Problem A. 财迷女团的双向奔赴

时间限制: 2.5 seconds
空间限制: 1024 MB

简单题意

给定一棵带权树和一条边权为 m 但端点不定的边。

对于独立的 q 次询问，可以用这条给定的边连接任意两个叶子节点，询问两个点之间任何简单路径上的边权最大值。

$2 \le n \le 10^5$

$1 \le m \le 10^6$

$1 \le q \le 10^4$

$1 \le w \le 100$

$1 \le u, v \le n$

Tag

诈骗、基环树、DFS、LCA、分类讨论

题解

我们记题目给的边的边权为 w_0，树上的边权为 w_i 。

对于一个 n 个节点、$(n-1)$ 条边的树形结构，我们总是能按是否含有一条从根节点出发的链这个条件将其分为无根链的树和有根链的树。

对于一棵普通的不带根链的树，查询任意两点时，我们总可以从一个节点到达叶子节点，然后将这条边连接最大边权所在的叶子节点，然后经过最大边权到达根节点然后到达任意一个节点。所以只要树不带根链，答案一定是 $\max\{w_0, \max\limits_i w_i\}$ 。

如 **图** 1 所示，我们在下图中查询 $u = 11$ 和 $v = 12$ 这两个点，我们总是能从 12 号节点经过边权为 m 的边到达 6 号节点，经过最大边权（红色边）之后，到达根节点 1，再到达 11 号节点，所以答案就是 $\max\{1, m\}$。

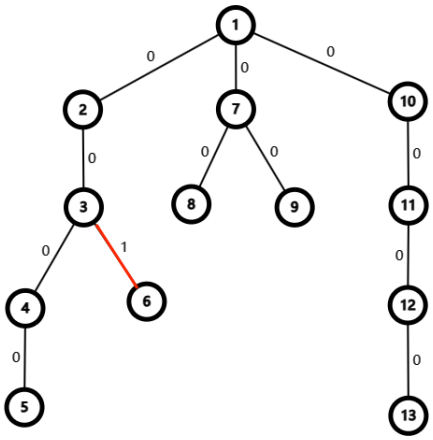

图 1: 无根链情况

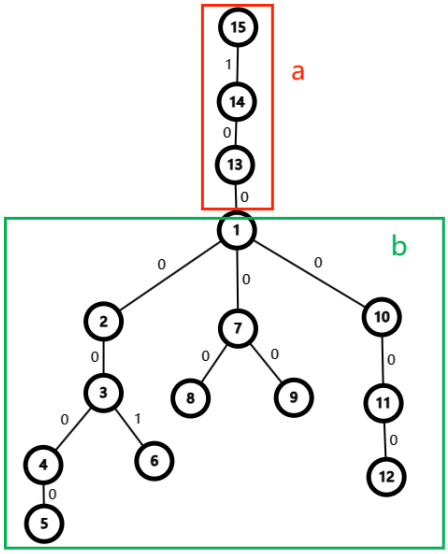

图 2: 含根链情况

但这是一种特殊情况，我们对当前的树进行扩展，当我们加上根链后，我们可以发现，当有查询的节点在根链上的时候，没办法像上述例子一样经过一个根节点再到达任意一个节点，那么我们可以将树分为两部分，如 **图** 2 所示。

（1）当两个节点都在 part a 的时候，直接查询两点直接的最大值。

（2）当两个节点都在 part b 的时候，直接输出 part b 和 w_0 之间的最大值。

（3）当有一个节点在 part b 一个节点在 part a 时，把查询转化为两个查询，把一个查询变为 part b 中树的根节点，这样就转化为了在无根链树上的查询和在单链查询，直接取 max 即可。

综上我们可以看出，特殊情况是包含在（2）中的。所以我们只需要 DFS 一遍将根链找出，然后再分类讨论查询两点最大值即可。

查询办法是通过类似 LCA 的算法来维护任意两点之间简单路径的最大值。

时间复杂度 $O(q \log n + n)$ 。

Problem B. 流萤

时间限制: 1 seconds
空间限制: 256 MB

简单题意

长度为 n 的数组，尽可能少地去修改位置上的数字，让修改后的数列任意两个相邻的数字的和是完全平方数。输出方案。

$1 \le n \le 10^6$

Tag

构造、数论、DP

题解

先考虑一些简单的情况：

（1）$n = 2$ 时，只要判断两个数的和是不是完全平方数。

（2）$n = 3$ 时：

首先判断掉不需要修改的情况。

然后判断，如果 $a_1 + a_2$ 与 $a_2 + a_3$ 有没有完全平方数，有的话只要修改一个即可。

然后只剩下 $a_1 + a_2$ 与 $a_2 + a_3$ 都不是完全平方数的情况，这个时候答案只可能是 1 或 2。

显然修改 2 次是完全足够的，那我们只要判断 1 次修改能否成功即可。

修改 1 次的情况，显然这个时候只能修改 a_2 。我们考虑一下一个转化的问题，给定整数 a, b ，何时存在整数 b , 使得 $a + b, b + c$ 都是完全平方数。

（i）$a = c$ 时，显然恒存在一个整数 b 使该条件成立。

(ii) $a \neq b$ 时，令 $d = b - a, a + b = x^2, b + c = y^2$ ，则有 $y^2 - x^2 = d = (x - y)(x + y)$ 。再令 $u = x - y, v = x + y$ ，若存在整数 u, v 且 u, v 同奇偶则存在 b 。

构造后有：

- $d \mod 2 = 1$, 则取 $u = d, v = 1$ 。
- $d \mod 4 = 0$, 则取 $u = \dfrac{d}{2}, v = 2$。
- $d \mod 4 = 2$, 此时无解。

然后我们将构造后的 u, v 带回，得到对应的 b 即可。

然后我们去思考一个问题，存不存在一种情况，使得我们需要连续修改五个数字。这显然是不可能的，因为连续修改五个的话，中间的那个数字是完全可以不修改的，所以我们只要考虑连续修改不超过四个数字的情况。

连续修改四个数字，可以调整为选 2 组连续修改 2 个。所以我们只要考虑连续修改一个，两个，三个的情况。

用 dp_i 表示第 i 个修改且前 i 个合法的情况，对于最后一个位置修改的情况特判。

然后一边 check 一边转移，同时用一个前驱数组记录路径，最后依据路径输出答案即可。

时间复杂度 $O(n)$ 。

题外话

星穹铁道一周年啦！建议大家去玩星穹铁道！

Problem C. A * B Problem

时间限制: 1 seconds
空间限制: 256 MB

简单题意

给定两个整数 A, B ，输出 $A \times B$ 。

$1 \leq A, B \leq 9$

Tag

语法/打表

题解

直接输出即可，也可打九九乘法表。

时间复杂度 $O(1)$ 。

Problem D. 不玩明日方舟的有难了

时间限制: 3 seconds
空间限制: 1024 MB

简单题意

给定一个长度为 T 的序列，要对该序列非 0 值染色，染色规则为选取不相交的区间，染成红蓝俩色，蓝色区间相隔至少为 k ，红色区间不能相邻且里面所含的非 0 值个数至多为 m 。

对这些区间按左端点从小到大排序后按顺序从 1 开始编号。

第 i 个红色区间的代价为 $(S+\gcd(a_l,a_{l+1},\ldots,a_{r-1},a_r))\times\sum_{k=l_i}^{r_i}[a_k>0]$ ，当 i 为最后一个时 $S=i$ ，否则 $S=i+1$ 。

第 i 个蓝色区间的代价为 $(S+w)\times\sum_{k=l_i}^{r_i}[a_k>0]$ ，$S=i$ 。

$1\le T,k,m\le 10^5$

$0\le a_i,w<10^7+19$

Tag

DP、后缀 gcd 、拆分转移方程、gcd 的性质、单调队列优化

题解

观察到红色区间不能相邻，因此考虑将红色区间纳入蓝色区间去考虑 DP 。

对于每个区间代价中的 S 我们可以提前处理，从左到右每次选取区间的时候对剩下的所有 $a_i>0$ 的数都增加一次贡献计算即可。

设 dp_i 表示序列中 $1\sim i$ 已经划分成若干个合法区间的最小花费。pre_i 表示序列中 $1\sim i$ 中非零的个数。

转移的时候只从上一个蓝色区间转移而来，并且考虑俩个蓝色区间内是否有红色区间的贡献即可。则有转移方程

$$dp_i = \min_{j=0}^{i-2k-1} \{dp_j + (pre_{n+k} - pre_j) \times (1 + [d \neq 0]) + (pre_{i-k-1} - pre_j) \times d + (pre_i - pre_{i-k-1}) \times w\}$$

，其中 $d = \gcd(a_{j+1}, a_{j+2}, \ldots, a_{i-k-1})$ ，且需要满足 $pre_{i-k-1} - pre_j \leq m$ 才能转移。

其中 $pre_{i-k-1} - pre_j \leq m$ 可以使用单调队列优化来处理，对于 gcd 考虑到其是从 j 开始到 $(i-k-1)$ 的后缀 gcd ，因此我们考虑通过维护从 j 开始的后缀 gcd 来处理，其变化次数不会超过 log 次，对于每个 gcd 的值维护一个单调队列和对应的下标，每次把需要更改的 gcd 暴力更改，这样子如果使用哈希表存储每个 gcd 对应单调队列的话时间复杂度为 $O(n \log^2 V)$ 。

std 在 Codeforces 上的运行时空：

- __gnu_pbds::pb_hash_table ： 701 ms ， 6200 kB 。
- std::map ： 78 ms ， 5200 kB 。
- std::unorder_map ： 156 ms ， 5200 kB 。

Problem E. 循环序列

时间限制: 1 seconds
空间限制: 256 MB

简单题意

给定一个周期为 n 的序列的前 n 项 $a_1, \cdots, a_n$ 。

有 q 个询问，每个询问给定三个整数 l, r, t ，表示询问从区间 $a_l, a_{l+1}, \cdots, a_r$ 中取两个相异的元素 a_i 和 a_j $(l \le i, j \le r, i \ne j)$ ，满足 $a_i = a_j = t$ 的方案数。

$1 \le n \le 1 \times 10^5$

$1 \le q \le 10$

$-10^5 \le a_i \le 10^5 \ \ (1 \le i \le n)$

$1 \le l < r \le 10^6$ ， $|l - r| \le 114514$

$-10^5 \le t \le 10^5$

Tag

暴力、组合计数

题解

将下标都减 1 ，即序列下标从 0 开始时，用 $\%\ n$ 可简便地实现循环。

对每个询问，暴力统计区间内有多少个等于 t 的元素，不妨设有 cnt 个，则方案数为 $C_{cnt}^2 = \dfrac{cnt(cnt-1)}{2}$ 。特别地，$cnt < 2$ 时，只可能是 $cnt = 0$ 或 $cnt = 1$ ，此时上式的结果仍为 0 ，无需特判。注意开 long long 。

时间复杂度 $O(q \cdot \max|l - r|)$ 。

Problem F. 强化学习 Is All You Need

时间限制: 1 seconds
空间限制: 256 MB

简单题意

有 n 个状态（不包括终止状态）和 m 个动作，给出两份表格分别表示，每个状态在每个动作下一步将会转移到哪一个状态，以及每个状态在每个动作下转移会的到的奖励。询问从状态 s_1 转移到终止状态 s_{n+1} 的所能积累的最大奖励。

$1 \leq n, m \leq 1000$

奖励值的范围：$[-10^6, 10^6]$

Tag

阅读理解、贪心、思维、记忆化搜索/SPFA

题解

可以使用记忆化搜索的方法，我们用 $f[i]$ 来表示状态 s_i 作为起点所可以积累到的最大奖励值。我们设 $tran[i][j] = x$ 表示状态 s_i 在动作 j 下会转移到状态 s_x，$reward[i][j]$ 表示状态 s_i 在动作 j 下转移会得到的奖励值，则有更新 $f[x] = max\{f[x], f[tran[x][i]] + reward[x][i]\}$。

另一种解法是：将每个状态之间建边权为对应奖励值的边，然后用最长路求最大值。注意到边权有可能为负，所以只能使用 SPFA，不能使用 Dijkstra 算法。

Problem D. 半吊子的华尔兹

时间限制: 2 seconds
空间限制: 256 MB

简单题意

给出一个由若干三角形拼接而成的地图（以 n 行 m 列的矩阵形式给出），每一个三角形会被涂上四种颜色（颜色 $0,1,2,3$）之一，判断能否以某个三角形为起点放置一个四面异色（分别为颜色 $0,1,2,3$）的正四面体，通过以下两种移动方式可重复的走完地图上所有三角形，同时要求整个过程中正四面体与所在的三角形格子颜色一致。

移动方式 1：绕边翻滚，即选择触底三角形面的任意一条边为轴，绕轴翻滚，移动到**相邻**的格子。

移动方式 2：绕点旋转，即选择触底三角形面的任意一个点为圆心，绕点旋转，移动到**相邻**的格子。

$1 \le n, m \le 1 \times 1000$

$1 \le \sum(n \cdot m) \le 1 \times 10^6$，其中 $\sum(n \cdot m)$ 表示 t 组测试数据的 $n \cdot m$ 之和

Tag

思维、DFS/BFS、DP

题解

不难发现当答案为 yes 时，选择地图上任意一个三角形作为起点都是可行的，所以下面我们默认始终从地图左上角的三角形格子作为起点。

一个朴素的思路为可以使用 DFS/BFS 方法遍历整个地图，判断能否走通。

可以得到正四面体在一个给定颜色的三角形格子上，根据其他三个面的颜色的分布不同，至多有 3 种状态（对于每一个状态，我们使用其他 3 个面的颜色数字进行表示，即 3 个数字表示一个状态）。再根据正四面体所处的是正三角形还是倒三角形分为 2 类，所以正四面体在地图上

至多有 $4 \times 3 \times 2$ 种状态。

可以得到正四面体仅在相邻格子同色的时候可以通过绕点旋转移动过去，并且还发现通过在两个相邻格子之间反复绕点旋转的方式，可以让正四面体在两个相邻格子上调整为 3 种状态的任意一种。

此题难点主要在于正四面体相邻格子之间的状态转移方法上。糟糕的实现可能会花费较多的时间，下面提供一个思路相对比较清晰的实现。

发现如果没有绕点旋转操作，正四面体移动的可能性会大幅减少。以 **图** 3 为例，当正四面体起始状态确定时，其可以移动的路线将会变得唯一且规律，即只要确定了当前所处三角形的颜色和“正倒”便可以唯一确定正四面体的状态。我们称 **图** 3 所示的可行路线为一张“通图”，可以发现一种“通图”刚好对应正四面体的 8 种状态。

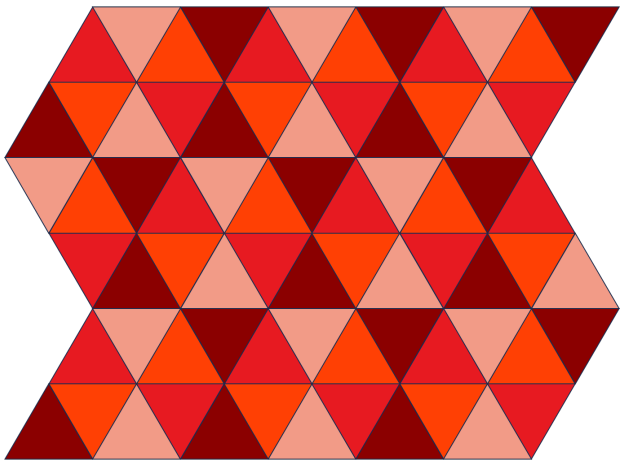

图 3: “通图”1

根据起始状态有 3 种，我们可以得到这道题目至多只有 3 张“通图”，对应的可以将正四面体全部 $4 \times 3 \times 2$ 种状态根据所处的“通图”分为 3 类。可以发现绕边翻滚操作是令正四面体从“通图”内的一种状态转移到“通图”内的另一种状态。**图** 4 和 **图** 5 为剩下的 2 种“通图”。

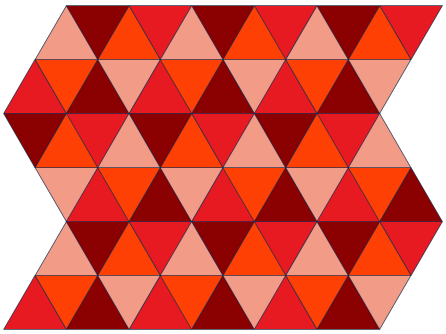

图 4: “通图”2

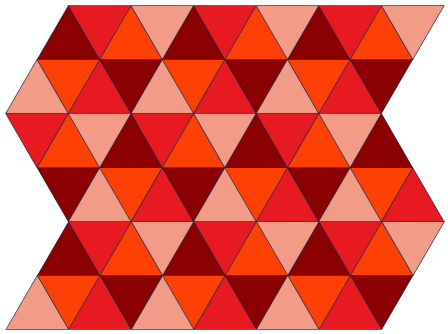

图 5: “通图”3

加入绕点旋转操作之后，正四面体通过该操作会从一张“通图”的状态转移到另一张“通图”的状态中。而再通过反复绕点旋转操作，正四面体在两个同色相邻的格子上可以调整为 3 种“通图”中的任意一种状态。

将 24 种状态分为 3 张“通图”之后，转移就很简单了。首先是可达性判断，前面我们使用 3 个数字表示一个状态，相邻格子异色时，我们可以通过判断状态中对应位置的数字和相邻格子的颜色是否相同来知道是否可达，相邻格子同色时则一定可达；其次是状态转移，我们给每一个三角形格子记录一个三维的 DP 数组，表示当前三角形格子对应的 3 种状态所属的“通图”是否可达，绕边翻滚会令相邻格子的同一“通图”内的状态置为可达，绕点旋转会令相邻格子的任意“通图”内的状态置为可达。

时间复杂度 $O\left(\sum(n \cdot m)\right)$ 。

为辅助理解，这里给出关键代码实现，如 **图** 6 所示。

```
bool dp[MAXN][MAXN][3];
// 第一维度表示三张“通图”，每张通图包含4*2种状态，每个状态由3个数字表示
int mp[3][4][2][3] = {
    {{{3, 1, 2}, {1, 2, 3}},
     {{2, 0, 3}, {0, 3, 2}},
     {{1, 3, 0}, {3, 0, 1}},
     {{0, 2, 1}, {2, 1, 0}}},
    {{{1, 2, 3}, {2, 3, 1}},
     {{0, 3, 2}, {3, 2, 0}},
     {{3, 0, 1}, {0, 1, 3}},
     {{2, 1, 0}, {1, 0, 2}}},
    {{{2, 3, 1}, {3, 1, 2}},
     {{3, 2, 0}, {2, 0, 3}},
     {{0, 1, 3}, {1, 3, 0}},
     {{1, 0, 2}, {0, 2, 1}}},
};
int dirx[2][3] = {{0, 0, 1}, {0, -1, 0}}, diry[2][3] = {{-1, 1, 0}, {-1, 0, 1}};
struct Node {
    int x, y, mpidx;
};
// stax，stay表示起点坐标，stampidx表示起始状态所属的“通图”
void bfs(int stax, int stay, int stampidx) {
    queue<Node> que;
    que.push(Node{stax, stay, stampidx});
    while (que.size()) {
        Node tmp = que.front(); que.pop();
        int x = tmp.x, y = tmp.y, mpidx = tmp.mpidx;
        int type = (x + y) % 2;
        int c = gra[x][y];
        for (int i = 0; i < 3; i++) {
            int xx = x + dirx[type][i], yy = y + diry[type][i];
            if (xx < 1 || xx > n || yy < 1 || yy > m) continue;
            if (gra[xx][yy] == -1) continue;


            int cc = gra[xx][yy];
            if (cc == c) { // 同色则通过绕点旋转
                // 三种“通图”都可达
                for (int mptmp = 0; mptmp < 3; mptmp++) {
                    if (dp[xx][yy][mptmp] == false) {
                        dp[xx][yy][mptmp] = true;
                        que.push(Node{xx, yy, mptmp});
                    }
                }
            } else { // 异色智能绕边翻滚
                // 颜色不一致，或者已经到达过了则跳过
                if (mp[mpidx][c][type][i] != cc || dp[xx][yy][mpidx] != false)
                    continue;
                dp[xx][yy][mpidx] = true;
                que.push(Node{xx, yy, mpidx});
            }
        }
    }
}
```

图 6: 关键代码实现

Problem H. 战术级白板在哪里

时间限制: 1 seconds
空间限制: 256 MB

简单题意

n 次询问，每次询问给定一个长度为偶数的数字 m，相邻两个数码两两组合，得到 $\frac{len(m)}{2}$ 个整数，输出最大的一个整数。

$1 \leq n \leq 1000$

$1 \leq m \leq 10^{1000}$

Tag

字符串、暴力、模拟

题解

用一个 std::string 接收后，扫一遍，暴力拆解取 max 即可。

时间复杂度 $O\left(\sum len(m)\right)$ 。

Problem I. 摇钱树

时间限制: 2.5 seconds
空间限制: 1024 MB

简单题意

n 个节点的树，q 个询问，每个询问给定节点 u 和整数 k ，以 u 为起点选 k 条可重叠的链使得链上的边权和最大（重叠的边只计算一次贡献）。

$1 \leq n \leq 2 \times 10^5$

$1 \leq q \leq 3 \times 10^5$

$1 \leq w \leq 1 \times 10^9$

$1 \leq k \leq n-1$

Tag

贪心、思维、离线、换根 DP、动态开点/(预处理 + 离散化)、权值线段树、线段树二分

题解

若只有一个询问，取询问的节点作为树的根节点。

对每个节点 u ，若边 (u, fa_u) 有贡献，由贪心：至少选了 u 的子树内的一个叶子。反之，若一棵子树内只选了一个叶子，则应贪心地选择最长链。由上述讨论：可将边 (u, fa_u) 的贡献挂在 u 的子树内的最长链上。这样操作后，所有叶子独立，直接选链长前 k 大即可。

对多个询问，考虑换根。注意到从节点 u 转移到节点 v 时，只有边 (u, v) 的归属发生改变，即从 v 的子树的最长链归到了 v 的子树外的最长链。如 **图** 7 和 **图** 8 所示。

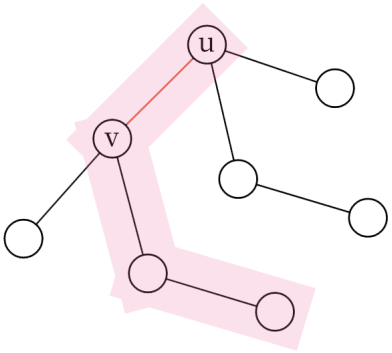

图 7: 以节点 u 为根

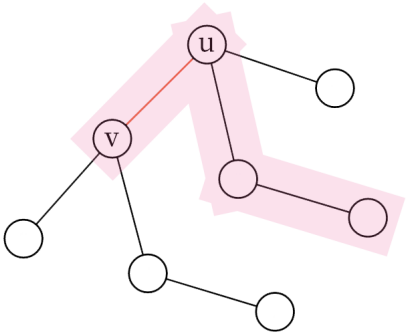

图 8: 以节点 v 为根

上述的转移可以用换根维护，即维护每个节点子树内来自不同子树的最长链和次长链，则节点 u 换根到节点 v 时，v 的子树外的最长链即边 (u,v) 拼上 u 的子树外的最长链或 u 的非 v 子树的最长链，而后者即 u 的子树的最长链或次长链。至此已维护出换根过程中的链长变化量。

将询问离线到询问节点上，每次询问即查询以询问节点为根时，叶子节点权值（即独立后的链长）的前 k 大之和。换根时在动态开点的权值线段树上二分，动态查询前 k 大即可。Alternative 的做法是：第一遍换根预处理出所有可能出现的链长并离散化，第二遍换根用普通的权值线段树维护。可能存在用其它支持动态加点、动态查询前 k 大的数据结构的做法。

时间复杂度 $O((n+q)\log n)$ 。

std 在 Codeforces 上的运行时空：

- 动态开点：935 ms ， 187100 kB 。
- 离散化：624 ms ， 100800 kB 。

题外话

本题在第一次验题的时候因为写错题面而出现了 Easy Version 。Easy Version 中选择的 k 条路径不可重叠。

Easy Version 的做法：换根过程维护最长链。时间复杂度 $O(n+q)$ 。

Easy Version 因为过于 easy ， 故没有放到比赛中。

Problem J. 守护大陆

时间限制: 1 seconds
空间限制: 256 MB

简单题意

给定两个数组 $a[1\dots n], b[1\dots m]$，记两数组的中位数分别为 M_a, M_b：

1. 若 $nM_a > mM_b$，则输出 Yes；
2. 若 $nM_a < mM_b$，则输出 No；
3. 若 $nM_a = mM_b$，则输出 NY，

其中 n, m 分别为数组 a、数组 b 的元素个数。

$1 \le a_i, b_i \le 10^9$

$1 \le n, m \le 10^6$，且保证一定为奇数

Tag

模拟、排序

题解

由于保证 n, m 为奇数，故寻找数组中位数，只需对数组进行排序，数组 a, b 的中位数即为 $M_a = a\left[\lfloor \frac{n}{2} \rfloor + 1\right]$ 和 $M_b = b\left[\lfloor \frac{m}{2} \rfloor + 1\right]$ 。排序可用 std::sort() 函数或其他复杂度为 $O(n \log n)$ 的排序。

之后计算 nM_a 与 mM_b，判断大小后输出答案即可。

也可直接使用 std::nth_element() 函数求中位数。

时间复杂度 $O(n \log n)$ 。

Problem K. DBer 最喜欢数据了

时间限制: 2 seconds
空间限制: 128 MB

简单题意

计算表达式 $\sum_{L=1}^{N}\sum_{R=L}^{N}\left(\sum_{i=L}^{R}a_i\right)^p \bmod p$ 的值。

$2 \le \sum n \le 2 \times 10^5$

$2 \le n \le 1 \times 10^5$

$1 \le a_i \le 10^8$

Tag

思维、数论、Fermat 小定理/二项式定理

题解

难点在于两个问题的转化：

- $\left(\sum a_i\right)^p \bmod p$ 怎么计算?
- 通过算贡献来推广到计算任意区间。

问题一

费马小定理说的是：若 p 是质数，设整数 a ，则有 $a^p \equiv a \pmod p$ 。

当然，也会有同学没有见过，这里给出 $(a+b)^p \bmod p$ 的数学推导。

由**二项式定理**：$(a+b)^p = \binom{p}{0} \times a^p + \binom{p}{1} \times a^{p-1}b^1 + \binom{p}{2} \times a^{p-2}b^2 + ... + \binom{p}{p} \times b^p$ 。

因为 $\binom{p}{k} = \frac{p(p-1)\cdots(p-k+1)}{k!}(1 \le k \le p-1)$ 成立，在模 p 意义下 $\binom{p}{1} \equiv \binom{p}{2} \equiv \cdots \equiv \binom{p}{p-1} \equiv 0 \pmod p$ 。

所以 $(a+b)^p \equiv a^p + b^p \pmod{p}$ 。

用快速幂预处理也能通过。

问题二

考验同学们对问题的转换能力。因为要把所有区间都计算在内，所以反过来考虑每个数对答案的贡献是多少，就是对于某一个数它会在几个区间内出现。

时间复杂度 $O(n)$ 或 $O(n \log p)$ 。

Problem L. 生稀盐酸！启动

时间限制: 2 seconds
空间限制: 512 MB

简单题意

每个敌袭每天会向上移动，会和其他敌袭合并、被哨站摧毁或摧毁哨站，要设置最少的防御力给驻扎地。

$1 \le T \le 1 \times 10^5$

$1 \le u, v \le n, u \ne v$

$1 \le \sum n \le 1 \times 10^5$ ，其中 $\sum n$ 表示所有测试数据的 n 之和

$0 \le |a_i| \le 1 \times 10^9$

Tag

思维、暴力/启发式合并/线段树合并/长链剖分

题解

解法一

直接使用启发式合并维护每个点的最大敌袭即可。时间复杂度 $O(n \log n)$。

解法二

从浅到深去往上推兵线，记录下这个点最高到哪里，或者到根，记录每个节点最多走到哪里，假如 a 最多走到了 b，以后节点走到 a 之后，直接跳到 b，因为中间路径的哨站肯定已经没了。然后把普通节点看成攻击力 0 的敌袭也往上走，这样就避免了走重复的路，每个节点只会被走到一次，时间复杂度 $O(n)$。

Problem M. 暂时没想法

时间限制: 2.5s seconds
空间限制: 512 MB

简单题意

t 组测试数据，每组测试数据有 n 个点和 m 条边，保证原图中有两个及以上的联通块。询问能否在不使用原图中已有的边的情况下重新建图使得所有的点全部联通。能的话，请输出至少需要几条边以及方案。

$$1 \leq \sum n \leq 1 \times 10^6$$

$$1 \leq \sum m \leq 1 \times 10^6$$

Tag

贪心、思维、DFS

题解

由题目中的从任意一个点出发总存在某一个点无法到达，可知，原图中有两个及以上的联通块，我们创建的新图要求在不能使用原图中已有的边的情况下建图使得所有的点在一个联通块内。我们可以发现，在一个联通块内的点之间可能有边相连，但是在两个联通块上的点一定没有边相连（不然这就变成了一个联通块了），因此我们可以先求出所有原图中的所有联通块，并将他们编号为 $1, 2, \cdots, k$，然后我们选择 1 号联通块中的某一个点，将其与 2 号联通块中的所有点建边，再选择 2 号联通块中的某一个点，将其与 3 号联通块中的所有点建边，以此类推，直到最后选择 $(k-1)$ 号联通块中的某一个点与 k 号联通块中的所有点某个点建边即可。这样即可再不使用原图已有的边的情况下，将所有的点全部联通。

用 DFS 求原图中联通块即可。

时间复杂度 $O(n)$ 。