{O}(n)$ 的, 共有 q 次转移, 因此时间复杂度为 $\mathcal{O}(n^2 + nq)$.

CF1630F: Making It Bipartite

图上有 n 个点, 点有点权 a_i , 点权互不相同, 如果两个点点权成倍数关系, 那么他们有权. 现在希望删去一些点, 来使图变成二分图, 求出最少删去多少点.

$n, a_i \leq 5 \times 10^4$.

Solution: CF1630F: Making It Bipartite

由于点权互不相同，所以实际的边很少，是 $\mathcal{O}(n \log n)$ 的，我们将无向边变为有向边，若 $a_i \mid a_j$ ，那么从 i 到 j 连一条有向边，这是一个偏序关系，最终的图是二分图的充分必要条件是 **每个点要么只有入度，要么只有出度**。考虑反证，假设一个点既有入度又有出度，那么入度的起点到出度的终点必然也有边，于是出现了一个长度为 3 的环；反之，假如没有既有入度又有出度的点，这个图一定是二分图。

套路地考虑拆点，将每个点拆为 u_1 和 u_2 表示“只有入度”和“只有出度”，将矛盾的关系连成**无向边**：

- 对每个 u_1 和 u_2 之间连边，因为一个点不可能同时满足“只有入度”和“只有出度”
- 接下来考虑一条原本的有向边 $i \rightarrow j$
- 在 i_1 和 j_1 之间连边，因为 i 有入度， j 也有入度说明 i, j 都没有被删，于是 i 既有入度（条件给定），又有出度（连向 j ），不合法
- 在 i_1 和 j_2 之间连边，同理
- 在 i_2 和 j_2 之间连边，同理
- 实际上就是除了“ i 只有出度”（ i_2 ）和“ j 只有入度”（ j_1 ）这种情况之外的所有情况都是矛盾的。

这样连完边建好新图后，要求的就是这个无向图的最大独立集（因为边表示矛盾），为了解决这个问题，我们将边定向为 DAG，由 Dilworth 定理，答案就是 $n -$ 最小链覆盖。

对于定向，只需要将 u_1, u_2 的边定为 u_1 连向 u_2 ，将 i 到 j 的边统一定为 i_- 连向 j_- 即可。

CF1368E: Ski Accidents

有一个由 n 个点 m 条边组成的有向无环图，每个点出度至多为 2。您需要标记一些点（不超过 $\frac{4}{7}n$ 个）。标记一个点 u 将会删除所有与 u 连接的边。

您需要找到一种标记点的方案，使得删边后的图中每一条路径至多有一条边。保证所有数据有合法的答案。 $1 \leq n \leq 2 \times 10^5$ 。

Solution: CF1368E: Ski Accidents

神仙构造。

先考虑为什么限制是 $\frac{4}{7}n$ ，想到把所有 n 个点不重不漏地分为三个集合 A, B, C ，满足 $|C| \leq 2|B| \leq 4|A|$ ，于是 $|C| \leq \frac{4}{7}n$ ，这启发我们就按照这样做下去。

怎么分类呢？这里是人类智慧的一步：

- 点 u 在 A 中当且仅当 u 的入度为 0 或所有入边都来自 C
- 点 u 在 B 中当且仅当 u 至少有一条入边来自 A 且没有来自 B 的入边
- 点 u 在 C 中当且仅当 u 至少有一条入边来自 B

由于每个点的出度至多为 2，所以 $|B| \leq 2|A|$ 且 $|C| \leq 2|B|$ ，跑一遍拓扑就能将所有点分到它应在的集合中，现在的任务就是验证删去 C 中的所有点的边后能否满足要求。

这是很显然的，我们删掉这些边后，剩下的所有边中没有从 B 到 B 的（定义），也没有从 B 到 A 的（ A 中点的入边都来自 C ），所以只剩下了 A 到 B 的边，满足条件。

AT_abc295_e

给你一个长度为 N 的数列 A_1, A_2, \dots, A_N 和两个数 M, K ，保证 $0 \leq A_i \leq M$ ，现在，我们独立地将每个 $A_i = 0$ 随机改为 $[1, M]$ 中的一个整数，之后升序排列 A ，求排序后 A_K 的期望。

$K \leq N \leq 2000, M \leq 2000$

Solution: AT_abc295_e

本题最核心的是一步转化：

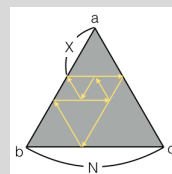
$$E(A_K) = \sum_{i=1}^M i \times P(A_K = i) = \sum_{i=1}^M P(A_K \geq i)$$

而每个 $P(A_K \geq i)$ 是好求的，先选出来一些填小于 i 的位置，同时保证小于 i 的数的个数小于 K ，剩下的再填大于等于 i 的值，乘起来即可。

AT_agc001_b

三个长为 N 的镜子构成正三角形 $\triangle ABC$ ，现有一点 P 在 AB 上，同时 $|AP| = X$ ，从 P 点出发，沿 BC 方向发出一道光，光会不断地反射，同时，我们规定，光已经经过的路径也会变成镜子，问光经过的路径长度是多少。

$$2 \leq N \leq 10^{12}, 1 \leq X \leq N-1, N \in \mathbb{Z}, X \in \mathbb{Z}$$



Solution: AT_agc001_b

第一道光和第二道光的长度和是 N ，我们先把它加上，之后容易发现，光总是在一个平行四边形中反弹若干次，变成一个更小的平行四边形，直到某一边长是另一边长的倍数。于是，我们设 $f(x, y)$ 为在相邻边边长分别为 x, y 的平行四边形中，从某一顶点出发，走过的路径长度，于是有

$$f(x, y) = \begin{cases} 2 \times \lfloor \frac{n}{m} \rfloor m + f(m, n \bmod m), & m \nmid n \\ 2 \times n - m, & m \mid n \end{cases}$$

于是答案就等于 $N + f(X, N - X)$ ，事实上，答案的封闭形式为 $3 \times (N - \gcd(N, X))$

AT_agc001_d

现有正整数数列 A, B ，满足 $\sum A_i = N, \sum B_i = N$ 。

同时，满足以下两个条件的长度为 N 的数列，所有元素必定是相同的。

- 最开始的 A_1 个字，接下来的 A_2 个字，更后面的 A_3 个，等等都是回文。
- 最开始的 B_1 个字，接下来的 B_2 个字，更后面的 B_3 个，等等都是回文。

但是，你遗忘了 A, B 的具体值，只记得 A 是另一个长度为 M 的序列 A' 的排列，请你构造出满足条件的 A, B 。

如果存在解，输出三行，第一行数列 A ，第二行 B 的长度，第三行数列 B ，否则输出 *Impossible*。

$$1 \leq N \leq 10^5, 1 \leq M \leq 100, 1 \leq A'_i \leq 10^5, \sum A'_i = N$$

Solution: AT_agc001_d

先来看一个特例： $M = 1$ 。此时， A' 只有一个元素，我们可以构造 $B_0 = 1, B_1 = N - 1$ 即可满足条件。

为什么我们会想到这样的构造呢？因为当 $M = 1$ 时，题中的限制等价于：给定一个长度为 N 的回文串，假如它最开始的 B_1 个字，接下来的 B_2 个字，更后面的 B_3 个，等等都是回文，那么这个串中的每一个字符都必然相等。而又由于串本身回文，即 $S_1 = S_N, S_2 = S_{N-1} \dots$ ，我们只需要再让 $S_2 = S_N, S_3 = S_{N-1}$ ，这两串等号联立起来，就有 $S_1 = S_N = S_2 = S_{N-1} = S_3 \dots$ 了。

回到一般的情况中，即 $M > 1$ ，我们先证明一个引理。

引理：存在这样的数列 A, B ，当且仅当 A' 中奇数的数量不超过 2。

证明：我们对已经确定相等的两个字符之间连边，即每个回文串的第一个元素和最后一个元素、第二个元素和倒数第二个元素，... 之间连边，那么，串中任意两个字符均相等，当且仅当所有字符之间连通：这是显然的。

一个长度为 L 的回文串，连边的数量为 $\lfloor \frac{L}{2} \rfloor$ ，为了让所有字符之间连通，我们至少需要连 $N - 1$ 条边，于是就有， A 连边的数量 + B 连边的数量应不小于 $N - 1$ ，即：

$$\left(\sum \lfloor \frac{A'_i}{2} \rfloor \right) + \lfloor \frac{N}{2} \rfloor \geq N - 1$$

也即：

$$\left(\sum \frac{A'_i}{2} \right) - \left(\sum_i \frac{[A'_i \text{ odd}]}{2} \right) + \lfloor \frac{N}{2} \rfloor \geq N - 1$$

$$\frac{N - \sum_i [A'_i \text{ odd}]}{2} \geq \frac{N - 2}{2}$$

于是我们就得到了引理一的必要性，下面我们通过构造这样的 A, B 来得到充分性的证明。

首先，假如有一个 A'_i 是奇数，我们把它放到 A 的开头，否则假如有两个，我们把一个放到开头，一个放到结尾，偶数在剩下的位置（即中间）随意排列就得到了 A ，接着我们构造 B 。

首先我们令 $B_1 = A_1 + 1$ ，这一定可以做到（因为已经排除了 $M = 1$ 的情况），接着，对于 $2 \leq i \leq M - 1$ ，令 $B_i = A_i$ ，最后， $B_M = A_M - 1$ （这里需要注意的是，假如 $A_M = 1$ ，那么 B 就只需要 $M - 1$ 个元素）。容易证明构造成立。

于是按照这个构造方法做即可。

AT_agc001_f

给出一个元素集合为 $\{1, 2, \dots, N\}$ ($1 \leq N \leq 500,000$) 的排列 P ，当有 i, j ($1 \leq i < j \leq N$) 满足 $j - i \geq K$ ($1 \leq K \leq N - 1$) 且 $|P_i - P_j| = 1$ 时，可以交换 P_i 和 P_j 。
求：可能排列中字典序最小的排列。

Solution: AT_agc001_f

首先，我们求出 P 的逆 Q 满足 $Q_{P_i} = i$ ，于是交换操作变为：当有 $i < N$ 满足 $|Q_i - Q_{i+1}| \geq K$ 时，可以交换 Q_i 与 Q_{i+1} ，于是给出引理：

引理：假如下标 i, j 满足 $|Q_i - Q_j| < K$ ，那么无论怎样交换， Q_i 和 Q_j 之间的相对顺序是不变的，同时，满足这些相对顺序的任意 Q 的排列都是可达的。

证明：这是显然的，因为我们的一次交换操作只能交换相邻的两个值。

我们把这个引理放到 P 中再看看：假如 i, j 满足 $|i - j| < K$ ，那么无论怎样交换，假如原本有 $P_i < P_j$ ，那么交换后必然还有 $P_i < P_j$ 。

我们的目标是字典序最小，也就是让 1 尽量靠前，之后在让 2 尽量靠前，依次类推。我们将相对位置的关系连成 $\mathcal{O}(KN)$ 条有向边，拓扑排序（若有 $P_i < P_j$ ，则从 j 向 i 连有向边，每次优先取堆中最大的值加入到拓扑排序的开头）即可解决，但时间复杂度有点高了。

考虑用数据结构优化，观察边的性质，目前入度为 0 的点必然是那些除去已经被删除了的点外，满足 P_i 为在区间 $(i - K, i + K)$ 的最大值的 i ，而删除一个点等价于把它的值改为 $-\infty$ ，同时，影响到开区间 $(i - K, i + K)$ 的边，即：区间最值，单点修改。线段树维护即可。

AT_agc002_e

桌上有 n 堆糖果，第 i 堆糖果有 A_i 个糖。两人轮流进行下列两个操作中的一个：

- 将当前最大的那堆糖果全部吃完
- 将每堆糖果吃掉一个

吃完的人输，问谁有必胜策略？

$$1 \leq n \leq 10^5, 1 \leq a_i \leq 10^9$$

Solution: AT_agc002_e

首先我们将 A 降序排列, 变成笛卡尔坐标系上的 n 个点 (i, A_i) , 取出从这些点向 x 轴、 y 轴作垂线, 构成的 n 的矩形的并. 问题化为: 最开始在点 $(0, 0)$, 每人每次可以选择向右走一个单位 (把当前最大的那堆糖果吃掉) 或者向上走一个单位 (将每堆糖果吃一个), 若当前所在点到了这 n 的矩形并组成的多边形的边界 (不含坐标轴) 时, 该步操作的执行者输.

什么时候后手必胜呢? 我们考虑走一个正方形 (假如先手一步向上走了, 我们就向右, 否则相反), 这样一定会到一个边界点 p, p 满足 $A_p > p, A_{p+1} \leq p + 1$, 假如这个点是必败点, 后手就可以必胜了. 反之, 若这个点是必胜点, 先手一定有办法走到它的上方或右方 (先走一步, 之后按后手相反的来), 使后手必败. 于是先后手的胜负就转化为这个点的状态了.

这个点的胜负状态也是容易得到的: 根据这个点到它上方的边界距离的奇偶性和到它右方边界距离的奇偶性就可以得到了, 细节留给读者思考.

AT_agc002_f

给你 n 种颜色的球, 每种颜色的球有 k 个, 把这 $n \times k$ 个球排成一排, 把每一种颜色的最左边出现的球涂成白色 (初始球不包含白色), 求有多少种不同的颜色序列, 答案对 $10^9 + 7$ 取模.

$1 \leq n, k \leq 2000$.

Solution: AT_agc002_f

首先注意到最终一定是 n 个白球和 n 组 $k - 1$ 个相同颜色的球, 不妨就这样将所有球分成两类.

设 $f_{i,j}$ 表示已经放了 i 个白球和 j 组 $k - 1$ 个相同颜色的球 (这些颜色的白色球都一定在 i 中包含了, 即已经被放进去, 故 $i \geq j$ 才是合法的状态) 的方案数, 为了不重不漏地计数, 我们规定每次放球时, 剩下的空位中的第一个空位必须要放此次的球, 初始值为 $f_{i,0} = 1, i \in [1, n]$. 转移也是容易的, 假如放白球, 直接转移过来即可. 假如放有色的一组球, 先枚举颜色, 再枚举空位即可.

$$f_{i,j} = f_{i-1,j} + (n - j + 1) \times \binom{n \times k - i - (j - 1) \times (k - 1) - 1}{k - 2} \times f_{i,j-1}$$

答案即为 $f_{n,n}$, 注意特判 $k = 1$ 的情况.

AT_agc003_d

给定 n 个数 s_i , 要求从中选出最多的数, 满足任意两个数之积都不是完全立方数.
 $n \leq 10^5, s_i \leq 10^{10}$.

Solution: AT_agc003_d

为了方便, 记 $M = 10^{10}$ 首先把 $[1, \sqrt[3]{M}]$ 的素数筛出来, 把 s_i 的每个质因子次数减到 0, 1, 2. 此时, 若原本的 s_i 就是一个完全立方数, 操作过后 s_i 一定会变成 1, 而对于这些数, 最终选出来的数里只可以有一个 (它乘一个不是完全立方数的数一定不是完全立方, 而它乘一个完全立方数一定是完全立方), 所以把最终答案加一, 将所有 1 从 s 中删去.

现在, 我们记 s'_i 为 s_i 删去所有不大于 $\sqrt[3]{M}$ 的质因子后的结果, 将所有数分类. 在之后的操作里, 我们开两个映射 (map), 其中, 映射 $Inv(x)$ 表示使得 $x \times y$ 是完全立方数的最小 y , $Siz(x)$ 表示一个可重集中, 数 x 出现的次数, 同时注意, 我们可以在对 s_i 分解质因数时顺便找到 $Inv(s_i)$

- 若 $s'_i = 1$, 说明它的所有质因子均不大于 $\sqrt[3]{M}$, 记录下 $Inv(s_i)$, 让 $Siz(s_i)$ 加一.
- 若 s'_i 是一个在 $[\sqrt[3]{M}, \sqrt{M}]$ 的质数 (这可以提前筛出质数后二分找到), 记录下 $Inv(s_i)$, 让 $Siz(s_i)$ 加一.
- 若 s'_i 是一个在 $[\sqrt[3]{M}, \sqrt{M}]$ 的质数 p 的平方 (这可以开根号后在质数表中二分判断), 记录下 $Inv(s_i)$, 让 $Siz(s_i)$ 加一.
- 否则有两种情况, 第一种是: s_i 有一个大于 \sqrt{M} 的质数, 那么, 不可能有 s_j 使得 $s_j \times s_i$ 是一个完全立方数 (因为这个质数的平方已经比 M 要大), 我们可以直接让答案加一.

- 第二种是： s_i 有两个不同的质因子 $\sqrt[3]{M} \leq p, q \leq \sqrt{M}$ ，我们仍然可以直接让答案加一，原因类似。于是这两种情况等价。

最后，考察所有在 Siz 中的元素，假如 $Siz(x) \geq Siz(Inv(x))$ ，那么我们就将所有 x 放进答案集合中，让答案加 $Siz(x)$ ，否则将所有 $Inv(x)$ 放进答案集合中，让答案加 $Siz(Inv(x))$ 。

AT_agc003_e

一串数，初始为 $1 \sim n$ ，现在给 Q 个操作，每次操作把数组长度变为 q_i ，新增的数为上一个操作后的数组的重复。问 Q 次操作后 $1 \sim n$ 每个数出现了多少次。

Solution: AT_agc003_e

首先一个显然的引理是：若 $q_i < q_j$ ，同时 $i > j$ ，那么 q_j 可以直接丢掉，这是显然的。于是，我们先在开头插入 $Q_1 = n$ ，然后从前往后遍历 q ，塞到 Q 的末尾，同时用单调栈不断弹出末尾元素，让 Q 中元素单调上升。

考虑从 Q_{i-1} 到 Q_i ，必然是先整段复制了 $\lfloor \frac{Q_i}{Q_{i-1}} \rfloor$ 次，再复制了一小段，整段的贡献是好维护的，对于小段的影响，我们递归下去考虑。

总的来说，我们从后往前遍历 Q ，写一个递归函数考虑最后那一小段的影响，每次递归，二分出一个位置，让这一小段拆成若干整段再加上更小的一段，直到拆完。维护可以用差分。

AT_agc003_f

有一个 H 行 W 列的网格，每个单元格都是黑色或白色。所有黑色单元格都是四联通的，也就是说，只做水平或垂直移动且只经过黑色单元格即可从任何黑色单元格移动到任何其他黑色单元格。

第 i 行第 j 列 ($1 \leq i \leq H, 1 \leq j \leq W$) 的单元格的颜色由字符 s_{ij} 表示。如果 s_{ij} 是 #，该单元格为黑色；如果 s_{ij} 是 .，该单元格为白色。至少一个单元格是黑色的。

我们定义「 n 级分形」如下：0 级分形是一个 1×1 的黑色单元格。 n 级分形由 $n - 1$ 级分形变化而来。具体地，将 $n - 1$ 级分形的每一个黑色单元格替换为 **Snuke** 的网格，每一个白色单元格替换为与 **Snuke** 的网格尺寸相同的全部为白色的网格，就成了 n 级分形。

Snuke 喜欢连通块。现在他想告诉你， K 级分形中，有多少个黑色格子组成的四联通块？答案对 $10^9 + 7$ 取模。

Solution: AT_agc003_f

记网格中黑色单元格的数量为 c 。首先，我们定义一个网格“水平连通”当且仅当存在一行使得该行的第一列和最后一列的这两个单元格均是黑色，而“竖直连通”当且仅当存在一列使得该列的第一行和最后一行的这两个单元格均是黑色。

当这个网格既“水平连通”又“竖直连通”时，不难发现，所有 K 级连通的黑色单元格均是连通的，答案为 1。当这个网格既不“水平连通”又不“竖直连通”时，每次分形出的 c 个小图形都是不连通的，答案为 c^{K-1} 。不失一般性，接下来以“水平连通”但不“竖直连通”为例。

容易发现，连通块个数就等于“黑格个数 - 水平相邻的两个黑格子的对数”，于是答案即为 $K - 1$ 级分形图中黑格的个数 - 左右相邻两个格子都是黑色的对数。

黑格个数显然是 c^{K-1} ，而后者比较难办。继续找性质，在每个块中都会有原图的相邻黑格的贡献，并且在边界上会多出一些来，多出的这些就是相邻黑格个数 \times 两个 **Snuke** 网格左右拼接后边界上多出来的水平相邻黑格对数。

于是，设 a 为水平相邻黑格个数， b 为左右拼接后边界产生的新黑格对的个数， c 表示黑格数量，那么可以写成矩阵的形式：

$$\begin{bmatrix} c & a \\ 0 & b \end{bmatrix}^{K-1}$$

答案即为最终矩阵 A 中的 $A_{1,1} - A_{1,2}$.

AT_agc004_e

有一个 $H \times W$ 的棋盘 ($H, W \leq 100$), 上面要么是空的, 要么有一个机器人, 要么是一个出口 (整个地图只有一个出口). 每次可以命令所有机器人向上下左右中的某个方向移动一格, 如果它超出了棋盘的边界就会消失. 如果它到了出口的位置就会被你救下 (并且从棋盘上消失). 求你能够救下的机器人的最大值.

Solution: AT_agc004_e

移动这一堆机器人不好处理, 我们将操作转换一下: 出口带着一个矩形在上下左右移动, 机器人一旦出过这个矩形就会消失, 问能救下的机器人的最大值.

很显然的 dp, 设 $f_{l,r,u,d}$ 是出口最多左移多 l , 右移过 r , 上移过 u , 下移过 d , 能救下的机器人数量的最大值, 转移时的边界比较麻烦. 答案即为所有 f 的最大值.

AT_agc004_f

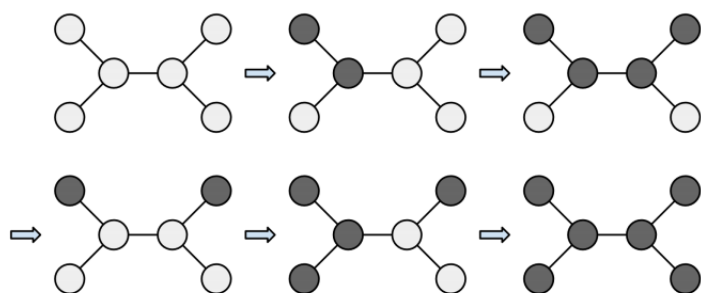
给定一个 N 个点, M 条边的图, 没有自环, 没有重边, 保证连通. 其中 $N-1 \leq M \leq N$, 每个点初始是白色. 每次操作可以处理一条边, 其两个点如果颜色相同则都变成相反的颜色 (黑变白, 白变黑). 询问能否将每个点都变为黑色. 如果能, 输出最少的操作数; 如果不能, 输出 -1 .

$2 \leq N \leq 10^5, N-1 \leq M \leq N$.

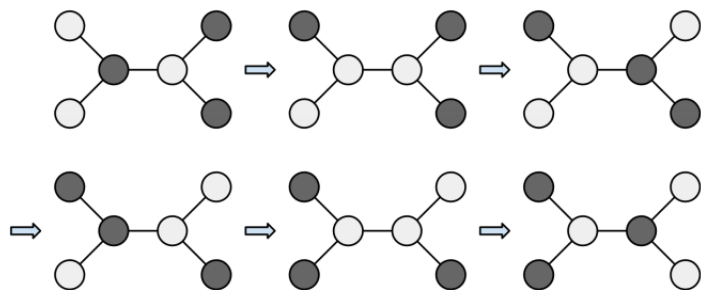
Solution: AT_agc004_f

数据范围给出的 M 让我们确定了这是一棵**树或基环树**, 我们先来考虑这是一棵树的情况.

此时, 我们将所有点黑白染色, 那么一次操作等价于交换相邻两个不同色的点的颜色, 又等价于将一个黑色点移动到相邻的白色空位上, 目标状态为: 所有点的颜色取反, 即原来是空位的点现在有黑色, 原来是黑色的点现在是白色空位, 这里给出官方题解中的例子:



Before the transformation



After the transformation

将黑点的权值设为 1, 白点的权值设为 -1 , 对以每个点为根的子树分别考虑, 设 dp_u 是以 u 为根的子树的权值和, 答案就是 $\sum_u |dp_u|$.

为什么呢? 假如 $dp_u < 0$, 说明子树中的空位要多于黑点数, 此时我们至少需要从 u 到 u 的父亲那条边拿来 $|dp_u|$ 个黑点并塞到子树中, 这至少需要 $|dp_u|$ 次操作, 而 $dp_u > 0$ 时类似.

可以证明, 这个下界是能够取到的. 而当 $dp_{root} \neq 0$ 时, 说明无解. 现在考虑基环树的情况.

我们分两类, 第一类为这个环中有偶数个点, 第二类为这个环中有奇数个点.

当环中有偶数个点时, 这个图仍然是二分图, 我们可以沿用树的做法: 断开环上的一个边, 将根设为环上的某一点, 做树形 dp 求出所有的 dp_u . 同理, 若 $dp_{root} \neq 0$, 仍然无解, 那这样看来, 断开的那条边的作用就是减少某些边的操作次数了. 设这条断边的操作次数为 x (正负表示方向, 绝对值表示次数), 那么, 环上在这条边右边的那些点, 它们对答案的贡献变为 $|dp_u - x|$, 对于左边的点, 贡献变为 $|dp_u + x|$, 于是答案是数轴上一堆点到某个点距离和的最小值, 可以用中位数解决.

当环上有奇数个点时, 这条边连接的两个点同色, 在这条边上操作可以使黑点个数减二或白点个数加二, 于是只需要 $dp_{root} \equiv 0 \pmod{2}$ 即可有解, 更新一下环上的 dp 值即可.

AT_agc005_d

如果一个排列 P 满足对于所有的 i 都有 $|P_i - i| \neq k$, 则称排列 P 为合法的. 现给出 n 和 k , 求有多少种合法的排列.

由于答案很大, 请输出答案对 924844033 取模的结果.

$2 \leq n \leq 2 \times 10^3, 1 \leq k \leq n - 1$.

Solution: AT_agc005_d

直接求是不容易的, 于是考虑反演. 设 f_i 为至少有 i 个位置 j 满足 $|P_j - j| = k$ 的排列数, 答案即为

$$\sum_{i=0}^n (-1)^i f_i$$

问题化为求 f_i .

将每个位置 A_i 排成一列, 每个数 i 排成一列, 若 A_i 不能填 j , 那么就连上这两个结点, 我们得到了一个二分图.

对于每个模 k 的同余类, 它们之间是互不干扰的, 于是我们有了一个多项式的思路: 将每个模 k 的同一类的方案写成 OGF 后卷积即可. 注意到钦定 i 个位置不合法, 等价于在图中选出 i 条互不相邻的边. 这里有一个结论是: 在 i 条边的链上选出 j 个互不相邻的边的方案数是 $\binom{i-j+1}{j}$, 于是就可以得出来了.

注意最后求出来的系数还应乘上 $(n-i)!$ 表示选出来剩下的那些位置, 这样才是 f_i .

第二种方式是 dp, 设 $dp_{i,j,1/0}$ 为前 i 个结点选了 j 个边, i 和 $i-1$ 之间是否连了边的方案数, 有转移:

$$dp_{i,j,0} = dp_{i-1,j,0} + dp_{i-1,j,1}$$

$$dp_{i,j,1} = dp_{i-1,j-1,0}$$

实现时, 可以把所有链首尾相连, 标记一下每个链的链头后转移, 则 $f_i = (n-i)! \times dp_{2n,i}$.

AT_agc005_e

现在 A 和 B 在玩游戏, 游戏是在两棵树上进行的, A 在树 a 上的点 x , B 在树 b 上的点 y , 两棵树上的点的编号是相同的, 只是连边方式不同.

对于奇数轮, A 可以选择走到它当前在树 a 上的点的相邻节点, 或者在原地不动, 对于偶数轮则是 B 进行选择, 当两个人到达编号相同的点时, 游戏结束.

现在 A 想最大化游戏轮数, B 想最小化游戏轮数. 问游戏的轮数, 如果可以进行无限轮游戏, 请输出 -1 .

$N \leq 2 \times 10^5$

Solution: AT_agc005_e

问题看起来就很棘手，我们先重新理解一下题意。

有两个树，记为红树和蓝树，A 在红树上操作，B 在蓝树上操作，每次可以沿着树边走，走到编号相同的点时，游戏结束。A 要最大化轮数，B 要最小化轮数。

记 $d_r(u, v)$ 为在红树上 u, v 两点间的距离， $d_b(u, v)$ 为在蓝树上 u, v 两点间的距离。假如存在这样的一条边 (u, v) 使得 $d_b(u, v) \geq 3$ ，A 走到了 u, v 中的一个点，并且在这之前没有被 B 抓到，那么 A 就可以在 u, v 上不断地反复横跳来让 B 永远抓不到它，设这样的点为关键点，此时应输出 -1 。

于是，A 的目标就是跳到这样的点上，B 的目标就是抓到 A。我们让蓝树以 y 为根，红树以 x 为根，A 的操作在蓝树上看，其实是不断地跳一个或两个边或不跳（如果一次能跳三个或更多边，说明我们已经到了关键点），于是，B 就可以保证 A 一直在它的子树中（因为最多跳两个边，如果跳出子树，B 就可以马上往上跳一步来抓到 A，这样和不动是等价的），B 的策略就是不断往 A 所在的那个子树中走，那么 A 能到某个点 u 当且仅当 $d_r(u, x) < d_b(u, y)$ ，游戏可以无限进行下去当且仅当这些可达点中有关键点。

假如这些点中没有关键点呢？这时，A 的策略一定是去到这些点中 $d_b(u, y)$ 最大的点 u ，然后原地等待，答案即为 $2 \times \max_u \{d_b(u, y)\}$ 。

AT_agc005_f

给定一棵无根树，定义 $f(i)$ ，对于所有大小为 i 的点集，求出能够包含它的最小连通块大小之和。对于 $i = 1 \rightarrow n$ 的所有 i ，求出 $f(i)$ 模 924844033 的结果。

$n \leq 2 \times 10^5$

Solution: AT_agc005_f

正向枚举连通块一看就很不可做，我们考虑每个点对每个 $f(i)$ 的贡献。首先，所有大小为 i 的点集共有 $\binom{n}{i}$ 个，这其中有一些点集，满足包含它的最小连通块不含 u ，我们需要把 u 对这些的贡献删去。

包含它的最小连通块不含 u 只有两种情况：所有 i 个点都在以 u 为根的树的某一子树中，于是：

$$f(i) = \sum_{u=1}^n \left(\binom{n}{i} - \sum_{v=son_u} \binom{siz_v}{i} \right)$$

其中， son_u 表示 u 的儿子（注意这里的整棵树的根是 u ）， siz_v 表示 v 为根的子树大小。这个式子可以接着化简为

$$f(i) = n \times \binom{n}{i} - \sum_{u=1}^n \sum_{v=son_u} \binom{siz_v}{i}$$

设 cnt_j 表示大小为 j 的子树数量，于是

$$\begin{aligned} f(i) &= n \times \binom{n}{i} - \sum_{j=i}^n cnt_j \binom{j}{i} \\ &= n \times \binom{n}{i} - \sum_{j=0}^{n-i} cnt_{j+i} \frac{(j+i)!}{j!i!} \\ &= n \times \binom{n}{i} - \frac{1}{i!} \sum_{j=0}^{n-i} \frac{cnt_{j+i}(j+i)!}{j!} \end{aligned}$$

设 $A_i = cnt_i \times i!$ ， $B_i = \frac{1}{i!}$ ，于是原式又化为

$$n \times \binom{n}{i} - \frac{1}{i!} \sum_{j=0}^{n-i} A_{j+i} \times B_j$$

我们希望用卷积加速，于是设 $C_i = A_{n-i}$ ，原式又化为

$$n \times \binom{n}{i} - \frac{1}{i!} \sum_{j=0}^{n-i} C_{n-i-j} \times B_j$$

而

$$\sum_{j=0}^{n-i} C_{n-i-j} \times B_j$$

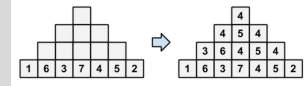
已经是卷积的形式了，NTT 加速即可，时间复杂度 $\mathcal{O}(n \log n)$ 。

AT_agc006_b

给出一个 N 层的方格金字塔，自顶向下依次标号为第 1 到第 N 层，其中第 $i (1 \leq i \leq N)$ 层有 $2i - 1$ 个方格。

第 N 层有一个 1 到 $2N - 1$ 的排列，其他层的数字按以下规则生成：方格 b 中填写的整数，是方格 b 正下方，左下方和右下方方格中所写整数的中位数。现在请你构造出一组第 N 层的数字，使得求得的第一层的数字为 X ，无解输出 No。

$2 \leq N \leq 10^5$, $1 \leq X \leq 2N - 1$



Solution: AT_agc006_b

首先，当 $X = 1$ 或 $X = 2N - 1$ 时必然无解：这是很显然的，否则我们尝试构造。

假如某一层出现了这样的情况： $a \ a \ y$ ，那么，这两个 a 的上方也一定是 a ，因为两个 a 和其他任意一个数的中位数总是 a 。这启发我们让某一层的中间出现这种情况，同时 $a = X$ ，问题就解决了。

这是好办的，当 $N \geq 3$ 时，我们只需要放 $x-2 \ x-1 \ x \ x+1$ 或 $x-1 \ x \ x+1 \ x+2$ 就行，这两种中总有一种是可行的。当 $N = 3$ 且 $X \neq 1, X \neq 3$ 时， $1 \ 2 \ 3$ 就是解。

AT_agc006_c

数轴上有 n 只兔子，编号为 i 的兔子位于 x_i ，现有一个长度为 m 的操作序列 A ，我们依次进行操作 $A_1, A_2, A_3, \dots, A_m$ ，其中操作 A_i 表示将编号为 A_i 的兔子等概率地以兔子 $A_i - 1$ 或兔子 $A_i + 1$ 为中心对称到另一侧，这意味着有 50% 的概率，兔子 A_i 跳到以兔子 $A_i - 1$ 的位置为中心的对称点，剩下 50% 的概率是跳到以兔子 $A_i + 1$ 的位置为中心的对称点。

依次执行完这 m 次操作称为**执行了一轮操作**，询问的是执行了 K 轮操作后，每个兔子所在位置的期望。

$n, m, x_i \leq 10^5$, $1 < A_i < n$, $K \leq 10^{18}$

Solution: AT_agc006_c

我们先看执行一个操作后发生了什么：以 o 为中心，点 x 的对称点就是 $o + (o - x) = 2o - x$ ，比如，以 1 为对称中心，3 的对称点就是 $2 \times 1 - 3 = -1$ ，于是，我们设兔子 i 所在位置的期望是 f_i ，那么对兔子 i 执行一次操作后，期望变为 $\frac{1}{2} (2f_{i-1} - f_i) + \frac{1}{2} (2f_{i+1} - f_i) = f_{i-1} - f_i + f_{i+1}$ 。

接下来是构造性的一步：我们考虑这次操作对 f 的差分 $d_i = f_i - f_{i-1}$ 的影响。将 f_i 变为 $f_{i-1} - f_i + f_{i+1}$ 后，在 d 中的表现为**交换了** d_i 和 d_{i+1} ！于是一轮操作等价于一个置换，而多次重复的置换是可以用类似快速幂的算法解决的。

时间复杂度 $\mathcal{O}(n \log K)$ 。

还可以考虑置换环，就得到了一个与 K 无关的线性做法。

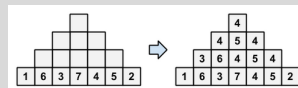
AT_agc006_d

给出一个 N 层的方格金字塔，自顶向下依次标号为第 1 到第 N 层，其中第 i ($1 \leq i \leq N$) 层有 $2i - 1$ 个方格。

第 N 层有一个 1 到 $2N - 1$ 的排列，其他层的数字按以下规则生成：方格 b 中填写的整数，是方格 b 正下方，左下方和右下方方格中所写整数的中位数。

给出最后一层数字，求第一层的那个数字是多少。

$2 \leq N \leq 10^5$



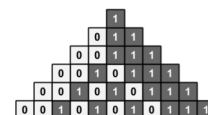
Solution: AT_agc006_d

首先二分答案，每次判断答案是否大于等于 mid 。现在只需 $\mathcal{O}(n)$ 判断即可。

首先将所有大于等于 mid 的值看作 1，小于 mid 的值看作 0，那就只需判断顶层是否为 1 即可。不难发现，两个 0 一个 1 会得到 0，而两个 1 一个 0 会得到 1，若干个连着的相同数字会一直传递到顶，如下图：

于是，我们只需要看距离中心最近的大于等于 2 个相邻同色块是 1 还是 0 即可，因为它距离中心最近，所以会最先传递到顶，答案也就是这个值了。需要特殊判断的是只有 01 交替的情况，此时只需看 N 的奇偶性和底层中间的数的值即可。

时间复杂度 $\mathcal{O}(n \log n)$ 。



AT_agc006_e



现在有一个 3 行 N 列的初始矩阵， (i, j) 位置的数为 $i + 3j - 3$ 。一次操作是指：选择一个 3×3 的子矩阵，将这个子矩阵旋转 180° 。

现在给出一个 3 行 N 列的矩阵（矩阵中的数各不相同），问能否通过若干次上述操作将初始矩阵变为给定的矩阵。

$5 \leq N \leq 10^5$

Solution: AT_agc006_e

首先排除掉一些很显然的错误情况：由于每次旋转后，每个数字所在列的奇偶性不变，所以如果和目标状态的奇偶性不同，必然无解。其次，注意到原先在同一列的三个数，经过若干次操作后，必然还在同一列，只可能完全不变或第一行与第三行的两个数互换了位置，于是，假如某一列不满足三个数相邻且中间的数是中位数，且第一行或最后一行的数是 3 的倍数，同样无解。

现在，我们将每一列的数编号，1 2 3 是 1 号，4 5 6 是 2 号， \dots ，假如某一列是 3 2 1，即是正常顺序的翻转，我们记作负数，这个例子就是 -1 。

目标状态就可以写作 $1 \ 2 \ 3 \dots N$ 了，初始状态一定是形如 $2 \ -3 \ 4 \dots$ 的这些数，它们的绝对值形成了 1 到 N 的一个排列，记它为数列 A 。

为了叙述方便，我们称 $\text{rotate}(x, x+2)$ 指对第 x 列到第 $x+2$ 列这个 3×3 子矩阵进行旋转操作，而 $\text{rotate}(x, x+2)$ 造成的影响就是：交换了第 A_x 和第 A_{x+2} ，同时让 A_x, A_{x+1}, A_{x+2} 都变成了原来的数的相反数。

引理 1：可以在其他列不发生变化的影响下翻转第 i 列和第 $i+2$ 列 $2 \leq i \leq N-2$ ，即让原本的 $x \ y \ z$ 变为 $-x \ y \ -z$ 。直接给出构造：

1	2	3	4	5	rotate(1,3)
-3	-2	-1	4	5	rotate(3,5)
-3	-2	-5	-4	1	rotate(1,3)
5	2	3	-4	1	rotate(3,5)
5	2	-1	4	-3	rotate(1,3)
1	-2	-5	4	-3	rotate(3,5)
1	-2	3	-4	5	

引理 2: 可以在其他列不发生变化的影响下让四个相邻的数变成原来数的相反数，仍然构造：

1	2	3	4	rotate(2,4)
1	-4	-3	-2	rotate(1,3)
3	4	-1	-2	rotate(2,4)
3	2	1	-4	rotate(1,3)
-1	-2	-3	-4	

由引理 1 和引理 2，我们可以对任意 $1 \leq i \leq N-2$ ，同时翻转第 i 列和第 $i+2$ 列，这和引理 1 的差别在于，我们可以同时翻转第 1 列和第 3 列了（先翻转 2,4 列，再用引理 2 翻转 1,2,3,4）。

于是得到**引理 3**：能够对任意奇偶性相同的 i, j ，同时翻转第 i 列和第 j 列（这是因为，我们可以不断旋转子矩阵，把原来的第 i 列移动到第 j 列旁边，翻转这两列后，再把这一列转回去）。

由此得到：假如我们已经用某种方法让 $|A_i| = i$ ，假如奇数列的负数个数是偶数并且偶数列的负数个数也是偶数，就一定有解（不断用引理 3 即可），否则一定无解。问题化为在对 A 整理顺序的过程中维护奇数、偶数列的负数个数奇偶性（分别记为 tot_1 和 tot_0 ）。

考虑记录 $goal(i)$ 表示现在的 A_i 应该到哪一列，即 $goal(i) = |A_i|$ ，再记录 $pos(i)$ 表示最终在第 i 列的数现在在哪一列，即 $|A_{pos(i)}| = i$ ，从 1 到 n 枚举 i ，不断让 $pos(i)$ 回到位置 i ，即直接交换了第 i 列和第 $pos(i)$ 列。注意到在这个过程中， $tot_{i \bmod 2}$ 是不变的， $tot_{(i+1) \bmod 2}$ 变为了相反的值（若原来有奇数个负数，则现在有偶数个，若原来有偶数个负数，则现在有奇数个）。

这是因为，我们交换了第 i 列和第 $pos(i)$ 列，实际上是两轮不断的旋转，每次旋转操作（以 $rotate(i, i+2)$ 举例），由于 $i \equiv i+2 \pmod{2}$ ，所以它们的正负号均变化时，对 $tot_{i \bmod 2}$ 是没有影响的，而 A_{i+1} 的正负号改变，就会使 $tot_{(i+1) \bmod 2}$ 变化。

因此，由于我们旋转的都是那些边界与 i 模 2 同余的子矩阵，故 $tot_{i \bmod 2}$ 无变化，而我们一共旋转了奇数次，所以 $tot_{(i+1) \bmod 2}$ 恰好改变。

维护即可，最后判断 tot_0 和 tot_1 是否均为 0（偶数个负数），时间复杂度 $\mathcal{O}(n)$ 。

AT_agc006_f

有一个 N 行 N 列的矩阵，第 i 行第 j 列的格子表示为 (i, j) 。

开始时，有 M 个格子 (a_i, b_i) ， $1 \leq i \leq M$ 是黑色，其他格子都是白色，作者会按照以下的规则尽可能多的将白色格子涂成黑色：

- 对于整数 $1 \leq x, y, z \leq N$ ，如果 (x, y) 和 (y, z) 都是黑色，那么就把 (z, x) 涂黑。

请计算出当再也没有白色格子能被涂黑时，黑色格子的个数。

$1 \leq N, M \leq 10^5$ ， $1 \leq a_i, b_i \leq N$ ，数据保证各黑格坐标互不相同

Solution: AT_agc006_f

先转化一下题意：平面上有 N 个点，格子 (x, y) 是黑色等价于存在有向边 (x, y) ，操作规则是若边 $(x, y), (y, z)$

同时存在, 则边 (z, x) 也应存在.

首先将图划分为若干个弱连通图, 分别求出答案后相加即可. 对一个弱连通图三染色, 黑点指向蓝点, 蓝点指向红点.

若三染色无解, 即有黑点指向黑点、红点, 或蓝点指向黑点、蓝点, 或红点指向蓝点、红点, **说明这个弱连通图最终一定会变成一个完全图**, 答案为 n^2 , 其中 n 为这个弱连通图的点数.

若可以三染色, 并且只染上了一种或两种颜色, 说明我们绝对无法加上任意一条边, 答案即为初始边数.

若可以三染色, 并且三种颜色在图中都有, 那么, 任意黑点到任意蓝点之间都会被加上边, 任意蓝点到任意红点之间都会被加上边, 任意红点到任意黑点之间都会被加上边, 答案即为 $cnt_{black} \times cnt_{blue} + cnt_{blue} \times cnt_{red} + cnt_{red} \times cnt_{black}$, 其中 cnt_x 指 x 色的点的数量.

证明可以考虑归纳, 不断地加边, 产生的新点与旧点之间必然满足上述关系.

AT_agc007_b

给出一个 1 到 n 的排列 P ($n \leq 2 \times 10^4$), 要求构造两个长度为 n 的数列 A, B 满足:

- $1 < A_i, B_i < 10^9$
- $A_1 < A_2 < A_3 < \dots < A_n, \quad B_1 > B_2 > B_3 > \dots > B_n$
- $A_{P_1} + B_{P_1} < A_{P_2} + B_{P_2} < \dots < A_{P_n} + B_{P_n}$

Solution: AT_agc007_b

首先我们考虑怎么在满足两个单调性的同时让 $A_{P_1} + B_{P_1} = A_{P_2} + B_{P_2} = \dots$. 显然, 直接让 $A_i = i, B_i = n - i$ 即可, 之后我们在让 B_{P_i} 变为 $B_{P_i} + i$, 第三个条件就可以满足了.

但是这样的问题是, 让 A_{P_i} 变为 $A_{P_i} + i$ 后, 原先的单调性可能会被破坏, 解决方法是: 让本来的两个相邻数之间的差充分大 (大于等于 n), 这样的话, 这一步操作对单调性就没有影响了, 具体来说, 我们在一开始时让 $A_i = N \times A_i, B_i = N \times (n - i)$, N 是一个足够大的数, 可以取 2×10^4 .

最后一个问题就是, 我们构造出的数组里可能有元素的值是 1, 让所有 A_i, B_i 再都加上 1 即可.

AT_agc007_c

数轴上有 $2n + 1$ 个点, 第 i 个点和第 $i + 1$ 个点之间的距离为 d_i , 同时 d_i 构成一个首项为 d , 公差为 x 的等差数列. 每次等概率随机选取相邻的两个点, 贡献为这两点之间的距离, 同时删去这两个点, 在剩下的点中接着操作, 直到最终剩下一个点.

询问最终所有贡献的和的期望.

$n \leq 2 \times 10^5$

Solution: AT_agc007_c

可以证明: 等概率选取两个相邻的点之后的期望状态仍然是一个等差数列, 而对于一个首项为 d , 公差为 x 的等差数列, 操作一次它的期望贡献就是

$$d + \frac{(2n-1) \times x}{2}$$

设长度为 $2n$ 的等差数列操作一次后所得期望等差数列的首项为 d' , 公差为 x' , 有

$$d' = d + \frac{2d+5x}{2n}, \quad x' = x + \frac{2x}{n}$$

AT_agc007_d

Shik 君在玩一个游戏，初始时他在数轴的 0 位置，出口在 E 位置，并且数轴上还有 n 只小熊，第 i 只小熊在 x_i 位置。

Shik 君拿着 n 块糖果出发，每走一个单位长度要花费一秒。到一个小熊的位置时，他可以送给这个小熊一块糖果，这个过程不花时间。小熊收到糖果后， T 秒以后会在它所在的位置产生一个金币。

Shik 君想知道，他从出发到收集了所有金币抵达出口，最少要花费多长时间。

$1 \leq N \leq 10^5$, $1 \leq T, E \leq 10^9$, $0 < x_i < E, x_i < x_{i+1}$ ($1 \leq i < N$)

Solution: AT_agc007_d

很明显的 dp 特征，注意到：假如我们在某一时刻选择往左走去收集金币，那么一定会把前面的所有金币都收集掉。这是容易证明的。假如没有被全收集掉，那之后一定还要再收集一次，不如把这次直接省略掉，在下次一起收集。

设 f_i 为已经收集了第 $1 \sim i$ 个熊对应的金币后回到 x_i ，所需时间的最小值，转移则考虑枚举本次收集的终点 j ：

$$f_i = \min_{j=0}^{i-1} \left\{ f_j + (x_i - x_j) + \max \{ 2(x_i - x_{j+1}), T \} \right\}$$

即：先用 $x_i - x_j$ 的时间走到 x_i ，然后，假如我们返回到 x_{j+1} 后（此时距离第一次经过 x_{j+1} 已经过了 $2 \times (x_i - x_{j+1})$ ）仍无法收集到这里的金币，我们就还要再等，否则直接走即可，故应该取 \max 。

答案为 $f_n + (E - x_n)$ ，时间复杂度 $\mathcal{O}(n^2)$ ，考虑优化。

我们不断从 f_n 回退，那么递推式里 $+(x_i - x_j)$ 这一项的总贡献就是 x_n ，所以我们可以把递推式里的这一项删去，答案即为 $f_n + E$ ，此时 f 的递推式为：

$$f_i = \min_{j=0}^{i-1} \left\{ f_j + \max \{ 2(x_i - x_{j+1}), T \} \right\}$$

分类讨论，注意到 x_i 和 f_i 均递增，所以我们只需要开一个队列维护满足 $2(x_i - x_{j+1}) < T$ 的那些 j ，转移时用队头，分两类 $\mathcal{O}(1)$ 转移即可，时间复杂度 $\mathcal{O}(n)$ 。

AT_agc007_e

一颗 n 个节点的二叉树，每个节点要么有两个儿子要么没有儿子。边有边权。

你从 1 号节点出发，走到一个叶子节点。然后每一天，你可以从当前点走到另一个叶子。最后回到 1 号节点，要求到过所有叶子并且每条边经过恰好两次。

每天的路费是你走过的路径上的边权和，你的公司会为你报销大部分路费，除了你旅行中所用路费最高的，行走路线是从叶子到叶子的那一天的路费。

求你自己最少要付多少路费？

$2 < N < 131,072$ ，边的权值满足 $0 \leq v_i \leq 131,072$

Solution: AT_agc007_e

首先，答案可以二分得到，问题转换为：判定是否存在一种方案使得每次的花费不超过 mid 。由于每条边要恰好经过两次，所以，只要我们进入到了一棵子树中，就必须先把这棵子树中的叶子全遍历到。

设 $f_{u,x,y}$ 为进入到以 u 为根的子树中，是否可以做到：进入时费用为 x ，出去时费用为 y ，全程的费用不超过 mid 。转移就是暴力枚举，考虑优化。

用单调性：若存在 $x_1 \leq x_2, y_1 \leq y_2$ ，同时， $f_{u,x_1,y_1}, f_{u,x_2,y_2}$ 这两个状态都是可行的，则 f_{u,x_2,y_2} 这个状态就是无用的（因为完全可以被另一个状态替换）。

因此，要得到根的可行状态，就根据两个子树的状态合并，选定了起点所在的子树和其 x_1, y_1 后，只需要选另一棵子树中，那个使 $y_1 + x_2 + val_{u,1} + val_{u,2} \leq mid$ 的最小的 y_2 ，得到状态 f_{u,x_1,y_2} ，而这可以尺取得

到（合并时，我们让点 u 的所有状态按照 x 第一关键字， y 第二关键字升序排列），每次转移增加的状态数是 $2 \times \min(size_{lc}, size_{rc})$ 的，总状态数是 $\mathcal{O}(n \log n)$ 。

AT_agc007_f

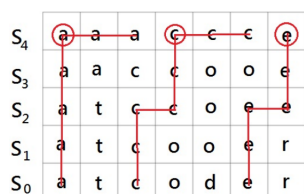
Shikk 的工作是复制。有一天，Shikk 从他的上司那里拿到了一个由小写英文字母组成的长度为 N 的字符串 S_0 （假设这天是第 0 天）。这之后第 i 天的工作是把 S_{i-1} 复制到 S_i 。下文中的 $S_i[j]$ 表示字符串 S_i 的第 j 个字母。

Shikk 还不太习惯这个工作。每天，当 Shikk 从第一个字母开始按顺序复制字符串时，他有可能会写下和刚刚写下的字母相同的字母，而不是本来应该写下的字母。也就是说， $S_i[j]$ 要么与 $S_{i-1}[j]$ 相同，要么与 $S_i[j-1]$ 相同。（特别地，字符串开头的字母不可能出错。也就是说， $S_i[1]$ 必然与 $S_{i-1}[1]$ 相同。）

输入两个字符串 S_0 和 T ，请求出使得 S_i 有可能与 T 相同的最小的整数 i 。如果这样的 i 不存在，请输出 -1。

$N \leq 10^6$

Solution: AT_agc007_f



借一下官方题解中的图，最终一定是一个这样的形式。为了让行数最小，我们倒序看，贪心地让拐点靠右，队列维护这些拐点即可。

具体来说，我们只关注满足 $T_i \neq T_{i-1}$ 的 i ，对于这些 T_i ，找到一个 j 满足 $j \leq i$ 并且 $S_j = T_i$ ，假如找不到，说明无解，否则队列维护拐点的位置，当此处拐不到队头位置时，弹出队头，之后将新拐点所在列加进去。答案是什么呢？就是队列大小的历史最大值再加一。

AT_agc008_b

有 N 个格子排成一列，从左起第 i 个格子中写着整数 A_i 。

开始时，每个格子被涂成白色。sunuke 君将重复进行以下操作：

选择连续的 K 个格子，将它们全部涂成白色或全部涂成黑色。此操作将会覆盖掉格子原来的颜色。

sunuke 君希望在操作完成后，黑色格子中整数的和最大。请求出此最大值。

$1 \leq N \leq 10^5$

Solution: AT_agc008_b

注意到最后一次操作会让一段长度为 K 的格子全变黑或变白，那么 $\mathcal{O}(n)$ 地枚举这一段格子，剩下的格子是可以在这之前随便选颜色的，显然除了这段格子外，我们会选那些正数，这段格子要么全选要么不选，预处理一下即可。

AT_agc008_d

给你一个长度为 N 的整数序列 X ，请判断是否存在一个满足下列条件的整数序列 A ，如果存在，请构造一种方案。条件如下：

- A 的长度为 N^2 ，并且满足数字 $1, 2, 3 \dots N$ 都各出现恰好 N 次。
- 对于 $1 \leq i \leq N$ ，数字 i 在 A 中第 i 次出现的位置是 X_i 。

$1 \leq N \leq 500$

Solution: AT_agc008_d

贪心地去构造，具体来说，每次考虑最小的 X_i ，将这 i 个数字 i 尽量往前填，填不下一定无解，填完这 N 个数字后，再每次考虑最大的 X_i ，将剩下的 $N - i$ 个数字 i 尽量往后填，填不下就无解。

填完之后，我们就得到了一组构造解。

CF1770D

初始时给定 N 以及两个长度为 N 的数列 A, B ，值域为 $[1, n]$. C 是一个任意的，值域为 $[1, n]$ 、长度为 N 的数列。

现在，对于 N 个可重集 $S_i = \{A_i, B_i, C_i\}$ ，先手删掉其中的一个元素，后手再选出剩下两个元素中的一个，这样，后手就得到了另一个长度为 N 的数列。

先手的任务是让这个数列是一个 1 到 N 的排列，后手的任务相反。询问有多少种不同的数列 C 可以使先手赢。

$N \leq 10^5$ ，答案对 998244353 取模。

Solution: CF1770D

一个显然的结论是：我们对每个 S_i ，必须让里面有两个相同的元素，之后，先手会删掉那个不同的元素，让后手的选择固定，否则假如后手有两种选择，先手的目的是达不到的。

于是，假如 $A_i \neq B_i$ ，那么 C_i 就必须等于其中的一个，留下来的是 C_i 和那个等于 C_i 的元素，否则 C_i 可以任意选，留下来的是 $A_i = B_i$ 。

转化成图论问题：对 N 个点 1 到 N ，有 N 条无向边 (A_i, B_i) 。我们的任务是将这些边定向，使得每个点的入度为 1，于是假如某个连通块的边数不等于点数，一定无解。

如果连通块中的某个点到它有一个自环，那么对答案的贡献就是 $\times N$ ，因为它形成了形如一棵树 + 一条根到根的边的结构，根的位置有 N 种填法，剩下的边的方向是固定的。否则是一个基环树，对答案的贡献是 $\times 2$ ，因为环上的那些边有逆时针和顺时针两种方向，至于那些环外的链，它们的方向也是固定的。

并查集维护即可，时间复杂度 $\mathcal{O}(n \log n)$ ，其中的 $\log n$ 是并查集的复杂度。

AT_agc008_e

给定正整数 n 和一个长度为 n 的序列 a ，问有多少长度为 n 的排列 p ，满足对于任意 i 有 $p_i = a_i$ 或 $p_{p_i} = a_i$ 。

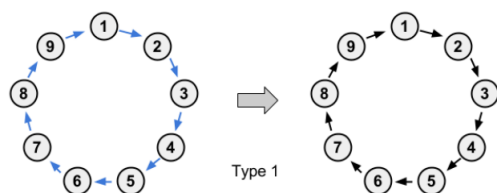
答案对 $10^9 + 7$ 取模。

$n \leq 10^5$ 。

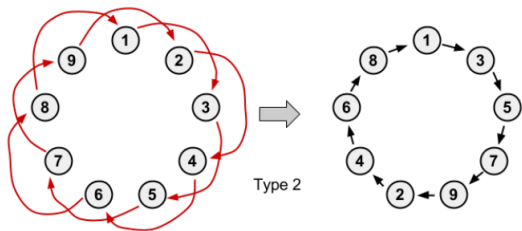
Solution: AT_agc008_e

我们构造两张图：第一张图中， i 向 p_i 连边，整个图由一堆环构成；第二张图中， i 向 a_i 连边。我们需要做的就是考虑有多少种第一张图。

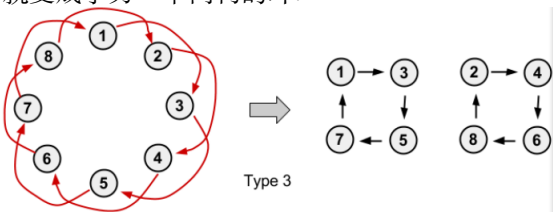
对于第一张图中的每一个环，擦掉它的所有边后让 i 向 a_i 连边，由题可知，这个 a_i 一定是原图中的前一个结点 (p_i) 或前一个结点再往前一个结点，于是我们就可以把所有连边方式分为四类：



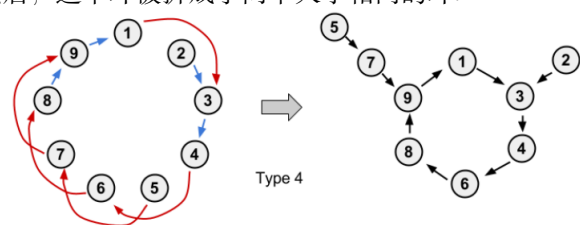
第一类即 $a_i = p_i$ ，没什么好说的。



第二类是这样的：对于一个奇数个点且点数大于 1 的环，将每个点连向它的前一个结点的前一个结点后，这个环就变成了另一个同构的环。



第三类与第二类的区别只有点数奇偶性的不同，当点数为偶数并且将每个点连向它的前一个结点的前一个结点后，这个环被拆成了两个大小相同的环。



第四类即最后一类就是剩下的情况了：既有 $a_i = p_i$ （连向前一个结点）的，又有 $a_i = p_{p_i}$ （连向前一个结点的前一个结点）的，此时这个环变成了一棵内向基环树，并且环外的树一定都是链。

可是我们得到的是变化完的图，目的还原原本的图。环和基环树是可以分别考虑的，最后用乘法原理乘起来即可，接下来我们再分别讨论这两个子图。

对于环的情况，只有“合并”与“不合并”两种选择，对于所有结点个数均为 x 的环，设它们的原图个数有 f_x 个，那么环的情况对于总答案的贡献即为 $\prod_{i=1}^n f_i$ 。

算 f_x 只需要 dp 一下：设 $g_{x,i}$ 为有 i 个结点个数为 x 的环，对总答案的贡献。当 x 为奇数时，

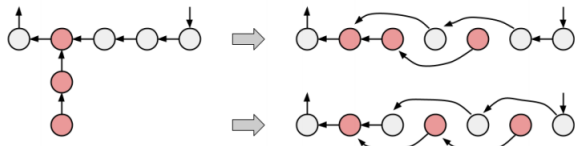
$$g_{x,i} = 2 \times g_{x,i-1} + (i-1) \times x \times g_{x,i-2}, \quad x \text{ odd}$$

简单解释一下就是：假如第 i 个环不和前面的环合并，由于环长是奇数，所以有两种原图可以得到这个环（第一类和第二类操作），贡献是 $2 \times g_{x,i-1}$ ，否则先从前面的环里挑出来一个合并，这里的合并有 x 种方式，最后再乘剩下的环的方案数 $g_{x,i-2}$ 。当 x 为偶数时是类似的：

$$g_{x,i} = g_{x,i-1} + (i-1) \times x \times g_{x,i-2}, \quad x \text{ even}$$

于是 $f_x = g_{x, cnt_x}$ ，其中 cnt_x 指的是整个图中长度为 x 的环的个数，环的贡献就算完了。

对于一棵基环树，它对答案的贡献是多少呢？我们想要将那些链塞到环的内部，类似这样：



这个链可以塞到哪里呢？就是连接这个链和环的那个点与上一条链连接环的那个点之间，并且两个环上的点之间最多插入一个点，设这段的长度为 l_1 ，链的长度为 l_2 。当 $l_1 < l_2$ 时是插入不完的，当 $l_1 = l_2$ 时只有一种插入方法，当 $l_1 > l_2$ 时有两种插入方法，对环上每条链算出数量再乘起来就是这个环的贡献了。

两部分乘起来就是最终答案，时间复杂度 $\mathcal{O}(n)$ 。

AT_agc008_f

Snuke 君有一棵 n 个节点的全白的树，其中有一些节点（至少一个）他喜欢，有一些节点他不喜欢。他会选择仅一个他喜欢的节点 x ，和一个非负整数 d ，将所有与 x 距离不超过 d 的节点都染成黑色，问最后有多少种可能的染色后状态。

两个状态不同当且仅当存在一个节点，它在两个状态中不同色。

$$2 \leq N \leq 2 \times 10^5$$

Solution: AT_agc008_f

肯定存在一种方案使得整棵树全黑，接下来我们只考虑不覆盖整棵树的情况，最后加一即可。

先考虑 Snuke 喜欢所有点的情况，不能直接对每个结点分别算的原因在于有些不同的 (x, d) 产生的效果是一样的，为了不重复计数，对于效果相同的染色方案，我们在 d 最小的那个 x 上算贡献。

设 $S_{x,d}$ 为距离 x 不超过 d 的点构成的集合，那么 (u, d) 产生贡献当且仅当不存在一个与 u 相邻的结点 v 使得 $S_{u,d} = S_{v,d-1}$ ，这又等价于：将 u 作为整棵树的根，删掉 v 那个子树后，存在一个点 x 使得 $\text{dist}_{x,u} > d - 2$ 。形式化地说，设 $f_{u,0}$ 为距离 u 最大的点与 u 之间的距离， $f_{u,1}$ 为距离 u 次大的点与 u 之间的距离， (u, d) 产生贡献当且仅当 $d < f_{u,1} + 2$ ，同时为了不覆盖整棵树，还应满足 $d < f_{u,0}$ ，合起来就是

$$d < \min \{f_{u,0}, f_{u,1} + 2\}$$

做一遍换根 dp 即可，接下来考虑有某些点 Snuke 不喜欢的情況。

此时有些原本应该被算到的方案没有被算到，这是因为对于某些 Snuke 喜欢的点 u ， $S_{u,d} = S_{v,d-1}$ 但 Snuke 不喜欢这些 v ，那么本应被算进答案的 $S_{u,d}$ 就没有被算进去，我们尝试在那些 Snuke 不喜欢的点里算上这部分贡献。

对于这些 Snuke 不喜欢的点，它们的上界不变 ($d < \min \{f_{u,0}, f_{u,1} + 2\}$)，但下界不是 0 了。设 u 是一个 Snuke 不喜欢的点，且 $S_{u,d}$ 满足它是所有相同染色方案中 d 最小的那个，且也是一个 Snuke 喜欢的点 v 的某个 $S_{v,d}$ ，那么这个集合 $S_{u,d}$ 必须包含以 u 为根的那个含有 v 的子树的所有结点¹，同样换根 dp 找到这个下界即可。总时间复杂度 $\mathcal{O}(n)$ 。

AT_agc009_c

给定 n 个不同的整数 S_i ，求将它们分成两个集合 X, Y ，并且 X 集合中任意两个数的差不小于 A ， Y 集合中任意两个数的差不小于 B 的方案数。

$$n \leq 10^5, 1 \leq A, B \leq 10^{18}, 0 \leq S_i \leq 10^{18} (1 \leq i \leq N), S_i < S_{i+1} (1 \leq i \leq N-1)$$

Solution: AT_agc009_c

不妨设 $A \leq B$ ，假如 $S_{i+2} - S_i < A$ 一定无解，先特判掉。

设 f_i 为考虑了前 i 个数后，第 i 个数被分到集合 Y 中的方案数，我们可以在开头加一个 $-\infty$ ，在结尾加一个 $+\infty$ ，于是 $f_1 = 1$ ，答案即为 f_{n+2} 。

转移时考虑哪些 f_j 可以转移到 f_i ，这些 j 一定满足：

- $j < i, A_i - A_j \geq B$.
- $A_{j+1}, A_{j+2}, \dots, A_{i-1}$ 可以一起放在集合 X 中。

满足这样条件的 j 是一段区间，用前缀和优化一下可以做到 $\mathcal{O}(1)$ 转移。找到转移区间的端点可以用二分、预处理之类的。

¹证明参考 <https://blog.csdn.net/little/article/details/83118814>

AT_agc009_d

定义一个单独的节点为一棵 Uninity 0 的树.

将 $x(x \geq 0)$ 棵 Uninity k 的树全部连到一个节点上形成的树, 称之为为一棵 Uninity $k+1$ 的树.

显然, 一棵 Uninity k 的树, 同样也是一棵 Uninity $k+1, k+2, k+3 \dots$ 的树.

现在给你一棵 n 个点的树, 求一个最小的 k 使得这棵树是一棵 Uninity k 的树.

$2 \leq n \leq 10^5$

Solution: AT_agc009_d

实际上就是将若干个 Uninity k_i 的树接到一个根节点上形成一棵 Uninity $\max\{k_i\} + 1$ 的树, 那么最矮的点分树的高度减一就是答案. 问题在于, 我们如果在直接构建点分树的某时刻时遇到了两个重心, 那么这两个重心所形成的不同的点分树的树高可能是不一样的, 而我们又没办法确定到底应该选哪个重心作为根.

但这至少给我们提供了一个重要信息: 答案是 $\log n$ 级别的.

设答案点分树上以点 u 为根的子树高度为 $f_u + 1$, 那么答案即为 $\max\{f_u\}$, 下面给出一个引理:

引理: 若 $f_u = f_v, u \neq v$, 则在原树上 u 到 v 的路径上必然有一点 x 满足 $f_x > f_u$, 同时, 对于满足这样条件的 f , 我们也可以反向构造出一棵点分树.

考虑点分树上的根, 结论显然.

于是, 我们只需要在原树中从叶子到根贪心的选择权值, 用类似状压的方式记录子树中有哪些权值还没有满足条件: 假如有两个子树中都有某个权值, 并且路径上还没有大于该权值的点, 那么该点的权值至少是这个权值加一. 之后, 整棵子树还没有满足条件的权值就是子树中那些大于等于根的权值的值.

时间复杂度 $\mathcal{O}(n \log n)$.

AT_agc009_e

黑板上有 n 个 0 和 m 个 1, 我们每次选择 k 个数字将其擦除, 然后把它们的平均数写上去, 这样一直操作直到只剩下一个数字, 问剩下的这个数字有多少种不同的情况.

答案对 $10^9 + 7$ 取模

$1 \leq n, m \leq 2000, 2 \leq k \leq 2000$

保证 $n + m - 1$ 能被 $k - 1$ 整除.

Solution: AT_agc009_e

将整个过程组织成 k 叉树的形式, 有 n 个权值为 0 的叶子和 m 个权值为 1 的叶子, 每个非叶子结点的权值是它的 k 个儿子权值的平均值, 那么最终留下来的数就是根节点的值.

对于一个权值为 1 的叶子, 它对最终留下来的数的贡献是加 $\frac{1}{k^d}$, 其中 d 代表该叶子的深度, 设 a_1, \dots, a_n 为那些 0 叶子的深度, b_1, \dots, b_m 为那些 1 叶子的深度, 那么根结点的值为 $\sum_{i=1}^m \frac{1}{k^{b_i}}$, 同时 $\sum_{i=1}^m \frac{1}{k^{b_i}} + \sum_{i=1}^n \frac{1}{k^{a_i}} = 1$, 这是因为假如我们将所有 0 换成 1, 根结点的值一定是 1.

将这个数写成 k 进制小数 $0.c_1c_2c_3 \dots c_z$, 不难发现以下性质:

$$\sum_{i=1}^z c_i \equiv m \pmod{k-1}$$

$$\sum_{i=1}^z (k-1-c_i) \equiv n-1 \pmod{k-1}$$

到这里考虑 dp, 设 $f_{i,j,0/1}$ 表示到了小数的第 i 位 (上界为树的层数, 即 $\frac{n+m-1}{k-1}$), 当前数字和为 j , 第 i 位是否为 0 的方案数, 转移如下:

$$f_{a,b,0} = f_{a-1,b,0} + f_{a-1,b,1}$$

$$f_{a,b,1} = \sum_{i=1}^{\min\{k-1,b\}} f_{a-1,b-i,0} + f_{a-1,b-i,1}$$

这里的状态要加上“第 i 位是否为 0”的原因是，假如不加，那么同一个小数会被重复计算（后面填 0 其实不会改变小数的值），于是我们只在小数的最后一位对答案产生贡献。

那么答案即为所有满足 $j \equiv m \pmod{k-1}$ 且 $(k-1) \times i - j \equiv n-1 \pmod{k-1}$ 且 $(k-1) \times i - j \leq n-1$ 的 $f_{i,j,1}$ 的和。

时间复杂度 $\mathcal{O}(m^2 + nm)$ 。

AT_agc010_c

一棵 n 个点的树，第 i 个结点上有 A_i 个石头，每次选择两个度数为 1 的结点，将路径上经过的（包括起点终点）所有结点上都取走一个石头，如果路径上有一个点上没石头这个操作就不能进行，问能不能取完所有石头。

$2 \leq n \leq 10^5, 0 \leq A_i \leq 10^9$

Solution: AT_agc010_c

特判掉 $n = 2$ 的情况，选出一个度数不为 1 的点作为根。

设 f_u 为结点 u 向上伸出的路径个数， $s_u = \sum_{v \in \text{son}_u} f_v$ ，则有：

$$f_u + \frac{s_u - f_u}{2} = A_u$$

其中 f_u 指每次向上伸出路径，石头数都会减一， $\frac{s_u - f_u}{2}$ 指让子树中伸出的路径两两匹配，每组匹配会让点 u 的石头数减一，最终一共要减 A_u 。

移项整理可得 $f_u = 2A_u - s_u$ ，注意特殊情况：叶子结点的 f 值就是其 A 值。

于是这些 f 其实是定值，我们只需要 dfs 一遍，求出 f ，假如根节点的 f 为 0 且对于所有结点 u ，满足 $0 \leq f_u \leq A_u, f_u \leq A_{\text{fa}_u}$ ，就一定可以构造出一种合法方案。时间复杂度 $\mathcal{O}(n)$ 。

AT_agc010_e

有一个长度为 n 的数列 A_i 。

高桥君会把整个序列任意排列，然后青木君可以进行任意次操作，每次选择两个相邻的互质的数交换位置。

高桥君希望最终序列的字典序尽量小，而青木君希望字典序尽量大。求最终序列。

$1 \leq n \leq 2000, 1 \leq A_i \leq 10^8$

Solution: AT_agc010_e

首先一个套路是：假如某两个数不互质，那么无论怎样交换，这两个数的相对位置不变。我们将 $\text{gcd}(A_i, A_j) \neq 1$ 的 i, j 之间连一条无向边，表示这两个数有相对位置关系，高桥君所做的实际上就是将这些无向边定向为一个 DAG，青木君所作的就是将这个 DAG 按照值大优先来拓扑排序（就是把正常拓扑排序所用队列换成大根堆，小于关系是两个位置上值的小于关系）。

只需要考虑高桥君怎样定向才能让青木君拓扑排序的结果的字典序尽量小，这又是一个技巧：对每个联通块分别考虑，我们只需要定向出一棵生成树，树的儿子向父亲连有向边，那么祖先与后代的边是不需要考的（根据已有的边就可以确定），具体来说，每次找出没有被考虑过的值最小的点，dfs 这个联通块，设 v_1, v_2, \dots 是从 u 扩展出的点，同时满足 $A_{v_1} \leq A_{v_2} \leq A_{v_3} \dots$ ，那么边的方向就是从 u 到 v_i ，然后按照 v_1, v_2, \dots 的顺序往下 dfs。

为什么这样做是对的呢？只需要证明对连通图，这样做是对的（因为不同的联通块之间互不干扰）。我们找到了值最小的那个点 u ，让 A_u 在其他值之前，之后接着保证那个接下来往后接的是可行最小值 A_{v_1} ，以此类推。时间复杂度 $\mathcal{O}(n^2)$ 。

AT_agc010_f

有一棵 n 个节点的树，第 i 条边连接 a_i, b_i ，每个节点 i 上有 A_i 个石子，高桥君和青木君将在树上玩游戏。

首先，高桥君会选一个节点并在上面放一个棋子，然后从高桥君开始，他们轮流执行以下操作：

- 从当前棋子占据的点上移除一个石子。
- 将棋子移动到相邻节点。

如果轮到一个人执行操作时棋子占据的点上没有石子，那么他就输了

请你找出所有的点 v ，使得如果高桥君在游戏开始时把棋子放到 v 上，他可以赢

$2 \leq N \leq 3000, 0 \leq A_i \leq 10^9$

Solution: AT_agc010_f

引理：走向石子数更大或相等的点一定不优。这是因为对方可以在下一步将棋子移回来，一直这样下去，走这一步的人必败。

于是，放在 v 上先手必胜，当且仅当存在一个相邻点 u 满足 $A_u < A_v$ 且在 u 的先手必败。

只需要将所有点的编号按照 A 升序排列，顺序扫一遍，对每个点判断即可。

AT_agc011_c

给定一张有 n 个点， m 条边的原图，现构成一张新图，其中每个点都是一个二元组 (a, b) 。

2 个二元组 $(a, b), (c, d)$ 有边当且仅当 a 和 c 有边且 b 和 d 有边，求新图的连通块个数。

$2 \leq N \leq 100,000, 0 \leq M \leq 200,000$ ，无重边或自环

Solution: AT_agc011_c

首先将所有孤立点单独算出来，剩下的图变成一堆点数大于 1 的连通块，设孤立点个数为 cnt_C ，则这些点对答案的贡献为

$$[2n - 1] + [2(n - 1) - 1] + [2(n - 2) - 1] + \cdots + [2(n - cnt_C + 1) - 1]$$

$$= (2n - cnt_C + 1) \times cnt_C - cnt_C$$

这是因为这个孤立点无论作为新点的第一维或第二维，同样是孤立的（因为在原图中没有点连向这个点），减去同时作为第一维和第二维的重复情况，就有 $2n - 1$ 个新点孤立，一直这样算下去构成一个等差数列。

下面考虑什么时候 (a, b) 和 (c, d) 联通，这等价于存在 a 到 c 的路径和 b 到 d 的路径，并且两者长度相同，而这又等价于存在 a 到 c 的路径和 b 到 d 的路径，并且两者奇偶性相同（原因是可以在一条边的两个端点之间打转等另一边慢的）。

考虑一种不重不漏的计数方法：对每个连通块，只在它最小的 (a, b) 上计数，将它作为这个连通块的代表元。将所有连通块分为两类：集合 A 中的连通块存在奇环，集合 B 中的不存在奇环，

怎样的 (a, b) 才能作为代表元呢？首先， a 必须是它所在连通块中编号最小的那一个，否则就一定可以连向一个第一维小于 a 的新点，接下来， (a, b) 不是代表元当且仅当存在一个 b' 使得 (a, b) 与 (a, b') 连通。

当 a 所在连通块在集合 A 中时，从 a 到 a 既有长度为奇数的路径，又有长度为偶数的路径，那么不存在这样的 b' 当且仅当 b 是其所在连通块的编号最小的点，这样的 b 有 $|A| + |B|$ 个，即每个连通块有一个。

当 a 所在连通块在集合 B 中时，从 a 到 a 的路径长度只能是偶数，那么不存在这样的 b' 当且仅当从 b 经过偶数条边后能够到达的点的编号均不小于 b ，对于每个在集合 A 中的连通块，这样的 b 有 1 个，因为任意两个

点之间都可以经过一条偶数路径到达，而对于每个在集合 B 中的连通块，这样的 b 有 2 个，这是因为在二分图中，经过偶数条路径，左部点只能到左部点，右部点只能到右部点，所以左部点和右部点各有一个，故这样的 b 共有 $|A| + 2|B|$ 个。

全部加起来即可，答案为

$$\left[(2n - cnt_C + 1) \times cnt_C - cnt_C \right] + \left[|A| \times (|A| + |B|) \right] + \left[|B| \times (|A| + 2|B|) \right]$$

AT_agc011_d

有 N 个机器排成一排，每个机器有两种状态：

- A：会把球反弹（即让球反方向滚动）
- B：让球自由通过（即让球沿原方向滚动）

每有一个球撞上机器，这个机器就会改变自身的状态，给你这 N 个机器的初始状态，将一个球从左向右滚进去 K 次（每次滚动直到从最左端或者最右端滚出来才算结束），询问最终所有机器的状态。

$$1 \leq N \leq 2 \times 10^5, 1 \leq K \leq 10^9$$

Solution: AT_agc011_d

将 A 视作 1，B 视作 -1， n 个机器状态构成数列 $A_{0..n-1}$ 。手模几个样例可以发现如下规律：

- 当 $A_0 = 1$ 时，该球的效果就是将 A_0 变为 -1。
- 否则，该球的效果是将 A_i 变为 $-A_{(i+1) \bmod n}$

第一条是显然的，我们来证明一下第二条。用 L 和 R 描述球应该向哪个方向走。假如一直没有碰到 A，结论成立，否则一定是形如...ARA... 的形式（因为在 R 之前的 B 会被修改为 A，符合我们的结论），之后，状态会被依次改为...ALB..., ...BRB..., ...BAR.... 这样，原本的 BA 被改成了 B_（其中 _ 是指这个位置的状态还未知），符合我们的结论。一直这样走下去即可。

于是第二条本质上相当于：将所有状态变为相反的状态后，让起点向右移动一位。于是就可以维护起点与全局变化次数，做到 $\mathcal{O}(n + K)$ 模拟了。

但是 K 有 10^9 ，还是有点吃力，考虑怎样减少放球的次数。每次执行第二条后，末尾都会加一个 B，之后再执行第二条时，这个 B 会变成 A，同时末尾再加一个 B 变成 AB。那么在执行 $2N$ 次操作后，整个状态一定变成 ABAB...AB 或 _BAB...ABA，具体是哪一个则取决于 N 的奇偶性。在这之后出现了循环节 2，所以我们只需要操作 $\min\{K, 2N + (K \bmod 2)\}$ 次，时间复杂度 $\mathcal{O}(\min\{N, K\})$ 。

AT_agc011_e

我们说一个数是“递增的”，当且仅当对于它的任意相邻的两位都有左边小于等于右边。如 1558, 11, 3 是递增的，20170312、19260817 就不是。

现在给你一个数 n ，问最少可以被表示成几个递增的数之和。比如 $80 = 56 + 24$ ， $2017 = 1349 + 668$ ， $2019 = 1669 + 237 + 113$

$$1 \leq n \leq 10^{500000}$$

Solution: AT_agc011_e

设 $Num_i = \underbrace{111\dots11}_{i \text{ 个 } 1}$ ，那么一个“递增的”数一定可以被拆成至多 9 个 Num_i 之和，例如

$$6778 = 6 \times 1111 + 111 + 1$$

设 n 被拆为了 k 个“递增的”数，再设这 k 个“递增的数”又可以被拆成 $r_{1..9k}$ 这 $9k$ 个 Num_i ，即

$$n = \sum_{i=1}^{9k} Num_{r_i}$$

我们要球的就是 k 的最小值.

注意到每个 Num_i 本质上是一个等比数列的和:

$$Num_i = \sum_{j=0}^{i-1} 10^j = \frac{10^i - 1}{9}$$

那么

$$n = \sum_{i=1}^{9k} \frac{10^{r_i} - 1}{9}$$

移项得

$$9n + 9k = \sum_{i=1}^{9k} 10^{r_i}$$

假如等式的右侧不进位, 那么各数位上的数字和就是 $9k$, 否则只会更小, 我们要找最小的 k , 实际上就是要找最小的 k , 满足 $9n + 9k$ 的各数位上数字的和大于等于 $9k$, 因为这样我们一定可以构造出一组解.

从小到大枚举 k 即可, k 最大是 $\mathcal{O}(\log_{10} n)$ 级别的, 代码实现上的麻烦之处在于高精度.

AT_agc011_f

有一条铁路, 上面有 $n+1$ 个站台和 n 段路, 站台编号为 $0 \sim n$, 铁路编号为 $1 \sim n$. 铁路分为单向铁路和双向铁路, 列车通过第 i 段铁路需要花费的时间为 t_i .

你的任务是设计一张列车时刻表, 每隔 K 时间会发出一个从 0 到 n 的列车, 同样, 每隔 K 时间会发出一个从 n 到 0 的列车, 这两种列车发出的时间可以不同, 如前者的发出时刻是 $0, K, 2K, \dots$ 而后者是 $3, K+3, 2K+3, \dots$. 列车能且只能在站台上停留若干时刻, 并且所有方向相同的列车的停留策略是相同的 (相当于在时间轴上直接平移 K), 其中 K 是给定常数.

时刻表的限制只有一个: 在单项铁道上不能有两个方向相反的车辆互相穿过, 你要求出满足条件的时刻表里从 0 到 n 的列车所需时间与从 n 到 0 的列车所需时间的和的最小值, 无合法方案输出-1.

样例见原题.

$$n \leq 10^5$$

Solution: AT_agc011_f

有解当且仅当对于任意单向铁道 i 有 $2t_i \leq K$.

记 p_i 为从 0 到 n 的车在站台 i 停靠的时间, 则车辆在 p_0 时刻发出.

记 q_i 为从 n 到 0 的车在站台 i 停靠的时间, T, P, Q 分别为 t, p, q 的前缀和, 下面所有的讨论均在模 K 意义下进行.

为了方便叙述, 将 0 到 n 的列车称为正向车, 另一个方向则为反向车. 对于第 i 段铁道, 正向车行驶的时间为 $(T_{i-1} + P_i, T_i + P_i)$, 反向车行驶的时间为 $(-T_i - Q_i, -T_{i-1} - Q_i)$.

当第 i 段是单向铁道时, 要求这两段区间不交, 即

$$\begin{cases} T_{i-1} + P_i + cK & \notin (-T_i - Q_i, -T_{i-1} - Q_i) \\ T_i + P_i + cK & \notin (-T_i - Q_i, -T_{i-1} - Q_i) \\ -T_i - Q_i + cK & \notin (T_{i-1} + P_i, T_i + P_i) \\ -T_{i-1} - Q_i + cK & \notin (T_{i-1} + P_i, T_i + P_i) \end{cases}$$

记 $x_i = P_i + Q_i$, 整理得到 $x_i + cK \notin (-2T_{i-1}, -2T_i)$, 而 x_i 的约束只有单调不减.

问题转化为有一个初始值任意的变量 x , 每次给出一个模 K 意义下的区间, 要求将 x 加上某值使得 x 落在这个区间上, 最小化加的数的和.

可以 dp, 设 f_i 为当前要使 $x = i$ 所需代价的最小值, 初始值为 $f_i = 0$, 每次操作时将区间以外的位置转移到区间中并置为 $+\infty$, 线段树维护 $f_i - i$ 的区间最小值即可, 最终答案还要加上 $2T_n$. 时间复杂度 $\mathcal{O}(n \log n)$.
还可以直接用珂朵莉树维护, 具体可看洛谷题解, 这里不再给出具体解法.

AT_agc012_c

我们称一个字符串 x 是好的当且仅当它满足以下条件:

- x 可以被表示为另外一个串 y 复制一遍得到, 即 $x = yy$.

例如 'aa' 和 'bubobubo' 是好的, 'a', 'abcabcabc' 和 'abba' 不是.

现在要求一个串 s 满足下列条件, 可以证明这个串存在:

- $|s| \leq 200$
- 字符集大小为 100, 即每个字符用 $[1, 100]$ 的整数表示.
- 在 s 的所有的 $2^{|s|}$ 个子序列中, 恰好有 N 个串是好的, 其中 N 是给出的.

$$1 \leq N \leq 10^{12}$$

Solution: AT_agc012_c

给出构造解: 让前 100 个数字依次为 1 到 100, 让后面的数字为 1 到 M 的一个排列, 那么这个串中, 好的子串的数目为这个排列的上升子序列的数目.

依次填 $1, 2, \dots, 100$, 最开始是空, 往末尾填可以让数目变为原来的 2 倍加 1, 填到开头可以让数目加 1, 按照 N 构造即可.

AT_agc012_d

N 个球排成一排, 第 i 个球的颜色为 c_i , 重量为 w_i . 我们定义「一次操作」为: 选择两个颜色**相同**, 且重量之和不大于 X 的球, 交换它们的位置; 或选择两个颜色**不同**, 且重量之和不大于 Y 的球, 交换它们的位置. 问进行任意次操作后, 可以得到多少种不同的颜色序列. 输出答案对 $10^9 + 7$ 取模的结果.

$$1 \leq c_i \leq N \leq 2 \times 10^5$$

Solution: AT_agc012_d

一个关键性质是“可交换”满足传递性, 即若 a 和 b 可以交换, b 和 c 可以交换, 那么 a 和 c 也就可以交换, 方法是先换 ab , 再换 bc , 再换 ab , 那么原本的 abc 就会被换成 cba 了.

于是一个 $\mathcal{O}(n^2)$ 做法就有了: 将“可交换”关系连成边, 每个连通块分别求解, 用乘法原理得到最终答案. 对于一个连通块, 设 s_i 为颜色为 i 的球的数量, 答案即为

$$\frac{(\sum_i s_i)!}{\prod (s_i!)}$$

复杂度瓶颈在于连边, 所以我们想办法连尽量少的边的同时保证得到的连通性正确.

对于同色球之间的边, 我们只需要连这个颜色中 **重量最小的球** (质量设为 w_{min}) 和其他这个颜色的球之间的边, 原因是假如 $w_1 + w_2 \leq X$, 那么一定满足 $w_1 + w_{min} \leq X$ 且 $w_2 + w_{min} \leq X$, 这部分是 $\mathcal{O}(n)$ 条边.

对于异色球之间的边, 首先找出 **所有球中质量最小的球的那一个**, 设其质量为 w_{min} , 颜色为 c_{min} , 连它和其他颜色的所有球之间的边, 共 $\mathcal{O}(n)$ 条边. 原因是两个异色球可以交换, 那么这两个球和质量最小的球也一定可以分别交换.

但是这样还有一个问题, 假如这两个球中有一个球的颜色恰是 c_{min} , 连通性就不一定对了, 这是因为 Y 可能小于 X , 这会导致一种情况的出现: 存在一个颜色为 c_{min} 但不能和同色球交换的球, 它可以和异色球交换, 而这些边都没有维护这个连通关系.

解决这个问题并不难, 找到颜色不为 c_{min} 的球中质量最小的那个球, 将它和其他颜色的球再连边即可, 正确性显然.

总共连了 $\mathcal{O}(n)$ 条边，直接连边，最后若干次 dfs，时间复杂度 $\mathcal{O}(n)$ ，假如用并查集和 `std::map` 维护就会多一个 \log 。

AT_agc012_e

给定 n 个绿洲，第 i 个绿洲的坐标为 x_i ，保证 $-10^9 \leq x_1 < x_2 \dots < x_n \leq 10^9$

现在有一个人在沙漠中进行旅行，他初始的背包的水容积为 V 升，同时他初始拥有 V 升水，每次到达一个绿洲时，他拥有的水的量将自动重置为容积上限（可以使用多次）。他现在可以选择两个操作来进行旅行：

- 消耗 d 升水走到距当前绿洲距离为 d 的绿洲。任意时刻水的数量不能为负数。
- 令 v 为你当前拥有的水量，若 $v > 0$ ，则你可以跳跃至任意一个绿洲，然后重置容积上界和所拥有的水量为 $\lfloor \frac{v}{2} \rfloor$ 。

对于每一个 i 满足 $1 \leq i \leq n$ ，你要求解，当你在第 i 个绿洲作为起点时，你能否依次遍历其他所有绿洲。如果可以，输出 Possible，否则输出 Impossible。 $n, V \leq 2 \times 10^5$ 。

Solution: AT_agc012_e

我们的策略显然是走路-跳跃-走路-跳跃... 并且每次走路时，我们会将一个区间全走完，最多跳跃 $\mathcal{O}(\log n)$ 次，这是因为每次水的容积都除以 2。

设 $L_{i,j}$ 为已经进行了 j 次跳跃后，从 i 出发能够走路到达的左边界， $R_{i,j}$ 为右边界，考虑把最开始的那一步留出来，先走 $\frac{v}{2}$ ， $\frac{v}{4}$ ，... 这些步，将它们状压，状态的二进制第 i 位为 1 表示已经走了水量为 $\frac{v}{2^i}$ 的这一步。

再令 f_{st} 表示当前状态为 st ，使 $1..x$ 都走过的最大的 x ， g_{st} 表示当前状态为 st ，使 $x..n$ 都走过的最小的 x ，那么两个二进制第 0 位为 0 的互补状态拼起来（如 1001010 和 0110100，它们拼起来是 1111110，正好还差最开始的一步没走），假如中间还没有走的那段区间可以在最开始的一步里全部访问，那么以**最开始的一步里可以访问到的点**为起点，一定可以访问完所有的点。这就是一个区间赋值的问题，可以上线段树或珂朵莉树。

f 和 g 可以每次枚举最右（左）边是哪一步做到 $\mathcal{O}(\log n)$ 转移，总复杂度 $\mathcal{O}(n \log n)$ 。

AT_agc012_f

给定一个长度为 $2n - 1$ 的序列 A ，你可以随意排列 A 中的元素，请求出有多少种不同的序列 B ，满足 B 的长度为 n 且 $B_i = \{A_1 \dots A_{2i-1}\}$ 的中位数。

$n \leq 50$ ，答案对 $10^9 + 7$ 取模。

Solution: AT_agc012_f

一个显然的结论是 B_n 的值是确定的：就是 $\{A_1 \dots A_{2n-1}\}$ 的中位数，无论怎样重排 A ，这些数的中位数都是不变的。

尝试倒着往前走，设当前已经确定了 B_i ，从这些数里删掉两个后找到新的中位数作为 B_{i-1} ，接着往下直到 B_1 ，在这个过程中，**新的中位数只可能变成原来中位数的前驱或后继或保持不变**。若删掉的两个数分别位于中位数的两侧，则中位数不变，否则若均不在中位数右边（可以删中位数），中位数变成后继，否则均不在左边（同样可以删中位数），中位数变成前驱。

这告诉我们什么呢？对于任意 $i < j$ ，不可能有 $B_j < B_i < B_{j+1}$ 或 $B_{j+1} < B_i < B_j$ ，原因就是之前所说的，只可能变成前驱、后继或不变，因此从 B_{j+1} 走到 B_j ，这两个数之间必然没有数。

这是 B 间不同位置的关系，称作限制一，接下来我们对每个位置找它本身的限制，称作限制二。

若 x 能成为 B_i ，应该满足什么条件呢？为了方便，先设 A 中数字两两不同，那么此时应当有 $i - 1$ 个数小于 x ， $i - 1$ 个数大于 x ，那么设排序后的 A 为 A' ，能成为 B_i 的 A'_j 应当满足 $j \geq i$ 且 $2n - j \leq i$ ，即

$$i \leq j \leq 2n - i$$

接下来我们直接用 A' 替换 A ，下文中的 A_i 均为排序后的值，即 A'_i 。

回顾我们得到的两个性质，根据限制一，我们每次填 B_i 后会有一些 A_j 永远无法变成某个 B ，这些被排除的 j 一起构成了一段区间；根据限制二，每次向左填都会扩展出两个额外可作为中位数的数字，那么限制一将限制二的区间割开成了左右两部分，这两部分中的值是可供我们选择的，根据这个设计出 dp 状态和转移方程。

具体来说，设 $dp_{i,l,r}$ 为填好了 $B_{i+1}, B_{i+2} \dots$ 后，左边区间有 l 种取法，右边区间有 r 种取法的方案数，转移用刷表法，枚举 $dp_{i+1,l,r}$ 和我们钦定的 B_i 即可。

假如有重复的数字，限制二就可能不能扩展或只能向左或向右扩展，额外判断即可，时间复杂度 $\mathcal{O}(n^4)$ 。

CF1819B

有一个 $h \times w$ 的矩形，每次横向切一刀（切成 $h \times i$ 和 $h \times (w - i)$, $1 \leq i < w$ ）或纵向切一刀（切成 $i \times w$ 和 $(h - i) \times w$, $1 \leq i < h$ ），每切一刀后取出切出的两块矩形中的一块放进盒子内，对剩下的那个矩形接着操作，这样操作 $n - 1$ 次后得到 n 个矩形。

现将这些矩形打乱顺序但不旋转（即如果切出来时是 $a \times b$ ，且 $a \neq b$ ，我们不会把它错误地变成 $b \times a$ ），询问有多少种原本的 $h \times w$ 矩形能够 $n - 1$ 次操作切出这些矩形。

$1 \leq n \leq 2 \times 10^5$

Solution: CF1819B

本题的关键性质是这样的矩形至多只有两个。考虑对 $h \times w$ 矩形的第一次操作，放进盒子中的小矩形必然是长为 h 或宽为 w 的，因此，最初的矩形至多有两种情况：长为所有矩形长的最大值或宽为所有矩形宽的最大值。

接下来就好办了，以长为所有矩形长的最大值为例，我们尝试拼出这个矩形，方法是先不断取出长与矩形长相等的矩形拼上去并且减少剩余宽的长度，拼不了时就拼宽和剩余宽长度相同的矩形，这样交替进行。

假如某时刻无论是长等于待拼长还是宽等于待拼宽都拼不了，说明该情况不合法。用 `std::map` 维护所有还没有被用的矩形，时间复杂度 $\mathcal{O}(n \log n)$ 。

CF1819C

给定整数 n 和一棵包含 n 个节点的树。记 $\text{Dist}(x, y)$ 表示树上节点 x, y 之间最短路径的边数，你需要判断是否存在一个 $1 \sim n$ 的排列 p ，满足：

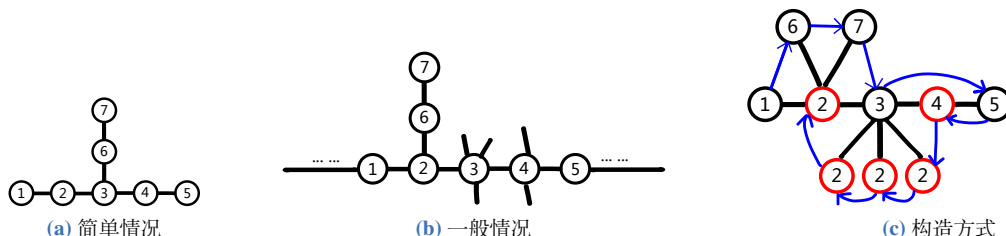
- $\text{Dist}(p_i, p_{i+1}) \leq 2$ 对任意整数 $i (1 \leq i < n)$ 成立。
- $\text{Dist}(p_1, p_n) \leq 2$ 。

存在则输出 Yes 并构造方案，否则输出 No。

$2 \leq n \leq 2 \times 10^5$

Solution: CF1819C

先找找规律试试，这个约束条件很苛刻，一般的图看上去不太可能存在这样的方案，观察图 (a) 这棵树：



我们发现即使是这样一个简单的结构也找不到一个符合条件的排列，那么我们将这个图扩展成图 (b) 的样子，这棵树也一定同样无解。

怎样描述图 (b) 这个结构呢？找出树的直径后，假如这棵树不能看做是一条链（直径）上的点再挂上一些叶子的形式，即有一个直径上的点挂了一个长度大于等于 2 的路径，一定无解。

否则这棵树是直径上挂着一堆叶子，一定有解吗？将图黑白染色即可得出构造解：先把与起点同色的全染掉，再转一圈染异色点，如图 (c) 所示。

找出直径判断即可，时间复杂度 $\mathcal{O}(n)$ ，实现较好的话代码量并不大。

CF1061E

现有两棵有根树，标号均为 $1 \sim n$ ，称为红树和蓝树，红树的根为 $root_r$ ，蓝树的根为 $root_b$ 。

编号为 i 的点有点权 w_i ，要求选出满足限制的一些点的编号，问权值和的最大值，限制如下：

- 对于红树，给定若干个有序对 $\langle u, x \rangle$ 表示在红树中以 u 为根的子树中恰有 x 个编号被选中。
- 对于蓝树，给定另外若干个有序对 $\langle u, x \rangle$ 表示在蓝树中以 u 为根的子树中恰有 x 个编号被选中。

保证对于红树的限制，一定存在 $\langle root_r, x \rangle$ ， x 为一个非负整数，对于蓝树的限制一定存在 $\langle root_b, x \rangle$ ， x 为一个非负整数，并且给出的限制不会矛盾（指不会让子树中恰有 x 个和 y 个且 $x \neq y$ ）。

假如无解，输出 -1。

$1 \leq n \leq 500$

Solution: CF1061E

题目看着就很费用流，首先把编号为点 i 拆成 i 和 $n+i$ ，建出源点 S 和汇点 T 。

若有对红树的限制 $\langle u, x \rangle$ ，就设在红树上 u 的限制是 x ，记为 $limr_u = x$ ，否则 $limr_u = 0$ ，相应地，蓝树的限制记为 $limb_u$ 。若有 $limr_u \neq 0$ 那么从 S 向 u 连一条费用为 0，容量为

$$limr_u - \sum_{v \in sonr_u} szr_v$$

的弧，其中

$$szr_u = \begin{cases} limr_u & limr_u \neq 0 \\ \sum_{v \in sonr_u} szr_v & limr_u = 0 \end{cases}$$

$sonr_u$ 指在红树上 u 的儿子， $sonb_u$ ， szb_u 的定义类似。

这个容量实际上就是将子树内的限制都满足后，还需要多几个子树内编号才能满足点 u 的需求。

若 $limb_u \neq 0$ ，那么从 $n+u$ 向 T 连一条费用为 0，容量为

$$limb_u - \sum_{v \in sonb_u} szb_v$$

的弧，和红树几乎一致。

接着，我们对每个点 u ，定义 $topr_u$ 为红树中根到 u （包括端点）的路径上，有限制的深度最大的点的编号， $topb_u$ 的定义是类似的。

对每个 u ，连一条从 $topr_u$ 到 $n+topb_u$ ，容量为 1，费用为 w_u 的弧，跑最大费用最大流，若最大流跑不满从 S 出发的弧或跑不满终点为 T 的弧，说明无解，否则解为费用。

CF1106E

新年就要到啦, Bob 打算去要很多红包! 不过要红包是一件很费时间的的事情.

我们假设从第 1 秒开始共有 n 秒, 第 i 个红包可以在第 s_i 到 t_i 秒 (包括端点) 收集, 并且其中有 w_i 元. 如果 Bob 拿了第 i 个红包, 那么他直到第 d_i 秒后 (不包括 d_i) 才可以继续收集红包. 其中 $s_i \leq t_i \leq d_i$.

Bob 是一个贪心的人, 他贪心地收集红包: 如果他在第 x 秒有红包可以收集, 他就会选择其中钱最多的那个红包. 如果这样的红包有多个, 他就选 d 最大的那个红包. 如果还是多个选择, 他就随便拿一个.

然而他的女儿 Alice 并不想他的爸爸拿到太多钱. 她可以干扰 Bob 最多 m 次. 如果 Alice 在第 x 秒干扰 Bob, 在这一秒 Bob 就不能收集红包, 然后下一秒 Bob 会继续用自己的策略收集.

现在请你求出如果 Alice 采用最优的策略, Bob 能拿到的最少数钱数.

$1 \leq n \leq 10^5, 0 \leq m \leq 200, 1 \leq k \leq 10^5, 1 \leq s_i \leq t_i \leq d_i \leq n, 1 \leq w_i \leq 10^9$

Solution: CF1106E

假如没有 Alice 干扰, Bob 的策略是固定的, 这可以 $\mathcal{O}(k \log k)$ 预处理出来.

设 $dp_{i,j}$ 为 Alice 已经干扰了 i 次, 现在到了时刻 j , Bob 收集的最少数钱数, 初始值 $dp_{0,1} = 0$, 其它为 $+\infty$, 转移平凡.

时间复杂度 $\mathcal{O}(k \log k + nm)$.

CF900D

给定 x, y , 询问有多少数列 a 满足 $a_i \geq 1$ 且 $\sum a_i = y$ 且 $\gcd(a_i) = x$.

$1 \leq x, y \leq 10^5$, 答案对 $10^9 + 7$ 取模.

Solution: CF900D

当 $x \nmid y$ 时显然无解, 否则答案即为所有数的和为 $\frac{y}{x}$ 且 \gcd 为 1 的数列个数.

先不考虑 \gcd , 设 $g(n)$ 为和为 n 的数列个数, 用枚举数列长度后用隔板法得

$$g(n) = \sum_{i=1}^n \binom{n-1}{i-1} = 2^{n-1}$$

设 $f(n)$ 表示 \gcd 为 1 的和为 n 的数列个数, 于是又有

$$g(n) = \sum_{d|n} f(d)$$

这是枚举数列 \gcd 的到的, 于是可以反演:

$$f(n) = \sum_{d|n} \mu\left(\frac{n}{d}\right) g(d)$$

令 $c = \frac{y}{x}$, 我们要求的就是 $f(c)$, 枚举因子只需要枚举 $\mathcal{O}(\sqrt{c})$, 另一侧可以对称过去, 计算 μ 可以预处理质数后暴力算, 时间复杂度是 $\mathcal{O}\left(\frac{\sqrt{c}}{\log \sqrt{c}}\right)$, 计算 g 直接快速幂, $\mathcal{O}(\log n)$.

总时间复杂度为 $\mathcal{O}\left(\frac{c}{\log c}\right)$.

Luogu P9345

令一个 $1 \sim n$ 的排列 p 的乡愁度为 b 中不同元素的个数, 其中 $b_i = \gcd(p_i, p_{i+1})$, 特别地, $p_{n+1} = p_1$.

现在给定 n 和乡愁度 k , 询问是否存在一个乡愁度为 k 的 $1 \sim n$ 排列, 存在则输出方案.
 $1 \leq k \leq n \leq 3 \times 10^5$

Solution: Luogu P9345

首先, 乡愁度的上界是 $\lfloor \frac{n}{2} \rfloor$, 这是因为 $1 \sim n$ 中两个数的最大公因数一定不会超过 $\lfloor \frac{n}{2} \rfloor$.

这个上界是一定可以得到的吗? 考虑对于每个 i , 假如 $2i$ 不超过 n , 就把 $2i$ 放在 i 后面, 将这些链顺次连接起来得到

$$(1\ 2\ 4\ 8 \cdots)(3\ 6\ 12 \cdots)(5\ 10 \cdots) \cdots$$

这样的结构, 不难验证, $1 \sim \lfloor \frac{n}{2} \rfloor$ 均在 b 中出现过.

那 k 小于上界时怎么办呢? 将 $n, n-1, \dots, 2k+1$ 接在 1 后面, 当这些数不存在即可, 剩下的数按之前的策略跟在后面.

时间复杂度 $\mathcal{O}(n)$.

AT_agc013_b

给一张简单无向连通图, 点数为 n , 边数为 m , 你需要找出一条满足以下条件的路径:

- 路径点数 ≥ 2 .
- 路径不经过相同的点.
- 如果点 x 与路径的一个端点直接相连, 那么 x 出现在路径中.

$$1 \leq n, m \leq 10^5$$

Solution: AT_agc013_b

从任意一个点出发 dfs 出一条不经过重复点的路径, 直到不能再走, 那么这条路径的另一端一定满足.

但与起点相邻的点却不一定都在路径中, 这时再从起点出发 dfs 出另外一条路径, 将两条路拼在一起就行了, 时间复杂度 $\mathcal{O}(n+m)$.

AT_agc013_c

有一个长度为 L 的圆环, 上面有 N 个蚂蚁, 位置分别为 X_i , 运动方向为 D_i , 1 表示顺时针, 2 表示逆时针.

每只蚂蚁将会同时开始以单位速度运动, 如果两只蚂蚁相遇, 那么它们会改变自己的方向继续运动.

求 T 秒之后每只蚂蚁的位置.

$$1 \leq n \leq 10^5, 0 \leq X_1 < X_2 < \dots < X_N \leq L - 1$$

Solution: AT_agc013_c

两只蚂蚁碰撞时, 我们将“改变方向继续移动”视作它们对穿而过且交换编号, 这样我们就能得到最终状态下所有蚂蚁的位置了.

但由于我们不能确定哪只蚂蚁在哪个位置, 所以还要进一步进行分析.

显然, “改变方向继续移动”不改变相对顺序, 因此我们只需要知道最开始编号为 1 的蚂蚁在哪个位置, 其它蚂蚁就可以按照相对顺序找到它们的位置了.

接下来是神仙的一步操作：每有一只蚂蚁顺时针穿过位置 0，原本编号为 1 的蚂蚁前面就会多一只蚂蚁；每有一只蚂蚁逆时针穿过位置 0，原本编号为 1 的蚂蚁前面就会少一只蚂蚁，按照这个得到一个偏移量 δ 表示编号为 1 的蚂蚁前面多了几只蚂蚁，那么原本编号为 i 的蚂蚁现在就在第 $i + \delta$ 个位置了。

时间复杂度 $\mathcal{O}(n)$ 。

UOJ 【UR #1】外星人

初始给定数 x 和长度为 n 的数组 A ，“数字 i 经过 A_j ”是指将 i 变为了 $i \bmod A_j$ ，例如若 $A_2 = 3$ ，则 8 经过 A_2 就会变成 2。

现在你可以随意对 A 数组重新排序，排序后将给定的 x 按照 $A_1, A_2, A_3 \dots$ 的顺序依次经过数组 A （这里是指，先经过 A_1 ，再用得到的数经过 A_2 ，一直到 A_n ），询问最终得到数字的最大值以及所有 A 的排序的方案中可以得到这个最大值的方案数，方案数对 998244353 取模。

$n \leq 1000, 1 \leq x, A_i \leq 5000$

Solution: UOJ 【UR #1】外星人

首先考虑怎么计算最大值。取模的一个性质是：将 x 对一个大于它的数取模等价于没有取模，于是我们将 A 降序排列，设 $f_{i,j}$ 为当前考虑了 A_1 到 A_i ，经过了其中若干个数（注意，**不必全部经过**）后得到的数字是否可以 j ，初始值为 $f_{0,x} = \text{True}$ ，转移平凡，可以用刷表法，即用 $f_{i,j}$ 去更新 $f_{i+1,j}$ 和 $f_{i+1,j \bmod A_{i+1}}$ ，时间复杂度 $\mathcal{O}(nw)$ ，其中 w 是值域，本题中为 5000。

于是第一问就是容易的了，找到最大的满足 $f_{n,i} = \text{True}$ 且小于 A_n 的 i 即可。

怎么做第二问，也就是方案数呢？先用类似第一问的思路走：设 $g_{i,j}$ 为考虑了 A_1 到 A_i ，经过了其中若干值后得到 j 的方案数，这时发现一个问题就是状态不能很好地转移，主要原因在状态设计中这个“经过其中若干值”的范围太宽泛了，于是改变策略为不断地填数，设 $g_{i,j}$ 表示已经填了 A_1 到 A_i ，依次经过这些数后得到的数字是 j 的方案数。

转移时要考虑下一个值可以填什么：那些小于等于 j 的值当然没有被填过，那些大于 j 的值已经被填过了 i 个。为了方便说明，我们开一个桶并前缀和一下，即设 C_i 为 A 中小于等于 i 的数的个数，转移就是：

$$g_{i+1,j} \leftarrow g_{i,j} \times (n - C_j - i)$$

$$g_{i+1,j \bmod A_k} \leftarrow g_{i,j}, A_k \leq j$$

时间复杂度是 $\mathcal{O}(n^2w)$ 的，还不足以通过，我们还要优化掉一个 $\mathcal{O}(n)$ 。

注意到我们记录这个 i 是没什么用的，因为我们填到最后一定是不能再填的，在将 j 变成 $j \bmod A_k$ 时，我们直接把那些新增的在区间 $[j \bmod A_k, j)$ 中的 A 的贡献算掉。

设 g_i 表示当前值为 i ，那些大于 i 的 A 都已经选好了位置的方案数，初始值为 $g_x = \binom{n}{C_{A_1} - C_x} \times (C_{A_1} - C_x)!$ ，即先让那些大于 x 的 A 随便选位置填，无论怎么模它们，我们的数都是不会变的（因为最开始就小于它们）。

转移呢？枚举下一个有效果的数：

$$g_{i \bmod A_k} \leftarrow g_i \times \binom{C_i - 1}{C_i - C_{i \bmod A_k} - 1} \times (C_i - C_{i \bmod A_k} - 1)!$$

A_k 应不大于 i ，因为那些大于 i 的 A 我们已经考虑完了。这里二项式系数的上面是 $C_i - 1$ 而不是 C_i 的原因是我们已经确定了 A_k 的位置，而 A_k 是满足条件的，要将 A_k 从剩下待排的数中删去。

整理一下其实就是

$$g_{i \bmod A_k} \leftarrow g_i \times \frac{(C_i - 1)!}{C_{i \bmod A_k}!}$$

预处理一下阶乘和逆元即可做到 $\mathcal{O}(nw)$ ，注意转移顺序，最终答案即为 g_{res} ，其中 res 为第一问答案。

CF1837D

定义空序列合法, 括号序列 A 合法当且仅当它可以写成以下两种形式之一:

- (B) , 其中 B 是一个合法的括号序列.
- BC , 其中 B 和 C 均为合法的括号序列.

现在给出一个长度为 n 和括号序列, 要求给每个括号字符染一种颜色, 使得染色后将同一颜色的括号字符从左到右或从右到左提出来的括号序列合法, 最小化颜色数或报告无解, 例如对于括号序列 $((()()))$, 我们可以将前 6 个字符染成颜色 1, 后两个字符染成颜色 2 (颜色为 2 的括号序列可以从右到左提出来成为 $()$, 这是合法的).

$$1 \leq n \leq 2 \times 10^5$$

Solution: CF1837D

首先, 假如左括号数不等于右括号数, 一定无解, 否则回忆一般的判断序列合法的方式: 用一个栈, 从左到右往栈顶加字符, 假如栈顶和当前字符恰为 $()$, 就将它们弹出. 最终栈为空就说明括号序列合法, 否则栈中字符从底到顶一定形如 $))))((((($, 即若干个右括号加上若干个左括号.

这时候本题的解法实际上就有了, 假如有解, 答案一定不大于 2, 因为我们可以跑一遍这个操作, 将弹出来的位置染成颜色 1, 剩下的染成颜色 2, 这样的染色一定合法.

时间复杂度 $\mathcal{O}(n)$.

【CSP-S 2022】数据传输

小 C 正在设计计算机网络中的路由系统.

测试用的网络总共有 n 台主机, 依次编号为 $1 \sim n$. 这 n 台主机之间由 $n-1$ 根网线连接, 第 i 条网线连接个主机 a_i 和 b_i . 保证任意两台主机可以通过有限根网线直接或者间接地相连. 受制于信息发送的功率, 主机 a 能够直接将信息传输给主机 b 当且仅当两个主机在可以通过不超过 K 根网线直接或者间接的相连.

在计算机网络中, 数据的传输往往需要通过若干次转发. 假定小 C 需要将数据从主机 a 传输到主机 b ($a \neq b$), 则其会选择出若干台用于传输的主机 $c_1 = a, c_2, \dots, c_{m-1}, c_m = b$, 并按照如下规则转发: 对于所有的 $1 \leq i < m$, 主机 c_i 将信息直接发送给 c_{i+1} .

每台主机处理信息都需要一定的时间, 第 i 台主机处理信息需要 v_i 单位的时间. 数据在网络中的传输非常迅速, 因此传输的时间可以忽略不计. 据此, 上述传输过程花费的时间为 $\sum_{i=1}^m v_{c_i}$.

现在总共有 Q 次数据发送请求, 第 i 次请求会从主机 s_i 发送数据到主机 t_i . 小 C 想要知道, 对于每一次请求至少需要花费多少单位时间才能完成传输.

$$1 \leq n, Q \leq 2 \times 10^5, 1 \leq K \leq 3$$

Solution: 【CSP-S 2022】数据传输

注意到 $1 \leq K \leq 3$, 这是一个很强的性质, 我们尝试从 K 等于 1 或 2 的情况入手.

当 K 等于 1 时, 每次传送必须经过一条边到达另一个端点, 这个路径是唯一确定的: 树上 s_i 到 t_i 的那条路径, 可以用倍增或树链剖分解决.

当 K 等于 2 时, 我们可不可能跳出树上 s_i 到 t_i (下文中直接写为 s 和 t) 呢? 手玩一下发现是不可能的: 跳出链后, 我们只能跳回链上原先点祖先的祖先, 而这本来是可以一步解决的. 这种情况只需要一个平凡的 dp.

难点在于 K 等于 3 的情况. 这时我们是可能跳出链的: 例如链上点的权值都很大, 但是它们都挂着一个不在链上的权值很小的点, 我们就可以通过这些挂在链上的点跳到链的结尾. 一个重要的转化是: 原本 K 的限制实际上可以写成分层图的形式, 即将点 u 拆为 u_0, u_1, u_2 , 点的第二维表示走到该点时已经连续走过了多少个没

有记录权值的点, 那么若有 u 到 v 的边, 就从 u_i 向 v_0 连一条权值为 val_v 的边, 从 u_i 向 v_{i+1} 连一条权值为 0 的边, 这样, 从 s_0 到 t_i 的最短路再加上 $val_s + val_t$ 就是答案.

重点在于这种转换方式是适用于一般的图的, 而我们解决的是树上问题! 可不可以利用树的性质避免每次求最短路呢? 树上倍增.

具体地, 设 $dist_{u,d,i,j}$ 表示从点 u_i 跳到 u 的第 2^d 级别祖先 (设为 anc) 的状态 ans_j 的最短路 (不包含起点, 但若 $j=0$ 则包含终点), 那么答案路径可以拆成 s 到 $lca_{s,t}$ 的路径和 t 到 $lca_{s,t}$ 的路径. 假如我们已经求出了两者的答案 (即两条链末跳到 anc_0, anc_1, anc_2 的最短路), 可以合并出整个链的答案吗?

设 s 到 $lca_{0,1,2}$ 的最短路为 $ds_{0,1,2}$, t 到 $lca_{0,1,2}$ 的最短路为 $dt_{0,1,2}$, 那么若 ds_i 与 dt_j 合并, 假如 $i=j=0$, 多算了一个 lca 的权值, 减掉即可, 假如 $i+j > K$, 我们就少算了一个 lca 的权值, 加上即可, 否则就是两者简单相加.

枚举 i, j , 维护最小的答案即可, 现在只需考虑如何求出 $dist$, 求出 $dist$ 后, 一条树上的链是容易倍增求出的.

显然 $dist_{u,d,i,j}$ 可以从 $dist_{u,d-1,i,k} + dist_{anc,d-1,k,j}$ 转移, 枚举 k 即可, 现在只需要得到 $dist_{u,0,i,j}$, 而这也是容易的, 当 $j=0$ 时即为 val_{father_u} , $i=0, j=1$ 或 $i=1, j=2$ 时为 0, 当 $i=j=2$ 时为与 $father_u$ 相邻 (包括它本身) 的点的权值的最小值.

总时间复杂度 $\mathcal{O}(n \log n)$, 本题还可以用动态 dp 解决, 这里不再给出具体做法.

CF1830B

给出长度为 n 的序列 a, b , 求满足 $1 \leq i < j \leq n$ 且 $a_i \times a_j = b_i + b_j$ 的数对 (i, j) 的数量.
 $2 \leq n \leq 2 \times 10^5, 1 \leq a_i, b_i \leq n$

Solution: CF1830B

看起来是不能 log 做的, 考虑根号做法: 由于 $b_i + b_j \leq 2n \leq 4 \times 10^5$, 因此若 $a_i > \sqrt{2n}$, 那么另一个 a_j 必然小于 $\sqrt{2n}$, 因此考虑如下分治:

记录数组 $f_{x,y}$ 表示 $a_i = x, b_i = y$ 的 i 的个数, 其中 $1 \leq x \leq \sqrt{n}$, 那么在统计 (i 从 1 到 n 向 f 贡献) 的过程中, 枚举另一个 a_j 的值 y (从 1 到 $\sqrt{2n}$), 将答案加上 $f_{y, a_i \times y - b_i}$ (这里的 f 不是最终的 f , 是只算了之前元素的临时 f 数组, 这样就可以保证同一对数不会在顺序不同的时候算两次).

$\mathcal{O}(n)$ 统计完 f 数组后, 考虑所有 $a_i > \sqrt{2n}$ 的 i , 枚举另一个 a_j 的值 y (同样从 1 到 $\sqrt{2n}$), 将答案加上 $f_{y, a_i \times y - b_i}$ (这里的 f 为最终的 f).

时间复杂度为 $\mathcal{O}(n\sqrt{n})$.

CF1830C

给定 n, k 和 k 组区间 $[l_i, r_i]$, 求长度为 n 且满足区间 $[l_i, r_i]$ 内的子括号序列均合法的合法括号序列的个数, 答案对 998244353 取模.
 $1 \leq n \leq 3 \times 10^5, 0 \leq k \leq 3 \times 10^5$

Solution: CF1830C

首先, 若没有这些区间的限制, 需要求出 n 个括号组成的合法括号序列的个数, 设为 f_n , 这等价于第 $\frac{n}{2}$ 项 Catalan 数, 我们在数学一部分中已经详细讲过, 第 n 项 Catalan 数等于 $\frac{1}{n+1} \binom{2n}{n}$. 现在加入了这么多限制区间, 尝试将这些限制减弱一些, 或者让它们变得更好处理.

挖掘题目中的性质, 若对于两个限制区间 $[l_1, r_1], [l_2, r_2]$ 满足 $l_1 \leq l_2 \leq r_2 \leq r_1$, 即前一个区间完全包含后一个区间, 那么这两个限制可以被拆分为:

- $[l_2, r_2]$ 这个子括号序列合法

- $[l_1, l_2 - 1] \cup [r_2 + 1, l_2]$ 这个子括号序列合法

这两组限制是互为充要条件的，证明平凡。

若对于两个限制区间 $[l_1, r_1], [l_2, r_2]$ 满足 $l_1 < l_2 \leq r_1 < r_2$ ，即这两个限制区间有交集，那么这两个限制等价于

- $[l_2, r_1]$ 是合法的
- $[l_1, l_2 - 1]$ 和 $[r_1 + 1, r_2]$ 均是合法的

证明同样平凡。

注意到，对这 k 个限制区间这样处理后，所有限制都没有了交集（注意 $[1, n]$ 这个限制是题目中给出的，需要我们自己加上），假如我们已经通过某种方式处理好了这些区间，设有 k' 个，第 i 个处理好的区间表示某个长度为 a_i 的子序列是合法的，那么答案即为

$$\prod_{i=1}^{k'} f_{a_i}$$

现在只需快速地处理好这些区间。

直接处理很难，考虑哈希，对每个题中给出的限制区间（包含 $[1, n]$ ）赋一个随机权值，开一个 b 数组，初始时全为 0，每加入一个限制区间 $[l_i, r_i]$ ，设随机到的权值为 v ，那么将 b_{l_i} 变为 $b_{l_i} \text{ xor } v$ ，将 b_{r_i+1} 变为 $b_{r_i+1} \text{ xor } v$ ，最后做一遍前缀异或和，得到数组 b' ，那么若 $b'_i = b'_j$ ，我们就可以说它们在同一个限制区间中，扫一遍统计即可。

若用 `std::map` 处理 Hash 和统计答案，时间复杂度为 $\mathcal{O}(k \log k + n)$ ，其中 $\mathcal{O}(n)$ 为预处理 f 的时间复杂度。

本题的另一个确定性做法是将 $[l_i, r_i]$ 的限制转化为 $s_{l_i} = s_{r_i}$ 且 $\forall j \in [l_i, r_i], s_j \geq s_{r_i}$ ，线段树优化建图后对每个强连通分量分别处理。

CF1830D

给定一个 n 个点，编号为 1 到 n 的树，你可以给每个点 u 分别赋一个权值 $v_u \in \{0, 1\}$ ，但需要最大化

$$\sum_{1 \leq i < j \leq n} \text{mex}\{v_i, v_j\}$$

其中 $\text{mex } A$ 表示最小的没有出现在集合 A 中的自然数，输出这个最大权值和。

$1 \leq n \leq 2 \times 10^5$

Solution: CF1830D

一个自然的想法是尽量让更多的 $\text{mex}\{v_i, v_j\}$ 等于 2，这可以通过将树二分图染色，把黑点和白点分别染成 0 和 1 实现，设黑点的数量为 c ，那么这样做的权值和为

$$n^2 - n + \max\{c, n - c\}$$

这种做法看起来已经很优秀了，但是它不一定是最优的，我们可以构造一棵形态如一条边的两端均挂着很多条边的树，这样，最优解就是将边的两端赋为 1，剩下挂在这两个点上的其他点的权值为 0。

正着考虑比较复杂，考虑反向做，初始时让所有点对的 mex 都是 2，这样的答案是 $n(n+1)$ ，但这是达不到的。怎样的点对才会让答案变小呢？路径上点的权值相同，于是，对于每一个点的颜色均为 $c \in \{0, 1\}$ 的极大连通块（设大小为 m ），它会让我们的答案减小 $\frac{n(n+1)}{2} \times (j+1)$ 。

转化后的问题看起来就很树上 dp，设 $f_{u,k,0/1}$ 为在以 u 为根的子树中， u 染的颜色为 0/1 且它所在的极大同色连通块的大小为 k 时子树让答案减小的最小值，转移很容易。

但是这么做，状态数已经达到了 $\mathcal{O}(n^2)$ ，回顾我们之前的二分图染色，它告诉了我们一个答案的下界，也就是最优解至多减 $\mathcal{O}(n)$ ，因此极大同色连通块的大小一定不会超过 $\mathcal{O}(\sqrt{n})$ ，因此状态数减少为了 $\mathcal{O}(n\sqrt{n})$ 。空间仍然是开不下的，但可以用 `std::vector` 动态地开。

【候选队互测 2022】毛估估就行

有时候,做人不能太斤斤计较.社会上讲究一个让步和妥协,大家都得给自己和对方留一些余地.所以有时候不必太追求精确,毛估估就行.

俗话说得好,“退一步海阔天空”,本题的出题人也想和大家和平相处.本着给选手多送分的原则,这道题你不必求出精确解,毛估估求一下就行啦!如果你的答案和精确值差不多,出题人会假装没看出来区别然后给你满分的!

那么接下来描述这个送分题的题面:有一张 n 个点的无向无权连通图, q 次查询两点之间的最短路长度.但正如前面所说的,只要你与答案差不超过 1,即如果最短路长度是 d ,输出 $d-1, d, d+1$ 之一时,出题人都会假装你算的是对的.

这时候大家可能会问了,这不是普及组题目吗?我刚学 OI 的时候就做过了!所以出题人决定加大题目的数据范围,但这下出题人都不会做了.不过,你会吗?

对于所有数据,保证 $2 \leq n \leq 8000, 1 \leq q \leq 10^6$

Solution: 【候选队互测 2022】毛估估就行

先考虑一些部分分,当 $n \leq 500$ 的时候,直接上 Floyd 就好了.但其实可以用 `std::bitset` 将单次 BFS 优化至 $O\left(\frac{n^2}{w}\right)$, 因此总时间复杂度变为 $O\left(\frac{n^3}{w} + q\right)$.

这样还是不行,因为我们始终求的是精确解,但实际上我们可以有 1 的误差,也就是全源最短路是不必要的,考虑选出若干个关键点并从它们出发求最短路.

假如我们从 u 出发求得了其最短路数组 d_u , 设点 u 和与它直接相连的点集为 V_u , 那么有如下引理:

引理: 若 x 与 y 之间的最短路经过了点集 V_u 中的某个点, 那么 $d_x + d_y$ 不会大于 x 与 y 之间最短路长度加 2.

分经过 u 与经过其它点两类讨论, 证明显然.

于是我们可以怎么做呢? 选出足够多的点构成点集 Ver , 使得对于任意两个点 u, v 都有 u, v 之间的最短路经过了某个 V_x 并且 $x \in Ver$, 那么对于 u, v 之间的最短路, 我们就可以输出 $\min_{x \in Ver} \{dist_{x,u} + dist_{x,v} - 1\}$.

怎么选出这个点集 Ver 呢? 每次选出一个当前度数最大的点 u 加入 Ver 中, 删除这个点和与其相邻的点, 不断重复这个操作直到没有点为止.

为了保证时间复杂度, 每次选出点并以其为起点 Bfs 后, 我们会真的删除 (打上删除标记) 它和它相邻的点, 在之后的 Bfs 中不经过这些点, 下面我们证明这样 Bfs 的总时间复杂度是 $O(n^2)$ 的, 因此总时间复杂度是 $O(n^2 + nq)$ 的, 可以拿到 70 分.

每次 Bfs 的复杂度为 $O(n + m)$, 其中 m 为 bfs 时的边数. 由于选择的点 u 为度数最大的一个点, 于是 $deg_u \geq \frac{2m}{n}$. 设每一次删除时边数分别为 m_1, m_2, \dots, m_s , 由于最多删除 n 个点, 于是

$$\sum_i \frac{2m_i}{n} \leq n$$

所以时间复杂度不超过

$$O\left(n^2 + \sum_{i=1}^s m_i\right) = O(n^2)$$

这样还是不够, 虽然预处理的时间复杂度已经足够优秀, 但是单次询问的时间复杂度过高了, 考虑平衡一下复杂度, 让 $|Ver| \leq B$, 具体来说, 在 $|Ver| = B$ 之后, 我们对剩下的点和边直接进行 $O(n^2 + nm)$ 的全源 Bfs, 看看剩下的点数和边数在什么数量级. 由于每次删除的点的度数单调不增, 所以此时剩下的点的度数均不超过 $\frac{n}{B}$, 因此边数为 $O\left(\frac{n^2}{B}\right)$, 因此对剩下的点全源 Bfs 的时间复杂度是 $O\left(\frac{n^3}{B}\right)$ 的.

将预处理和查询的时间复杂度拼起来, 总时间复杂度为 $O\left(\frac{n^3}{B} + Bq\right)$, 用一个基本不等式, 取 $B = \Theta\left(n^{\frac{3}{2}}q^{-\frac{1}{2}}\right)$ 即可做到 $O(n^{\frac{3}{2}}q^{\frac{1}{2}})$ 的时间复杂度, 可以拿到 100 分.

CF1839D

n 个颜色不同 (编号为 1 到 n) 的小球排在一列, 设第 i 个小球的颜色为 C_i . 你的手中有 $k \geq 1$ 个颜色为 0 的球, 你希望通过这 k 个球用最小的代价将 n 个球的颜色编号升序排列, 具体来说, 你可以执行下面两种操作:

- 选出一个位置 (两个相邻小球之间或开头或结尾), 不花费任何代价往里放进一个颜色为 0 的球. 由于你只有 k 个这样的球, 所以这种操作最多进行 k 次.
- 选出一个小球 (它的左右必须有至少一个颜色为 0 的球), 用 1 的代价将其插入到任意一个位置, 这个操作可以进行无限次.

你的任务是对于每个 $1 \leq k \leq n$ 求出当你有 k 个颜色为 0 的小球时的最小花费.

$1 \leq n \leq 500$

Solution: CF1839D

考虑那些最终没有被操作 2 操作过的数, 它们的相对位置是不变的, 那么它们一定构成了原序列的一个上升子序列. 能不能让那些被操作 2 操作的数只被操作一次呢? 答案是可以的: 直接把它换到它应该在的位置即可 (不变的数字构成了若干个区间, 我们将操作 2 操作的数直接换到某个这样的区间的中间或开头或结尾, 它一定不需要再被操作).

在上面的分析中, 我们将这些没有被操作过的数看成了若干个区间, 类似的, 将那些被操作过的数也看成若干个区间, 我们只需要往每个区间里放一个 0 就够了.

因此, k 的限制相当于: 选出不超过 k 个区间, 使得删除这些区间中的数后原数列严格上升, 求这些区间长度和的最小值.

很 dp 的形式, 设 $f_{n,k}$ 为只考虑前 n 个数并且钦定最后这个数不在区间中, 区间长度的最小值.

转移是 $\mathcal{O}(n)$ 的, 因此总时间复杂度为 $\mathcal{O}(n^3)$, 可以用线段树优化至 $\mathcal{O}(n^2 \log n)$.

CF1839E

Alice 和 Bob 在玩游戏, 这里有一个长度为 n 的数列 A , 在游戏的每一轮中, Alice 先选出其中的一个非零数 (如果不能选择, Alice 输), Bob 在这之后选出另外一个非零数 (同样的, 如果选不出这样的数, Bob 输), 之后将这两个数均减去它们的最小值 (这意味着至少有一个数变成了 0).

这样的操作只能进行有限次, 现在你要选择你是先手还是后手, 并赢下交互库.

$1 \leq n, A_i \leq 300$

Solution: CF1839E

先给出结论: 后手必胜当且仅当所有数可以被划分为和相等的两部分, 接下来我们给出证明.

首先, 若它们可以被划分, 后手必胜, 这是因为每次选出不和先手在同一部分的任意一个剩下的数, 就可以保证两部分的和始终相等, 直到最后变为 0.

若它们不可以被划分, 先手必胜, 只是因为将 1 到 n 看作结点, 若某一轮操作操作了 A_i 和 A_j , 就在 i, j 之间连一条边, 这样连出来的图一定是一棵树, 而树一定是二分图, 那么由于最终所有数都变成了 0, 二分图的左部和右部的和必然相等, 这就找出了一个划分, 矛盾.

有了充要条件, 构造也是容易的, $f_{i,x}$ 表示只考虑前 i 个数, 是否可以划分为两部分使其中一部分的和为 x , 那么后手必胜当且仅当所有数的和是偶数并且 $f_{n,S/2}$ 为真, 其中 $S = \sum_{i=1}^n A_i$. 根据 f 数组可以从后往前倒推出两部分的具体元素, 剩下的构造是平凡的.

先手的构造方案就更容易了: 每次随便选个数就行. 这是因为随便选两个数后, 剩下的数仍不能被正确划分 (设选的两个数值为 $a > b$, 那么设 $a - b$ 在剩下的左部, 这说明将 $a - b$ 删掉, 将 a 放到左部, b 放在右部同样可行).

时间复杂度瓶颈在于 dp , 为 $\mathcal{O}(n^2w)$, 其中 w 是 A 的值域.

CF1830E

给定一个 1 到 n 的排列 p , 定义一次 **bully swap** 是这样的一组操作:

- 找到一个下标 i 使得 $p_i \neq i$ 并且这个 p_i 是满足条件的下标对应的值中最大的那个.
- 找到另一个下标 j 满足 $j > i$ 并且 p_j 是所有满足条件的下标对应的值中最小的那个.
- 交换 p_i 和 p_j

可以证明, 经过有限次 **bully swap** 后 p 将会升序排列, 定义 $f(p)$ 为排列 p 需要多少次 **bully swap** 才可以有序.

给定初始的排列 p 和 q 组操作, 每组操作会交换某对 p_i 和 p_j , 操作不会撤销 (即会影响之后的操作), 你的任务是在每次操作后求出 $f(p)$.

$2 \leq n \leq 5 \times 10^5, 1 \leq q \leq 5 \times 10^4$

Solution: CF1830E

若排列满足 $p_n = n$, 那么在之后的操作中我们完全不会考虑它, 可以直接将它删掉并更新 n , 现在的 **bully swap** 找到的下标 i 一定满足 $p_i = n$.

这个 **bully swap** 看起来十分诡异, 但有一个很重要的性质:

设某次 **bully swap** 操作的两个下标是 i 和 j ($i < j$), 那么一定有 $p_j \leq i$, 即往左交换过的元素不会再返回来.

可以这样证明: 在 $[i+1, n]$ 之间的数如果都大于 i , 由于 n 一定不在这个区间, 那么大于 i 的数就只剩下 $n-1-i$ 个, 是填不满这个区间的, 于是一定有一个数是不大于 i 的.

考虑一次交换操作导致什么量发生了变化: 设 $i-j=k$, 那么这次交换操作让 $\sum_{i=1}^n |i-p_i|$ 减少了 $2k$, 让区间的逆序对数减少了 $2k-1$. 设区间逆序对数为 D , $\sum_{i=1}^n |i-p_i|$ 为 C , 那么最终排列有序时有 $C=0, D=0$, 那么操作的次数就是初始的 $C-D$.

C 是容易单次 $\mathcal{O}(1)$ 维护的, 求每次操作后的 D 是一个动态逆序对问题, 可以用很多经典方法解决, 如果用 BIT 加 CDQ 分治, 时间复杂度为 $\mathcal{O}(n \log^2 n)$

【NOIP】2022 建造军营

A 国与 B 国正在激烈交战中, A 国打算在自己的国土上建造一些军营.

A 国的国土由 n 座城市组成, m 条双向道路连接这些城市, 使得任意两座城市均可通过道路直接或间接到达. A 国打算选择一座或多座城市 (至少一座), 并在这些城市上各建造一座军营.

众所周知, 军营之间的联络是十分重要的. 然而此时 A 国接到情报, B 国将会于不久后袭击 A 国的一条道路, 但具体的袭击目标却无从得知. 如果 B 国袭击成功, 这条道路将被切断, 可能会造成 A 国某两个军营无法互相到达, 这是 A 国极力避免的. 因此 A 国决定派兵看守若干条道路 (可以是一条或多条, 也可以一条也不看守), A 国有信心保证被派兵看守的道路能够抵御 B 国的袭击而不会被切断.

A 国希望制定一个建造军营和看守道路的方案, 使得 B 国袭击的无论是 A 国的哪条道路, 都不会造成某两座军营无法互相到达. 现在, 请你帮 A 国计算一下可能的建造军营和看守道路的方案数共有多少. 由于方案数可能会很多, 你只需要输出其对 $1,000,000,007 (10^9+7)$ 取模的值即可. 两个方案被认为是不同的, 当且仅当存在至少一座城市在一个方案中建造了军营而在另一个方案中没有, 或者存在至少一条道路在一个方案中被派兵看守而在另一个方案中没有.

$1 \leq n \leq 5 \times 10^5, n-1 \leq m \leq 10^6, 1 \leq u_i, v_i \leq n, u_i \neq v_i$.

Solution: 【NOIP】2022 建造军营

首先,很自然地将边双连通分量缩起来,原图变成一棵树,边双中的边守不守都是无所谓的,所以这些边对答案的贡献是 $\times 2^x$, 其中 x 是所有边双中边的个数.

先看树上的情况,自然地想到树形 DP, 设 f_u 为只考虑以 u 为根的子树中的点时的合法方案数, 发现不能转移, 考虑加状态: 设 $f_{u,0}$ 为只考虑以 u 为根的子树中的点, 并且点 u 必选的方案数, $f_{u,1}$ 为只考虑以 u 为根的子树中的点, 选的点数不能为 0 并且 u 不能选的合法方案数, 这次可以转移了.

那么, 对于一种点的选法, 考虑在它们的 lca 处计算贡献, 设 g_u 为在点 u 处计算的贡献, $g_u = f_{u,0} + f_{u,1}$ 吗? 不是这样的. 在 $f_{u,1}$ 中, 可能存在某些点的选法, 它们的 lca 不是 u , 而是 u 的某个后代 (例如 u 的某个儿子是 v , 我们选的点全在 v 的子树中), 我们应当减去这些方案.

这也是容易的, 枚举它们的 lca 在哪个儿子的子树中, 减掉即可.

在这个子树之外的边可以任意选择守或者不守, 贡献乘一个 2 的若干次幂即可. 将所有 g_u 乘上在这个子树之外的边的贡献全部加起来就是最终答案.

回到非树上的情况, 我们缩点完后, 只需将单个点选择的情况用一个 2 的幂代替正常树的 1 跑 DP 即可.

【NOIP2022】比赛

小 N 和小 O 会在 2022 年 11 月参加一场盛大的程序设计大赛 NOIP! 小 P 会作为裁判主持竞赛. 小 N 和小 O 各自率领了一支 n 个人的队伍, 选手在每支队伍内都是从 1 到 n 编号. 每一个选手都有相应的程序设计水平. 具体的, 小 N 率领的队伍中, 编号为 i ($1 \leq i \leq n$) 的选手的程序设计水平为 a_i ; 小 O 率领的队伍中, 编号为 i ($1 \leq i \leq n$) 的选手的程序设计水平为 b_i . 特别地, $\{a_i\}$ 和 $\{b_i\}$ 还分别构成了从 1 到 n 的排列.

每场比赛前, 考虑到路途距离, 选手连续参加比赛等因素, 小 P 会选择两个参数 l, r ($1 \leq l \leq r \leq n$), 表示这一场比赛会邀请两队中编号属于 $[l, r]$ 的所有选手来到现场准备比赛. 在比赛现场, 小 N 和小 O 会以掷骰子的方式挑选出参数 p, q ($l \leq p \leq q \leq r$), 只有编号属于 $[p, q]$ 的选手才能参赛. 为了给观众以最精彩的比赛, 两队都会派出编号在 $[p, q]$ 内的、程序设计水平值最大的选手参加比赛. 假定小 N 派出的选手水平为 m_a , 小 O 派出的选手水平为 m_b , 则比赛的精彩程度为 $m_a \times m_b$.

NOIP 总共有 Q 场比赛, 每场比赛的参数 l, r 都已经确定, 但是 p, q 还没有抽取. 小 P 想知道, 对于每一场比赛, 在其所有可能的 p, q ($l \leq p \leq q \leq r$) 参数下的比赛的精彩程度之和. 由于答案可能非常之大, 你只需要对每一场答案输出结果对 2^{64} 取模的结果即可.

$1 \leq n, Q \leq 2.5 \times 10^5$, $1 \leq l_i \leq r_i \leq n$, $1 \leq a_i, b_i \leq n$ 且 $\{a_i\}$ 和 $\{b_i\}$ 分别构成了从 1 到 n 的排列.

Solution: 【NOIP2022】比赛

首先根据套路, 将询问离线下来后按照 r 升序排列, 依次处理这些询问.

区间问题优先考虑线段树, 由于询问按照 r 升序, 所以可以动态地移动右端点, 每次向右移动 1 并维护一个 C 数组. 设当前右端点移动到了 r , 将要处理右端点恰好为 r 的询问, 用 C_l 表示左端点为 l , 右端点在 $[l+1, r]$ 的所有区间的贡献, 那么一次询问的答案就是一个 C 数组的区间和, 假如能够用线段树使得在 $\mathcal{O}(\log n)$ 的时间内维护出右端点右移一位后的 C 数组, 问题就已经在 $\mathcal{O}(n \log n + q \log n)$ 的时间复杂度内解决了.

难点在于维护这棵线段树, 我们再维护两个数组 X, Y , 其中 X_l 表示 $\max_{i=l}^r \{a_i\}$, Y_l 表示 $\max_{i=l}^r \{b_i\}$ (这里的 r 就是我们不断动态移动的右端点), 那么当右端点移动到 r 时, 我们维护出 X 数组, C 的变化其实就是让 C_i 加上 $X_i \times Y_i$.

不难想到用单调栈维护 X, Y , 那么将右端点右移后, 这两个数组的变化就是区间推平为某个值, 问题变为区间推平、区间中的每个元素加对应的 $X_i \times Y_i$ 、区间历史和, 这可以用懒标记维护.

Luogu P3765 总统选举

黑恶势力的反攻计划被小 C 成功摧毁，黑恶势力只好投降。秋之国的人民解放了，举国欢庆。此时，原秋之国总统因没能守护好国土，申请辞职，并请秋之国人民的大救星小 C 钦定下一任。

作为一名民主人士，小 C 决定举行全民大选来决定下一任。为了使最后成为总统的人得到绝大多数人认同，小 C 认为，一个人必须获得超过全部人总数的一半的票数才能成为总统。如果不存在符合条件的候选人，小 C 只好自己来当临时大总统。为了尽可能避免这种情况，小 C 决定先进行几次小规模预选，根据预选的情况，选民可以重新决定自己选票的去向。

由于秋之国人数较多，统计投票结果和选票变更也成为了麻烦的事情，小 C 找到了你，让你帮他解决这个问题。

秋之国共有 n 个人，分别编号为 $1, 2, \dots, n$ ，一开始每个人都投了一票，范围 $1 \sim n$ ，表示支持对应编号的人当总统。

共有 m 次预选，每次选取编号 $[l_i, r_i]$ 内的选民展开小规模预选，在该区间内获得超过区间大小一半的票的人获胜。如果没有人获胜，则由小 C 钦定一位候选者获得此次预选的胜利（获胜者可以不在该区间内），每次预选的结果需要公布出来，并且每次会有 k_i 个人决定将票改投向该次预选的获胜者。

全部预选结束后，公布最后成为总统的候选人。

对于 100% 的数据， $1 \leq n, m \leq 5 \times 10^5$ ， $\sum k_i \leq 10^6$ ， $1 \leq l_i \leq r_i \leq n$ ， $1 \leq s_i \leq n$ 。

Solution: Luogu P3765 总统选举

这里介绍一种求这类特殊众数的方法：摩尔投票法。它适用于求一个数列的**绝对众数**，其中绝对众数是指出现次数大于元素个数一半的数。

摩尔投票法的流程是这样的，我们挨个考虑数列中的元素，记录两个变量 c 和 v ， c 表示元素数值是 v 的数的个数。每考虑到下一个数时，若这个数和当前记录的 v 不同，我们就将 c 变为 $c - 1$ ，即这两个数互相抵消了一次出现，否则让 c 变为 $c + 1$ ，意味着这个数又出现了一次。

形式化地说，我们每次取出两个数，假如相同就放回，假如不同就相互抵消，直到最后。

扫完所有数之后，若这个数列存在绝对众数，那么 v 就是这个绝对众数。

这个算法是很直观的：不同的数字之间互相抵消，假如存在绝对众数，它一定是怎样都抵消不完的，一定会留到最后。

刚才的分析有着“数列存在绝对众数”的前提，若数列本身就没有绝对众数，摩尔投票法最终可能会求出一个错误答案，也就是说摩尔投票法只是给出了一个绝对众数的可能性。

回到本题，求区间选出的候选人就是求区间的绝对众数，由于摩尔投票法维护的信息可以合并（两个区间的投票结果可以直接合并为总结果，方法为考虑 v 是否相等以及哪个的票数，即 c 更多），因此可以用线段树维护投票结果，每次对区间查询，时间复杂度为 $\mathcal{O}(\log n)$ 。

注意本题不保证区间绝对众数存在，因此我们还应该判断一下，方法是用 n 棵平衡树，第 i 棵平衡树维护了所有投了 i 的人的位置信息，那么我们在投票结果那个人的平衡树上看位置在 $[l, r]$ 之间的结点个数的二倍是否大于 $r - l + 1$ 即可。

由于 $\sum k_i \leq 10^6$ ，因此投票的修改是可以暴力做的（线段树上的一次修改和平衡树上的一个删除、一个插入）。

总时间复杂度 $\mathcal{O}(m \log n + \sum k_i \log n)$

【NOI2014】购票

今年夏天, NOI 在 SZ 市迎来了她三十周岁的生日。来自全国 n 个城市的 Oler 们都会从各地出发, 到 SZ 市参加这次盛会。

全国的城市构成了一棵以 SZ 市为根的有根树, 每个城市与它的父亲用道路连接。为了方便起见, 我们将全国的 n 个城市用 $1 \sim n$ 的整数编号。其中 SZ 市的编号为 1。对于除 SZ 市之外的任意一个城市 v , 我们给出了它在这棵树上的父亲城市 f_v 以及到父亲城市道路的长度 s_v 。

从城市 v 前往 SZ 市的方法为: 选择城市 v 的一个祖先 a , 支付购票的费用, 乘坐交通工具到达 a 。再选择城市 a 的一个祖先 b , 支付费用并到达 b 。以此类推, 直至到达 SZ 市。

对于任意一个城市 v , 我们会给出一个交通工具的距离限制 l_v 。对于城市 v 的祖先 A , 只有当它们之间所有道路的总长度不超过 l_v 时, 从城市 v 才可以通过一次购票到达城市 A , 否则不能通过一次购票到达。

对于每个城市 v , 我们还会给出两个非负整数 p_v, q_v 作为票价参数。若城市 v 到城市 A 所有道路的总长度为 d , 那么从城市 v 到城市 A 购买的票价为 $dp_v + q_v$ 。

每个城市的 Oler 都希望自己到达 SZ 市时, 用于购票的总资金最少。你的任务就是, 告诉每个城市的 Oler 他们所花的最少资金是多少。

对于所有数据, $n \leq 2 \times 10^5, 0 \leq p_v \leq 10^6, 0 \leq q_v \leq 10^{12}, 1 \leq f_v < v, 0 < s_v \leq l_v \leq 2 \times 10^{11}$, 且任意城市到 SZ 市的总路程长度不超过 2×10^{11} 。

Solution: 【NOI2014】购票

首先, 设点 u 到树根 (点 1) 的距离为 dep_u , 对于点 u 的答案 ans_u , 枚举能够一次购票达到的祖先 anc , 那么有

$$ans_u = \min\{ans_{anc} + (dep_u - dep_{anc}) \times p_u + q_u\} = \min\{ans_{anc} - dep_{anc} \times p_u\} + dep_u \times p_u + q_u$$

那么可以一次树形 dp 在 $\mathcal{O}(n^2)$ 的时间内解决这个问题。状态转移很斜率优化, 考虑对于两个可以一次到达的祖先 x, y , 其中 $dep_x < dep_y$, 那么决策 x 优于决策 y 当且仅当

$$ans_x - dep_x \times p_u < ans_y - dep_y \times p_u$$

整理得

$$p_u < \frac{ans_x - ans_y}{dep_x - dep_y}$$

因此可以用斜率优化, 在 dfs 的过程中, 设当前所在点为 u , 考虑用根到 u 这条链来更新以 u 为根的子树中不含 u 的所有点的答案: 将这些点按照 $dep - l$ 降序排列 (也就是按照可以跳到更高点的点排在后面), 每次往决策点集合中加入链尾的一些点, 在维护的上凸包中二分得到决策点即可。

但是这样做的时间复杂度是不对的, 因此可以在外层套一个点分治, 时间复杂度 $\mathcal{O}(n \log^2 n)$ 。

UOJ #10. 【UTR #1】pyx 的难题

pyx 喜欢出题给 cyb 做。为了增加难度，他有可能在 cyb 还没做完前面的题的时候就把新题目出给他。

假设 pyx 总共要给 cyb 出 n 道题，其中第 i 道题是在 t_i 时刻出给 cyb 的，cyb 需要花 s_i 秒才能 AC 此题。

pyx 比较高产，出题时刻可能相同（也就是 pyx 可能同一时刻丢出了动态仙人掌和可持久化带插入仙人掌路径 k 大值给 cyb）。

按照 cyb 的个人喜好，他给 n 道题目标上了互不相同的优先级 p_i 。

cyb 做题过程是这样的：

- cyb 收到题目是瞬间的事情。
- 每次，cyb 先收到所有 pyx 新出的题。
- 接着，cyb 把现在所有待解决的题目按照优先级瞬间排序。（当然优先级是互不相同的）
- 然后，cyb 会花恰好 1 秒钟时间在那个优先级最高的题，然后解决该题需要花的时间减少 1 秒。如此循环。

由于是神国之间的交流，十分有记录的意义，需要载入史册，这个光荣的任务就交给修电脑的 oier 们了。

现在给你 n 道题的 t_i, s_i, p_i 。然而不幸的是，由于奇怪的原因，你一不小心把某一道题的优先级弄丢了。

你为了掩饰自己的错误，通过不断打听，得知了 cyb 在时刻 T 时 AC 了这道优先级未知的题目。（即第 T 秒末，第 T 秒结束的时刻）

你想试着推测出优先级，然而很显然，方案可能不止一种。你决定编造这个题目的优先级，使得 cyb 仍会在时刻 T AC 此题。

除此之外，你还要帮 cyb 算出所有题目被解决的时刻，以展示你是多么的负责。

也就是说，你要求出一个可行的优先级，并求出在这个优先级下，所有题目被解决掉的时刻。

保证 $0 \leq t_i \leq 10^9, 1 \leq s_i, p_i \leq 10^9, 1 \leq T \leq 10^{15}$ 。（ p_x 例外， $p_x = -1$ ）同时，保证有解。

Solution: UOJ #10. 【UTR #1】pyx 的难题

将所有出现过的优先级排序，那么选择两种优先级之间的所有优先级都是等价的，枚举这个优先级，将所有题按照 t 升序，暴力模拟可以得到 60 分。

注意到我们如果将 p_x 变小，那么它被解决掉的时刻只会变小，于是可以简单做到 $\mathcal{O}(n^2 \log n)$ 。

实际上，我们可以用一个堆来优化模拟的过程，相邻两个题之间的一段时间是可以合并处理的，优化后再加上二分答案，时间复杂度是 $\mathcal{O}(n \log^2 n)$ 。

二分是可以去掉的，因为我们保证有解，在模拟的过程中，假如到了 t_x 时刻，那么可以 $\mathcal{O}(1)$ 得出这个时刻到 T 之间有多少时间是用来处理别的题目的，将这段时间内的时间也加进堆中，按照优先级不断处理堆顶，最终一定正好处理完某些题目，那么 p_x 就应当小于那些处理的题目的优先级，并且大于堆中剩下题目的优先级，时间复杂度 $\mathcal{O}(n \log n)$ 。

UOJ #11. 【UTR #1】ydc 的大树

ydc 有一棵 n 个结点的黑白相间的大树，从 1 到 n 编号。

这棵黑白树中有 m 个黑点，其它都是白点。

对于一个黑点我们定义他的好朋友为离他最远的黑点。如果有多个黑点离它最远那么都是它的好朋友。两点间的距离定义为两点之间的最短路的长度。

现在你要摧毁一个白点。

摧毁后有一些黑点会不高兴。一个黑点不高兴当且仅当他不能到达任何一个在摧毁那个白点前的好朋友。

请你最大化不高兴的黑点数，并求出最大点数以及方案数。

$n \leq 10^5$

Solution: UOJ #11. 【UTR #1】ydc 的大树

看到树上的类似“最远距离”这样的东西，很自然地想到树的直径。找出黑点之间的直径（即两端点都是黑点的最长路径），依据树网的核一题，将这条直径的中心提出来（假如中心是边，任意提出这条边的一个端点）作为树的根。

这样做有什么用呢？对于树上的任意一黑点，可以证明，它到距离它最远的黑点必定经过中心，也就是我们钦定的根，接下来就很舒服了。

做一次 dfs，对于每个白点，我们想要知道删除它之后会有多少黑点不高兴。首先，它子树中的黑点一定会不高兴，其次，若不在它子树的某个黑点也会不高兴，当且仅当这个不高兴的黑点得到的所有最长路径都伸进了子树中，通过预处理子树中黑点最大深度、子树中最大深度黑点的个数以及子树中黑点的个数，这部分可以方便地得到。

时间复杂度 $\mathcal{O}(n)$ ，但是细节很多。

【NOIP2014】解方程

已知多项式方程：

$$a_0 + a_1x + a_2x^2 + \dots + a_nx^n = 0$$

求这个方程在 $[1, m]$ 内的整数解（ n 和 m 均为正整数）。

对于 100% 的数据， $0 < n \leq 100$ ， $|a_i| \leq 10^{10000}$ ， $a_n \neq 0$ ， $m \leq 1000000$ 。

Solution: 【NOIP2014】解方程

诈骗题。方程的次数很高，肯定不是正常做，考虑暴力地去尝试每个数是否是解。

但是这样做复杂度太大了， a_i 是个高精度数，用到的 trick 是考虑将每个 x 带进去后，求式子模大质数的结果，假如模大质数后是 0，我们就认为这个式子是 0，因此高精度就可以避免了。

实测没有卡 998244353，如果不放心可以多取几个模数。

UOJ #21. 【UR #1】缩进优化

小 O 是一个热爱短代码的选手。在缩代码方面，他是一位身经百战的老手。世界各地的 OJ 上，很多题的最短解答排行榜都有他的身影。这令他感到十分愉悦。

最近，他突然发现，很多时候自己的程序明明看起来比别人的更短，实际代码量却更长。这令他感到很费解。经过一番研究，原来是因为他每一行的缩进都全是由空格组成的，大量的空格让代码量随之增大。

现在设小 O 有一份 n 行的代码，第 i 行有 a_i 个空格作为缩进。

为解决这一问题，小 O 要给自己文本编辑器设定一个正整数的默认 TAB 宽度 x ，然后对于每一行，编辑器从头至尾不断把连续 x 个空格替换成一个 TAB，直到剩余空格数不足 x 个。

最终缩进所占代码量为空格数与 TAB 数的和。请你帮小 O 选择一个合适的 x ，使得缩进所占代码量最小。

对于 100% 的数据，满足 $1 \leq n, a_i \leq 10^6$ 。

Solution: UOJ #21. 【UR #1】缩进优化

考虑最大化删除的代码量，即最大化

$$\sum (x-1) \frac{a_i}{x} = \sum a_i - \sum \frac{a_i}{x}$$

问题转化为最小化 $\sum \frac{a_i}{x}$ ，直接枚举 x ，在 $[tx, (t+1)x-1]$, $t \in \mathbb{N}$ 这个范围内的 a_i 可以一起处理，时间复杂度为 $\mathcal{O}(n \log n)$ 。

UOJ #22. 【UR #1】外星人

2044 年，Picks 建成了人类第一台基于量子理论的银河系信息传递机。

Picks 游遍了宇宙，雇用了 n 个外星人来帮他作为信息传递机的中转站。我们将外星人依次编号为 1 到 n ，其中 i 号外星人有 a_i 根手指。

外星人都是很低级的，于是 Picks 花费了很大的精力，才教会他们学会扳手指数数。

Picks 现在准备传递 x 个脉冲信号给 VFleaKing，于是他把信号发给 1 号外星人，然后 1 号外星人把信号发送给 2 号外星人，2 号外星人把信号发送给 3 号外星人，依次类推，最后 n 号外星人把信号发给 VFleaKing。

但是事情没有 Picks 想象的那么顺利，由于外星人手指个数有限，所以如果 i 号外星人收到了 t 个脉冲信号，他会错误的以为发送过来的是 $t \bmod a_i$ 个脉冲信号，导致只发送了 $t \bmod a_i$ 个脉冲信号出去。

Picks 希望他发送出去的脉冲信号数量 x 与 VFleaKing 收到的脉冲信号数量 y 的差的绝对值尽量小。于是他决定通过重新排列这些外星人的顺序来达到这一目的。请你求出与 x 之差最小的 y 。除此之外，请求出有多少种排列外星人的方式能达到最优解，你只需要输出方案数对 998244353 ($7 \times 17 \times 2^{23} + 1$ ，一个质数) 取模后的结果。

对于 100% 的数据，满足 $n \leq 5000$, $x \leq 5000$, $a_i \leq 5000$ 。

Solution: UOJ #22. 【UR #1】外星人

注意到取模的一个性质：先模了一个较小的数后，再去模更大的数就没用了，于是可以将所有模数降序排序，dp 解决第一问。

具体来说，设 $f_{i,j}$ 为考虑了前 i 个外星人后，当前脉冲信号数量可不可以为 j ，转移是显然的，时间复杂度 $\mathcal{O}(nx)$ 。

第二问显然也应该是一个 dp，考虑抛弃之前排好序的外星人，自己不断地确定外星人的顺序。设 $f_{i,j,k}$ 为考虑完了前 i 个外星人，当前的脉冲信号数量为 j ，手指数大于 j 的外星人还有 k 个没有用过的方案数，转移有两

种：放一个手指数小于等于 j 的外星人和放一个手指数大于 j 的外星人，对于前者，这些外星人一定没有被用过，因此可以随便挑一个转移，对于后者，放哪个都一样，根据 k 一起转移即可。

转移时可以发现 i 没什么用，直接去掉这一维就可以做到 $\mathcal{O}(n^2x)$ 。

仔细想想会发现 k 其实也是没用的，重新设计状态为 f_j 表示当前数字是 j ，只有那些手指数小于等于 j 的外星人没有确定顺序的方案数，转移时枚举一个合法外星人，不妨设其手指数为 z ，那么手指数在 $(j \bmod z, j]$ 的外星人在之后也就没用了，根据状态的设计，我们在这时就需要将它们的贡献算上：这也是容易的，因为这相当于在剩下的外星人中间插入已知数量的外星人的方案数，用一个排列数就可以了，时间复杂度 $\mathcal{O}(nx)$ 。

YDRS001C. 科尔沃的闪烁法术

终于，科尔沃的闪烁法术不再是用来瞬移了！他打算用来消除异界魔留下的符文

科尔沃面前从左至右一共放有 n 个异界魔的符文，编号 $1 \sim n$ 。同时，他也准备了 m 个闪烁法术，编号 $1 \sim m$ 。每个闪烁法术会摧毁一段连续的符文。其中闪烁法术 i 会破坏编号范围 $[l_i, r_i]$ 的符文（包括编号 l_i 和 r_i ）并回复 b_i 的魔力值，任意两个闪烁法术所破坏的符文的编号范围都不同，且魔力值的回复具有累加效应。

显然，每个符文不可能被破坏多次。因此，要想一个闪烁法术能够回复魔力，需要该闪烁法术至少破坏一个之前未曾破坏的符文。而无论破坏多少个符文，回复的魔力值都将是 b_i 。

那么科尔沃想知道，该如何安排闪烁法术的施展顺序，可以使得在施展完 m 次闪烁法术后，回复的魔力值总和最大呢？

对于 100% 的数据，保证 $1 \leq n \leq 300, 1 \leq m \leq \frac{n \times (n-1)}{2}, 1 \leq w_i \leq 10^6$

Solution: YDRS001C. 科尔沃的闪烁法术

考虑最终这 n 个符文是什么状态：被划分为若干个区间，因此想到区间 dp。

设 $f_{l,r}$ 为只考虑 $[l, r]$ 之内的符文和被这个区间完全包含的闪烁法术，能够获得的最大魔力值，答案即为 $f_{1,n}$ ，初始时将每个法术对应的整个区间贡献到 f 上，麻烦的地方在于转移。

按照区间 dp 的讨论，考虑转移 $f_{l,r} = \max\{f_{l,mid} + f_{mid+1,r}\}$ ，这显然是不够的，我们可以构造一种这样的情况：一个大区间包含着两个互不相交的小区间，中间留有空隙，那么我们可以先把这两个小区间破坏掉，再把大区间破坏掉，三个法术的魔力值都可以获得，但我们的转移没有考虑这种情况。

怎么办呢？打个补丁就行了：最后一个有贡献的区间一定覆盖了一个点，枚举这个最终被覆盖的点，让它选那个区间内的最优选择，转移方程形如

$$f_{l,r} = \max\{f_{l,mid-1} + val(mid, l, r) + f_{mid+1,r}\}$$

将这个方程和之前的方程合并起来再取一个 max 就可以得到正确的答案了，但实际上有了这个补丁，之前的转移方程就可以丢掉了：因为这个补丁完全包含了最开始的转移方程。

还有一个问题没有解决： $val(mid, l, r)$ 怎么算？实际上是可以预处理的，再用一个 dp 预处理出这个东西然后转移时 $\mathcal{O}(1)$ 查就行。

时间复杂度 $\mathcal{O}(n^3 + m)$ 。

CF487E Tourists

Cyberland 有 n 座城市, 编号从 1 到 n , 有 m 条双向道路连接这些城市。第 j 条路连接城市 a_j 和 b_j 。每天, 都有成千上万的游客来到 Cyberland 游玩。

在每一个城市, 都有纪念品售卖, 第 i 个城市售价为 w_i 。这个售价有时会变动。

每一个游客的游览路径都有固定起始城市和终止城市, 且不会经过重复的城市。

他们会在路径上的城市中, 售价最低的那个城市购买纪念品。

你能求出每一个游客在所有合法的路径中能购买的最低售价是多少吗?

你要处理 q 个操作:

- **C a w**: 表示 a 城市的纪念品售价变成 w 。
- **A a b**: 表示有一个游客要从 a 城市到 b 城市, 你要回答在所有他的旅行路径中最低售价的最低可能值。

对于 100% 的数据, 保证 $1 \leq n, m, q \leq 10^5$, $1 \leq w_i \leq 10^9$ 。

Solution: CF487E Tourists

对原图建出广义圆方树, 方点的权值是所有它连接的圆点的权值的最小值, 那么所求即为 a, b 两点在圆方树上唯一路径上的最小权值: 这可以用树链剖分 $\mathcal{O}(n \log^2 n)$ 解决。

但还有权值的动态修改, 如果我们对每个方点维护一个可重集, 集合中元素是其连接的所有圆点的权值, 修改一个圆点的权值时, 它所连接的每个方点是可以 $\mathcal{O}(\log n)$ 修改的。

问题在于, 一个圆点可能连接 $\mathcal{O}(n)$ 的方点, 例如一个菊花图, 这样时间就炸了。怎么解决这个问题呢? 对于每个方点, 它有一个圆点父亲和若干个圆点儿子, 我们只让方点维护其儿子权值的最小值, 那么维护是容易的: 假如圆点有方点父亲 (这最多只有一个), 在其可重集中更新, 并且在树链剖分的线段树上维护即可。

对于查询操作, 结果还是路径权值最小值吗? 通常是这样的, 但是有一种特殊情况: 假如两点的 lca 是一个方点, 那么答案应该再对这个方点的父亲 (一个圆点) 的权值取最小值。

总时间复杂度 $\mathcal{O}(n \log^2 n)$ 。

UOJ #31. 【UR #2】猪猪侠再战括号序列

有一个由 n 个左括号“(”和 n 个右括号”)”组成的序列。每次操作时可以选定两个数 l, r , 然后把第 l 到第 r 个括号的顺序翻转 (括号的朝向保持不变)。例如将 “()((()” 翻转第 3 到第 7 个括号后的结果为 “()()((”。

我希望使用不超过 n 次操作, 将这个序列变为一个合法的括号序列, 最终输出操作的次数和每次操作的区间。

众所周知, 合法括号序列的定义如下:

- () 是合法括号序列;
- 如果 A 是合法括号序列, 则 (A) 是合法括号序列;
- 如果 A, B 是合法括号序列, 则 AB 是合法括号序列。

对于 100% 的数据, 保证 $n \leq 10^5$ 。

Solution: UOJ #31. 【UR #2】猪猪侠再战括号序列

一个很自然的想法是跑一遍用栈 check 括号序列是否合法的操作, 每次遇到一个单独的右括号变成栈的唯一一个元素时, 找到在它右边的一个左括号, 并将这个区间翻转。

但是每次翻转是 $\mathcal{O}(n)$ 的, 这样暴力的做法带来了 $\mathcal{O}(n^2)$ 的时间复杂度, 有没有更好的做法呢?

为了让时间复杂度尽可能的优秀, 我们尝试设计一种能够不真正翻转括号序列的算法, 为了能够实现这样的算法, 我们一定要在检查到翻转区间的右端点的时候去翻转这个区间, 因为假如在左端点就翻转, 后续的处理会非常麻烦。

也就是说,我们希望翻转这个已知右端点的区间之后,在这个右端点之前的括号序列已经合法.应该怎么做呢?按照括号序列 check 的讨论,假如栈中剩下一个形如 $)\dots(($ 的序列,其中左括号和右括号个数相同,那么我们翻转这个序列,括号序列就合法了.

实际情况还可能是这样 $)\dots))(($,或这样 $)\dots))(($,但无论如何,可以证明,翻转后的序列是合法的括号序列.

于是记录一下栈中左括号与右括号的数量差与栈底位置即可,时间复杂度 $\mathcal{O}(n)$.

Bonus: 最开始的思路真的不行吗?如果不是在右边随便找一个左括号,而是找一个特殊的左括号比如第一个呢?

CF1842F. Tenzing and Tree

有一个点数为 n 的无根树,你要选出 k 个黑点,选定后,一条边的贡献为删掉它后两个连通块所含黑点个数的差(取绝对值,即若个数分别为 a 和 b ,那么这条边的贡献为 $|a-b|$),整棵树的权值定义为所有边的贡献和.

对于每个 $0 \leq k \leq n$,你要求出选出 k 个黑点后这棵树的权值的最大值。
对于 100% 的数据,保证 $1 \leq n \leq 5000$

Solution: CF1842F. Tenzing and Tree

首先,一个重要的性质是:最优解的所有黑点一定是原图的一个连通块,调整法容易证明。

猜测接下来是一个 dp,但是绝对值太难看了,拆掉绝对值,把 $|a-b|$ 写成 $a+b-2\min\{a,b\}$,那么对于选 k 个黑点,让树的权值最大等价于让 $\sum \min\{w_{u_i}, w_{v_i}\}$ 最小,其中 w_{u_i} 和 w_{v_i} 对应了之前的 a 和 b ,枚举的 i 是每条边。

这样还是很不好算,有没有办法把这个 min 也干掉呢?考虑树中一个特殊结点:树的重心。假如我们将所有黑点构成的树的重心作为根,那么树的权值就是

$$k(n-1) - 2 \sum \min\{w_{u_i}, w_{v_i}\} = k(n-1) - 2 \sum w_{u_i}$$

其中 u_i 是这条边中深度较大的点。

这样还是不好统计,将子树贡献拆到每个点上,规定根的深度为 0,那么答案即为

$$k(n-1) - 2 \sum_{u=1}^n dep_u [\text{u is black}]$$

这就好多了,考虑枚举所有黑点构成的树的重心,每次从重心开始 bfs,不断加入点,更新对于当前黑点数量的答案即可。

还有一个问题:在 bfs 加点的过程中,我们没有保证根是所有黑点的重心,这其实是不影响最终答案的,因为最优答案一定会被枚举到而不比这个劣。

时间复杂度 $\mathcal{O}(n^2)$ 。

CF1842G. Tenzing and Random Operations

现有一个长度为 n 的数列 a ,下标从 1 开始,给出整数 n ,执行 m 次下述操作:

- 随机选择一个整数 $i \in [1, n]$
- 对于所有 $j \in [i, n]$, 让 a_j 变为 $a_j + v$

求最终 $\prod_{i=1}^n a_i$ 的期望,模 $10^9 + 7$ 。

$1 \leq n \leq 5000, 1 \leq m, v, a_i \leq 10^9$

Solution: CF1842G. Tenzing and Random Operations

不妨把期望内部的东西展开来,即

$$(a_1 + v + v + \cdots)(a_2 + v + v + \cdots) \cdots (a_n + v + v + \cdots)$$

更标准一点, 设 $x_{j,i} = v \times [\text{第 } j \text{ 次操作让 } x_i \text{ 加了 } v]$, 那么期望内部就是

$$\prod_{i=1}^n \left(a_i + \sum_{j=1}^m x_{j,i} \right)$$

思路是求出所有方案的答案和, 最终除以 n^m 就是期望了.

考虑暴力地把这个式子展开, 从每个括号中选出一项乘起来求贡献. 不同括号中的 v 可能是由同一个操作产生的, 若某次操作让 a_i 加了 v , 那么对于 $j > i$, 必然会让 a_j 加 v , 这会影响我们的转移, 于是考虑加一维, 设 $dp_{i,j}$ 为考虑了前 i 个括号, 已经钦定了 j 个操作让某个 x 不为 0 的答案, 那么有转移

$$dp_{i,j} = dp_{i-1,j} \times (a_i + jv) + dp_{i-1,j-1} \times (m - j + 1) \times i \times v$$

加号前面是指从这个括号里选出 a_i 或者选一个已经被钦定的操作, 加号后面是指选一个之前未被钦定的操作, 它需要从剩下 $m - j + 1$ 个操作中选, 再选出一个 $[1, i]$ 中的位置, 所以有系数 $(m - j + 1)i$.

对于没有被钦定过的操作, 随便选出一个位置即可, 最终答案即为

$$\frac{\sum_{i=0}^{\min\{n,m\}} dp_{n,i} \times n^{m-i}}{n^m}$$

时间复杂度 $\mathcal{O}(n \times \min\{n, m\})$.

CF1842H. Tenzing and Random Real Numbers

现有 n 个在 $[0, 1]$ 范围内的随机实数 x_1, x_2, \dots, x_n 和 m 个限制, 每个限制形如 $x_i + x_j \leq 1$ 或 $x_i + x_j \geq 1$, 询问满足所有限制的概率, 模 998244353。
 $1 \leq n \leq 20, 1 \leq m \leq n^2 + n$

Solution: CF1842H. Tenzing and Random Real Numbers

看起来是一个 dp, 但是限制不容易刻画, 考虑以 $\frac{1}{2}$ 为边界, 由于某个数取到 0.5 的概率为 0, 所以不妨设没有 $x_i = 0.5$, 将所有 x_i 按照在数轴上到 0.5 的距离升序排列.

对于一个 x_i , 若它小于 0.5, 称其为黑点, 否则为白点, 那么两个白点之间肯定满足小于等于 1 的限制而不满足大于等于 1 的限制, 两个黑点类似, 所以我们只关心黑点和白点之间的限制. 设 $g_0(i)$ 为与第 i 个数有小于等于 1 限制的点的集合, $g_1(i)$ 为大于等于 1 的限制点的集合, 按照我们之前的排序 dp, 设 dp_S 为只考虑所有 $x_s, s \in S$ 的值 (它们到 0.5 的距离小于其他数到 0.5 的距离) 满足限制的概率.

考虑 $x_i + x_j \leq 1$ 等价于 $x_i \leq 1 - x_j$, $x_i + x_j \geq 1$ 等价于 $x_i \geq 1 - x_j$, 有转移方程

$$dp_{S \cup i} \leftarrow dp_S \times \frac{1}{|S \cup i|} \frac{1}{2}, \quad i \notin S \text{ and } (S \cup i) \cap g_0(i) = \emptyset$$

$$dp_{S \cup i} \leftarrow dp_S \times \frac{1}{|S \cup i|} \frac{1}{2}, \quad i \notin S \text{ and } (S \cup i) \cap g_1(i) = \emptyset$$

解释一下, 首先, 我们限制了 x_i 是第 $|S \cup i|$ 名, 这个概率是 $\frac{1}{|S \cup i|}$, 其次, 放在这个位置合法当且仅当:

1. 将它放在 0.5 的左侧, 同时, 由于我们的限制, 这个方案合法当且仅当所有和它有大于等于限制的点都没有在 $S \cup i$ 中
2. 将它放在 0.5 的右侧, 同时, 由于我们的限制, 这个方案合法当且仅当所有和它有小于等于限制的点都没有在 $S \cup i$ 中

又由于我们限定了它在 0.5 的左侧还是右侧, 所以概率要再乘 $\frac{1}{2}$.

预处理 [1, 20] 的逆元, 跑一个状压 dp 即可, 时间复杂度 $\mathcal{O}(n2^n)$.

CF1842I. Tenzing and Necklace

给一个长度为 n 的环, i 号点与 $(i \bmod n) + 1$ 之间连边, 边权为 a_i 。

给定正整数 $K \leq n$, 删除一条边的代价为其权值, 求删去若干条边后使得图中每个联通块的大小均不超过 K 的最小代价。

$1 \leq n \leq 5 \times 10^5$

Solution: CF1842I. Tenzing and Necklace

考虑钦定删除 m 条边的最小代价, 最终答案就是对所有 m 的代价取最小值, 现给出如下三条引理:

设两种删边的方案为删除第 a_1, a_2, \dots, a_m 条边和第 b_1, b_2, \dots, b_m 条边, 满足 $a_i < a_{i+1}$, $b_i < b_{i+1}$, $a_1 < b_1$, 同时, 满足钦定删除的编号最小的边为 a_1 后, 删除 a_2, \dots, a_m 为一种最优方案, 对于 b 同理。

引理一: 可以在保证代价不变的同时调整 b_2, \dots, b_m 使得对于任意 $i \leq m$ 有 $a_i \leq b_i$ 。

引理二: 在满足 $a_i \leq b_i$ 的条件后, 若 $b_1 > a_2$, 那么存在一种调整 b_1, \dots, b_m 的方式使得总代价不会更劣且使得 $b_1 \leq a_2$ 。

引理三: 在满足 $a_i \leq b_i$ 且 $a_1 < b_1 \leq a_2$ 的条件下, 存在一种调整 b_2, \dots, b_m 的方式使总代价不变且使得 $a_i \leq b_i \leq a_{i+1}$ 。

我们一一给出证明:

对于引理一, 该调整方案为: 找到第一个使得 $a_i > b_i$ 的位置, 设为 l , 再找到大于 l 的第一个使得 $a_i \leq b_i$ 的位置, 设为 r , 若不存在则为 $m+1$, 发现可以将 $a_l, a_{l+1}, \dots, a_{r-1}$ 和 $b_l, b_{l+1}, \dots, b_{r-1}$ 互相替换且保证替换后方案合法, 又由于两个方案均为钦定 $a_1(b_1)$ 后的最优方案, 因此有这两段的权值和也相同, 所以不断执行这个操作即可。

对于引理二, 不妨设 $a_{m+1} = +\infty$, 找到最小的 j 使得 $b_j \leq a_{j+1}$, 可以将 a_2, a_3, \dots, a_j 与 b_1, b_2, \dots, b_{j-1} 互相替换且保证替换方案而合法, 同引理一, 这个互换不会使代价变多。

对于引理三, 不妨设 $a_{m+1} = +\infty$ 找到最小的满足 $b_i > a_{i+1}$ 的 i , 找到 i 后第一个满足 $b_j \leq a_{j+1}$ 的 j , 可以将 $b_i, b_{i+1}, \dots, b_{j-1}$ 和 $a_{i+1}, a_{i+2}, \dots, a_j$ 互换且保持代价不变。

因此, 假如一定要割掉 m 条边, 我们可以先令 $a_1 = 1$, 找到之后的一组最有解 a_1, a_2, \dots, a_m , 这可以用 dp 在 $\mathcal{O}(n)$ 的时间复杂度内完成 (用一个单调队列), 之后假设对于所有 b_i 有 $a_i \leq b_i \leq a_{i+1}$. 设计一个分治, 让 $solve(\langle l_1, r_1 \rangle, \langle l_2, r_2 \rangle, \dots, \langle l_m, r_m \rangle)$ 表示要求 $b_i \in [l_i, r_i]$ 的最优答案. 假如 $l_1 > r_1$, 我们就可以结束递归了, 否则用 dp 求出 $b_1 = \lfloor \frac{l_1+r_1}{2} \rfloor$ 时的答案, 递归调用 $solve(\langle l_1, b_1 - 1 \rangle, \langle l_2, r_2 \rangle, \dots, \langle l_m, r_m \rangle)$ 和 $solve(\langle b_1 + 1, r_1 \rangle, \langle l_2, r_2 \rangle, \dots, \langle l_m, r_m \rangle)$ 。

怎么求出所有 m 的答案呢? 类似于之前三个引理的调整, 可以证明: 假如割掉第一条边后, 最优方案需要割掉 m 条边, 那么一定存在一种全局最优方案使得割掉的边的数量 m' 满足 $|m' - m| \leq 1$ 。

最后分析一下时间复杂度, 是 $\mathcal{O}(\sum r_i - l_i + 1) = \mathcal{O}(n \log k + mk)$, 由于最优方案一定不存在两个相邻的部分, 长度和 $\leq K$ (因为可以合并使答案更小), 所以不妨设 $m \leq 2\lceil \frac{n}{k} \rceil$, 因此时间复杂度为 $\mathcal{O}(n \log k)$ 。

CF1835B. Lottery

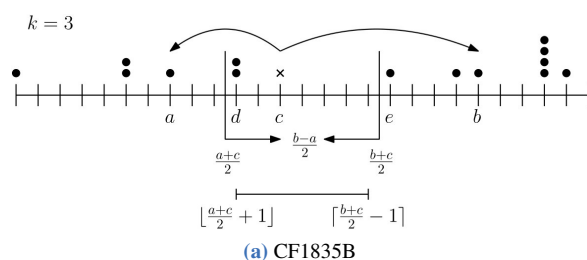
n 个人买彩票, 每个彩票是一个在 $[0, m]$ 之间的整数, 第 i 号人买到的彩票编号为 a_i , 现在 Bytek 也想买彩票, 他可以被看做第 $n+1$ 个人。

假如彩票的中奖编号为 x , 那么买到的彩票编号距离 x 最近的 k 个人将会中奖, 假如距离相同, 人的编号较小的优先中奖。Bytek 想要中奖, 他想知道, 他买哪个彩票的中奖概率最大, 并且求出此时有多少种中奖编号能使他中奖。

$1 \leq n \leq 10^6, 0 \leq m \leq 10^{18}, 1 \leq k \leq 10^6$

Solution: CF1835B. Lottery

贺一下官方题解里的图：



假如我们已经确定了 Bytek 买的彩票为 c ，想要知道哪些中奖编号可以使他中奖，可以找到在他左边离他最近的人 d 和在他右边离他最近的人 e ，在他左边的第 k 个人 a 和在他右边的第 k 个人 b ，假如 Bytek 要中奖，那么中奖位置必须离他比离 a 近且比离 b 近，也就是说我们应该让中奖编号在区间 $(\frac{a+c}{2}, \frac{b+c}{2})$ ，换成整数就是

$$\left[\left\lceil \frac{a+c}{2} \right\rceil, \left\lfloor \frac{b+c}{2} \right\rfloor \right]$$

注意到这个长度只和 c 的奇偶性有关而和具体位置无关，所以对于每组 d, e ，我们只需要随便选两个奇偶性不同的位置对答案取 \max 即可，对于开头和结尾也可以类似分析。

时间复杂度 $\mathcal{O}(n \log n)$ ，瓶颈在于最开始对 a 的排序。

CF1835C. Twin Clusters

给出非负整数 k ，定义 $n = 2^{k+1}$ ，现有 n 个数 g_1, g_2, \dots, g_n ，求出两个不相交区间 $[l, r], [L, R]$ 使得

$$\bigotimes_{i=l}^r g_i = \bigotimes_{i=L}^R g_i$$

其中 \otimes 指二进制下的异或操作，若不存在，输出 -1。

$$0 \leq k \leq 17, 0 \leq g_i < 4^k$$

Solution: CF1835C. Twin Clusters

首先套路地搞一个异或前缀和，不难发现两个区间不相交的限制是假的：假如相交的话可以将相交的部分都去掉，剩下的两个不相交区间仍然符合条件。

到这里之后有两种做法，一种是基于生日悖论的随机化算法，这里不做过多说明，接下来我们讲解一种基于本题特殊数据范围的确定性算法：注意到 $n = 2^{k+1}$ 且 $g_i < 4^k$ ，将每个数分成两部分，二进制下的前 k 位和后 k 位，观察到区间总数有 $\frac{n(n+1)}{2} > \frac{(2^{k+1})^2}{2} = 2^{2k+1} = 2 \times 4^k$ ，而值域是 4^k 级别的，于是一定是有解的。

既然有解，我们就看看是否一定有一些特殊的解，例如区间异或值的前 k 个二进制位都是 0？

根据前缀异或和数组，区间异或值的前 k 个二进制为都是 0 的区间数至少有 $2^k + 1$ 个（可以用抽屉原理），再用一次抽屉原理，在这些结果中必然有两个区间的值相等。

因此我们得到了这个确定性算法：先建出来前缀异或和，从前往后扫一次，每次看看有没有和当前数字前 k 位相同的值，有就尝试更新答案，维护区间，时间复杂度 $\mathcal{O}(2^k)$ 。

CF1835D. Doctor's Brown Hypothesis

给定一张 n 个点 m 条边的简单有向图和常数 k ，请求出有多少组 (x, y) 满足 $1 \leq x \leq y \leq n$ 且存在一条 x 到 y 和一条 y 到 x 的长为 k 的路径， $k \geq n^3$ 。

$$1 \leq n \leq 10^5, 0 \leq m \leq 2 \cdot 10^5, n^3 \leq k \leq 10^{18}$$

Solution: CF1835D. Doctor's Brown Hypothesis

不妨假设边有权值（本题中全为 1）， x, y 满足条件有一个必要条件是他们在同一个强连通分量中，因此不妨先把所有强连通分量求出来，对每一个强连通分量单独求解，下面我们的讨论在同一个强连通分量中进行。

由于 k 很大，我们是走到每个点、每个环上的，我们必然要走到某些环上去不断绕圈，形式化地，假设共有 p 个环，第 i 个环的长度为 l_i ，那么我们在环上走的长度可以写成

$$\sum_{i=1}^p l_i c_i, \quad c_i \geq 0$$

接下来证明一个引理：

引理： 设 $d = \gcd_{i=1}^p l_i$ ，那么充分大的 d 的倍数 y 一定可以被表为形如 $\sum_{i=1}^p l_i c_i, \quad c_i \geq 0$ 的样子。

首先，去掉 $c_i \geq 0$ 的限制后容易用裴蜀定理证明，又由于 y 充分大，所以一定可以被调整到 $c_i \geq 0$ 。

现在的任务转为求出这个 d 。我们断言： $p \mid d$ 的充分必要条件为存在一组 $dist$ ，使得对任意边 (u, v, w) 有 $dist_v \equiv dist_u + w \pmod{p}$ ，下面给出证明：

先证充分性，若 $dist_v \equiv dist_u + w \pmod{p}$ ，那么对于任意的一个环 $(u_1, u_2, w_1), (u_2, u_3, w_2), \dots, (u_r, u_1, w_r)$ ，它的长度为

$$\sum_{i=1}^r w_i \equiv \left(\sum_{i=1}^{r-1} dist_{u_{i+1}} - dist_{u_i} \right) + dist_{u_1} - dist_{u_r} \pmod{p}$$

把这个求和号里面的相同项消掉，结果是

$$(dist_{u_r} - dist_{u_1}) + dist_{u_1} - dist_{u_r} = 0$$

于是我们得到

$$\sum_{i=1}^r w_i \equiv 0 \pmod{p}$$

因此，对于任意的一个环， p 都是其长度的因数，故有 $p \mid d$ 。

再证必要性。由于原图强连通，所以任取一点作为根（设为 $root$ ），那么可以搞出来一棵 dfs 树，定义对于树边 (u, v, w) ，有 $dist_v = dist_u + w$ ，下面证明对于非树边， $dist_v \equiv dist_u + w \pmod{p}$ 。

首先，对于返祖边，这个返祖边和树边构成了一个环，设为 $(u_1, u_2, w_1), \dots, (u_r, u_1, w_r)$ ，其中 (u_r, u_1, w_r) 为返祖边，那么有

$$\sum_{i=1}^r w_i \equiv 0 \pmod{p}$$

将左边展开有

$$\left(\sum_{i=1}^{r-1} dist_{u_{i+1}} - dist_{u_i} \right) + w_r = dist_{u_r} - dist_{u_1} + w_r \equiv 0 \pmod{p}$$

因此有 $dist_{u_r} \equiv dist_{u_1} + w_r \pmod{p}$ ，得证！

对于另外的边（从一棵子树中结点指向另外一棵子树中结点的边），先证明对于一条边 (u, v, w) ，必然有一条从 v 到 u ，长度在模意义下为 $-w$ 的路径：这是容易的，一是由于强连通，必然有路径，其次路径和边构成环，因此必然有长度为 $-w$ 的路径。

那么对于待证的边 (u, v, w) ，他和路径 $root \rightarrow u, v \rightarrow root$ 构成了一个环，而对于路径 $v \rightarrow root$ ，设前一条路径的权值为 w_1 ，后一条路径的权值为 w_2 ，那么有 $w_1 + w + w_2 \equiv 0 \pmod{p}$ 。移项得 $w \equiv -w_1 - w_2 \pmod{p}$ 。

根据之前得到的结果，一定存在一条从 v 到 $root$ 的路径，权值在模意义下为 $-dist_v$ ，取这个路径，由于 $w_1 \equiv dist_u \pmod{p}, w_2 \equiv -dist_v \pmod{p}$ ，替换上式得到

$$w \equiv -dist_u + dist_v \pmod{p}$$

证毕.

有了这个结论, 下面的东西就很显然了: d 就等于所有边 (u, v, w) 得到的 $dist_v - dist_u - w$ 的 gcd, 点对 (u, v) 满足条件当且仅当 $dist_v - dist_u \equiv k \pmod{d}$ 且 $dist_u - dist_v \equiv k \pmod{d}$, 要么 $k \equiv 0 \pmod{p}$, 要么 $k \equiv \frac{d}{2} \pmod{p}$ 且 $2 \mid d$, 此时就只需要满足 $dist_v - dist_u \equiv k \pmod{p}$, 容易计数.

时间复杂度为 $\mathcal{O}(n \log n)$, 比较精细的实现可以做到线性.

【CSP-S 2021】括号序列

小 w 在赛场上遇到了这样一个题: 一个长度为 n 且符合规范的括号序列, 其有些位置已经确定了, 有些位置尚未确定, 求这样的括号序列一共有多少个.

身经百战的小 w 当然一眼就秒了这题, 不仅如此, 他还觉得一场正式比赛出这么简单的模板题也太小儿科了, 于是他把这题进行了加强之后顺手扔给了小 c.

具体而言, 小 w 定义“超级括号序列”是由字符 $(,), *$ 组成的字符串, 并且对于某个给定的常数 k , 给出了“符合规范的超级括号序列”的定义如下:

1. $()$ 、 (S) 均是符合规范的超级括号序列, 其中 S 表示任意一个仅由不超过 k 个字符 $*$ 组成的非空字符串 (以下两条规则中的 S 均为此含义);
2. 如果字符串 A 和 B 均为符合规范的超级括号序列, 那么字符串 AB 、 ASB 均为符合规范的超级括号序列, 其中 AB 表示把字符串 A 和字符串 B 拼接在一起形成的字符串;
3. 如果字符串 A 为符合规范的超级括号序列, 那么字符串 (A) 、 (SA) 、 (AS) 均为符合规范的超级括号序列.
4. 所有符合规范的超级括号序列均可通过上述 3 条规则得到.

例如, 若 $k = 3$, 则字符串 $((**())*(**))$ 是符合规范的超级括号序列, 但字符串 $*()$ 、 $(**())$ 、 $((**))$ 、 $(****)$ 均不是. 特别地, 空字符串也不被视为符合规范的超级括号序列.

现在给出一个长度为 n 的超级括号序列, 其中有一些位置的字符已经确定, 另外一些位置的字符尚未确定 (用 $?$ 表示). 小 w 希望能计算出: 有多少种将所有尚未确定的字符一一确定的方法, 使得得到的字符串是一个符合规范的超级括号序列?

可怜的小 c 并不会做这道题, 于是只好请求你来帮忙.

对于 100% 的数据, 保证 $1 \leq k \leq n \leq 500$.

Solution: 【CSP-S 2021】括号序列

看数据范围很像是区间 dp, 实际上的确是这样的. 套路地设 $f_{l,r}$ 为区间 $[l, r]$ 内方案数, 发现不能合并, 于是加状态, 设

- $dp_{l,r,0}$ 为区间 $[l, r]$ 构成形如 $**...**$ 的方案数 (只可能是 0 或 1), 并且要求 $*$ 的数量不超过 k .
- $dp_{l,r,1}$ 为区间 $[l, r]$ 构成形如 $(..)$ 的合法方案数, 要求最左边的左括号必须和最右边的右括号配对.
- $dp_{l,r,2}$ 为区间 $[l, r]$ 构成形如 $*..*(..)*..()$ 的方案数, 实际上就是一个合法的方案左边接上若干个 $*$, 特别的, 最左边 $*$ 的数量不能为 0 但后面的可以.
- $dp_{l,r,3}$ 为区间 $[l, r]$ 构成形如 $(..)*..*(..)*..*$ 的方案数, 要求和上一种类似, 最右边 $*$ 的数量不能为 0.
- $dp_{l,r,4}$ 为区间 $[l, r]$ 构成形如 $*..*(..)*..*$ 的方案数, 要求最左边和最右边都必须有至少一个 $*$.
- $dp_{l,r,5}$ 为区间 $[l, r]$ 构成形如 $(..)*..*(..)*..*$ 的合法方案数, 要求最左边和最右边不能有 $*$, 特别地, $dp_{l,r,1}$ 包含于这种情况.

可以发现这 6 种状态之间可以互相转移, 答案即为 $dp_{1,n,5}$, 时间复杂度 $\mathcal{O}(n^3)$.

【CTS2019】重复

贾樟柯在《山河故人》里说,「生活就是重复」。在生活中,人们总是喜欢重复自己做过的事情。语言就是一个很经典的例子。

比如,我们在表示疑问时,总不满足于使用一个问号「?」;使用一连串的问号「????」总是显得比较有力。

在表示抱歉时,一句「对不起」总显得不够情愿;连着表示「对不起对不起」才足够表达自己的真诚。

A 国就是一个喜欢重复的国家。在这个国家中,一个基本句子可以用一个长度恰好为 m 的小写字母字符串表示。为了表达自己对重复的喜爱,A 国的人们总喜欢把自己想要表达的句子重复无限多次。

有时,这样的重复是充满意义的。A 国的人们把一个字典序小于给定的字符串 s ,且长度和 s 相同的小写字母字符串称为一个有意义的语义片段。他们想知道,有多少个不同的基本句子(即长度恰好为 m 的小写字母字符串)在经过无限重复后,可以从中找出至少一个有意义的语义片段?

对于 100% 的数据,保证 $1 \leq n, m \leq 2000$ 。

Solution: 【CTS2019】重复

用 T^∞ 表示重复 ∞ 次的串 T , 所求即为满足长度为 m 的, 所有循环同构都 $< s$ 的串 T 的数量. 正难则反, 考虑求出所有循环同构都 $\geq s$ 的串 T 的数量, 用 26^m 减去结果就是最终的答案。

为了求出这个东西, 我们建出串 s 的 KMP 自动机, 若 T^∞ 最终落在 KMP 自动机上的结点 u , 那么往后再加一个 T , 匹配后仍然会位于结点 u , 这是因为 $T^\infty + T = T^\infty$ 。

如果还要满足题目中条件, 即任意循环同构均 $\geq s$, 怎么办呢? 将这个限制等价一下: 设在结点 u , 可以跳到除根之外的失配指针构成的字符集为 Σ_u , 那么假如我们走了 δ_1 这个字母, 并且它不是 Σ_u 中最大的那一个, 那么 s 一定会小于某一个循环同构(考虑用更大的失配指针跳到的那个字符串), 于是限制等价于每次只能跳到根或者跳最大的那个字符。

这时候就可以 dp 了, 先预处理一下: 设 $f_{i,j}$ 为从根开始跳 i 步到达点 j 的方案数, 这部分是 $\mathcal{O}(m|s||\Sigma|)$ 的. 然后枚举 T^∞ 最终落在那个结点, 我们的任务是从这个结点出发走 m 步回到这里. 这里有两种选择: 走到根或不走到根, 前者的贡献是可以直接算出来的, 后者的走法是唯一的, 走下去即可, 时间复杂度是 $\mathcal{O}(m|s|)$ 。

于是就做完了, 时间复杂度为 $\mathcal{O}(m|s||\Sigma|)$

CF1847C. Vampiric Powers, anyone?

给定长度为 n 的初始数列 A , 你可以进行任意次操作, 每次操作形如:

- 假设当前数列长度为 m , 其中元素分别为 A_1, A_2, \dots, A_m
- 任选一个整数 $i \in [1, m]$
- 向数列中添加一个元素 $A_{m+1} = A_i \text{ xor } A_{i+1} \text{ xor } \dots \text{ xor } A_m$

其中 xor 为按位异或操作。

询问可以得到的 A 数列中最大的可能元素。

对于 100% 的数据, 满足 $1 \leq n \leq 10^5, 0 \leq A_i < 2^8$

Solution: CF1847C. Vampiric Powers, anyone?

可以证明新添加进去的元素一定是初始的 A_i 中一段连续区间的异或和, 至此可以用 01Trie, 但本题 2^8 的小值域范围让我们可以枚举两个存在的前缀异或和对答案取 max, 时间复杂度为 $\mathcal{O}(n + w^2)$, 其中 w 为值域。

AGC032C

给定一张无向联通图，点数为 n ，边数为 m ，问是否可以将所有边不重不漏地划分为三部分，使得每部分是一个环，一个点可以在同一个环中出现多次。

$$1 \leq n \leq m \leq 10^5$$

Solution: AGC032C

首先如果可以划分，这个图必然是一个欧拉图，因此先判断所有点的度数是否都是偶数。

假如一个欧拉图存在某个点，其度数不小于 6，那么显然可以将边集划分为经过该点的 ≥ 3 个环（该点的每个度可以组成一个环），不断将环合并，一定可以合并为三个环。

假如所有点的度数都为 2，该图为一个环，一定不符合条件，剩下没有讨论的情况就是只有度数为 4 和 2 的点。

若图中只有一个度数为 4 的点，那么只能是形如经过这个点的两个环在此处连起来的情况，不符合条件。

若图中有两个度数为 4 的点，设为 u_1, u_2 有两种情况：这四个度都相连，即图可以用四条路径 $(u_1, v_{11}, v_{12}, \dots, u_2)$ 和 $(u_1, v_{21}, v_{22}, \dots, u_2)$ 、 $(u_1, v_{31}, v_{32}, \dots, u_2)$ 、 $(u_1, v_{41}, v_{42}, \dots, u_2)$ 来描述，此时不符合题意。否则该图形如： u_1 的两个度构成一个环， u_2 的两个度构成一个环，剩下的四个度共同构成一个环，一定符合条件。

若有多于两个度数为 4 的点，可以证明一定符合条件，分类讨论结束。

时间复杂度 $\mathcal{O}(n)$ 。

CF1325F. Ehab's Last Theorem

给出一幅 n 个点， m 条边的图。让你找出至少有 $\lceil \sqrt{n} \rceil$ 个点的环，或正好有 $\lceil \sqrt{n} \rceil$ 个点的独立集。（两者请自己选一个做）

如找的是最大独立集，输出 1 并输出最大独立集的顶点。

如找的是环，输出 2，输出环的顶点个数并输出环的所有顶点。

$$5 \leq n \leq 10^5, n-1 \leq m \leq 2 \times 10^5$$

Solution: CF1325F. Ehab's Last Theorem

在图上 dfs 找到 dfs 树，边被分为返祖边和树边，若存在一条返祖边使得其与树边构成的环满足环条件，直接输出，下面假设不存在这样的边。

设 $D = \lceil \sqrt{n} \rceil$ 将所有的点按照深度模 $D-1$ 分为 $D-1$ 类，根据抽屉原理，一定存在至少一个类中含有不少于 $\lceil \frac{n}{D-1} \rceil \geq D$ 个点，而根据我们之前的假设（不存在满足条件的返祖边），同一类中的点之间一定两两无边。

时间复杂度 $\mathcal{O}(n+m)$ 。

寻宝

小 Y 是一位寻宝游戏狂热者，经过他的不懈努力，他获得了一份珍贵的藏宝图。

藏宝图中给出了 n 个宝箱的地点和具体信息，每个宝箱内都有大量的宝物。宝箱有不同的颜色，可以用 1 至 m 之间的整数来表示 m 种不同颜色，其中第 i 个宝箱的颜色为 c_i 。

然而，打开宝箱需要使用钥匙。钥匙也是有颜色的，一把钥匙只能打开和它颜色相同的宝箱，并且一把钥匙只能使用一次（即用钥匙打开一个宝箱之后，该钥匙即损坏）。

只有打开宝箱，才能得到宝箱内的宝物。

幸运的是，藏宝图中显示了某些宝箱内有钥匙 k_i （每个宝箱至多 1 把），也可能没有，用 0 表示，在打开装有钥匙的宝箱之后，可以用宝箱内的钥匙打开其它宝箱。

现在小 Y 手上没有任何钥匙。小 Y 可以购买任意数量、任意颜色的钥匙，以打开所有的宝箱。钥匙十分昂贵，不过为了得到更昂贵的宝物，小 Y 想知道，至少需要购买多少把钥匙才能打开所有的宝箱？

对于 100% 的数据，满足 $1 \leq m \leq n \leq 5 \times 10^5$, $1 \leq c_i \leq m$, $0 \leq k_i \leq m$

Solution: 寻宝

将钥匙看作点，宝箱看成钥匙与钥匙之间的有向边，即宝箱 i 等价于边 $\langle c_i, k_i \rangle$ ，要求等价于设定最少的起点个数，从每个起点引一条路径，经过所有边。

由于每个边都要被经过，所以答案即为所有点的出度数减去入度数对 0 取 max 后的和，但是发现不太对：比如图是一个环。

打个补丁就好了，统计对答案贡献为 0 的极大连通块个数，若这个连通块不是一个孤立点，将答案加一，用并查集维护的时间复杂度为 $\mathcal{O}(n + m\alpha(m))$ 。

博弈

都什么年代了，还在下传统围棋？

小 Y 闲来无事，约小 C 玩新游戏。黑板上有 n 个正整数 a_0, a_1, \dots, a_{n-1} 排成一个环。每局游戏将进行多轮，每轮中，小 C 须先任选一个还未划掉的数并划掉它（设为 a_i ）；之后小 Y 从 $a_{(i-1+n) \bmod n}$ 和 $a_{(i+1) \bmod n}$ 中选择一个还未划掉的数并划掉它，或者选择跳过这一轮，然而如果这两个数都已被划掉，小 Y 就只能跳过这一轮。小 C 没得选时，这一局游戏结束。在每局游戏里，小 Y 和小 C 都想最大化自己划掉的数的总和，且双方都采取了最优策略。

由于每次玩同样的游戏太无聊了，他们请大 Y 来改变改变游戏局面。在每局游戏之前，大 Y 将选定 x, y ，把 a_x 修改为 y 。在每局游戏结束之后，被划掉的数将恢复为未划掉的状态，但大 Y 修改的数不会变回去，共修改 q 次。

求每局游戏小 Y 划掉的数的总和。

对于 100% 的数据，满足 $1 \leq n, q \leq 10^5$, $1 \leq a_i, y \leq 10^9$

Solution: 博弈

将 a_i 和 $a_{(i+1) \bmod n}$ 之间连边，边权为 $\min\{a_i, a_{(i+1) \bmod n}\}$ ，答案即为该图的最大权匹配，证明如下：

首先证明后手存在策略使得答案不小于最大权匹配：每次选先手所选点的匹配点即可。

再证先手存在策略使得答案不大于最大权匹配：每次选点的时候，选两个相匹配的点中权值较大的那个即可。证毕。

用线段树维护边权即可实现 $\mathcal{O}(\log n)$ 的维护，方法是相邻两个边不能同时选，于是线段树维护的信息要加上线段左端选或没选、右端选或没选的限制，之后是平凡的。

时间复杂度 $\mathcal{O}(n + q \log n)$ 。

邮递

小 Y 觉得写代码太苦了, 决定成为一名快递员, 在一条公路上取件送件。我们把这条公路抽象为一根数轴, 记实数 x 在数轴上对应的点为 (x) 。

小 Y 需要送 m 个快递。第 i 个快递从 (s_i) 处运到 (t_i) 处。为了送这个快递, 小 Y 必须先到达 (s_i) 处取快递, 后到 (t_i) 处送快递。如果小 Y 成功送了这个快递, 会获得 v_i 元工资。特别地, 当 $s_i = t_i$ 时, 小 Y 只需要到达 (s_i) 处即可获得工资。

最开始, 小 Y 在原点, 而车里的电量只够他行驶 n 个时刻了。由于小 Y 喝醉了酒, 他乱开车: 在这 n 个时刻里的每一个时刻, 他都会等概率随机选择一个方向并向这个方向行驶 1 个单位长度。

假设小 Y 同时可以携带足够多的快递, 并且在行至一个点 (x) 时总会把所有 $s_i = x$ 的快递取上车, 然后把车上所有 $t_i = x$ 的快递送出, 小 Y 掉头的时间以及取快递上车、送出快递的时间忽略不计。

求小 Y 期望获得多少元工资, 答案对 998244353 取模。

对于 100% 的数据, 满足 $1 \leq n \leq 2 \times 10^5$, $1 \leq m \leq 5 \times 10^5$, $-n \leq s_i, t_i \leq n$, $1 \leq v_i < 998244353$

Solution: 邮递

首先根据期望的线性性, 每个快递是独立的, 答案即为 $\sum_{i=1}^m Pr(w_i)v_i$, 其中 $Pr(w_i)$ 为送达快递 i 的概率。我们用合法方案数除以总方案数来求概率, 将路径放到二维笛卡尔坐标系上, 限制等价于从 $(0,0)$ 出发走 n 步, 每次向右上角或右下角走一步 (即横坐标加一, 纵坐标加一或减一), 先经过了 $y = s$ 后经过 $y = t$ 。

如果 $|s_i| \leq |t_i|$, 那么先后的限制是没用的, 这样会方便很多, 尝试将所有情况都转化到这种情况: 第 i 个快递永远等价于起点为 0, 终点为 $|s_i| + |t_i - s_i|$ 的快递, 这是很直观的。

于是设 $|s_i| + |t_i - s_i| = d$, 限制等价于从 $(0,0)$ 出发走 n 步, 经过了 $y = d$ 的方案数。

若终点的纵坐标大于 d , 一定满足条件, 这类的方案数为 $\sum_{i=d+1}^n [2 \mid n+i] \binom{n+i}{\frac{n+i}{2}}$, i 就是枚举的终点纵坐标。

若终点的纵坐标小于 d 且在终点之前的某处满足了条件, 那么找到 $y = d$ 与这个折线的最后一个交点, 将折线在这之后的线都沿着 $y = x$ 翻折就变成了之前讨论的那一类 (终点的纵坐标大于 d), 并且构成双射, 于是这类的方案数也为 $\sum_{i=d+1}^n [2 \mid n+i] \binom{n+i}{\frac{n+i}{2}}$ 。

最后若终点的纵坐标等于 d 是容易的, 方案数即为 $[2 \mid n+d] \binom{n+d}{\frac{n+d}{2}}$ 。

三部分加起来除以 2^n 就是所求概率, 注意到前两类就是一个组合数的下指标区间求和, 前缀和或者后缀和预处理一下就可以做到单次查询 $\mathcal{O}(1)$ 。

总时间复杂度 $\mathcal{O}(n+m)$ 。

loj#6089. 小 Y 的背包计数问题

小 Y 有一个大小为 n 的背包, 并且小 Y 有 n 种物品。

对于第 i 种物品, 共有 i 个可以使用, 并且对于每一个 i 物品, 体积均为 i 。

求小 Y 把该背包装满的方案数为多少, 答案对于 23333333 取模。

定义两种不同的方案为: 当且仅当至少存在一种物品的使用数量不同。

对于 100% 的数据, 保证 $1 \leq n \leq 10^5$

Solution: loj#6089. 小 Y 的背包计数问题

优先考虑根号分治, 设 $D = \lfloor \sqrt{n} \rfloor$ 。

对于 $i > D$, 由于 $\lfloor \frac{n}{i} \rfloor \leq i$, 因此可以当成完全背包来做, 但是对于 $i \leq D$, 个数的限制就有用了, 是一个多重背包。

对于 $i \leq D$ 的部分做多重背包可以做到 $\mathcal{O}(n\sqrt{n})$, 设 $f_{i,j}$ 为考虑了前 i 类物品, 体积为 j 的方案数, 转移方程为

$$f_{i,j} = \sum_{k=0}^i f_{i-1,j-ki}$$

稍微变形一下得到

$$\begin{aligned} f[i][j] &= \sum_{k=0}^i f[i-1][j-ki] = \sum_{k=-1}^{i-1} f[i-1][j-i-ki] \\ &= \sum_{k=0}^i f[i-1][j-i-ki] + f[i-1][j] - f[i-1][j-i-i^2] \\ &= f[i][j-i] + f[i-1][j] - f[i-1][j-i-i^2] \end{aligned}$$

复杂度就变成 $\mathcal{O}(n\sqrt{n})$ 了.

对于后面的完全背包, 直接做是 $\mathcal{O}(n^2)$ 的, 太烂, 考虑怎样优化.

注意选的物品个数很少, 一定不超过 D , 于是设 $g_{i,j}$ 为放了 i 个物品, 体积为 j 的方案数, 直接做是困难的, 考虑它的另一个意义.

我们不如强制让每次加入的数字只能是 $D+1$, 但是可以在加入 i 个数字后让这 i 个数字整体加 1, 不难发现每个方案有且仅有唯一的一种操作序列.

在这个意义下, 转移是容易的:

$$g_{i,j} = g_{i-1,j-D-1} + g_{i,j-i}$$

同样是 $\mathcal{O}(n\sqrt{n})$ 的, 最终答案即为

$$\sum_{i=0}^n f_{D,i} \times \left(\sum_{j=0}^D g_{j,n-i} \right)$$

总时间复杂度 $\mathcal{O}(n\sqrt{n})$.

loj#3278. 「JOISC 2020 Day3」收获

现在 IOI 庄园有 N 名员工, 在周长为 L 的湖的湖岸边有 M 棵苹果树。

第 i 名员工从湖的最北点顺时针走了 A_i 米, 第 i 棵苹果树长在从湖的最北点顺时针的 B_i 米。

因为特殊原因, 每棵苹果树上最多长一个苹果, 初始时刻每棵苹果树上都有 1 个苹果, 如果一棵树上的苹果被摘掉了, 在恰好 C s 后会长出一个苹果。

每名员工在初始时刻都在自己原本的位置, 每过一个时刻就会顺时针走 1 米, 遇到有成熟苹果的苹果树就会把苹果摘下来。

现在 JOI 君给定了 Q 个询问, 第 i 个询问为:

- 询问第 V_i 个员工在时刻 T_i 结束后收获到几个苹果。

对于 100% 的数据, $1 \leq N, M, Q \leq 2 \times 10^5$, $N + M \leq L$, $1 \leq C, L \leq 10^9$, $0 \leq A_i, B_i < L$, $A_i < A_{i+1}$, $B_i < B_{i+1}$, $A_i \neq B_i$, $1 \leq V_i \leq N$, $1 \leq T_i \leq 10^{18}$ 。

Solution: loj#3278. 「JOISC 2020 Day3」收获

一道非常恶心的 DS 题.

首先, 注意到询问是对人的, 但是题目中是人在顺时针动. 我们人为地将其变一下: 人不动, 苹果树逆时针动, 考虑固定了人的结构后怎样统计答案.

将图论的模型建出来：每个人向他逆时针方向第一个距离不小于 C 的人连边，权值为两者距离（注意可能是绕了很多圈之后距离才不小于 C ，这时的边权仍然应该加上绕圈的距离）表示若这个人吃了一个苹果后，这个苹果树的下一个苹果会被边终点的那个人吃掉，时间间隔为边权。

对于每个苹果树，向它逆时针方向第一个人之间连边，边权为两者距离，表示它的第一个苹果会被这个人吃掉。

由于每个点的出度均为 1，整个图就是一个基环树森林，具体的，是内向树森林，这时对每个基环树是独立的，分别考虑。

将询问离线下来，对于一个询问，实际上要求的就是：从每个苹果树开始不断沿着出边走，在一定距离内询问的人代表的点会被经过多少次，考虑在基环树上做这个问题应当转换成树上问题：因为在树上，这个问题是很容易的。

我们应该断掉环上的任意一条边，那么此时对一个询问的贡献就被分为了两类：

- 经过了这条边的贡献
- 没有经过这条边的贡献

对于第一类，这是一个求子树内距离不超过一定值的贡献的问题，可以转换为 dfs 序-点深度的二维数点问题， $\mathcal{O}(n \log n)$ 解决。

对于第二类，我们先把所有苹果树都走到该边的终点上，设所需时间为 t_i 。

对于一个询问 u, T ，第二类对其有贡献仅当 u 在环上，此时第二类贡献就是

$$\sum_{t_i \leq T} \lfloor \frac{T - t_i + \text{len} - \text{dist}(y, u)}{\text{len}} \rfloor$$

其中 len 为基环树的环长， y 为断边的终点， $\text{dist}(u, v)$ 表示环上从 u 走到 v 的距离。

这个其实也是个二维数点：先对询问按照 T ，对贡献按照 t_i 升序排，扫的时候记录 num 为之前的苹果树数目， sum 为当前 $\sum \lfloor \frac{t_i}{\text{len}} \rfloor$ ，那么到一个询问时，贡献就是 $\text{num} \times \lfloor \frac{T + \text{len} - \text{dist}(y, u)}{\text{len}} \rfloor - \text{sum}$ 减去一个值（因为下取整内部的东西是不能直接拆成两个下取整的和的）。

减去什么呢？减去的的就是所有苹果树中 $t_i \bmod \text{len} > T \bmod \text{len}$ 的树的数量，于是又变成了一个第一维为 $\bmod \text{len}$ ，第二维为 T or t_i 的二维数点问题，离散化后用数据结构维护即可。

总时间复杂度 $\mathcal{O}(n \log n)$ 。

【ABC214G】Three Permutations

给定两个 $1 \sim n$ 的排列 p, q ，询问 $1 \sim n$ 的排列 r 的数量， r 应满足 $\forall i \in [1, n] \cap \mathbb{Z}, r_i \neq p_i \wedge r_i \neq q_i$ ，模 $10^9 + 7$ 。

对于 100% 的数据，满足 $1 \leq n \leq 3000$

Solution: 【ABC214G】Three Permutations

直接计算是不容易的，考虑容斥：设 f_i 为钦定 i 个位置不满足要求的方案数，那么答案即为

$$\sum_{i=0}^n (-1)^i (n-i)! f_i$$

问题化为求 f ，先看看假如我们已经确定了这 i 个位置（设为 j_1, j_2, \dots, j_i ）后的方案数：将这些位置上的 p 和 q 之间连边，即连边 $p_{j_1} - q_{j_1}, p_{j_2} - q_{j_2}, \dots$ ，得到一张由若干个环和若干个链的图，边数为 i ，要让这些位置不符合条件，转化到图上就是让每个边选择它的一个端点，且每个端点至多被一条边选择的方案数，称对于每条边，它选择的这个端点为其关键点。

对于环，方案数是 2，对于链，方案数是 siz ，其中 siz 为链上的点数，这是因为 $\text{siz} - 1$ 条边会选择 $\text{siz} - 1$ 个关键点，枚举那个没有被选择的关键点，剩下的方案就是固定的了，那么计数也是不困难的：全乘起来就行。

直接这样做还是指数级的，考虑优化：在一开始就将图全部建出来，即对每个 i 都让 p_i 和 q_i 连边，得到了一个一堆环的图，考虑在这个图上计数。

对于 f_i 实际上就是在这张图上选 i 条边的方案的贡献的和，dp 去算，设 $g_{i,j}$ 为考虑了前 i 个环，选出 j 条边的方案的贡献的和。

若第 i 个环是一个自环，转移是显然的： $g_{i,j} = g_{i-1,j} + g_{i-1,j-1}$ 。

若第 i 个环不是自环，转移就变得复杂起来，思路还是考虑前 $i-1$ 个环选了 j 条边，第 i 个环选了 k 条边，乘起来贡献到 $g_{i,j+k}$ 上。

前 $i-1$ 个环选出 j 条边就是 $g_{i-1,j}$ ，第 i 个环选 k 条边呢？设第 i 个环的大小为 siz ，我们将一条边 (u,v) 拆成 (u,w) 和 (v,w) ，假如我们选择了这条边，这条边选择的关键点为 u ，就可以等价为了选择了 (u,w) 这条边，于是问题化为一个大小为 $2siz$ 的环，选择 k 个两两不相邻的边的方案数。

这是一个组合问题，不妨将这个环重新编号为 1 到 $2siz$ ，其中编号相邻的点之间有边，考虑边 $(1, 2siz)$ 这条边是否被选：

1. 这条边被我们选了，剩下的问题就是在一个边数为 $2siz-1$ 的链上选择 $k-1$ 条两两不相邻的边，首尾都不能选的方案数，再将首尾去掉，转化为 $2siz-3$ 个小球，选 $k-1$ 个小球，要求两两不相邻的方案数，这是一个经典问题：选出了 $k-1$ 个，没有选 $2siz-k-2$ 个，答案就是将这 $k-1$ 个插入到没有选择的这些小球的 $2siz-k-1$ 个空隙中的方案数，为 $\binom{2siz-k-1}{k-1}$ 。
2. 这条边我们不选，剩下的问题就是在一个边数为 $2siz-1$ 的链上选择 k 条两两不相邻的边，首尾可以选的方案数，和上一类是本质相同的，方案数即为 $\binom{2siz-k}{k}$ 。

于是转移就是

$$g_{i,j+k} \leftarrow g_{i-1,j} \times \left(\binom{2siz_i-k}{k} + \binom{2siz_i-k-1}{k-1} \right)$$

看起来复杂度是 $\mathcal{O}(n^3)$ 的，其实在枚举 k 时只需要枚举到环长，复杂度不超过 $\mathcal{O}(n \sum siz)$ 即 $\mathcal{O}(n^2)$ 。

艾莎

艾莎给你一个长度 n 的序列 a_1, \dots, a_n ，共 m 次操作，共两种操作类型：

1. 给定 l, r, x ，将 a_l, \dots, a_r 加上 x 。

2. 给定 l, r ，查询 $\max_{l \leq L < R \leq r} \frac{\sum_{i=L}^R a_i}{R-L+1}$ 。

输出为分数形式，对于 100% 的数据，满足 $1 \leq n, m \leq 10^6$ ， $|a_i|, |x| \leq 10^3$ ，所有数值为整数。对于操作 2，保证不存在 $l = r$ 的情况。

Solution: 艾莎

这是一个子区间平均数最大值，限制中 L 不能等于 R 很扎眼，考虑当 $L = R$ 时会发生什么。

分析一下平均数的性质：两个数的平均数一定不会大于其中较大的那个，因此若 L 可以等于 R ，那么最终结果一定是区间中的最大值。

但是这时候有了这个限制，结果必须是一段长度不为 1 的区间，此时有引理：答案区间的长度一定可以不大于 3。

引理的证明是简单的：任何一段长度大于等于 4 的区间都可以拆为两个长度不小于 2 的区间，我们取其中平均数较大的那个，就得到了一个不劣的解，一直取下去即可得到一个长度为 2 或 3 的答案区间。

静态的问题就直接上线段树解决了，区间加也是容易的，时间复杂度 $\mathcal{O}((n+q) \log n)$ 。

沙奈朵

沙奈朵给定一个长为 n 的序列 a ，每个位置是一个 $[1, n]$ 内的整数。

定义 $f(i, j)$ 表示有多少 x 满足 $i \leq x < j$ 且 $a_x \neq a_{x+1}$ 。

有 m 次操作：

- 1 l r x: 表示将 l 位置的元素修改为 x 。
- 2 l r x: 表示查询区间 $[l, r]$ 中，对任意 $l \leq i < j \leq r$ ，且 $a_i = a_j = x$ ， $f(i, j)$ 的和。

注意，为了让两种操作读入格式一致，1 操作中的 r 变量没有作用。

对于 100% 的数据，满足 $1 \leq n, m \leq 5 \times 10^5$ ， $1 \leq l \leq r \leq n$ ， $1 \leq a_i, x \leq n$ ，所有输入均为整数。

Solution: 沙奈朵

这个 f 函数其实就是区间颜色段数量减一，所求即为区间左右端点均为某个颜色的子区间颜色段数量和，带修改。考虑没有修改且只有一次询问的时候怎么做：设询问颜色为 x ，区间为 $[l, r]$ ，那么从左到右扫一遍，记录在该位置之前的颜色 x 的数目和在以该位置之前的 x 为左端点，该位置为右端点的区间的颜色段个数和，每次遇到 x 后加进答案即可，时间复杂度 $\mathcal{O}(n)$ 。

将所有操作离线下来并将序列分块，从左到右对每一块都扫一遍询问，贡献到该询问的答案上，由于每一块的颜色数是 $\mathcal{O}(\sqrt{n})$ 的，所以我们可以对每个颜色都在这个块上跑一次之前的操作，单个块的复杂度是 $\mathcal{O}(\sqrt{n})$ 的，共 $\mathcal{O}(\sqrt{n})$ 个块，这部分的时间复杂度为 $\mathcal{O}(n)$ 。

下面细说怎样做到单个块 $\mathcal{O}(\sqrt{n})$ 处理出每个颜色的答案：首先对于不在块中的颜色，它们的答案都是一样的，于是我们可以只关注块中颜色的答案，共有 $\mathcal{O}(\sqrt{n})$ 个。

从左到右扫块中元素，每遇到一个颜色，不妨设为 x ，更新这个颜色的答案，具体怎样更新呢？我们不关心这个 x 和上一个 x 之间的具体情况，只关心这其中的颜色段个数，而这是很容易用前缀相减得到的，于是可以怎么做呢？之前的答案 + 这段我们不关心具体情况的区间答案 + 该位置答案，注意这里的加是指合并两个结构体。

最后将每个颜色加上最后的该颜色到区间右端点的那段区间即可。

问题在于怎样快速地合并两段区间得到新区间的答案：只维护之前的三个值已经不够了，我们还要考虑跨过区间交点的区间的贡献：将每个这样的区间拆成两部分：在左侧小区间的部分和在右侧小区间的部分，每个这样跨过中点的区间被拆为了两端小区间。

那么对于左侧小区间的那一堆部分，对大区间的贡献就是右侧 x 个数乘左侧的所有 x 到左侧小区间右端点的贡献和，右侧小区间的部分同理。还有一个小细节是：假如左侧区间的右端点等于右侧区间的左端点，还要减掉这个点的一次贡献，于是我们对每个结构体，维护的信息是：答案、左端点、右端点、该颜色数量、除了右端点所在段的颜色段个数、该颜色到右边界颜色段个数和、该颜色到左边界颜色段个数-1 的和，可以 $\mathcal{O}(1)$ 合并。

现在关注单个块对询问的贡献：从头开始扫询问，假如遇到了一个修改，暴力地在数组上操作后重构，复杂度是 $\mathcal{O}(q\sqrt{n})$ ，假如遇到了一个询问并且和该块有交，假如是整块都包含在其中，直接查表加进去，否则暴力去算，加进去，时间复杂度 $\mathcal{O}(q\sqrt{n})$ 。

总时间复杂度 $\mathcal{O}((n+q)\sqrt{n})$ ，实际上本题的修改可以做到区间修改，复杂度不变。

绒绒鸽

给一个长 n 的序列，有 m 次操作，每次操作会给你一个数 x ，你需要在序列的 x 位置放一只绒绒鸽，之后查询 x 位置的绒绒鸽数。

每次操作结束后，绒绒鸽会变得很活泼，于是会开始蹦蹦跳跳，即对每个 1 到 n 之间的整数 i ， i 位置的所有绒绒鸽都会跳到 a_i 位置。

强制在线，对于 100% 的数据，满足 $1 \leq n, m \leq 5 \times 10^5$ ， $1 \leq a_i \leq n$ 。

Solution: 绒绒鸽

假如从 i 向 a_i 连有向边, 构成的图会是一个内向树森林, 每个内向树之间独立, 接下来我们只讨论一个内向树的情况.

假如只有一个环, 维护是容易的, 用时间模环长得到原本应该在的位置, 在这上面加、查即可. 到了树上也是类似的, 若一个绒绒鸽在时刻 t_1 在点 u , 那么它在时刻 t_2 所在的位置 v 应当满足 $dep_u - dep_v = t_2 - t_1$, 同时 v 是 u 的祖先.

将内向树的环去掉, 对剩下的森林轻重链剖分, 绒绒鸽绝大部分时间都在重链上跳, 于是我们可以对每个重链开一个桶维护 $dep_u + t_1$, 每个绒绒鸽跳 \log 次轻链就会跳到环上, 每个时刻暴力地解决需要跳轻链的绒绒鸽, 时间复杂度 $\mathcal{O}(m \log n)$.

G-CAT

有一个长度为 n 的序列, 第 i 个位置的值为 c_i .

q 次询问, 每次给定 $1 \leq l \leq r \leq n$, 考虑子序列 c_l, c_{l+1}, \dots, c_r , 你需要选出若干个互不相交的区间, 满足每个区间的元素之和为 0. 要求最大化选择区间的数量, 求出你可以选出多少个区间.

对于 100% 的数据, $1 \leq n, q \leq 4 \times 10^5$, $-10^9 \leq c_i \leq 10^9$, $1 \leq l_i \leq r_i \leq n$.

Solution: G-CAT

指定一个左端点, 考虑所有满足条件的右端点, 显然它们之中至多选一个, 显然我们应该选择最靠左的那个右端点, 于是有用的区间本质上只有 $\mathcal{O}(n)$ 个, 这也同样可以用前缀和或后缀和 $\mathcal{O}(n \log n)$ 用一个 `std::map` 得到.

由于没有修改, 序列的结构是固定的, 我们选择的那些区间实际上也是固定的: 也就是说, 我们选完一个区间, 选完某一个区间之后的将要选择的下一个区间本质上是固定的: 应该选哪一个呢? 右端点最靠左的一个, 这也同样是显然的.

于是对于每个左端点为 i 的区间, 设其右端点为 j , 选完这个区间后假如还能选择, 一定会选择一个左端点 $> i$, 右端点最小的区间, 这同样可以 $\mathcal{O}(n)$ 处理出来.

每个区间向它之后将会选择的那个区间连边, 对于每一个询问, 先找到一个在区间 $[l, r]$ 中且右端点最小的结点, 不断跳父亲直到右端点大于 r , 答案即为中途经过的结点数, 倍增优化可以做到单次 $\mathcal{O}(\log n)$, 总时间复杂度 $\mathcal{O}((n + q) \log n)$.

OIL

给你一个长为 n 的序列 a .

定义 $\text{maxpre}(l, r)$ 是区间 $[l, r]$ 的最大前缀和, 即 $\max \sum_{j=l}^i a_j, i \in [l, r]$.

定义 $\text{maxsuf}(l, r)$ 是区间 $[l, r]$ 的最大后缀和, 即 $\max \sum_{j=i}^r a_j, i \in [l, r]$.

最大前缀与最大后缀都可以是空串.

求:

$$\sum_{l=1}^n \sum_{r=l+1}^n \sum_{i=l}^{r-1} \text{maxpre}(l, i) \text{maxsuf}(i+1, r)$$

输出答案对 $10^9 + 7$ 取模的结果.

对于 100% 的数据, 满足 $1 \leq n \leq 10^5$, $-10^9 \leq a_i \leq 10^9$.

Solution: OIL

先枚举中间的那个 i ，问题化为求

$$\sum_{i=1}^{n-1} \left(\sum_{l=1}^i \text{maxpre}(l, i) \right) \times \left(\sum_{r=i+1}^n \text{maxsuf}(i+1, r) \right)$$

考虑左边怎么对每个 i 快速预处理，右侧同理，预处理出之后就可以 $\mathcal{O}(n)$ 求答案了。

将所有从 $j(1 \leq j \leq i)$ 到 i 的区间拆分为两部分：maxpre 和剩下的那个后缀，每次将 i 移动到 $i+1$ 时，所有区间的后缀部分会加上一个值，假如某个区间的后缀部分大于了 0，我们就需要将这个后缀部分加到前缀部分去，每个 i 的答案就是所有这样的区间的前缀部分的和。

暴力维护是 $\mathcal{O}(n^2)$ 的，考虑每次更新前缀部分时，若两个区间都被更新，那么它们的后缀部分在之后将会永远相同，可以一起处理（实现时对每个结点多记录一个数量，每个结点对答案贡献时乘上其数量即可），至于后缀加，是可以打全局 tag 做的。

用堆维护，时间复杂度 $\mathcal{O}(n \log n)$ ，总时间复杂度 $\mathcal{O}(n \log n)$ 。

武器 (weapon)

现在 Y 国从仓库里取出了 n 个机甲，第 i 个机甲版本号为 i 。仓库里还有 m 个武器，第 i 个武器攻击力为 a_i 。

武器的驱动、接口、协议会随着机甲版本的更新而改变，所以，对于一个武器而言，太老或者太新版本的机甲都不能装配。具体的，对于第 i 个武器，它只能给版本号在 $[l_i, r_i]$ 内的机甲装配。一个机甲至多只能装配一个武器。

现在，你需要帮助 Y 国分配武器，使得尽可能多的机甲装配了武器。在此基础上，你还需要使得被装配的武器的攻击力和最大。

对于 100% 的数据，满足 $1 \leq n \leq 500, 1 \leq m \leq 10^6, 1 \leq l_i \leq r_i \leq n, 1 \leq a_i \leq 10^3$ 。

Solution: 武器 (weapon)

不难想到所求就是一个最大权匹配，考虑动态地加武器，维护匈牙利算法。直接动态加显然是不对的，应该按照 a_i 降序往里加，可以塞到匹配中就塞进去，这样一定是最优的。

不妨设机甲为左部点，每次加完之后维护一下每个左部点是否可以引出增广路，那么每次想要判断某个武器是否可以加进去的时候，本质上就是判断区间中点是否可以引出增广路，前缀和后可以 $\mathcal{O}(1)$ 判断。

这样，我们只需要增广不超过 n 次，总时间复杂度 $\mathcal{O}(n^3 + m \log m)$ 。

病毒 (virus)

W 国决定攻略 Y 国的网络系统。

Y 国的网络系统由 n 台编号依次为 $1, 2, \dots, n$ 的服务器组成。为了防止被入侵，Y 国每天都会更改网络结构，具体的，初始时服务器之间都没有边，然后 $\forall 2 \leq i \leq n$ ，第 i 台服务器会从编号在 $[1, i-1]$ 内的服务器中等概率选出一个服务器，向其连一条有向边。显然，最后 Y 国的网络系统会呈现出一个以 1 为根的内向树结构。

W 国早就在 Y 国的若干台服务器中种下病毒，假如有一天，存在某个点到 1 的路径上出现了所有带病毒的服务器，那么 Y 国的网络系统就会崩溃。现在，Y 国请求你帮他们计算，每次更改完网络结构后，网络系统崩溃的概率。

由于 Y 国并不知道哪些服务器被种了病毒，所以他们只能对 W 国的行为进行猜测后来询问你。具体的，Y 国会询问你 q 次，每次给定一个点集 S_i ，你要求出，假如只有 S_i 内的服务器中了病毒，那每次更改完网络结构后，网络系统崩溃的概率是多少。

对于 100% 的数据，满足 $1 \leq n, q \leq 3 \times 10^5, 1 \leq \sum k_i \leq 3 \times 10^5, 1 \leq k_i \leq n$ 。

Solution: 病毒 (virus)

不妨强制让 1 号点带病毒，答案不变。

之后设计一个 dp ，设 f_i 为从 1 到 i 考虑到第 i 号点时，前面的点是满足要求的方案数，若 i 带病毒，它必须接到之前的链的末尾，有 $f_i = f_{i-1}$ ，否则可以随意， $f_i = f_{i-1} \times i$ ，最后除以总方案数就是概率了。直接暴力 dp 是 $\mathcal{O}(nq)$ 的，注意到每个询问中，两个病毒结点之间的转移是可以合并的，阶乘预处理一下，时间复杂度 $\mathcal{O}(\sum k_i)$ 。

同时，我们还得到了一个结论：若带病毒的结点编号为 $id_{1..m}$ ，那么答案即为 $\prod_{i=1}^m \frac{1}{id_i}$ ，这其实也是可以直接得到的：每个带病毒的点只需要和之前链的末尾接上，对答案概率的贡献是 $\frac{1}{id}$ ，而中间的非病毒节点怎么接是无所谓的。

【ARC093F】Dark Horse

有 2^N 个人，按照满二叉树的形态进行淘汰赛，一开始的排列顺序为所有 $(2^N)!$ 个排列之一。你是第 1 个人，已知每一对人之间的实力关系，具体地说：

- 给出 M 个人 $A_1 \sim A_M$ 。
- 这 M 个人都打得过你。
- 你打得过除了这 M 个人之外的所有其他人。
- 对于剩下的情况（你不参与的情况），编号小的人胜利。

问你在所有的 $(2^N)!$ 种情况中，有多少种情况可以取得最终胜利。答案对 $10^9 + 7$ 取模 $1 \leq N \leq 16, 0 \leq M \leq 16$

Solution: 【ARC093F】Dark Horse

把这个满二叉树画出来，不难发现 1 号点在哪个位置是无所谓的，将 1 号固定在最左端后，方案数乘 2^n 就是最终的答案。

1 号点每次和别的点淘汰赛的时候，对面的点一定是它们那 2^k 个点里的最小值， k 取遍 $[0, n-1]$ ，于是问题化为将 $[2, 2^n]$ 划分为 n 组，第 i 组的人数为 2^{i-1} ，每一组的最小值都不等于 A_i 的方案数。为了方便，我们让组的编号从 0 开始，第 i 组的人数为 2^i 。

直接做是困难的，考虑容斥：设 f_S 表示钦定集合 S 中的组的最小值是某个 A_i 的合法方案数，答案为

$$ans = \sum_S (-1)^{\text{popcount}(S)} f_S$$

问题化为求 f_S ，思路肯定是尝试往里填 A_i 的同时将剩下的 $2^k - 1$ 个数填进去，可以用一个组合数算，但我们没办法知道有多少个还没有被填进去的数是大于 A_i 的。

要解决这个问题，我们可以从状态的设计入手，设 $g_{i,S}$ 为从大到小考虑 A_i ，已经考虑了 i 个，同时满足了集合 S 中组已经被填满了的方案数，转移的时候，由于我们是从大到小考虑 A_i 的，因此填进去的数一定都大于当前考虑的这个数，转移方程为

$$f_{i,S} = f_{i-1,S} + \sum_{0 \leq j \leq n-1, j \in S} f_{i-1,S-j} \times \binom{2^n - A_i - (S-j)}{2^j - 1} \times (2^j)!$$

这里整数减去一个集合的含义是将集合看做一个二进制整数。用刷表法，时间复杂度为 $\mathcal{O}(nm2^n)$ 。

【ARC108E】Random IS

从左到右排列了 N 张椅子，第 i 张的编号为 a_i （保证 a_i 互不相同）。

Snuke 想要标记一些椅子并把剩下的丢掉，一开始所有椅子都没有被标记。我们称一种标记方案是好的，当且仅当其标号递增。即，若标记的编号为 $i_1 < i_2 < \dots < i_k$ ，则有 $a_{i_1} < a_{i_2} < \dots < a_{i_k}$ 。

Snuke 将重复一下操作来标记椅子：

1. 称 x 是不错的当且仅当把 x 加入后标记方案仍是好的，记其数量为 k ；
2. 若 $k = 0$ 结束操作，否则均匀随机选择一个标记并继续操作 1；

求最终标记个数的期望，模 $10^9 + 7$ 。

$1 \leq N \leq 2000, 1 \leq a_i \leq N$

Solution: 【ARC108E】Random IS

若我们第一次选择了 a_x ，那么 a_x 将这个序列分成了左右两边，互相独立，这启发我们用类似区间 dp 的思路。

具体地，设 $f_{l,r}$ 表示若 a_{l-1} 和 a_{r+1} 被选择之后，在区间 $[l, r]$ 中期望选出多少个数，特别地，设 $a_0 = -\infty, a_{n+1} = +\infty$ 。

转移方程枚举第一个选出来的位置：

$$f_{l,r} = \frac{\sum_{i=l}^r [a_{l-1} < a_i < a_{r+1}] (f_{l,i-1} + f_{i+1,r} + 1)}{\sum_{i=l}^r [a_{l-1} < a_i < a_{r+1}]}$$

暴力做是 $\mathcal{O}(n^3)$ 的，考虑用数据结构优化：将三个部分分别考虑，以 $\sum_{i=l}^r [a_{l-1} < a_i < a_{r+1}] f_{l,i-1}$ 为例，开 n 个树状数组，按照区间 dp 的转移顺序，先算小区间后算大区间，每次算完一个 $f_{l,r}$ ，将 $(a_{r+1}, f_{l,r})$ 挂到第 l 个树状数组上，查询时，只需要查询第 l 个树状数组中满足第一维在区间 $[a_{l-1} + 1, a_{r+1} - 1]$ 的元素的第二维和，实际上就是做一个二维偏序。

时间复杂度 $\mathcal{O}(n^2 \log n)$ 。

Luogu P9499 「RiOI-2」change

给定 n 种物品，每种物品 i 价值为 v_i ，个数为 c_i 。

定义总价值为 $\sum_{i=1}^n c_i v_i$ ，你可以进行一些（可能为 0）次操作来最大化总价值。

一次操作为：选定一个 i 满足 $c_i \geq x_i$ ，让 $c_i \leftarrow c_i - x_i$ ， $c_{i+1} \leftarrow c_{i+1} + 1$ 。

输出最大的总价值对 998,244,353 取模。

对于所有数据， $1 \leq t \leq 10^5$ ， $2 \leq n$ ， $\sum n \leq 2 \times 10^5$ ， $1 \leq x_i \leq 10^9$ ， $0 \leq c_i, v_i \leq 10^9$ 。

Solution: Luogu P9499 「RiOI-2」change

维护一个 (v, c) 的二元组，从前往后不断往里加元素，每遇到一个 (v_i, c_i, x_{i-1}) ，将当前组里从后往前每 x_{i-1} 个元素合并为一个新的元素，第一维为其代价，第二维为可以以这个代价合并出的 i 元素的个数，再从后往前考虑，每当我们维护的二元组末尾的 v 要比当前 v 更小，我们就用这个代价换出来元素 i ，不断进行这个操作直到不能做。

首先我们来证明，每次维护完二元组后，它们的 v 是单调不升的：最开始只有一个元素时显然成立，往里加元素之前我们会先合并，由于本来就单调不升，合并后一定同样单调不升，之后在往二元组末尾加入 (v_i, c_i) 前，我们保证了末尾元素的 v 都大于 v_i （否则它会被合并到 (v_i, c_i) 当中），证毕！

之后我们来证明这是最优的：我们所做的操作都不会使答案更劣，同时不劣于其他的操作。

但是这样做的复杂度是平方的，注意到当 $x_{i-1} = 1$ 的时候我们没必要在加入 (v_i, c_i) 之前重新合并二元组，而当 $x_{i-1} > 1$ 的时候，二元组至多合并 $\mathcal{O}(\log v)$ 次，因此只在 $x > 1$ 的时候合并，时间复杂度 $\mathcal{O}(n \log v)$ ，具体实现时还有一个细节：当合并完的 v 要大于 v_{max} 时直接删掉它即可。

【ARC126E】Infinite Operations

你有一个长度为 n 的正整数序列 A_1, A_2, \dots, A_n 和 q 次修改，各个修改之间相互影响。

第 i 次修改给定 x_i, y_i ，你需要把 A_{x_i} 改为 y_i ，然后输出下列问题的答案：

- 令 $f(n)$ 表示进行 n 次如下操作所能得到的最大分数：选择两个下标 $i \neq j$ 满足 $A_i < A_j$ 和一个正实数 x 满足 $A_i + 2x \leq A_j$ 并将 A_i 变为 $A_i + x$ ，将 A_j 变为 $A_j - x$ ，得到 x 分。
- 可以证明， $\lim_{n \rightarrow +\infty} f(n)$ 存在且为有理数，输出其对 998244353 取模后的结果。

对于 100% 的数据，满足 $2 \leq n, q \leq 3 \times 10^5, A_i \leq 10^9$

Solution: 【ARC126E】Infinite Operations

令 $f = \sum_{i=1}^n \sum_{j=i+1}^n |A_i - A_j|$ ，答案即为 $\frac{f}{2}$ ，下面给出证明。

不难发现上式与 A 的排列顺序无关，因此将它们升序排列，现在满足 $A_1 \leq A_2 \leq \dots \leq A_n$ 。

定义一次操作是好的，当且仅当它操作的两个下标 i, j 相邻，即 $|i - j| = 1$ 。

引理 1：一次好的操作可以让 f 变为 $f - 2x$ 。

证明：对于其他的数，它们对答案的贡献不变 ($|A_k - A_i| + |A_k - A_j|$ 不变)，对于 A_i, A_j 这一对数，它们对答案的贡献减少了 $2x$ ，证毕。

引理 2：一次操作至少让 f 减少 $2x$ 。

证明：不妨设操作的两个下标为 $i < j$ ，若 $j - i = 1$ ，则这个操作是好的，由引理 1，命题成立，否则存在 $k \in (i, j) \cap \mathbb{Z}$ 。对于 $k' < i$ 或 $k' > j$ ，它们对答案的贡献是不变的，对于 $k \in (i, j) \cap \mathbb{Z}$ ， $|A_k - A_i| + |A_k - A_j|$ 即 A_k 到 A_i, A_j 在数轴上的距离和一定会减少，那么 f 也会相应地减少，证毕。

因此每次选择一个好的操作使得 f 减少 $2x$ 的同时获得 x 的分数是最优的，由于每次操作后 f 都会减少直至为 0，之后所有数都相同，无法操作，所以最终结果一定。

问题化为怎样快速地求 f ，当满足 $A_1 \leq A_2 \leq \dots \leq A_n$ 时，化一下式子，有 $\frac{f}{2} = \sum_{i=1}^n i A_i - \frac{n+1}{2} \sum_{i=1}^n A_i$ 。减号后面是容易维护的，关键在于减号左边，这里的 i 实际上应该是 A_i 的排名，每次的操作可以看作删除一个数、插入一个数，对应到排名即为某些数的排名 -1 ，某些数的排名 $+1$ ，动态开点的值域线段树维护，每个结点上的信息为区间答案、区间和、区间中值的个数，时间复杂度 $\mathcal{O}((n + q) \log n)$ 。

杜鹃花

给你一个竞赛图，点数为 n ，求

$$\min_{i=1}^n \left(\max_{j=1}^n \text{dist}(i, j) \right)$$

其中 $\text{dist}(i, j)$ 指从 i 走到 j 经过边数的最小值，无法到达则为 $+\infty$ ，可以证明答案一定不是 $+\infty$ 。

对于 100% 的数据，满足 $1 \leq n \leq 10000$ ，时限为 1 秒，要求常数很小。

Solution: 杜鹃花

首先，点数为 1 时答案为 0，下面我们证明：点数不为 1 时，答案一定不会超过 2。

证明也是容易的，假如答案超过了 2，我们就任取一个最短距离超过 2 的点，并将它设为起点，答案会更优，一直调整即可。

什么时候答案为 1 呢？当且仅当存在一个点，它连向所有其他点，这是容易判断的。

时间复杂度 $\mathcal{O}(n^2)$ ，常数几乎为 1.

斐波那契

数列 f 满足

$$f_1 = 1, f_2 = 2$$

$$f_n = f_{n-1} + f_{n-2}, \quad \text{when } n \geq 3$$

若对于整数 c ，有 $c = \sum_{i \geq 1}^n a_i \times f_i$ ，其中 $a_i \in \{0, 1\}$ ，并且 a 中没有两个相邻的 1，则称 01 串 a 为 c 的斐波那契表示，例如 $114 = 1 + 3 + 21 + 89$ ，其斐波那契表示为 1010001001，可以证明一个数的斐波那契表示唯一存在。

现给出两个数 x, y 的斐波那契表示 a, b ，你需要求出 $x + y$ 的斐波那契表示。
对于 100% 的数据，满足 $|a|, |b| \leq 10^6$ 。

Solution: 斐波那契

首先考虑一个简化的问题：将 x 的斐波那契表示的某一位加一，求出新的合法表示，例如将 101 的第二位加一，新的斐波那契表示应该为 1001 而非 111。

设将第 i 位加一的操作为 $Add(i)$ ，我们大力分类讨论：

- 若 $a_i = 0$ ，只需要考虑 $i - 1, i + 1$ 是否为 1：若 $a_{i+1} = 1$ ，就将 a_i, a_{i+1} 置为 0 后 $Add(i + 2)$ ，否则若 $a_{i-1} = 1$ ，就将 a_i, a_{i-1} 置为 0 后 $Add(i + 1)$ ，否则只需要直接将 a_i 变为 $a_i + 1$ 。
- 若 $a_i = 1$ ，那么就将操作 $Add(i)$ 变为 $Add(i - 1)$ 和 $Add(i - 2)$ 两个操作。

注意当 $i = 1$ 和 $i = 2$ 时需要特殊处理一下。

这样的做法看起来是 $\mathcal{O}(n^2)$ 的，其中 n 是串长，但是我们可以加上一些转移技巧，例如，初始时答案为 0，从高位向低位考虑，每次让答案的某一位加零、加一或加二。

例如当 $a = 101, b = 001$ ，初始时答案为 0，从高位开始考虑， a 和 b 的第三位都为 1，那么就让答案的第三位加两次 1 得到 1001，之后考虑第二位，两个都为 0，直接跳过，最后是第一位， a 的第一位是 1 而 b 不是，所以只需要加一次，答案变为 0101。

至于怎样将答案的某一位加一，用之前的递归方法即可，接下来证明这样做的时间复杂度为 $\mathcal{O}(|a| + |b|)$ 。

首先考虑加一时 $a_i = 0$ 的情况，由于我们是从高位向低位考虑的，每次递归都会使答案的 1 的数量减一，因此这部分的递归是线性的。

然后考虑 $a_i = 1$ 的情况， $Add(i - 2)$ 一定会转化到 $a_i = 0$ 的情况，于是只需要考虑另一部分 $Add(i - 2)$ 。

引理：当加完第 t 位置后，答案串的前 t 位只可能形如下面五种情况：

- 00..00000
- 00..00010
- 00..00001
- 00..00100
- 00..00101

证明可以考虑从后往前归纳，这里略去。

由这个引理，第二部分的每次递归至多进行一次，因此这部分也是线性的，总时间复杂度就是线性的。

Red Haze

给定一棵结点编号为 $1 \sim n$ 的树和 q 次询问，每次询问给一个集合 S ，求满足 $u, v \in S$ 且 u, v 之间有边的无序点对 (u, v) 的数量。

对于 100% 的数据，满足 $1 \leq n \leq 2.5 \times 10^5, \sum |S| \leq 10^6$ 。

Solution: Red Haze

脑筋急转弯题，一个简单的想法是将 $|S|$ 中的每个点都标记一下，答案就是两端都被标记的边的数量，暴力做是 $\mathcal{O}(nq)$ 的。

但是这样真的太蠢了：随便指定一个根，满足条件的边必然是一个点和其父亲之间连边的形式，枚举每个 $|S|$ 中的点，若其父亲被标记就让答案加一即可，时间复杂度 $\mathcal{O}(n + \sum |S|)$ 。

三色堇

有 n 个质数 $p_1 < p_2 < p_3 < \dots < p_n$ ，其中 $p_1 \leq 100$ ，称 x 是好的，当且仅当 $x = 0$ 或 $\exists i \in [1, n] \text{ s.t. } p_i \mid x$ 。

现在将 $[0, \prod_{i=1}^n p_i]$ 中所有好的数升序拿出来构成序列 a ，设其长度为 m ，你需要求出

$$\sum_{i=1}^{m-1} (a_{i+1} - a_i)^2$$

答案对 998244353 取模。

对于 100% 的数据，满足 $1 \leq n \leq 10^5$, $p_i \leq 10^{18}$, $p_1 \leq 100$

Solution: 三色堇

注意到 $p_1 \leq 100$ 这个很奇怪的条件：这说明“好的数”是很稠密的，稠密到相邻的好数之间的距离不会超过 100。这有什么用呢？注意到我们要求的其实就是距离平方的和，因此我们可以设 c_i 表示差为 i 的两个相邻的好数的个数，答案即为

$$\sum_{i=1}^{p_1} c_i \times i^2$$

关键在于求出 c_i ：我们尝试增量地去做，初始时只有 p_1 ，有且仅有 $c_{p_1} = 1$ ，其他全为 0，在之后不断加入 p_2, p_3, \dots 并维护 c 。

考虑加入 p_i 时发生了什么，首先，我们要求的“好数”的范围从 $\prod_{j=1}^{i-1} p_j$ 扩大到了 $\prod_{j=1}^i p_j$ ，假如只有值域的放大而没有新质数的加入，对 c 的影响就是让每个 c_j 变为 $c_j \times p_i$ ：这是容易理解的，因为本质上其实就是将原来的东西复制了 p_i 份，为了之后说明的方便，下文中的 p 就是我们新加进来的这个质数，即上文的 p_i 。

我们必须考虑新加进来的质数导致某些数变为了好数：我们新加进来的这些数实际上会使原本的一些“相邻的好数”变得不再相邻，这些数是 $p, 2p, 3p, \dots, \prod_{j=1}^{i-1} p_j \times p$ ，显然这些数字中会有一些数本来就是好数，我们先不管这些细节，只关注这个整体：由于 p_i 和 $\prod_{j=1}^{i-1} p_j$ 互质，因此 $p, 2p, 3p, \dots, \prod_{j=1}^{i-1} p_j \times p$ 在模 $\prod_{j=1}^{i-1} p_j$ 意义下恰好是 1 到 $\prod_{j=1}^{i-1} p_j$ 的一个排列，这又表明我们可以将这些数对 c 的影响缩小了来看，即在值域没有变化之前的时候。

具体的说，因为这构成了一个排列，所以原本的每两个相邻的好数 $x < y$ ，他们中的每个值 k 满足 $x < k < y$ ，都会且仅会在之后的 p 份复制品中被“变成好数”一次。

由此可以导出 c_i 的变化，即：

$$c_i = c_i \times (p - i + 1) + 2 \left(\sum_{j=i+1}^{p_1} c_j \right)$$

可能这步有点快，具体分析一下这个式子：首先让 c_i 变成 $c_i \times (p - i + 1)$ 是为什么呢？第一，原本的值域扩大了 p 倍，因此每一个原本的长度为 i 的区间都会被复制 p 次，所以 c_i 应该先变成 $c_i \times p$ 。其次，一共有 $i - 1 \times c_i$ 个位置是位于这 c_i 个区间之间的，根据我们之前的分析，每个位置都会被“变成好数”一次，这意味着区间两端的两个好数在变化后不再相邻，不应该被计入答案，这部分的贡献就是 $-(i - 1) \times c_i$ ，两部分合起来即可。

那后面的 $+2 \left(\sum_{j=i+1}^{p_1} c_j \right)$ 又是什么呢？别忘了，将一个数变成好数后，我们将原本的相邻好数构成的区间从答案中删掉之后，还没有考虑新得到的两个区间：每个长度大于 i 的区间，都可以两次对长度为 i 的区间产生

贡献（考虑对称的两个点），因此要再加上这部分。

总的时间复杂度就是 $\mathcal{O}(p_1 n)$ 。

矢车菊

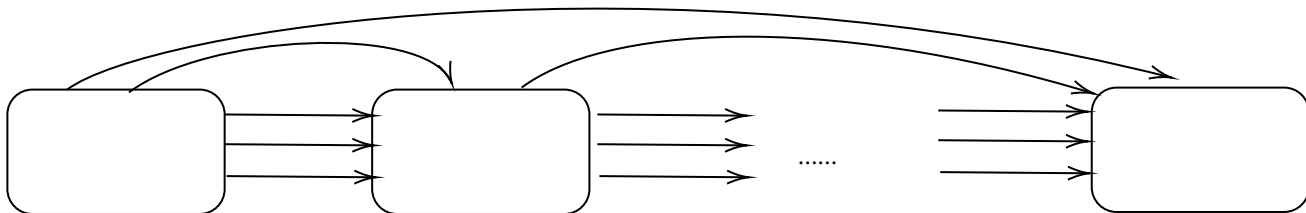
现在有一个点从 1 到 n 标号的竞赛图，初始时对于任意 $i < j$ ，都有一条从 i 到 j 的有向边。之后给出 q 次操作，每次操作会给出两点 u, v ，你的任务是反转这两点间有向边的方向，并在反转后求出有多少对点 u, v 满足 $u < v$ 且它们相互可达，操作之间互相影响。

对于 50% 的数据，满足 $1 \leq n, q \leq 3 \times 10^3$ ，对于 100% 的数据，满足 $1 \leq n, q \leq 5 \times 10^5$

Solution: 矢车菊

我们先看 50% 的数据应该怎么做：所求的相互可达的点对数量实际上是什么呢？实际上就是将原图强连通分量缩点后，每个强连通分量的点对数量的和，而若某个强连通分量的点数确定了，其点对数也是好算的，就是 $\binom{m}{2}$ ，其中 m 表示该强连通分量的点数。

做完这个转化之后我们再来考虑怎么在每次修改后 $\mathcal{O}(n)$ 求出每个强连通分量的点数。



上图中的每个方块代表一个强连通分量，注意这张图有什么特点：根据竞赛图缩点后必然是一个完全拓扑图的性质，我们选取这个拓扑图的后 k 个强连通分量（在上图中即为从右到左的 k 个方块），这些点的出度数恰好为 $\binom{m}{2}$ ，其中 m 表示这 k 个强连通分量中点的数量。这也是好理解的，任选两个这 m 个点之间的点，它们之间的边会贡献一个出度，若选出了这 m 个点以外的点，出度必然不会贡献到这 m 个点之间的点（因为这是按照拓扑序来的，拓扑序后的点不会有指向拓扑序之前的点的边）。

于是我们得到了一个非常巧妙的思路：将所有点的出度升序排列后，若前 k 的出度和恰好为 $\binom{k}{2}$ ，说明我们到了某个强连通分量的交接处，将答案加上 $\binom{k - LastK}{2}$ 即可，其中 $LastK$ 指上一个满足条件的 k ，初始为 0，而出度的变化是很容易维护的。这样做的时间复杂度是 $\mathcal{O}(qn \log n)$ 的，足够过掉 50% 的数据。

大体的思路已经有了，现在考虑优化。将问题抽象，我们现在要解决的是：

给定一个长度为 n 的序列 a ，最开始 $a_i = i - 1$ ，每次操作会指定两个 i, j 并使 $a_i \leftarrow a_i + 1, a_j \leftarrow a_j - 1$ ，操作后设 b 为 a 升序排序后的序列，满足 $\sum_{i=1}^k b_i = \binom{k}{2}$ 的下标 k 升序构成长度为 m 的序列 p ，求

$$\sum_{i=2}^m \binom{p_i - p_{i-1}}{2}$$

由于有一个将原数列排序的操作，所以应该把数据结构开在排名上，具体地，让 $c_i = b_i - (i - 1)$ ，所求下标 k 转化为满足 $\sum_{i=1}^k c_i = 0$ ，这是一个前缀和的形式，考虑直接维护，设 s 为 c 的前缀和数组，最开始显然全为 0，每次单点加一、减一并不会影响这个值的排名，因此在 c 上同样表现为单点加减一，在 s 上表现为后缀加减一，容易线段树维护 s 。

维护好了 s ，问题在于求答案，即怎样合并两个区间。如果只维护第一个 0 的位置，最后一个 0 的位置和答案，在合并时出现的问题是：我们无法得知新区间的第一个 0、最后一个 0 的位置，因为某个区间中可能没有 0 但我们无法判断。

这里要用到一个性质，考虑到我们维护的 a 实际上是竞赛图中点的出度，也就是说 $s_i \geq 0$ 是恒成立的（考虑前 k 个点的出度至少为 $0 + 1 + 2 + \dots + (k - 1)$ ），也就是说，判断一个区间内是否有 0，等价于判断这个区间的最小值是否为 0，之后分三类合并：左区间没有 0，右区间没有 0，左右都有 0 来合并区间信息。

总时间复杂度 $\mathcal{O}(n \log n + q \log n)$ 。

CF1860 D.Balanced String

给你一个长为 n 的 01 字符串，每次操作交换两个元素，求最少几次操作能使字符串 01 对数量等于 10 对数量，例如在串“1001”中，10 对和 01 对分别有两个。

对于 100% 的数据，保证有解，同时 01 串的长度不超过 100。

Solution: CF1860 D.Balanced String

显然是一个 dp，但是这里很容易想歪：很容易注意到交换一个下标分别为 i, j 的 01 对会使“01 对数量”“减 10 对数量”变化 $j - i$ ，之后就没有后文了。

尝试换一个思路：每对 0 和 1 一定会对“01 对”和“10 对”的其中之一造成贡献，那么两者的和应当是固定的，设 c_0, c_1 分别表示串中 0 的数量和 1 的数量， c_{01}, c_{10} 表示 01 对和 10 对的数量，那么有

$$c_{01} + c_{10} = c_0 \times c_1$$

恒成立，而根据题意，应当满足 $c_{01} = c_{10}$ ，于是我们只需要满足 $c_{01} = \frac{c_0 \times c_1}{2}$ 即可。

我们关注怎样计算 c_{01} ，对于每个 $a_i = 1$ ，它的贡献就是前面 $i - 1$ 个数中 0 的个数，我们不妨将前面的 1 也算进去，最终减去即可，那最终减去的实际就是 11 对的数量，这也是固定的，因此有

$$c_{01} = \left(\sum_{i=1}^n [s_i = 1](i - 1) \right) - \frac{c_1(c_1 - 1)}{2}$$

变形有

$$c_{01} = \left(\sum_{i=1}^n [s_i = 1]i \right) - \frac{c_1(c_1 + 1)}{2}$$

于是我们的要求等价于

$$\sum_{i=1}^n [s_i = 1]i = \frac{c_0 c_1}{2} + \frac{c_1(c_1 + 1)}{2} = \frac{(n + 1)c_1}{2}$$

于是就可以 dp 了，设 $f_{n,i,j}$ 表示考虑前 n 个位置，填了 i 个 1，这些 1 的下标和为 j 时，和原串最少有几位不同，转移平凡，答案为 $\frac{f_{n,c_1,(n+1)c_1/2}}{2}$ ，除以二是因为我们每次可以交换一对 01，消除两个差异。

时间复杂度 $\mathcal{O}(n^4)$ ，这是因为状态的第三维是 $\mathcal{O}(n^2)$ 的。用一下滚动数组优化空间，空间复杂度 $\mathcal{O}(n^3)$ 。

CF1860 F.Evaluate RBS

现有 $2n$ 个形如 (a, b, c) 的三元组，其中 a, b 均为正整数， c 是字符（或）（左圆括号或右圆括号）。其中恰有 n 个三元组的 c 为（，另外 n 个为）。

对于两个正实数 x, y ，记一个三元组的特征值为 $\lambda = ax + by$ 。你需要选择 x, y 并对三元组按其特征值升序排序。若多个三元组的特征值相同，可以选择随意排列。

问，是否存在一种选法，使得排序后的序列能让 c 组成一个合法的括号序列。

对于 100% 的数据，保证 $1 \leq a, b \leq 10^6, 1 \leq n \leq 1500$

Solution: CF1860 F.Evaluate RBS

首先，每个括号的 (a, b) 可以看作是一个向量，我们选择的 (x, y) 也可以看作是一个向量，排序方式就是按照点积升序，根据二维向量点积的公式 $\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos \langle \mathbf{a}, \mathbf{b} \rangle$ ，这只会和 (a, b) 的模长与所选向量的方向有关。

因此, 我们不妨让所选向量从 x 轴正方向逆时针转到 y 轴正方向 (两边都是开区间, 因为要求 $x > 0, y > 0$, 换句话说, 取不到坐标轴), 在转的过程中, 某些括号的大小关系会变化, 因此我们只关心会使某些括号的大小关系发生变化的那些角度, 对每两个括号间解一个方程可以得到至多一个位置, 因此这样的角度有 $\mathcal{O}(n^2)$ 个.

对每个角度做一次排序 (若点积相同, 让 (排在前面), 让 (变成 1,) 变成 -1, 括号序列合法当且仅当任意前缀和都不小于 0, 可以 $\mathcal{O}(n)$ 判断, 总时间复杂度 $\mathcal{O}(n^3)$, 还不够.

考虑增量维护, 我们之前说过, 让选择的向量 (x, y) 从 x 轴正方形向 y 轴正方形转, 每次转到一个新位置, 我们只需要对改变了大小关系的点排序后, 重新插入原序列即可, 由于每个点至多被排序 $\mathcal{O}(n)$ 次, 总时间复杂度 $\mathcal{O}(n^2 \log n)$.

交上去发现挂了, 这是因为我们排序时 “若点积相同, 让 (排在前面” 的策略, 当角度再次转了之后, 之前按照这个策略排序的点并没有即使纠正, 因此每次排 “该角度发生变化的点” 和 “上一个角度发生变化的点” 即可, 时间复杂度仍然是 $\mathcal{O}(n^2 \log n)$, 常数乘 2.

Acrobat

给定两棵 n 个结点的树 T_1, T_2 , 它们的结点编号均为 $1 \sim n$.

对于任意两点 u, v (可相同), 设 $f_k(u, v)$ 为在 T_k 中 u 到 v 的最短路径包含的点集 (包含 u, v).

对于任意一个树上所有点的子集 S , 设 $F_k(S)$ 为 $\bigcup_{u, v \in S} f_k(u, v)$, 即 S 中任选 2 点 (可相同) 对应的 f_k 的并集.

请你求出

$$\sum_{S \subseteq \{1, 2, \dots, n\}} |F_2(F_1(S))|$$

对 $10^9 + 7$ 取模的结果.

对于 100% 的数据, 保证 $1 \leq n \leq 1.5 \times 10^5$.

Solution: Acrobat

套路地考虑每个点的贡献, 即对于每个点 u , 有多少集合 S 满足 $u \in F_2(F_1(S))$, 设为 c_u , 假如能够算出 c_u , 那么答案即为

$$\sum_{u=1}^n c_u$$

考虑 S 满足要求的充要条件: 设 $F_1(S) = T$, 那么 S' 应满足: $u \in S'$ 或者在 T_1 删去 u 后形成的若干连通块中, S' 中的点不全在同一个连通块. 正难则反, 考虑 S' 不满足条件当且仅当在 T_1 删去 u 后形成的若干连通块中, S' 全在同一个连通块内. 对每个连通块分别考虑, 设某个连通块的大小为 x , 那么全在这个连通块内的 S' 的个数有 $2^x - 1$.

钦定 1 号点为 T_1, T_2 的根, 那么删去点 u 形成的若干连通块一定是某个子树或者整棵树删去某个子树的形式, 先考虑前者: 对每个 T_2 的子树, 求出有多少集合 S 满足 $F_1(S)$ 全在该子树中.

用可撤销的按秩合并并查集维护连通性, 每次加入一个点或删除一个点时, $\mathcal{O}(d_u \log n)$ 维护并查集, 其中 d_u 表示点 u 的度数, 然后套一个 dsu on tree 即可.

具体来说, 将 T_2 轻重链剖分, 从上往下 dfs, 每次先求出轻子树的答案, 然后求重儿子的答案 (求出重儿子的答案后, 不还原并查集, 但求出轻儿子的答案后, 还原并查集), 然后再将所有轻子树中的点加入到并查集中, 求出答案. 由于每个点至多被加入并查集 $\mathcal{O}(\log n)$ 次, 所以总时间复杂度为 $\mathcal{O}(n \log^2 n)$.

之后再考虑后者: 即对每个 T_2 的子树, 求出有多少集合 S 满足 $F_1(S)$ 于该子树无交. 同样地从上往下 dfs T_2 , 不同的是, 我们这次对于每个重链, 一起求出答案.

具体来说, 我们只考虑每个重链的链顶 (此时应满足树中所有不在该链顶的子树中的点均被加入并查集中), 最开始考虑的是根节点, 此时, 我们顺着重链不断向下走, 每走到一个结点, 往并查集中加入其所有轻子树和

该结点，得到其重儿子的答案，这一条重链的答案就被求出来了。之后考虑中途加入的所有轻子树，按照子树大小排序，每次找出带权中点后分成左右两部分。先将左部分中的点全部加入并查集，递归算右部分，之后撤销左部分的并查集，将右部分加入并查集，递归算左部分，若递归到单个子树，重复之前的 dfs 过程。

由于我们按照带权中点来划分，因此在递归树上每向上两层，大小必然翻倍，因此每个结点同样只会被加入并查集 $\mathcal{O}(\log n)$ 次，总时间复杂度 $\mathcal{O}(n \log^2 n)$ 。

两部分分别处理后合并即可，总时间复杂度 $\mathcal{O}(n \log^2 n)$ 。

煎鱼

有一个长度为 n 的序列 a ，一开始全 0。对于集合 S 中的每个区间 $[l, r]$ ，可以选择整数 $k \in [l, r]$ ，并将 a_k 增加 1，求 $\sum_{i=1}^n a_i^2$ 的最大值。

其中，集合 $S = \{[l, r] : 1 \leq l \leq r \leq n \text{ 且 } l, r \in \mathbb{N} \text{ 且存在 } k \in \mathbb{N} \text{ 使得 } r - l + 1 = 2^k\}$ 。
对于 100% 的数据，满足 $1 \leq n \leq 10^9$ ，对于 20% 的数据，满足 $n = 2^k - 1, k \in \mathbb{N}^*$ 。

Solution: 煎鱼

首先证明一个性质：设 c_i 为选择了 i 的区间的个数，那么 c_i 最大的那个点一定满足：所有包含 i 的区间均选择了 i 。证明考虑调整：将没有选择 i 的区间选上 i ，答案不劣。

于是就可以愉快的区间 dp 了，设 $f_{l,r}$ 表示只考虑区间 $[l, r]$ 中的 S 中区间的答案，转移考虑枚举 $[l, r]$ 中的那个最大值，若每次暴力算，时间复杂度为 $\mathcal{O}(n^4 \log n)$ 。

接下来考虑优化转移：转移的时候假如可以直接得出来区间中包含某个点的合法区间个数，时间复杂度就可以变成 $\mathcal{O}(n^3)$ ，设 g_i 表示集合 S 中包含点 i 的区间个数， $s_{l,r}$ 表示包含区间 $[l, r]$ 的合法区间个数，对于前者，差分前后缀和做到 $\mathcal{O}(n \log n)$ ，对于后者，用一个类似二维前缀和的方式， $s_{l,r} = s_{l-1,r} + s_{l,r+1} - s_{l-1,r+1} + e_{l,r}$ ，其中 $e_{l,r}$ 表示区间 $[l, r]$ 是否在集合 S 中。那么，在区间 $[l, r]$ 中且包含点 k 的集合 S 中区间数为 $c_k - s_{k,r+1} - s_{l-1,k} + s_{l-1,r+1}$ ，转移的时间复杂度变为 $\mathcal{O}(n)$ ，总时间复杂度 $\mathcal{O}(n^3)$ 。

进一步地， f 值只和区间长度有关，和区间的具体位置无关，因此可以简化状态： f_l 表示长度为 l 的答案数量，转移仍为 $\mathcal{O}(n)$ ，总时间复杂度 $\mathcal{O}(n^2)$ ，空间复杂度 $\mathcal{O}(n)$ 。

我们还没有看那 20% 的部分：手玩可以得到此时转移点恰好在 $\frac{n+1}{2}$ ，转移 $\log n$ 层，时间复杂度 $\mathcal{O}(\log n)$ 。下面我们证明一个引理。

引理：使得答案最优的转移点 k 必然满足覆盖了 k 的区间个数最多，且为满足条件的最左侧或最右侧的那个点（因为 S 中区间的对称性）。

假如这个引理是正确的，那么上面 20% 的部分分就有严格的证明而不是手玩得到的结论了，并且，对于一个长度为 n 的区间，我们可以 $\mathcal{O}(1)$ 得到其正确的转移点且通过预处理长度为 $2^k - 1$ 的区间的答案，在 $\mathcal{O}(\log n)$ 的时间复杂度内解决这个问题。

具体的，若 $n = 2^k - 1$ ，转移点即为 2^{k-1} ，否则设 $k = \lfloor \log_2(n+1) \rfloor$ ，若 $n+1 \geq 2^k + 2^{k-1}$ ，转移点为 2^k ，否则转移点为 2^{k-1} ，因为我们已经预处理出了长度为 $2^k - 1$ 的区间的答案，所以只需要递归一边，同时递归深度为 $\mathcal{O}(\log n)$ ，总时间复杂度为 $\mathcal{O}(\log n)$ 。下面只需要证明这个引理。

为了证明这个引理，我们需要再证明下面的两个引理：

引理 1：当每次选择的转移点为被覆盖最多次的点时， $dp_j + dp_i \geq dp_{j+1} + dp_{i-1}, j < i$

证明：当 i 变为 $i+1$ 时，让转移点为被覆盖最多次的点，有 $dp_{i+1} - dp_i \geq dp_i - dp_{i-1}$ ，不断展开，得到 $dp_i - dp_{i-1} \geq dp_j - dp_{j-1}, j \leq i$ ，移项得到 $dp_i + dp_{j-1} \geq dp_j + dp_{i-1}$ ，将 j 换为 $j-1$ 得到 $dp_i + dp_j \geq dp_{j+1} + dp_{i-1}, j < i$ ，证毕！

引理 2：当 dp_i 从某个被覆盖次数最多的点 k 转移过来时答案最优。

证明：由于对称性，我们只考虑在区间 $[1, \frac{i+1}{2}]$ 的点 k 。反证法，若存在一个 k' 使答案更优，则

$$dp_i = dp_{k'-1} + dp_{i-k'} + k' \text{ 在长度为 } i \text{ 时被覆盖的次数}$$

$$dp_{i-1} \geq dp_{k'-1} + dp_{i-1-k'} + k' \text{ 在长度为 } i-1 \text{ 时被覆盖的次数}$$

分类讨论：

- 当 k' 在长度为 i 时被覆盖的次数等于 k' 在长度为 $i-1$ 时被覆盖的次数时，两式相减有 $dp_i - dp_{i-1} \leq dp_{i-k'} - dp_{i-1-k'}$ ，不如从 k 转移过来的 $dp_i - dp_{i-1} \geq dp_{i-k'} - dp_{i-1-k'}$ 。
- 当 k' 在长度为 i 时被覆盖的次数不等于 k' 在长度为 $i-1$ 时被覆盖的次数时，由于新增区间的右端点均为 $i+1$ ，因此至多有一个区间会跨过中点覆盖左边的点。先去掉这个区间，用上面的结论得到被覆盖最多次的 k 优，之后考虑这个区间，它一定会覆盖到这个 k ，将它加到这个 k 上，答案更优。

由引理 1,2，引理的正确性显然。

捞饭

小 X 是带厨师，他今天要做香翅捞饭。

小 X 总共做了 n 锅香翅捞饭。第 i 锅的质量为整数 a_i ，满足 $1 \leq a_i \leq k$ 。他想从这些捞饭中选取若干锅，并品尝其中的所有捞饭，即他品尝的捞饭质量是所有被选择的锅的 a_i 的和。

小 X 想考考你，因此他会进行 q 次询问。每一次会给出 l, r, v ，表示询问小 X 在 $[l, r]$ 的区间中选择若干锅捞饭后，能享用恰好 v 质量的捞饭的方案数。保证 $1 \leq v \leq k$ 。

你需要帮助小 X 找到每一次询问的答案。由于答案可能很大，输出答案对 $10^9 + 7$ 取模的结果。

对于 100% 的数据， $n \leq 40000, q \leq 3 \times 10^5, k \leq 501 \leq v, a_i \leq k, 1 \leq l \leq r \leq n$ 。

Solution: 捞饭

突破点在于 k 只有 50，因此可以莫队：每次 $\mathcal{O}(k)$ 单点加、删，只需要维护背包，设块大小为 B ，则时间复杂度为 $\mathcal{O}\left(qBk + \frac{n^2k}{B}\right)$ ，取 $B = \frac{n}{\sqrt{q}}$ 可以做到 $\mathcal{O}(nk\sqrt{q})$ ，不是很行，并且没有优化空间了。

找一个更优秀的做法：将线段树的结构建出来，对每个结点做背包：设结点 $[l, r]$ 的 $mid = \frac{l+r}{2}$ ，那么我们存 $r-l+1$ 个背包， mid 上的背包只有一个元素 a_{mid} ，对于在 $[l, mid-1]$ 的 x ，里面有 $a_x, a_{x+1}, \dots, a_{mid}$ ，对于 $mid+1$ 上的背包，里面只有一个元素 a_{mid+1} ，对于在 $[mid+2, r]$ 中的 x ，里面有 $a_{mid+1}, a_{mid+2}, \dots, a_x$ 可以通过我们之前说的 $\mathcal{O}(k)$ 加一个元素做到 $\mathcal{O}(nk \log n)$ 建树。

建完树之后，对于每次查询，我们可以先向下走到一个完全包含查询区间且查询区间经过了 mid 的结点，那么答案就可以根据左半部分处理出的背包和右半部分处理出的背包拼出来了。

更具体的，设查询区间为 $[l, r]$ ，值为 v ，那么我们先找到一个结点 $[L, R]$ 满足 $L \leq l, R \geq r, \frac{L+R}{2} \in [l, r]$ ，根据我们预处理出的背包，答案即为

$$\sum_{i=0}^v [l, mid] \text{ 拼出 } i \text{ 的方案数} \times [mid+1, r] \text{ 拼出 } v-i \text{ 的方案数}$$

可以 $\mathcal{O}(k)$ 计算。

总时间复杂度 $\mathcal{O}(nk \log n + q(\log n + k))$ 。

虚伪的最小公倍数

给定 n, k ，求

$$\prod_{i_1=1}^n \prod_{i_2=1}^n \dots \prod_{i_k=1}^n \frac{\prod_{j=1}^k i_j}{\gcd_{j=1}^k i_j}$$

结果对 998244353 取模。

对于 100% 的数据，保证 $1 \leq n \leq 3 \times 10^5, 2 \leq k \leq 10^{18}$ 。

Solution: 虚伪的最小公倍数

分子分母分别考虑，先看比较简单的分子：

$$\prod_{i_1=1}^n \prod_{i_2=1}^n \cdots \prod_{i_k=1}^n \left(\prod_{j=1}^k i_j \right)$$

考虑每个数字出现了几次，不妨以数字 $x \in [1, n]$ 为例， i_1 为 x 的共有 n^{k-1} 次，这是因为剩下的 i_2, i_3, \dots, i_k 可以随便选， i_2, i_3 为 x 同理，因此数字 x 共被乘了 kn^{k-1} 次。

于是在上式中，这些数字的积为

$$\prod_{i=1}^n i^{kn^{k-1}}$$

这又等于

$$(n!)^{kn^{k-1}}$$

这是因为 $a^c \times b^c = (a \times b)^c$ 。

预处理一下阶乘，由费马小定理，上面的 kn^{k-1} 可以对 $MOD - 1 = 998244352$ 取模而不影响结果，快速幂，时间复杂度 $\mathcal{O}(\log k)$ 。

再看分母：

$$\prod_{i_1=1}^n \prod_{i_2=1}^n \cdots \prod_{i_k=1}^n (\gcd_{j=1}^k i_j)$$

这个比较麻烦，按照套路，先枚举 gcd，上式化为

$$\prod_{d=1}^n d^{\sum_{i_1=1}^{\lfloor \frac{n}{d} \rfloor} \sum_{i_2=1}^{\lfloor \frac{n}{d} \rfloor} \cdots \sum_{i_k=1}^{\lfloor \frac{n}{d} \rfloor} [\gcd_{j=1}^k i_j = 1]}$$

只关注上面的那个幂，莫反得

$$\begin{aligned} & \sum_{i_1=1}^{\lfloor \frac{n}{d} \rfloor} \sum_{i_2=1}^{\lfloor \frac{n}{d} \rfloor} \cdots \sum_{i_k=1}^{\lfloor \frac{n}{d} \rfloor} [\gcd_{j=1}^k i_j = 1] \\ &= \sum_{c=1}^{\lfloor \frac{n}{d} \rfloor} \mu(c) \left\lfloor \frac{n}{cd} \right\rfloor^k \end{aligned}$$

$\mathcal{O}(n \log k)$ 预处理 i^k ， $\mathcal{O}(n)$ 筛出 μ ，算分母的时间复杂度为 $\mathcal{O}(n \log k)$ 。

总时间复杂度 $\mathcal{O}(n \log k)$ ，但是原题有 T 组数据，这样的时间复杂度会超时。

考虑进一步优化，上面的那个幂是可以整除分块后整除分块的，算一下 μ 的前缀和， $\mathcal{O}(\sqrt{n} \log k)$ 预处理需要用到 x^k 即可。

诈骗

定义函数

$$f(n) = \begin{cases} n^2, & 0 \leq n \leq 9 \\ f(\lfloor \frac{n}{10} \rfloor) + (n \bmod 10)^2, & n \geq 10 \end{cases}$$

现在给定 K, L, R ，求有多少个数 $x \in [L, R]$ ，满足 $f(x) \times K = x$ 。

对于 100% 的数据，满足 $1 \leq K \leq 10^{18}$ ， $1 \leq L \leq R \leq 10^{18}$ 。

Solution: 诈骗

诈骗题，看起来像做一个数位 dp 实际上不用。注意到 $f(n)$ 就是 n 的各数位平方和，而 10^{18} 内，这个平方和最多是 18×9^2 ，因此我们枚举 $i = f(n)$ ，再检查 $f(i \times K)$ 是否等于 i 即可，时间复杂度 $\mathcal{O}(\log^3 n)$ 。

Permutation Problem

给定 n, k , 求有多少长度为 n 的 $[1..n]$ 的排列 P , 满足对于所有 $i \in [1, n]$ 都有 $|P_i - i| \leq k$, 答案对 $10^9 + 7$ 取模。

对于 100% 的数据, 保证 $0 \leq k \leq 9, 1 \leq n \leq 10^{10-k}$ 。

Solution: Permutation Problem

首先当 $k = 0$ 的时候答案为 1。

考虑怎么暴力做, 设 $f_{i,S}$ 表示填了 $[1, i]$ 到位置集合 S 上, 暴力转移, 时间复杂度为 $\mathcal{O}(n^2 2^n)$ 。

由于 i 填的位置 $p_i \leq i + k$, $[i + 1, n]$ 填的位置 $\geq i + 1 - k$, 所以我们只需要知道填完 i 后这些位置上是否被填即可, 时间复杂度变为 $\mathcal{O}(n 2^{2k})$ 。

再考虑这 $2k$ 个位置必然有 k 个位置被填了, 所以有用的状态只有 $\binom{2k}{k}$ 个, 预处理出来, 复杂度变为 $\mathcal{O}(n \binom{2k}{k})$ 。

当 $k \geq 5$ 的时候, 由于 $n \leq 10^{10-k}$, 所以这个时间复杂度是够的, 但是当 k 比较小的时候这个复杂度就不行了, 解决方法是当 $k \leq 4$ 的时候, 用矩阵快速幂优化, 这样做的时间复杂度为 $\mathcal{O}\left(\binom{2k}{k}^3 \log n\right)$ 。

出租车

出租车有 k 个停靠点, 这 k 个停靠点依次排成一条直线, 从左到右编号为 1 到 k 。出租车一次最多带 4 个人。共有 n 个人正在等待乘车, 每个人都想从某个停靠点, 去往另一个停靠点。这 n 个人到达各自上车的停靠点的时间均不相同, 我们按到达时间的前后依次给这 n 个人编号为 1 到 n , 即先到的编号小。出租车公司有一个规定, 先到的人必须比后到的人先上车, 当然下车顺序则任意。

出租车每次可以移动到相邻编号的停靠点, 需要耗费一个单位的时间, 每个人上车或者下车都需要耗费一个单位时间, 出租车一开始为空, 且位于 1 号停靠点, 出租车最后停靠点任意, 那么将则 n 个人送到目的地至少需要多少时间呢? (注: 题目保证出租车到达时需要上车的人已到达出发点)

对于 100% 的数据, 保证 $n \leq 2000, k \leq 10$ 。

Solution: 出租车

首先, 每个人都要上车一次, 下车一次, 花费的时间是 2, 因此可以先将答案加上 $2n$, 让每个人上下车不花费时间。

k 很小, 因此我们可以只关注车上的人要到的地方而不关注他到底是谁, 设 $d_{i,p,j,k,l,m}$ 表示下一个需要上车的是 i , 车当前在位置 p , 车上坐着 j, k, l, m 所需的最小时间, 当 $j, k, l, m = 0$ 意味着这些位置上现在没有人, 转移可以考虑是接上 i 还是将谁送到目的地, 时空复杂度是 $\mathcal{O}(nk^5)$, 接受不了。

考虑缩减一下状态数, 当车上坐满四个人的时候, 下一步必然是到某一个下车点把一个人踢下去, 所以我们可以只让车上坐三个人, 假如某时刻车上有三个人 j, k, l , 又来了一个人 m , 我们就枚举接上 m 后会把谁踢下去, 直接转移到新的状态, 这样时空复杂度都是 $\mathcal{O}(nk^4)$ 。

不稳定的导弹系统

王国发生了叛乱，整个国家处处都有叛军，在这危急时刻，国王下令将刚研制的导弹系统投入使用，以消灭叛军。但导弹发射器分散在全国各地，每个导弹发射器只来得及配备一枚超级导弹。更糟糕的是，导弹发射器的转向器还没装上，以至于只能攻击上下左右中的某一个方向上的叛军，一枚超级导弹可以将一个单位区域的所有叛军歼灭。超级导弹将在同一时间发射，超级导弹的飞行路线不能有交叉（即不能有交点），否则一旦碰撞后果不堪设想。

现在将王国简化成一个 $N \times M$ 的网格图，每一小格代表一个单位区域，单位区域上要么是空地，要么是导弹发射器，要么有一定数量的叛军。现在国王希望能歼灭尽可能多的叛军，你需要求出这个值。

注：保证不会有导弹发射器能炸到另一个导弹发射器，输入时某个区域是 $-1, -2, -3, -4$ 表示是朝上下左右的导弹，否则表示该区域的叛军数量。

对于 100% 的数据，保证 $N, M \leq 50$ ，单位区域中的叛军不超过 1000 人。

Solution: 不稳定的导弹系统

网络流，将每个点拆为横向点和纵向点，设源点为 S ，汇点为 T ， S 向每个朝向左的导弹的横点连容量为 $+\infty$ 的边，每个朝向上的导弹的纵点向 T 连容量为 $+\infty$ 的边，每个点的横点向其纵点连容量为 $+\infty$ 的边。

对于每个向左打的炮台，设其为 (i, j) ，找到他可以打到的最多的士兵个数，设为 v ， $w_{i,j}$ 表示 (i, j) 这个位置上有多少士兵，那么对每个 $k \in [2, j]$ ，从 (i, k) 的横点向 $(i, k-1)$ 的横点连容量为 $v - w_{i,k-1}$ 的边。

对于每个向右打的炮台，设其为 (i, j) ，找到他可以打到的最多的士兵个数，设为 v ，对每个 $k \in [j, m-1]$ ，从 (i, k) 的横点向 $(i, k+1)$ 的横点连容量为 $v - w_{i,k+1}$ 的边。

对于每个向上打的炮台，设其为 (i, j) ，找到他可以打到的最多的士兵个数，设为 v ，对每个 $k \in [2, i]$ ，从 (k, j) 的横点向 $(k-1, j)$ 的横点连容量为 $v - w_{k-1,j}$ 的边。

对于每个向下打的炮台，设其为 (i, j) ，找到他可以打到的最多的士兵个数，设为 v ，对每个 $k \in [i, n-1]$ ，从 (k, j) 的横点向 $(k+1, j)$ 的横点连容量为 $v - w_{k+1,j}$ 的边。

答案为每个炮台的 v 的和再减去从 S 到 T 的最小割。

宝石

有一个魔法阵，小 L 想要激活这个魔法阵。这个魔法阵是一个树形结构，可以看成一棵 n 个点的树，根节点为 1 号点。激活魔法阵的方式是在每个节点上放宝石。每个节点上可以放红宝石或蓝宝石，每个点都必须放恰好一个宝石。在第 i 个点上放红宝石的代价为 a_i ，放蓝宝石的代价为 b_i 。想要激活这个魔法阵，放的宝石种类还要满足如下的条件：对于任意一个点，它的子树内的点上面的红宝石总数和蓝宝石总数的差的绝对值不超过 1。

求最小能使魔法阵激活的总代价。

对于 100% 的数据，保证 $3 \leq n \leq 10^6$ ， $0 \leq a_i, b_i \leq 10^9$ 。

Solution: 宝石

做一个 dp，设 $dp_{u,-1/0/1}$ 表示以 u 为根的子树中，红宝石数减蓝宝石数为 $-1/0/1$ 时所需的最小代价，转移可以做一个背包，时间复杂度 $\mathcal{O}(n^2)$ 。

考虑一个菊花图怎么做：我们先钦定全选蓝宝石，然后问题变为：要选出 k 个点，将他变成红宝石，求最小代价。

将 b_i 变为 a_i ，代价变多了 $a_i - b_i$ ，显然我们应该选 $a_i - b_i$ 最小的 k 个，用 `std::nth_element` 可以做到 $\mathcal{O}(n)$ 。

再回到原题：对于一个大小为偶数的子树，它内部的红宝石数必然等于蓝宝石数，贡献到父亲方案唯一，否则差必然为 -1 或 1 ，贡献到父亲时又变为了之前菊花图的情况，套上来即可，总时间复杂度 $\mathcal{O}(n)$ 。

三元环

给定一个 n 个点的完全图，你要给每条边定一个方向，使得得到的有向完全图（又称竞赛图）的三元环个数最多。

三元环的定义为无序三元组 (a, b, c) ，使得 a 到 b 有一条有向边， b 到 c 有一条有向边， c 到 a 有一条有向边。

你只需要保证三元环个数最多即可，如果有多种最优解，输出任意一种即可。
对于 100% 的数据， $1 \leq n \leq 1000$ 。

Solution: 三元环

对于任意三个点，要么这是一个三元环，要么存在一个点 u ，它的入度是 2，设点 i 的入度为 deg_i ，那么三元环的个数为

$$\binom{n}{3} - \sum_{i=1}^n \binom{deg_i}{2}$$

所以我们只需要最小化 $\sum_{i=1}^n \binom{deg_i}{2}$ ，这等价于最小化 $\sum_{i=1}^n deg_i^2$ ，因为 $\sum_{i=1}^n deg_i$ 固定，所以我们需要让所有 deg_i 尽量平均，因为若 $deg_i > deg_j + 1$ ，让 deg_i 减一， deg_j 加一，平方和变小。

当 n 为奇数时，最优方案显然是每个点的 deg 都为 $\frac{n-1}{2}$ ，这是可以做到的：将点从 0 编号到 $n-1$ ，第 i 号点向 $(i+1) \bmod n, (i+2) \bmod n, \dots, (i+\frac{n-1}{2}) \bmod n$ 连有向边即可。

当 n 为偶数时，前 $n-1$ 个点按照上述方案构造，最后一个点向前面任意 $\frac{n}{2}$ 个点连有向边，剩下的 $\frac{n}{2}-1$ 个点向最后一个点连边即可。

时间复杂度 $\mathcal{O}(n^2)$ 。

树

给定一个 n 个点的树 G ，称它的一个子图 G' 为“子树”当且仅当 $G = G'$ ，或者存在 G 上的一条边，将 G 断掉这条边之后 G' 是分成的两个连通块之一。

对于所有的 $k = 1, 2, 3 \dots n$ ，求从这棵树中等概率选出 k 个不同的点，包含它们的最小的“子树”的大小的期望模 998244353 的值。

对于 100% 的数据，保证 $3 \leq n \leq 7000$ 。

Solution: 树

用和除以方案数来算期望，问题化为对每个 k ，求出每种选法所需大小的和。

找出树的重心，设为 $root$ ，定它为树根，此时若选出的 k 个点的 lca 不为根，那么最优的“子树”一定是断开 lca 和其父亲之间的边，这部分是容易计算的，只需要枚举 lca ，设当前枚举的 lca 为 u ，其儿子结点构成的集合为 $son(u)$ ，以 u 为根的子树大小为 $size(u)$ ，那么当选出 k 个点且它们的 lca 为 u 时，所需大小的和为

$$\binom{size(u)}{k} - \sum_{v \in son(u)} \binom{size(v)}{k}$$

容易发现对于不同的 k ，其上指标对应的系数是相同的，因此可以先枚举 lca ，求出每个上指标的系数和，最后一起计算，总时间复杂度 $\mathcal{O}(n^2)$ 。

现在看 lca 为根结点的情况，最优的“子树”应该是断掉一个不包含所选结点的最大子树，当选 k 个结点的时候，所有选法所需大小的和为

$$\sum_{i=1}^n \text{所需“子树”大小} \geq i \text{ 的方案数}$$

只需对每个 i 求出和与内容即可，所需“子树”大小 $\geq i$ 等价于每一个大小大于 $n-i$ 的子树，其内部至少有一个结点被选择，设子树大小大于 $n-i$ 且其子树内没有其他结点的子树大小大于 $n-i$ 的结点构成集合 S ，显然 S 内结点的子树互不相交，问题等价于要从 n 个点里选出来若干个，满足 S 中元素里有至少一个点，同时这些点的 lca 是根结点。

我们先忽略 lca 是根结点的限制，这个问题的答案可以容斥，算出来是

$$\sum_{T \subseteq S} (-1)^{|S|-|T|} \binom{b + \sum_{v \in T} \text{size}(v)}{k}$$

其中 b 是不在 S 内任一子树中的结点个数。

这个可以 dp，设 $dp_{i,j}$ 为考虑了 S 中的前 i 个点，二项式系数的上指标为 j 的那一项的系数，容易 $\mathcal{O}(1)$ 转移，暴力做，时间复杂度是 $\mathcal{O}(n^3)$ 。

但是我们还应该保证 lca 是根结点，可以分类讨论：设根结点的儿子中，子树大小最大的是 $Maxsize$ ，次大的是 $maxsize$ 。

只要 lca 是根结点，那么所需的“子树”大小必然不会小于 $n - Maxsize$ ，因为我们最多把这个最大的子树割掉，这部分就可以套用之前 lca 不为根结点时的做法了。

当要求所需“子树”大小大于等于一个在区间 $[n - Maxsize + 1, n - maxsize]$ 的数 x 时， S 中的点都在根的同个子树中，具体来说是大小为 $Maxsize$ 的那个子树，这时候我们还需要额外保证其他的子树以及根结点中至少选一个，将这些点当成一个“子树”放到 S 中即可。

其余情况， S 中的点必然不会在根的同个子树，因此 lca 必然是根，我们容斥出来的就是正确结果。

再来说怎么把 $\mathcal{O}(n^3)$ 优化为 $\mathcal{O}(n^2)$ ，观察我们更改 i 的时候， S 发生了什么变化，实际上就是一些点被加进去了，一些点被删除出来了，而这个 dp 的转移是可以撤销的（将 dp_i 看作多项式，往 S 中新加入一个值相当于乘 $x^a - 1$ ，其中 a 为子树大小，显然可以再除回去），而每个点最多进 S 一次，出 S 一次，所以我们增量维护，时间复杂度 $\mathcal{O}(n^2)$ 。

陈太阳与粉丝

给你一棵 n 个点的树，根结点为 1，你可以从根开始随意得到一个 dfs 序，最终回到根结点，可以证明一定经过了 $2n - 2$ 条边。对于一个 dfs 序，它的值为 $\sum_{u=1}^n c_u$ ，其中 c_u 表示在该 dfs 序中第一次到达点 u 后经过了多少条边（返回到根结点的边也算，也就是说，一条树边会被经过两次）。

询问对于所有可能的 dfs 序，所能得到的最大值。

对于 100% 的数据，保证 $1 \leq n \leq 10^7$ 。

Solution: 陈太阳与粉丝

对树的大小归纳，容易证明最终答案和具体怎样 dfs 无关，所以随便 dfs 算出来一种答案就行，时间复杂度 $\mathcal{O}(n)$ 。

CF526 F. Pudding Monsters

给定一个 $n \times n$ 的棋盘，上面每行只有一个棋子，每列只有一个棋子，共有 n 个。

现求有多少个 $k \times k$ 的子棋盘满足上面的每行、每列也只有一个棋子。

对于 100% 的数据，保证 $n \leq 3 \times 10^5$ 。

Solution: CF526 F. Pudding Monsters

先把二维转为一维：若位置 (x, y) 有棋盘，则 $A_x = y$ ，显然 A 是一个 1 到 n 的排列，问题等价于有多少区间 $[l, r]$ 满足 $\max_{i=l}^r A_i - \min_{i=l}^r A_i = r - l$ 。

这是一个经典问题，用两个单调栈从左到右扫过去维护所有以当前点为右端点的后缀的最大、最小值，每次查询以当前点为右端点的所有后缀对答案的贡献，由于 $(\max v - \min v) - (r - l) \geq 0$ 恒成立，并且每次统计贡献时总有一个后缀区间取到等号 $([r, r])$ ，所以我们只需要维护这个东西的最小值个数即可，在维护单调栈的同时维护这个东西，是 $\mathcal{O}(n)$ 次区间加的形式，线段树维护即可。时间复杂度 $\mathcal{O}(n \log n)$ 。

CF526G. Spiders Evil Plan

给定一棵 n 个节点的无根树，每条边有边权。

有 q 次询问，每次询问给出 x, y ，你需要选择 y 条树上的路径，使这些路径形成一个包含 x 的连通块，且连通块中包含的边权和最大，输出这个最大的边权和。

对于 100% 的数据，保证 $n, q \leq 10^5$ ，强制在线。

Solution: CF526G. Spiders Evil Plan

引理： k 条路径可以完全覆盖一棵叶子结点个数不大于 $2k$ 的树。

证明平凡，显然每条路径可以连接两个没有被连接过的叶子结点，因此若 $2y \geq c$ ，其中 c 表示树中度数为 1 的点的个数，那么所有边都是可以被选上的，我们现在考虑 $2y < c$ 的情况。

对于询问 x, y ，以 x 为根将树长链剖分，我们需要连接哪 $2y$ 个叶子结点呢？将每条长链的边权和挂到其对应的叶子结点上，我们选择的就权值最大的那 $2y$ 个叶子，所得到的答案就是这些叶子的权值和。

问题在于我们不能对每个点为根做长链剖分，解决方案是找到树的一条直径，以 x 为根的最长的一条链显然是连接到某个直径的某一端的，因此对这条直径的两个端点分别作为根做树链剖分， x 的答案在这两棵树的答案中取最大值即可。注意直径的两端必然是叶子结点，因此只能再选 $2y - 1$ 个非根叶子。

我们不能保证这些长链包含了 x ，有两种调整的方案：第一种是去掉权值最小的那个叶子，加上包含点 x 的长链，第二种是找到 x 的祖先中，所在长链被选中了的深度最大的点，将它向下选择的那个叶子改为 x 所在轻链的叶子，这两种调整都是可以用树上倍增 $\mathcal{O}(n \log n)$ 做到的。总时间复杂度 $\mathcal{O}((n + q) \log n)$ 。

CF527E. Data Center Drama

给定一张 n 个点 m 条边的连通无向图，你需要加尽可能少的边，然后给所有边定向，使得每一个点的出入度都是偶数。边可以是自环，也可以有重边。输出最终的图，多种方案则任意输出一种。

对于 100% 的数据，保证 $n \leq 10^5$ ， $m \leq 2 \times 10^5$ 。

Solution: CF527E. Data Center Drama

首先，每个点的度数一定是偶数，这是该图存在欧拉回路的充要条件，考虑将图中度数为奇数的点抽出来，一定有偶数个这样的点（总度数是偶数，因为每条边会贡献两个度数），两两配对让每个点的度数都是偶数。

这样还不行，由于自环的存在，我们还应该保证总边数是偶数，只需要随便找个点加自环即可。

之后跑欧拉回路，隔一条边换一个方向即可得到构造方案，并且总边数最小。时间复杂度 $\mathcal{O}(n + m)$ 。

CF1868C. Travel Plan

给出一个 n 个点的完全二叉树，你可以给每个点赋一个 $[1, m]$ 之间的整数权值。定义 $s(u, v)$ 表示 u, v 之间路径上的点（包含两个端点）的权值最大值，求每种方案 $\sum_{i=1}^n \sum_{j=i}^n s(i, j)$ 的值的值模 998244353 的结果。

对于 100% 的数据，满足 $1 \leq n \leq 10^{18}$ ， $1 \leq m \leq 10^5$ 。

Solution: CF1868C. Travel Plan

对每条路径分别考虑贡献，注意到长度一样的路径对答案的贡献实际上是一样的，而完全二叉树的深度是 $\mathcal{O}(\log n)$ ，因此我们可以枚举路径长度 d ，答案即为

$$\sum_{d \geq 1} c_d \times \sum_{k=1}^m \text{长度为 } d \text{ 的路径答案为 } k \text{ 的方案数} \times k$$

根据套路，后面对 k 求的和等价于

$$\sum_{k=1}^m \text{长度为 } d \text{ 的路径答案大于等于 } k \text{ 的方案数}$$

这又可以化为

$$\sum_{k=1}^m (m^d - (k-1)^d) \times m^{(n-d)}$$

现在的问题在于怎么快速地算出每个长度的路径数，由于完全二叉树不同的子树只有 $\mathcal{O}(\log n)$ 个，所以完全可以考虑直接做基础的树形 dp，设 f_i 表示大小为 i 的完全二叉树的不同长度路径的个数， g_i 表示大小为 i 的完全二叉树的不同深度的点的个数，转移平凡，总时间复杂度 $\mathcal{O}(\log n)$ ，但实际上下标太大了，我们需要用 `std::map`，所以时间复杂度为 $\mathcal{O}(\log^2 n)$ 。

求出了 c_d ，上面的答案就可以 $\mathcal{O}(m \log n)$ 求出了，总时间复杂度 $\mathcal{O}(m \log n + \log^2 n)$ 。

【NOI2023】方格染色

有一个 n 列 m 行的棋盘，共 $n \times m$ 个方格，我们约定行、列均从 1 开始标号，且第 i 列、第 j 行的方格坐标为 (i, j) 。初始时，所有方格的颜色均为白色。现在，你要对这个棋盘进行 q 次染色操作。

染色操作分为三种，分别为：

1. 将一条横线染为黑色。具体地说，给定两个方格 (x_1, y_1) 和 (x_2, y_2) ，保证 $x_1 \leq x_2$ ， $y_1 = y_2$ ，将这两个方格之间的所有方格（包括这两个方格）染为黑色。
2. 将一条竖线染为黑色。具体地说，给定两个方格 (x_1, y_1) 和 (x_2, y_2) ，保证 $x_1 = x_2$ ， $y_1 \leq y_2$ ，将这两个方格之间的所有方格（包括这两个方格）染为黑色。
3. 将一条斜线染为黑色。具体地说，给定两个方格 (x_1, y_1) 和 (x_2, y_2) ，保证 $x_1 \leq x_2$ ， $x_2 - x_1 = y_2 - y_1$ ，将这两个方格之间斜线上所有形如 $(x_1 + i, y_1 + i)$ ($0 \leq i \leq x_2 - x_1$) 的方格染为黑色。**这种染色操作发生的次数不超过 5 次。**

现在你想知道，在经过 q 次染色后，棋盘上有多少个黑色的方格。

对于所有测试数据保证： $1 \leq n, m \leq 10^9$ ， $1 \leq q \leq 10^5$ ， $1 \leq x_1, x_2 \leq n$ ， $1 \leq y_1, y_2 \leq m$ ，且最多有 5 次第三种染色操作。

Solution: 【NOI2023】方格染色

先考虑 $n \leq 10^5$ 的情况，对于每一行的横线，我们将其改为不相交的形式，每一列的竖线以及那 $\mathcal{O}(1)$ 个斜线也是一样的，这部分的时间复杂度是 $\mathcal{O}(q)$ 。

处理完后我们先算横线和竖线，答案就是所有的和再减去相交的点数，相交的点只可能是一条横线和一条竖线相交，二维数点的形式可以 $\mathcal{O}(q \log q)$ 做。

接着是斜线，处理完后每一条斜线是不会相交的，因此可以分开考虑：对于每一条斜线，先让答案加上它的点数，再减去重合的部分，怎么得出和横（竖）线重合的点数呢？ $\mathcal{O}(q)$ 扫一遍所有的横线、竖线看一看有没有相交即可，因为斜线是 $\mathcal{O}(1)$ 的，因此这部分是 $\mathcal{O}(q)$ 的。

对于 $1 \leq n, m \leq 10^9$ ，做一个离散化即可，总时间复杂度 $\mathcal{O}(q \log q)$ 。

【NOI2023】桂花树

小 B 八年前看到的桂花树是一棵 n 个节点的树 T ，保证 T 的非根结点的父亲的编号小于自己。给定整数 k ，称一棵 $(n+m)$ 个节点的有根树 T' 是繁荣的，当且仅当以下所有条件满足：

1. 对于任意满足 $1 \leq i, j \leq n$ 的 (i, j) ，在树 T 和树 T' 上，节点 i 和 j 的最近公共祖先编号相同。
2. 对于任意满足 $1 \leq i, j \leq n+m$ 的 (i, j) ，在树 T' 上，节点 i 和 j 的最近公共祖先编号不超过 $\max(i, j) + k$ 。

注意题目中所有树的节点均从 1 开始编号，且根结点编号为 1。 T' 不需要满足非根结点的父亲编号小于自己。

小 B 想知道有多少棵 $(n+m)$ 个节点的树是繁荣的，认为两棵树不同当且仅当存在某一个节点在两棵树上的父亲不同。你只输出方案数在模 $(10^9 + 7)$ 意义下的值，每个测试点有 t 组数据。

对于所有测试数据保证： $1 \leq t \leq 15$ ， $1 \leq n \leq 3 \times 10^4$ ， $0 \leq m \leq 3000$ ， $0 \leq k \leq 10$ ， $1 \leq f_i \leq i - 1$ 。

Solution: 【NOI2023】桂花树

第一个限制等价于 $1 \sim n$ 构成的虚树是原树，这是容易理解的。对于第二个限制，那个 \max 很扎眼，我们把他去掉：条件二等价于对于任意 $1 \leq j < i \leq n+m$ 的 (i, j) 在树 T' 上的 lca 编号不超过 $i + k$ ，这等价于对每个 $0 \leq t \leq m$ 让 $1 \sim n+t$ 构成的虚树中结点编号最大为 $n+t+k$ 。

先考虑 $k=0$ 的情况，从 $n+1$ 号点开始不断往原树中加点，不难发现每次加点只能加到某个点的儿子上或是插入到某条边的中间，于是答案就是

$$\prod_{i=n}^{n+m-1} (2i-1)$$

再考虑 $k>0$ 的情况，这时候除了上面的两种情况，还可能是：在某条边上插入了一个编号在 $[i+1, i+k]$ 的点， i 号点其实是被加在这个点的儿子中的，但我们好像没办法很好地处理这个问题。

其实是可以处理的，注意到 $k \leq 10$ ，因此完全可以承受一个 $\mathcal{O}(tm2^k)$ 的复杂度，我们可以设计一个状压 dp，记录当前编号到当前编号加 $k-1$ 这些编号中哪些已经被之前的点预定了，转移是平凡的。

总时间复杂度 $\mathcal{O}(tm2^k)$ 。

CF1868D. Flower-like Pseudotree

给你一棵基环树，定义一个基环树是像花的当且仅当删去环上的边后（不删去点）形成的森林中的每棵树高度相同，现给出点数 n 和每个点的度数 d_i ，要求构造出一棵合法的像花的基环树，或输出无解。

对于 100% 的数据，保证 $1 \leq n \leq 10^6$ 。

Solution: CF1868D. Flower-like Pseudotree

毒瘤分类讨论题，首先应该满足 $\sum_{i=1}^n d_i = 2n$ ，否则一定无解，之后先把 $d_i = 2$ 的情况特判掉，此时整棵基环树就是一个环，再把 $n=1$ 的情况判掉，此时一定不合法。

将所有的 d_i 降序排列，按照 d_1 分类讨论。

当 $d_1 = 1$ 的时候， $\sum_{i=1}^n d_i = 2n$ 一定不成立，不需要考虑这种情况。

当 $d_1 = 2$ 的时候，由于 $\sum_{i=1}^n d_i = 2n$ 已经成立，因此一定有 $d_i = 2$ ，这已经被我们判掉了。

当 $d_1 = 3$ 的时候，如果 $d_2 < 3$ ，一定不合法，这是因为无论是 d_1 在环上还是不在环上，一定不成立，否则我们将度数最大的两个点作为基环树的环。设度数 ≥ 2 的点有 c 个，若 c 是偶数，那么我们完全可以将这些点分

成两条链挂在环上的两个点下面，再将那些度数为 1 的点随便分配到度数还没有满的点的叶子上，这样一定成立。若 c 是奇数，当 $d_3 < 3$ 的时候，无论如何也是构造不出来的。当 $d_3 \geq 3$ 且 $c \geq 7$ 的时候，我们可以将 d_3 对应的点挂在 d_1 对应的点下方，将 4 挂在 2 下面，然后将 5, 6 挂在 3 下面，将 7 挂在 4 下面，剩下的度数不为 1 的点可以作为两条等长的链挂在 6, 7 下面。当 $d_3 \geq 3$ 的其他情况，我们只能将度数不为 1 的点全部连成一个环，若有度数为 2 的点则无解。

当 $d_1 > 3$ 的时候， c 是偶数还是一样， c 为奇数时只需要 $c \geq 5$ 就一定可以像之前那样构造了，否则若 $c = 3$ ，只需要将这三个点连成环，若某个点的度数为 2 则无解。

总时间复杂度 $\mathcal{O}(n \log n)$ ，瓶颈在于最开始的排序。

CF1868E. Min-Sum-Max

给你一个长度为 n 的数组 A ，标号从 1 开始，你需要将它分为若干段，满足每段中的下标连续并且每个下标在且仅在一段中，同时，设段数为 m ，第 i 段的下标是 $[l_i, r_i]$ ，那么你还满足对于任意 $1 \leq i \leq j \leq m$ 有

$$\min_{k=i}^j s_k \leq \sum_{k=i}^j s_k \leq \max_{k=i}^j s_k$$

其中 s_k 表示第 k 段的元素和，即 $\sum_{i=l_k}^{r_k} A_i$ 。

求最多可以将 A 分为几段。

对于 100% 的数据，保证 $-10^9 \leq A_i \leq 10^9$ ， $1 \leq n \leq 300$ 。

Solution: CF1868E. Min-Sum-Max

设 $r_0 = 0$ ，做一个前缀和，不去管题面中 s 的定义，设 $s_i = \sum_{j=1}^i A_j$ ，那么限制等价于

$$\min_{k=i}^j (s_{r_k} - s_{r_{k-1}}) \leq s_{r_j} - s_{r_i} \leq \max_{k=i}^j (s_{r_k} - s_{r_{k-1}})$$

我们关注最大的 s ，设为 s_{r_x} ，同理，最小的 s 设为 s_{r_y} ，若有多个最大（小）值，取 $|x - y|$ 最小的一对。不妨设 $x > y$ ，那么若 $x > y + 1$ ，限制一定不成立，这是因为对于 $i = y, j = x$ 的限制， $s_{r_j} - s_{r_i}$ 一定不会小于 $\max_{k=i}^j (s_{r_k} - s_{r_{k-1}})$ ，若等于，那么一定是存在某个 $t \in (y, x)$ 满足 $s_{r_t} = s_{r_x}$ 或 $s_{r_t} = s_{r_y}$ ，这与我们先前的假设（取 $|x - y|$ 最小的一对）矛盾。

于是我们得到了**性质 1**：对于一组合法的构造，一定存在某个 x 满足 s_{r_x} 和 $s_{r_{x+1}}$ 为 s_{r_i} 的最大值和最小值（或最小值和最大值）。

有了性质 1，一个自然的想法是将整个序列再根据 x 分为两侧递归考虑，这需要另一个性质的支撑：即若某一侧的限制满足，整个序列的限制同样会被满足。

这里只证右侧，左侧同理。假设 $\forall i \geq x + 1, i \leq j \leq m$ 的 i, j 限制均被满足，现证当 $1 \leq i \leq x$ 时限制仍满足。

固定 j 考虑不同的 i ，首先 $\text{sum} \leq \max$ 的限制是一定被满足的，因为 \max 就是最大值减最小值， $s_{r_i} - s_{r_j}$ 不会更大，于是只需要考虑 $\min \leq \text{sum}$ 的限制，不妨令 s_{r_x} 为最小值， $s_{r_{x+1}}$ 为最大值，由于

$$s_{r_j} - s_{r_i} \geq s_{r_j} - s_{r_{x+1}} \geq \min_{k=x+1}^j (s_{r_k} - s_{r_{k-1}}) \geq \min_{k=i}^j (s_{r_k} - s_{r_{k-1}})$$

命题得证！对于 s_{r_x} 为最大值的情况、左侧的情况，同理可证。

于是就可以做一个简单 dp 了，设 $dp_{i,j,l,r}$ 为考虑 $[i, j]$ 这个子区间， s_{r_i} 必须在区间 $[l, r]$ ，最多能分为几段，转移平凡，时间复杂度 $\mathcal{O}(n^6)$ ，若转移时用二维前缀 max 优化，时空复杂度均为 $\mathcal{O}(n^4)$ 。

现在考虑优化空间，假如我们把状态转移方程写出来，会发现我们用到状态必然满足 $l = S_i$ 或 $l = S_j$ 或 $r = S_i$ 或 $r = S_j$ ，于是用二维记录 l, r 是浪费的，可以将状态换为 $dp_{i,j,k,0/1/2/3}$ ，第四维表示该状态满足上述哪个条件，第三维表示另一个边界的值，空间复杂度降为 $\mathcal{O}(n^3)$ 。

接着考虑优化状态的转移，以 $dp_{i,j,k,0}$ 为例，我们已经知道了选择的 S 的左边界为 S_i ，我们必须在 $[i + 1, j]$ 中再选一个 S_x ，使得 $S_x - S_i$ 是一个在问题的原始表述中被选择的一段，又因为状态限制了所选 S 不能小于 S_i ，

因此 S_i 实际上就是上面朴素 dp 中我们枚举的极值中的一个，我们只需要枚举另一个极值，设为 S_y ，找到该位置前面第一个等于 S_i 的和后面第一个等于 S_i 的，将它们作为最小值， S_y 作为最大值转移两次即可，总时间复杂度 $\mathcal{O}(n^3)$ 。

CF1870E. Another MEX Problem

给你一个长度为 n 的数组 A ，你可以将它分为互不相交的若干段（可以存在某些元素不属于任何一段），设第 i 段的 mex 为 c_i （ $mex(S)$ 表示最小的没有在 S 中出现过的自然数），该划分的权值就是 $\text{xor } c_i$ ，其中 xor 表示按位异或运算。

求出 A 数组可能的最大权值。

对于 100% 的数据，保证 $1 \leq n \leq 5000$ 。

Solution: CF1870E. Another MEX Problem

$\mathcal{O}(n^2)$ 预处理出每个子区间的 mex ，很容易得到一个朴素的 $\mathcal{O}(n^3)$ dp，方法是设 $dp_{i,x}$ 表示前 i 个元素是否可以凑出 x 的权值，转移枚举一个右端点在该点的区间即可。

怎么进一步优化呢？注意到若两个区间 $[l_1, r_1], [l_2, r_2]$ 满足 $[l_2, r_2] \subset [l_1, r_1]$ 并且两区间的 mex 相等，那么 $[l_1, r_1]$ 这个区间就是没用的了，这是因为选上这个区间一定不如选更小的区间。

假如只保留这些有用的区间再去转移呢？换句话说，有用的区间有多少个？接下来我们证明有用的区间只有 $\mathcal{O}(n)$ 个，之后只用有用的区间进行转移，时间复杂度 $\mathcal{O}(n^2)$ 。

为了证明有用的区间只有 $\mathcal{O}(n)$ 个，我们转而证明：若以 l 为区间左端点，那么满足 $[l, r]$ 为有用区间且 $A_l \geq A_r$ 的区间至多只有一个。

反证法，设有两个，它们的右端点为 $r_1 < r_2$ ，由于 $[l, r_1]$ 有用，所以这段区间的 mex 不小于 $A_l + 1$ （否则我们可以删去 A_l ， mex 不变），又因为 $A_{r_2} \leq A_l$ ，因此 r_2 这个点是完全没用的，因此 $[l, r_2]$ 不可能成为有用的区间（我们总可以删掉 r_2 得到更好的区间）。

对于 $A_r \geq A_l$ 的情况同理，所有的情况都被这两种子情况包含，每种子情况只有至多 n 个有用的区间，因此有用的区间至多 $2n$ 个，证毕！

CF1870F. Lazy Numbers

给定正整数 n 和 B ，将 $[1, n]$ 内的正整数全部用 B 进制字符串表示出来并按照字典序升序排列，求有多少个数 i 满足它在排序后的第 i 个位置。

对于 100% 的数据，保证 $1 \leq n, B \leq 10^{18}$

Solution: CF1870F. Lazy Numbers

将 $[1, n]$ 表示的字符串插入到 trie 里，若在搜索时先走字典序较小的边，设 $dfs(x)$ 为从根开始 dfs，走到结点 x 时经过的边数， $bfs(s)$ 表示从根开始 bfs，走到结点 x 时经过的边数，要求等价于 $dfs(x) = bfs(x)$ 。

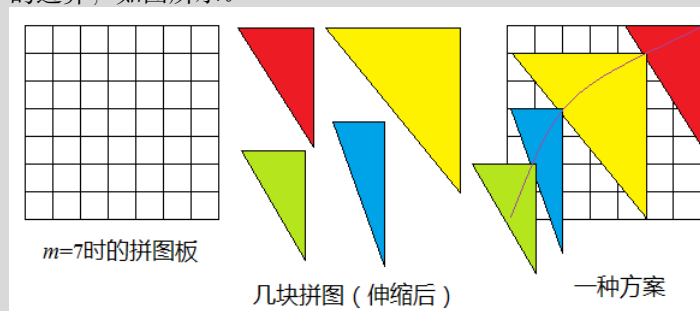
我们每次关注 trie 上的一层结点，它们的 $dfs(x) - bfs(x)$ 是单调不增的，即 $dfs(x) - bfs(x) \geq dfs(x+1) - bfs(x+1)$ ，这里的 $+1$ 是指本层的下一个结点。这是容易理解的，因为 $bfs(x+1) = bfs(x) + 1$ ，但 $dfs(x+1) \geq dfs(x) + 1$ （dfs 会走完一个子树），于是可以在每一层两次二分求出满足 $dfs(x) - bfs(x) = 0$ 的区间，加到答案里。

最后的问题是怎么快速求一个点的 $dfs(x) - bfs(x)$ ，显然 $bfs(x) = x$ ，而 $dfs(x)$ 可以 $\mathcal{O}(\log n)$ 地求（trie 高度为 $\mathcal{O}(\log n)$ ）。

trie 有 $\mathcal{O}(\log n)$ 层，每层的时间复杂度为 $\mathcal{O}(\log^2 n)$ ，总时间复杂度 $\mathcal{O}(\log^3 n)$ 。

Loj500. 「LibreOJ β Round」ZQC 的拼图

ZQC 和他的妹子在玩拼图。她们有 n ($1 \leq n \leq 100$) 块神奇的拼图，还有一块拼图板。拼图板是一个 $m \times m$ ($1 \leq m \leq 100$) 的正方形网格，每格边长为 1，如图所示。每块拼图都是直角三角形，正面为白色，反面为黑色，第 i 块拼图初始时的水平直角边长为 $\frac{1}{a_i}$ ，垂直直角边长为 $\frac{1}{b_i}$ 。拼图放在拼图板上时，必须正面朝上，直角顶点必须与拼图板上的一个格点重合，两条直角边分别向左和向下。拼图可以重叠在一起。拼图的左下部分可以超过拼图板的边界，如图所示。



这些拼图有一个好，就是能伸缩，当然，拼图伸缩是要按基本法来的，具体说来就是：你可以选择一个正整数 k ，并使所有拼图的每条边长都变成原来的 k 倍。

妹子摆好拼图后，ZQC 需要控制一个小人从拼图板的左下角跑到右上角，小人路线上的任何一点（包括端点）都要在某块拼图板上（边界或顶点也可以），现在 ZQC 想知道他的妹子最少要把拼图的边长扩大到原来的几倍才存在一种摆放方式使得他能找到这样一条路线。

Solution: Loj500. 「LibreOJ β Round」ZQC 的拼图

显然可以二分答案，转化为判定放大 mid 倍是否可行，这可以 dp 做，设 $f_{i,j}$ 表示只考虑前 i 个直角三角形，到达了第 i 列时，最多到第几行。于是放大 mid 倍可行当且仅当 $f_{n,m} \geq m$ 。

但是这样有一个问题：我们固定了放直角三角形的顺序，好在这并不会影响答案的正确性：将直角三角形看作一堆向量，向量的顺序是无关紧要的。

容易做到 $\mathcal{O}(nm^2)$ 算出 $f_{n,m}$ ，总时间复杂度 $\mathcal{O}(nm^2 \log w)$ 。

Loj501. 「LibreOJ β Round」ZQC 的数列

ZQC 和妹子来到了一片小树林，看到一列参差不齐的树，ZQC 想到了一种评价每一列树的美观程度的方法——将相邻两棵树高度差的总和作为一列树的美观度。即，设一列树的高度组成的序列为 A ，则其美观度为

$$f(A) = \sum_{i=1}^{|A|-1} |A_i - A_{i+1}|$$

ZQC 想，每一列树都有一些特征，于是他钦定了一列树的特征值——设 S 为 A 的美观度最大的子序列，若 A 有 k 个子序列 T_i 的美观度与 S 相同（即 $f(T_i) = f(S)$ ），则称 A 的特征值为 k 。注意，子序列不能为空。

ZQC 的妹子非常喜欢 k 这个特征值，她希望 ZQC 能给她种一列特征值为 k 的树。按照套路，这么简单的问题 ZQC 当然不会亲自出马，所以他钦定你来解决这个问题。

给出一个序列 A 的特征值 k ($1 \leq k \leq 10^{18}$)，要求构造一个这样的序列 A ($1 \leq |A| \leq 5000$, $0 \leq A_i \leq 2^{31} - 1$)。

Solution: Loj501. 「LibreOJ β Round」ZQC 的数列

将相同的数字缩到一起，容易发现：对于一段极长的单调不减（增）序列，它们的第一个元素和最后一个元素是必选的，中间的元素可以任意，于是，将 A 序列划分为若干个单调区间，每一个区间对特征值的贡献为

$$(2^a - 1) \times 2^b \times (2^c - 1)$$

总特征值就是每一个区间的贡献乘起来，于是我们得到：最终的特征值一定形如 $2^k \prod (2^{c_i} - 1)$ 。

将 k 的 2 的因子先提出来，将他们放到最终构造的序列的某个单调区间中间即可，例如对于 $k = 12$ 我们可以构造 1 1 2 2 3，即 $(2^2 - 1) \times 2^2 \times (2^1 - 1)$ ，对于 $k = 84$ ，我们可以类似地构造 1 1 2 2 3 1 1 1。

问题转化为判定一个数是否可以被写成 $\prod (2^{c_i} - 1)$ 的形式，可以直接记忆化搜索搜出来正确的分解，然后直接构造即可，时间复杂度 $\mathcal{O}(\tau(k) \log k)$ 。

一个错误的想法是从大到小考虑 $2^c - 1$ ，能除就除： $315 = 63 * 5 = 15 * 7 * 3$ 可以成功 hack。

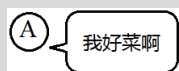
Loj502. 「LibreOJ β Round」ZQC 的截图

一个阳光明媚的午后，ZQC 在水 QQ 群，突然他发现一位神 [?] 发了一条装弱消息，机智的 ZQC 立即截图保存。接着，不出 ZQC 所料，群里发生了著名的“无穷递降装弱效应”。“无穷递降装弱效应”是指一位神 [?] 装弱后，只要有一个人截图，就会发生大量的人嵌套截图的现象……

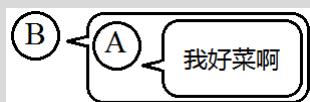
ZQC 对屏幕上不断滚动的嵌套截图消息十分感兴趣，他想让你帮他算一个东西。在这之前，他认为有必要再强调一些关于截图消息的概念：

一条截图消息 M 可以由一个二元组 (u, M') 来表示，其中 u 是这条消息的发送者， M' 是另一条截图消息或 NIL。 $M' = \text{NIL}$ 表示这条消息记录是最开始的装弱消息。

例如，这条消息（最开始的装弱消息）可以表示为 (A, NIL) ：



而这条消息可以表示为 $(B, (A, \text{NIL}))$ ：



如果设第一条消息为 $S = (A, \text{NIL})$ ，第二条消息也可以表示为 (B, S) 。

消息中一个人的出现次数定义如下：设 $f(M, v)$ 表示消息 $M = (u, M')$ 中 v 的出现次数，则：

$$f(M, v) = \begin{cases} [u = v] & M' = \text{NIL} \\ f(M', v) + [u = v] & M' \neq \text{NIL} \end{cases}$$

其中 $[u = v]$ 这个表达式当 $u = v$ 时值为 1，否则为 0。

ZQC 终于想好要考你什么了。对于每条消息，你需要帮他算出这些：

- 是否这条消息中所有人的出现次数都是 3 的倍数。
- 如果 1 的答案是“否”，这条消息中是否只有一个人的出现次数不是 3 的倍数。
- 如果 2 的答案是“是”，那个人是谁。

因为屏幕上的消息是一条一条发出来的，所以你也需要一条一条依序回答，即强制在线。

设有 n 个人， m 条消息，则对于 100% 的数据，保证 $3 \leq n \leq 10^6$ ， $1 \leq m \leq 2 \times 10^6$ 。

Solution: Loj502. 「LibreOJ β Round」ZQC 的截图

逆天随机化, 问题可以变为: 有一棵树, 最开始为空, 每次插入一个叶子, 询问该叶子到跟的路径中, 是否所有编号的出现次数都为 3 的倍数等等. 假如可以离线, 就可以先把图建出来, $\mathcal{O}(n+m)$ 随便做了, 可惜本题强制在线.

怎么办呢? 如果非要用确定性算法, 只能是可持久化线段树等等, 但是空间完全不够用.

这时候就要用随机化了, 如果我们给每个人随机一个 $[0, 2]$ 之间的权值, 若到根的路径和为 0, 说明所有人的出现次数都是 3 的倍数, 否则若路径和等于某个人的权值或某个人权值的 2 倍, 说明只有这个人的出现次数不是 3 的倍数, 否则有多个人的出现次数不是 3 的倍数.

可是这个随机化太不靠谱了, 错误概率几乎是 100%, 解决方法是给每个人随一堆权值 (就是随一个多维向量), 然后如果对于向量的每一维都符合上面的判断, 那么我们就认为该判断正确. 向量应该选几维的呢? w 维向量的错误率为 $1 - (1 - \frac{n}{3^w})^m$, 取一个 $w = 40$ 就差不多了.

但是问题不在这里: 我们当然可以很容易地判断和是否为 0 (存一下每个点到根的路径权值和, 父亲到根的路径权值和是容易 $\mathcal{O}(1)$ 得到的), 但还需要 $\mathcal{O}(n)$ 判断是否只有一个人的出现次数不是三.

怎么做呢? 将所有权值模一个大数, 我们只需要在模大数相同的那一堆人里面找就行, 选取的好可以做到近似 $\mathcal{O}(1)$, 总时间复杂度就是 $\mathcal{O}(n+m)$.

Loj503. 「LibreOJ β Round」ZQC 的课堂

叮铃铃……上课铃响了。

「啊, 又是无聊的 math」, 坐在教室里的 ZQC 这样想道. Mr.Sam 今天在课上讲了平面直角坐标系上的向量. 「这不是幼儿园姿势么」, ZQC 实在忍不住, 睡着了. Mr.Sam 把 ZQC 给叫醒, 并给了他这样一道题:

假设有一平面直角坐标系, ZQC 有一支画笔, 起点是 $(1, 1)$, 现在有 n 个向量, 第 i 个向量形如 (x_i, y_i) , 且满足每一个向量都满足 x_i, y_i 均为偶数. ZQC 按顺序根据这些向量改变自己的画笔的位置, 即位置依次改变成 $(1+x_1, 1+y_1), (1+x_1+x_2, 1+y_1+y_2) \dots$. 每次改变位置时, 画笔都沿两点之间的最短距离移动. 结束时, 画笔的运动轨迹一定由 n 条线段组成. Mr.Sam 要 ZQC 回答这些线段穿过 x 轴和 y 轴的总次数之和是多少, 由于起点为奇数, 每次走偶数, 因此一定不会落在坐标轴上.

但这样的问题对 ZQC 来说太简单了, 于是 Mr.Sam 设定了一个指针, 一开始指在第一个向量. 现在他做了 $q(q \leq 3 \times 10^5)$ 个操作, 操作有四种, 分别是:

1. B 表示把指针向后移动, 如果越界则视为无效. 即, 如果设指针移动前的位置是 i , 那么移动后的位置是 $\max(1, i-1)$.
2. F 表示把指针向前移动, 如果越界则视为无效. 即, 如果设指针移动前的位置是 i , 那么移动后的位置是 $\min(n, i+1)$.
3. C nx ny, 把当前指针所指的向量修改为 (nx, ny) , 这里同样满足 nx, ny 为偶数.
4. Q 假设 ZQC 从起点开始, 按第 1 个到第 $n(n \leq 10^5)$ 个的顺序沿向量走, 询问画出的 n 条线段穿过 x 轴和 y 轴次数的总和.

Solution: Loj503. 「LibreOJ β Round」ZQC 的课堂

首先, x 和 y 是独立的, 我们只需要关注一维, 问题化为有一个长度为 n 的数组 A , 每次询问

$$\sum_{i=1}^n [S_i \times S_{i-1} < 0]$$

其中 $S_i = 1 + \sum_{j=1}^i A_j$.

每次修改的时候, 指针左侧的 S 是不受影响的, 这启发我们把指针两边分开算, 指针左侧的答案是容易在每次操作后 $\mathcal{O}(1)$ 维护的, 关键在于指针右边, 我们希望求出上面的式子.

容斥，答案等于右侧点的个数减去不符合的，不符合当且进当 $\min(S_{i-1}, S_i) > 0$ 或 $\max(S_{i-1}, S_i) < 0$ ，这两部分也是独立的，同样可以分开计算，问题化为：有若干个数，要支持删除某个数（指针右移）、插入某个数（指针左移）、将所有数都加上某个值（修改操作），查询小于 0 或大于 0 的数的个数。

对于修改操作，我们可以全局打标记，剩下的内容可以平衡树维护，时间复杂度 $\mathcal{O}(q \log n)$ 。

Loj504. 「LibreOJ β Round」ZQC 的手办

众所周知，ZQC 是个很喜欢收纳手办的大佬，他平时在写题前会先扫视一下桌面上排开的小姐姐们以获取灵感。假设他有 $n(1 \leq n \leq 5 \times 10^5)$ 个手办，小手办们排成一排，每个手办按照入手批次从第 1 个到第 n 个被贴上了一个标号 $a_i(1 \leq a_i \leq 10^9)$ 。有两个熊孩子到 ZQC 家里玩，熊孩子 A 不断地改掉标签并不停地提问熊孩子 B。由于熊孩子 B 太笨，经常回答不上来，熊孩子 A 表示很生气，ZQC 为了世界和平（把熊孩子哄高兴好让它们帮忙把标签贴回去），大发慈悲地帮助熊孩子 B 回答一系列问题。假设一共 $m(1 \leq m \leq 5 \times 10^5)$ 次操作，两种操作分别为：

- 1 a b k 将数列 $[a, b]$ 这个区间中所有比 $k(1 \leq k \leq 10^9)$ 小的数改为 k ；
- 2 a b k x 查询 $[a, b]$ 的区间中比 $k(1 \leq k \leq 10^9)$ 小的最小的 $x(1 \leq x \leq 10^5)$ 个数。

ZQC 最后成功维护了世界正义，请在每次查询时输出熊孩子 A 所要的回答。

对于 100% 的数据，保证 $\sum x \leq 5 \times 10^6$ 。

Solution: Loj504. 「LibreOJ β Round」ZQC 的手办

对 A 建线段树，维护区间最小值和最小值所在的某个位置，支持区间 \max ，这是容易的。

对于一个询问，我们开一个小根堆，堆中元素为四元组 (l, r, v, p) ，其中 $[l, r]$ 表示一个区间， v 表示区间内最小值， p 表示最小值所在的某个位置，四元组按照 v 排序。每次询问将 $[a, b, v, p]$ 放入堆中（ v, p 是线段树求出来的），然后不断将区间分裂为 $[l, p-1]$ 和 $[p+1, r]$ ，线段树再次查询并塞进堆中，弹出 x 次堆顶就可以求出这 x 个数了。

总时间复杂度 $\mathcal{O}((n + \sum x) \log n)$ 。

Loj506. 「LibreOJ β Round」ZQC 的作业

ZQC 有一道作业题：给定 $p(1 \leq p < 2^{31}), q(0 \leq q < 2^{31}), n(2 \leq n < 2^{31})$ ，求 $x^p + q$ 在模 n 意义下有多少取值。

由于 ZQC 急着去找妹子，所以这道题由你来解决。

Solution: Loj506. 「LibreOJ β Round」ZQC 的作业

由于笔者水平有限，本题解完全来源于 Loj 官方题解。

这个 $+q$ 是没用的，假设 k 是定值，记 $f(m)$ 是模 m 意义下不同的 x^k 的数量。

首先，由中国剩余定理 f 是积性的，于是只需要计算 $m = p^n$ 的情况。（对 $m = \prod_i p_i^{r_i}$ ，中国剩余定理断言 $Z_m \cong \prod Z_{p_i^{r_i}}$ （环同构），它们的乘法群也同构，从而 Z_m 上的 k 次幂就是（精确到同构） $Z_{p_i^{r_i}}$ 上的 k 次幂之间的乘积。）下面假设 $m = p^n$ 。

设 b 和 p 互素，注意到 b^k 模 m 下是一个 k 次方幂当且仅当 b 是一个模 p^{n-k} 下的 k 次方幂。（对于「当」，设 $b = x^k + tp^{n-k}$ ， $\gcd(t, p) = 1$ ，则 $(px)^k = p^k x^k + tp^{n-k} p^k$ ，在模 m 下它就是 b ，「仅当」是类似的。）

于是

$$f(p^n) = \sum_{i=1}^{ik \leq n} \psi(p^{n-ik})$$

，其中 $\psi(x)$ 指的是模 x 下与 x 互素的 k 次方幂数量。这个和式来自于分不同的 c 对形如 $(xp^c)^k, \gcd(x, m) = 1$ 的 k 次幂的统计。

问题归结于计算 $\sum_i \psi(p^{n-ik})$, 对 $p \neq 2$, 熟知 $Z_{p^{n-ik}}$ 的乘法群是循环的, 于是它同构于 $Z_{\phi(p^{n-ik})}$, 计算 $Z_{p^{n-ik}}$ 中的 k 次幂也就是计算 $Z_{\phi(p^{n-ik})}$ 中 k 的倍数, 由此可以知道 $\psi(p^{n-ik}) = \frac{\phi(p^{n-ik})}{\gcd(\phi(p^{n-ik}), k)}$ 。 (考虑 $Z_{\phi(p^{n-ik})}$ 上的自同态 $f(x) = kx$, 它的象就是那些被 k 整除的数, 于是这些数的个数就是 $\frac{\phi(p^{n-ik})}{|\ker f|}$, 不难证明 $\ker f$ 就是那些阶整除 k 的元素的集合, 熟知这样的元素的个数是 $\gcd(\phi(p^{n-ik}), k)$ 。) 对 $p = 2$, 情况类似。

注意到 $\phi(p^{n-ik}) = p\phi(p^{n-(i+1)k})$, 从而数列 $a_i = \frac{\phi(p^{n-ik})}{\gcd(\phi(p^{n-ik}), k)}$ 对前几项是常数列, 对之后的项是等比数列, 它的和不难求出。

时间复杂度 $\mathcal{O}(\text{PrimeFactorization}(n))$ 。如果用朴素算法, 就是 $\mathcal{O}(\sqrt{n})$ 。

qoj5414. Stop, Yesterday Please No More

给定一张 n 行 m 列的网格, 在位于第 i_h 行第 j_h 列的格子上有一个洞, 其它每个格子都是空地并且都有一只袋鼠。

相似的, 袋鼠可以被键盘上的 U, D, L, R 键控制。所有袋鼠会同时根据按下的按键移动。具体来说, 对于一只位于第 i 行第 j 列的格子 (用 (i, j) 表示) 上的袋鼠:

- 1. 按键 U : 它会移动到 $(i-1, j)$ 。
- 2. 按键 D : 它会移动到 $(i+1, j)$ 。
- 3. 按键 L : 它会移动到 $(i, j-1)$ 。
- 4. 按键 R : 它会移动到 $(i, j+1)$ 。

如果一只袋鼠踩到了洞 (也就是说, $i = i_h$ 且 $j = j_h$) 或者移动到了网格外面, 它将从网格上移除。

问题在于, i_h 和 j_h 的值是未知的。您只知道一个仅由字符 U, D, L, R 组成的操作序列, 以及一个整数 k 表示应用这个操作序列之后, 网格上恰有 k 只袋鼠存留。

请计算有多少位置可能存在洞。也就是说, 计算满足以下条件的整数对 (i_h, j_h) 的数量:

- $1 \leq i_h \leq n, 1 \leq j_h \leq m$ 。
- 洞位于 (i_h, j_h) 。
- 应用给定的操作序列后, 网格上恰有 k 只袋鼠存留。

对于 100% 的数据, 保证 $1 \leq n, m \leq 10^3$, 操作序列的长度不超过 10^6 。

Solution: qoj5414. Stop, Yesterday Please No More

设 S 表示操作序列, 一个经典操作是将“袋鼠移动”变为“洞移动”, 洞会沿着操作序列的反方向移动, 将覆盖到的袋鼠删掉, 同时边界也会移动, 但是不难用 $\mathcal{O}(|S|)$ 的时间求出没有被删去的矩形的边界, 设 siz 表示最终没有被边界删掉的矩形大小。

先设洞在 $(0, 0)$, 将洞移动到的位置去重存起来, 对于每个位置检查它是否可以作为洞: 它可以作为洞当且仅当洞移动到的没有被删掉的位置有 $siz - k$ 个, 怎么快速算这个东西呢? 设当前检查的位置为 (x, y) , 要求 $(x + x_i, y + y_i)$ 中有多少个数在 (没有被边界删掉的矩形中), 其中 (x_i, y_i) 是我们记录的洞在 $(0, 0)$ 时根据操作序列可以走到的位置。

二维前缀和直接算, 若用 `std::set` 去重, 时间复杂度 $\mathcal{O}(nm + |S| \log |S|)$ 。

qoj5415. Ropeway

有一个长度为 $n + 2$ ，标号从 0 开始的数组 A 和一个 01 串，满足 $A_0 = A_{n+1} = 0$ ，对于剩下的 A_i ，满足 $1 \leq A_i \leq 10^9$ ，给定限制 k ，你要从 $A_0, A_1, A_2, \dots, A_{n+1}$ 中选出若干个数，满足：

- A_0, A_{n+1} 必选。
- 01 串中是 1 的位置必选，是 0 的位置可选。
- 任意两个被选择的下标 i, j ，满足 $|i - j| \leq k$ 。

你的任务是找出所有选择方案中，被选择的 A_i 之和最小的方案，输出此时的 A_i 之和。

特别的，还有 q 次修改操作，每次修改形如将 A_x 改为 y ，**修改之间相互独立，也就是说回答完问题后 A_x 会被改回去**，你需要在每次修改结束之后求出最小的 A_i 之和。

对于 100% 的数据，满足 $1 \leq n \leq 5 \times 10^5$ ， $1 \leq k, q \leq 3 \times 10^3$

Solution: qoj5415. Ropeway

首先看没有修改怎么做，这是一个简单 dp，设 f_i 表示考虑完前 i 个数的同时选择了 A_i 的答案，容易做到 $\mathcal{O}(n^2)$ ，显然可以单调队列优化到 $\mathcal{O}(n)$ 。

为了做单点修改，我们先看一个引理。

引理：一个长度为 k 的区间中必然有至少一个数被选。

这个引理是显然的，如果没有的话，必然存在某两个相邻的被选的数之间的距离超过了 k 。

这时候就用到一个经典 trick 了：我们倒着再做一次 dp，设 g_i 表示考虑了 $[i, n + 1]$ 这些数，同时选择了 A_i 但是没有将它计入贡献时的答案，此时所求可以等于什么呢？我们随便选一段长度为 k 的区间，不妨设为 l, r ，答案一定等于

$$\min_{i=l}^r (f_i + g_i)$$

当我们单点修改的时候，选择 $[x, x + k - 1]$ 这段区间， $\mathcal{O}(k)$ 更新它们的 f 值然后 $\mathcal{O}(k)$ 用上面的式子算答案，最后还原 f 即可，总时间复杂度 $\mathcal{O}(n + kq)$ 。

qoj5417. Chat Program

给定一个长度为 n 的整数序列 a_1, a_2, \dots, a_n ，同时给定另外四个整数 k, m, c 与 d ，可以进行以下操作至多一次：选择一个长度恰为 m 的连续子数组，并将一个长度为 m ，首项为 c ，公差为 d 的等差数列加到该连续子数组上。

最大化序列中第 k 大的值，求出来这个值。

对于 100% 的数据，保证 $1 \leq k, m \leq n \leq 2 \times 10^5$ 。

Solution: qoj5417. Chat Program

二分答案，将大于等于 mid 的数看作 1，小于 mid 的数看作 0，问题变为检查是否可以有大于等于 k 个 1。

让这个长度为 m 的连续子数组不断向右移动，考虑每一个 a_i ，当这个子数组第一次碰到它的时候， a_i 会被加到最大值，然后子数组不断向右移动， a_i 不断变小，因此 $a_i \geq mid$ 的时刻恰好是一段区间，并且这段区间是可以 $\mathcal{O}(1)$ 算出来的。

每次 check 的时候对每个 a_i 找到它大于等于 mid 的那段时刻，区间加一，问题化为检查是否有某个时刻的值大于等于 k ：区间加一直接前缀和，检查的时候暴力检查所有时刻，总时间复杂度 $\mathcal{O}(n \log A)$ ，其中 A 为答案的范围。

qoj5418. Color the Tree

一棵 n 个结点的树，一开始所有点都是白色。你可以进行若干次操作，每次操作选定一个点 u 并用 a_i 的代价将其子树内距离 u 为 i 的点染黑（同一个点可以被染黑多次），问将所有点染黑的最小代价。

对于 100% 的数据， $n \leq 3 \times 10^5$ 。

Solution: qoj5418. Color the Tree

有两种做法，暴力长链剖分和虚树，这里讲后者。

随便选一个点作为根，由于我们每次染色的点的深度必然相同，所以将不同深度的点分开考虑，设 p_i 表示将深度为 i 的点染黑的最小代价，答案即为 $\sum p_i$ 。

假如所有叶子深度相同，我们想让叶子这一深度的点被染黑，可以怎么办呢？简单做一个 dp 就行，设 u 的深度为 d_u ，叶子的深度为 D ， dp_u 表示将 u 子树内的叶子染黑的最小代价，那么有转移方程 $f_u = \min\{D - d_u, \sum_{v \in \text{son}(u)} dp_v\}$ 。

当我们考虑某一深度的结点时，深度更大的结点是没用的，可以直接删掉，于是当我们要统计深度为 D 的结点的答案时，我们可以以它们为关键点建虚树，并在虚树上跑上述 dp，总时间复杂度 $\mathcal{O}(n \log n)$ 。

qoj5423. Perfect Matching

有 n 个数 A_1, A_2, \dots, A_n 和 n 个点 $1, 2, \dots, n$ ，点 i 和点 j 之间有边当且仅当 $|i - j| = |A_i - A_j|$ ，问该图是否存在完美匹配，若存在，输出方案。

完美匹配是指选出若干边，使得每个点在且仅在一个边的某一个端点上。

对于 100% 的数据，保证 $1 \leq n \leq 10^5$ 。

Solution: qoj5423. Perfect Matching

转换一下条件， $|i - j| = |A_i - A_j|$ 等价于 $i - j = \pm(A_i - A_j)$ ，等价于 $i - A_i = j - A_j$ 或 $i + A_i = j + A_j$ 。

新建一张二分图，对于每个 i ，从左部点的 $i - A_i$ 向右部点的 $i + A_i$ 连边，问题转化为求一种方案使得将这 n 条边划分为若干个长度为 2 的链，其中链的顶点可以相交。

对每一个联通块分别考虑，若边数为奇数一定无解，否则随便搞一棵 dfs 树，从底向上考虑每个点，若它除了和父亲之间的边外，剩余的没有被选中的连边有偶数个，直接将它们配对，否则就再添上它和父亲之间的边进行配对，若最终无法成功将所有边匹配，当且仅当到了根结点后剩下了奇数条边，也就是总边数是奇数，这是我们判断过的。

如果对二分图点的编号用 `std::sort+std::unique` 处理，时间复杂度为 $\mathcal{O}(n \log n)$ ，若用哈希处理，时间复杂度 $\mathcal{O}(n)$ 。

芝士

有 n 个数构成数列 a ，依次将这些数塞进 b 中，每次可以选择放到 b 的开头或结尾，要求最小化

$$\max_{i=1}^{n-1} |b_i - b_{i+1}|$$

对于 100% 的数据， $1 \leq n \leq 10^6, 1 \leq a_i \leq 10^9$ 。

Solution: 芝士

先把问题转化为：将 a 划分为两个子列，要求最小化两个子列相邻元素的差的最大值，并且需要保证 a_1 和 a_2 分别在两个不同的子列中。

二分答案，每次检查答案是否可以大于等于 mid . 先把 a_1, a_2 放到子列，此时两个子列下一个位置可以放的数字区间分别为 $[a_1 - mid, a_1 + mid], [a_2 - mid, a_2 + mid]$ ，考虑放 a_3 到子列，若只能放入某一个子列，直接放进去并且更新该子列的下一个元素的合法区间，若都能放进去，则将其中一个合法区间改为 $[a_1 - mid, a_1 + mid] \cup [a_2 - mid, a_2 + mid]$ ，这仍然是一个区间（因为两者有交），另一个合法区间改为 $[a_3 - mid, a_3 + mid]$ 。

不断进行如上操作，若某个时刻两个合法区间都放不了，则答案不可能 $\geq mid$ 。

时间复杂度 $\mathcal{O}(n \log w)$ 。

猗猗

一个环上 n 个整数 a_0, a_1, \dots, a_{n-1} ，每次可以选择相邻的两个并把它们减少 1。

问是否存在把所有数都变成 0 的方案。

对于 100% 的数据， $n \leq 5 \times 10^5, 0 \leq a_i \leq 10^9$ 。

Solution: 猗猗

设 t_i 表示操作了几次 $(a_i, a_{i+1 \bmod n})$ ，那么有方程组

$$t_0 + t_1 = a_1, t_1 + t_2 = a_2, \dots, t_{n-1} + t_0 = a_0$$

看起来有 n 个未知量和 n 个方程组，可以直接解出来，方法是一式减去二式加三式等等，当 n 是奇数时有 $2t_0 = a_1 - a_2 + a_3 - \dots + a_0$ ，于是就可以直接解方程了，无解当且仅当某个 t_i 不是非负整数。

但是当 n 为偶数的时候，我们会得到 $0 = \sum_{i=0}^{n-1} (-1)^{i+1} a_i$ 的结果，这是因为最后一个方程被前 $n-1$ 个方程表出了，当等式不成立的时候显然无解。

此时有 n 个变量， $n-1$ 个方程组，怎么办呢？考虑 t_0 的限制：

$$t_0 + t_1 = a_1, t_0 - t_2 = a_2, t_0 + t_3 = a_3, \dots, t_0 + t_{n-1} = a_0$$

由于 t_i 是非负整数，因此上面的式子都可以化为关于 t_0 的不等式，例如 $t_0 + t_1 = a_1$ 化为 $t_0 \leq a_1$ ，若这些不等式解集的交存在某个非负整数，那么一定有解，否则无解。

qoj664. Triangle

给定 n 个数 a_i ，从中选出 6 个数组成两个三角形，求三角形周长和的最大值，若无法组成两个三角形，输出 0。

对于 100% 的数据，保证 $1 \leq n \leq 10^5, 1 \leq a_i \leq 10^{15}$ 。

Solution: qoj664. Triangle

将 a 降序排序，首先看只选出一个三角形的情况，若钦定了最长的边，那么剩下的两个边必然选次大的和次次大的，时间复杂度 $\mathcal{O}(n \log n)$ 。

解法一：最坏情况下 a 为 Fibonacci 数列，由于它是指数级别的， $fib_{75} \geq 10^{15}$ ，因此只需要抽出最大的 200 个数就差不多包括了最优解， $\binom{200}{3}$ 枚举第一个三角形，再根据之前的结论找到第二个三角形即可。

解法二：降序排列后，设两个三角形的边的编号分别为 $i < j < k, u < v < w$ ，并且 $k < w$ ，那么若 $k < u$ ，可以通过 $\mathcal{O}(n)$ 预处理前缀选一个三角形的最优解、后缀选一个三角形的最优解 $\mathcal{O}(1)$ 得到，若 $k > u$ ，这时候 i, j, k, u, v, w 必然是相邻的两个数， $\binom{6}{3}$ 暴力枚举每一种情况后检查合法性即可，时间复杂度 $\mathcal{O}(n \log n)$ 。

关于结论的证明：简单调整即可。首先 i, j 必然相邻，否则可以让 j 减小，答案更大且仍然合法，其次 u 必然为 $j+1$ ，否则可以让 k 变为 $j+1$ ，答案更大且仍然合法。类似地，若 v, w 不和前面的数相邻，可以将他们减小，答案更大且仍然合法，证毕。

【APIO】2016 划艇

在首尔城中，汉江横贯东西。在汉江的北岸，从西向东星星点点地分布着 N 个划艇学校，编号依次为 1 到 N 。每个学校都拥有若干艘划艇。同一所学校的所有划艇颜色相同，不同的学校的划艇颜色互不相同。颜色相同的划艇被认为是一样的。每个学校可以选择派出一些划艇参加节日的庆典，也可以选择不出派任何划艇参加。如果编号为 i 的学校选择派出划艇参加庆典，那么，派出的划艇数量可以在 a_i 至 b_i 之间任意选择 ($1 \leq a_i \leq b_i \leq 10^9$)。

值得注意的是，编号为 i 的学校如果选择派出划艇参加庆典，那么它派出的划艇数量必须大于任意一所编号小于它的学校派出的划艇数量。

输入所有学校的 a_i, b_i 的值，求出参加庆典的划艇有多少种可能的情况模 $10^9 + 7$ 的结果，必须有至少一艘划艇参加庆典。两种情况不同当且仅当有参加庆典的某种颜色的划艇数量不同。

对于 100% 的数据，保证 $1 \leq N \leq 500$ 。

Solution: 【APIO】2016 划艇

一个自然的想法是 dp，设 $f_{i,j}$ 表示最后一个派出游艇的学校为 i ，并且派出了 j 个游艇时的方案数，那么有转移

$$f_{0,0} = 1$$

$$f_{i,j} = \begin{cases} \sum_{k=0}^{i-1} \sum_{p=0}^{j-1} f_{k,p}, & j \in [a_i, b_i] \\ 0, & j \notin [a_i, b_i] \end{cases}$$

但是第二维是 10^9 级别的，考虑将所有区间离散化为 $\mathcal{O}(n)$ 个左闭右开的区间，在离散化后的若干不交区间上操作。

具体来说，设 $f_{i,j}$ 表示最后一个派出游艇的学校为 i ，并且派出的游艇在离散化后的第 j 个小区间 $([p_j, p_{j+1}))$ 中的方案数，初始化有 $f_{0,0} = 1$ 。转移可以从前面的区间转移过来：

$$f_{i,j} = \sum_{k=0}^{i-1} \sum_{p=0}^{j-1} f_{k,p}$$

但是我们忽略了两区间相同的转移，怎么修呢？钦定转移时枚举的 k 为最后一个不在区间 j 的派出游艇的学校，剩下了 m 个可以在区间 j 派出游艇的学校，这 m 个学校有多少种放的方法呢？设区间 $[p_j, p_{j+1})$ 的长度为 L_j ，问题等价于有 m 个位置，每个位置可以放 $[0, L_j]$ 之间的一个整数，要求至少选一个正数，且除了 0 之外的数单调递增的方案数。

这个方案数是 $\binom{L_i + m - 1}{m}$ ，证明可以考虑在 $m - 1$ 个 0 和 $[1, L_j]$ 中选出 m 个数，若选了第 i 个 0 就说明第 i 个位置填了 0，剩下被选择的正数依次填到空位中，我们就得到了一个双射，方案数为 $\binom{L_i + m - 1}{m}$ 。

因此有转移

$$f_{i,j} = \sum_{k=0}^{i-1} \sum_{p=0}^{j-1} f_{k,p} \binom{L_j + m_{j,p} - 1}{m_{j,p}}$$

第一个循环顺序枚举离散化后的每个小区间顺便处理下标为 $[0, n]$ 时的组合数，第二个循环逆序更新每个 f_i ，总时间复杂度 $\mathcal{O}(n^3)$ 。

方舟

给定一个可重集 S ，求

$$\sum_{\substack{T \subseteq S \\ |T| \bmod 2 = 0}} \prod_{x \in T} x$$

答案对大质数 P 取模。特别地，定义 $\prod_{x \in \emptyset} = 1$ 。

我们这样给出可重集 S ：给定 n 对数 $\langle a_i, m_i \rangle$ 表示 S 中有 m_i 个 a_i ，再给出自然数 K 表示 S 中还有 K 个元素为 $\frac{1}{k}$ ，其中 k 取遍 $[1, K]$ 中的整数。

对于 100% 的数据，保证 $0 \leq n \leq 10^6$ ， $1 \leq m_i \leq 10^9$ ， $0 \leq a_i$ ， $K < P$ ，其中 P 为给定的模数。

Solution: 方舟

套路地 dp，先一个一个考虑 S 中的数，先看这 n 对数，设 f_0 表示在已经考虑完的所有元素中，元素个数为偶数的子集对答案的贡献， f_1 表示元素个数为奇数的子集对答案的贡献，初始时正在考虑的集合为空集，有 $f_0 = 1, f_1 = 0$ 。当往正在考虑的数构成的集合加入一个数字 x 的时候，有转移

$$f_0 \leftarrow f_0 + x f_1$$

$$f_1 \leftarrow f_1 + x f_0$$

暴力地做，时间复杂度 $\mathcal{O}(\sum m_i + k)$ ，肯定是不行的。

考虑优化这个转移，一个显然的思路是：这是一个线性变换，用矩阵维护，相同的 x 造成的转移可以矩阵快速幂，时间复杂度 $\mathcal{O}(n \sum \log m_i + k)$ 。

但是 k 也很大，怎么做呢？我们将那 k 个分数的转移写出来，上下两式相加，有

$$f_0 + f_1 \leftarrow (1 + \frac{1}{i})(f_0 + f_1)$$

经过 k 次转移后，有

$$f_0 + f_1 \leftarrow \left(\prod_{i=1}^k \frac{i+1}{i} \right) (f_0 + f_1)$$

经典的上下分母消掉，结果为 $f_0 + f_1 \leftarrow (k+1)(f_0 + f_1)$ ，可以 $\mathcal{O}(1)$ 做。

但是我们求的是 f_0 啊！看起来我们还要求出来 $f_0 - f_1$ ，两式相加除以二就是最终要求的答案，但是我们细看一下：当 $k > 0$ 的时候，用 $\frac{1}{i}$ 转移 f 会使 $f_0 = f_1$ ，这造成的后果就是在后面始终有 $f_0 = f_1$ ，因此若 $k > 0$ ， $\frac{f_0 + f_1}{2} = f_0$ 就是最终答案。

当 $k = 0$ 的时候呢？矩阵快速幂都已经把 f_0, f_1 都求出来了。

总时间复杂度 $\mathcal{O}(n \sum \log m_i)$ 。

回过头来看一看，实际上“矩阵快速幂”是没有必要的，两式相加得到 $f_0 + f_1 \leftarrow (x+1)(f_0 + f_1)$ ，直接自然数快速幂就能维护 $(f_0 + f_1)$ 了。两式相减，自然数快速幂同样能维护出来 $f_0 - f_1$ ，于是我们只需要写一个快速幂即可。

铁道

给定一张点数为 n 的竞赛图，边的方向为编号小的点指向编号大的点，你需要将所有边划分成尽量少的链，并且满足每条链的边数不超过 3，求出最少划分为多少条链，输出方案。

对于 100% 的数据，保证 $2 \leq n \leq 3000$

Solution: 铁道

尝试找到一个下界并构造：一个显然是下界是 $\frac{\binom{n}{2}}{3}$ ，这是因为共有 $\binom{n}{2}$ 条边而每条链至多覆盖 3 个边。但是这个界太松了， $n = 3$ 就可以卡掉，有没有更紧的界呢？

将 $[1, n]$ 划分为 $[1, m] \cap [m+1, n]$ ，那么对任意的 $x \in [1, m]$ ， $y \in [m+1, n]$ ， (x, y) 这条边都应该在某个链上，并且对于 $(x, y) \neq (x', y')$ ，这两条边一定不会在同一条链上，于是有下界 $m(n-m)$ 。

当 n 为偶数时，下界为 $\frac{n}{2} \times \frac{n}{2} = \frac{n^2}{4}$ ，有以下构造：

对每个 $i \in [1, \frac{n}{2}]$ ，找到 $j = n+1-i$ ，对任意 $k \in [i+1, j]$ ，有链 $\langle (i, k), (k, j) \rangle$ ，不难看出每条边都唯一在一条链上，并且链的数量为

$$\sum_{i=1}^{\frac{n}{2}} (n+1-2i) = \frac{n^2}{4}$$

当 n 为奇数时，下界为 $\frac{n-1}{2} \times \frac{n+1}{2} = \frac{(n-1)^2 + 2n - 2}{4} = \frac{(n-1)^2}{4} + \frac{n-1}{2}$ 。

这时可以先构造出 $n-1$ 的方案，对于 $i \in [1, \frac{n-1}{2}]$ ，直接新建一条链，其中只有一条边 (i, n) ，对于 $i \in [\frac{n-1}{2} + 1, n-1]$ ，必然有某条以 i 为结尾的边数小于 3 的链，把 (i, n) 接到这条链上即可。

时间复杂度 $\mathcal{O}(n^2)$ 。

启动

对于正整数 $n \geq 3$ ，定义 $f(n)$ 为最大的小于 $n-1$ 的整数 k 使得 $k^2 \equiv 1 \pmod{n}$ 。例如 $f(15) = 11$ 。

共有 q 次询问，对每次询问 n 你需要求出 $f(n)$ 。

对于 100% 的数据，保证 $1 \leq q \leq 10^3$ ， $3 \leq n \leq 10^{18}$ 且数据随机。

Solution: 启动

首先化一下式子：

$$k^2 \equiv 1 \pmod{n} \iff (k-1)(k+1) \equiv 0 \pmod{n} \iff n \mid (k-1)(k+1)$$

若 n 为素数，上式说明了 k 只能等于 1 或 $n-1$ 。

若 $n = p^\alpha$ ，其中 p 为奇素数，那么我们归纳证明如下引理：

引理：此时 k 只能等于 1 或 $p^\alpha - 1$ 。

证明：当 $\alpha = 1$ 时命题成立，下证若当 $\alpha = a$ 时成立，则当 $\alpha = a+1$ 时也成立。

由于 $k^2 \equiv 1 \pmod{p^{a+1}} \implies k^2 \equiv 1 \pmod{p^a} \implies k \equiv \pm 1 \pmod{p^a} \implies k = tp^a \pm 1$ 。

接着推下去，有 $(tp^a \pm 1)^2 = t^2 p^{2a} \pm 2tp^a + 1 \equiv 1 \pmod{p^{a+1}}$ ，这等价于 $2t \equiv p \pmod{p^{a+1}}$ ，由于 $t \in [0, p-1]$ ，因此必有 $t = 0$ ，于是 $k \equiv \pm 1 \pmod{2^{a+1}}$ ，证毕！

当 p 是偶素数，也就是 2 的时候呢？此时最后一步中的 t 可以等于 1，于是 k 还可以等于 $\frac{n}{2} \pm 1$ 。

当 n 不是一个素数的幂的形式，可以将 n 唯一分解，之后得到一些同余方程组，例如 $k \equiv \pm 1 \pmod{9}$ and $k \equiv \pm 1 \text{ or } 3 \text{ or } 5 \pmod{8}$ 就是 72 需要满足的，用 Pollard-rho 在 $\mathcal{O}(n^{0.25})$ 唯一分解 n ，之后爆搜每个方程的每种可能取值，中国剩余定理（或扩展中国剩余定理）求出来最终的 k 对答案取 max 即可。

乐乐乐

小 X 有一个集合 S 。一开始, S 中包含 1 到 n 的所有整数。

然后, 小 X 会给出一个大小为 m 的集合 T , 满足 $T \subset S$, 然后将 T 中的元素从 S 中删去, 得到新的 S 。

小 X 喜欢二的次幂, 看到它们会让小 X 觉得快乐。因此他决定拿 S 做一个游戏:

每一次, 小 X 会取一对正整数 u, v , 满足 $u \in S, v \in S, u \neq v$, 且存在整数 p , 使得 $u + v = 2^p$ 。

然后, 小 X 会将 u, v 从 S 中删去。之后重复步骤 1, 直到无法操作为止。

小 X 想知道, 他最多能取出多少对数。

对于 100% 的数据, 保证 $1 \leq n \leq 2^{63} - 1, 0 \leq m \leq \min(10^5, n)$ 。

Solution: 乐乐乐

一个比较自然的想法是: 若 $u + v$ 是 2 的幂, 那么将 u, v 连边, 答案即为在图中选尽量多的边使得每个点相连的边中至多有一个被选择。

但是这样很难做, 本题的重点在于接下来的转化: 若 $u > v$ 并且 $u + v = 2^t$, 那么其实这个 t 对于 u 是唯一确定的: 就是最小的比 u 大的 2 的幂。因此我们给边定向, 钦定大点连向小点, 图变成了一棵树的结构, 贪心地尽可能选叶子即可, 时空复杂度 $\mathcal{O}(n)$ 。

当 $m = 0$ 且 $n = 2^p - 1$ 的时候, 将点分层, 第 i 层结点为 $[2^{i-1}, 2^i - 1]$, 对于最后一层, 也就是第 p 层的结点, 若它不是 2^{p-1} , 则它一定是一个叶子, 或者更普遍地, 对于 $i < j$, 第 i 层除了 2^{i-1} 这个点的所有点都会和第 j 层的点之间有边。

这个性质有什么用呢? 因为我们会贪心地选叶子, 而最后一层的结点有 $2^{p-1} - 1$ 个点是叶子, 将它们选完之后就只剩下 2^{p-1} 一个点了, 因此答案为 $2^{p-1} - 1$ 。

再扩展一下, 当 $m = 0$ 且 $n \neq 2^p - 1$ 的时候怎么做呢? n 所在的那一行中, $[n + 1, 2^p - 1]$ 这些点会被删去, 而它们的父亲恰好是 $[1, 2^p - n - 1]$, 最后一层的结点依然会被优先匹配, 剩下的部分就是一个规模为 $2^p - n - 1$ 的子问题, 递归地算就行, 时间复杂度 $\mathcal{O}(\log n)$ 。

当 $m \neq 0$ 的时候呢? 考虑 $[2^p - n, n]$ 这些结点, 若其中的某个结点被删除且它是叶子, 那么它的父亲就会空出来, 否则它就无法和其叶子匹配。对于空出来的那些结点, 先考虑它们在大于等于 $2^p - n$ 部分的连边, 贪心的匹配, 可以用 `std::set` 维护, 之后有一些树的根会空出来, 若其父亲在 $[1, 2^p - n - 1]$ 且还没有匹配, 将它和其父亲匹配并将其父亲作为下一轮递归中被删除的结点, 递归到规模为 $[2^p - n - 1]$ 的子问题, 时间复杂度 $\mathcal{O}(m \log m \log n)$ 。

典典典

小 X 有一张画布，他想要创作一幅传世经典。

这张画布是一个 n 行 m 列的网格，一开始每一个格子都是白色的。

然后，小 X 要画 Q 笔，每一笔可以用 4 个整数 $op\ p\ l\ r$ 表示：

- 若 $op = 0$ ，表示小 X 将画布的第 p 行从左往右的第 l 到第 r 个格子涂成黑色。
- 若 $op = 1$ ，表示小 X 将画布的第 p 列从上往下的第 l 到第 r 个格子涂成黑色。

每一个格子可以被涂黑多次。

现在，小 X 把自己的作品拿给专家品鉴。专家品鉴的依据是图中的连通块个数。

专家会从上往下展开小 X 的画作，第 t ($1 \leq t \leq n$) 个时刻，专家只能看到这幅画的前 t 行，即后 $n - t$ 行的内容不会产生任何影响。

每个时刻，专家会根据图画中他看到的部分的黑色连通块个数来评价小 X 的画作。两个黑色格子在同一连通块当且仅当从一个格子开始，通过边相邻的黑色格子，可以到达另一个格子。

小 X 想知到专家的评价结果。即，对于每个 t ，他想知道自己的画作的前 t 行包含了多少个黑色连通块。

对于 100% 的数据，保证 $n, m, Q \leq 10^5$ 。

Solution: 典典典

将竖着的操作看成结点，横着的操作看作边，从上往下扫描线，用一个 `std::set(S)` 维护当前行中有黑色格子的列，用另一个 `std::set(T)` 和一个并查集维护目前某些连续的列必然联通（这是操作 0 带来的贡献）。

当这一行要加上一个新格子的时候，将格子所在的列加入 S ，若 T 中某个元素包含了该列，就将该元素分裂为两半，然后加入一个新的长度为 1 的元素表示该列。

当某一列的黑色格子在上一行结束的时候，将它从 S 中删掉同时若它是 T 中某个元素的端点则更新 T 。

考虑每一行的边时，取出 T 中的那些元素，在并查集中合并。

总时间复杂度 $\mathcal{O}(n + q \log q)$ 。

【APIO2016】Gap

有 N 个严格递增的非负整数 a_1, a_2, \dots, a_N ($0 \leq a_1 < a_2 < \dots < a_N \leq 10^{18}$)。你需要找出 $a_{i+1} - a_i$ ($0 \leq i \leq N-1$) 里的最大的值。

你的程序不能直接读入这个整数序列,但是你可以通过给定的函数来查询该序列的信息。关于查询函数的细节,请根据你所使用的语言,参考下面的实现细节部分。

你需要实现一个函数,该函数返回 $a_{i+1} - a_i$ ($0 \leq i \leq N-1$) 中的最大值。

你需要实现一个函数 `findGap(T, N)`, 该函数接受下面的参数,并返回一个 `long long` 类型的整数:

- T : 子任务的编号 (1 或者 2)
- N : 序列的长度

你的函数 `findGap` 可以调用系统提供的查询函数 `MinMax(s, t, &mn, &mx)`, 该函数的前两个参数 s 和 t 是 `long long` 类型的整数, 后两个参数 `&mn` 和 `&mx` 是 `long long` 类型的整数的指针 (`mn` 和 `mx` 是 `long long` 类型的整数)。当 `MinMax(s, t, &mn, &mx)` 返回时, 变量 `mn` 将会存储满足 $a_i \in [s, t]$ 中 a_i 的最小值, 变量 `mx` 将会存储满足 $a_i \in [s, t]$ 中 a_i 的最大值。如果区间 $[s, t]$ 中没有序列中的数, 则 `mn` 和 `mx` 都将存储 -1 。在查询时需要满足 $s \leq t$, 否则程序将会终止, 该测试点计为 0 分。

对于所有的测试点, 有 $2 \leq N \leq 100000$, 每一个测试点开始测试之前, M 都将被初始化为 0。

子任务 1 (30 分): 每一次调用 `MinMax` 都将使 M 加 1。为了获得所有分数, 需要满足对于该子任务下的所有测试点, 都有 $M \leq \frac{N+1}{2}$ 。

子任务 2 (70 分): 定义 k 为调用 `MinMax` 时, 区间 $[s, t]$ 中的序列中数的数量。每次调用 `MinMax`, 将使 M 加上 $k+1$ 。对于每一个测试点, 如果 $M \leq 3N$, 你将得到 70 分, 否则将得到 $\frac{60}{\sqrt{\frac{M}{N} + 1} - 1}$ 分。你的该子任务的得分是其下所有测试点中的最低分。

Solution: 【APIO2016】Gap

子任务 1 是来送分的, 先查询全局最大、最小值, 然后收缩范围求次大、次小值, 依次类推即可, 查询次数为 $\lfloor \frac{n+1}{2} \rfloor$ 。

子任务 2 更难想一些, 先查询出全局极值 $Minv, Maxv$, 那么最劣情况下, 剩下的 $n-2$ 个数会均匀地分布在 $(Minv, Maxv)$, 于是我们将这个区间分为 $n-1$ 个块, 答案必定会出现在一个块的结尾和下一个块的开头之间, 中间数字只会被查询到两次, 两侧的数字为一次, 同时查询次数为 N , 最终 $M = 3N - 2$ 。

【CEOI2017】 Mousetrap

有一个有 n 个房间和 $n - 1$ 条走廊的迷宫，保证任意两个房间可以通过走廊互相到达，换句话说，这个迷宫的结构是一棵树。

一个老鼠被放进了迷宫，迷宫的管理者决定和老鼠做个游戏。

一开始，有一个房间被放置了陷阱，老鼠出现在另一个房间。老鼠可以通过走廊到达别的房间，但是会弄脏它经过的走廊。老鼠不愿意通过脏的走廊。

每个时刻，管理者可以进行一次操作：堵住一条走廊使得老鼠不能通过，或者擦干净一条走廊使得老鼠可以通过。然后老鼠会通过一条干净的并且没被堵住的走廊到达另一个房间。只有在没有这样的走廊的情况下，老鼠才不会动。一开始所有走廊都是干净的。管理者不能疏通已经被堵住的走廊。

现在管理者希望通过尽量少的操作将老鼠赶到有陷阱的房间，而老鼠则希望管理者的操作数尽量多。请计算双方都采取最优策略的情况下管理者需要的操作数量。

注意：管理者可以选择在一些时刻不操作。

对于所有的数据， $1 \leq n \leq 10^6$ 。

Solution: 【CEOI2017】 Mousetrap

把陷阱结点作为根，老鼠如果在第一回合向下走，那么它一定会被自己走过的路堵死在一个叶子结点，管理者需要做的就是将该叶子到根的路径上冗余的边堵上。

设 dp_u 表示老鼠当前在结点 u ，它如果选择向下走，所能消耗的最大时间，显然管理者会把 $son(u)$ 中 dp 值最大的给堵死，于是 dp_u 等于其所有儿子中 dp 值的次大值再加上儿子个数（因为管理者总要堵上剩下的儿子，擦掉老鼠走的这条边）。

问题在于老鼠第一步可能是向上走的，这也不难解决：设 sum_u 表示将老鼠从点 u 逼到根结点需要多少操作，转移为 $sum_u = sum_{fa} + |son(u)| - 1 + [u = mouse]$ ，二分答案转化为判定性问题，每次让老鼠不断跳到父亲，若当前有某个儿子可以使得跳到那里之后 $dp_v + sum_u > mid$ ，那么我们就应该让老鼠跳进去而是提前堵死，答案不合法当且仅当管理者来不及堵住（老鼠跳到这里的时候已经挡不住了）或者 mid 不足以堵住所有的这些位置，总时间复杂度 $\mathcal{O}(n \log n)$ 。

qoj7589. Game Prediction

Alice 和 Bob 玩游戏：有一个长度为 n 的正整数序列 A ，每次操作可以选 A 的开头或末尾的一个元素，并从 A 中删除，Alice 先操作，每个人的目标都是让自己所取元素的和尽量大，他们都会采取最优策略。

给定 q 次询问，每次询问两个整数 $1 \leq l \leq r \leq n$ ，表示若 Alice 和 Bob 在 $A_{l..r}$ 这个子区间内玩游戏，两者最终的得分会是多少。

对于 100% 的数据，保证 $1 \leq n \leq 10^5$ ， $1 \leq q \leq 2 \times 10^5$ ，保证 A 序列随机生成。

Solution: qoj7589. Game Prediction

若 A 中仅有三个元素 x, y, z 并且 $x \leq y, z \leq y$ ，那么 Alice 取走 x 或 z 后，Bob 必取走 y ，Alice 小亏，这时候就要取走 z 或 x （剩下的那个元素）来弥补损失。

我们尝试计算最终 Alice 的得分减去 Bob 的得分，再根据得分的和求出两者各自的分数，因此对于上面的那三个数，我们可以直接将 x, y, z 合并为 $x + z - y$ 表示取走这些元素的得分。

对于多个数的情况也是可以这样做的：不断加入元素到末尾并检查末尾的三个数是否可以合并，最终序列会变为一个下凸壳，可以贪心：操作的人必定会选择可以选择的最大值。

注意到这样的下凸壳是可以合并的，方法就是不断将右边的下凸壳的开头塞进左边下凸壳的末尾并尝试合并。

因此可以线段树：每个结点维护该区间的下凸壳，由于数据随机，下凸壳的顶点不会很多，按照上面的策略暴力合并即可，询问在线段树上正常做，得到对应下凸壳后贪心。

CF618F. Double Knapsack

给你两个序列 A, B ，元素个数均为 n 且里面的元素为在 $[1, n]$ 的整数，请你分别找出 A, B 的子列，满足它们的元素和相等，无解输出 -1 。
对于 100% 的数据，保证 $1 \leq n \leq 10^6$ 。

Solution: CF618F. Double Knapsack

不妨加强一下原问题：变为找到两个 A, B 的子区间，满足它们的元素和相等，下面我们设 P 为 A 的前缀和， Q 为 B 的前缀和，同时满足 $P_n < Q_n$ （否则交换 A, B ）。

对于每一个 $i \in [1, n]$ ，找到最大的 j 满足 $P_i \geq Q_j$ ，若取到了等号，我们就已经找到了一组解。

注意到我们在前提条件中已经有了 $P_n < Q_n$ ，因此对于每一个 (i, j) 对必然有 $P_i - Q_j \in [0, n)$ （否则我们可以让 j 变为 $j + 1$ ），而一共有 n 对这样的 (i, j) ，由抽屉原理，必然有两对 $(i_1, j_1), (i_2, j_2)$ 满足 $P_{i_1} - Q_{j_1} = P_{i_2} - Q_{j_2}$ ， $i_1 < i_2$ ，那么 A 序列的 $[i_1 + 1, i_2]$ ， B 序列的 $[j_1 + 1, j_2]$ 就是符合条件的两个子区间。

在构造的同时我们证明了：必然有这样的两个子区间，因此不会有输出 -1 的情况。用双指针维护，时间复杂度 $\mathcal{O}(n)$ 。

CF1223F. Stack Exterminable Arrays

给一个序列进行栈操作，从左到右入栈，若当前入栈元素等于栈顶元素则栈顶元素出栈，否则当前元素入栈。若进行完操作后栈为空，这说这个序列是可以被消除的。

给你一个长度为 n 的序列 a ，问 a 有多少子串是可以被消除的。

对于 100% 的数据，保证 $1 \leq n \leq 3 \times 10^5$ ， $1 \leq a_i \leq n$ 。

Solution: CF1223F. Stack Exterminable Arrays

当 a_i 的值域很小时，问题变为「CSP-S2023」T4. 消消乐，我们先讨论该题的解法（值域为 26）。

引理：若一个序列可消，那么每次任取两个相邻的可消元素消掉均可。

感性理解一下就是，随便选两个可消的元素消掉，不会使原本可以被消除的一对元素变得不可消除（不能变为相邻的元素）。

因此有了一个 $\mathcal{O}(n^2)$ 做法，枚举右端点，从右往左不断加元素，若栈顶两个元素相等就直接弹出来，当栈为空的时候让答案加一。

这样做很蠢，因为每一个元素会入栈 $\mathcal{O}(n)$ 次，不同的右端点之间也没有联系，改进方法是：既然我们要在栈为空的时候让答案加一，不妨就直接找到这些位置，具体的，当右端点为 r 时，什么时候栈会第一次变空呢？就是以 $r - 1$ 为右端点时栈中仅有一个元素 a_r 的最早的时刻，此时这个元素就可以和 a_r 消掉了。

因此对每个右端点 r 记录 $f_{r,x}$ 表示以 r 为右端点做栈操作时，栈中仅有一个元素 x 的最早的时刻， ans_r 表示以 r 为右端点的可消区间的个数，那么 $ans_r = ans_{f_{r-1,a_r}-1} + 1$ ，意义是 $[f_{r-1,a_r}, r]$ 这个区间是以 r 为右端点的，左端点最靠右的可消区间，它会让 ans_r 加一，剩下的固定右端点的可消区间 $[l, r]$ 必然可以被拆分为两个可消区间 $[l, f_{r-1,a_r}-1] \cup [f_{r-1,a_r}, r]$ ，方案数就是 $ans_{f_{r-1,a_r}-1}$ 。

f 的维护更容易，先从 f_{r-1} 继承过来，然后让 $f_{r,a_r} = r$ 即可，时间复杂度 $\mathcal{O}(n\Sigma)$ ，其中 Σ 为值域（字符集大小）。

用 `std::map` 维护 f 即可做到 $\mathcal{O}(n \log n)$ ，下面我们讲另一种 $\mathcal{O}(n \log n)$ 的做法。

以 1 为左端点，不断将元素塞进栈中，策略和上个做法一样，不同之处在于我们用哈希对每一个右端点时栈的状态进行记录，设为 $Hash_r$ 。

显然，若区间 $[l, r]$ 可消，当且仅当 $Hash_{l-1} = Hash_r$ ，因此用一个 `std::map` 记录之前出现过的哈希值和次数即可，时间复杂度 $\mathcal{O}(n \log n)$ 。

uoj#327. 【UTR #3】去月球

Scape 和 Mythological 交流了玩几何冲刺的经验之后, Mythological 非常高兴, 又推荐给 Scape 一款游戏 To the moon。

游戏中一位老人 John 的记忆被药物尘封, 进行了解除尘封的仪式之后, Scape 走进了他的记忆。

John 的记忆中有一个唤做 River 女孩的身影, 有着数不尽的纸兔子, 有一个沙包, 一只鸭嘴兽玩偶, 一座灯塔。Scape 被深深的打动了。

”我的鸭嘴兽和沙包又在哪里呢?” Scape 这样想到, 不禁幻想出 Mythological 决定给他送 n 份礼物, 其中第 i 份的种类是 a_i 。这些礼物按顺序排成一行。

她挑选礼物的方式很特别, 她每次会选择两份种类相同的礼物, 并且这对礼物满足它们之间没有尚未拿走的礼物, 并将这对礼物拿走。

现在 Scape 给出了若干次询问, 每次询问如果他送给她的是区间 $[L_i, R_i]$ 之间的礼物, 那么她最多能拿走多少份礼物。

对于 100% 的数据, 保证 $1 \leq n \leq 10^5, 1 \leq q \leq 2 \times 10^6$ 。

Solution: uoj#327. 【UTR #3】去月球

显然若可以拿走一对礼物时, 直接拿走它们不会让答案更劣。

建一个栈和 trie, 从左到右扫 a 数组, 假如栈顶元素和 a_i 相等, 就弹出栈顶, 同时在 trie 中向父亲走, 否则往栈中 push 进 a_i , 在 trie 中走向边为 a_i 的儿子 (没有就新建), 同时记录下每次操作后在 trie 中的位置, 记为 p_i 。

若一个子区间 $[l, r]$ 可以被消除, 那么从 p_{l-1} 开始按照上述策略在 trie 上走 a_l, a_{l+1}, \dots, a_r , 必然会走回到 p_{l-1} , 也就是说, 每个询问的答案其实就是 $r - l + 1$ 再减去剩下的元素个数, 而剩下的元素个数其实就是 trie 中 p_{l-1} 和 p_r 之间的距离。

问题转化为求 lca, 倍增处理即可, 时间复杂度 $\mathcal{O}((n + q) \log n)$ 。

qoj6504. Flower's Land 2

给定一个长度为 n 的序列 A , A 中元素为 $0, 1, 2$, 定义一次消除为: 选取两个相邻的相同元素并删掉, 一个子区间 “可消” 当且仅当该子区间可以通过若干次消除, 最终没有任何元素。

给定 q 次操作, 每次操作形如将一个子区间内的元素在模 3 意义下加一或查询一个子区间是否可消, 输出 Yes 或 No。

对于 100% 的数据, 保证 $1 \leq n, q \leq 5 \times 10^5$ 。

Solution: qoj6504. Flower's Land 2

首先, 若一个序列可消, 那么每次消除的时候随便选一组即可, 这不会使原本可消的序列变得不可消。

序列问题我们希望放到线段树上, 具体来说我们希望找到一个非阿贝尔群, 使得两个相邻的相同元素对应的群中元素互逆, 乘起来恰为单位元, 那么序列可消当且仅当所有元素乘起来是单位元, 矩阵恰好满足这个性质: 矩阵乘法几乎不满足交换律, 并且具有结合律。在本题中, 随机三个可逆方阵 T_0, T_1, T_2 , 定义一个长度为 n 的矩阵序列 M , 若 i 为奇数那么 $M_i = T_{A_i}$, 否则 $M_i = T_{A_i}^{-1}$, 一个子区间可消当且仅当这些矩阵乘起来恰好为 I (单位矩阵), 线段树维护平移 $0, 1, 2$ 后的矩阵乘积即可实现模意义下加的操作, 时间复杂度 $\mathcal{O}(k^3(n + q) \log n)$, 其中 k 为构造的方阵的行 (列) 数, 取 2 即可。

【APIO】2016 烟火表演

给定一棵边带权的树，每次可以花费 1 的代价将边权减少 1 或增加 1（不能小于 0），你可以任意次进行上述操作，求让所有叶子结点到根的路径权值和相等的最小代价。

对于 100% 的数据，保证点数不超过 3×10^5 。

Solution: 【APIO】2016 烟火表演

设 $dp_u(x)$ 表示将以 u 为根的子树中的叶子到 u 的距离都变为 x 的最小代价，由于值域很大，直接做肯定不行。看这个转移有没有什么特点：将 $dp_u(x)$ 视作一个函数，那么这是一个分段线性的下凸函数，我们要做的操作有两个：一是将当前子树的贡献算进父亲里，而是合并所有儿子的贡献。

对于前者，设当前要将 $dp_u(x)$ 向上合并，设 u 到父亲 f 之间的边权为 w ， $[L, R]$ 表示 $dp_u(x)$ 中斜率为 0 的那段区间，于是

$$dp_f(x) = \begin{cases} dp_u(x) + w & x \in (-\infty, L] \\ dp_u(L) + (w - (x - L)) & x \in (L, L + w] \\ dp_u(L) & x \in (L + w, R + w] \\ dp_u(L) + ((x - R) - w) & x \in (R + w, +\infty) \end{cases}$$

上面的式子都是由

$$dp_f(x) = \min_{0 \leq k \leq x} |w - (x - k)| + dp_u(k)$$

得来的，当 $x \leq L$ 时，由于凸包在 $(-\infty, L]$ 一侧斜率都比 -1 小，而让 w 增加或减少 1 的代价为 1，因此我们希望 k 尽可能大，于是直接取 $k = x$ 有 $dp_f(x) = w + dp_u(x)$ ；当 $x \in (L, L + w]$ 时，我们只需要保证 $k = L$ 就能取到最小值，因此 $dp_f(x) = dp_u(L) + w - (x - L)$ ；当 $x \in (L + w, R + w]$ 的时候，什么都不用动就在最小值，有 $dp_f(x) = dp_u(L)$ ；当 $x > R + w$ 的时，要让 $dp_u(k)$ 取到最小值，取 $k = R$ 有 $dp_f(x) = dp_u(R) + ((x - R) - w) = dp_u(L) + ((x - R) - w)$ 。

凸包是怎么变化的呢？将 $x \leq L$ 的部分向上平移 w ，在 $[L, L + w]$ 部分插入了一条斜率为 -1 的直线，在 $> R + w$ 的部分改为斜率为 1 的直线。

我们只存凸壳顶点的横坐标，并规定从左到右，每遇到一个顶点，斜率就加一，不存在的斜率用两个相同位置的点表示。

但是我们最终要求的实际上是 $dp_1(x)$ 的最低点的函数值而非取到最小值的 x ，但是我们可以很容易地知道 $dp_1(0)$ 为所有边权的和（将所有边都删成 0），于是最低点的函数值实际上就是 $dp_1(0)$ 减去所有 $x \leq L$ 部分的顶点的横坐标值（相同的两个顶点算两次）。

再看怎么快速维护这些凸壳：先将斜率 > 1 的部分删掉，只需要弹出 $|son_u| - 1$ 个右侧顶点即可，之后弹出两个右侧顶点，它们是 R 和 L ，然后再将 $L + w$ 和 $R + w$ 塞进凸壳右侧即可，然后将 f 的凸壳和得到的新凸壳的顶点合并起来。

上面的操作都可以用可并堆维护，弹出右侧顶点就是弹出大根堆的根，合并两个凸壳的顶点就是合并两个堆，时间复杂度 $\mathcal{O}(n \log n)$ 。

CTT2018. 芒果冰加了空气

作为 C 市的中考状元，小针毫无悬念地进入了面子中学学习信息学竞赛。

在进入面子中学后，他在王学长的指引下学习了许多算法，其中一个就是点分治。

小针学的点分治是王学长教给他的，它是一个这样的算法：

- 初始时当前连通块是整棵树。
- 首先，在当前连通块中找到任意一个点 u 作为该次的分治中心（不必是重心）。
- 其次，把点 u 在当前连通块中删去，可以得到若干个连通块。对于每个连通块再递归进行这样的操作。

不难发现，小针从黑心王学长那里学到的点分治在最坏情况下递归层数可以达到 $O(n)$ 层。现在，好奇的小针想要知道，对于一棵给定的包含 n 个节点的树，他有多少种不同的点分治方案呢？因为答案可能很大，你只需要输出它对 $10^9 + 7$ 取模的值即可。

两种点分治方案不同当且仅当某一个连通块所选的点分治中心不同。

对于 100% 的数据，保证 $1 \leq n \leq 5000$

Solution: CTT2018. 芒果冰加了空气

设 $dp_{u,d}$ 表示在以 u 为根的子树中进行题目中所给点分治，点 u 的深度为 d 的方案数，合并两个点分治结构就按照经典的树上 dp 的合并方式写两个循环合并，后缀和优化一下，时间复杂度 $O(n^2)$ 。合并的时候需要乘一个组合数，可以预处理出来。

CTT2018. esperar

立冬之后，北京的空气就变得稀薄似的，每次呼吸都要十分谨慎，生怕多吸一口空气就会将自己冻伤似的，如此在风中战栗者不一而足。公交车站前，等待 819 路的人们排起长龙，或许是行将周末的缘故罢，他们的脸上似乎少了那么一丝疲倦。此刻，小 W 也只是看着夜空中那一轮月，看着熙熙攘攘走过的人们，抖动着身体以争取一丝温暖，静静地，等待着公交车的到来。

等车时，小 S 写给了小 W 共 n 个正整数 a_i ，此时，小 W 会先写出 n 个正整数 b_i ，使得对每一个 i ，都有 a_i 是 b_i 的倍数，写完了之后，小 W 会再写出 n 个正整数 d_i ，使得对于每一个 i ，都有 b_i 是 d_i 的倍数。

如果小 W 写出的数，满足 $\prod_{i=1}^n d_i^2 \geq \prod_{i=1}^n b_i$ ，小 S 就会认为小 W 选数的方案是“资资”的。请问小 W 有多少种选数方案，是被小 S 认为“资资”的，结果对质数 998244353 取模。两种方案是不一样的，当且仅当存在一个 b_i 或一个 d_j 不同。

对于 100% 的数据，保证 $n \leq 100$ ， $1 \leq a_i \leq 10^9$ 。

Solution: CTT2018. esperar

首先若一个方案满足 $\prod_{i=1}^n d_i^2 < \prod_{i=1}^n b_i$ ，那么将 d_i 变为 $\frac{b_i}{d_i}$ 就可以得到一个 $\prod_{i=1}^n d_i^2 > \prod_{i=1}^n b_i$ 的方案，因此这两个方案数是相同的。

假如我们能算出总方案数以及 $\prod_{i=1}^n d_i^2 = \prod_{i=1}^n b_i$ 的方案数，那么两者相加除以 2 就是我们要求的方案数。

对于总方案数，对每个 a_i 分别考虑最终乘法原理，将 a_i 唯一分解为 $\prod_{i=1}^m p_i^{\alpha_i}$ ，那么选取 (b_i, d_i) 的方案数就是

$$\prod_{i=1}^m \frac{(\alpha_i + 1)(\alpha_i + 2)}{2}$$

于是总方案数就可以 $O(n\sqrt{w})$ 求出来了，其中 w 是值域。

对于后者, 即求 $\prod_{i=1}^n d_i^2 = \prod_{i=1}^n b_i$ 的方案数, 对每个分解出来过的质因子分别考虑, 让 $\prod b_i$ 中的幂等于 $\prod d_i$ 中的幂的两倍, 可以 dp.

具体的, 还是乘法原理, 接下来只考虑某个质数 p . 设 c_i 表示 a_i 中 p 的出现次数, 即最大的 k 满足 $p^k \mid a_i$, 设 $dp_{i,x}$ 表示考虑完 a_1 到 a_i 后, $\prod b_i$ 中 p 的幂减去 2 倍的 $\prod d_i$ 中 p 的幂为 x 时的方案数, 那么答案即为 $dp_{n,0}$. 转移时枚举 b_i 中 p 的幂减去 2 倍的 d_i 中 p 的幂, 方案数可以 $\mathcal{O}(1)$ 算, 再枚举上一层的差转移即可, 空间可以用滚动数组优化.

AT_abc326_g Unlock Achievement

有 n 项技能和 m 项成就, 初始时你的每项技能等级为 1, 你可以花费 C_i 的金钱将第 i 项技能的等级提升 1, 不限次数.

每项成就有限制, 第 i 项成就有限制 $L_{i,j}$ 表示必须使第 j 项技能的等级不小于 $L_{i,j}$, 若所有的 n 个限制都没满足, 那么第 i 项成就就会给你带来 A_i 的金钱.

问最多可以获得多少金钱.

对于 100% 的数据, 保证 $1 \leq n, m \leq 50, 1 \leq L_{i,j} \leq 5$.

Solution: AT_abc326_g Unlock Achievement

一眼网络流, 先让答案为 $\sum_{i=1}^m A_i$, 考虑最小的花费.

将每项技能拆为 4 个点表示提升的次数, 从原点向这 4 个点分别连容量为 C_i 的边, 之后对每个成就, 向汇点连容量为 A_i 的边.

对于成就的限制: 考虑 $L_{i,j}$ 的限制, 从技能 j 拆出的点中的前 $L_{i,j} - 1$ 个点向装备 i 连容量为 $+\infty$ 的边, 跑出最小割 P , 答案即为 $\sum_{i=1}^m A_i - P$.

理由是: 对于装备连向汇点的边, 我们定义割掉这条边表示不满足该成就, 对于源点连向某一技能的边, 我们定义割掉前 x 条边表示提升该技能 x 次, 那么一个合法的割就对应了一种合法的方案.

区间

给定 m 个区间和一个自然数 $k \leq m$, 设第 i 个区间为 $[l_i, r_i]$, $l_i, r_i \in [1, n]$, $r_i - l_i \geq 1$, 要求构造一个长度为 n 的序列 A 满足恰有 k 个给定区间满足 $\exists i \neq j \in [l, r], \text{ s.t. } A_i = A_j$. 对于 100% 的数据, 保证 $1 \leq n, m \leq 5 \times 10^5, k \leq m$.

Solution: 区间

将所有区间按照左端点为第一关键字升序, 右端点为第二关键字降序, 找到排序后的第 k 个区间, 设为 $[l, r]$, 构造序列

$$A_i = \begin{cases} i, & i \in [l+1, r-1] \cup [r+1, n] \\ x, & i \in [1, l] \cup r \end{cases}$$

其中 $x \notin [l+1, r-1] \cup [r+1, n]$.

做法正确性的证明: 若某个区间 $[l_i, r_i]$ 符合条件, 那么必然有 $l_i < l$ 或 $l_i = l, r_i \geq r$, 只有前 k 个区间满足条件.

【UR #26】石子合并

目前有 n 个非空数组 A_1, A_2, \dots, A_n (每个数组内部不一定有序), 其元素均在 $[1, m]$ 之间。

对 $i = 2, 3, \dots, n$ 依次执行 $A_1 \leftarrow \text{merge}(A_1, A_i)$ 后, A_1 大小恰为 m , 且 A_1 中的元素依次为 a_1, a_2, \dots, a_m 。

merge 操作的 c++ 实现如下:

```
1 #define stone int
2 #define stones std::vector<stone>
3 stones merge(stones A, stones B)
4 {
5     stones C;
6     while (A.size() && B.size())
7         if (A[0] <= B[0])
8             C.push_back(A[0]), A.erase(A.begin());
9         else
10            C.push_back(B[0]), B.erase(B.begin());
11     while(A.size())
12         C.push_back(A[0]), A.erase(A.begin());
13     while(B.size())
14         C.push_back(B[0]), B.erase(B.begin());
15     return C;
16 }
```

现给定 a_1, a_2, \dots, a_m , 问有多少种可能的初始数组 A_1, A_2, \dots, A_n , 对 998244353 取模。
对于 100% 的数据, 保证 $2 \leq n \leq 2 \times 10^5$, $n \leq m \leq 5 \times 10^5$, $1 \leq a_i \leq m$ 。

Solution: 【UR #26】石子合并

容易发现 merge 操作实际上就是一个归并, 因此若 a 单调不降, 可以直接 dp, 因为不能有空的 A_i , 因此还要容斥一下, 具体来说: 设 f_i 表示 $n = i$ 且 A_x 可以为空时的方案数, g_i 表示 $n = i$ 且 A_x 不能为空的方案数, 答案即 g_n 并且由二项式反演有

$$g_n = \sum_{i=0}^n (-1)^{n-i} \binom{n}{i} f_i$$

问题化为求 f , dp, 设 c_i 表示有多少个 j 满足 $a_j = i$, 有

$$f_n = \prod_{i=1}^m \binom{c_i + n - 1}{n - 1}$$

后面的组合数是插板法算出来的, 组合意义是从小到大考虑所有 a (相同的一起考虑), 然后将这若干个相同的数分到 n 个盒子中, 每个盒子可以放任意自然数个。

暴力计算的时间复杂度是 $\mathcal{O}(nm)$, 考虑优化, 设 t_i 表示有多少 j 满足 $c_j = i$, 有新的公式

$$f_n = \prod_{i=1}^m \binom{i + n - 1}{n - 1}^{t_i}$$

显然我们只需要考虑那些 $t_i > 0$ 的 i , 而一共有多少个大于 0 的 t_i 呢? 由于 $\sum c_i \leq m$, 即 $\sum_{i=1}^m i t_i \leq m$, 因此大于 0 的 t_i 只会是 $\mathcal{O}(\sqrt{m})$ 个, 暴力枚举这些 t_i 算 f , 时间复杂度 $\mathcal{O}(n\sqrt{m})$ 。

可以发现我们的复杂度好像没有算上快速幂, 实际上不是这样的, 比较严格的时间复杂度其实是 $\mathcal{O}\left(n \sum_{t_k \geq 1} \log t_k\right)$,

而

$$\sum_{t_k \geq 1} (1 + \lfloor \log_2 t_k \rfloor) = \sum_{p \geq 0} \sum_k [t_k \geq 2^p] = \sum_{p \geq 0} \sum_k [\lfloor \frac{t_k}{2^p} \rfloor > 0]$$

我们还知道 $\sum_k k \lfloor \frac{t_k}{2^p} \rfloor \leq \frac{m}{2^p}$, 因此

$$\mathcal{O} \left(\sum_{t_k \geq 1} \log t_k \right) = \mathcal{O} \left(\sum_{p \geq 0} \sqrt{\frac{m}{2^p}} \right) = \mathcal{O}(\sqrt{m})$$

因此总时间复杂度为 $\mathcal{O}(n\sqrt{m})$.

官方题解中还有更为优秀的 $\mathcal{O}(m \log \text{MOD})$ 算法, 这里不作展开.

CF508D. Tanya and Password

给出长度为 $n+2$ 的字符串 s 的全部 n 个「长度为 3 的子串」, 求原串. 若有解则输出 YES 与任意一个合法解, 反之输出 NO.

字符集为大小写字母与数字, $1 \leq n \leq 2 \times 10^5$.

Solution: CF508D. Tanya and Password

对于一个长度为 3 的子串 abc , 从点 ab 向点 bc 连一条边, 在该图上跑欧拉路即可, 时间复杂度 $\mathcal{O}(n)$.

CF348C. Subset Sums

给定一个 n 个数的序列 a , m 个下标集合, 记 $S_k = \{S_{k,i}\}$.

有两种操作:

- ? k 求集合 k 的和, 即求集合 k 所有元素做原数组下标对应值的和.
- + k w 给集合 k 的所有下标代表的数加 w .

设所有集合的元素个数和为 c , 对于 100% 的数据, 保证 $1 \leq n, m, q \leq 10^5$, $c \leq 10^5$.

Solution: CF348C. Subset Sums

下面的分析中设 n, m, q, c 同阶.

根号分治, 称势大于 \sqrt{c} 的集合为重集合, 其他为轻集合, 显然重集合的个数不超过 \sqrt{c} .

$\mathcal{O}(n\sqrt{n})$ 预处理出每个重集合和其他所有集合有多少相同元素, 预处理出每个重集合的答案.

对于加操作, 若操作的是轻集合, 枚举重集合后 $\mathcal{O}(1)$ 更新其维护的答案, 时间复杂度 $\mathcal{O}(\sqrt{n})$, 然后暴力更新 a 数组, 由于是轻集合, 所以时间复杂度也是 $\mathcal{O}(\sqrt{n})$; 若操作的是重集合, 同样 $\mathcal{O}(\sqrt{n})$ 维护其他重集合的答案, 然后在该重集合上打一个 $+w$ 的 tag (显然 tag 可以合并, 维护的是该重集合中的每个元素被加过多少).

对于查询操作, 若查询的是重集合, 直接输出我们维护的答案, $\mathcal{O}(1)$; 若查询的是轻集合, 先 $\mathcal{O}(\sqrt{n})$ 求出对应下标的 a 的和, 然后枚举所有重集合, 考虑重集合的加操作对该轻集合和的贡献, 用 tag 直接 $\mathcal{O}(1)$ 算即可, 时间复杂度 $\mathcal{O}(\sqrt{n})$.

总时间复杂度 $\mathcal{O}(n\sqrt{n})$.

【NOI Online 2022 提高组】丹钧战

有 n 个二元组 (a_i, b_i) ，编号为 1 到 n 。

有一个初始为空的栈 S ，向其中加入元素 (a_i, b_i) 时，先不断弹出栈顶元素直至栈空或栈顶元素 (a_j, b_j) 满足 $a_i \neq a_j$ 且 $b_i < b_j$ ，然后再将其加入栈中。

如果一个二元组入栈后栈内只有这一个元素，则称该二元组是“成功的”。

有 q 个询问 $[l_i, r_i]$ ，表示若将编号在 $[l_i, r_i]$ 中的二元组按编号从小到大依次入栈，会有多少个二元组是“成功的”。

询问之间相互独立。对于 100% 的数据，保证 $1 \leq n, q \leq 5 \times 10^5$ 。

Solution: 【NOI Online 2022 提高组】丹钧战

本题有很多做法，这里讲单调栈 + 倍增。

对每一个二元组，预处理出以它为起点做栈操作时，第一个能把它弹出来的二元组的位置记为 R_i （这可以从右到左单调栈 $\mathcal{O}(n)$ 做），之后，询问 $[l, r]$ 等价于求 $l, R_l, R_{R_l}, R_{R_{R_l}} \dots$ 中有多少个位置不超过 r ，倍增预处理然后做，总时间复杂度 $\mathcal{O}((n+q)\log n)$ 。

知らない

小 X 得到了 $1, 2, \dots, n$ 的排列 p ，她想要找到它的一个 LIS 和一个 LDS，使得它们没有相同元素。如果有解，他希望你给他一个方案。

形式化的说，给你一个排列 p_1, p_2, \dots, p_n ，求是否有一个 $a_1 < a_2 < \dots < a_k$ 和一个 $b_1 < b_2 < \dots < b_l$ ，其中 k 是最长上升子序列长度， l 是最长下降子序列长度，使得 $p_{a_1} < p_{a_2} < \dots < p_{a_k}, p_{b_1} > p_{b_2} > \dots > p_{b_l}$ ，并且所有 a_i 以及 b_j 互不相同，输出方案或判断无解。

对于 100% 的数据，保证 $n \leq 5 \times 10^5$ 。

Solution: 知らない

由于上升子序列递增，下降子序列递减，于是将 (i, p_i) 画在坐标系上，连上 LIS 和 LDS，它们之间一定相交且仅相交一次。

为什么一定会相交？若不相交，不妨设 LIS 在 LDS 左侧，那么为了让 LIS 最长，LDS 中元素必须小于 LIS 的末尾，但是此时 LDS 一定不对：可以将 LIS 的元素再添加到原有 LDS 的开头得到一个更长的 DS。

因此，题目限制等价于要求交点不落在某个 (i, p_i) 处，也就是它们相交但刚好在线段处。

不妨设 LIS 上两点 $(a, p_a), (b, p_b)$ 构成的线段 $(a < b)$ 和 LDS 上两点 $(c, p_c), (d, p_d)$ 构成的线段 $(c < d)$ 相交，要求 a, b, c, d 互不相同，假如 a 接上前面的以 (a, p_a) 结尾的前缀 LIS 拼上 b 后面以 b 为开头的 LIS 就是序列的一个 LIS，并且 c 接上前缀 LDS 再拼上 d 后缀 LDS 也是原序列的 LDS，那么我们就找到了一个方案。

得到了一个四次方算法就是预处理前后缀 LIS，LDS 和转移点，暴力枚举 a, b, c, d 判断，接下来考虑优化。

假如这两个线段要有交，首先 $[a, b]$ 和 $[c, d]$ 要有交，也就是 $\max(a, c) < \min(b, d)$ ，其次要保证 $[p_a, p_b]$ 和 $[p_d, p_c]$ 要有交，也就是 $p_a < p_c, p_b > p_d$ 。

为了方便，下面直接钦定 $p_c < p_b$ 寻找这样的 a, b, c, d 对，至于 $p_c > p_b$ 的情况，我们只需要将整个排列反转过来，IS 变为 DS，DS 变为 IS 再做一遍。

此时还有没有别的限制可以加上来？ $a \in (c, d)$ ，这是因为若 $a < c$ ，那么 a, c, b 就是一个比 a, b 更长的 IS 而 c 已经被 DS 选走了，所以此时一定拼不出 LIS。

类似的， p_a 必须小于 p_d ，否则 c, a, d 构成了一个更长的 DS，进一步又有 $d > b$ ，否则 a, d, b 构成了更长的 IS。

整理一下，需要找到这样的 a, b, c, d 四元组：

$$\begin{cases} c < a < b < d \\ p_a < p_d < p_c < p_b \end{cases}$$

(上面的条件可以自己纸上整合一下, 不难得到)。

如果对着这个条件强行做, 会发现根本做不了或者很难做, 尝试变一下: 首先如果确定了某个 (a, b) 和 c , 应该选和这个 c 配对的哪个 d ? 看起来不好做, 因为如果选最靠右的 d , 那么 p_d 说不定会小于 p_a , 而如果选最靠左的 d , 说不定 d 会小于 b 。

实际上根本不需要担心 $p_a < p_d$ 这个限制, 因为假如 d 被这个限制住, 那么 c, a, d 比 c, d 优, 那么 c 的前缀 LDS 和 d 的后缀 LDS 一定拼不出来 LDS, 而配对的时候显然应该只考虑能拼出来的数对。

因此应该对每个 i 找出来它作为 c 时能拼上的最靠右的 d 。

(a, b) 对怎么找? 类似的, 我们根本不需要考虑 $p_b > p_c$ 这个限制, 为了让 $b < d, p_b > p_d$ 这两个限制尽可能成立, b 应该尽量靠左 (此时 b 最小且 p_b 最大), 也就是对每个 i 找到它作为 a 时最靠左的 b 。

这样的 (a, b) 对共有 $\mathcal{O}(n)$ 个, (c, d) 对共有 $\mathcal{O}(n)$ 个, 暴力枚举然后判断就可以做到 $\mathcal{O}(n^2)$ 了, 但是这个形式已经很容易接着优化:

从左到右枚举 (a, b) 对, 只需要找到一个 (c, p_c) 满足 $c < a, p_c \in (p_a, p_b)$ 中的, 对应的 d 最大的 c , 就是单点修改, 区间 max, 线段树优化即可。

时间复杂度 $\mathcal{O}(n \log n)$ 。

物語

树上有 N 个有标号的点 (标号为 1 到 N), 每个点都有一个 0 到 $N-1$ 的权值, 并且任意两个节点的权值互不相同。

一些路径十分的优美。一个长度为 L (L 个点) 的路径是**好路径**当且仅当对于任意权值为 $0 \leq i < L$ 的点都在这一条路径上 ($u \rightarrow v$ 和 $v \rightarrow u$ 被认为是同一条路径)。

同时, 邪恶的出题人会对这个树做 M 次操作, 每次操作形如选择标号为 x, y 的点, 并交换它们的权值。

他现在想要知道每次操作之后**好路径**的个数, 然而由于小 P 太菜了, 他并不会算这个东西。你能帮帮他吗?

对于 100% 的数据, 保证 $1 \leq N, M \leq 3 \times 10^5$ 。

Solution: 物語

先看链怎么做: 这是一个经典 **trick**, 名字叫“点减边”, 具体来说, 按照 p 从小到达加入每个点, 问题等价于有多少时刻图恰好联通 (某条边被加入图中当且仅当两个端点均在图中)。

维护每个时刻的点减边数量, 图联通等价于点减边等于 1, i 时刻的设为 C_i , 初始为 $C_i = i$ (该时刻点数), 之后考虑边, 每个边会对一个后缀的时刻造成 -1 的贡献, 具体来说是让 $C_{\max(p_u, p_v) \dots n}$ 减一, 其中 u, v 为该边的两个端点。

交换两个 p_i 只会对 $\mathcal{O}(1)$ 条边的贡献区间产生影响, 因此可以用线段树维护 C , 实现区间加法, 区间求 min. 为什么是区间求 min 而不是求 1 的个数? 因为点减边至少为 1, 求出最小值和最小值个数是线段树可以做到的, 而区间加法、全局某个值的个数是困难的。

于是就可以 $\mathcal{O}((n+q) \log n)$ 做到链的情况了。

树的情况有些不同, 但思路是可以借鉴的: 同样维护每个时刻的点减边个数. 先随便定一个根, 不妨设为 1 号点, 将两端点互为祖先后代的好路径和存在一个不同于它们 lca 的好路径分为两类分别求个数。

先考虑前者, 最开始还是 $C_i = i$, 考虑边对 C 的贡献: 对于每一个父亲而言, 它和它儿子的边中只有 p 最小的那个儿子是有用的, 因为对于每个点, 它会让点减边加一当且仅当其儿子中权值没有在该时刻之前的, 用

一个 `std::set` 维护每个点的所有儿子的权值，初始化的时候对每个点维护它对点减边的贡献，交换两个权值也只会影响 $\mathcal{O}(1)$ 个点的贡献，因此两 endpoint 互为祖先后代的好路径个数就是 $C_i = 1$ 的 i 的个数，线段树维护。

再考虑后者，首先我们可以容易求出每个结点儿子个数大于等于二的最小时刻，满足条件的好路径显然应该有一个儿子个数恰等于 2 的结点作为 lca，并且其父亲还没有被加进来，找到这些时刻构成的一段区间，查询这段区间中没有儿子的点的数量恰为 2 的时刻的个数，没有儿子的点的个数至少是 2，因此也是区间查询 min 及个数，而这就是 $C_i = 2$ 的数 i 的个数，直接在线段树上查询。

时间复杂度 $\mathcal{O}((n+q)\log n)$ 。

I got smoke

给定两个自然数 A, B ，每次可以选择如下操作之一：

- 将 A 变成 $A + 11 \bmod 998244353$ 。
- 将 A 变成 $A \times 21 \bmod 998244353$ 。
- 将 A 变成 $A^{31} \bmod 998244353$ 。

问至少需要多少次操作才能将 A 变成 B 。可以证明一定有解。

对于所有数据， $0 \leq A, B < 998244353$

Solution: I got smoke

直接搜不太妙，考虑 meet in the middle，但是我们要找到这三个操作的逆操作。

第一个操作的逆显然就是 $A \leftarrow A - 11$ ，第二个操作的逆就是乘 21 的逆元，第三个操作的逆是什么？

有 $x^{31} = y$ 而我们现在知道 y 想要求得 x ，如果可以找到一个数 t 使得 $x^{31 \times t} = x$ ，那么就有 $x = y^t$ 了，直接快速幂即可。

怎么找这个 x ？根据费马小， $31 \times t \equiv 1 \pmod{P-1}$ ，其中 $P = 998244353$ ，为模数，也就是要找到 31 在模 $P-1$ 意义下的逆元，这是容易的。

从 A, B 分别作为起点搜 12 层就差不多了。

为什么这样是对的？因为操作得到的数实际上很随机，生日悖论攻击。

【JLOI2015】骗我呢

说起来，毕业之后 B 君也就见过 R 君两面而已。

R 君有一个 $n \times m$ 的数组 $x_{i,j} (1 \leq i \leq n; 1 \leq j \leq m)$ 。

对于 $1 \leq i \leq n; 1 \leq j \leq m$ ，满足 $0 \leq x_{i,j} \leq m$ 。求可能的数组 $x_{i,j}$ 的解数。

B 君觉得限制太宽松，还要求对于 $1 \leq i \leq n; 1 \leq j < m$ ，满足 $x_{i,j} < x_{i,j+1}$ ，对于 $1 < i \leq n; 1 \leq j < m$ ，满足 $x_{i,j} < x_{i-1,j+1}$ 。

B 君认为 R 君可以直接 pwn 掉这个题。

R 君说：「黑的实在逼真 =.=，你起码把解数模 $10^9 + 7$ 吧。」B 君觉得 R 君说的有道理，于是想让你求解数模 $10^9 + 7$ 的结果。

对于 100% 的数据， $1 \leq m, n \leq 10^6$ 。

Solution: 【JLOI2015】骗我呢

那里没有括号

对于括号串 S , 记 $|S|$ 为 S 的长度, 记 $f(S)$ 为其 $2^{|S|}$ 个子序列中合法括号序列的个数。
给定括号串 S , 定义 $S[l, r]$ 为从 S 的第 l 个字符开始到第 r 个字符结尾的子串, 求

$$\left(\sum_{l=1}^{|S|} \sum_{r=l}^{|S|} (l \oplus r \oplus f(S[l, r])) \right) \bmod 998244353$$

其中 \oplus 是异或运算。即对所有 $1 \leq l \leq r \leq |S|$, 求出子串 $S[l, r]$ 中合法括号子序列的个数异或 l 异或 r 之后, 输出这些数的和对 998244353 取模后的结果。
对于 100% 的数据, 保证 $1 \leq |S| \leq 500$ 。

Solution: 那里没有括号

主要问题在于怎么 $\mathcal{O}(|S|^3)$ 对每个 $1 \leq l \leq r \leq |S|$ 算出来 $f(S[l, r])$, 设 $dp_{l,r}$ 表示 $f(S[l, r])$, 考虑转移。

对于一个合法的括号序列, 每个左(右)括号对应的另一半括号是唯一确定的, 并且第一个字符一定是左括号(, 根据类似区间 dp 的思路, 我们按照长度算 dp, 先算出来 $r - l + 1$ 比较小的 $dp_{l,r}$, 再根据它们转移出来长度比较大的区间的答案。初始值 $dp_{i,i} = 1$, 这是因为有空串, 其次, 对于 $i > j$, 规定 $dp_{i,j}$ 也为 1, 原因也是空串。

怎么转移? 也就是说怎么算出来 $dp_{l,r}$? 将所有合法的括号子列分为两类, 第一类是有 S_l 这个字符的, 第二类是没有 S_l 这个字符的, 第二类的方案数显然是 $dp_{l+1,r}$, 对于第一类, 仅当 $S_l = ($ 的时候可能存在, 枚举与 S_l 匹配的那个右括号的位置(从 $l+1$ 到 r), 不妨设为 k , 那么根据合法的括号序列的要求, 这两个匹配的括号之间也应该是一个合法的括号序列(或者是空串), 方案数是 $dp_{l+1,k-1}$, 其次 k 右边也应当是一个合法的括号序列, 方案数是 $dp_{k+1,r}$, 也就是说此时的状态是 $(A)B$, 其中 A, B 都是合法的括号序列, 左括号是 S_l , 右括号是 S_k 。

方程就是

$$dp_{l,r} = dp_{l+1,r} + \sum_{i=l+1}^r [S_l = (, S_i =)] dp_{l+1,i-1} \times dp_{i+1,r}$$

时间复杂度是 $\mathcal{O}(|S|^3)$ 的。

但是这样有一个问题, 比如 $dp_{l,r}$, 它可能很大 ($2^{|S|}$), 这个时候我们不能将它模 P 后再异或, 但是显然我们是存不下不模的结果的。

解决这个问题的方法是, 和 f 异或的是 l 和 r , 它们的二进制最高位并不大, 也就是说用它们异或 f , f 的比较高的二进制位是没有影响的, 相当于没有异或, 所以我们可以把 f 的比较低二进制(例如, int 范围内)留下来和 l, r 异或, 其他的二进制位对答案的贡献就是他本身模 P , 它们可以随时模 P , 将两类贡献分开算, 放到代码里的话, 当算 dp 的时候, 值超出了我们限定的用来异或的范围时, 将它的贡献分开即可。

还有一个算 $dp_{l,r}$ 的方法是, 将 $($ 看作 $+1$, $)$ 看作 -1 , 一个括号序列合法, 当且仅当每一个前缀的和都不小于 0 并且整个串的和为 0。

因此, 我们固定 l 算对应的 $dp_{l,r}$, 从左到右枚举 r , 记录一下当前的区间里能够取到和为 x 的子串有多少个(设为 $g(x)$), 转移就是考虑 S_r 不放进去(方案数不变), $S_r = +1$ 并且放进去就是让 $g(x)$ 变为 $g(x) + g(x-1)$, $S_r = -1$ 并且放进去就是让 $g(x)$ 变为 $g(x) + g(x+1)$, 同样是 $\mathcal{O}(|S|^3)$ 的。

【省选联考 2023】城市建造

在这个国度里面有 n 座城市，一开始城市之间修有若干条双向道路，导致这些城市形成了 $t \geq 2$ 个连通块，特别的，这些连通块之间两两大小差的绝对值不超过 $0 \leq k \leq 1$ 。为了方便城市建设与发展， n 座城市中的某 t 座城市在这 t 座城市之间额外修建了至少一条双向道路，使得所有城市连通。

现在已经知道额外修建后的所有道路，你需要算出有哪些双向道路集合 E' ，满足这些道路有可能是后来额外修建的，请输出答案对 998,244,353 取模的结果。

即给定一张 n 个点 m 条边的无向连通图 $G = (V, E)$ ，询问有多少该图的子图 $G' = (V', E')$ ，满足 $E' \neq \emptyset$ 且 $G - E'$ 中恰好有 $|V'|$ 个连通块，且任意两个连通块大小之差不超过 k ，保证 $0 \leq k \leq 1$ ，请输出答案对 998,244,353 取模的结果。

对于所有的数据，保证： $3 \leq n \leq 10^5$ ， $n - 1 \leq m \leq 2 \times 10^5$ ， $0 \leq k \leq 1$ 。

Solution: 【省选联考 2023】城市建造

其实问的就是：有多少种选出点集 S 的方案使得删除端点均在 S 中的边后， S 中的点两两不联通（此时联通块个数为 $|S|$ ）并且联通块大小的极差不超过 k 。

先不考虑极差的限制，看有多少种 S 。若 $u, v \in S$ ，那么对于任意一条从 u 到 v 的简单路径上的任意一点，它都必然在 S 中。假如某两个点在同一个点双中，要求删边后它们不联通，也就等价于该点双中的每个点都在 S 中（否则，这两个点会被没有被删去的点联通起来），因此我们得出结论：

$|S|$ 中的点必然构成一个联通块并且对于一个原图中的点双联通分量，要么它们全在 S 中，要么只有一个点在 S 中，要么都不在 S 中。

接下来考虑极差的问题，建出来圆方树并找出带权重心（方点的权为 0，圆点的权为 0），定义一个方点被选择表示该方点对应的点双被选择，那么只需满足：所选方点联通并且删掉所有被选择的方点之后，所有联通块权值的极差不超过 k 。

$k \in \{0, 1\}$ 应该有一些神秘性质：**带权重心必被选**。

若带权重心是方点并且它没有被选，那么删去被选方点后，它所在联通块的权值 $\geq \frac{n}{2} + 1$ ，还有一部分必然 $\leq \frac{n}{2} - 1$ ，两者的差为 $2 > k$ ，不符合题意！

若带权重心是圆点并且它没有被选，同样的，极差必然 $> k$ ，不符合题意。

若带权重心为方点，选相邻的任意一个圆点作为根，否则将带权重心作为根，将圆方树变为有根树，那么根据所选方点构成一个联通块，我们得知：若某个方点被选，那么其祖先方点也必定被选。

接下来就开始真正做这道题了，先看 $k = 0$ 。

设 siz_u 表示以 u 为根的子树权值和，枚举最终联通块的大小 x ，对于方点 u ，若 $siz_u > x$ ，那么 u 必被选，若 $siz_u < x$ ，那么 u 必不被选，若 $siz_u = x$ ，此时若 u 的儿子数量为 1， u 必被选，否则必不被选，读者可以自行验证。

于是可以从小到大枚举 x ，并查集维护联通性，同时开一个桶记录特定联通块权值的联通块数量，时间复杂度 $\mathcal{O}(n\alpha(n))$ 。

当 $k = 1$ 的时候，可以想到一个比较显然的容斥：联通块大小为 x 和 $x + 1$ 的方案数就是钦定联通块大小可能为 x 或 $x + 1$ ，再减去全为 x 或全为 $x + 1$ 的方案数。

主体还是一样的，若 $siz_u < x$ ，那么 u 必不选，若 $siz_u > x$ ，那么 u 必选，但当 $siz_u = x$ 的时候，若儿子数量 ≥ 2 ，必不选，否则特殊处理。

怎样特殊处理？由于联通块大小可以是 $x + 1$ ，所以其实可以将这个子树接到方点的父亲上，让这个方点不被选。显然，对于固定的父亲，只能有一个方儿子接上来。

先钦定所有需要特殊处理的方点被选，此时若某需要被特殊处理的点的父亲所在联通块大小恰为 1，就可以选择一个特殊方儿子接上来，否则它的特殊方儿子只能被选。

也就是说, 对于所有存在需要特殊处理的儿子、且当前所在连通块大小 = 1 的 fa , 给答案乘上 fa 需要特殊处理的方儿子数量。

但当 $x = 1$ 的时候, 乘的就是 fa 需要特殊处理的方儿子数量再加一了, 这是因为此时若 fa_{fa} 存在, 那么它必然被选, 此时 fa 的儿子可以都被选。

同样用并查集维护联通性, 时间复杂度 $\mathcal{O}(n\alpha(n))$, 据说实现精细可以把并查集去掉做到线性。

【IOI2024 集训队互测 Round 1】优惠购物

小 C 要购买 n 个物品, 这些物品有前置关系, 必须依次购买 (即在购买了第 i 个后才能购买第 $i+1$ 个)。

他初始有 m 张优惠券和无穷多个金币。每个物品有两个属性, 价格 a_i 和优惠券的使用上限 $b_i (0 \leq b_i \leq a_i)$ 。

购买一个物品的流程如下:

- 选择使用 $x (0 \leq x \leq b_i)$ 张优惠券, 付出 $a_i - x$ 个金币和 x 张优惠券。
- 购买完后可得到 $\lfloor \frac{a_i - x}{c} \rfloor$ 张优惠券 (即一次购买中, 每付出 c 个金币可以得到一张优惠券, c 为给定常数)

小 C 想求出最少花费多少个金币能购买全部物品。

多组数据, 对于每个测试点保证, $1 \leq \sum n \leq 10^6, 1 \leq m, a_i, b_i \leq 10^9, 2 \leq c \leq 10^9, 1 \leq \sum n \leq 10^6, 1 \leq m, a_i, b_i \leq 10^9, 2 \leq c \leq 10^9$ 。

Solution: 【IOI2024 集训队互测 Round 1】优惠购物

设计一个朴素的 dp: 设 $f_{i,j}$ 表示考虑完前 i 个物品, 新得到了 j 个优惠券 (不包括一开始就有的, 包括已经使用过的) 时, 花费的优惠券的最大数量, 记 A 为 a 的前缀和, 即 $A_k = \sum_{i=1}^k a_i$, 那么答案就是 $A_n - \max_{i=1}^{A_n} f_{n,i}$ 。

转移就是 $f_{i,j} + x \rightarrow f_{i+1, j + \lfloor \frac{a_{i+1} - x}{c} \rfloor}$, 其中箭头指取 \max , x 是枚举本次操作用多少张优惠券, 有 $x \in [0, \min(b_i, m + j - f_{i,j})]$ 。

朴素 dp 的时间复杂度是

$$\mathcal{O}\left(\sum_{i=1}^n A_{i-1} b_i\right) = \mathcal{O}\left(\left(\sum_{i=1}^n a_i\right)^2\right)$$

钦定全过程中不使用优惠券, 那么最后会多出若干张优惠券, 我们的目标是用尽可能多的优惠券。

什么时候才可以用优惠券呢? 设 s_i 表示买完前 i 个物品后剩下的优惠券数量, x_i 表示买第 i 个物品所用的优惠券数量, 那么有 $s_i = m + \sum_{j=1}^{i-1} (-x_j + \lfloor \frac{a_{j+1} - x_j}{c} \rfloor)$, 需要保证 $s_{i-1} - x_i \geq 0$ 。

设 $s_n = S$, 那么对一个物品使用优惠券, 可以分为三个阶段:

1. 用 $[0, a_i \bmod c]$ 张优惠券, 不改变 S 。
2. 每次用 c 张优惠券, 让 S 变为 $S-1$ 。
3. 用 $[0, c-1]$ 张优惠券并让 S 变为 $S-1$ 。

显然我们应该贪心地让阶段 1 尽可能多, 其次阶段 2, 最后阶段 3。

先看阶段 1, 从前往后扫即可, 若 $a_i - x_i$ 无法变成 c 的倍数, 那么在之后, x_i 一定也不会变化。

然后看阶段 2, 从后往前贪心, x_i 可以增加

$$c \times \min(\lfloor \frac{b_i - x_i}{c} \rfloor, \lfloor \frac{s_{i-1} - x_i}{c} \rfloor, \min_{j>i} \lfloor \frac{s_{j-1} - x_j}{c+1} \rfloor)$$

最后看阶段 3, 优先用 $c-1$ 张, 其次用 $c-2$ 张..., 可以 $\mathcal{O}(c)$ 枚举, 然后用类似阶段 2 的转移做。

总时间复杂度 $\mathcal{O}(nc)$, 瓶颈在阶段 3 的处理。

阶段 3 怎么优化呢? 设 $t_i = s_{i-1} - x_i$, $del_i = \min(b_i - x_i, t_i, \min_{j>i} (t_j - 1))$, 那么阶段 3 等价于每次取一个 del 最大的 i 并将 x_i 变为 $x_i + del_i$, 并做修改 $t_i \leftarrow t_i - del_i$, $\forall j > i, t_j \leftarrow t_j - del_i - 1$ 。

由于 $\min(t_i, \min_{j>i}(t_j - 1))$ 是单增的, $b_i - x_i$ 没有单调性, 所以我们可以先将所有 i 按照 $b_i - x_i$ 递减排列, 每次取出 $b - x$ 相同的一些下标, 设下一个更小的 $b - x$ 是 lim , 那么我们想要操作所有 $del > lim$ 的下标。

显然取下标更靠右的会更优 (因为 $\min(t_i, \min_{j>i}(t_j - 1))$ 递增), 因此可以用一个大根堆维护当前需要考虑的下标, 线段树维护 t_i , 若堆顶满足 $del > lim$ 的限制, 就修改其 t , 弹出, 否则停止检查当前堆。

线段树只需要维护区间加、区间查询 \min , 时间复杂度 $\mathcal{O}(n \log n)$ 。

【IOI2024 集训队互测 Round 2】序列

称一个序列 (a_1, a_2, \dots, a_n) 是避免 120 模式的, 当且仅当不存在 $1 \leq i < j < k \leq n$ 使得 $a_k < a_i < a_j$ 。

给定质数 P 。 q 次询问, 每次给定 n, m , 求有多少个长度为 n 的、值域在 $[0, m]$ 内的整数序列 a 是避免 120 模式的, 结果对 P 取模。

对于全部数据, 满足 $10^8 \leq P \leq 10^9, 1 \leq q \leq 8 \times 10^4, 1 \leq n \leq 1000 \leq m \leq 10^9$ 。

Solution: 【IOI2024 集训队互测 Round 2】序列

设 $dp_{i,j}$ 表示长度为 i 的序列, 里面恰有 j 个数字, 避免 120 模式的方案数, 转移枚举序列中最靠左的最大值, 设其下标为 p , 只需要满足下标为 $[1, p-1]$ 的数的最大值小于等于下标为 $[p+1, i]$ 的数的最大值, 同时左右两侧避免 120 模式即可, 再枚举右侧不同数字的数量 siz , 有转移

$$\begin{aligned} dp_{len, val} &\leftarrow dp_{p-1, val-siz} \times dp_{len-p, siz} \\ dp_{len, val} &\leftarrow dp_{p-1, val-siz-1} \times dp_{len-p, siz} \\ dp_{len, val} &\leftarrow dp_{p-1, val-siz+1} \times dp_{len-p, siz} \\ dp_{len, val} &\leftarrow dp_{p-1, val-siz} \times dp_{len-p, siz} \end{aligned}$$

其中, 第一行是指左侧最大值小于右侧最小值, 并且右侧最大值恰是整个序列的最大值, 这时候左侧不同数字的个数就是 $val - siz$; 第二行是指左侧最大值小于右侧最小值, 并且右侧最大值不是整个序列的最大值, 左侧不同数字的个数是 $val - siz - 1$ (因为还要去掉下标为 p 的那个全局唯一最大值); 第三、四行指左侧最大值等于右侧最小值, 区别在于第三行的右侧最大值恰是全局最大值, 第四行不是。

边界是 $dp_{0,0} = dp_{1,1} = 1$, 转移的时候还要加一些细节的判断, 预处理出所有 $dp_{1..n, 1..n}$, 时间复杂度 $\mathcal{O}(n^4)$ 。

对于一个 n, m 的询问, 答案即为

$$\sum_{i=1}^n dp_{n,i} \times \binom{m+1}{i}$$

二项式系数的上指标是 $m+1$ 而不是 m 是由于 $[0, m]$ 有 $m+1$ 个数。

枚举 i , 每次根据 $\binom{m+1}{i-1} \mathcal{O}(1)$ 得出 $\binom{m+1}{i}$ 即可做到单次询问 $\mathcal{O}(n)$, 总时间复杂度 $\mathcal{O}(n^4 + qn)$ 。

【IOI2024 集训队互测 Round 2】没有创意的题目名称

给定一个正整数 n 和一个长度为 $n+1$ 的序列 $lim_0, lim_1, lim_2, \dots, lim_n$, 求长度为 $n+1$ 的整数序列 $f_0, f_1, f_2, \dots, f_n$ 的方案数, 使得以下条件成立。

- 对于所有 $0 \leq i \leq n$, 有 $0 \leq f_i \leq lim_i$ 。
- 对于任意 $0 \leq m \leq n$ 满足, 对于所有 $0 \leq i \leq m$, 有 $f_{f_i+f_{m-i}} = f_m$ 。(当 $i > n$ 时, $f_i = -1$)

答案对 998244353 取模。

对于 100% 的数据, 有 $1 \leq n \leq 2000, 0 \leq lim_i \leq n$ 。

Solution: 【IOI2024 集训队互测 Round 2】没有创意的题目名称

定义等价关系 \div 满足 $i \div j \iff f_i = f_j$, 那么题目要求等价于

- $0 \leq f_i \leq \lim_i$
- $\forall 0 \leq i+j \leq n$, 有 $f_i + f_j \leq n$
- $f_i + f_j \div i+j$

显然 \div 具有传递性, 也就是说 $f_i = f_j, f_j = f_k \implies f_i = f_k$, 即 $i \div j, j \div k \implies i \div k$.

同时, \div 还可以同加, 即若 $i_1 \div i_2, j_1 \div j_2$, 那么 $i_1 + j_1 \div i_2 + j_2$.

证明: $f_{i_1+j_1} = f_{f_{i_1}+f_{j_1}} = f_{f_{i_2}+f_{j_2}} = f_{i_2+j_2}$.

引理 1: 若 $f_i = f_j$, 那么 $f_{i+1} = f_{j+1}$.

证明: $f_{i+1} = f_{f_i+f_1} = f_{f_j+f_1} = f_{j+1}$.

引理 1 告诉我们, f_i 要么两两不同, 要么存在循环节, 虽然可能不完整.

分类讨论, 先考虑 $f_0 = 0$ 的情况, 这时 $f_i = f_{i+0} = f_{f_i+f_0} = f_{f_i}$, 也就是说 $i \div f_i$.

第一种情况: f_i 两两不同, 此时必然有 $f_i = i$, 检验一下是否满足 $f_i \leq \lim_i$ 即可.

第二种情况: $\exists p < q$, s.t. $f_p = f_q$, 找到最小的 (p, q) 对, 那么我们就找到了一个长度为 $q-p$ 的循环节 $[p, q-1]$, 并设 $k = q-p$, 并且要求 $f_{1..q-1}$ 两两不同, 根据这个我们又可以得到 $\forall 0 \leq i < p, f_i = i$ (这是因为不存在 $j \neq i$ 使得 $i \div j$).

此时对于 $i \in [p, q-1]$, 有 $f_i \geq i$ 并且 $k \mid f_i - i$, 更进一步, 此时若第二个条件成立, 那么第三个条件必然成立.

证明: 当 $i+j < p$, $f_{i+j} = f_{f_i+f_j}$, 当 $i+j \geq p$, $f_{f_i+f_j} = f_{i+t_1k+j+t_2k} = f_{i+j}$, 证毕.

此时我们只需要考虑条件二, 进一步分类讨论, 当 $q-1 \leq \lfloor \frac{n}{2} \rfloor$, 对任意 $i \in [1, q-1]$ 都有 $f_i + f_i \leq n$, 也就是 $f_i \leq \lfloor \frac{n}{2} \rfloor$, 此时条件二恒成立.

否则, 对于 $i \in [p, \lfloor \frac{n}{2} \rfloor]$ 都有 $f_i \leq \lfloor \frac{n}{2} \rfloor$ 而 $i+k \geq q > \lfloor \frac{n}{2} \rfloor$, 因此也有 $f_i = i$, 至于 $i \in (\lfloor \frac{n}{2} \rfloor, q)$, 有 $f_i + f_{n-i} \leq n$ 而 $f_{n-i} = n-i$, 于是 $f_i \leq i$, 又由于 $f_i \geq i$, 因此也有 $f_i = i$.

综上, 当 $f_0 = 0$, 可以枚举 p, q , $\forall i \in [1, p), f_i = i$; 当 $q-1 \leq \lfloor \frac{n}{2} \rfloor$ 有 $\forall i \in [p, q-1], f_i = i+t(q-p)$ 且 $f_i \leq \lfloor \frac{n}{2} \rfloor$, 当 $q-1 > \lfloor \frac{n}{2} \rfloor$ 有 $\forall i \in [p, q-1], f_i = i$.

否则, 有 $f_0 \neq 0$, 此时 $f_{f_0+f_0} = f_{0+0} = f_0$, 这意味着存在一个 $i > 0$ 使得 $f_i = f_0$, 也就是循环节从 0 开始, 即 $p=0, q=k$.

$f_{2f_0} = f_0$ 意味着 $k \mid 2f_0$, 分类讨论: 当 $k \mid f_0$ 的时候有 $f_0 \geq k$ 于是 $2k \leq 2f_0 \leq n$, 因此 $k \leq \lfloor \frac{n}{2} \rfloor$, 于是此时条件三等价于 $f_i \equiv i \pmod{k}$, 于是可以先枚举 k , 方案数是一段连乘.

当 $k \nmid f_0$, 有 $f_0 \equiv \frac{k}{2} \pmod{k}$, 有 $f_i \equiv i + \frac{k}{2} \pmod{k}$, 带入 $i = \frac{k}{2} - 1$ 有 $f_{\frac{k}{2}-1} \equiv k-1 \pmod{k}$, 因此 $f_{\frac{k}{2}-1} \geq k-1$, 考虑条件二, $f_{\frac{k}{2}-1} + f_{\frac{k}{2}-1} \leq n$ 因此 $k-1 \leq \lfloor \frac{n}{2} \rfloor$, 因此对任意 i 都有 $f_i \leq \lfloor \frac{n}{2} \rfloor$, 于是就一样了, 枚举 k 将连乘贡献到答案.

总时间复杂度 $\mathcal{O}(n^2)$.

FBCHEF: Final Battle of Chef^a

给出一棵以 1 号点为根的有根树, 初始每个点有一个值, 要求支持以下两种操作:

1. 给定点 x 和值 v , 对每个点 u , 让 u 的权值减少 $\lfloor \frac{v}{2^{\text{dist}(u,x)}} \rfloor$, 其中 $\text{dist}(u, v)$ 表示 u, v 两点间的边数。
2. 给定点 u , 询问以 u 为根的子树中有多少点的权值 ≤ 0 。

设操作数量为 q , 点数为 n 。

对于 100% 的数据, 保证 $1 \leq n, q \leq 10^5$, 每个点的初始权值不超过 10^9 , 每次操作给出的 v 不超过 10^5 。

^a<http://www.codechef.com/APRIL14/problems/FBCHEF>

Solution: FBCHEF: Final Battle of Chef²

bfs 序可以很好的处理操作 1 (每次操作变为让 $\mathcal{O}(\log^2 v)$ 段区间减少某个正数), dfs 序可以很好的处理操作 2 (将每个数变为非负的时间确定下来, 查询即为子树代表的一段 dfs 序中有多少个点符合要求), 考虑将两者结合.

先将询问离线, 将 $\mathcal{O}(q \log^2 v)$ 段修改处理出来, 按照 bfs 序从左到右扫描线, 树状数组维护对于当前位置来说, 每个时刻的修改会让它的值减少多少, 再套一层二分就可以得出当前位置开始 ≤ 0 的时间, 存起来.

然后我们就可以处理操作 2 的询问了, 按照询问时间顺序扫, 用一个树状数组维护 dfs 序, 一个点变 ≤ 0 就让它权值为 1, 否则权值为 0, 每次将新的变化的点再树状数组上维护一下, 询问就是树状数组的区间和, 时间复杂度是三只 log.

假如想要在线, 可以将 $\mathcal{O}(q \log^2 v)$ 个修改用维护区间最小值的线段树维护, 某个点的权值从正数变到 ≤ 0 就暴力修改, 每个点只会被修一次所以复杂度是对的.

同时这道题处理每个点变化时间的扫描线 + 二分可以换成整体二分, 精细的实现可以做到两只 log.

圆

求有多少长度为 n 的排列可以被划分为不超过三个上升子序列, 答案对大质数取模。
对于 100% 的数据, 保证 $1 \leq n \leq 500$ 。

Solution: 圆

根据 Dilworth 定理, 要求等价于排列的最长不升子序列长度不超过 3, 因为没有相等元素, 所以就是排列的最长下降子序列长度不超过 3.

从 1 到 n 不断往原有排列中添加新元素, 假如某三个位置已经构成了一个长度为 3 的下降子序列, 那么就肯定不能加到它们最左侧的位置之前.

尝试用 dp 满足这个约束关系, 将整个序列分为三个连续相邻段, 最左边的一段是不能插入元素的位置, 最右边的一段是一段极长连续上升子序列, 设 $dp_{i,j,k}$ 表示已经加入了 $1 \sim i$, 中间一段的长度为 j , 最右边一段的长度为 k 的方案数, 转移时枚举最大的新元素在中间一段还是右边一段, 新的状态表示不难得到.

暴力转移是 $\mathcal{O}(n^4)$ 的, 简单优化一下就可以 $\mathcal{O}(n^3)$.

Bloodline Counter^a

给定 n 和 k , 问有多少个数为 n 的竞赛图满足最大环的点数恰为 k 。
对于 100% 的数据, 保证 $3 \leq k \leq n \leq 5 \times 10^5$ 。

^a<https://ac.nowcoder.com/acm/contest/57362>

Solution: Bloodline Counter³

引理: 最大环的点数恰为 k 等价于最大强联通分量点数为 k .

证明: 竞赛图的每个强联通分量都可以找到一个经过所有该强联通分量中点的环, 反之同理.

设 g_k 表示 k 个点的竞赛图强联通的方案数, 那么 n 个点的竞赛图方案数就是

$$2^{\binom{n}{2}} = \sum_{i=1}^n \binom{n}{i} g_i 2^{\binom{n-i}{2}}$$

左侧是直接考虑每条边的方向, 右侧是枚举缩点后链的最末端的强联通分量点数 i .

²<http://www.codechef.com/APRIL14/problems/FBCHEF>

³<https://ac.nowcoder.com/acm/contest/57362>

将右侧组合数拆开并将 $n!$ 移到左边可以得到

$$\frac{2^{\binom{n}{2}}}{n!} = \sum_{i=1}^n \frac{g_i}{i!} \frac{2^{\binom{n-i}{2}}}{(n-i)!}$$

上式告诉我们 $2^{\binom{n}{2}}$ (设为 f_n) 的 EGF F 和 g 的 EGF G 有

$$F = F * G + 1$$

加一是因为要补上 G 没有的常数项, 于是有 $G = 1 - \frac{1}{F}$.

最大环上点数至多是 k 的方案数就是

$$n![x^n] \sum_{j=0}^{+\infty} \left(\sum_{i=1}^k \frac{g_i}{i!} x^i \right)^j$$

等比数列公式套上去就是

$$n![x^n] \frac{1}{1 - \sum_{i=1}^k \frac{g_i}{i!} x^i} = n![x^n] \frac{1}{\frac{1}{F}}$$

于是对 $\frac{2^{\binom{n}{2}}}{n!}$ 求两次逆元 (第一次模 x^{k+1} , 第二次模 x^{n+1}) 即可算出最大环点数至多为 k 的方案数, 同理求出至多为 $k-1$ 的方案数, 两数相减即为答案, 时间复杂度 $\mathcal{O}(n \log n)$.

Permutation and Primes^a

给定 n , 构造一个长度为 n 的排列满足对于任意相邻两数, 它们的和为奇素数或它们的差为奇素数。

对于 100% 的数据, 保证 $1 \leq n \leq 10^6$ 。

^a<https://ac.nowcoder.com/acm/contest/57362>

Solution: Permutation and Primes⁴

对于长度小于 8 的排列可以直接构造: 若不超过 4 就直接倒序, 否则将大于 4 的部分倒序后拼上 2 1 4 3。

若长度不小于 8, 考虑不断将最大的 8 个数构造出来, 即 $n-3 \ n-6 \ n-1 \ n-4 \ n-7 \ n-2 \ n-5$, 注意到它同时也是可以往后拼 $n-8$ 的, 于是这样不断连起来就可以将长度变为小于 8 了, 这时候再往后拼上之前构造好的方案即可, 时间复杂度 $\mathcal{O}(n)$, 本题还有其它构造方法, 这里不再给出, 链接中有详细说明。

Scheming Furry^a

1 到 $n \times m$ 这 $n \times m$ 个正整数分布在 $n \times m$ 的矩阵中, Alice 每次可以交换两个不同行, Bob 每次可以交换两个不同列, 两人交替操作, Alice 先手, 每个人的目标都是在自己操作后矩阵有序, 并且有序后立即停止, 最后一次操作的人胜利 (但是至少操作一次), 给出矩阵, 询问谁会胜利, 或者游戏永远不会结束。

$n \times m$ 的矩阵有序, 指的是第 i 行第 j 列上的元素恰为 $(i-1) \times m + j$ 。

对于 100% 的数据, 保证 $2 \leq n, m \leq 200$ 。

^a<https://ac.nowcoder.com/acm/contest/57362>

Solution: Scheming Furry⁵

⁴<https://ac.nowcoder.com/acm/contest/57362>

⁵<https://ac.nowcoder.com/acm/contest/57362>

若 Alice 可以一次让矩阵有序，直接胜利，特判掉无论如何都变不成有序矩阵的情况。

若 $n \geq 3$ 并且 $m \geq 3$ ，那么谁也不可能胜利，因为另一方可以捣乱。

若可能变得有序，则这个矩阵可以看作是一个 n 排列和一个 m 排列，Alice 每次交换 n 排列的两个元素，Bob 每次交换 m 排列的两个元素，由于每次交换，排列的逆序对奇偶性一定改变，因此：

若 $n = 2, m = 2$ ，那么当两者奇偶性相等，Bob 胜利，否则 Alice 胜利；否则若 $n = 2$ ，那么当两者奇偶性相等，Bob 胜利，否则平局；若 $m = 2$ ，那么当两者奇偶性质不相等，Alice 胜利，否则平局。

硬币

给定 n ，对每个 $1 \leq x \leq n$ 求 $x^2 + 1$ 的最小质因子。
对于 100% 的数据，保证 $1 \leq n \leq 10^6$ 。

Solution: 硬币

把这些 $x^2 + 1$ 写出来，变成 2, 5, 10, 17, 26, 37...，设为 A_i 。先看第一项 2，找到所有 $2 \mid x^2 + 1$ ，显然这样的 $x^2 + 1$ 应该形如 $(1 + k \times 2)^2 + 1$ ，也就是 $x = 1 + k \times 2$ ，将这些 x 的答案用 2 更新并除掉这个因子，数列变为 1, 5, 5, 17, 13, 37...

接着看第二项 5，除掉所有 5 的倍数，一定形如 $(2 + k \times 5)^2 + 1$ ，变为 1, 1, 1, 17, 13, 37... 现在第三项已经是 1 了，跳过去看第四项，将 $(4 + k \times 17)^2 + 1$ 更新掉，一直这么进行下去，时间复杂度是 $\mathcal{O}(n \log \log n)$ 。

要说明这样是对的，只需要证明每次遇到的数不是质数就是 1，这也是容易证明的：若当前数的最小质因子是 p ，那么 $x^2 + 1$ 应该会在 $(x - p + k \times p)^2 + 1$ 也就是遇到第 $x - p$ 项的时候被筛掉，如果 $x - p$ 在更之前被筛掉了 p ，那么那次筛也一定会把 $x^2 + 1$ 除掉 p 。

阿克

强强的 xyr2005 阿克完 Codeforces 打开 QQ，发现有 n 个妹子给他发了消息，因为 xyr2005 又强又可爱，所以他一旦回复一个妹子就会和她聊一段时间，且这个时间区间 $[l_i, r_i]$ 对于一个妹子是固定的。

因为为人做事需要有始有终，所以 xyr2005 一旦在一个时刻和妹子 i 聊天，那么他在 $[l_i, r_i]$ 这段时间内都必须和妹子 i 聊天。

因为和妹子聊天需要集中精力，所以 xyr2005 同时只能和一个妹子聊天。

因为和妹子聊天非常舒服，所以 xyr2005 希望和每一个妹子聊天。但显然这非常困难，所以他可以发动一个技能：

- 时光倒流，假设当前时刻为 t ，花费 w 的体力回到时刻 $t - w$ 。

因为 xyr2005 刚刚阿克完 Codeforces，所以当前时刻为 T ，保证 $r_i \leq T$ 。因为和妹子聊天需要体力，所以他希望花费最少的体力在使时间倒流上。

现在你作为 xyr2005 的工具人，你需要帮他算出最少的体力花费，并构造出方案。

输入的第一行为 n, T ，之后 n 行给定区间 $[l_i, r_i]$ ，输出的第一行输出最小体力，第二行输出一个排列表示依次和哪个妹子聊天。

对于 100% 的数据，保证 $1 \leq n \leq 5 \times 10^5$ ， $l_i \leq r_i$ ， $0 \leq l_i, r_i, T \leq 10^9$ 。

Solution: 阿克

以下这组数据可以卡掉一些错误的贪心：

```
1 input
2 5 10
3 0 10
4 2 5
```

```

5 6 9
6 1 8
7 5 7

```

```

1 output
2 25
3 2 5 3 4 1

```

若所有区间联通，那么答案的一个下界是 $(T - \max r_i) + \sum_{i=1}^n r_i - l_i$ ，否则按照联通块从左到右，处理完一个联通块后处理下一个，不影响答案。

考虑构造得到这个下界：将所有线段按照左端点升序，那么合并之前的线段 $[l_1, r_1]$ 和当前线段 $[l_2, r_2]$ 只有三种情况：

1. $l_1 \leq l_2 \leq r_1 < r_2$ ，此时我们应该先倒流到 l_1 走 $[l_1, r_1]$ ，再倒流到 l_2 走 $[l_2, r_2]$ 。
2. $l_1 \leq l_2 \leq r_2 \leq r_1$ ，此时我们应该先倒流到 l_2 走 $[l_2, r_2]$ ，再走 $[l_1, r_1]$ 。
3. $l_1 \leq r_1 < l_2 \leq r_2$ ，先走 $[l_1, r_1]$ 再走 l_2, r_2 。

根据这样的构造，维护一个双端队列即可得到下界，时间复杂度 $\mathcal{O}(n \log n)$ ，瓶颈在于排序，代码如下：

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int N = 5e5 + 10;
6 int n, T;
7 vector<pair<pair<int, int>, int>> vec;
8
9 int main() {
10     ios::sync_with_stdio(false), cin.tie(0);
11
12     cin >> n >> T;
13     for(int i = 1; i <= n; i++) {
14         int l, r; cin >> l >> r;
15         vec.push_back({{l, r}, i});
16     }
17     sort(vec.begin(), vec.end());
18
19     long long res = T;
20     int p = vec[0].first.first;
21     deque<int> Q;
22     for(auto ppr : vec) {
23         int l = ppr.first.first, r = ppr.first.second;
24         res += r - l;
25         if(p < r) {
26             res += max(0, l - p), p = r;
27             Q.push_back(ppr.second);
28         } else Q.push_front(ppr.second);
29     }

```

```

30     res -= p;
31     cout << res << '\n';
32     while(!Q.empty()) {
33         cout << Q.front() << ' ';
34         Q.pop_front();
35     }
36     cout << '\n';
37
38     return 0;
39 }

```

全连

E.Space 喜欢打音游。

但是他技术不好，总是拿不到全连（Full Combo）。

现在他面前有一份乐谱，乐谱的其中一段有 n 个连续的单键音符。

相邻两个音符的到来时间均相等，我们可以认为第 i 个音符会在第 i 个时刻到来。

点击一个音符，E.Space 需要一段准备时间来进行移动手指之类的操作。由于音符的位置和周围情况不同，点击每个音符的准备时间也不同。在一个音符的准备时间内，E.Space 没法做到去点击其它音符，但是不同音符的准备时间范围可以互相重叠。

形式化地，令第 i 个音符的准备时间为 t_i 个单位时间，那么如果 E.Space 选择去点击第 i 个音符，那么他就没法点击所有到来时刻在 $(i - t_i, i + t_i)$ 中的音符。

为了获得更高的分数，E.Space 还计算了每个音符的性价比 a_i 。一个音符的性价比等于点击这个音符得到的分数除以 E.Space 点击它所需要的准备时间。

E.Space 就不指望全连了，他只是想让你帮他计算一下他最多可以得到多少分数。

对于 100% 的数据，保证 $t_i \leq n \leq 10^6, a_i \leq 10^9$ 。

Solution: 全连

容易想到 dp，从前往后考虑每个音符，并设 f_i 表示考虑了 $1 \sim i$ 个音符并且点击了音符 i 的最大分数，有转移：

$$f_i = a_i + \max_{j=1}^{i-t_i} [j + t_j \leq i] f_j$$

后面的是一个前缀 max，艾佛森括号的处理只需要将 f_j 挂在 $j + t_j$ 上，算出 f_j 时先不处理它的贡献（先不加到树状数组中）而是等到 $i \geq j + t_j$ 的时候再处理，就可以把艾佛森括号去掉了，剩下的就是朴素的前缀 max 和单点修改，树状数组直接做，时间复杂度 $\mathcal{O}(n \log n)$ 。

AT_arc132_e Paw

有 n 个方块排列在一排。每个正方形都有一个向左或向右的脚印或一个洞，以 $<, >, .$ 表示。

Snuke，将重复下面的程序，直到不再有一个有洞的方块。

1. 以相等的概率随机选择一个有洞的正方形。
2. 填上所选正方形的洞，站在那里，并以相同的概率随机面向左边或右边。
3. 沿着 Snuke 所面对的方向一直走，直到他踩到一个有洞的方块或离开这排方块。

这里，方块和方向的选择是相互独立的。

当 Snuke 踩到一个方块（没有洞）时，该方块在他行走的方向上会有一个脚印。如果该方块已经有了一个脚印，那么它就会被抹去并被一个新的脚印取代。

当 Snuke 完成这些程序时，求向左的脚印数量的期望值，模 998244353。

对于 100% 的数据，保证 $n \leq 10^5$ 。

Solution: AT_arc132_e Paw

观察最终脚印的形式：最终必然是 $<<<<====>>>>$ 的形式，其中 $====$ 指的是原串中的一段极长的不含 $.$ 的段。

证明考虑最后一次操作，假如是向左走，那么它左边的洞对应的操作就无关紧要了（最终都会被最后一次操作覆盖），然后接着考虑没有被覆盖的最后一次操作，依次考虑下去，最终会剩下来一个极长的不含 $.$ 的段没有被任何操作覆盖。

于是可以 $\mathcal{O}(n)$ 枚举这个极长段，将答案加上对应的概率乘 $<$ 的个数（即该段中 $<$ 的个数加上左边被覆盖的位置数量），后者容易计算，只需要找到概率。

这个概率是什么呢？其实就是 k 个洞按某种顺序操作，得到 $<<<$ 的概率乘剩下的洞得到 $>>>$ 的概率，两者本质上是对称的，因此只需要对每个 k 求出前者，设为 f_k 。

显然有 $f_0 = 1$, $f_k = (1 - \frac{1}{2k}) f_{k-1}$ ，这是因为考虑最左侧的洞，若它是最后一个被操作的，并且它选择了向右，显然不可以，否则只要后 $k-1$ 个洞的操作合法，这 k 个洞的操作就一定合法。

线性预处理出逆元就可以 $\mathcal{O}(n)$ 递推算出 f_i ，然后就可以 $\mathcal{O}(n)$ 求答案了。

CF1610H. Squid Game

猫猫们看了鱿鱼游戏，对其很感兴趣，决定自己举办一个猫猫游戏。

有 m 只猫猫参与了游戏，游戏失败的猫猫要被拉出去和铃酱贴贴。

蓝是游戏的裁判，由于所有猫猫都想和铃酱贴贴，于是她要淘汰所有的猫猫。下面是她淘汰猫猫的方式：

- 有一棵 n 个结点的无根树，每只猫猫都有两个特殊结点 x_i 与 y_i 。
- 在一次操作中，蓝可以选择某个结点 v 。接下来，对于所有未被淘汰的猫猫 i ，蓝将找到一个结点 w ，满足其在 x_i 到 y_i 的简单路径上且距离 v 最近，若 $w \neq x_i$ 且 $w \neq y_i$ ，那么猫猫 i 将会被淘汰。

现在蓝想要知道她至少要进行多少次操作才能淘汰所有的猫猫。如果无论如何都不能淘汰所有的猫猫，请输出 -1 。

本题数据满足： $1 \leq n, m \leq 3 \times 10^5$, $1 \leq \text{par}_i < i$, $1 \leq x_i, y_i \leq n$, $x_i \neq y_i$ 。

Solution: CF1610H. Squid Game

随便取一个点作为根，不妨设为 1。注意到，若 x_i, y_i 不是祖先后代关系，那么 w 合法当且仅当 w 不在 x_i, y_i 的子树中，否则，不妨设 y_i 是 x_i 的祖先， z_i 表示这条链上深度比 y_i 恰大 1 的点，那么 w 合法当且仅当 w 在 z_i 子树内且不在 x_i 子树内。

由此得出答案不为 -1 的充要条件：所有 x_i, y_i 链上的点数 ≥ 3 。

在之后，为了方便，我们将 x_i, y_i 不为祖先后代关系的链成为曲链，否则为直链，那么对于所有的曲链， w 若合法，那么 w 的父亲也一定合法，反之不一定，进一步的，将 w 设为根可以一次性解决所有曲链，因此接下来我们只关注直链。

按照 z_i 的深度降序贪心地考虑所有直链，若当前对 x_i, y_i 还不存在合法的 w ，那么我们新加的 w 的深度一定越小越好（因为剩下的直链不会有深度更大的了），进一步，新选取的点一定恰为 z_i 。用树状数组维护 dfs 序，可以单次 $\mathcal{O}(\log n)$ 查询子树中有多少个被选取过的 w ，于是可以 $\mathcal{O}(\log n)$ 查询 x_i, y_i 是否存在合法的 w ，总时间复杂度 $\mathcal{O}(n \log n)$ ，我们处理完了直链。

对于剩下的曲链，答案要么加一（选上根），要么不变，当所有曲链已经被满足的时候，答案显然不变，接下来我们只需要证明，若某个曲链无法被已有点满足，那么答案必然加一。

若答案不需要加一，那么显然是变化已有的被选点，进一步的，让某些被选点深度更小。但是根据我们的贪心策略，这显然是不可能的（可以看作新加一个深度更小的点，并删去一个原有被选点，但原有被选点是不可以被删的）。

还是可以用树状数组判断，总时间复杂度 $\mathcal{O}(n \log n)$ ，本题还有一种 dp 解法。

CF704C. Black Widow

给定 m 个布尔变量 x_1, x_2, \dots, x_m ，以及 n 个形如 x_i 或者 $x_i \text{ or } x_j$ 的布尔表达式，其中规定 $x_{-i} = \neg x_i$ 。并且，保证 x_i 和 x_{-i} 在所有的布尔表达式中一共只会出现最多两次。

请你求出，在一共 2^m 种布尔变量取值的情况下，有多少种情况，使得所有布尔表达式的值的异或为 1。此处认为，True 为 1，False 为 0。

由于结果可能过大，请输出答案模 $10^9 + 7$ 的结果。

对于 100% 的数据，保证 $1 \leq n, m \leq 10^5$ 。

Solution: CF704C. Black Widow

若两个表达式同时含有 x 或同时含有 $\neg x$ 或一个含 x ，另一个含 $\neg x$ ，在两表达式之间连边，当一个含 x ，另一个含 $\neg x$ 时权值为 1，否则权值为 0。

根据题意， x_i 和 x_{-i} 只会出现最多两次，因此所有表达式以及它们之间的边构成了若干个链和环（可能有自环，即一个表达式为 $x_i \text{ or } x_{-i}$ 或 $x_i \text{ or } x_i$ ）的时候。由于每个联通块异或的值独立（因为没有共同的变量），因此若我们知道了每个联通块取值为 0 或 1 的方案数，那么它们异或起来为 1 的方案数也是容易 dp 求得的。

同样对每个联通块 dp，特判孤立点和自环、长度为 2 的环，对于一个链的联通块，从一端 dp 到另一端，设 $f_{i,j}$ 表示前面的表达式异或和为 i ，当前表达式和之前一个表达式的公共变量为 j 时的方案数，转移平凡，细节略恶心。

对于一个环，断掉一条边，枚举边对应的变量的取值即可，同样略恶心。

为了减少码量和思维难度，我们可以直接写一个函数 $\text{solve_chain}(u, a, b)$ 表示解决一条链的情况，不同之处在于， a, b 表示链两端的表达式需要分别或上 a, b ，并且直接删掉表达式中没有边的变量。这样，对于环的情况，我们就只需要断掉一条边，当断掉的边权值为 1 时，取 $(a, b) = (0, 1), (1, 0)$ ，方案数相加即可；当为 0 时取 $(a, b) = (0, 0), (1, 1)$ 。

同样对于链的情况，当链端只有一个变量时，钦定 a （或 b ）为 0，否则可以是 0 也可以是 1，方案数加起来就行。

时间复杂度 $\mathcal{O}(n)$ 。

ARC112F Die Siedler

现在有 n 种牌和 m 种牌包，第 i 种牌包有 $s_{i,j}$ 张第 j 种牌。

初始时你有 c_j 张第 j 种牌。你可以以任意顺序执行下列操作之一任意多次：

- 选择 $1 \leq i \leq m$ ，获得第 i 种牌包所有牌。每种牌包都有无穷个。
- 选择 $1 \leq j \leq n$ ，要求有 $\geq 2j$ 个第 j 类牌。扔掉 $2j$ 个第 j 类牌，并获得 1 张第 $j \bmod n + 1$ 类牌。

最小化最终剩余牌数。

对于 100% 的数据，保证 $2 \leq n \leq 16, 1 \leq m \leq 50, 0 \leq s_{i,j}, c_j < 2j, c_1 + \dots + c_n > 0, s_{i,1} + \dots + s_{i,n} > 0$

Solution: ARC112F Die Siedler

特判初始时没有牌的情况，将每个状态看作一个特殊进制数字：第一位是二进，第二位是四进，第三位是八进...，即若有 A_i 张第 j 种牌，那么对应的状态就是

$$\sum_{i=1}^n A_i (2i - 2)!!$$

显然，假如可以进位（对应原题中的地 j 类牌换第 $j \bmod n + 1$ 类牌），我们是一定会进位的，于是换牌操作自然地变成了数字的进位，通过数字还原每类牌的数量也是容易的。

还有一个特殊的换牌是第 n 类换第 1 类，这可以看作是将数对 $2n!! - 1$ 取模，但是取模的要求是 $> 2n!! - 1$ 而不是 \geq 。

根据裴蜀定理，我们可以求出 m 中牌包对应数字与 $2n!! - 1$ 的 gcd，设为 d ，于是我们可以到达的状态数量就有 $\frac{2n!!-1}{d}$ 个了，暴力枚举，时间复杂度为 $\mathcal{O}(n^{\frac{2n!!-1}{d}})$ 。

假如不暴力枚举呢？容易发现我们要找的其实是牌数最少的、和初始状态模 d 同余的那个状态，可以跑同余最短路，时间复杂度是 $\mathcal{O}(n^2 d)$ 。特殊处理一下初始状态模 d 为 0 的情况，钦定这个时候必须经过至少一条边即可。

显然我们有两种取决于 d 的算法，可以考虑根号分治，当 $d > \sqrt{2n!! - 1}$ 的时候选暴力枚举的算法，否则选同余最短路算法。

算一下复杂度发现好像过不去，但是这里的根号是卡不满的，由于 d 是 $2n!! - 1$ 的因数，而其小于根号的因数最大其实是 1214827。

猜数

你要从数集 $\{1, 2, \dots, n\}$ 中猜一个数，每次提问一个数 x ，花费 x 的代价，交互器会返回大于、小于或等于。

定义 $C(n)$ 表示在最优策略下的总成本。最优策略指的是在**最坏情况**下最小化总成本的策略。

例如 $C(1) = 0, C(2) = 1, C(8) = 12$ ，给定 n ，求

$$\sum_{i=1}^n C(i)$$

对于 100% 的数据， $1 \leq n \leq 50000$ 。

Solution: 猜数

显然我们猜的数应该会比较偏右而不是偏左，具体的，它离右边的距离应该是 $\mathcal{O}\left(\frac{n}{\log n}\right)$ 。限于水平，笔者不会证明。

并且还有一个比较显然的性质：随着决策点 k 的右移，设 $f(l, r)$ 表示从 $[l, r]$ 猜数的代价，那么最开始是 $f(1, k-1) \leq f(k+1, r)$ ，之后就会变成 $f(l, k-1) > f(k+1, r)$ 。

根据之前的引理，我们本质上只需要维护 $f(1, [1..n])$ 以及 $f(a, b)$ ，其中 $b - a = \mathcal{O}\left(\frac{n}{\log n}\right)$ ，空间复杂度 $\mathcal{O}\left(\frac{n^2}{\log^2 n}\right)$ 。

怎么计算呢？当决策点 k 满足 $f(l, k-1) \leq f(k+1, r)$ 的时候，让 k 尽量靠右，否则让决策点尽量靠左，从小到大枚举右端点，再从右到左枚举左端点，单调队列维护决策点，时空复杂度均为 $\mathcal{O}\left(\frac{n^2}{\log^n}\right)$ 。

原题来自 ProjectEuler Problem 328，有一个 $\mathcal{O}(n \log n)$ 的解法，NFLS 的讨论区甚至有一个 $\mathcal{O}(n)$ 解法，笔者将它直接搬运到下面了：

首先我并不知道这是否是线性的，但是易得其为线性对数

考虑递推计算 $C(n)$ ，对于第一次的决策点，其左方已被计算，考虑维护其右方区间的长度 $L = 2^x + y (0 \leq y < 2^x)$ 并计算其最优方案在最坏情况的答案，考虑右方的方案（显然其决策点左方区间长度不小于右方区间长度，且要求其左右方区间的长度均小于 2^x ）：

1. 若 $y < 2^{x-1}$ ，则其决策点的右方区间长度必为 $2^{x-1} - 1$ 且在 n 足够大时左方的答案比右方答案大（显然不可能左右方区间长度均不大于 $2^{x-1} - 1$ ，故左方区间长度大于它，故需要右方区间在不大于的情况下尽量长）
2. 否则，则其决策点的左方区间长度必为 $2^x - 1$ 且右方的答案比左方大（可以类似分析）

因此，可以递推计算左端点（第一次决策点 +1）为 p ，长度为 L 时的答案 $y = f(L) \times p + g(L)$ ，其中 $f(L) = \lfloor \log L \rfloor$ ， g 的计算可以由以上得出（或看代码）

通过使用其他算法打表发现：区间总长度 +1 时，右方区间的长度的变化量必为 0 或 2 的幂，若特判每次的变化量至多为以前的最大值的两倍则可以通过（初始值赋到 $n = 100$ 也可以）。

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 int t[10000001], g[10000001], knots = 1, maxadd = 1, n;
4 long long sum;
5
6 int main() {
7     freopen("guess.in", "r", stdin);
8     freopen("guess.out", "w", stdout);
9     scanf("%d", &n);
10    for (int i = 2; i <= n; i++) {
11        int b = 2 << __lg(i / 3);
12        g[i] = g[i - b] - (b == 1 << __lg(i) ? i - b : b * __lg(i) - 1);
13        t[i] = i - knots + max(t[i - knots - 1], i * __lg(knots) + g[knots]);
14        for (int a = min(maxadd * 2, 1 << __lg(i - knots - 1)); a >= 2; a /= 2) {
15            int x = i - knots - a + max(t[i - knots - a - 1], i * __lg(knots + a) + g[knots + a]);
16            if (t[i] >= x) {
17                t[i] = x;
18                knots += a;
19                maxadd = max(maxadd, a);
20                break;
21            }
22        }
23        sum += t[i];

```

```

24 }
25 printf("%lld\n", sum);
26 }

```

Find a city

有一个 n 个点的竞赛图，每次你可以向交互库询问一条边的方向，目标是在 q 次询问之内找到任意一个出度 $\geq n - 2$ 的点或报告不存在这样的点。
对于 100% 的数据，保证 $3 \leq n \leq 1000$, $\sum n^2 \leq 10^7$, $q \geq 4n$ 。

Solution: Find a city

首先我们可以用 $n - 1$ 次询问构造出竞赛图的一个叶向生成树，方式是先随意钦定一个点作为根，每次找到一个之前没有被塞进生成树的点，询问它与根之间边的方向，若根指向它，不改变根，否则将根变为新点，原来的根变为新根的儿子。

之后我们可以每次找两个非根、非父子关系的点，查询它们之间边的方向，这样就一定可以确定某个点的入度至少为 2 从而排除这个点了。

这样操作之后，还剩下根和一个点，或者是跟和两个父子关系的点，操作次数是 $n - 2$ 。

若只剩下根和一个点可能成为答案，暴力查询即可，现在关注后者，暴力查询就总共是 $5n$ 次了，不可取。

我们尝试给这个算法打补丁，但是很难做，尝试稍微换一下思路：在构造出叶向树后先检查一下根是否可能是答案，若已经找到两条入边，直接停止。

假设上述操作检查了 k 条边（注意不能查根的儿子），那么我们就确定了 $k - 2$ 个点一定不是答案。这时再对剩下的点进行之前 $5q$ 次查询时的操作，就可以操作 $(n - 1) - (k - 2) = n - k + 1$ 次操作找到那两个点了，此时一共操作了 $(n - 1) + k + (n - k + 1) = 2n$ 次，暴力检查这两个点即可做到总查询次数不超过 $4n$ 。

AGC034E. Complete Compress

给你一颗 n 个节点的树，并用二进制串告诉你哪些节点上有棋子（恰好一颗）。

可以进行若干次操作，每次操作可以将两颗距离至少为 2 的棋子向中间移动一步。

问能否通过若干次操作使得所有的棋子都在一个点上，如果能，输出最小操作次数，如果不能，输出 -1 。

对于 100% 的数据， $2 \leq n \leq 2000$ 。

Solution: AGC034E. Complete Compress

枚举最终的汇点作为树根，操作可以分为两类：

- 让某两个点都向父亲走一步，这要求它们不为祖先后代关系。
- 让后代向上走，祖先向下走。

既然我们想让所有点都到根，那么操作一会让关键点的总深度减二而操作二不会改变，进一步的，操作二一定是没用的。

为什么呢？操作二有用处当且仅当这样之后某个点向上的机会变多，但这也是不存在的。

于是问题变成了判定性问题：判断是否可以汇到根，假如可以，用 $\sum \frac{dep_u}{2}$ 更新答案，其中 u 是关键点。

考虑一个经典模型：有若干个盒子，每个盒子里有若干个小球，每次可以选两个不同盒子里的小球并消掉，问最终最少剩下多少个小球。

问题的解决方法是：考虑小球数量最多的那个盒子，设数量为 x ，设所有盒子中小球的总数量为 y ，若 $x \geq y - x$ ，答案就是 $x - (y - x)$ ，即用数量最多的那个盒子与其它盒子消，否则，答案就是 $y \bmod 2$ ，即若 y 是偶数，可以消空，否则剩下一个。

前者显然，后者的一种构造是：将所有小球排成一排（同一个盒子的排在一起），第 i 个小球和第 $i + \lfloor \frac{n}{2} \rfloor$ 个小球消，显然它们两个一定不在同一个盒子，因为最大的盒子的数量也不超过 $\frac{n}{2}$ 。

对于这道题，设 $dist(u, v)$ 表示 u, v 之间的距离， $siz(u)$ 表示以 u 为根的子树内有多少个关键点， $g(u)$ 表示 $\sum_v dist(v, u)$ ，其中 v 是关键点， $f(u)$ 表示以 u 为根的子树最多操作多少次，转移考虑 u 的所有儿子的 $g + siz$ 做上面的模型，不同之处在于，当 $\max(g(v) + siz(v))$ （设这个取到最大值的 v 是 son ）超过总数的 $\frac{1}{2}$ 时，我们还应该再对 v 子树内做操作，设 $t = g + siz$ ，那么此时 $f(u)$ 应该是

$$f(u) = g(u) - t(son) + \min(f(son), \lfloor \frac{t(son) - (g(u) - t(son))}{2} \rfloor)$$

时间复杂度是 $\mathcal{O}(n^2)$ ，事实上换根 dp 可以做到 $\mathcal{O}(n)$ 。

CF908G. New Year and Original Order

给定 $n \leq 10^{700}$ ，问 1 到 n 中每个数在十进制下将各数位排序后得到的数的和，答案模 $10^9 + 7$ 。

例如，2311 在十进制下将各数位排序后得到的数就是 1123。

Solution: CF908G. New Year and Original Order

一看数据范围就知道是数位 dp 了，设 $c(x), 0 \leq x \leq 9$ 表示排序后将 x 看作 1，其他数字看作 0 后得到的数字的和，那么答案就是

$$\sum_{i=1}^9 i \times c(i)$$

用经典转换做一下：

$$\sum_{i=1}^9 \sum_{j=1}^i c(i)$$

交换求和顺序：

$$\sum_{j=1}^9 \sum_{i=j}^9 c(i)$$

设 $\sum_{i=j}^9 c(i)$ 设为 $d(j)$ ，含义是将 $\geq j$ 的看作 1，其余看作 0 得到的数字的和，发现可以数位 dp 求出 $d(j)$ ，做 9 次把答案加起来即可，时间复杂度 $\mathcal{O}(\log^2 n)$ 。

AGC024F. Simple Subsequence Problem

有一个 01 串集合 S ，其中每个串的长度都不超过 N ，你要求出至少是 S 中 K 个串的子序列的最长串，如果有多解，输出字典序最小的那组解。

由于 S 可能很大，因此我们是这样描述 S 的：

- 你将得到 $(N + 1)$ 个 01 串，第 i 个串的长度为 2^{i-1} 。
- 第 i 个字符串的第 j 个字符，代表数字 $(j - 1)$ 的、长度为 $(i - 1)$ 的二进制表示是否出现在 S 中。

对于 100% 的数据， $N \leq 20$ 。

Solution: AGC024F. Simple Subsequence Problem

精妙 dp，设 $f(a, b)$ 表示有多少个 S 中的串 A 满足：将 A 与 a 匹配后， A 剩下的串恰为 b ，这里的匹配指的是贪心匹配，即从左到右找 A 中第一个可以和 a 接着匹配的字符并匹配上。

初始时有 $f(\epsilon, x) = 1$ ，其中 ϵ 表示空串， $x \in S$ ，串 T 满足条件当且仅当 $f(T, \epsilon) \geq K$ ，有转移：

$$f(P, \epsilon) \leftarrow f(P, Q), Q \neq \epsilon$$

$$f(P + 0, Q') \leftarrow f(P, Q)$$

$$f(P + 1, Q') \leftarrow f(P, Q)$$

其中 $+$ 指的是字符串的拼接, Q' 是匹配后的 Q .

按照长度从小到大枚举 P 转移, 单次转移是 $\mathcal{O}(1)$ 的, 由于 $|P| + |Q| \leq n$, 所以空间复杂度是 $\mathcal{O}(n2^n)$ 的, 于是时间复杂度也是 $\mathcal{O}(n2^n)$.

【清华集训 2017】某位歌姬的故事

IA 是一名会唱歌的女孩子。

IOI2018 就要来了, IA 决定给参赛选手们写一首歌, 以表达美好的祝愿。这首歌一共有 n 个音符, 第 i 个音符的音高为 h_i 。IA 的音域是 A , 她只能唱出 $1 \sim A$ 中的正整数音高。因此 $1 \leq h_i \leq A$ 。

在写歌之前, IA 需要确定下这首歌的结构, 于是她写下了 Q 条限制, 其中第 i 条为: 编号在 l_i 到 r_i 之间的音符的最高音高为 m_i 。在确定了结构之后, 她就可以开始写歌了。不过她还是想知道, 一共有多少种可能的歌曲满足她的所有限制? 她听说你还有 9 个月就要去 IOI 了, 于是希望你帮她计算一下这个值, 对 998244353 取模。

对于 100% 的数据, 保证 $n \leq 9 \times 10^8$, $Q \leq 500$, $1 \leq m_i \leq A \leq 9 \times 10^8$ 。

Solution: 【清华集训 2017】某位歌姬的故事

离散化后容易得到每个位置的限制, 设为 up_i 。对于题中给出的限制 $[l, r], m$, 显然只有 $up_j = m, j \in [l, r]$ 的 j 可能成为这个最高音。枚举每个 m , 将 $up_j = m$ 的单独抽出来做 dp 就可以求出这些位置的方案数 (经典模型: 给出了若干个区间, 要求每个区间内都有至少一个关键点, 这里的关键点就是最高音)。

显然每个有限制的点都会在上述过程中算到且仅算一次 (因为 up 固定), 再乘上没有限制的点的方案数即可。

笔者认为离散化成左闭右开的区间容易写代码, 根据 dp 实现的不同可以做到 $\mathcal{O}(TQ^2)$ 或 $\mathcal{O}(TQ \log q)$, 均可通过本题。

【POI2015】Myjnie

有 n 家洗车店从左往右排成一排, 每家店都有一个正整数价格 p_i 。

有 m 个人要来消费, 第 i 个人会驶过第 a_i 个开始一直到第 b_i 个洗车店, 且会选择这些店中最便宜的一个进行一次消费。但是如果这个最便宜的价格大于 c_i , 那么这个人就不洗车了。

请给每家店指定一个价格, 使得所有人花的钱的总和最大。

对于 100% 的数据, 保证 $1 \leq n \leq 50$, $1 \leq m \leq 4000$, $1 \leq a_i \leq b_i \leq n$, $1 \leq c_i \leq 5 \times 10^5$ 。

Solution: 【POI2015】Myjnie

笛卡尔树上 dp, 容易发现一定存在一种最优解满足每个价格都是 b_i 中的某个值, 于是可以先把 b_i 离散化掉。

设 $f_{l,r,x}$ 表示只考虑区间 $[l, r]$ 中的人, 区间最小值为 x 时的最大收益, 二维前缀和预处理一下对于每个价格, 每个区间中会消费的人的数量, 容易做到 $\mathcal{O}(n^3m)$ 预处理和求答案。

bzoj2616. SPOJ PERIODNI

有 n 个底边长为 1 的矩形排成一排，第 i 个矩形的高为 h_i ，并且它们的底边共线，每个矩形可以看作 h_i 个格子，你可以往每个格子里放车。

给定 $0 \leq k \leq n$ ，你要求出往这些矩形中放入恰好 k 个车的方案数，满足同一个矩形中只有一个车，并且对于任意两个高度相同的车，它们所在行的中间有空格子。

Solution: bzoj2616. SPOJ PERIODNI

建出根为区间最小值的笛卡尔树，设 $f_{u,k}$ 表示在以 u 为根的子树中放 k 个车，并且它们的高度均比 u 的父亲高的方案数，转移枚举左、右儿子放多少个车在比 h_u 高的位置，然后组合数将剩余的填到 $(h_{fa}, h_u]$ 这个高度之间，预处理阶乘算组合数和排列数，时间复杂度 $\mathcal{O}(n^3 + h)$ 。