# T1 XOR

## 题意解释

给定数字矩阵，反复给某些三角形区域进行区间加的操作，求最后矩阵元素的异或和，其中异或只是个幌子，于解题并无影响。

## 考察知识点：

二维差分（前缀和）

### 30%：直接暴力

将每个矩阵加上对应的值计算，复杂度$O(qn^2)$，代码略

### 60%~90% 按行差分

容易想到，本题的数据结构只有一次查询，Q次修改，所以差分思想可维护。

将一个n行的三角形拆成n行线性结构进行差分数组维护（差分数组是一种高效地执行数组区间增减操作的方法，具体而言，给定一个数组 `a`，其差分数组 `diff` 的第 `i` 个元素是 `a[i] - a[i-1]`，对差分数组求前缀和即可得到原数组，可认为差分是前缀和的逆过程，两者之间有密不可分的联系）。

由于题目数据比较水，可以直接拿到90分，复杂度$O(qn)$。

```cpp
#include<cstdio>
#define maxn 3039
using namespace std;
int n, m, r, c, l, s;
long long ans,f[maxn][maxn];
void add(int x, int y, int s) {
    if(y <= n) f[x][y] += s; // 确保y坐标在矩阵范围内，然后进行更新
}

int main() {
    freopen("xor.in", "r", stdin);
    freopen("xor.out", "w", stdout);
    scanf("%d%d", &n, &m);
    while(m--) {
        scanf("%d%d%d%d", &r, &c, &l, &s);
        // 对于给定的三角形区域，使用差分数组更新其元素
        for(int i = 1; i < l+1; i++) {
            add(r+i-1, c, s);       // 左边界加上s
            add(r+i-1, c+i, -s);   // 右边界减去s，构成下三角形的形状
        }
    }
    // 根据差分数组还原矩阵的值，并计算所有元素的异或和
    for(int i = 1; i < n+1; i++) {
        for(int j = 1; j < n+1; j++) {
            f[i][j] += f[i][j-1]; // 使用累加还原矩阵的值
            ans ^= f[i][j];       // 异或操作
        }
    }
    printf("%lld\n", ans); // 输出最终的异或和
    return 0;
}
```

## 100%：

还是考虑差分，刚才我们只对行内部进行了差分，但是行与行之间的信息我们并没有利用好，所以我们先退化问题，如果这道题目要求的不是三角形，而是矩形，那题目就简单很多了，我们可以从二维前缀和的思路理解二维差分。

考虑如何快速标记（差分）一个矩阵，对于矩阵前缀和来说，$f[i][j]$表示以$(i,j)$为右下角的矩形，那么$f[i][j]=f[i-1][j]+f[i][j-1]-f[i-1][j-1]+a[i][j]$($a[i][j]$表示矩阵原信息)

以下图为例，$f[5][6]$为右下角的矩形，等于**黄色矩形+红色矩形-蓝色矩形**，即容斥思想

image-20230901111808283

用二维差分数组的思想进行标记，用二维前缀和的方式还原矩阵即可

但是题目是任意矩形，例如我们要将左上角为$(1,2)$，右下角为$(3,4)$的矩阵统一加1，首先在左上角设置差分标记：

```
0 1 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
```

但这样一来，进行二维前缀和求解后的矩阵是这样的：

```
0 1 1 1 1 1
0 1 1 1 1 1
0 1 1 1 1 1
0 1 1 1 1 1
0 1 1 1 1 1
```

所以需将矩形根据右下角拆成四个部分，进行相应的减操作，只需下图四个位置做差分标记即可。

注意，在差分时，仍要考虑容斥的情况，因为每个差分值都影响了右下范围，所以在矩阵右下方区域受到$(1,5)$和$(4,2)$区域的同时影响，需要将多减的加回来。

```
0 1 0 0-1 0
0 0 0 0 0 0
0 0 0 0 0 0
0-1 0 0 1 0
0 0 0 0 0 0
```

接下来再考虑三角形，三角形同样可以差分，需要细心推理，注意到三角形是由最上方顶点标记地，所以三角形的前缀和可以理解为一个从上往下方向的前缀和：以$(2,3)$为顶点的三角形为例：

```
0 0 0 0 0 0
0 0 1 0 0 0
0 0 1 1 0 0
0 0 1 1 1 0
0 0 1 1 1 1
```

令$f[i][j]$表示以$(i,j)$为顶点的三角形的数据的话，注意到，$f[i][j]=f[i+1][j]+f[i+1][j+1]-f[i+2][j+1]$，见图（同样是容斥原理：蓝+绿-红）：

image-20230901120457180

但是这是三角形的前缀和，对于某一个区域的子三角形如何处理，这里的处理其实同矩阵一样，进行差分即可，比如要更新 $a$ 区域，可以先让 $a$、$b$、$c$ 同时加上 $s$，然后再分别减去 $b$、$c$ 的部分。：

```
..a.......
..aa......
..aaa.....
..aaaa....
..bbbbc...
..bbbbcc..
..bbbbccc.
```

其中b区域的矩阵我们已经可以差分处理了，三角形的话我们刚刚也找到了规律，只需标记a区域的顶点与c区域的顶点即可，以统一加1为例（暂不考虑矩形）：

```
..1.......
.........
..........
..........
......-1...
..........
..........
```

如此，两个三角形一加一减，在做前缀和时互相抵消。

代码：

```
#include<cstdio>
#define maxn 3039
using namespace std;
int n, m, r, c, l, s;
long long ans,f[maxn][maxn], g[maxn][maxn];
void add1(int x, int y, int s){if(x<=n&&y<=n)f[x][y] += s;} //三角形
void add2(int x, int y, int s){if(x<=n&&y<=n)g[x][y] += s;} //矩形
int main(){
	scanf("%d%d", &n, &m);
	while(m--){
		scanf("%d%d%d%d", &r, &c, &l, &s);
		add1(r, c, s); //顶点加s
		add1(r+l, c+l, -s);//右下方减s
		add2(r+l, c, -s);//维护需要减去的矩阵，注意是减去
		add2(r+l, c+l, s);
	}
	for(int i = 1; i < n+1; i++)
		for(int j = 1; j < n+1; j++){
			if(i>1)f[i][j] -= f[i-2][j-1];
			f[i][j] += f[i-1][j-1] + f[i-1][j]; //三角形的前缀和
			g[i][j] += g[i-1][j] + g[i][j-1] - g[i-1][j-1];//矩阵的前缀和
			ans  ^= f[i][j]+g[i][j];
		}
	printf("%lld\n", ans);
	return 0;
}
```

# T2 游戏

## 题意解释

$A$和$B$两人轮流处理一个整数集合，第 $i$ 轮中，集合$S$根据是否为$b_i$倍数被分割为两部分，此时可以保留下其中一部分，扔掉另一部分，最终m轮后剩下数字和即为权值，A希望最大化权值，

B则反之，博弈状态下问最终结果。

# 考察知识点：

搜索、思维

# 20%~40% 直接暴力

考虑暴力枚举每个人的决策，同时维护在前 $i$ 轮决策下，当前的 $S$ 集合，如果 $S=\phi$，那么立刻返回 $0$。时间复杂度是 $O(mn)$ 的，因为每次决策都会把 $S$ 分成两个不交的集合，每个元素都只会向一侧递归，一共递归 $m$ 层。

具体地，Alice希望权值最小，而Bob希望权值最大，因此在每一轮操作中，他们都会尽可能地选择最优的策略。我们可以构建一个数据结构来模拟每轮操作后的状态。为了满足这个需求，我们选择使用二叉树来构建这个模型。

**数据结构设计**

在代码中，我们定义了一个二叉树`Tree`，其中`lc`和`rc`数组分别表示树中每个节点的左子节点和右子节点。`sum`数组存储当前子树中所有节点的权值和。

**插入操作**

`insert`方法用于在树中插入一个新值。首先，我们检查当前节点是否存在，如果不存在，我们就创建一个新节点。然后，根据当前深度和给定的轮数参数`b`，我们决定应该在左子树还是右子树中继续插入。这里的逻辑是，如果当前值是`b[dep]`的倍数，我们就在右子树中继续插入，否则在左子树中插入。

**查询操作**

`query`方法用于查询在当前轮数下的最大或最小权值。首先，我们检查当前节点是否存在，如果不存在，直接返回0。然后，我们在左子树和右子树中分别查询权值，并根据当前轮数的奇偶性返回对应的值。如果当前轮数是奇数，表示Alice正在操作，我们返回两个权值中的较小值；否则，表示Bob正在操作，我们返回两个权值中的较大值。

**主函数流程**

1. 首先，我们读取输入数据，包括初始集合的大小`n`，操作轮数`m`，初始集合`a`和每轮的参数`b`。
2. 接下来，我们将初始集合中的所有数字插入到树中。

3. 最后，我们查询并输出在所有轮数结束后的权值。

由于每次决策都会把集合分成两个不交的集合，因此每个元素在递归中只会被处理一次。因此，这个算法的时间复杂度为$O(mn)$。

## 100% 结论分析

考虑对于先手来说，每次进行决策，如果他选择集合大小更小的那一侧，那么每次操作集合大小至少减半，只需要 $log_n$ 次操作就必定可以使答案为 $0$，考虑上“两人是轮流操作”的这个因素，所以当 $m > 2 * log_n$ 的时候，答案不可能大于 $0$。同理答案也不可能小于 $0$（因为后手可以采取同样的策略），因此答案一定是 $0$。于是只有在$m < 2 * log_n$的时候才考虑进行搜索，保证了时间复杂度为 $O(nlogn)$。

```cpp
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

template <class T = ll> T read() {
    T ret = 0, _f = 1;
    char ch = getchar();
    while (!isdigit(ch)) _f = (ch == '-' ? -1 : _f), ch = getchar();
    while (isdigit(ch)) ret = ret * 10 + (ch ^ 48), ch = getchar();
    return ret * _f;
}

// 定义最大的集合和操作轮数
constexpr int N(20005);
constexpr int M(200005);

int n, m;
ll a[N], b[M], rt;

// 定义一棵二叉树用于存储数据和进行查询
struct Tree {
    ll sum[N * 30], lc[N * 30], rc[N * 30], tot;  // lc, rc表示左右子节点

    // 插入数据到树中
    void insert(ll& p, ll dep, ll val) {
        if (!p) p = ++tot;
        if (dep > m) return sum[p] += val, void();
        insert((val % b[dep] ? lc : rc)[p], dep + 1, val);  // 根据倍数进行左右分支
    }

    // 查询权值
    ll query(ll p, ll dep) {
        if (!p) return 0ll;
        if (dep > m) return sum[p];
        ll foo = query(lc[p], dep + 1);
        ll bar = query(rc[p], dep + 1);
        // 根据操作轮数的奇偶性返回权值
```

```
        return (dep & 1) ? min(foo, bar) : max(foo, bar);
    }
} tr;

int main() {
    // 输入数据
    n = read(), m = read();
    if (m > 28) return printf("0\n"), 0;  // 当操作次数超过28轮时，答案是0
    for (int i = 1; i <= n; ++i) a[i] = read();
    for (int i = 1; i <= m; ++i) b[i] = read();

    // 将数据插入到树中
    for (int i = 1; i <= n; ++i) tr.insert(rt, 1ll, a[i]);
    // 输出查询的权值
    printf("%lld\n", tr.query(rt, 1ll));
    return 0;
}
```

# T3 连通块

## 题意解释

给出一章$n$个点的图，$u$与$v$之间有边的条件是$gcd(u, v)$为合数。现要删除一个点，使得最大连通块最小。

## 考察知识点：

tarjan、欧拉筛预处理、图论建模

### 16%: $n \leq 300$

按题意暴力建边模拟即可。

#### 1. 基本思路

对于小规模的数据，我们可以不考虑优化，而直接模拟题目的过程来解决。具体来说：

1. 首先，我们根据给定的点权计算出所有满足条件（gcd为合数）的点对，然后在这两点之间建边。
2. 然后，我们尝试删除每一个节点，并计算其对最大连通块大小的影响。
3. 我们需要找到那个删除后剩下的最大连通块最小的节点，以及这个最小的大小。

#### 2. 步骤详解

##### 步骤1：建边

对于每两个节点i和j，计算它们点权的gcd。如果这个gcd是合数，那么在这两点之间加一条边。

```
for (int i = 1; i <= n; ++i) {
    for (int j = i + 1; j <= n; ++j) {
        if (gcd(points[i], points[j]) > 1) {
            // 这里，points表示点权数组
            add_edge(i, j); // 建立从i到j的边
        }
    }
}
```

##### 步骤2：模拟删除节点

对于每一个节点i，我们尝试删除它，并查看剩余图中的连通块大小。

```
int ans = INT_MAX; // 记录最终答案，初始为最大值
for (int i = 1; i <= n; ++i) {
    int maxSize = getMaxConnectedComponentSizeAfterRemoving(i);
    // 获取删除节点i后的最大连通块大小
    ans = min(ans, maxSize); // 更新答案
}
```

其中，`getMaxConnectedComponentSizeAfterRemoving(int node)` 函数意为 `获取删除节点后的最大连通块`，可以使用DFS或BFS从任意未被访问的节点开始，遍历整个连通块来实现。

##### 步骤3：输出答案

简单地输出第2步中计算得到的 `ans` 。

**3. 总结**

暴力解法的时间复杂度主要取决于两个部分：建边部分和删除节点部分。

1. 建边部分的时间复杂度为 $O(n^2)$。
2. 删除节点并计算最大连通块的时间复杂度也是 $O(n^2)$。

所以，暴力解法的总时间复杂度是 $O(n^2)$。当$n$的值较小，例如$n <= 300$，这种方法是可以接受的。但当 $n$ 增大，我们就需要更好的优化方法来加速计算。

# 2. 优化枚举

对于稍大的数据，直接的暴力方法效率不足，首先，题解提出的问题是：建立边和枚举删除点是两个效率瓶颈。

其中枚举删除点的优化方法是基于一个观察：删除的节点肯定是最大连通块的一个割点。我们可以先使用并查集找到最大连通块，然后用 tarjan 算法来求出每个割点删除后的最大连通块大小。

**1. 使用并查集找到最大的连通块：**

初始化每个节点为一个单独的集合。对于每两个节点，如果它们之间有边，那么合并这两个集合。最后，检查每个集合的大小，找到最大的那个连通块。

**2. 使用Tarjan算法寻找最大连通块中的割点：**

运行Tarjan算法来确定哪些节点是割点。

**3. 对每个割点模拟其删除过程：**

现在，我们知道了哪些节点是割点，所以只需要考虑这些点，模拟它们的删除过程，看看删除后的最大连通块的大小是多少。这明显减少了需要考虑的节点数量。

# 3. 优化建边

对于更大的数据规模，上述方法中建边的效率仍然不足。考虑优化建边。

我们先定义一个数是单位合数，当且仅当它能够表示为 $2$ 个质数的乘积。

考察联通的性质。如果两个点之间存在一条边，那么可以建一个虚拟节点（这个节点应是两个点的单位合数），向两个点连边，这跟原图在联通性上是等价的。

因此我们定义单位合数为两个质数的乘积，数量为 $O(log^2V)$ 级别的， 其中 $V$ 是值域。

若任意两个数的 $gcd$ 为合数，那么必定有至少一个单位合数同时是两者的因数，于是它们通过这个单位合数联通。

此时边的数量就是 $O(nlog^2V)$ 的了，剩下的按 40% 的做法即可。

```cpp
#include<bits/stdc++.h>
using namespace std;
#define pii pair<int, int>
#define mp make_pair
#define pb push_back
#define fi first
#define se second
#define ull unsigned long long
inline int read() {
	int x = 0, f = 1; char c = getchar();
	while (c < '0' || c > '9') {if (c == '-') f = -f; c = getchar();}
	while (c >= '0' && c <= '9') {x = (x << 3) + (x << 1) + (c ^ 48); c = getchar();}
	return x * f;
}
const int N = 4e6 + 10, M = 1e7 + 10;
int n;
// 欧拉筛变量和函数，用于获取质数以及每个数的最小质因子
int ct, pri[M], id[M], mn[M]; bool vis[M];
inline void seive() {
	for (int i = 2; i < M; ++ i) {
		if (!vis[i]) pri[++ ct] = i, mn[i] = i;
		for (int j = 1; j <= ct && i * pri[j] < M; ++ j) {
			vis[i * pri[j]] = 1; mn[i * pri[j]] = pri[j];
			if (i % pri[j] == 0) break;
		}
	} ct = 0;
	for (int i = 2; i < M; ++ i) if (vis[i] && !vis[i/mn[i]]) id[i] = ++ ct;
}

int tot = 1, hd[N], nxt[M], ver[M];
inline void add_edge(int u, int v) {ver[++tot] = v; nxt[tot] = hd[u]; hd[u] = tot;}

int f[N], g[N];
inline int find(int x) {return x == f[x] ? x : f[x] = find(f[x]);}
inline void merge(int x, int y) {
	x = find(x), y = find(y);
	if (x == y) return;
	f[x] = y; g[y] += g[x];
}
```

```
int tim, dfn[N], low[N], siz[N];
int top, stc[N];

inline void init() {
        for (int i = 1; i <= n+ct; ++ i) f[i] = i, g[i] = (i <= n);
        memset(dfn, 0, sizeof dfn); tim = top = 0;
        tot = 1; memset(hd, 0, sizeof hd);
}
int mx, se, mxp;
inline void upd(int v, int x) {
        if (v > mx) se = mx, mx = v, mxp = x;
        else if (v > se) se = v;
}

int ans, all;
inline void tarjan(int u, int eid) {
        dfn[u] = low[u] = ++ tim; stc[++ top] = u; siz[u] = 0;
        int ret = 0, flg = 0;
        for (int i = hd[u]; i; i = nxt[i]) {
                if (i == (eid ^ 1)) continue;
                int v = ver[i];
                if (!dfn[v]) {
                        tarjan(v, i); low[u] = min(low[u], low[v]);
                        if (low[v] >= dfn[u]) flg ++;
                        if (low[v] > dfn[u]) top --, ret = max(ret, siz[v]), siz[u] += siz[v];
                        else if (low[v] == dfn[u]) {
                                int t, s = 0;
                                do {
                                        t = stc[top --];
                                        siz[u] += siz[t]; s += siz[t];
                                } while (t != v);
                                ret = max(ret, s);
                        }
                } else low[u] = min(low[u], dfn[v]);
        } if (u <= n) {
                siz[u] ++;
                if ((eid && flg) || (!eid && flg > 1)) ans = min(ans, max(ret, all - siz[u]));
                else ans = min(ans, all - 1);
```

```
        }
}

inline void solve() {
        n = read(); init();
        for (int i = 1, x; i <= n; ++ i) {
        // 对每个数的质因子进行处理，建立图
                int p[15] = {0}, c[15] = {0};
                x = read();
                while (x > 1) {
                        int w = mn[x];
                        p[++p[0]] = w;
                        while (x % w == 0) c[p[0]] ++, x /= w;
                }
                for (int j = 1; j <= p[0]; ++ j)
                        for (int k = j + (c[j] <= 1); k <= p[0]; ++ k)
                                if (1ll * p[j] * p[k] < M) {
                                        int v = id[p[j] * p[k]] + n;
                                        add_edge(i, v); add_edge(v, i);
                                        merge(i, v);
                                }
        }
    // 对图中的连通块进行处理，更新答案
        mx = 0, se = 0, mxp = 0;
        for (int i = 1; i <= ct + n; ++ i) {
                if (f[i] != i || !g[i]) continue;
                upd(g[i], i);
        } ans = all = mx; tarjan(mxp, 0);
        printf("%d\n", max(ans, se));
}

int main () {
        freopen("connect.in", "r", stdin);
        freopen("connect.out", "w", stdout);
        seive();
        int T = read(); while (T --) solve();
        return 0;
}
```

# T4 公交路线

## 题意解释

一棵$n$个节点的树，每个节点有类型$c_i$。给出$m$个询问，每个询问将临时破坏一个点，选出两条简单路径，要求如下：
1.两条路径不经过重复点(包括被破坏的点)

2.一条路径的两个端点必须同一类型

3.被破坏的点不可作为端点

对于每个询问，输出符合条件的路径方案数，每个询问独立。

## 考察知识点：

树上计数（算贡献思维）、LCA、虚树

## 题目分析

首先考虑固定其中一条路径，选择另一条路径的方案数。记 $f_u$ 为 $u$ 子树内的路径数量，等于每种颜色个数的 $\binom{x}{2}$ 之和。记 $g_u$ 为 $u$ 子树外的路径数量，为总路径数量 $-f_u-$ 跨过 $u$ 的路径数量。容易使用虚树 / dsu on tree / 线段树合并 等解决。

记 $s_u$ 为 $u$ 所有儿子的 $f_v$ 的和，则固定一条路径 $(u,v)$，选择另一条路径的方案数 $F(u,v)$ 为 $\sum\limits_{i\neq w} s_i - \sum\limits_{i\neq w} f_i + s_w + g_w$，可以整理成 $A_u + A_v + B_w$。其中 $w$ 为 $u,v$ 的 LCA。

每次询问要求点 $u$ 不能作为一个端点，则需要求出 $h_u$ 表示 $\sum\limits_{v,c_v=c_u} F(u,v)$。总答案为 $\dfrac{\sum h_i}{4}$

，每次询问减去一个 $h_u$ 即可。

求 $h_u$ 只需要把 $A_u, A_v, B_w$ 的贡献分开来算。$A_u, A_v$ 的贡献容易计算，$B_w$ 的贡献使用虚树比较方便，$w$ 一定在 $c_u$ 颜色构成的虚树上。令 $siz_i$ 表示 $i$ 子树内 $c_u$ 颜色的数量，$w$ 包含 $u$ 的儿子为 $v$，则 $w$ 对 $u$ 的贡献即为 $(siz_w - siz_v)B_w$。再计算 $u=w$ 时 $B_u$ 的贡献。虚树上 dfs 一遍即可完成。

```cpp
vi G[N], vg[N], vec[N], vtn[N];
ll f[N], sf[N], g[N], h[N], A[N], B[N], tmp[N];
ll sum, asr;
int col[N], dfn[N], nod[N], fa[N], siz[N];
int dep[N], son[N], top[N];
int seq[N * 2], stk[N], vf[N], csub[N];
int n, m, q, cnt;

ll C2(ll n) {
        return n * (n - 1) / 2;
}

bool cmp_dfn(int x, int y) {
        return dfn[x] < dfn[y];
}

bool in(int u, int v) {
        return dfn[u] <= dfn[v] && dfn[v] < dfn[u] + siz[u];
}

void dfs1(int u, int f) {
        siz[u] = 1;
        fa[u] = f;
        dep[u] = dep[f] + 1;
        for(int v : G[u]) {
                if(v == f) continue;
                dfs1(v, u);
                siz[u] += siz[v];
                if(siz[v] > siz[son[u]]) son[u] = v;
        }
}

void dfs2(int u, int tf) {
        top[u] = tf;
        dfn[u] = ++ cnt;
        nod[cnt] = u;
        if(son[u]) dfs2(son[u], tf);
        for(int v : G[u])
                if(!dfn[v]) dfs2(v, v);
```

```
}

int LCA(int x, int y) {
        for(; top[x] != top[y]; y = fa[top[y]])
                if(dep[top[x]] > dep[top[y]]) swap(x, y);
        return dep[x] < dep[y] ? x : y;
}

void getfg() {
        per(i, n, 2) {
                int u = nod[i];
                f[fa[u]] += f[u];
                g[fa[u]] += g[u];
        }
        rep(i, 1, n) {
                int u = nod[i];
                A[u] = A[fa[u]] - f[u];
                for(int v : G[u]) if(v != fa[u])
                        A[u] += f[v], B[u] += f[v];
                g[u] = asr - f[u] - g[u];
                B[u] += g[u] - A[u] * 2;
        }
}

void make_vtree(int c) {
        cnt = 0;
        for(int x : vec[c]) seq[++ cnt] = dfn[x];
        sort(seq + 1, seq + cnt + 1);
        rep(i, 2, cnt) seq[++ cnt] = dfn[LCA(nod[seq[i - 1]], nod[seq[i]])];
        sort(seq + 1, seq + cnt + 1);
        cnt = unique(seq + 1, seq + cnt + 1) - seq - 1;
        rep(i, 1, cnt) vtn[c].pb(nod[seq[i]]);
}

void getfa(int c) {
        cnt = 0;
        for(int x : vtn[c]) seq[++ cnt] = x;
        int top = 0, u;
        rep(i, 1, cnt) {
```

```
            u = seq[i];
            for(; top && !in(stk[top], u); -- top) ;
            vf[u] = stk[top];
            stk[++ top] = u;
        }
}

void solve_1(int c) {
        int all = vec[c].size(), u;
        rep(i, 1, cnt) csub[seq[i]] = 0;
        per(i, cnt, 1) {
                u = seq[i];
                csub[u] += col[u] == c;
                f[u] += C2(csub[u]);
                g[u] += 1ll * csub[u] * (all - csub[u]);
                if(vf[u]) {
                        csub[vf[u]] += csub[u];
                        f[vf[u]] -= C2(csub[u]);
                        g[vf[u]] -= 1ll * csub[u] * (all - csub[u]);
                }
        }
}

void solve_2(int c) {
        int u, m = vec[c].size();
        ll sumA = 0;
        if(m == 1) return ;
        rep(i, 1, cnt) {
                u = seq[i];
                csub[u] = tmp[u] = 0;
                if(col[u] == c) sumA += A[u];
        }
        per(i, cnt, 1) {
                u = seq[i];
                csub[u] += col[u] == c;
                if(vf[u]) csub[vf[u]] += csub[u];
        }
        rep(i, 2, cnt) {
                u = seq[i];
```

```
                tmp[u] = tmp[vf[u]] + B[vf[u]] * (csub[vf[u]] - csub[u]);
        }
        rep(i, 1, cnt) {
                u = seq[i];
                if(col[u] == c) h[u] = tmp[u] + A[u] * (m - 2) + sumA + B[u] * (csub[u] - 1);
        }
}

signed main() {
        freopen("route.in", "r", stdin);
        freopen("route.out", "w", stdout);
        int x, y;
        n = read(); q = read(); m = read();
        rep(i, 1, n) vec[col[i] = read()].pb(i);
        rep(i, 1, m) asr += C2(vec[i].size());
        rep(i, 2, n) {
                x = read(); y = read();
                G[x].pb(y); G[y].pb(x);
        }
        dfs1(1, 0); dfs2(1, 1);
        rep(i, 1, m) make_vtree(i);
        rep(i, 1, m) getfa(i), solve_1(i);
        getfg();
        rep(i, 1, m) getfa(i), solve_2(i);
        rep(i, 1, n) sum += h[i];
        sum /= 4;
        printf("%lld\n", sum);
        while(q --) printf("%lld\n", sum - h[read()]);
        return 0;
}
```