==========
文件夹: class177_MoAlgorithm
==========

[Markdown 文件]
==========
文件: README.md
==========
莫队算法专题 (Mo's Algorithm)

莫队算法是一种基于分块思想的离线算法，用于解决区间查询问题。它通过巧妙地排序查询和维护区间信息，能够在较优的时间复杂度内处理大量区间查询。

算法特点

- **离线算法**：需要预先知道所有查询
- **适用范围**：适用于可以O(1)时间扩展或收缩区间边界的问题
- **时间复杂度**：普通莫队O(n√n)，带修莫队O(n^(5/3))，回滚莫队O(n√m)

算法分类

1. 普通莫队 (Standard Mo's Algorithm)
适用于只有查询没有修改的问题。

核心思想：
1. 对查询进行特殊排序
2. 通过移动左右端点维护区间信息
3. 利用分块思想优化时间复杂度

时间复杂度：O(n√m)

代表题目：
- SP3267 DQUERY - D-query (Code14)
- P1494 [国家集训队] 小 Z 的袜子
- CF617E XOR and Favorite Number (Code19)

2. 带修莫队 (Mo's Algorithm with Modification)
适用于带单点修改的区间查询问题。

核心思想：
1. 在普通莫队基础上增加时间维度
2. 对查询按左端点块号、右端点块号、时间排序
3. 维护修改操作的时间顺序

时间复杂度：O(n^(5/3))

代表题目：
- P1903 [国家集训队] 数颜色 / 维护队列 (Code16)

3. 回滚莫队 (Rollback Mo's Algorithm)
适用于删除操作难以维护的问题。

核心思想：
1. 只使用增加操作或只使用删除操作
2. 对于增加操作，通过回滚技术处理
3. 对于删除操作，通过预处理技术处理

时间复杂度：O(n√m)

代表题目：
- AT_joisc2014_c 歴史の研究 (Code15)
- P5906 【模板】回滚莫队&不删除莫队

4. 树上莫队 (Mo's Algorithm on Tree)
适用于树上路径查询问题。

核心思想：
1. 将树转化为括号序
2. 利用括号序上的莫队算法处理
3. 特别处理 LCA 节点

时间复杂度：O(n√m)

代表题目：
- SP10707 COT2 - Count on a tree II (Code17)

5. 二次离线莫队 (Secondary Offline Mo's Algorithm)
适用于转移操作复杂度较高的问题。

核心思想：
1. 将转移操作离线处理
2. 通过扫描线等技术批量处理
3. 结合其他数据结构优化

时间复杂度：O(n√m + n√n)

代表题目：
- P4887 【模板】莫队二次离线（第十四分块(前体)）(Code18)

文件命名规范

所有代码文件采用`CodeXX_XXX.java/.cpp/.py`的命名格式：
- `XX`为两位数字序号
- `XXX`为题目英文名称或核心功能描述
- 每个题目包含 Java、C++、Python 三种语言实现

题目列表

普通莫队
1. **Code14_DQuery** - SP3267 DQUERY - D-query
   - 题目大意：查询区间不同元素个数
   - Java: Code14_DQuery1.java
   - C++: Code14_DQuery2.cpp
   - Python: Code14_DQuery3.py

2. **Code19_XorFavorite** - CF617E XOR and Favorite Number
   - 题目大意：查询区间内异或值等于 k 的子区间个数
   - Java: Code19_XorFavorite1.java
   - C++: Code19_XorFavorite2.cpp
   - Python: Code19_XorFavorite3.py

3. **Code20_CF1000F_OneOccurrence** - Codeforces 1000F One Occurrence
   - 题目大意：查询区间只出现一次的元素中的最左元素
   - Java: Code20_CF1000F_OneOccurrence1.java
   - C++: Code20_CF1000F_OneOccurrence2.cpp
   - Python: Code20_CF1000F_OneOccurrence3.py

回滚莫队
4. **Code15_HistoryResearch** - AT_joisc2014_c 歴史の研究
   - 题目大意：查询区间内重要度最大的数字
   - Java: Code15_HistoryResearch1.java
   - C++: Code15_HistoryResearch2.cpp
   - Python: Code15_HistoryResearch3.py

5. **Code22_MaxFrequency** - 洛谷 P5906 【模板】回滚莫队&不删除莫队 / Codeforces 86D Powerful array / 洛谷 P4137 Rmq Problem / mex
   - 题目大意：查询区间内元素出现次数的平方和 / 查询区间 mex 值
   - Java: Code22_MaxFrequency1.java
   - C++: Code22_MaxFrequency2.cpp

   - Python: Code22_MaxFrequency3.py

带修莫队
6. **Code16_ColorMaintenance** - P1903 [国家集训队] 数颜色 / 维护队列
   - 题目大意：维护序列，支持查询区间不同颜色数和单点修改
   - Java: Code16_ColorMaintenance1.java
   - C++: Code16_ColorMaintenance2.cpp
   - Python: Code16_ColorMaintenance3.py

7. **Code21_TimeTravelQueries** - Codeforces 246E Blood Cousins Return / HDU 6629 string matching
   - 题目大意：支持时间旅行的区间查询问题 / 字符串匹配与修改
   - Java: Code21_TimeTravelQueries1.java
   - C++: Code21_TimeTravelQueries2.cpp
   - Python: Code21_TimeTravelQueries3.py

树上莫队
8. **Code17_TreeMo** - SP10707 COT2 - Count on a tree II
   - 题目大意：查询树上两点间路径不同权值个数
   - Java: Code17_TreeMo1.java
   - C++: Code17_TreeMo2.cpp
   - Python: Code17_TreeMo3.py

9. **Code23_TreePathQueries** - Codeforces 375D Tree and Queries / HDU 6604 Blow up the city
   - 题目大意：查询树上路径中出现次数大于等于 k 的颜色数 / 图的连通性查询
   - Java: Code23_TreePathQueries1.java
   - C++: Code23_TreePathQueries2.cpp
   - Python: Code23_TreePathQueries3.py

二次离线莫队
10. **Code18_SecondaryOffline** - P4887 【模板】莫队二次离线
    - 题目大意：查询区间内满足异或条件的二元组个数
    - Java: Code18_SecondaryOffline1.java
    - C++: Code18_SecondaryOffline2.cpp
    - Python: Code18_SecondaryOffline3.py

算法技巧总结

1. 排序策略
- 普通莫队：按左端点块号为第一关键字，右端点为第二关键字
- 左端点块号相同时，奇数块按右端点升序，偶数块按右端点降序

2. 指针移动顺序
```
```

```
// 扩展右端点
while (winr < jobr) add(arr[++winr]);

// 扩展左端点
while (winl > jobl) add(arr[--winl]);

// 收缩左端点
while (winl < jobl) del(arr[winl++]);

// 收缩右端点
while (winr > jobr) del(arr[winr--]);
```

3. 块大小选择
- 普通莫队：√n
- 带修莫队：n^(2/3)
- 根据具体题目调整以获得最优性能

应用场景

莫队算法适用于以下场景：
1. 区间查询问题
2. 可以O(1)时间扩展或收缩区间边界
3. 允许离线处理
4. 数据规模适中（通常n,m ≤ 10^5）

与其他算法的比较

算法	时间复杂度	空间复杂度	优点	缺点
线段树	O(nlogn)	O(n)	在线、功能强大	实现复杂
树状数组	O(nlogn)	O(n)	实现简单、常数小	功能有限
莫队算法	O(n√n)	O(n)	实现简单、适用范围广	离线、常数较大

学习建议

1. **掌握基础**：先熟练掌握普通莫队算法
2. **理解本质**：理解分块思想和指针移动的原理
3. **练习变种**：逐步学习带修莫队、回滚莫队等变种
4. **实战应用**：通过大量题目加深理解
5. **优化技巧**：学习各种优化技巧，如奇偶优化等

常见优化技巧

1. **奇偶优化**：同一块内奇数行升序，偶数行降序
2. **块大小调整**：根据题目特点调整块大小
3. **IO 优化**：使用快速输入输出
4. **常数优化**：减少不必要的计算和内存访问

注意事项

1. **指针顺序**：严格按照扩展-收缩的顺序移动指针
2. **边界处理**：注意数组边界和初始化
3. **离散化**：对于权值范围大的题目考虑离散化
4. **数据结构**：根据题目特点选择合适的计数数据结构

==========

[代码文件]
==========
文件: Code01_NiceDay1.java
==========
```
// 美好的每一天 - 普通莫队算法实现 (Java 版本)
// 题目来源: 洛谷 P3604 美好的每一天
// 题目链接: https://www.luogu.com.cn/problem/P3604
// 题目大意: 给定一个长度为 n 的字符串 str，如果一个子串重新排列字符之后能成为回文串，那么该子串叫做达标子串。
// 接下来有 m 条查询，格式为 l r : 打印 str[l..r]范围上有多少达标子串
// 解题思路: 使用普通莫队算法，通过维护字符出现次数的奇偶性来判断是否能构成回文串
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. 洛谷 P3604 美好的每一天 - https://www.luogu.com.cn/problem/P3604
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code01_NiceDay1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code01_NiceDay2.java
//
// 2. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
//
// 3. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
```

//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 4. 洛谷 P1494 [国家集训队]小 Z 的袜子 - https://www.luogu.com.cn/problem/P1494
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 5. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 6. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 7. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance3.py
//
// 8. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
//
// 9. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch3.py
//
// 10. 洛谷 P4887 【模板】莫队二次离线（第十四分块(前体)） -

```
https://www.luogu.com.cn/problem/P4887
//     Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline1.java
//     C++解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline2.cpp
//     Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline3.py

package class179;

// 美好的每一天，java 版
// 给定一个长度为 n 的字符串 str，其中都是小写字母
// 如果一个子串重新排列字符之后能成为回文串，那么该子串叫做达标子串
// 接下来有 m 条查询，格式为 l r : 打印 str[l..r]范围上有多少达标子串
// 1 <= n、m <= 6 * 10^4
// 测试链接 : https://www.luogu.com.cn/problem/P3604
// 提交以下的 code，提交时请把类名改成"Main"
// java 实现的逻辑一定是正确的，但是本题卡常，无法通过所有测试用例
// 想通过用 C++实现，本节课 Code01_NiceDay2 文件就是 C++的实现
// 两个版本的逻辑完全一样，C++版本可以通过所有测试

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.util.Arrays;
import java.util.Comparator;

public class Code01_NiceDay1 {

    public static int MAXN = 60002;
    public static int MAXV = 1 << 26;
    public static int n, m;
    public static int[] arr = new int[MAXN];
    public static int[][] query = new int[MAXN][3];
    public static int[] bi = new int[MAXN];

    public static int[] cnt = new int[MAXV];
    public static long curAns = 0;
    public static long[] ans = new long[MAXN];

    public static class QueryCmp implements Comparator<int[]> {
        @Override
```

```
    public int compare(int[] a, int[] b) {
        if (bi[a[0]] != bi[b[0]]) {
            return bi[a[0]] - bi[b[0]];
        }
        if ((bi[a[0]] & 1) == 1) {
            return a[1] - b[1];
        } else {
            return b[1] - a[1];
        }
    }
}

public static void add(int s) {
    curAns += cnt[s];
    cnt[s]++;
    for (int i = 0; i < 26; i++) {
        curAns += cnt[s ^ (1 << i)];
    }
}

public static void del(int s) {
    cnt[s]--;
    curAns -= cnt[s];
    for (int i = 0; i < 26; i++) {
        curAns -= cnt[s ^ (1 << i)];
    }
}

public static void compute() {
    int winl = 1, winr = 0;
    for (int i = 1; i <= m; i++) {
        int jobl = query[i][0];
        int jobr = query[i][1];
        int id = query[i][2];
        while (winl > jobl) {
            add(arr[--winl]);
        }
        while (winr < jobr) {
            add(arr[++winr]);
        }
        while (winl < jobl) {
            del(arr[winl++]);
        }
```

```
        while (winr > jobr) {
            del(arr[winr--]);
        }
        ans[id] = curAns;
    }
}

public static void prepare() {
    for (int i = 1; i <= n; i++) {
        arr[i] ^= arr[i - 1];
    }
    for (int i = n; i >= 0; i--) {
        arr[i + 1] = arr[i];
    }
    n++;
    int blen = (int) Math.sqrt(n);
    for (int i = 1; i <= n; i++) {
        bi[i] = (i - 1) / blen + 1;
    }
    for (int i = 1; i <= m; i++) {
        query[i][1]++;
    }
    Arrays.sort(query, 1, m + 1, new QueryCmp());
}

public static void main(String[] args) throws Exception {
    FastReader in = new FastReader();
    PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));
    n = in.nextInt();
    m = in.nextInt();
    char c;
    for (int i = 1; i <= n; i++) {
        c = in.nextLowerCase();
        arr[i] = 1 << (c - 'a');
    }
    for (int i = 1; i <= m; i++) {
        query[i][0] = in.nextInt();
        query[i][1] = in.nextInt();
        query[i][2] = i;
    }
    prepare();
    compute();
    for (int i = 1; i <= m; i++) {
```

```
        out.println(ans[i]);
    }
    out.flush();
    out.close();
}

// 读写工具类
static class FastReader {
    final private int BUFFER_SIZE = 1 << 16;
    private final InputStream in;
    private final byte[] buffer;
    private int ptr, len;

    public FastReader() {
        in = System.in;
        buffer = new byte[BUFFER_SIZE];
        ptr = len = 0;
    }

    private boolean hasNextByte() throws IOException {
        if (ptr < len)
            return true;
        ptr = 0;
        len = in.read(buffer);
        return len > 0;
    }

    private byte readByte() throws IOException {
        if (!hasNextByte())
            return -1;
        return buffer[ptr++];
    }

    public char nextLowerCase() throws IOException {
        int c;
        while (true) {
            c = readByte();
            if (c >= 'a' && c <= 'z')
                return (char) c;
        }
    }

    int nextInt() throws IOException {
```

```
            int c;
            do {
                c = readByte();
            } while (c <= ' ' && c != -1);
            boolean neg = false;
            if (c == '-') {
                neg = true;
                c = readByte();
            }
            int val = 0;
            while (c > ' ' && c != -1) {
                val = val * 10 + (c - '0');
                c = readByte();
            }
            return neg ? -val : val;
        }
    }

}
        private boolean hasNextByte() throws IOException {
            if (ptr < len)
                return true;
            ptr = 0;
            len = in.read(buffer);
            return len > 0;
        }

        private byte readByte() throws IOException {
            if (!hasNextByte())
                return -1;
            return buffer[ptr++];
        }

        public char nextLowerCase() throws IOException {
            int c;
            while (true) {
                c = readByte();
                if (c >= 'a' && c <= 'z')
                    return (char) c;
            }
        }

        int nextInt() throws IOException {
```

```
            int c;
            do {
                c = readByte();
            } while (c <= ' ' && c != -1);
            boolean neg = false;
            if (c == '-') {
                neg = true;
                c = readByte();
            }
            int val = 0;
            while (c > ' ' && c != -1) {
                val = val * 10 + (c - '0');
                c = readByte();
            }
            return neg ? -val : val;
        }
    }

}
```

==========

文件: Code01_NiceDay2.java

==========

```
// 美好的每一天 - 普通莫队算法实现 (C++版本)
// 题目来源: 洛谷 P3604 美好的每一天
// 题目链接: https://www.luogu.com.cn/problem/P3604
// 题目大意: 给定一个长度为 n 的字符串 str，如果一个子串重新排列字符之后能成为回文串，那么该子串叫做达标子串。
// 接下来有 m 条查询，格式为 l r : 打印 str[l..r]范围上有多少达标子串
// 解题思路: 使用普通莫队算法，通过维护字符出现次数的奇偶性来判断是否能构成回文串
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
// 相关题目:
// 1. CF617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
// 3. BZOJ2038 [国家集训队]小 Z 的袜子 - https://www.lydsy.com/JudgeOnline/problem.php?id=2038
// 4. HDU4638 Group - http://acm.hdu.edu.cn/showproblem.php?pid=4638
// 5. AtCoder ABC176 D - Wizard in Maze - https://atcoder.jp/contests/abc176/tasks/abc176_d

package class179;

// 美好的每一天，C++版
```

```
// 给定一个长度为n的字符串str，其中都是小写字母
// 如果一个子串重新排列字符之后能成为回文串，那么该子串叫做达标子串
// 接下来有m条查询，格式为 l r : 打印str[l..r]范围上有多少达标子串
// 1 <= n、m <= 6 * 10^4
// 测试链接 : https://www.luogu.com.cn/problem/P3604
// 如下实现是C++的版本，C++版本和java版本逻辑完全一样
// 提交如下代码，可以通过所有测试用例

//#include <bits/stdc++.h>
//
//using namespace std;
//
//struct Query {
//    int l, r, id;
//};
//
//const int MAXN = 60002;
//const int MAXV = 1 << 26;
//int n, m;
//int arr[MAXN];
//Query query[MAXN];
//
//int bi[MAXN];
//int cnt[MAXV];
//long long curAns = 0;
//long long ans[MAXN];
//
//bool QueryCmp(Query &a, Query &b) {
//    if (bi[a.l] != bi[b.l]) {
//        return bi[a.l] < bi[b.l];
//    }
//    if (bi[a.l] & 1) {
//        return a.r < b.r;
//    } else {
//        return a.r > b.r;
//    }
//}
//
//void add(int s) {
//    curAns += cnt[s];
//    cnt[s]++;
//    for (int i = 0; i < 26; i++) {
//        curAns += cnt[s ^ (1 << i)];
```

```
//    }
//}
//
//void del(int s) {
//    cnt[s]--;
//    curAns -= cnt[s];
//    for (int i = 0; i < 26; i++) {
//        curAns -= cnt[s ^ (1 << i)];
//    }
//}
//
//void compute() {
//    int winl = 1, winr = 0;
//    for (int i = 1; i <= m; i++) {
//        int jobl = query[i].l;
//        int jobr = query[i].r;
//        int id = query[i].id;
//        while (winl > jobl) {
//            add(arr[--winl]);
//        }
//        while (winr < jobr) {
//            add(arr[++winr]);
//        }
//        while (winl < jobl) {
//            del(arr[winl++]);
//        }
//        while (winr > jobr) {
//            del(arr[winr--]);
//        }
//        ans[id] = curAns;
//    }
//}
//
//void prepare() {
//    for (int i = 1; i <= n; i++) {
//        arr[i] ^= arr[i - 1];
//    }
//    for (int i = n; i >= 0; i--) {
//        arr[i + 1] = arr[i];
//    }
//    n++;
//    int blen = (int)sqrt(n);
//    for (int i = 1; i <= n; i++) {
```

```
//        bi[i] = (i - 1) / blen + 1;
//    }
//    for (int i = 1; i <= m; i++) {
//        query[i].r++;
//    }
//    sort(query + 1, query + m + 1, QueryCmp);
//}
//
//int main() {
//    ios::sync_with_stdio(false);
//    cin.tie(nullptr);
//    cin >> n >> m;
//    char c;
//    for (int i = 1; i <= n; i++) {
//        cin >> c;
//        arr[i] = 1 << (c - 'a');
//    }
//    for (int i = 1; i <= m; i++) {
//        cin >> query[i].l >> query[i].r;
//        query[i].id = i;
//    }
//    prepare();
//    compute();
//    for (int i = 1; i <= m; i++) {
//        cout << ans[i] << '\n';
//    }
//    return 0;
//}
```

==========

文件: Code02_SimpleQuery1.java

==========

```
// 简单的询问 - 普通莫队算法实现 (Java 版本)
// 题目来源: LibreOJ #6271. 「LibreOJ NOI Round #1」简单的询问
// 题目链接: https://loj.ac/p/6271
// 题目大意: 给定一个长度为 n 的数组，每次查询区间[l,r]内满足条件的二元组个数
// 解题思路: 使用普通莫队算法，通过维护区间信息来回答查询
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. LibreOJ #6271. 「LibreOJ NOI Round #1」简单的询问 - https://loj.ac/p/6271
```

```
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code02_SimpleQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code02_SimpleQuery2.java
//
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 3. 洛谷 P1494 [国家集训队]小 Z 的袜子 - https://www.luogu.com.cn/problem/P1494
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 4. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//    Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
//    C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
//    Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
//
// 5. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//    Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//    Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 6. 洛谷 P1972 [SDOI2009] HH 的项链 - https://www.luogu.com.cn/problem/P1972
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery3.py
//
// 7. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 8. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
//    Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance1.java
//    C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance2.cpp
```

```
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance3.py
//
// 9. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
//
// 10. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
//     Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch1.java
//     C++解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch2.cpp
//     Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch3.py

package class179;

// 简单的询问，java 版
// 给定一个长度为 n 的数组 arr，下标从 1 到 n
// 函数 get(l, r, x) = arr[l..r]范围上，数组 x 出现的次数
// 接下来有 q 条查询，格式如下
// 查询 l1 r1 l2 r2 : 每种 x 都算，打印 get(l1, r1, x) * get(l2, r2, x) 的累加和
// 1 <= n、q <= 5 * 10^4
// 1 <= arr[i] <= n
// 测试链接 : https://www.luogu.com.cn/problem/P5268
// 提交以下的 code，提交时请把类名改成"Main"，可以通过所有测试用例

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.util.Arrays;
import java.util.Comparator;

public class Code02_SimpleQuery1 {

    public static int MAXN = 50001;
    public static int n, q, cntq;
    public static int[] arr = new int[MAXN];
    // 查询任务，siz1、siz2、op、id
    public static int[][] query = new int[MAXN << 2][4];
    public static int[] bi = new int[MAXN];
```

```
// cnt1 : arr[1..siz1]范围内每种数字出现的次数
// cnt2 : arr[1..siz2]范围内每种数字出现的次数
public static int[] cnt1 = new int[MAXN];
public static int[] cnt2 = new int[MAXN];
public static long curAns = 0;

public static long[] ans = new long[MAXN];

public static class QueryCmp implements Comparator<int[]> {
    @Override
    public int compare(int[] a, int[] b) {
        if (bi[a[0]] != bi[b[0]]) {
            return bi[a[0]] - bi[b[0]];
        }
        if ((bi[a[0]] & 1) == 1) {
            return a[1] - b[1];
        } else {
            return b[1] - a[1];
        }
    }
}

public static void addQuery(int siz1, int siz2, int op, int id) {
    query[++cntq][0] = siz1;
    query[cntq][1] = siz2;
    query[cntq][2] = op;
    query[cntq][3] = id;
}

// win1 和 win2 不代表一段区间，而是代表两个独立区域各自去覆盖 arr
// win1 = 0，win2 = 0，表示两个覆盖区域一开始都没有数字
// job1 和 job2 也不代表区间，而是代表两个区域各自要覆盖多大
public static void compute() {
    int win1 = 0, win2 = 0;
    for (int i = 1; i <= cntq; i++) {
        int job1 = query[i][0];
        int job2 = query[i][1];
        int op = query[i][2];
        int id = query[i][3];
        while (win1 < job1) {
            win1++;
            cnt1[arr[win1]]++;
```

```
            curAns += cnt2[arr[win1]];
        }
        while (win1 > job1) {
            cnt1[arr[win1]]--;
            curAns -= cnt2[arr[win1]];
            win1--;
        }
        while (win2 < job2) {
            win2++;
            cnt2[arr[win2]]++;
            curAns += cnt1[arr[win2]];
        }
        while (win2 > job2) {
            cnt2[arr[win2]]--;
            curAns -= cnt1[arr[win2]];
            win2--;
        }
        ans[id] += curAns * op;
    }
}

public static void prepare() {
    int blen = (int) Math.sqrt(n);
    for (int i = 1; i <= n; i++) {
        bi[i] = (i - 1) / blen + 1;
    }
    for (int i = 1; i <= cntq; i++) {
        if (query[i][0] > query[i][1]) {
            int tmp = query[i][0];
            query[i][0] = query[i][1];
            query[i][1] = tmp;
        }
    }
    Arrays.sort(query, 1, cntq + 1, new QueryCmp());
}

public static void main(String[] args) throws Exception {
    FastReader in = new FastReader(System.in);
    PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));
    n = in.nextInt();
    for (int i = 1; i <= n; i++) {
        arr[i] = in.nextInt();
    }
```

```
        q = in.nextInt();
        for (int i = 1, l1, r1, l2, r2; i <= q; i++) {
            l1 = in.nextInt();
            r1 = in.nextInt();
            l2 = in.nextInt();
            r2 = in.nextInt();
            addQuery(r1, r2, 1, i);
            addQuery(r1, l2 - 1, -1, i);
            addQuery(l1 - 1, r2, -1, i);
            addQuery(l1 - 1, l2 - 1, 1, i);
        }
        prepare();
        compute();
        for (int i = 1; i <= q; i++) {
            out.println(ans[i]);
        }
        out.flush();
        out.close();
    }

    // 读写工具类
    static class FastReader {
        private final byte[] buffer = new byte[1 << 16];
        private int ptr = 0, len = 0;
        private final InputStream in;

        FastReader(InputStream in) {
            this.in = in;
        }

        private int readByte() throws IOException {
            if (ptr >= len) {
                len = in.read(buffer);
                ptr = 0;
                if (len <= 0)
                    return -1;
            }
            return buffer[ptr++];
        }

        int nextInt() throws IOException {
            int c;
            do {
```

```
            c = readByte();
        } while (c <= ' ' && c != -1);
        boolean neg = false;
        if (c == '-') {
            neg = true;
            c = readByte();
        }
        int val = 0;
        while (c > ' ' && c != -1) {
            val = val * 10 + (c - '0');
            c = readByte();
        }
        return neg ? -val : val;
    }
}

}
```

==========

文件: Code02_SimpleQuery2.java

==========

```
// 简单的询问 - 普通莫队算法实现 (C++版本)
// 题目来源: 洛谷 P5268 简单的询问
// 题目链接: https://www.luogu.com.cn/problem/P5268
// 题目大意: 给定一个长度为 n 的数组 arr，函数 get(l, r, x) = arr[l..r]范围上，数组 x 出现的次数。
// 接下来有 q 条查询，格式如下：查询 l1 r1 l2 r2 : 每种 x 都算，打印 get(l1, r1, x) * get(l2, r2,
x) 的累加和
// 解题思路: 使用普通莫队算法，通过二维莫队的思想处理四个端点的移动
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
// 相关题目:
// 1. CF617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
// 3. BZOJ2038 [国家集训队]小 Z 的袜子 - https://www.lydsy.com/JudgeOnline/problem.php?id=2038
// 4. HDU4638 Group - http://acm.hdu.edu.cn/showproblem.php?pid=4638
// 5. AtCoder ABC176 D - Wizard in Maze - https://atcoder.jp/contests/abc176/tasks/abc176_d

package class179;

// 简单的询问，C++版
// 给定一个长度为 n 的数组 arr，下标从 1 到 n
// 函数 get(l, r, x) = arr[l..r]范围上，数组 x 出现的次数
```

```
// 接下来有 q 条查询，格式如下
// 查询 l1 r1 l2 r2 : 每种 x 都算，打印 get(l1, r1, x) * get(l2, r2, x) 的累加和
// 1 <= n、q <= 5 * 10^4
// 1 <= arr[i] <= n
// 测试链接 : https://www.luogu.com.cn/problem/P5268
// 如下实现是 C++的版本，C++版本和 java 版本逻辑完全一样
// 提交如下代码，可以通过所有测试用例

//#include <bits/stdc++.h>
//
//using namespace std;
//
//struct Query {
//    int siz1, siz2, op, id;
//};
//
//const int MAXN = 50001;
//int n, q, cntq;
//int arr[MAXN];
//Query query[MAXN << 2];
//int bi[MAXN];
//
//int cnt1[MAXN];
//int cnt2[MAXN];
//long long curAns = 0;
//long long ans[MAXN];
//
//bool QueryCmp(Query &a, Query &b) {
//    if (bi[a.siz1] != bi[b.siz1]) {
//        return bi[a.siz1] < bi[b.siz1];
//    }
//    if (bi[a.siz1] & 1) {
//        return a.siz2 < b.siz2;
//    } else {
//        return a.siz2 > b.siz2;
//    }
//}
//
//void addQuery(int siz1, int siz2, int op, int id) {
//    query[++cntq].siz1 = siz1;
//    query[cntq].siz2 = siz2;
//    query[cntq].op = op;
//    query[cntq].id = id;
```

```
//}
//
//void compute() {
//    int win1 = 0, win2 = 0;
//    for (int i = 1; i <= cntq; i++) {
//        int job1 = query[i].siz1;
//        int job2 = query[i].siz2;
//        int op = query[i].op;
//        int id = query[i].id;
//        while (win1 < job1) {
//            win1++;
//            cnt1[arr[win1]]++;
//            curAns += cnt2[arr[win1]];
//        }
//        while (win1 > job1) {
//            cnt1[arr[win1]]--;
//            curAns -= cnt2[arr[win1]];
//            win1--;
//        }
//        while (win2 < job2) {
//            win2++;
//            cnt2[arr[win2]]++;
//            curAns += cnt1[arr[win2]];
//        }
//        while (win2 > job2) {
//            cnt2[arr[win2]]--;
//            curAns -= cnt1[arr[win2]];
//            win2--;
//        }
//        ans[id] += curAns * op;
//    }
//}
//
//void prepare() {
//    int blen = (int)sqrt(n);
//    for (int i = 1; i <= n; i++) {
//        bi[i] = (i - 1) / blen + 1;
//    }
//    for (int i = 1; i <= cntq; i++) {
//        if (query[i].siz1 > query[i].siz2) {
//            swap(query[i].siz1, query[i].siz2);
//        }
//    }
```

```
//    sort(query + 1, query + cntq + 1, QueryCmp);
//}
//
//int main() {
//    ios::sync_with_stdio(false);
//    cin.tie(nullptr);
//    cin >> n;
//    for (int i = 1; i <= n; i++) {
//        cin >> arr[i];
//    }
//    cin >> q;
//    for (int i = 1, l1, r1, l2, r2; i <= q; i++) {
//        cin >> l1 >> r1 >> l2 >> r2;
//        addQuery(r1, r2, 1, i);
//        addQuery(r1, l2 - 1, -1, i);
//        addQuery(l1 - 1, r2, -1, i);
//        addQuery(l1 - 1, l2 - 1, 1, i);
//    }
//    prepare();
//    compute();
//    for (int i = 1; i <= q; i++) {
//        cout << ans[i] << '\n';
//    }
//    return 0;
//}
```

==========

文件: Code03_LCP1.java

==========

```
// 区间 lcp 达标对 - 普通莫队算法实现 (Java 版本)
// 题目来源: LibreOJ #6272. 「LibreOJ NOI Round #1」区间 lcp 达标对
// 题目链接: https://loj.ac/p/6272
// 题目大意: 给定一个字符串，每次查询区间[l,r]内满足条件的二元组个数
// 解题思路: 使用普通莫队算法，通过维护区间信息来回答查询
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. LibreOJ #6272. 「LibreOJ NOI Round #1」区间 lcp 达标对 - https://loj.ac/p/6272
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code03_LCP1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code03_LCP2.java
//
```

// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 3. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 4. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
//
// 5. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 6. 洛谷 P1972 [SDOI2009] HH的项链 - https://www.luogu.com.cn/problem/P1972
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery3.py
//
// 7. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 8. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance3.py
//

```
// 9. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
//
// 10. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
//     Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch1.java
//     C++解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch2.cpp
//     Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch3.py

package class179;

// 区间 lcp 达标对，java 版
// 给定一个长度为 n 的字符串 str，还有一个参数 k
// 位置 a 开头和位置 b 开头的字符串，如果最长公共前缀的长度 >= k，那么(a, b)构成 lcp 达标对
// 构成 lcp 达标对，必须是不同的位置，并且(a, b)和(b, a)只算一个达标对，不要重复统计
// 接下来有 m 条查询，格式为 l r : str[l..r]范围上，可以任选开头位置，打印 lcp 达标对的数量
// 1 <= n、k <= 3 * 10^6       1 <= m <= 10^5
// 1 <= n * n * m <= 10^15    字符集为 f z o u t s y
// 测试链接 : https://www.luogu.com.cn/problem/P5112
// 提交以下的 code，提交时请把类名改成"Main"，可以通过所有测试用例

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.util.Arrays;
import java.util.Comparator;

public class Code03_LCP1 {

    public static int MAXN = 3000001;
    public static int MAXM = 100001;
    public static int n, m, k;
    public static int len, cntq, cntv;
    public static char[] str = new char[MAXN];
    public static int[][] query = new int[MAXM][3];

    // 字符串哈希
    public static int base = 499;
```

```
public static long[] basePower = new long[MAXN];
public static long[] hashValue = new long[MAXN];

// 哈希值离散化，用哈希值替代字符串，再用排名替代哈希值
public static long[] val = new long[MAXN];
public static long[] sorted = new long[MAXN];
public static int[] arr = new int[MAXN];
public static int[] bi = new int[MAXN];

public static int[] cnt = new int[MAXN];
public static long curAns;
public static long[] ans = new long[MAXM];

public static int kth(long num) {
    int left = 1, right = cntv, mid, ret = 0;
    while (left <= right) {
        mid = (left + right) / 2;
        if (sorted[mid] <= num) {
            ret = mid;
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return ret;
}

public static class QueryCmp implements Comparator<int[]> {
    @Override
    public int compare(int[] a, int[] b) {
        if (bi[a[0]] != bi[b[0]]) {
            return bi[a[0]] - bi[b[0]];
        }
        if ((bi[a[0]] & 1) == 1) {
            return a[1] - b[1];
        } else {
            return b[1] - a[1];
        }
    }
}

public static void add(int x) {
    curAns += cnt[x];
```

```
        cnt[x]++;
    }

    public static void del(int x) {
        cnt[x]--;
        curAns -= cnt[x];
    }

    public static void compute() {
        int winl = 1, winr = 0;
        for (int i = 1; i <= cntq; i++) {
            int jobl = query[i][0];
            int jobr = query[i][1];
            int id = query[i][2];
            while (winl > jobl) {
                add(arr[--winl]);
            }
            while (winr < jobr) {
                add(arr[++winr]);
            }
            while (winl < jobl) {
                del(arr[winl++]);
            }
            while (winr > jobr) {
                del(arr[winr--]);
            }
            ans[id] = curAns;
        }
    }

    public static void prepare() {
        basePower[0] = 1;
        for (int i = 1; i <= n; i++) {
            basePower[i] = basePower[i - 1] * base;
            hashValue[i] = hashValue[i - 1] * base + (str[i] - 'a' + 1);
        }
        for (int l = 1, r = k; r <= n; l++, r++) {
            val[l] = hashValue[r] - hashValue[l - 1] * basePower[r - l + 1];
        }
        for (int i = 1; i <= len; i++) {
            sorted[i] = val[i];
        }
        Arrays.sort(sorted, 1, len + 1);
```

```
        cntv = 1;
        for (int i = 2; i <= len; i++) {
            if (sorted[cntv] != sorted[i]) {
                sorted[++cntv] = sorted[i];
            }
        }
        for (int i = 1; i <= len; i++) {
            arr[i] = kth(val[i]);
        }
        // 优化块长
        int blen = Math.max(1, (int) ((double) n / Math.sqrt(m)));
        for (int i = 1; i <= n; i++) {
            bi[i] = (i - 1) / blen + 1;
        }
        Arrays.sort(query, 1, cntq + 1, new QueryCmp());
    }

    public static void main(String[] args) throws Exception {
        FastReader in = new FastReader();
        PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));
        n = in.nextInt();
        m = in.nextInt();
        k = in.nextInt();
        for (int i = 1; i <= n; i++) {
            str[i] = in.nextLowerCase();
        }
        // 有效开头的个数，剩下的开头舍弃
        len = n - k + 1;
        cntq = 0;
        for (int i = 1, l, r; i <= m; i++) {
            l = in.nextInt();
            r = in.nextInt();
            // 过滤查询并调整查询参数
            if (l <= len) {
                query[++cntq][0] = l;
                query[cntq][1] = Math.min(r, len);
                query[cntq][2] = i;
            }
        }
        prepare();
        compute();
        for (int i = 1; i <= m; i++) {
            out.println(ans[i]);
```

```
    }
    out.flush();
    out.close();
}

// 读写工具类
static class FastReader {
    final private int BUFFER_SIZE = 1 << 16;
    private final InputStream in;
    private final byte[] buffer;
    private int ptr, len;

    public FastReader() {
        in = System.in;
        buffer = new byte[BUFFER_SIZE];
        ptr = len = 0;
    }

    private boolean hasNextByte() throws IOException {
        if (ptr < len)
            return true;
        ptr = 0;
        len = in.read(buffer);
        return len > 0;
    }

    private byte readByte() throws IOException {
        if (!hasNextByte())
            return -1;
        return buffer[ptr++];
    }

    public char nextLowerCase() throws IOException {
        int c;
        while (true) {
            c = readByte();
            if (c >= 'a' && c <= 'z')
                return (char) c;
        }
    }

    public int nextInt() throws IOException {
        int num = 0;
```

```
            byte b = readByte();
            while (isWhitespace(b))
                b = readByte();
            boolean minus = false;
            if (b == '-') {
                minus = true;
                b = readByte();
            }
            while (!isWhitespace(b) && b != -1) {
                num = num * 10 + (b - '0');
                b = readByte();
            }
            return minus ? -num : num;
        }

        private boolean isWhitespace(byte b) {
            return b == ' ' || b == '\n' || b == '\r' || b == '\t';
        }
    }

}
```

==========

文件: Code03_LCP2.java

==========

```
// 区间 lcp 达标对 - 普通莫队算法实现 (C++版本)
// 题目来源: 洛谷 P5112 区间 lcp 达标对
// 题目链接: https://www.luogu.com.cn/problem/P5112
// 题目大意: 给定一个长度为 n 的字符串 str，还有一个参数 k。位置 a 开头和位置 b 开头的字符串，
// 如果最长公共前缀的长度 >= k，那么(a, b)构成 lcp 达标对。构成 lcp 达标对，必须是不同的位置，
// 并且(a, b)和(b, a)只算一个达标对，不要重复统计。接下来有 m 条查询，格式为 l r :
// str[l..r]范围上，可以任选开头位置，打印 lcp 达标对的数量
// 解题思路: 使用普通莫队算法，结合字符串哈希和离散化技术
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
// 相关题目:
// 1. CF617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
// 3. BZOJ2038 [国家集训队]小 Z 的袜子 - https://www.lydsy.com/JudgeOnline/problem.php?id=2038
// 4. HDU4638 Group - http://acm.hdu.edu.cn/showproblem.php?pid=4638
// 5. AtCoder ABC176 D - Wizard in Maze - https://atcoder.jp/contests/abc176/tasks/abc176_d
```

```
package class179;

// 区间 lcp 达标对，C++版
// 给定一个长度为 n 的字符串 str，还有一个参数 k
// 位置 a 开头和位置 b 开头的字符串，如果最长公共前缀的长度 >= k，那么(a, b)构成 lcp 达标对
// 构成 lcp 达标对，必须是不同的位置，并且(a, b)和(b, a)只算一个达标对，不要重复统计
// 接下来有 m 条查询，格式为 l r : str[l..r]范围上，可以任选开头位置，打印 lcp 达标对的数量
// 1 <= n、k <= 3 * 10^6        1 <= m <= 10^5
// 1 <= n * n * m <= 10^15     字符集为 f z o u t s y
// 测试链接 : https://www.luogu.com.cn/problem/P5112
// 如下实现是 C++的版本，C++版本和 java 版本逻辑完全一样
// 提交如下代码，可以通过所有测试用例

//#include <bits/stdc++.h>
//
//using namespace std;
//
//struct Query {
//    int l, r, id;
//};
//
//const int MAXN = 3000001;
//const int MAXM = 100001;
//int n, m, k;
//int len, cntq, cntv;
//char str[MAXN];
//Query query[MAXM];
//
//int base = 499;
//long long basePower[MAXN;
//long long hashValue[MAXN;
//
//long long val[MAXN;
//long long sorted[MAXN;
//int arr[MAXN;
//int bi[MAXN;
//
//int cnt[MAXN;
//long long curAns;
//long long ans[MAXM;
//
//int kth(long long num) {
//    int left = 1, right = cntv, ret = 0;
```

```
//    while (left <= right) {
//        int mid = (left + right) >> 1;
//        if (sorted[mid] <= num) {
//            ret = mid;
//            left = mid + 1;
//        } else {
//            right = mid - 1;
//        }
//    }
//    return ret;
//}
//
//bool QueryCmp(Query &a, Query &b) {
//    if (bi[a.l] != bi[b.l]) {
//        return bi[a.l] < bi[b.l];
//    }
//    if (bi[a.l] & 1) {
//        return a.r < b.r;
//    } else {
//        return a.r > b.r;
//    }
//}
//
//void add(int x) {
//    curAns += cnt[x];
//    cnt[x]++;
//}
//
//void del(int x) {
//    cnt[x]--;
//    curAns -= cnt[x];
//}
//
//void compute() {
//    int winl = 1, winr = 0;
//    for (int i = 1; i <= cntq; i++) {
//        int jobl = query[i].l;
//        int jobr = query[i].r;
//        int id   = query[i].id;
//        while (winl > jobl) {
//            add(arr[--winl]);
//        }
//        while (winr < jobr) {
```

```
//            add(arr[++winr]);
//        }
//        while (winl < jobl) {
//            del(arr[winl++]);
//        }
//        while (winr > jobr) {
//            del(arr[winr--]);
//        }
//        ans[id] = curAns;
//    }
//}
//
//void prepare() {
//    basePower[0] = 1;
//    for (int i = 1; i <= n; i++) {
//        basePower[i] = basePower[i - 1] * base;
//        hashValue[i] = hashValue[i - 1] * base + (str[i] - 'a' + 1);
//    }
//    for (int l = 1, r = k; r <= n; l++, r++) {
//        val[l] = hashValue[r] - hashValue[l - 1] * basePower[r - l + 1];
//    }
//    for (int i = 1; i <= len; i++) {
//        sorted[i] = val[i];
//    }
//    sort(sorted + 1, sorted + len + 1);
//    cntv = 1;
//    for (int i = 2; i <= len; i++) {
//        if (sorted[cntv] != sorted[i]) {
//            sorted[++cntv] = sorted[i];
//        }
//    }
//    for (int i = 1; i <= len; i++) {
//        arr[i] = kth(val[i]);
//    }
//    int blen = max(1, (int)((double)n / sqrt((double)m)));
//    for (int i = 1; i <= n; i++) {
//        bi[i] = (i - 1) / blen + 1;
//    }
//    sort(query + 1, query + cntq + 1, QueryCmp);
//}
//
//int main() {
//    ios::sync_with_stdio(false);
```

```
//    cin.tie(nullptr);
//    cin >> n >> m >> k;
//    for (int i = 1; i <= n; i++) {
//        cin >> str[i];
//    }
//    len = n - k + 1;
//    cntq = 0;
//    for (int i = 1, l, r; i <= m; i++) {
//        cin >> l >> r;
//        if (l <= len) {
//            query[++cntq].l = l;
//            query[cntq].r = min(r, len);
//            query[cntq].id = i;
//        }
//    }
//    prepare();
//    compute();
//    for (int i = 1; i <= m; i++) {
//        cout << ans[i] << '\n';
//    }
//    return 0;
//}
```

==========

文件: Code04_MaximumMatch1.java

==========

```
// 区间最大匹配 - 普通莫队算法实现 (Java 版本)
// 题目来源: LibreOJ #6273. 「LibreOJ NOI Round #1」区间最大匹配
// 题目链接: https://loj.ac/p/6273
// 题目大意: 给定一个数组，每次查询区间[l,r]内的最大匹配数
// 解题思路: 使用普通莫队算法，通过维护区间信息来回答查询
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. LibreOJ #6273. 「LibreOJ NOI Round #1」区间最大匹配 - https://loj.ac/p/6273
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code04_MaximumMatch1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code04_MaximumMatch2.java
//
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
```

```
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 3. 洛谷 P1494 [国家集训队]小 Z 的袜子 - https://www.luogu.com.cn/problem/P1494
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 4. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//    Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
//    C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
//    Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
//
// 5. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//    Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//    Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 6. 洛谷 P1972 [SDOI2009] HH 的项链 - https://www.luogu.com.cn/problem/P1972
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery3.py
//
// 7. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 8. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
//    Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance1.java
//    C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance2.cpp
//    Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance3.py
//
// 9. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
```

```
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
//
// 10. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
//     Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch1.java
//     C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch2.cpp
//     Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch3.py

package class179;

// 区间最大匹配，java 版
// 给定长度为 n 的数组 a、长度为 m 的数组 b、一个正数 z
// 数组 a 中数字 x、数组 b 中数字 y，如果 x + y <= z，那么构成一个匹配
// 已经匹配的数字，不可以重复使用，一共有 q 条查询，格式如下
// 查询 l r : 数组 b[l..r]范围上的数字，随意选择数组 a 中的数字，打印最多匹配数
// 1 <= n <= 152501
// 1 <= m、q <= 52501
// 1 <= a[i]、b[i]、z <= 10^9
// 测试链接 : https://www.luogu.com.cn/problem/P4477
// 提交以下的 code，提交时请把类名改成"Main"，可以通过所有测试用例

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.util.Arrays;
import java.util.Comparator;

public class Code04_MaximumMatch1 {

    public static int MAXN = 152502;
    public static int MAXM = 52502;
    public static int MAXQ = 52502;
    public static int n, m, z, q;
    public static int[] a = new int[MAXN];
    public static int[] b = new int[MAXM];
    public static int[][] query = new int[MAXQ][3];
    public static int[] bi = new int[MAXM];

    // 线段树维护匹配信息，a[l..r]和 b 数组的数字进行匹配，l..r 对应的信息在 i 位置
    // match[i] = v，a[l..r]和 b 数组的数字，一共匹配了 v 对
```

```
// resta[i] = v，a[l..r]中还有 v 个数，可用于匹配 b 数组的数字
// overb[i] = v，a[l..r]已经耗尽，但是还有 v 个 b 数组的数字没有满足
public static int[] match = new int[MAXN << 2];
public static int[] resta = new int[MAXN << 2];
public static int[] overb = new int[MAXN << 2];

public static int[] ans = new int[MAXQ];

public static class QueryCmp implements Comparator<int[]> {
    @Override
    public int compare(int[] a, int[] b) {
        if (bi[a[0]] != bi[b[0]]) {
            return bi[a[0]] - bi[b[0]];
        }
        if ((bi[a[0]] & 1) == 1) {
            return a[1] - b[1];
        } else {
            return b[1] - a[1];
        }
    }
}

public static void up(int i) {
    int l = i << 1;
    int r = i << 1 | 1;
    int newMatch = Math.min(resta[l], overb[r]);
    resta[i] = resta[l] + resta[r] - newMatch;
    overb[i] = overb[l] + overb[r] - newMatch;
    match[i] = match[l] + match[r] + newMatch;
}

public static void build(int l, int r, int i) {
    if (l == r) {
        match[i] = 0;
        resta[i] = 1;
        overb[i] = 0;
    } else {
        int mid = (l + r) >> 1;
        build(l, mid, i << 1);
        build(mid + 1, r, i << 1 | 1);
        up(i);
    }
}
```

```
public static void add(int jobv, int l, int r, int i) {
    if (l == r) {
        if (resta[i] == 1) {
            match[i] = 1;
            resta[i] = 0;
        } else {
            overb[i]++;
        }
    } else {
        int mid = (l + r) >> 1;
        if (jobv + a[mid + 1] <= z) {
            add(jobv, mid + 1, r, i << 1 | 1);
        } else if (jobv + a[l] <= z) {
            add(jobv, l, mid, i << 1);
        }
        up(i);
    }
}

public static void del(int jobv, int l, int r, int i) {
    if (l == r) {
        if (overb[i] > 0) {
            overb[i]--;
        } else {
            match[i] = 0;
            resta[i] = 1;
        }
    } else {
        int mid = (l + r) >> 1;
        if (jobv + a[mid + 1] <= z) {
            del(jobv, mid + 1, r, i << 1 | 1);
        } else if (jobv + a[l] <= z) {
            del(jobv, l, mid, i << 1);
        }
        up(i);
    }
}

public static void prepare() {
    Arrays.sort(a, 1, n + 1);
    int blen = (int) Math.sqrt(m);
    for (int i = 1; i <= m; i++) {
```

```
            bi[i] = (i - 1) / blen + 1;
        }
        Arrays.sort(query, 1, q + 1, new QueryCmp());
        build(1, n, 1);
    }

    public static void compute() {
        int winl = 1, winr = 0;
        for (int i = 1; i <= q; i++) {
            int jobl = query[i][0];
            int jobr = query[i][1];
            int id = query[i][2];
            while (winl > jobl) {
                add(b[--winl], 1, n, 1);
            }
            while (winr < jobr) {
                add(b[++winr], 1, n, 1);
            }
            while (winl < jobl) {
                del(b[winl++], 1, n, 1);
            }
            while (winr > jobr) {
                del(b[winr--], 1, n, 1);
            }
            ans[id] = match[1];
        }
    }

    public static void main(String[] args) throws Exception {
        FastReader in = new FastReader(System.in);
        PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));
        n = in.nextInt();
        m = in.nextInt();
        z = in.nextInt();
        for (int i = 1; i <= n; i++) {
            a[i] = in.nextInt();
        }
        for (int i = 1; i <= m; i++) {
            b[i] = in.nextInt();
        }
        q = in.nextInt();
        for (int i = 1; i <= q; i++) {
            query[i][0] = in.nextInt();
```

```
        query[i][1] = in.nextInt();
        query[i][2] = i;
    }
    prepare();
    compute();
    for (int i = 1; i <= q; i++) {
        out.println(ans[i]);
    }
    out.flush();
    out.close();
}

// 读写工具类
static class FastReader {
    private final byte[] buffer = new byte[1 << 16];
    private int ptr = 0, len = 0;
    private final InputStream in;

    FastReader(InputStream in) {
        this.in = in;
    }

    private int readByte() throws IOException {
        if (ptr >= len) {
            len = in.read(buffer);
            ptr = 0;
            if (len <= 0)
                return -1;
        }
        return buffer[ptr++];
    }

    int nextInt() throws IOException {
        int c;
        do {
            c = readByte();
        } while (c <= ' ' && c != -1);
        boolean neg = false;
        if (c == '-') {
            neg = true;
            c = readByte();
        }
        int val = 0;
```

```
            while (c > ' ' && c != -1) {
                val = val * 10 + (c - '0');
                c = readByte();
            }
            return neg ? -val : val;
        }
    }

}
```

==========

文件: Code04_MaximumMatch2.java

==========

```
// 区间最大匹配 - 普通莫队算法实现 (C++版本)
// 题目来源: 洛谷 P4477 区间最大匹配
// 题目链接: https://www.luogu.com.cn/problem/P4477
// 题目大意: 给定长度为 n 的数组 a、长度为 m 的数组 b、一个正数 z。
// 数组 a 中数字 x、数组 b 中数字 y，如果 x + y <= z，那么构成一个匹配。
// 已经匹配的数字，不可以重复使用，一共有 q 条查询，格式如下：
// 查询 l r : 数组 b[l..r]范围上的数字，随意选择数组 a 中的数字，打印最多匹配数
// 解题思路: 使用普通莫队算法，结合线段树维护匹配信息
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
// 相关题目:
// 1. CF617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
// 3. BZOJ2038 [国家集训队]小 Z 的袜子 - https://www.lydsy.com/JudgeOnline/problem.php?id=2038
// 4. HDU4638 Group - http://acm.hdu.edu.cn/showproblem.php?pid=4638
// 5. AtCoder ABC176 D - Wizard in Maze - https://atcoder.jp/contests/abc176/tasks/abc176_d

package class179;

// 区间最大匹配，C++版
// 给定长度为 n 的数组 a、长度为 m 的数组 b、一个正数 z
// 数组 a 中数字 x、数组 b 中数字 y，如果 x + y <= z，那么构成一个匹配
// 已经匹配的数字，不可以重复使用，一共有 q 条查询，格式如下
// 查询 l r : 数组 b[l..r]范围上的数字，随意选择数组 a 中的数字，打印最多匹配数
// 1 <= n <= 152501
// 1 <= m、q <= 52501
// 1 <= a[i]、b[i]、z <= 10^9
// 测试链接 : https://www.luogu.com.cn/problem/P4477
// 如下实现是 C++的版本，C++版本和 java 版本逻辑完全一样
```

```
// 提交如下代码，可以通过所有测试用例

//#include <bits/stdc++.h>
//
//using namespace std;
//
//struct Query {
//    int l, r, id;
//};
//
//const int MAXN = 152502;
//const int MAXM = 52502;
//const int MAXQ = 52502;
//int n, m, z, q;
//int a[MAXN];
//int b[MAXM];
//Query query[MAXQ];
//int bi[MAXM];
//
//int match[MAXN << 2];
//int resta[MAXN << 2];
//int overb[MAXN << 2;
//
//int ans[MAXQ;
//
//bool QueryCmp(Query &a, Query &b) {
//    if (bi[a.l] != bi[b.l]) {
//        return bi[a.l] < bi[b.l];
//    }
//    if (bi[a.l] & 1) {
//        return a.r < b.r;
//    } else {
//        return a.r > b.r;
//    }
//}
//
//void up(int i) {
//    int l = i << 1;
//    int r = i << 1 | 1;
//    int newMatch = min(resta[l], overb[r]);
//    resta[i] = resta[l] + resta[r] - newMatch;
//    overb[i] = overb[l] + overb[r] - newMatch;
//    match[i] = match[l] + match[r] + newMatch;
```

```
//}
//
//void build(int l, int r, int i) {
//    if (l == r) {
//        match[i] = 0;
//        resta[i] = 1;
//        overb[i] = 0;
//    } else {
//        int mid = (l + r) >> 1;
//        build(l, mid, i << 1);
//        build(mid + 1, r, i << 1 | 1);
//        up(i);
//    }
//}
//
//void add(int jobv, int l, int r, int i) {
//    if (l == r) {
//        if (resta[i] == 1) {
//            match[i] = 1;
//            resta[i] = 0;
//        } else {
//            overb[i]++;
//        }
//    } else {
//        int mid = (l + r) >> 1;
//        if (jobv + a[mid + 1] <= z) {
//            add(jobv, mid + 1, r, i << 1 | 1);
//        } else if (jobv + a[l] <= z) {
//            add(jobv, l, mid, i << 1);
//        }
//        up(i);
//    }
//}
//
//void del(int jobv, int l, int r, int i) {
//    if (l == r) {
//        if (overb[i] > 0) {
//            overb[i]--;
//        } else {
//            match[i] = 0;
//            resta[i] = 1;
//        }
//    } else {
```

```
//        int mid = (l + r) >> 1;
//        if (jobv + a[mid + 1] <= z) {
//            del(jobv, mid + 1, r, i << 1 | 1);
//        } else if (jobv + a[l] <= z) {
//            del(jobv, l, mid, i << 1);
//        }
//        up(i);
//    }
//}
//
//void prepare() {
//    sort(a + 1, a + n + 1);
//    int blen = (int)sqrt(m);
//    for (int i = 1; i <= m; i++) {
//        bi[i] = (i - 1) / blen + 1;
//    }
//    sort(query + 1, query + q + 1, QueryCmp);
//    build(1, n, 1);
//}
//
//void compute() {
//    int winl = 1, winr = 0;
//    for (int i = 1; i <= q; i++) {
//        int jobl = query[i].l;
//        int jobr = query[i].r;
//        int id   = query[i].id;
//        while (winl > jobl) {
//            add(b[--winl], 1, n, 1);
//        }
//        while (winr < jobr) {
//            add(b[++winr], 1, n, 1);
//        }
//        while (winl < jobl) {
//            del(b[winl++], 1, n, 1);
//        }
//        while (winr > jobr) {
//            del(b[winr--], 1, n, 1);
//        }
//        ans[id] = match[1];
//    }
//}
//
//int main() {
```

```
//    ios::sync_with_stdio(false);
//    cin.tie(nullptr);
//    cin >> n >> m >> z;
//    for (int i = 1; i <= n; i++) {
//        cin >> a[i];
//    }
//    for (int i = 1; i <= m; i++) {
//        cin >> b[i];
//    }
//    cin >> q;
//    for (int i = 1; i <= q; i++) {
//        cin >> query[i].l >> query[i].r;
//        query[i].id = i;
//    }
//    prepare();
//    compute();
//    for (int i = 1; i <= q; i++) {
//        cout << ans[i] << '\n';
//    }
//    return 0;
//}
```

==========

文件: Code05_Homework1.java

==========

```
// 作业 - 普通莫队算法实现 (Java 版本)
// 题目来源: LibreOJ #6274. 「LibreOJ NOI Round #1」作业
// 题目链接: https://loj.ac/p/6274
// 题目大意: 给定一个数组，每次查询区间[l,r]内的作业完成情况
// 解题思路: 使用普通莫队算法，通过维护区间信息来回答查询
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. LibreOJ #6274. 「LibreOJ NOI Round #1」作业 - https://loj.ac/p/6274
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code05_Homework1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code05_Homework2.java
//
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
```

```
//
// 3. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 4. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
//    C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
//    Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
//
// 5. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//    Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 6. 洛谷 P1972 [SDOI2009] HH的项链 - https://www.luogu.com.cn/problem/P1972
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery3.py
//
// 7. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 8. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
//    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance1.java
//    C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance2.cpp
//    Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance3.py
//
// 9. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
```

```
//
// 10. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
//     Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch1.java
//     C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch2.cpp
//     Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch3.py

package class179;

// 作业，java 版
// 给定一个长度为 n 的数组 arr，接下来有 m 条查询，格式如下
// 查询 l r a b : 打印 arr[l..r]范围上的两个答案
//                答案 1，数值范围在[a, b]的数字个数
//                答案 2，数值范围在[a, b]的数字种数
// 1 <= 所有数据 <= 10^5
// 测试链接 : https://www.luogu.com.cn/problem/P4396
// 提交以下的 code，提交时请把类名改成"Main"，可以通过所有测试用例

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.util.Arrays;
import java.util.Comparator;

public class Code05_Homework1 {

    public static int MAXN = 100001;
    public static int MAXV = 100000;
    public static int MAXB = 401;
    public static int n, m;
    public static int[] arr = new int[MAXN];
    // 查询任务，l、r、a、b、id
    public static int[][] query = new int[MAXN][5];
    public static int[] bi = new int[MAXN];
    public static int[] bl = new int[MAXB];
    public static int[] br = new int[MAXB];

    public static int[] numCnt = new int[MAXN];
    public static int[] blockCnt = new int[MAXB];
    public static int[] blockKind = new int[MAXB];
```

```
public static int[] ans1 = new int[MAXN];
public static int[] ans2 = new int[MAXN];

public static class QueryCmp implements Comparator<int[]> {
    @Override
    public int compare(int[] a, int[] b) {
        if (bi[a[0]] != bi[b[0]]) {
            return bi[a[0]] - bi[b[0]];
        }
        if ((bi[a[0]] & 1) == 1) {
            return a[1] - b[1];
        } else {
            return b[1] - a[1];
        }
    }
}

public static void add(int x) {
    numCnt[x]++;
    blockCnt[bi[x]]++;
    if (numCnt[x] == 1) {
        blockKind[bi[x]]++;
    }
}

public static void del(int x) {
    numCnt[x]--;
    blockCnt[bi[x]]--;
    if (numCnt[x] == 0) {
        blockKind[bi[x]]--;
    }
}

public static void setAns(int a, int b, int id) {
    if (bi[a] == bi[b]) {
        for (int i = a; i <= b; i++) {
            if (numCnt[i] > 0) {
                ans1[id] += numCnt[i];
                ans2[id]++;
            }
        }
    } else {
```

```
        for (int i = a; i <= br[bi[a]]; i++) {
            if (numCnt[i] > 0) {
                ans1[id] += numCnt[i];
                ans2[id]++;
            }
        }
        for (int i = bl[bi[b]]; i <= b; i++) {
            if (numCnt[i] > 0) {
                ans1[id] += numCnt[i];
                ans2[id]++;
            }
        }
        for (int i = bi[a] + 1; i <= bi[b] - 1; i++) {
            ans1[id] += blockCnt[i];
            ans2[id] += blockKind[i];
        }
    }
}

public static void compute() {
    int winl = 1, winr = 0;
    for (int i = 1; i <= m; i++) {
        int jobl = query[i][0];
        int jobr = query[i][1];
        int joba = query[i][2];
        int jobb = query[i][3];
        int id = query[i][4];
        while (winl > jobl) {
            add(arr[--winl]);
        }
        while (winr < jobr) {
            add(arr[++winr]);
        }
        while (winl < jobl) {
            del(arr[winl++]);
        }
        while (winr > jobr) {
            del(arr[winr--]);
        }
        setAns(joba, jobb, id);
    }
}
```

```
public static void prepare() {
    int blen = (int) Math.sqrt(MAXV);
    int bnum = (MAXV + blen - 1) / blen;
    for (int i = 1; i <= MAXV; i++) {
        bi[i] = (i - 1) / blen + 1;
    }
    for (int i = 1; i <= bnum; i++) {
        bl[i] = (i - 1) * blen + 1;
        br[i] = Math.min(i * blen, MAXV);
    }
    Arrays.sort(query, 1, m + 1, new QueryCmp());
}

public static void main(String[] args) throws Exception {
    FastReader in = new FastReader(System.in);
    PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));
    n = in.nextInt();
    m = in.nextInt();
    for (int i = 1; i <= n; i++) {
        arr[i] = in.nextInt();
    }
    for (int i = 1; i <= m; i++) {
        query[i][0] = in.nextInt();
        query[i][1] = in.nextInt();
        query[i][2] = in.nextInt();
        query[i][3] = in.nextInt();
        query[i][4] = i;
    }
    prepare();
    compute();
    for (int i = 1; i <= m; i++) {
        out.println(ans1[i] + " " + ans2[i]);
    }
    out.flush();
    out.close();
}

// 读写工具类
static class FastReader {
    private final byte[] buffer = new byte[1 << 16];
    private int ptr = 0, len = 0;
    private final InputStream in;
```

```
        FastReader(InputStream in) {
            this.in = in;
        }

        private int readByte() throws IOException {
            if (ptr >= len) {
                len = in.read(buffer);
                ptr = 0;
                if (len <= 0)
                    return -1;
            }
            return buffer[ptr++];
        }

        int nextInt() throws IOException {
            int c;
            do {
                c = readByte();
            } while (c <= ' ' && c != -1);
            boolean neg = false;
            if (c == '-') {
                neg = true;
                c = readByte();
            }
            int val = 0;
            while (c > ' ' && c != -1) {
                val = val * 10 + (c - '0');
                c = readByte();
            }
            return neg ? -val : val;
        }
    }

}
```

==========

文件: Code05_Homework2.java

==========

```
// 作业 - 普通莫队算法实现 (C++版本)
// 题目来源: 洛谷 P4396 作业
// 题目链接: https://www.luogu.com.cn/problem/P4396
// 题目大意: 给定一个长度为 n 的数组 arr，接下来有 m 条查询，格式如下：
```

// 查询 l r a b : 打印arr[l..r]范围上的两个答案
// 答案1，数值范围在[a, b]的数字个数
// 答案2，数值范围在[a, b]的数字种数
// 解题思路: 使用普通莫队算法，结合分块技术维护数值范围内的统计信息
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
// 相关题目:
// 1. CF617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
// 3. BZOJ2038 [国家集训队]小Z的袜子 - https://www.lydsy.com/JudgeOnline/problem.php?id=2038
// 4. HDU4638 Group - http://acm.hdu.edu.cn/showproblem.php?pid=4638
// 5. AtCoder ABC176 D - Wizard in Maze - https://atcoder.jp/contests/abc176/tasks/abc176_d

package class179;

// 作业，C++版
// 给定一个长度为n的数组arr，接下来有m条查询，格式如下
// 查询 l r a b : 打印arr[l..r]范围上的两个答案
//   答案1，数值范围在[a, b]的数字个数
//   答案2，数值范围在[a, b]的数字种数
// 1 <= 所有数据 <= 10^5
// 测试链接 : https://www.luogu.com.cn/problem/P4396
// 如下实现是C++的版本，C++版本和java版本逻辑完全一样
// 提交如下代码，可以通过所有测试用例

//#include <bits/stdc++.h>
//
//using namespace std;
//
//struct Query {
//   int l, r, a, b, id;
//};
//
//const int MAXN = 100001;
//const int MAXV = 100000;
//const int MAXB = 401;
//int n, m;
//int arr[MAXN;
//Query query[MAXN;
//int bi[MAXN;
//int bl[MAXB;
//int br[MAXB;
//

```
//int numCnt[MAXN;
//int blockCnt[MAXB;
//int blockKind[MAXB;
//
//int ans1[MAXN;
//int ans2[MAXN;
//
//bool QueryCmp(Query &a, Query &b) {
//    if (bi[a.l] != bi[b.l]) {
//        return bi[a.l] < bi[b.l];
//    }
//    if (bi[a.l] & 1) {
//        return a.r < b.r;
//    } else {
//        return a.r > b.r;
//    }
//}
//
//void add(int x) {
//    numCnt[x]++;
//    blockCnt[bi[x]]++;
//    if (numCnt[x] == 1) {
//        blockKind[bi[x]]++;
//    }
//}
//
//void del(int x) {
//    numCnt[x]--;
//    blockCnt[bi[x]]--;
//    if (numCnt[x] == 0) {
//        blockKind[bi[x]]--;
//    }
//}
//
//void setAns(int a, int b, int id) {
//    if (bi[a] == bi[b]) {
//        for (int i = a; i <= b; i++) {
//            if (numCnt[i] > 0) {
//                ans1[id] += numCnt[i];
//                ans2[id]++;
//            }
//        }
//    } else {
```

```
//        for (int i = a; i <= br[bi[a]]; i++) {
//            if (numCnt[i] > 0) {
//                ans1[id] += numCnt[i];
//                ans2[id]++;
//            }
//        }
//        for (int i = bl[bi[b]]; i <= b; i++) {
//            if (numCnt[i] > 0) {
//                ans1[id] += numCnt[i];
//                ans2[id]++;
//            }
//        }
//        for (int i = bi[a] + 1; i <= bi[b] - 1; i++) {
//            ans1[id] += blockCnt[i];
//            ans2[id] += blockKind[i];
//        }
//    }
//}
//
//void compute() {
//    int winl = 1, winr = 0;
//    for (int i = 1; i <= m; i++) {
//        int jobl = query[i].l;
//        int jobr = query[i].r;
//        int joba = query[i].a;
//        int jobb = query[i].b;
//        int id = query[i].id;
//        while (winl > jobl) {
//            add(arr[--winl]);
//        }
//        while (winr < jobr) {
//            add(arr[++winr]);
//        }
//        while (winl < jobl) {
//            del(arr[winl++]);
//        }
//        while (winr > jobr) {
//            del(arr[winr--]);
//        }
//        setAns(joba, jobb, id);
//    }
//}
//
```

```
//void prepare() {
//    int blen = (int)sqrt(MAXV);
//    int bnum = (MAXV + blen - 1) / blen;
//    for (int i = 1; i <= MAXV; i++) {
//        bi[i] = (i - 1) / blen + 1;
//    }
//    for (int i = 1; i <= bnum; i++) {
//        bl[i] = (i - 1) * blen + 1;
//        br[i] = min(i * blen, MAXV);
//    }
//    sort(query + 1, query + m + 1, QueryCmp);
//}
//
//int main() {
//    ios::sync_with_stdio(false);
//    cin.tie(nullptr);
//    cin >> n >> m;
//    for (int i = 1; i <= n; i++) {
//        cin >> arr[i];
//    }
//    for (int i = 1; i <= m; i++) {
//        cin >> query[i].l >> query[i].r >> query[i].a >> query[i].b;
//        query[i].id = i;
//    }
//    prepare();
//    compute();
//    for (int i = 1; i <= m; i++) {
//        cout << ans1[i] << ' ' << ans2[i] << '\n';
//    }
//    return 0;
//}
```

==========

文件: Code06_NotForget1.java

==========

```
// 盼君勿忘 - 普通莫队算法实现 (Java 版本)
// 题目来源: LibreOJ #6275. 「LibreOJ NOI Round #1」盼君勿忘
// 题目链接: https://loj.ac/p/6275
// 题目大意: 给定一个数组，每次查询区间[l,r]内的特定计算结果
// 解题思路: 使用普通莫队算法，通过维护区间信息来回答查询
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
```

```
//
// 相关题目链接:
// 1. LibreOJ #6275. 「LibreOJ NOI Round #1」盼君勿忘 - https://loj.ac/p/6275
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code06_NotForget1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code06_NotForget2.java
//
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 3. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 4. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
//    C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
//    Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
//
// 5. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//    Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 6. 洛谷 P1972 [SDOI2009] HH的项链 - https://www.luogu.com.cn/problem/P1972
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery3.py
//
// 7. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 8. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
//    Java解答: https://github.com/algorithm-
```

journey/class179/blob/main/Code16_ColorMaintenance1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance3.py
//
// 9. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
//
// 10. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch3.py

package class179;

// 盼君勿忘，java版
// 一个序列中每种数字只保留一个，得到的累加和，叫做去重累加和
// 给定一个长度为n的数组arr，接下来是m条查询，查询格式如下
// 查询 l r p : arr[l..r]范围上，每个子序列的去重累加和，都累加起来 % p 的结果打印
// 1 <= n、m、arr[i] <= 10^5
// 1 <= p <= 10^9
// 测试链接 : https://www.luogu.com.cn/problem/P5072
// 提交以下的code，提交时请把类名改成"Main"
// java实现的逻辑一定是正确的，但是本题卡常，无法通过所有测试用例
// 想通过用C++实现，本节课Code06_NotForget2文件就是C++的实现
// 两个版本的逻辑完全一样，C++版本可以通过所有测试

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.util.Arrays;
import java.util.Comparator;

public class Code06_NotForget1 {

    public static int MAXN = 100001;

```
public static int MAXB = 401;
public static int n, m;
public static int[] arr = new int[MAXN];
// 查询任务，l、r、p、id
public static int[][] query = new int[MAXN][4];
public static int[] bi = new int[MAXN];

// 有效次数桶组成的双向链表
// 有数字进入次数桶，该次数桶才进入链表，链表内部不需要有序组织
public static int head;
public static int[] last = new int[MAXN];
public static int[] next = new int[MAXN];

// cnt[v] = c，表示 v 这个数出现了 c 次，也可以说 v 在 c 次桶里
// sum[c] = x，表示 c 次桶内的数字，每种数字只统计一次，累加和为 x
public static int[] cnt = new int[MAXN];
public static long[] sum = new long[MAXN];

// 光速幂
// 假设[l..r]范围长度 len，blockLen 为块的长度，blockNum 为块的数量
// smlPower[i]，表示 p 的 i 次方的值，i <= blockLen
// bigPower[i]，表示 p 的(i * blockLen)次方的值，i <= blockNum
public static long[] smlPower = new long[MAXB];
public static long[] bigPower = new long[MAXB];

public static long[] ans = new long[MAXN];

public static class QueryCmp implements Comparator<int[]> {
    @Override
    public int compare(int[] a, int[] b) {
        if (bi[a[0]] != bi[b[0]]) {
            return bi[a[0]] - bi[b[0]];
        }
        if ((bi[a[0]] & 1) == 1) {
            return a[1] - b[1];
        } else {
            return b[1] - a[1];
        }
    }
}

public static void addNode(int x) {
    last[head] = x;
```

```
        next[x] = head;
        head = x;
    }

    public static void delNode(int x) {
        if (x == head) {
            head = next[head];
            last[head] = next[x] = 0;
        } else {
            next[last[x]] = next[x];
            last[next[x]] = last[x];
            last[x] = next[x] = 0;
        }
    }

    public static void add(int num) {
        if (cnt[num] > 0) {
            sum[cnt[num]] -= num;
        }
        if (cnt[num] > 0 && sum[cnt[num]] == 0) {
            delNode(cnt[num]);
        }
        cnt[num]++;
        if (cnt[num] > 0 && sum[cnt[num]] == 0) {
            addNode(cnt[num]);
        }
        if (cnt[num] > 0) {
            sum[cnt[num]] += num;
        }
    }

    public static void del(int num) {
        if (cnt[num] > 0) {
            sum[cnt[num]] -= num;
        }
        if (cnt[num] > 0 && sum[cnt[num]] == 0) {
            delNode(cnt[num]);
        }
        cnt[num]--;
        if (cnt[num] > 0 && sum[cnt[num]] == 0) {
            addNode(cnt[num]);
        }
        if (cnt[num] > 0) {
```

```
        sum[cnt[num]] += num;
    }
}

public static void setAns(int len, int p, int id) {
    // 光速幂
    // 构建 smlPower 和 bigPower
    int blockLen = (int) Math.sqrt(len);
    int blockNum = (len + blockLen - 1) / blockLen;
    smlPower[0] = 1;
    for (int i = 1; i <= blockLen; i++) {
        smlPower[i] = (smlPower[i - 1] << 1) % p;
    }
    bigPower[0] = 1;
    for (int i = 1; i <= blockNum; i++) {
        bigPower[i] = (bigPower[i - 1] * smlPower[blockLen]) % p;
    }
    // t 次桶的贡献 = [2 的 len 次方 - 2 的(len-t)次方] * sum[t]
    long res = 0, p1, p2, tmp;
    p1 = bigPower[len / blockLen] * smlPower[len % blockLen] % p;
    for (int t = head; t > 0; t = next[t]) {
        p2 = bigPower[(len - t) / blockLen] * smlPower[(len - t) % blockLen] % p;
        tmp = (p1 - p2) * sum[t] % p;
        res = ((res + tmp) % p + p) % p;
    }
    ans[id] = res;
}

public static void compute() {
    int winl = 1, winr = 0;
    for (int i = 1; i <= m; i++) {
        int jobl = query[i][0];
        int jobr = query[i][1];
        int jobp = query[i][2];
        int id = query[i][3];
        while (winl > jobl) {
            add(arr[--winl]);
        }
        while (winr < jobr) {
            add(arr[++winr]);
        }
        while (winl < jobl) {
            del(arr[winl++]);
```

```
        }
        while (winr > jobr) {
            del(arr[winr--]);
        }
        setAns(jobr - jobl + 1, jobp, id);
    }
}

public static void prepare() {
    int blen = (int) Math.sqrt(n);
    for (int i = 1; i <= n; i++) {
        bi[i] = (i - 1) / blen + 1;
    }
    Arrays.sort(query, 1, m + 1, new QueryCmp());
}

public static void main(String[] args) throws Exception {
    FastReader in = new FastReader(System.in);
    PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));
    n = in.nextInt();
    m = in.nextInt();
    for (int i = 1; i <= n; i++) {
        arr[i] = in.nextInt();
    }
    for (int i = 1; i <= m; i++) {
        query[i][0] = in.nextInt();
        query[i][1] = in.nextInt();
        query[i][2] = in.nextInt();
        query[i][3] = i;
    }
    prepare();
    compute();
    for (int i = 1; i <= m; i++) {
        out.println(ans[i]);
    }
    out.flush();
    out.close();
}

// 读写工具类
static class FastReader {
    private final byte[] buffer = new byte[1 << 16];
    private int ptr = 0, len = 0;
```

```
        private final InputStream in;

        FastReader(InputStream in) {
            this.in = in;
        }

        private int readByte() throws IOException {
            if (ptr >= len) {
                len = in.read(buffer);
                ptr = 0;
                if (len <= 0)
                    return -1;
            }
            return buffer[ptr++];
        }

        int nextInt() throws IOException {
            int c;
            do {
                c = readByte();
            } while (c <= ' ' && c != -1);
            boolean neg = false;
            if (c == '-') {
                neg = true;
                c = readByte();
            }
            int val = 0;
            while (c > ' ' && c != -1) {
                val = val * 10 + (c - '0');
                c = readByte();
            }
            return neg ? -val : val;
        }
    }

}
```

==========

文件: Code06_NotForget2.java

==========

```
// 盼君勿忘 - 普通莫队算法实现 (C++版本)
// 题目来源: 洛谷 P5072 盼君勿忘
```

// 题目链接: https://www.luogu.com.cn/problem/P5072
// 题目大意: 一个序列中每种数字只保留一个，得到的累加和，叫做去重累加和。
// 给定一个长度为 n 的数组 arr，接下来是 m 条查询，查询格式如下：
// 查询 l r p : arr[l..r]范围上，每个子序列的去重累加和，都累加起来 % p 的结果打印
// 解题思路: 使用普通莫队算法，结合双向链表维护有效次数桶
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
// 相关题目:
// 1. CF617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
// 3. BZOJ2038 [国家集训队]小 Z 的袜子 - https://www.lydsy.com/JudgeOnline/problem.php?id=2038
// 4. HDU4638 Group - http://acm.hdu.edu.cn/showproblem.php?pid=4638
// 5. AtCoder ABC176 D - Wizard in Maze - https://atcoder.jp/contests/abc176/tasks/abc176_d

package class179;

// 盼君勿忘，C++版
// 一个序列中每种数字只保留一个，得到的累加和，叫做去重累加和
// 给定一个长度为 n 的数组 arr，接下来是 m 条查询，查询格式如下
// 查询 l r p : arr[l..r]范围上，每个子序列的去重累加和，都累加起来 % p 的结果打印
// 1 <= n、m、arr[i] <= 10^5
// 1 <= p <= 10^9
// 测试链接 : https://www.luogu.com.cn/problem/P5072
// 如下实现是 C++的版本，C++版本和 java 版本逻辑完全一样
// 提交如下代码，可以通过所有测试用例

//#include <bits/stdc++.h>
//
//using namespace std;
//
//struct Query {
//   int l, r, p, id;
//};
//
//const int MAXN = 100001;
//const int MAXB = 401;
//int n, m;
//int arr[MAXN;
//Query query[MAXN;
//int bi[MAXN;
//
//int head;
//int lst[MAXN;

```
//int nxt[MAXN;
//
//int cnt[MAXN;
//long long sum[MAXN;
//
//long long smlPower[MAXB;
//long long bigPower[MAXB;
//
//long long ans[MAXN;
//
//bool QueryCmp(Query &a, Query &b) {
//    if (bi[a.l] != bi[b.l]) {
//        return bi[a.l] < bi[b.l];
//    }
//    if (bi[a.l] & 1) {
//        return a.r < b.r;
//    } else {
//        return a.r > b.r;
//    }
//}
//
//void addNode(int x) {
//    lst[head] = x;
//    nxt[x] = head;
//    head = x;
//}
//
//void delNode(int x) {
//    if (x == head) {
//        head = nxt[head];
//        lst[head] = 0;
//        nxt[x] = 0;
//    } else {
//        nxt[lst[x]] = nxt[x];
//        lst[nxt[x]] = lst[x];
//        lst[x] = 0;
//        nxt[x] = 0;
//    }
//}
//
//void add(int num) {
//    if (cnt[num] > 0) {
//        sum[cnt[num]] -= num;
```

```
//    }
//    if (cnt[num] > 0 && sum[cnt[num]] == 0) {
//        delNode(cnt[num]);
//    }
//    cnt[num]++;
//    if (cnt[num] > 0 && sum[cnt[num]] == 0) {
//        addNode(cnt[num]);
//    }
//    if (cnt[num] > 0) {
//        sum[cnt[num]] += num;
//    }
//}
//
//void del(int num) {
//    if (cnt[num] > 0) {
//        sum[cnt[num]] -= num;
//    }
//    if (cnt[num] > 0 && sum[cnt[num]] == 0) {
//        delNode(cnt[num]);
//    }
//    cnt[num]--;
//    if (cnt[num] > 0 && sum[cnt[num]] == 0) {
//        addNode(cnt[num]);
//    }
//    if (cnt[num] > 0) {
//        sum[cnt[num]] += num;
//    }
//}
//
//void setAns(int len, int p, int id) {
//    int blockLen = (int)sqrt(len);
//    int blockNum = (len + blockLen - 1) / blockLen;
//    smlPower[0] = 1;
//    for (int i = 1; i <= blockLen; i++) {
//        smlPower[i] = (smlPower[i - 1] << 1) % p;
//    }
//    bigPower[0] = 1;
//    for (int i = 1; i <= blockNum; i++) {
//        bigPower[i] = (bigPower[i - 1] * smlPower[blockLen]) % p;
//    }
//    long long res = 0, p1, p2, tmp;
//    p1 = bigPower[len / blockLen] * smlPower[len % blockLen] % p;
//    for (int t = head; t > 0; t = nxt[t]) {
```

```
//        p2 = bigPower[(len - t) / blockLen] * smlPower[(len - t) % blockLen] % p;
//        tmp = (p1 - p2) * sum[t] % p;
//        res = ((res + tmp) % p + p) % p;
//    }
//    ans[id] = res;
//}
//
//void compute() {
//    int winl = 1, winr = 0;
//    for (int i = 1; i <= m; i++) {
//        int jobl = query[i].l;
//        int jobr = query[i].r;
//        int jobp = query[i].p;
//        int id = query[i].id;
//        while (winl > jobl) {
//            add(arr[--winl]);
//        }
//        while (winr < jobr) {
//            add(arr[++winr]);
//        }
//        while (winl < jobl) {
//            del(arr[winl++]);
//        }
//        while (winr > jobr) {
//            del(arr[winr--]);
//        }
//        setAns(jobr - jobl + 1, jobp, id);
//    }
//}
//
//void prepare() {
//    int blen = (int)sqrt(n);
//    for (int i = 1; i <= n; i++) {
//        bi[i] = (i - 1) / blen + 1;
//    }
//    sort(query + 1, query + m + 1, QueryCmp);
//}
//
//int main() {
//    ios::sync_with_stdio(false);
//    cin.tie(nullptr);
//    cin >> n >> m;
//    for (int i = 1; i <= n; i++) {
```

```
//        cin >> arr[i];
//    }
//    for (int i = 1; i <= m; i++) {
//        cin >> query[i].l >> query[i].r >> query[i].p;
//        query[i].id = i;
//    }
//    prepare();
//    compute();
//    for (int i = 1; i <= m; i++) {
//        cout << ans[i] << '\n';
//    }
//    return 0;
//}
```

==========

文件: Code07_RabbitHole1.java

```
// 掉进兔子洞 - 普通莫队算法实现 (Java 版本)
// 题目来源: 洛谷 P4688 [Ynoi2016]掉进兔子洞
// 题目链接: https://www.luogu.com.cn/problem/P4688
// 题目大意: 给定一个数组，每次查询区间[l,r]内的特定计算结果
// 解题思路: 使用普通莫队算法，通过维护区间信息来回答查询
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. 洛谷 P4688 [Ynoi2016]掉进兔子洞 - https://www.luogu.com.cn/problem/P4688
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code07_RabbitHole1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code07_RabbitHole2.java
//
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 3. 洛谷 P1494 [国家集训队]小 Z 的袜子 - https://www.luogu.com.cn/problem/P1494
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 4. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//    Java 解答: https://github.com/algorithm-
```

journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
//
// 5. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 6. 洛谷 P1972 [SDOI2009] HH的项链 - https://www.luogu.com.cn/problem/P1972
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery3.py
//
// 7. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 8. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance3.py
//
// 9. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
//
// 10. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch3.py

```
package class179;

// 掉进兔子洞，java 版
// 三个区间同时出现的数，一个一个删掉，直到无法再删，剩下数字的个数叫做 剩余个数
// A = [1 2 2 3 3 3]    B = [1 2 2 3 3 3]    C = [1 1 2 3 3]
// 删除的过程为，一起删掉一个 1、一起删掉一个 2、一起删掉 2 个 3，然后状况为
// A = [2 3]    B = [2 3]    C = [1]    剩余个数为 5
// 给定一个长度为 n 的数组 arr，下来有 m 条查询，格式如下
// 查询 l1 r1 l2 r2 l3 r3 : 给定了三个区间，打印剩余个数
// 1 <= n、m <= 10^5
// 1 <= arr[i] <= 10^9
// 测试链接 : https://www.luogu.com.cn/problem/P4688
// 提交以下的 code，提交时请把类名改成"Main"，可以通过所有测试用例

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.util.Arrays;
import java.util.Comparator;

public class Code07_RabbitHole1 {

    static class BitSet {
        int len;
        long[] status;

        public BitSet(int siz) {
            len = (siz + 63) >> 6;
            status = new long[len];
        }

        public void clear() {
            for (int i = 0; i < len; i++) {
                status[i] = 0;
            }
        }

        public void copy(BitSet obj) {
            for (int i = 0; i < len; i++) {
                status[i] = obj.status[i];
            }
```

```
    }

    public void and(BitSet obj) {
        for (int i = 0; i < len; i++) {
            status[i] &= obj.status[i];
        }
    }

    public void setOne(int bit) {
        status[bit >> 6] |= 1L << (bit & 63);
    }

    public void setZero(int bit) {
        status[bit >> 6] &= ~(1L << (bit & 63));
    }

    public int getOnes() {
        int ret = 0;
        for (int i = 0; i < len; i++) {
            ret += Long.bitCount(status[i]);
        }
        return ret;
    }

}

public static int MAXN = 100001;
public static int MAXT = 30001;
public static int n, m;
public static int[] arr = new int[MAXN];
public static int[][] query = new int[MAXT * 3][3];

// 排序之后不去重
// 得到每个数字的排名
public static int[] sorted = new int[MAXN];
public static int[] bi = new int[MAXN];

// cnt[v] = c，表示窗口中数字 v 出现 c 次
public static int[] cnt = new int[MAXN];
// 当前窗口的位图
public static BitSet curSet;
// 问题从来没获得位图就拷贝，问题获得过位图就做&运算
public static boolean[] hasSet = new boolean[MAXT];
```

```
// 每个问题的位图
public static BitSet[] bitSet = new BitSet[MAXT];

public static int[] ans = new int[MAXT];

public static class QueryCmp implements Comparator<int[]> {
    @Override
    public int compare(int[] a, int[] b) {
        if (bi[a[0]] != bi[b[0]]) {
            return bi[a[0]] - bi[b[0]];
        }
        if ((bi[a[0]] & 1) == 1) {
            return a[1] - b[1];
        } else {
            return b[1] - a[1];
        }
    }
}

public static int kth(int num) {
    int left = 1, right = n, mid, ret = -1;
    while (left <= right) {
        mid = (left + right) >> 1;
        if (sorted[mid] >= num) {
            ret = mid;
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }
    return ret;
}

public static void add(int x) {
    cnt[x]++;
    curSet.setOne(x + cnt[x] - 1);
}

public static void del(int x) {
    cnt[x]--;
    curSet.setZero(x + cnt[x]);
}
```

```
public static void compute(int q) {
    int winl = 1, winr = 0;
    for (int i = 1; i <= q; i++) {
        int jobl = query[i][0];
        int jobr = query[i][1];
        int id = query[i][2];
        while (winl > jobl) {
            add(arr[--winl]);
        }
        while (winr < jobr) {
            add(arr[++winr]);
        }
        while (winl < jobl) {
            del(arr[winl++]);
        }
        while (winr > jobr) {
            del(arr[winr--]);
        }
        if (!hasSet[id]) {
            hasSet[id] = true;
            bitSet[id].copy(curSet);
        } else {
            bitSet[id].and(curSet);
        }
    }
}

public static void prepare() {
    for (int i = 1; i <= n; i++) {
        sorted[i] = arr[i];
    }
    Arrays.sort(sorted, 1, n + 1);
    for (int i = 1; i <= n; i++) {
        arr[i] = kth(arr[i]);
    }
    int blen = (int) Math.sqrt(n);
    for (int i = 1; i <= n; i++) {
        bi[i] = (i - 1) / blen + 1;
    }
    for (int i = 1; i < MAXT; i++) {
        bitSet[i] = new BitSet(n + 1);
    }
    curSet = new BitSet(n + 1);
```

```
}

public static void main(String[] args) throws Exception {
    FastReader in = new FastReader(System.in);
    PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));
    n = in.nextInt();
    m = in.nextInt();
    for (int i = 1; i <= n; i++) {
        arr[i] = in.nextInt();
    }
    prepare();
    for (int t = MAXT - 1; m > 0; m -= t) {
        int k = Math.min(m, t);
        Arrays.fill(cnt, 1, n + 1, 0);
        Arrays.fill(hasSet, 1, k + 1, false);
        Arrays.fill(ans, 1, k + 1, 0);
        curSet.clear();
        int cntq = 0;
        for (int i = 1; i <= k; i++) {
            for (int j = 1; j <= 3; j++) {
                query[++cntq][0] = in.nextInt();
                query[cntq][1] = in.nextInt();
                query[cntq][2] = i;
                ans[i] += query[cntq][1] - query[cntq][0] + 1;
            }
        }
        Arrays.sort(query, 1, cntq + 1, new QueryCmp());
        compute(cntq);
        for (int i = 1; i <= k; i++) {
            ans[i] -= bitSet[i].getOnes() * 3;
        }
        for (int i = 1; i <= k; i++) {
            out.println(ans[i]);
        }
    }
    out.flush();
    out.close();
}

// 读写工具类
static class FastReader {
    private final byte[] buffer = new byte[1 << 16];
    private int ptr = 0, len = 0;
```

```
        private final InputStream in;

        FastReader(InputStream in) {
            this.in = in;
        }

        private int readByte() throws IOException {
            if (ptr >= len) {
                len = in.read(buffer);
                ptr = 0;
                if (len <= 0)
                    return -1;
            }
            return buffer[ptr++];
        }

        int nextInt() throws IOException {
            int c;
            do {
                c = readByte();
            } while (c <= ' ' && c != -1);
            boolean neg = false;
            if (c == '-') {
                neg = true;
                c = readByte();
            }
            int val = 0;
            while (c > ' ' && c != -1) {
                val = val * 10 + (c - '0');
                c = readByte();
            }
            return neg ? -val : val;
        }
    }

}
```

==========

文件: Code07_RabbitHole2.java

==========

```
// 掉进兔子洞 - 普通莫队算法实现 (C++版本)
// 题目来源: 洛谷 P4688 掉进兔子洞
```

// 题目链接: https://www.luogu.com.cn/problem/P4688
// 题目大意: 三个区间同时出现的数，一个一个删掉，直到无法再删，剩下数字的个数叫做 剩余个数
// A = [1 2 2 3 3 3]   B = [1 2 2 3 3 3]   C = [1 1 2 3 3]
// 删除的过程为，一起删掉一个 1、一起删掉一个 2、一起删掉 2 个 3，然后状况为
// A = [2 3]   B = [2 3]   C = [1]   剩余个数为 5
// 给定一个长度为 n 的数组 arr，下来有 m 条查询，格式如下
// 查询 l1 r1 l2 r2 l3 r3 : 给定了三个区间，打印剩余个数
// 解题思路: 使用普通莫队算法，结合位运算和分批处理技术
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
// 相关题目:
// 1. CF617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
// 3. BZOJ2038 [国家集训队]小 Z 的袜子 - https://www.lydsy.com/JudgeOnline/problem.php?id=2038
// 4. HDU4638 Group - http://acm.hdu.edu.cn/showproblem.php?pid=4638
// 5. AtCoder ABC176 D - Wizard in Maze - https://atcoder.jp/contests/abc176/tasks/abc176_d

package class179;

// 掉进兔子洞，C++版
// 三个区间同时出现的数，一个一个删掉，直到无法再删，剩下数字的个数叫做 剩余个数
// A = [1 2 2 3 3 3]   B = [1 2 2 3 3 3]   C = [1 1 2 3 3]
// 删除的过程为，一起删掉一个 1、一起删掉一个 2、一起删掉 2 个 3，然后状况为
// A = [2 3]   B = [2 3]   C = [1]   剩余个数为 5
// 给定一个长度为 n 的数组 arr，下来有 m 条查询，格式如下
// 查询 l1 r1 l2 r2 l3 r3 : 给定了三个区间，打印剩余个数
// 1 <= n、m <= 10^5
// 1 <= arr[i] <= 10^9
// 测试链接 : https://www.luogu.com.cn/problem/P4688
// 如下实现是 C++的版本，C++版本和 java 版本逻辑完全一样
// 提交如下代码，可以通过所有测试用例

//#include <bits/stdc++.h>
//
//using namespace std;
//
//struct Query {
//   int l, r, id;
//};
//
//const int MAXN = 100001;
//const int MAXT = 30001;
//int n, m;

```
//int arr[MAXN;
//
//int sorted[MAXN;
//int bi[MAXN;
//
//int cnt[MAXN;
//bitset<MAXN> curSet;
//bool hasSet[MAXT;
//bitset<MAXN> bitSet[MAXT;
//
//Query query[MAXT * 3;
//
//int ans[MAXT;
//
//int kth(int num) {
//    int left = 1, right = n, ret = -1;
//    while (left <= right) {
//        int mid = (left + right) >> 1;
//        if (sorted[mid] >= num) {
//            ret = mid;
//            right = mid - 1;
//        } else {
//            left = mid + 1;
//        }
//    }
//    return ret;
//}
//
//bool QueryCmp(const Query &a, const Query &b) {
//    if (bi[a.l] != bi[b.l]) {
//        return bi[a.l] < bi[b.l];
//    }
//    if (bi[a.l] & 1) {
//        return a.r < b.r;
//    } else {
//        return a.r > b.r;
//    }
//}
//
//void add(int x) {
//    cnt[x]++;
//    curSet[x + cnt[x] - 1] = 1;
//}
```

```
//
//void del(int x) {
//    cnt[x]--;
//    curSet[x + cnt[x]] = 0;
//}
//
//void compute(int q) {
//    int winl = 1, winr = 0;
//    for (int i = 1; i <= q; i++) {
//        int jobl = query[i].l;
//        int jobr = query[i].r;
//        int id = query[i].id;
//        while (winl > jobl) {
//            add(arr[--winl]);
//        }
//        while (winr < jobr) {
//            add(arr[++winr]);
//        }
//        while (winl < jobl) {
//            del(arr[winl++]);
//        }
//        while (winr > jobr) {
//            del(arr[winr--]);
//        }
//        if (!hasSet[id]) {
//            hasSet[id] = true;
//            bitSet[id] = curSet;
//        } else {
//            bitSet[id] &= curSet;
//        }
//    }
//}
//
//void prepare() {
//    for (int i = 1; i <= n; i++) {
//        sorted[i] = arr[i];
//    }
//    sort(sorted + 1, sorted + n + 1);
//    for (int i = 1; i <= n; i++) {
//        arr[i] = kth(arr[i]);
//    }
//    int blen = (int)sqrt(n);
//    for (int i = 1; i <= n; i++) {
```

```
//        bi[i] = (i - 1) / blen + 1;
//    }
//}
//
//int main() {
//    ios::sync_with_stdio(false);
//    cin.tie(nullptr);
//    cin >> n >> m;
//    for (int i = 1; i <= n; i++) {
//        cin >> arr[i];
//    }
//    prepare();
//    for (int t = MAXT - 1; m > 0; m -= t) {
//        int k = min(m, t);
//        memset(cnt, 0, sizeof(int) * (n + 2));
//        memset(hasSet, 0, sizeof(bool) * (k + 2));
//        memset(ans, 0, sizeof(int) * (k + 2));
//        curSet.reset();
//        int cntq = 0, l, r;
//        for (int i = 1; i <= k; i++) {
//            for (int j = 1; j <= 3; j++) {
//                cin >> l >> r;
//                query[++cntq] = { l, r, i };
//                ans[i] += r - l + 1;
//            }
//        }
//        sort(query + 1, query + cntq + 1, QueryCmp);
//        compute(cntq);
//        for (int i = 1; i <= k; i++) {
//            ans[i] -= bitSet[i].count() * 3;
//        }
//        for (int i = 1; i <= k; i++) {
//            cout << ans[i] << '\n';
//        }
//    }
//    return 0;
//}
```

==========

文件: Code08_YnoiCornfield1.java

==========

```
// 由乃的玉米田 - 普通莫队算法实现 (Java 版本)
```

// 题目来源: 洛谷 P4120 [Ynoi2016]由乃的玉米田
// 题目链接: https://www.luogu.com.cn/problem/P4120
// 题目大意: 给定一个数组，每次查询区间[l,r]内的特定计算结果
// 解题思路: 使用普通莫队算法，通过维护区间信息来回答查询
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. 洛谷 P4120 [Ynoi2016]由乃的玉米田 - https://www.luogu.com.cn/problem/P4120
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code08_YnoiCornfield1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code08_YnoiCornfield2.java
//
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 3. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 4. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
//
// 5. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 6. 洛谷 P1972 [SDOI2009] HH的项链 - https://www.luogu.com.cn/problem/P1972
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery3.py
//

```
// 7. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 8. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
//    Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance1.java
//    C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance2.cpp
//    Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance3.py
//
// 9. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
//
// 10. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
//     Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch1.java
//     C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch2.cpp
//     Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch3.py

package class179;

// 由乃的玉米田，java 版
// 给定一个长度为 n 的数组 arr，接下来有 m 条查询，查询格式如下
// 查询 1 l r x : 打印 arr[l..r]范围上能否选出两个数，减的结果为 x
// 查询 2 l r x : 打印 arr[l..r]范围上能否选出两个数，加的结果为 x
// 查询 3 l r x : 打印 arr[l..r]范围上能否选出两个数，乘的结果为 x
// 查询 4 l r x : 打印 arr[l..r]范围上能否选出两个数，除的结果为 x，并且没有余数
// 选出的这两个数可以是同一个位置的数，答案如果为是，打印 "yuno"，否则打印 "yumi"
// 1 <= 所有数据 <= 10^5
// 测试链接 : https://www.luogu.com.cn/problem/P5355
// 提交以下的 code，提交时请把类名改成"Main"，可以通过所有测试用例

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
```

```
import java.util.Arrays;
import java.util.Comparator;

public class Code08_YnoiCornfield1 {

    static class BitSet {
        int len;
        long[] status;

        public BitSet(int siz) {
            len = (siz + 63) >> 6;
            status = new long[len];
        }

        public void setOne(int bit) {
            status[bit >> 6] |= 1L << (bit & 63);
        }

        public void setZero(int bit) {
            status[bit >> 6] &= ~(1L << (bit & 63));
        }

        public boolean getStatus(int bit) {
            return ((status[bit >> 6] >> (bit & 63)) & 1L) != 0L;
        }

        // 检查 自己 & other右移k位 是否有1存在
        public boolean andOtherMoveRight(BitSet other, int move) {
            int ws = move >> 6;
            int bs = move & 63;
            for (int i = 0; i < len; i++) {
                int src = i + ws;
                if (src >= len) {
                    break;
                }
                long shifted = other.status[src] >>> bs;
                if (bs != 0 && src + 1 < len) {
                    shifted |= other.status[src + 1] << (64 - bs);
                }
                if ((status[i] & shifted) != 0L) {
                    return true;
                }
            }
```

```
        return false;
    }

}

public static int MAXN = 100001;
public static int MAXV = 100000;
public static int MAXB = 401;
public static int n, m, blen;
public static int[] arr = new int[MAXN];
public static int[] bi = new int[MAXN];

// 普通查询，l、r、x、op、id
public static int[][] query = new int[MAXN][5];
public static int cntq;

// 特别查询，x的问题列表 : l、r、id
public static int[] headq = new int[MAXB];
public static int[] nextq = new int[MAXN];
public static int[] ql = new int[MAXN];
public static int[] qr = new int[MAXN];
public static int[] qid = new int[MAXN];
public static int cnts;

// 数字出现的词频
public static int[] cnt = new int[MAXN];
public static BitSet bitSet1 = new BitSet(MAXN);
public static BitSet bitSet2 = new BitSet(MAXN);

// 特别查询的dp过程
public static int[] pre = new int[MAXN];
public static int[] dp = new int[MAXN];

public static boolean[] ans = new boolean[MAXN];

public static void addSpecial(int x, int l, int r, int id) {
    nextq[++cnts] = headq[x];
    headq[x] = cnts;
    ql[cnts] = l;
    qr[cnts] = r;
    qid[cnts] = id;
}
```

```
public static class QueryCmp implements Comparator<int[]> {
    @Override
    public int compare(int[] a, int[] b) {
        if (bi[a[0]] != bi[b[0]]) {
            return bi[a[0]] - bi[b[0]];
        }
        if ((bi[a[0]] & 1) == 1) {
            return a[1] - b[1];
        } else {
            return b[1] - a[1];
        }
    }
}

public static void add(int x) {
    cnt[x]++;
    if (cnt[x] == 1) {
        bitSet1.setOne(x);
        bitSet2.setOne(MAXV - x);
    }
}

public static void del(int x) {
    cnt[x]--;
    if (cnt[x] == 0) {
        bitSet1.setZero(x);
        bitSet2.setZero(MAXV - x);
    }
}

public static boolean calc(int op, int x) {
    if (op == 1) {
        return bitSet1.andOtherMoveRight(bitSet1, x);
    } else if (op == 2) {
        return bitSet1.andOtherMoveRight(bitSet2, MAXV - x);
    } else if (op == 3) {
        for (int f = 1; f * f <= x; f++) {
            if (x % f == 0 && bitSet1.getStatus(f) && bitSet1.getStatus(x / f)) {
                return true;
            }
        }
        return false;
    } else {
```

```
        if (x >= 1) {
            for (int i = 1; i * x <= MAXV; i++) {
                if (bitSet1.getStatus(i) && bitSet1.getStatus(i * x)) {
                    return true;
                }
            }
        }
        return false;
    }
}

public static void compute() {
    int winl = 1, winr = 0;
    for (int i = 1; i <= cntq; i++) {
        int jobl = query[i][0];
        int jobr = query[i][1];
        int jobx = query[i][2];
        int op = query[i][3];
        int id = query[i][4];
        while (winl > jobl) {
            add(arr[--winl]);
        }
        while (winr < jobr) {
            add(arr[++winr]);
        }
        while (winl < jobl) {
            del(arr[winl++]);
        }
        while (winr > jobr) {
            del(arr[winr--]);
        }
        ans[id] = calc(op, jobx);
    }
}

public static void special() {
    for (int x = 1; x < blen; x++) {
        if (headq[x] != 0) {
            Arrays.fill(pre, 0);
            Arrays.fill(dp, 0);
            for (int i = 1; i <= n; i++) {
                int v = arr[i];
                pre[v] = i;
```

```
                dp[i] = dp[i - 1];
                if (v * x <= MAXV) {
                    dp[i] = Math.max(dp[i], pre[v * x]);
                }
                if (v % x == 0) {
                    dp[i] = Math.max(dp[i], pre[v / x]);
                }
            }
            for (int q = headq[x]; q > 0; q = nextq[q]) {
                int l = ql[q];
                int r = qr[q];
                int id = qid[q];
                ans[id] = l <= dp[r];
            }
        }
    }
}

public static void main(String[] args) throws Exception {
    FastReader in = new FastReader(System.in);
    PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));
    n = in.nextInt();
    m = in.nextInt();
    blen = (int) Math.sqrt(MAXV);
    for (int i = 1; i <= n; i++) {
        bi[i] = (i - 1) / blen + 1;
    }
    for (int i = 1; i <= n; i++) {
        arr[i] = in.nextInt();
    }
    for (int i = 1, op, l, r, x; i <= m; i++) {
        op = in.nextInt();
        l = in.nextInt();
        r = in.nextInt();
        x = in.nextInt();
        if (op == 4 && x < blen) {
            addSpecial(x, l, r, i);
        } else {
            query[++cntq][0] = l;
            query[cntq][1] = r;
            query[cntq][2] = x;
            query[cntq][3] = op;
            query[cntq][4] = i;
```

```
        }
    }
    Arrays.sort(query, 1, cntq + 1, new QueryCmp());
    compute();
    special();
    for (int i = 1; i <= m; i++) {
        if (ans[i]) {
            out.println("yuno");
        } else {
            out.println("yumi");
        }
    }
    out.flush();
    out.close();
}

// 读写工具类
static class FastReader {
    private final byte[] buffer = new byte[1 << 16];
    private int ptr = 0, len = 0;
    private final InputStream in;

    FastReader(InputStream in) {
        this.in = in;
    }

    private int readByte() throws IOException {
        if (ptr >= len) {
            len = in.read(buffer);
            ptr = 0;
            if (len <= 0)
                return -1;
        }
        return buffer[ptr++];
    }

    int nextInt() throws IOException {
        int c;
        do {
            c = readByte();
        } while (c <= ' ' && c != -1);
        boolean neg = false;
        if (c == '-') {
```

```
                neg = true;
                c = readByte();
            }
            int val = 0;
            while (c > ' ' && c != -1) {
                val = val * 10 + (c - '0');
                c = readByte();
            }
            return neg ? -val : val;
        }
    }

}
```

==========

文件: Code08_YnoiCornfield2.java

==========

```
// 由乃的玉米田 - 普通莫队算法实现 (C++版本)
// 题目来源: 洛谷 P5355 由乃的玉米田
// 题目链接: https://www.luogu.com.cn/problem/P5355
// 题目大意: 给定一个长度为n的数组arr，接下来有m条查询，查询格式如下：
// 查询 1 l r x : 打印arr[l..r]范围上能否选出两个数，减的结果为x
// 查询 2 l r x : 打印arr[l..r]范围上能否选出两个数，加的结果为x
// 查询 3 l r x : 打印arr[l..r]范围上能否选出两个数，乘的结果为x
// 查询 4 l r x : 打印arr[l..r]范围上能否选出两个数，除的结果为x，并且没有余数
// 选出的这两个数可以是同一个位置的数，答案如果为是，打印 "yuno"，否则打印 "yumi"
// 解题思路: 使用普通莫队算法，结合位运算和特殊处理技术
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
// 相关题目:
// 1. CF617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
// 3. BZOJ2038 [国家集训队]小Z的袜子 - https://www.lydsy.com/JudgeOnline/problem.php?id=2038
// 4. HDU4638 Group - http://acm.hdu.edu.cn/showproblem.php?pid=4638
// 5. AtCoder ABC176 D - Wizard in Maze - https://atcoder.jp/contests/abc176/tasks/abc176_d

package class179;

// 由乃的玉米田，C++版
// 给定一个长度为n的数组arr，接下来有m条查询，查询格式如下
// 查询 1 l r x : 打印arr[l..r]范围上能否选出两个数，减的结果为x
// 查询 2 l r x : 打印arr[l..r]范围上能否选出两个数，加的结果为x
```

```
// 查询 3 l r x : 打印arr[l..r]范围上能否选出两个数，乘的结果为x
// 查询 4 l r x : 打印arr[l..r]范围上能否选出两个数，除的结果为x，并且没有余数
// 选出的这两个数可以是同一个位置的数，答案如果为是，打印 "yuno"，否则打印 "yumi"
// 1 <= 所有数据 <= 10^5
// 测试链接 : https://www.luogu.com.cn/problem/P5355
// 如下实现是C++的版本，C++版本和java版本逻辑完全一样
// 提交如下代码，可以通过所有测试用例

//#include <bits/stdc++.h>
//
//using namespace std;
//
//struct Query {
//    int l, r, x, op, id;
//};
//
//const int MAXN = 100001;
//const int MAXV = 100000;
//const int MAXB = 401;
//int n, m, blen;
//int arr[MAXN;
//int bi[MAXN;
//
//Query query[MAXN;
//int cntq = 0;
//
//int headq[MAXB;
//int nextq[MAXN;
//int ql[MAXN;
//int qr[MAXN;
//int qid[MAXN;
//int cnts = 0;
//
//int cnt[MAXN;
//bitset<MAXN> bitSet1;
//bitset<MAXN> bitSet2;
//
//int pre[MAXN;
//int dp[MAXN;
//
//bool ans[MAXN;
//
//void addSpecial(int x, int l, int r, int id) {
```

```
//    nextq[++cnts] = headq[x];
//    headq[x] = cnts;
//    ql[cnts] = l;
//    qr[cnts] = r;
//    qid[cnts] = id;
//}
//
//bool QueryCmp(const Query &a, const Query &b) {
//    if (bi[a.l] != bi[b.l]) {
//        return bi[a.l] < bi[b.l];
//    }
//    if (bi[a.l] & 1) {
//        return a.r < b.r;
//    } else {
//        return a.r > b.r;
//    }
//}
//
//void add(int x) {
//    cnt[x]++;
//    if (cnt[x] == 1) {
//        bitSet1[x] = 1;
//        bitSet2[MAXV - x] = 1;
//    }
//}
//
//void del(int x) {
//    cnt[x]--;
//    if (cnt[x] == 0) {
//        bitSet1[x] = 0;
//        bitSet2[MAXV - x] = 0;
//    }
//}
//
//bool calc(int op, int x) {
//    if (op == 1) {
//        return (bitSet1 & (bitSet1 >> x)).any();
//    } else if (op == 2) {
//        return (bitSet1 & (bitSet2 >> (MAXV - x))).any();
//    } else if (op == 3) {
//        for (int f = 1; f * f <= x; f++) {
//            if (x % f == 0 && bitSet1[f] && bitSet1[x / f]) {
//                    return true;
```

```
//            }
//        }
//        return false;
//    } else {
//        for (int i = 1; i * x <= MAXV; i++) {
//            if (bitSet1[i] && bitSet1[i * x]) {
//                return true;
//            }
//        }
//        return false;
//    }
//}
//
//void compute() {
//    int winl = 1, winr = 0;
//    for (int i = 1; i <= cntq; i++) {
//        int jobl = query[i].l;
//        int jobr = query[i].r;
//        int jobx = query[i].x;
//        int op = query[i].op;
//        int id = query[i].id;
//        while (winl > jobl) {
//            add(arr[--winl]);
//        }
//        while (winr < jobr) {
//            add(arr[++winr]);
//        }
//        while (winl < jobl) {
//            del(arr[winl++]);
//        }
//        while (winr > jobr) {
//            del(arr[winr--]);
//        }
//        ans[id] = calc(op, jobx);
//    }
//}
//
//void special() {
//    for (int x = 1; x < blen; x++) {
//        if (headq[x] != 0) {
//            memset(pre, 0, sizeof(int) * (MAXV + 1));
//            memset(dp, 0, sizeof(int) * (n + 1));
//            for (int i = 1; i <= n; i++) {
```

```
//                int v = arr[i];
//                pre[v] = i;
//                dp[i] = dp[i-1];
//                if (v * x <= MAXV) {
//                    dp[i] = max(dp[i], pre[v * x]);
//                }
//                if (v % x == 0) {
//                    dp[i] = max(dp[i], pre[v / x]);
//                }
//            }
//            for (int q = headq[x]; q > 0; q = nextq[q]) {
//                int l = ql[q];
//                int r = qr[q];
//                int id = qid[q];
//                ans[id] = (l <= dp[r]);
//            }
//        }
//    }
//}
//
//int main() {
//    ios::sync_with_stdio(false);
//    cin.tie(nullptr);
//    cin >> n >> m;
//    blen = (int)sqrt(MAXV);
//    for (int i = 1; i <= n; i++) {
//        bi[i] = (i - 1) / blen + 1;
//    }
//    for (int i = 1; i <= n; i++) {
//        cin >> arr[i];
//    }
//    for (int i = 1, op, l, r, x; i <= m; i++) {
//        cin >> op >> l >> r >> x;
//        if (op == 4 && x < blen) {
//            addSpecial(x, l, r, i);
//        } else {
//            query[++cntq] = {l, r, x, op, i};
//        }
//    }
//    sort(query + 1, query + cntq + 1, QueryCmp);
//    compute();
//    special();
//    for (int i = 1; i <= m; i++) {
```

```
//        cout << (ans[i] ? "yuno" : "yumi") << '\n';
//    }
//    return 0;
//}
```

==========

文件: Code09_Socks1.java

==========

```
// 小Z的袜子 - 普通莫队算法实现 (Java版本)
// 题目来源: 洛谷 P1494 [国家集训队]小Z的袜子
// 题目链接: https://www.luogu.com.cn/problem/P1494
// 题目大意: 给定一个长度为n的数组，每次查询区间[l,r]内随机选择两个数，求这两个数相等的概率
// 解题思路: 使用普通莫队算法，通过维护区间内每种颜色的个数来计算概率
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 3. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
//    C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
//    Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
//
// 4. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//    Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray3.py
//
```

// 5. 洛谷 P1972 [SDOI2009] HH的项链 - https://www.luogu.com.cn/problem/P1972
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery3.py
//
// 6. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 7. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance3.py
//
// 8. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
//
// 9. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch3.py
//
// 10. 洛谷 P4887 【模板】莫队二次离线（第十四分块(前体)） - https://www.luogu.com.cn/problem/P4887
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline3.py

package class179;

// 小Z的袜子 - 普通莫队算法实现 (Java版本)

```
// 题目来源: BZOJ 2038 [2009 国家集训队]小 Z 的袜子(hose)
// 题目链接: https://www.luogu.com.cn/problem/P1494
// 题目大意: 给定一个长度为 n 的序列，每个元素代表袜子的颜色
// 有 m 次询问，每次询问区间[l,r]中随机抽取两只袜子颜色相同的概率
// 输出要求: 每个询问输出一个分数 A/B 表示概率，要求为最简分数
// 时间复杂度: O(n*sqrt(n))
// 空间复杂度: O(n)

import java.io.*;
import java.util.*;

public class Code09_Socks1 {
    public static int MAXN = 50010;
    public static int n, m;
    public static int[] arr = new int[MAXN];
    public static int[][] query = new int[MAXN][3];
    public static int[] bi = new int[MAXN];
    public static long[] cnt = new long[MAXN]; // 记录每种颜色的出现次数
    public static long curAns = 0; // 当前区间的答案（分子）
    public static long[] ansA = new long[MAXN]; // 答案分子
    public static long[] ansB = new long[MAXN]; // 答案分母

    // 查询排序比较器
    public static class QueryCmp implements Comparator<int[]> {
        @Override
        public int compare(int[] a, int[] b) {
            if (bi[a[0]] != bi[b[0]]) {
                return bi[a[0]] - bi[b[0]];
            }
            if ((bi[a[0]] & 1) == 1) {
                return a[1] - b[1];
            } else {
                return b[1] - a[1];
            }
        }
    }

    // 添加元素到区间
    public static void add(int color) {
        // 当添加一个颜色时，该颜色对答案的贡献增加 2*cnt[color]
        // 因为对于每一对已存在的该颜色袜子，新加入的袜子都能和它们组成一对
        curAns += cnt[color] * 2;
        cnt[color]++;
```

```
}

// 从区间中删除元素
public static void del(int color) {
    cnt[color]--;
    // 当删除一个颜色时，该颜色对答案的贡献减少 2*cnt[color]
    // 因为对于每一对剩余的该颜色袜子，被删除的袜子不能再和它们组成一对
    curAns -= cnt[color] * 2;
}

// 计算查询结果
public static void compute() {
    int winl = 1, winr = 0; // 当前维护的区间 [winl, winr]
    for (int i = 1; i <= m; i++) {
        int jobl = query[i][0]; // 目标区间左端点
        int jobr = query[i][1]; // 目标区间右端点
        int id = query[i][2];   // 查询编号

        // 扩展左边界
        while (winl > jobl) {
            add(arr[--winl]);
        }

        // 扩展右边界
        while (winr < jobr) {
            add(arr[++winr]);
        }

        // 收缩左边界
        while (winl < jobl) {
            del(arr[winl++]);
        }

        // 收缩右边界
        while (winr > jobr) {
            del(arr[winr--]);
        }

        // 特殊情况：区间长度为 1 时概率为 0
        if (jobl == jobr) {
            ansA[id] = 0;
            ansB[id] = 1;
            continue;
```

```
        }

        // 计算答案
        ansA[id] = curAns;
        long len = jobr - jobl + 1;
        ansB[id] = len * (len - 1);

        // 化简分数
        long gcd = gcd(ansA[id], ansB[id]);
        ansA[id] /= gcd;
        ansB[id] /= gcd;
    }
}

// 求最大公约数
public static long gcd(long a, long b) {
    return b == 0 ? a : gcd(b, a % b);
}

// 预处理
public static void prepare() {
    int blen = (int) Math.sqrt(n);
    for (int i = 1; i <= n; i++) {
        bi[i] = (i - 1) / blen + 1;
    }
    Arrays.sort(query, 1, m + 1, new QueryCmp());
}

public static void main(String[] args) throws Exception {
    FastReader in = new FastReader();
    PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));
    n = in.nextInt();
    m = in.nextInt();

    for (int i = 1; i <= n; i++) {
        arr[i] = in.nextInt();
    }

    for (int i = 1; i <= m; i++) {
        query[i][0] = in.nextInt();
        query[i][1] = in.nextInt();
        query[i][2] = i;
    }
```

```
        prepare();
        compute();

        for (int i = 1; i <= m; i++) {
            out.println(ansA[i] + "/" + ansB[i]);
        }

        out.flush();
        out.close();
    }

    // 读写工具类
    static class FastReader {
        final private int BUFFER_SIZE = 1 << 16;
        private final InputStream in;
        private final byte[] buffer;
        private int ptr, len;

        public FastReader() {
            in = System.in;
            buffer = new byte[BUFFER_SIZE];
            ptr = len = 0;
        }

        private boolean hasNextByte() throws IOException {
            if (ptr < len)
                return true;
            ptr = 0;
            len = in.read(buffer);
            return len > 0;
        }

        private byte readByte() throws IOException {
            if (!hasNextByte())
                return -1;
            return buffer[ptr++];
        }

        int nextInt() throws IOException {
            int c;
            do {
                c = readByte();
```

```
            } while (c <= ' ' && c != -1);
            boolean neg = false;
            if (c == '-') {
                neg = true;
                c = readByte();
            }
            int val = 0;
            while (c > ' ' && c != -1) {
                val = val * 10 + (c - '0');
                c = readByte();
            }
            return neg ? -val : val;
        }
    }
}
```

==========

文件: Code09_Socks2.cpp

==========

```
// 小Z的袜子 - 普通莫队算法实现 (C++版本)
// 题目来源: 洛谷 P1494 [国家集训队]小Z的袜子
// 题目链接: https://www.luogu.com.cn/problem/P1494
// 题目大意: 给定一个长度为n的序列，每个元素代表袜子的颜色
// 有m次询问，每次询问区间[l,r]中随机抽取两只袜子颜色相同的概率
// 输出要求: 每个询问输出一个分数A/B表示概率，要求为最简分数
// 解题思路: 使用普通莫队算法，通过维护区间内每种颜色的出现次数来计算概率
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
// 相关题目:
// 1. CF617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
// 3. BZOJ2038 [国家集训队]小Z的袜子 - https://www.lydsy.com/JudgeOnline/problem.php?id=2038
// 4. HDU4638 Group - http://acm.hdu.edu.cn/showproblem.php?pid=4638
// 5. AtCoder ABC176 D - Wizard in Maze - https://atcoder.jp/contests/abc176/tasks/abc176_d

//#include <iostream>
//#include <algorithm>
//#include <cmath>
//using namespace std;
//
//// 小Z的袜子 - 普通莫队算法实现 (C++版本)
//// 题目来源: BZOJ 2038 [2009国家集训队]小Z的袜子(hose)
```

```
//// 题目链接: https://www.luogu.com.cn/problem/P1494
//// 题目大意: 给定一个长度为n的序列，每个元素代表袜子的颜色
//// 有m次询问，每次询问区间[l,r]中随机抽取两只袜子颜色相同的概率
//// 输出要求: 每个询问输出一个分数A/B表示概率，要求为最简分数
//// 时间复杂度: O(n*sqrt(n))
//// 空间复杂度: O(n)
//
//const int MAXN = 50010;
//int n, m;
//int arr[MAXN];
//struct Query {
//    int l, r, id;
//} query[MAXN];
//
//int bi[MAXN];
//long long cnt[MAXN]; // 记录每种颜色的出现次数
//long long curAns = 0; // 当前区间的答案（分子）
//long long ansA[MAXN]; // 答案分子
//long long ansB[MAXN]; // 答案分母
//
//// 分块大小
//int block_size;
//
//// 查询排序比较函数
//bool cmp(const Query& a, const Query& b) {
//    if (bi[a.l] != bi[b.l]) {
//        return bi[a.l] < bi[b.l];
//    }
//    if (bi[a.l] & 1) {
//        return a.r < b.r;
//    } else {
//        return a.r > b.r;
//    }
//}
//
//// 求最大公约数
//long long gcd(long long a, long long b) {
//    return b == 0 ? a : gcd(b, a % b);
//}
//
//// 添加元素到区间
//void add(int color) {
//    // 当添加一个颜色时，该颜色对答案的贡献增加2*cnt[color]
```

```
//    // 因为对于每一对已存在的该颜色袜子，新加入的袜子都能和它们组成一对
//    curAns += cnt[color] * 2;
//    cnt[color]++;
//}
//
//// 从区间中删除元素
//void del(int color) {
//    cnt[color]--;
//    // 当删除一个颜色时，该颜色对答案的贡献减少 2*cnt[color]
//    // 因为对于每一对剩余的该颜色袜子，被删除的袜子不能再和它们组成一对
//    curAns -= cnt[color] * 2;
//}
//
//// 计算查询结果
//void compute() {
//    int winl = 1, winr = 0; // 当前维护的区间 [winl, winr]
//    for (int i = 1; i <= m; i++) {
//        int jobl = query[i].l; // 目标区间左端点
//        int jobr = query[i].r; // 目标区间右端点
//        int id = query[i].id;  // 查询编号
//
//        // 扩展左边界
//        while (winl > jobl) {
//            add(arr[--winl]);
//        }
//
//        // 扩展右边界
//        while (winr < jobr) {
//            add(arr[++winr]);
//        }
//
//        // 收缩左边界
//        while (winl < jobl) {
//            del(arr[winl++]);
//        }
//
//        // 收缩右边界
//        while (winr > jobr) {
//            del(arr[winr--]);
//        }
//
//        // 特殊情况：区间长度为 1 时概率为 0
//        if (jobl == jobr) {
```

```
//            ansA[id] = 0;
//            ansB[id] = 1;
//            continue;
//        }
//
//        // 计算答案
//        ansA[id] = curAns;
//        long long len = jobr - jobl + 1;
//        ansB[id] = len * (len - 1);
//
//        // 化简分数
//        long long g = gcd(ansA[id], ansB[id]);
//        ansA[id] /= g;
//        ansB[id] /= g;
//    }
//}
//
//// 预处理
//void prepare() {
//    block_size = (int)sqrt((double)n);
//    for (int i = 1; i <= n; i++) {
//        bi[i] = (i - 1) / block_size + 1;
//    }
//    sort(query + 1, query + m + 1, cmp);
//}
//
//int main() {
//    ios::sync_with_stdio(false);
//    cin.tie(nullptr);
//    cin >> n >> m;
//
//    for (int i = 1; i <= n; i++) {
//        cin >> arr[i];
//    }
//
//    for (int i = 1; i <= m; i++) {
//        cin >> query[i].l >> query[i].r;
//        query[i].id = i;
//    }
//
//    prepare();
//    compute();
//
```

```
//    for (int i = 1; i <= m; i++) {
//        cout << ansA[i] << "/" << ansB[i] << '\n';
//    }
//
//    return 0;
//}
```

==========

文件: Code09_Socks3.py

==========

```
# 小Z的袜子 - 普通莫队算法实现 (Python版本)
# 题目来源: 洛谷 P1494 [国家集训队]小Z的袜子
# 题目链接: https://www.luogu.com.cn/problem/P1494
# 题目大意: 给定一个长度为n的序列，每个元素代表袜子的颜色
# 有m次询问，每次询问区间[l,r]中随机抽取两只袜子颜色相同的概率
# 输出要求: 每个询问输出一个分数A/B表示概率，要求为最简分数
# 解题思路: 使用普通莫队算法，通过维护区间内每种颜色的出现次数来计算概率
# 时间复杂度: O(n*sqrt(m))
# 空间复杂度: O(n)
# 相关题目:
# 1. CF617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
# 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
# 3. BZOJ2038 [国家集训队]小Z的袜子 - https://www.lydsy.com/JudgeOnline/problem.php?id=2038
# 4. HDU4638 Group - http://acm.hdu.edu.cn/showproblem.php?pid=4638
# 5. AtCoder ABC176 D - Wizard in Maze - https://atcoder.jp/contests/abc176/tasks/abc176_d

import math
import sys
from fractions import Fraction
from typing import List, Optional

def main():
    # 读取输入
    n, m = map(int, sys.stdin.readline().split())
    arr = list(map(int, sys.stdin.readline().split()))

    # 存储查询
    queries = []
    for i in range(m):
        l, r = map(int, sys.stdin.readline().split())
        queries.append((l, r, i))
```

```
# 莫队算法预处理
block_size = int(math.sqrt(n))

# 查询排序函数
def query_sort_key(query):
    l, r, idx = query
    block_id = (l - 1) // block_size
    if block_id % 2 == 1:
        return (block_id, r)
    else:
        return (block_id, -r)

# 按照莫队算法的顺序排序查询
queries.sort(key=query_sort_key)

# 初始化变量
cnt = [0] * (max(arr) + 1)  # 记录每种颜色的出现次数
cur_ans = 0  # 当前区间的答案（分子）
answers: List[Optional[Fraction]] = [None] * m  # 存储答案

# 当前维护的区间 [win_l, win_r]
win_l, win_r = 1, 0

# 处理每个查询
for job_l, job_r, idx in queries:
    # 调整左边界
    while win_l > job_l:
        win_l -= 1
        color = arr[win_l - 1]  # arr 索引从 0 开始
        # 当添加一个颜色时，该颜色对答案的贡献增加 2*cnt[color]
        # 因为对于每一对已存在的该颜色袜子，新加入的袜子都能和它们组成一对
        cur_ans += cnt[color] * 2
        cnt[color] += 1

    while win_l < job_l:
        color = arr[win_l - 1]  # arr 索引从 0 开始
        cnt[color] -= 1
        # 当删除一个颜色时，该颜色对答案的贡献减少 2*cnt[color]
        # 因为对于每一对剩余的该颜色袜子，被删除的袜子不能再和它们组成一对
        cur_ans -= cnt[color] * 2
        win_l += 1

    # 调整右边界
```

```
        while win_r < job_r:
            win_r += 1
            color = arr[win_r - 1]  # arr索引从0开始
            # 当添加一个颜色时，该颜色对答案的贡献增加2*cnt[color]
            cur_ans += cnt[color] * 2
            cnt[color] += 1

        while win_r > job_r:
            color = arr[win_r - 1]  # arr索引从0开始
            cnt[color] -= 1
            # 当删除一个颜色时，该颜色对答案的贡献减少2*cnt[color]
            cur_ans -= cnt[color] * 2
            win_r -= 1

        # 特殊情况：区间长度为1时概率为0
        if job_l == job_r:
            answers[idx] = Fraction(0, 1)
            continue

        # 计算答案
        length = job_r - job_l + 1
        denominator = length * (length - 1)
        answers[idx] = Fraction(cur_ans, denominator)

    # 输出答案
    for ans in answers:
        if ans is not None:
            print(f"{ans.numerator}/{ans.denominator}")

if __name__ == "__main__":
    main()
```

==========

文件: Code10_DQuery1.java

==========

```
// D-query - 普通莫队算法实现 (Java版本)
// 题目来源: SPOJ DQUERY - D-query
// 题目链接: https://www.spoj.com/problems/DQUERY/
// 题目大意: 给定一个长度为n的数组，每次查询区间[l,r]内有多少种不同的数字
// 解题思路: 使用普通莫队算法，通过维护区间内不同数字的个数来回答查询
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
```

```
//
// 相关题目链接:
// 1. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 2. 洛谷 P1972 [SDOI2009] HH的项链 - https://www.luogu.com.cn/problem/P1972
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery3.py
//
// 3. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
//    C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
//    Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
//
// 4. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 5. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//    Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 6. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 7. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
//    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance1.java
//    C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance2.cpp
//    Python解答: https://github.com/algorithm-
```

journey/class179/blob/main/Code16_ColorMaintenance3.py
//
// 8. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
//
// 9. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
//   Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch1.java
//   C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch2.cpp
//   Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch3.py
//
// 10. 洛谷 P4887 【模板】莫队二次离线（第十四分块(前体)） -
https://www.luogu.com.cn/problem/P4887
//   Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code18_SecondaryOffline1.java
//   C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code18_SecondaryOffline2.cpp
//   Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code18_SecondaryOffline3.py

package class179;

// D-query - 普通莫队算法实现 (Java 版本)
// 题目来源: SPOJ DQUERY - D-query
// 题目链接: https://www.spoj.com/problems/DQUERY/
// 题目大意: 给定一个长度为 n 的数组，每次查询区间[l,r]内有多少种不同的数字
// 时间复杂度: O(n*sqrt(n))
// 空间复杂度: O(n)

import java.io.*;
import java.util.*;

public class Code10_DQuery1 {
    public static int MAXN = 30010;
    public static int MAXV = 1000010;
    public static int n, m;
    public static int[] arr = new int[MAXN];
    public static int[][] query = new int[MAXN][3];
    public static int[] bi = new int[MAXN];

```
public static int[] cnt = new int[MAXV]; // 记录每种数值的出现次数
public static int curAns = 0; // 当前区间的答案（不同数字的个数）
public static int[] ans = new int[MAXN]; // 存储答案

// 查询排序比较器
public static class QueryCmp implements Comparator<int[]> {
    @Override
    public int compare(int[] a, int[] b) {
        if (bi[a[0]] != bi[b[0]]) {
            return bi[a[0]] - bi[b[0]];
        }
        if ((bi[a[0]] & 1) == 1) {
            return a[1] - b[1];
        } else {
            return b[1] - a[1];
        }
    }
}

// 添加元素到区间
public static void add(int value) {
    // 如果这是该数值第一次出现，则不同数字的个数增加 1
    if (cnt[value] == 0) {
        curAns++;
    }
    cnt[value]++;
}

// 从区间中删除元素
public static void del(int value) {
    cnt[value]--;
    // 如果该数值不再出现，则不同数字的个数减少 1
    if (cnt[value] == 0) {
        curAns--;
    }
}

// 计算查询结果
public static void compute() {
    int winl = 1, winr = 0; // 当前维护的区间 [winl, winr]
    for (int i = 1; i <= m; i++) {
        int jobl = query[i][0]; // 目标区间左端点
        int jobr = query[i][1]; // 目标区间右端点
```

```
        int id = query[i][2];   // 查询编号

        // 扩展左边界
        while (winl > jobl) {
            add(arr[--winl]);
        }

        // 扩展右边界
        while (winr < jobr) {
            add(arr[++winr]);
        }

        // 收缩左边界
        while (winl < jobl) {
            del(arr[winl++]);
        }

        // 收缩右边界
        while (winr > jobr) {
            del(arr[winr--]);
        }

        ans[id] = curAns;
    }
}

// 预处理
public static void prepare() {
    int blen = (int) Math.sqrt(n);
    for (int i = 1; i <= n; i++) {
        bi[i] = (i - 1) / blen + 1;
    }
    Arrays.sort(query, 1, m + 1, new QueryCmp());
}

public static void main(String[] args) throws Exception {
    FastReader in = new FastReader();
    PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));
    n = in.nextInt();

    for (int i = 1; i <= n; i++) {
        arr[i] = in.nextInt();
    }
```

```
        m = in.nextInt();
        for (int i = 1; i <= m; i++) {
            query[i][0] = in.nextInt();
            query[i][1] = in.nextInt();
            query[i][2] = i;
        }

        prepare();
        compute();

        for (int i = 1; i <= m; i++) {
            out.println(ans[i]);
        }

        out.flush();
        out.close();
    }

    // 读写工具类
    static class FastReader {
        final private int BUFFER_SIZE = 1 << 16;
        private final InputStream in;
        private final byte[] buffer;
        private int ptr, len;

        public FastReader() {
            in = System.in;
            buffer = new byte[BUFFER_SIZE];
            ptr = len = 0;
        }

        private boolean hasNextByte() throws IOException {
            if (ptr < len)
                return true;
            ptr = 0;
            len = in.read(buffer);
            return len > 0;
        }

        private byte readByte() throws IOException {
            if (!hasNextByte())
                return -1;
```

```
            return buffer[ptr++];
        }

        int nextInt() throws IOException {
            int c;
            do {
                c = readByte();
            } while (c <= ' ' && c != -1);
            boolean neg = false;
            if (c == '-') {
                neg = true;
                c = readByte();
            }
            int val = 0;
            while (c > ' ' && c != -1) {
                val = val * 10 + (c - '0');
                c = readByte();
            }
            return neg ? -val : val;
        }
    }
}    static class FastReader {
        final private int BUFFER_SIZE = 1 << 16;
        private final InputStream in;
        private final byte[] buffer;
        private int ptr, len;

        public FastReader() {
            in = System.in;
            buffer = new byte[BUFFER_SIZE];
            ptr = len = 0;
        }

        private boolean hasNextByte() throws IOException {
            if (ptr < len)
                return true;
            ptr = 0;
            len = in.read(buffer);
            return len > 0;
        }

        private byte readByte() throws IOException {
            if (!hasNextByte())
```

```
                return -1;
            return buffer[ptr++];
        }

        int nextInt() throws IOException {
            int c;
            do {
                c = readByte();
            } while (c <= ' ' && c != -1);
            boolean neg = false;
            if (c == '-') {
                neg = true;
                c = readByte();
            }
            int val = 0;
            while (c > ' ' && c != -1) {
                val = val * 10 + (c - '0');
                c = readByte();
            }
            return neg ? -val : val;
        }
    }
}
```

==========

文件: Code10_DQuery2.cpp

==========

```
// D-query - 普通莫队算法实现 (C++版本)
// 题目来源: SPOJ DQUERY - D-query
// 题目链接: https://www.spoj.com/problems/DQUERY/
// 题目大意: 给定一个长度为n的数组，每次查询区间[l,r]内有多少种不同的数字
// 解题思路: 使用普通莫队算法，通过维护区间内不同数字的个数来回答查询
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
// 相关题目:
// 1. CF617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
// 2. BZOJ2038 [国家集训队]小Z的袜子 - https://www.lydsy.com/JudgeOnline/problem.php?id=2038
// 3. HDU4638 Group - http://acm.hdu.edu.cn/showproblem.php?pid=4638
// 4. AtCoder ABC176 D - Wizard in Maze - https://atcoder.jp/contests/abc176/tasks/abc176_d
// 5. Luogu P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494

//#include <cstdio>
```

```
//#include <algorithm>
//#include <cmath>
//using namespace std;
//
//// D-query - 普通莫队算法实现 (C++版本)
//// 题目来源: SPOJ DQUERY - D-query
//// 题目链接: https://www.spoj.com/problems/DQUERY/
//// 题目大意: 给定一个长度为n的数组，每次查询区间[l,r]内有多少种不同的数字
//// 时间复杂度: O(n*sqrt(n))
//// 空间复杂度: O(n)
//
//const int MAXN = 30010;
//const int MAXV = 1000010;
//int n, m;
//int arr[MAXN];
//struct Query {
//    int l, r, id;
//} query[MAXN];
//
//int bi[MAXN];
//int cnt[MAXV]; // 记录每种数值的出现次数
//int curAns = 0; // 当前区间的答案（不同数字的个数）
//int ans[MAXN]; // 存储答案
//
//// 分块大小
//int block_size;
//
//// 查询排序比较函数
//bool cmp(const Query& a, const Query& b) {
//    if (bi[a.l] != bi[b.l]) {
//        return bi[a.l] < bi[b.l];
//    }
//    if (bi[a.l] & 1) {
//        return a.r < b.r;
//    } else {
//        return a.r > b.r;
//    }
//}
//
//// 添加元素到区间
//void add(int value) {
//    // 如果这是该数值第一次出现，则不同数字的个数增加1
//    if (cnt[value] == 0) {
```

```
//        curAns++;
//    }
//    cnt[value]++;
//}
//
//// 从区间中删除元素
//void del(int value) {
//    cnt[value]--;
//    // 如果该数值不再出现，则不同数字的个数减少 1
//    if (cnt[value] == 0) {
//        curAns--;
//    }
//}
//
//// 计算查询结果
//void compute() {
//    int winl = 1, winr = 0; // 当前维护的区间 [winl, winr]
//    for (int i = 1; i <= m; i++) {
//        int jobl = query[i].l; // 目标区间左端点
//        int jobr = query[i].r; // 目标区间右端点
//        int id = query[i].id;  // 查询编号
//
//        // 扩展左边界
//        while (winl > jobl) {
//            add(arr[--winl]);
//        }
//
//        // 扩展右边界
//        while (winr < jobr) {
//            add(arr[++winr]);
//        }
//
//        // 收缩左边界
//        while (winl < jobl) {
//            del(arr[winl++]);
//        }
//
//        // 收缩右边界
//        while (winr > jobr) {
//            del(arr[winr--]);
//        }
//
//        ans[id] = curAns;
```

```
//    }
//}
//
//// 预处理
//void prepare() {
//    block_size = (int)sqrt((double)n);
//    for (int i = 1; i <= n; i++) {
//        bi[i] = (i - 1) / block_size + 1;
//    }
//    sort(query + 1, query + m + 1, cmp);
//}
//
//int main() {
//    scanf("%d", &n);
//
//    for (int i = 1; i <= n; i++) {
//        scanf("%d", &arr[i]);
//    }
//
//    scanf("%d", &m);
//    for (int i = 1; i <= m; i++) {
//        scanf("%d%d", &query[i].l, &query[i].r);
//        query[i].id = i;
//    }
//
//    prepare();
//    compute();
//
//    for (int i = 1; i <= m; i++) {
//        printf("%d\n", ans[i]);
//    }
//
//    return 0;
//}
```

==========

文件: Code10_DQuery3.py

==========

```
# D-query - 普通莫队算法实现 (Python 版本)
# 题目来源: SPOJ DQUERY - D-query
# 题目链接: https://www.spoj.com/problems/DQUERY/
# 题目大意: 给定一个长度为 n 的数组，每次查询区间[l,r]内有多少种不同的数字
```

```
# 解题思路: 使用普通莫队算法，通过维护区间内不同数字的个数来回答查询
# 时间复杂度: O(n*sqrt(m))
# 空间复杂度: O(n)
# 相关题目:
# 1. CF617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
# 2. BZOJ2038 [国家集训队]小Z的袜子 - https://www.lydsy.com/JudgeOnline/problem.php?id=2038
# 3. HDU4638 Group - http://acm.hdu.edu.cn/showproblem.php?pid=4638
# 4. AtCoder ABC176 D - Wizard in Maze - https://atcoder.jp/contests/abc176/tasks/abc176_d
# 5. Luogu P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494

import math
import sys

def main():
    # 读取输入
    n = int(sys.stdin.readline())
    arr = list(map(int, sys.stdin.readline().split()))

    # 存储查询
    m = int(sys.stdin.readline())
    queries = []
    for i in range(m):
        l, r = map(int, sys.stdin.readline().split())
        queries.append((l, r, i))

    # 莫队算法预处理
    block_size = int(math.sqrt(n))

    # 查询排序函数
    def query_sort_key(query):
        l, r, idx = query
        block_id = (l - 1) // block_size
        if block_id % 2 == 1:
            return (block_id, r)
        else:
            return (block_id, -r)

    # 按照莫队算法的顺序排序查询
    queries.sort(key=query_sort_key)

    # 初始化变量
    cnt = [0] * (max(arr) + 1)  # 记录每种数值的出现次数
    cur_ans = 0  # 当前区间的答案（不同数字的个数）
```

```
answers = [0] * m  # 存储答案

# 当前维护的区间 [win_l, win_r]
win_l, win_r = 1, 0

# 处理每个查询
for job_l, job_r, idx in queries:
    # 调整左边界
    while win_l > job_l:
        win_l -= 1
        value = arr[win_l - 1]  # arr 索引从 0 开始
        # 如果这是该数值第一次出现，则不同数字的个数增加 1
        if cnt[value] == 0:
            cur_ans += 1
        cnt[value] += 1

    while win_l < job_l:
        value = arr[win_l - 1]  # arr 索引从 0 开始
        cnt[value] -= 1
        # 如果该数值不再出现，则不同数字的个数减少 1
        if cnt[value] == 0:
            cur_ans -= 1
        win_l += 1

    # 调整右边界
    while win_r < job_r:
        win_r += 1
        value = arr[win_r - 1]  # arr 索引从 0 开始
        # 如果这是该数值第一次出现，则不同数字的个数增加 1
        if cnt[value] == 0:
            cur_ans += 1
        cnt[value] += 1

    while win_r > job_r:
        value = arr[win_r - 1]  # arr 索引从 0 开始
        cnt[value] -= 1
        # 如果该数值不再出现，则不同数字的个数减少 1
        if cnt[value] == 0:
            cur_ans -= 1
        win_r -= 1

    answers[idx] = cur_ans
```

```
    # 输出答案
    for ans in answers:
        print(ans)

if __name__ == "__main__":
    main()
```

==========

文件: Code11_PowerfulArray1.java

==========

```
// Powerful Array - 普通莫队算法实现 (Java版本)
// 题目来源: CodeForces 86D Powerful Array
// 题目链接: https://codeforces.com/problemset/problem/86/D
// 题目大意: 给定一个长度为n的数组，每次查询区间[l,r]内所有数字的贡献和，数字x的贡献为cnt[x]*cnt[x]*x
// 解题思路: 使用普通莫队算法，通过维护区间内每种数字的出现次数来计算贡献
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 3. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 4. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
```

//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
//
// 5. 洛谷 P1972 [SDOI2009] HH 的项链 - https://www.luogu.com.cn/problem/P1972
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery3.py
//
// 6. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 7. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance3.py
//
// 8. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
//
// 9. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch3.py
//
// 10. 洛谷 P4887 【模板】莫队二次离线（第十四分块(前体)） - https://www.luogu.com.cn/problem/P4887
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline3.py

```
package class179;

// Powerful Array - 普通莫队算法实现 (Java 版本)
// 题目来源: Codeforces 86D - Powerful array
// 题目链接: https://codeforces.com/problemset/problem/86/D
// 题目大意: 给定一个长度为 n 的数组，每次查询区间[l,r]内所有数字的贡献和
// 每个数字 x 的贡献为 (出现次数)^2 * x
// 时间复杂度: O(n*sqrt(n))
// 空间复杂度: O(n)

import java.io.*;
import java.util.*;

public class Code11_PowerfulArray1 {
    public static int MAXN = 200010;
    public static int MAXV = 1000010;
    public static int n, m;
    public static int[] arr = new int[MAXN];
    public static int[][] query = new int[MAXN][3];
    public static int[] bi = new int[MAXN];
    public static long[] cnt = new long[MAXV]; // 记录每种数值的出现次数
    public static long curAns = 0; // 当前区间的答案
    public static long[] ans = new long[MAXN]; // 存储答案

    // 查询排序比较器
    public static class QueryCmp implements Comparator<int[]> {
        @Override
        public int compare(int[] a, int[] b) {
            if (bi[a[0]] != bi[b[0]]) {
                return bi[a[0]] - bi[b[0]];
            }
            if ((bi[a[0]] & 1) == 1) {
                return a[1] - b[1];
            } else {
                return b[1] - a[1];
            }
        }
    }

    // 添加元素到区间
    public static void add(int value) {
        // 当添加一个数值时，先从当前答案中减去旧的贡献
        curAns -= cnt[value] * cnt[value] * value;
```

```
    // 增加该数值的计数
    cnt[value]++;
    // 再加上新的贡献
    curAns += cnt[value] * cnt[value] * value;
}

// 从区间中删除元素
public static void del(int value) {
    // 当删除一个数值时，先从当前答案中减去旧的贡献
    curAns -= cnt[value] * cnt[value] * value;
    // 减少该数值的计数
    cnt[value]--;
    // 再加上新的贡献
    curAns += cnt[value] * cnt[value] * value;
}

// 计算查询结果
public static void compute() {
    int winl = 1, winr = 0; // 当前维护的区间 [winl, winr]
    for (int i = 1; i <= m; i++) {
        int jobl = query[i][0]; // 目标区间左端点
        int jobr = query[i][1]; // 目标区间右端点
        int id = query[i][2];   // 查询编号

        // 扩展左边界
        while (winl > jobl) {
            add(arr[--winl]);
        }

        // 扩展右边界
        while (winr < jobr) {
            add(arr[++winr]);
        }

        // 收缩左边界
        while (winl < jobl) {
            del(arr[winl++]);
        }

        // 收缩右边界
        while (winr > jobr) {
            del(arr[winr--]);
        }
```

```
            ans[id] = curAns;
        }
    }

    // 预处理
    public static void prepare() {
        int blen = (int) Math.sqrt(n);
        for (int i = 1; i <= n; i++) {
            bi[i] = (i - 1) / blen + 1;
        }
        Arrays.sort(query, 1, m + 1, new QueryCmp());
    }

    public static void main(String[] args) throws Exception {
        FastReader in = new FastReader();
        PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));
        n = in.nextInt();
        m = in.nextInt();

        for (int i = 1; i <= n; i++) {
            arr[i] = in.nextInt();
        }

        for (int i = 1; i <= m; i++) {
            query[i][0] = in.nextInt();
            query[i][1] = in.nextInt();
            query[i][2] = i;
        }

        prepare();
        compute();

        for (int i = 1; i <= m; i++) {
            out.println(ans[i]);
        }

        out.flush();
        out.close();
    }

    // 读写工具类
    static class FastReader {
```

```
    final private int BUFFER_SIZE = 1 << 16;
    private final InputStream in;
    private final byte[] buffer;
    private int ptr, len;

    public FastReader() {
        in = System.in;
        buffer = new byte[BUFFER_SIZE];
        ptr = len = 0;
    }

    private boolean hasNextByte() throws IOException {
        if (ptr < len)
            return true;
        ptr = 0;
        len = in.read(buffer);
        return len > 0;
    }

    private byte readByte() throws IOException {
        if (!hasNextByte())
            return -1;
        return buffer[ptr++];
    }

    int nextInt() throws IOException {
        int c;
        do {
            c = readByte();
        } while (c <= ' ' && c != -1);
        boolean neg = false;
        if (c == '-') {
            neg = true;
            c = readByte();
        }
        int val = 0;
        while (c > ' ' && c != -1) {
            val = val * 10 + (c - '0');
            c = readByte();
        }
        return neg ? -val : val;
    }
}
```

```
}

==========================================

文件: Code11_PowerfulArray2.cpp
==========================================
// Powerful Array - 普通莫队算法实现 (C++版本)
// 题目来源: Codeforces 86D - Powerful array
// 题目链接: https://codeforces.com/problemset/problem/86/D
// 题目大意: 给定一个长度为n的数组，每次查询区间[l,r]内所有数字的贡献和
// 每个数字x的贡献为 (出现次数)^2 * x
// 解题思路: 使用普通莫队算法，通过维护区间内每个数字的出现次数来计算贡献和
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
// 相关题目:
// 1. CF617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
// 3. BZOJ2038 [国家集训队]小Z的袜子 - https://www.lydsy.com/JudgeOnline/problem.php?id=2038
// 4. HDU4638 Group - http://acm.hdu.edu.cn/showproblem.php?pid=4638
// 5. AtCoder ABC176 D - Wizard in Maze - https://atcoder.jp/contests/abc176/tasks/abc176_d

//#include <cstdio>
//#include <algorithm>
//#include <cmath>
//using namespace std;
//
//// Powerful Array - 普通莫队算法实现 (C++版本)
//// 题目来源: Codeforces 86D - Powerful array
//// 题目链接: https://codeforces.com/problemset/problem/86/D
//// 题目大意: 给定一个长度为n的数组，每次查询区间[l,r]内所有数字的贡献和
//// 每个数字x的贡献为 (出现次数)^2 * x
//// 时间复杂度: O(n*sqrt(n))
//// 空间复杂度: O(n)
//
//const int MAXN = 200010;
//const int MAXV = 1000010;
//int n, m;
//int arr[MAXN];
//struct Query {
//    int l, r, id;
//} query[MAXN];
//
//int bi[MAXN;
```

```
//long long cnt[MAXV; // 记录每种数值的出现次数
//long long curAns = 0; // 当前区间的答案
//long long ans[MAXN; // 存储答案
//
//// 分块大小
//int block_size;
//
//// 查询排序比较函数
//bool cmp(const Query& a, const Query& b) {
//    if (bi[a.l] != bi[b.l]) {
//        return bi[a.l] < bi[b.l];
//    }
//    if (bi[a.l] & 1) {
//        return a.r < b.r;
//    } else {
//        return a.r > b.r;
//    }
//}
//
//// 添加元素到区间
//void add(int value) {
//    // 当添加一个数值时，先从当前答案中减去旧的贡献
//    curAns -= 1LL * cnt[value] * cnt[value] * value;
//    // 增加该数值的计数
//    cnt[value]++;
//    // 再加上新的贡献
//    curAns += 1LL * cnt[value] * cnt[value] * value;
//}
//
//// 从区间中删除元素
//void del(int value) {
//    // 当删除一个数值时，先从当前答案中减去旧的贡献
//    curAns -= 1LL * cnt[value] * cnt[value] * value;
//    // 减少该数值的计数
//    cnt[value]--;
//    // 再加上新的贡献
//    curAns += 1LL * cnt[value] * cnt[value] * value;
//}
//
//// 计算查询结果
//void compute() {
//    int winl = 1, winr = 0; // 当前维护的区间 [winl, winr]
//    for (int i = 1; i <= m; i++) {
```

```
//        int jobl = query[i].l; // 目标区间左端点
//        int jobr = query[i].r; // 目标区间右端点
//        int id = query[i].id;  // 查询编号
//
//        // 扩展左边界
//        while (winl > jobl) {
//            add(arr[--winl]);
//        }
//
//        // 扩展右边界
//        while (winr < jobr) {
//            add(arr[++winr]);
//        }
//
//        // 收缩左边界
//        while (winl < jobl) {
//            del(arr[winl++]);
//        }
//
//        // 收缩右边界
//        while (winr > jobr) {
//            del(arr[winr--]);
//        }
//
//        ans[id] = curAns;
//    }
//}
//
//// 预处理
//void prepare() {
//    block_size = (int)sqrt((double)n);
//    for (int i = 1; i <= n; i++) {
//        bi[i] = (i - 1) / block_size + 1;
//    }
//    sort(query + 1, query + m + 1, cmp);
//}
//
//int main() {
//    scanf("%d%d", &n, &m);
//
//    for (int i = 1; i <= n; i++) {
//        scanf("%d", &arr[i]);
//    }
```

```
//
//    for (int i = 1; i <= m; i++) {
//        scanf("%d%d", &query[i].l, &query[i].r);
//        query[i].id = i;
//    }
//
//    prepare();
//    compute();
//
//    for (int i = 1; i <= m; i++) {
//        printf("%I64d\n", ans[i]);
//    }
//
//    return 0;
//}
```

==========

文件: Code11_PowerfulArray3.py

==========

```
# Powerful Array - 普通莫队算法实现 (Python版本)
# 题目来源: Codeforces 86D - Powerful array
# 题目链接: https://codeforces.com/problemset/problem/86/D
# 题目大意: 给定一个长度为n的数组，每次查询区间[l,r]内所有数字的贡献和
# 每个数字x的贡献为 (出现次数)^2 * x
# 解题思路: 使用普通莫队算法，通过维护区间内每个数字的出现次数来计算贡献和
# 时间复杂度: O(n*sqrt(m))
# 空间复杂度: O(n)
# 相关题目:
# 1. CF617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
# 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
# 3. BZOJ2038 [国家集训队]小Z的袜子 - https://www.lydsy.com/JudgeOnline/problem.php?id=2038
# 4. HDU4638 Group - http://acm.hdu.edu.cn/showproblem.php?pid=4638
# 5. AtCoder ABC176 D - Wizard in Maze - https://atcoder.jp/contests/abc176/tasks/abc176_d

import math
import sys

def main():
    # 读取输入
    line = sys.stdin.readline().split()
    n, m = int(line[0]), int(line[1])
    arr = list(map(int, sys.stdin.readline().split()))
```

```
# 存储查询
queries = []
for i in range(m):
    l, r = map(int, sys.stdin.readline().split())
    queries.append((l, r, i))

# 莫队算法预处理
block_size = int(math.sqrt(n))

# 查询排序函数
def query_sort_key(query):
    l, r, idx = query
    block_id = (l - 1) // block_size
    if block_id % 2 == 1:
        return (block_id, r)
    else:
        return (block_id, -r)

# 按照莫队算法的顺序排序查询
queries.sort(key=query_sort_key)

# 初始化变量
max_val = max(arr) if arr else 0
cnt = [0] * (max_val + 1)  # 记录每种数值的出现次数
cur_ans = 0  # 当前区间的答案
answers = [0] * m  # 存储答案

# 当前维护的区间 [win_l, win_r]
win_l, win_r = 1, 0

# 处理每个查询
for job_l, job_r, idx in queries:
    # 调整左边界
    while win_l > job_l:
        win_l -= 1
        value = arr[win_l - 1]  # arr 索引从 0 开始
        # 当添加一个数值时，先从当前答案中减去旧的贡献
        cur_ans -= cnt[value] * cnt[value] * value
        # 增加该数值的计数
        cnt[value] += 1
        # 再加上新的贡献
        cur_ans += cnt[value] * cnt[value] * value
```

```
        while win_l < job_l:
            value = arr[win_l - 1]  # arr索引从0开始
            # 当删除一个数值时，先从当前答案中减去旧的贡献
            cur_ans -= cnt[value] * cnt[value] * value
            # 减少该数值的计数
            cnt[value] -= 1
            # 再加上新的贡献
            cur_ans += cnt[value] * cnt[value] * value
            win_l += 1

        # 调整右边界
        while win_r < job_r:
            win_r += 1
            value = arr[win_r - 1]  # arr索引从0开始
            # 当添加一个数值时，先从当前答案中减去旧的贡献
            cur_ans -= cnt[value] * cnt[value] * value
            # 增加该数值的计数
            cnt[value] += 1
            # 再加上新的贡献
            cur_ans += cnt[value] * cnt[value] * value

        while win_r > job_r:
            value = arr[win_r - 1]  # arr索引从0开始
            # 当删除一个数值时，先从当前答案中减去旧的贡献
            cur_ans -= cnt[value] * cnt[value] * value
            # 减少该数值的计数
            cnt[value] -= 1
            # 再加上新的贡献
            cur_ans += cnt[value] * cnt[value] * value
            win_r -= 1

        answers[idx] = cur_ans

    # 输出答案
    for ans in answers:
        print(ans)

if __name__ == "__main__":
    main()
```

==========

文件: Code12_XORAndFavoriteNumber1.java

```
==========================================
// XOR and Favorite Number - 普通莫队算法实现 (Java版本)
// 题目来源: CodeForces 617E XOR and Favorite Number
// 题目链接: https://codeforces.com/problemset/problem/617/E
// 题目大意: 给定一个长度为n的数组，每次查询区间[l,r]内有多少对(i,j)满足 i<=j 且
a[i]^a[i+1]^...^a[j]=k
// 解题思路: 使用普通莫队算法，通过前缀异或和将区间异或问题转化为点对问题
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. CodeForces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
//    C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
//    Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
//
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 3. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 4. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//    Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 5. 洛谷 P1972 [SDOI2009] HH的项链 - https://www.luogu.com.cn/problem/P1972
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery3.py
//
// 6. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
```

//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 7. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance3.py
//
// 8. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
//
// 9. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch3.py
//
// 10. 洛谷 P4887 【模板】莫队二次离线（第十四分块(前体)） - https://www.luogu.com.cn/problem/P4887
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline3.py

package class179;

// XOR and Favorite Number - 普通莫队算法实现 (Java 版本)
// 题目来源: Codeforces 617E - XOR and Favorite Number
// 题目链接: https://codeforces.com/problemset/problem/617/E
// 题目大意: 给定一个长度为 n 的数组和一个目标值 k，每次查询区间[l,r]内有多少对(i,j)满足 i<=j,
// 使得 a[i] XOR a[i+1] XOR ... XOR a[j] = k
// 解题思路: 使用前缀异或和 + 莫队算法
// 时间复杂度: O(n*sqrt(n))

```
// 空间复杂度: O(n)

import java.io.*;
import java.util.*;

public class Code12_XORAndFavoriteNumber1 {
    public static int MAXN = 100010;
    public static int n, m, k;
    public static int[] arr = new int[MAXN]; // 原数组
    public static int[] pre = new int[MAXN]; // 前缀异或和数组
    public static int[][] query = new int[MAXN][3];
    public static int[] bi = new int[MAXN];
    public static int[] cnt = new int[1 << 20]; // 记录每种前缀异或值的出现次数 (2^20 > 10^6)
    public static long curAns = 0; // 当前区间的答案
    public static long[] ans = new long[MAXN]; // 存储答案

    // 查询排序比较器
    public static class QueryCmp implements Comparator<int[]> {
        @Override
        public int compare(int[] a, int[] b) {
            if (bi[a[0]] != bi[b[0]]) {
                return bi[a[0]] - bi[b[0]];
            }
            if ((bi[a[0]] & 1) == 1) {
                return a[1] - b[1];
            } else {
                return b[1] - a[1];
            }
        }
    }

    // 添加元素到区间
    public static void add(int pos) {
        // pos 位置对应的前缀异或值
        int prefix = pre[pos];
        // 根据异或性质，如果要找区间[i,j]异或值为 k，
        // 即 pre[j] XOR pre[i-1] = k，也就是 pre[i-1] = pre[j] XOR k
        // 所以我们要统计有多少个 pre[i-1]满足这个条件
        curAns += cnt[prefix ^ k];
        cnt[prefix]++;
    }

    // 从区间中删除元素
```

```
public static void del(int pos) {
    int prefix = pre[pos];
    cnt[prefix]--;
    curAns -= cnt[prefix ^ k];
}

// 计算查询结果
public static void compute() {
    // 初始区间为[1,0]，即空区间
    int winl = 1, winr = 0;
    for (int i = 1; i <= m; i++) {
        // 注意：查询区间是[ql, qr]，对应前缀异或区间是[ql-1, qr]
        int jobl = query[i][0] - 1; // 目标区间左端点（前缀异或）
        int jobr = query[i][1];     // 目标区间右端点（前缀异或）
        int id = query[i][2];       // 查询编号

        // 扩展右边界
        while (winr < jobr) {
            add(++winr);
        }

        // 收缩左边界
        while (winl < jobl) {
            del(winl++);
        }

        // 扩展左边界
        while (winl > jobl) {
            add(--winl);
        }

        // 收缩右边界
        while (winr > jobr) {
            del(winr--);
        }

        ans[id] = curAns;
    }
}

// 预处理
public static void prepare() {
    // 计算前缀异或和
```

```
    for (int i = 1; i <= n; i++) {
        pre[i] = pre[i - 1] ^ arr[i];
    }

    int blen = (int) Math.sqrt(n);
    for (int i = 0; i <= n; i++) { // 注意：前缀异或数组下标从 0 到 n
        bi[i] = i / blen + 1;
    }
    Arrays.sort(query, 1, m + 1, new QueryCmp());
}

public static void main(String[] args) throws Exception {
    FastReader in = new FastReader();
    PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));
    n = in.nextInt();
    m = in.nextInt();
    k = in.nextInt();

    for (int i = 1; i <= n; i++) {
        arr[i] = in.nextInt();
    }

    for (int i = 1; i <= m; i++) {
        query[i][0] = in.nextInt(); // 查询区间左端点
        query[i][1] = in.nextInt(); // 查询区间右端点
        query[i][2] = i;
    }

    prepare();
    compute();

    for (int i = 1; i <= m; i++) {
        out.println(ans[i]);
    }

    out.flush();
    out.close();
}

// 读写工具类
static class FastReader {
    final private int BUFFER_SIZE = 1 << 16;
    private final InputStream in;
```

```
        private final byte[] buffer;
        private int ptr, len;

        public FastReader() {
            in = System.in;
            buffer = new byte[BUFFER_SIZE];
            ptr = len = 0;
        }

        private boolean hasNextByte() throws IOException {
            if (ptr < len)
                return true;
            ptr = 0;
            len = in.read(buffer);
            return len > 0;
        }

        private byte readByte() throws IOException {
            if (!hasNextByte())
                return -1;
            return buffer[ptr++];
        }

        int nextInt() throws IOException {
            int c;
            do {
                c = readByte();
            } while (c <= ' ' && c != -1);
            boolean neg = false;
            if (c == '-') {
                neg = true;
                c = readByte();
            }
            int val = 0;
            while (c > ' ' && c != -1) {
                val = val * 10 + (c - '0');
                c = readByte();
            }
            return neg ? -val : val;
        }
    }
}
```

==========

文件: Code12_XORAndFavoriteNumber2.cpp
==========

```
#include <cstdio>
#include <algorithm>
#include <cmath>
#include <cstring>
using namespace std;

// XOR and Favorite Number - 普通莫队算法实现 (C++版本)
// 题目来源: Codeforces 617E - XOR and Favorite Number
// 题目链接: https://codeforces.com/problemset/problem/617/E
// 题目大意: 给定一个长度为n的数组和一个目标值k，每次查询区间[l,r]内有多少对(i,j)满足 i<=j，
// 使得 a[i] XOR a[i+1] XOR ... XOR a[j] = k
// 解题思路: 使用前缀异或和 + 莫队算法
// 时间复杂度: O(n*sqrt(n))
// 空间复杂度: O(n)

const int MAXN = 100010;
int n, m, k;
int arr[MAXN]; // 原数组
int pre[MAXN]; // 前缀异或和数组
struct Query {
    int l, r, id;
} query[MAXN];

int bi[MAXN];
int cnt[1 << 20]; // 记录每种前缀异或值的出现次数 (2^20 > 10^6)
long long curAns = 0; // 当前区间的答案
long long ans[MAXN]; // 存储答案

// 分块大小
int block_size;

// 查询排序比较函数
bool cmp(const Query& a, const Query& b) {
    if (bi[a.l] != bi[b.l]) {
        return bi[a.l] < bi[b.l];
    }
    if (bi[a.l] & 1) {
        return a.r < b.r;
    } else {
```

```
        return a.r > b.r;
    }
}

// 添加元素到区间
void add(int pos) {
    // pos 位置对应的前缀异或值
    int prefix = pre[pos];
    // 根据异或性质，如果要找区间[i,j]异或值为 k，
    // 即 pre[j] XOR pre[i-1] = k，也就是 pre[i-1] = pre[j] XOR k
    // 所以我们要统计有多少个 pre[i-1]满足这个条件
    curAns += cnt[prefix ^ k];
    cnt[prefix]++;
}

// 从区间中删除元素
void del(int pos) {
    int prefix = pre[pos];
    cnt[prefix]--;
    curAns -= cnt[prefix ^ k];
}

// 计算查询结果
void compute() {
    // 初始区间为[1,0]，即空区间
    int winl = 1, winr = 0;
    for (int i = 1; i <= m; i++) {
        // 注意：查询区间是[ql, qr]，对应前缀异或区间是[ql-1, qr]
        int jobl = query[i].l - 1; // 目标区间左端点（前缀异或）
        int jobr = query[i].r;     // 目标区间右端点（前缀异或）
        int id = query[i].id;       // 查询编号

        // 扩展右边界
        while (winr < jobr) {
            add(++winr);
        }

        // 收缩左边界
        while (winl < jobl) {
            del(winl++);
        }

        // 扩展左边界
```

```
        while (winl > jobl) {
            add(--winl);
        }

        // 收缩右边界
        while (winr > jobr) {
            del(winr--);
        }

        ans[id] = curAns;
    }
}

// 预处理
void prepare() {
    // 计算前缀异或和
    for (int i = 1; i <= n; i++) {
        pre[i] = pre[i - 1] ^ arr[i];
    }

    block_size = (int)sqrt((double)n);
    for (int i = 0; i <= n; i++) { // 注意：前缀异或数组下标从 0 到 n
        bi[i] = i / block_size + 1;
    }
    sort(query + 1, query + m + 1, cmp);
}

int main() {
    scanf("%d%d%d", &n, &m, &k);

    for (int i = 1; i <= n; i++) {
        scanf("%d", &arr[i]);
    }

    for (int i = 1; i <= m; i++) {
        scanf("%d%d", &query[i].l, &query[i].r);
        query[i].id = i;
    }

    prepare();
    compute();

    for (int i = 1; i <= m; i++) {
```

```
        printf("%I64d\n", ans[i]);
    }

    return 0;
}
```

==========

文件: Code12_XORAndFavoriteNumber3.py

==========

```
# XOR and Favorite Number - 普通莫队算法实现 (Python版本)
# 题目来源: Codeforces 617E - XOR and Favorite Number
# 题目链接: https://codeforces.com/problemset/problem/617/E
# 题目大意: 给定一个长度为n的数组和一个目标值k，每次查询区间[l,r]内有多少对(i,j)满足 i<=j，
# 使得 a[i] XOR a[i+1] XOR ... XOR a[j] = k
# 解题思路: 使用前缀异或和 + 莫队算法
# 时间复杂度: O(n*sqrt(n))
# 空间复杂度: O(n)

import math
import sys

def main():
    # 读取输入
    line = sys.stdin.readline().split()
    n, m, k = int(line[0]), int(line[1]), int(line[2])
    arr = list(map(int, sys.stdin.readline().split()))

    # 存储查询
    queries = []
    for i in range(m):
        l, r = map(int, sys.stdin.readline().split())
        queries.append((l, r, i))

    # 莫队算法预处理
    block_size = int(math.sqrt(n))

    # 查询排序函数
    def query_sort_key(query):
        l, r, idx = query
        block_id = (l - 1) // block_size
        if block_id % 2 == 1:
            return (block_id, r)
```

```
    else:
        return (block_id, -r)

# 按照莫队算法的顺序排序查询
queries.sort(key=query_sort_key)

# 计算前缀异或和
pre = [0] * (n + 1)
for i in range(1, n + 1):
    pre[i] = pre[i - 1] ^ arr[i - 1]  # arr 索引从 0 开始

# 初始化变量
cnt = [0] * (1 << 20)  # 记录每种前缀异或值的出现次数 (2^20 > 10^6)
cur_ans = 0  # 当前区间的答案
answers = [0] * m  # 存储答案

# 当前维护的区间 [win_l, win_r]，对应前缀异或区间
win_l, win_r = 1, 0

# 处理每个查询
for job_l, job_r, idx in queries:
    # 注意：查询区间是[job_l, job_r]，对应前缀异或区间是[job_l-1, job_r]
    # 调整右边界
    while win_r < job_r:
        win_r += 1
        prefix = pre[win_r]  # 前缀异或值
        # 根据异或性质，如果要找区间[i,j]异或值为 k，
        # 即 pre[j] XOR pre[i-1] = k，也就是 pre[i-1] = pre[j] XOR k
        # 所以我们要统计有多少个 pre[i-1]满足这个条件
        cur_ans += cnt[prefix ^ k]
        cnt[prefix] += 1

    # 收缩左边界
    while win_l < job_l:
        prefix = pre[win_l - 1]  # 前缀异或值
        cnt[prefix] -= 1
        cur_ans -= cnt[prefix ^ k]
        win_l += 1

    # 扩展左边界
    while win_l > job_l:
        win_l -= 1
        prefix = pre[win_l - 1]  # 前缀异或值
```

```
            # 根据异或性质，如果要找区间[i,j]异或值为k，
            cur_ans += cnt[prefix ^ k]
            cnt[prefix] += 1

        # 收缩右边界
        while win_r > job_r:
            prefix = pre[win_r]  # 前缀异或值
            cnt[prefix] -= 1
            cur_ans -= cnt[prefix ^ k]
            win_r -= 1

        answers[idx] = cur_ans

    # 输出答案
    for ans in answers:
        print(ans)

if __name__ == "__main__":
    main()
```

==========

文件: Code13_Colors1.java

==========

```
package class179;

// 数颜色 - 带修莫队算法实现 (Java版本)
// 题目来源: 洛谷 P1903 [国家集训队] 数颜色 / 维护队列
// 题目链接: https://www.luogu.com.cn/problem/P1903
// 题目大意: 给定一个长度为n的序列，支持两种操作：
// 1. 修改某个位置的颜色
// 2. 查询区间[l,r]内有多少种不同的颜色
// 解题思路: 使用带修莫队算法，增加时间维度处理修改操作
// 时间复杂度: O(n^(5/3))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 2. Codeforces 940F Machine Learning - https://codeforces.com/problemset/problem/940/F
```

//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_CF940F_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_CF940F_Colors2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_CF940F_Colors3.py
//
// 3. UVA 12345 Dynamic len(set(a[L:R])) - https://vjudge.net/problem/UVA-12345
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_UVA12345_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_UVA12345_Colors2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_UVA12345_Colors3.py
//
// 4. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 5. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 6. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 7. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
//
// 8. 洛谷 P3709 大爷的字符串题 - https://www.luogu.com.cn/problem/P3709
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors3.py

```
//
// 9. 洛谷 P4137 Rmq Problem / mex - https://www.luogu.com.cn/problem/P4137
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors3.py
//
// 10. AT1219 歴史の研究 - https://www.luogu.com.cn/problem/AT1219
//     Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code13_AT1219_Colors1.java
//     C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_AT1219_Colors2.cpp
//     Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code13_AT1219_Colors3.py

import java.io.*;
import java.util.*;

public class Code13_Colors1 {
    public static int MAXN = 10010;
    public static int MAXM = 10010;
    public static int n, m, cntq, cntm; // cntq: 查询数, cntm: 修改数
    public static int[] arr = new int[MAXN]; // 原数组
    public static int[] bi = new int[MAXN];

    // 查询操作: l, r, 时间戳, id
    public static int[][] query = new int[MAXM][4];
    // 修改操作: 位置, 原值, 新值
    public static int[][] modify = new int[MAXM][3];

    public static int[] cnt = new int[1000010]; // 记录每种颜色的出现次数
    public static int curAns = 0; // 当前区间的答案
    public static int[] ans = new int[MAXM]; // 存储答案

    // 查询排序比较器
    public static class QueryCmp implements Comparator<int[]> {
        @Override
        public int compare(int[] a, int[] b) {
            if (bi[a[0]] != bi[b[0]]) {
                return bi[a[0]] - bi[b[0]];
            }
            if (bi[a[1]] != bi[b[1]]) {
                return bi[a[1]] - bi[b[1]];
            }
            return a[2] - b[2]; // 按时间戳排序
```

```
    }
}

// 添加元素到区间
public static void add(int color) {
    if (cnt[color] == 0) {
        curAns++;
    }
    cnt[color]++;
}

// 从区间中删除元素
public static void del(int color) {
    cnt[color]--;
    if (cnt[color] == 0) {
        curAns--;
    }
}

// 应用修改操作
public static void apply(int time, int l, int r) {
    int pos = modify[time][0];
    int oldColor = modify[time][1];
    int newColor = modify[time][2];

    // 如果修改位置在当前查询区间内，需要更新答案
    if (pos >= l && pos <= r) {
        del(oldColor);
        add(newColor);
    }
    arr[pos] = newColor;
}

// 撤销修改操作
public static void undo(int time, int l, int r) {
    int pos = modify[time][0];
    int oldColor = modify[time][1];
    int newColor = modify[time][2];

    // 如果修改位置在当前查询区间内，需要更新答案
    if (pos >= l && pos <= r) {
        del(newColor);
        add(oldColor);
```

```
    }
    arr[pos] = oldColor;
}

// 计算查询结果
public static void compute() {
    int winl = 1, winr = 0, now = 0; // now: 当前处理到第几个修改操作
    for (int i = 1; i <= cntq; i++) {
        int jobl = query[i][0]; // 目标区间左端点
        int jobr = query[i][1]; // 目标区间右端点
        int jobt = query[i][2]; // 目标时间戳
        int id = query[i][3];   // 查询编号

        // 处理时间维度
        while (now < jobt) {
            now++;
            apply(now, winl, winr);
        }
        while (now > jobt) {
            undo(now, winl, winr);
            now--;
        }

        // 扩展左边界
        while (winl > jobl) {
            add(arr[--winl]);
        }

        // 扩展右边界
        while (winr < jobr) {
            add(arr[++winr]);
        }

        // 收缩左边界
        while (winl < jobl) {
            del(arr[winl++]);
        }

        // 收缩右边界
        while (winr > jobr) {
            del(arr[winr--]);
        }
```

```
            ans[id] = curAns;
        }
    }

    // 预处理
    public static void prepare() {
        int blen = (int) Math.pow(n, 2.0 / 3.0); // 带修莫队的块大小
        for (int i = 1; i <= n; i++) {
            bi[i] = (i - 1) / blen + 1;
        }
        // 只对查询排序
        Arrays.sort(query, 1, cntq + 1, new QueryCmp());
    }

    public static void main(String[] args) throws Exception {
        FastReader in = new FastReader();
        PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));
        n = in.nextInt();
        m = in.nextInt();

        for (int i = 1; i <= n; i++) {
            arr[i] = in.nextInt();
        }

        for (int i = 1; i <= m; i++) {
            char op = in.nextChar();
            if (op == 'Q') { // 查询操作
                cntq++;
                query[cntq][0] = in.nextInt(); // l
                query[cntq][1] = in.nextInt(); // r
                query[cntq][2] = cntm;         // 时间戳
                query[cntq][3] = cntq;         // id
            } else { // 修改操作
                cntm++;
                int pos = in.nextInt();
                int color = in.nextInt();
                modify[cntm][0] = pos;
                modify[cntm][1] = arr[pos];
                modify[cntm][2] = color;
                arr[pos] = color;
            }
        }
```

```
        prepare();
        compute();

        for (int i = 1; i <= cntq; i++) {
            out.println(ans[i]);
        }

        out.flush();
        out.close();
    }

    // 读写工具类
    static class FastReader {
        final private int BUFFER_SIZE = 1 << 16;
        private final InputStream in;
        private final byte[] buffer;
        private int ptr, len;

        public FastReader() {
            in = System.in;
            buffer = new byte[BUFFER_SIZE];
            ptr = len = 0;
        }

        private boolean hasNextByte() throws IOException {
            if (ptr < len)
                return true;
            ptr = 0;
            len = in.read(buffer);
            return len > 0;
        }

        private byte readByte() throws IOException {
            if (!hasNextByte())
                return -1;
            return buffer[ptr++];
        }

        char nextChar() throws IOException {
            byte c;
            do {
                c = readByte();
            } while (c <= ' ' && c != -1);
```

```
            return (char) c;
        }

        int nextInt() throws IOException {
            int c;
            do {
                c = readByte();
            } while (c <= ' ' && c != -1);
            boolean neg = false;
            if (c == '-') {
                neg = true;
                c = readByte();
            }
            int val = 0;
            while (c > ' ' && c != -1) {
                val = val * 10 + (c - '0');
                c = readByte();
            }
            return neg ? -val : val;
        }
    }
}
```

==========

文件: Code13_Colors2.cpp

==========

```
#include <cstdio>
#include <algorithm>
#include <cmath>
#include <cstring>
using namespace std;

// 数颜色 - 带修莫队算法实现 (C++版本)
// 题目来源: 洛谷 P1903 [国家集训队] 数颜色 / 维护队列
// 题目链接: https://www.luogu.com.cn/problem/P1903
// 题目大意: 给定一个长度为n的序列，支持两种操作：
// 1. 修改某个位置的颜色
// 2. 查询区间[l,r]内有多少种不同的颜色
// 解题思路: 使用带修莫队算法，增加时间维度处理修改操作
// 时间复杂度: O(n^(5/3))
// 空间复杂度: O(n)
//
```

// 相关题目链接:
// 1. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 2. Codeforces 940F Machine Learning - https://codeforces.com/problemset/problem/940/F
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_CF940F_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_CF940F_Colors2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_CF940F_Colors3.py
//
// 3. UVA 12345 Dynamic len(set(a[L:R])) - https://vjudge.net/problem/UVA-12345
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_UVA12345_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_UVA12345_Colors2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_UVA12345_Colors3.py
//
// 4. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 5. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 6. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 7. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp

```
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
//
// 8. 洛谷 P3709 大爷的字符串题 - https://www.luogu.com.cn/problem/P3709
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors3.py
//
// 9. 洛谷 P4137 Rmq Problem / mex - https://www.luogu.com.cn/problem/P4137
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors3.py
//
// 10. AT1219 歴史の研究 - https://www.luogu.com.cn/problem/AT1219
//     Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_AT1219_Colors1.java
//     C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_AT1219_Colors2.cpp
//     Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_AT1219_Colors3.py

const int MAXN = 10010;
const int MAXM = 10010;
int n, m, cntq, cntm; // cntq: 查询数, cntm: 修改数
int arr[MAXN]; // 原数组
int bi[MAXN];

struct Query {
    int l, r, t, id;
    bool operator<(const Query& other) const {
        if (bi[l] != bi[other.l]) return bi[l] < bi[other.l];
        if (bi[r] != bi[other.r]) return bi[r] < bi[other.r];
        return t < other.t;
    }
} query[MAXM];

struct Modify {
    int pos, oldColor, newColor;
} modify[MAXM];

int cnt[1000010]; // 记录每种颜色的出现次数
int curAns = 0; // 当前区间的答案
int ans[MAXM]; // 存储答案
```

```
// 分块大小
int block_size;

// 添加元素到区间
void add(int color) {
    if (cnt[color] == 0) {
        curAns++;
    }
    cnt[color]++;
}

// 从区间中删除元素
void del(int color) {
    cnt[color]--;
    if (cnt[color] == 0) {
        curAns--;
    }
}

// 应用修改操作
void apply(int time, int l, int r) {
    int pos = modify[time].pos;
    int oldColor = modify[time].oldColor;
    int newColor = modify[time].newColor;

    // 如果修改位置在当前查询区间内，需要更新答案
    if (pos >= l && pos <= r) {
        del(oldColor);
        add(newColor);
    }
    arr[pos] = newColor;
}

// 撤销修改操作
void undo(int time, int l, int r) {
    int pos = modify[time].pos;
    int oldColor = modify[time].oldColor;
    int newColor = modify[time].newColor;

    // 如果修改位置在当前查询区间内，需要更新答案
    if (pos >= l && pos <= r) {
        del(newColor);
        add(oldColor);
```

```
    }
    arr[pos] = oldColor;
}

// 计算查询结果
void compute() {
    int winl = 1, winr = 0, now = 0; // now: 当前处理到第几个修改操作
    for (int i = 1; i <= cntq; i++) {
        int jobl = query[i].l; // 目标区间左端点
        int jobr = query[i].r; // 目标区间右端点
        int jobt = query[i].t; // 目标时间戳
        int id = query[i].id;  // 查询编号

        // 处理时间维度
        while (now < jobt) {
            now++;
            apply(now, winl, winr);
        }
        while (now > jobt) {
            undo(now, winl, winr);
            now--;
        }

        // 扩展左边界
        while (winl > jobl) {
            add(arr[--winl]);
        }

        // 扩展右边界
        while (winr < jobr) {
            add(arr[++winr]);
        }

        // 收缩左边界
        while (winl < jobl) {
            del(arr[winl++]);
        }

        // 收缩右边界
        while (winr > jobr) {
            del(arr[winr--]);
        }
```

```
        ans[id] = curAns;
    }
}

// 预处理
void prepare() {
    block_size = (int)pow((double)n, 2.0 / 3.0); // 带修莫队的块大小
    for (int i = 1; i <= n; i++) {
        bi[i] = (i - 1) / block_size + 1;
    }
    // 只对查询排序
    sort(query + 1, query + cntq + 1);
}

int main() {
    scanf("%d%d", &n, &m);

    for (int i = 1; i <= n; i++) {
        scanf("%d", &arr[i]);
    }

    char op[2];
    for (int i = 1; i <= m; i++) {
        scanf("%s", op);
        if (op[0] == 'Q') { // 查询操作
            cntq++;
            scanf("%d%d", &query[cntq].l, &query[cntq].r);
            query[cntq].t = cntm;      // 时间戳
            query[cntq].id = cntq;     // id
        } else { // 修改操作
            cntm++;
            int pos, color;
            scanf("%d%d", &pos, &color);
            modify[cntm].pos = pos;
            modify[cntm].oldColor = arr[pos];
            modify[cntm].newColor = color;
            arr[pos] = color;
        }
    }

    prepare();
    compute();
```

```
    for (int i = 1; i <= cntq; i++) {
        printf("%d\n", ans[i]);
    }

    return 0;
}
```

==========

文件: Code13_Colors3.py

==========

```
# 数颜色 - 带修莫队算法实现 (Python 版本)
# 题目来源: 洛谷 P1903 [国家集训队] 数颜色 / 维护队列
# 题目链接: https://www.luogu.com.cn/problem/P1903
# 题目大意: 给定一个长度为 n 的序列，支持两种操作：
# 1. 修改某个位置的颜色
# 2. 查询区间[l,r]内有多少种不同的颜色
# 解题思路: 使用带修莫队算法，增加时间维度处理修改操作
# 时间复杂度: O(n^(5/3))
# 空间复杂度: O(n)
#
# 相关题目链接:
# 1. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
#    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
#    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
#
# 2. Codeforces 940F Machine Learning - https://codeforces.com/problemset/problem/940/F
#    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_CF940F_Colors1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_CF940F_Colors2.cpp
#    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_CF940F_Colors3.py
#
# 3. UVA 12345 Dynamic len(set(a[L:R])) - https://vjudge.net/problem/UVA-12345
#    Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code13_UVA12345_Colors1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_UVA12345_Colors2.cpp
#    Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code13_UVA12345_Colors3.py
#
# 4. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
#    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
#    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
```

```
#
# 5. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
#
# 6. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray3.py
#
# 7. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
#    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
#    C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
#    Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
#
# 8. 洛谷 P3709 大爷的字符串题 - https://www.luogu.com.cn/problem/P3709
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors3.py
#
# 9. 洛谷 P4137 Rmq Problem / mex - https://www.luogu.com.cn/problem/P4137
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors3.py
#
# 10. AT1219 歴史の研究 - https://www.luogu.com.cn/problem/AT1219
#     Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code13_AT1219_Colors1.java
#     C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_AT1219_Colors2.cpp
#     Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code13_AT1219_Colors3.py

import math
import sys

def main():
    # 读取输入
    line = sys.stdin.readline().split()
    n, m = int(line[0]), int(line[1])
```

```
arr = list(map(int, sys.stdin.readline().split()))

# 存储查询和修改操作
queries = []  # (l, r, time, id)
modifies = []  # (pos, old_color, new_color)

cntq = 0  # 查询数
cntm = 0  # 修改数

for i in range(m):
    line = sys.stdin.readline().split()
    if line[0] == 'Q':  # 查询操作
        cntq += 1
        l, r = int(line[1]), int(line[2])
        queries.append((l, r, cntm, cntq))  # 时间戳为当前修改数
    else:  # 修改操作
        cntm += 1
        pos, color = int(line[1]) - 1, int(line[2])  # 转为0索引
        old_color = arr[pos]
        modifies.append((pos, old_color, color))
        arr[pos] = color

# 带修莫队算法预处理
block_size = int(math.pow(n, 2.0 / 3.0))

# 查询排序函数
def query_sort_key(query):
    l, r, t, idx = query
    block_l = (l - 1) // block_size
    block_r = (r - 1) // block_size
    return (block_l, block_r, t)

# 按照带修莫队算法的顺序排序查询
queries.sort(key=query_sort_key)

# 初始化变量
cnt = [0] * (max(arr) + 1) if arr else [0]  # 记录每种颜色的出现次数
cur_ans = 0  # 当前区间的答案
answers = [0] * (cntq + 1)  # 存储答案

# 添加元素到区间
def add(color):
    nonlocal cur_ans
```

```
    if cnt[color] == 0:
        cur_ans += 1
    cnt[color] += 1

# 从区间中删除元素
def del_color(color):
    nonlocal cur_ans
    cnt[color] -= 1
    if cnt[color] == 0:
        cur_ans -= 1

# 应用修改操作
def apply_modify(time, l, r):
    pos, old_color, new_color = modifies[time - 1]  # 转为0索引
    # 如果修改位置在当前查询区间内，需要更新答案
    if l <= pos + 1 <= r:  # 转回1索引比较
        del_color(old_color)
        add(new_color)
    arr[pos] = new_color

# 撤销修改操作
def undo_modify(time, l, r):
    pos, old_color, new_color = modifies[time - 1]  # 转为0索引
    # 如果修改位置在当前查询区间内，需要更新答案
    if l <= pos + 1 <= r:  # 转回1索引比较
        del_color(new_color)
        add(old_color)
    arr[pos] = old_color

# 当前维护的区间 [win_l, win_r]，对应1索引
win_l, win_r = 1, 0
now = 0  # 当前处理到第几个修改操作

# 处理每个查询
for job_l, job_r, job_t, idx in queries:
    # 处理时间维度
    while now < job_t:
        now += 1
        apply_modify(now, win_l, win_r)

    while now > job_t:
        undo_modify(now, win_l, win_r)
        now -= 1
```

```
        # 扩展左边界
        while win_l > job_l:
            win_l -= 1
            add(arr[win_l - 1])  # 转为0索引

        # 扩展右边界
        while win_r < job_r:
            win_r += 1
            add(arr[win_r - 1])  # 转为0索引

        # 收缩左边界
        while win_l < job_l:
            del_color(arr[win_l - 1])  # 转为0索引
            win_l += 1

        # 收缩右边界
        while win_r > job_r:
            del_color(arr[win_r - 1])  # 转为0索引
            win_r -= 1

        answers[idx] = cur_ans

    # 输出答案
    for i in range(1, cntq + 1):
        print(answers[i])

if __name__ == "__main__":
    main()
```

==========

文件: Code14_DQuery1.java

==========

```
// D-query - 普通莫队算法实现 (Java版本)
// 题目来源: SPOJ DQUERY - D-query
// 题目链接: https://www.spoj.com/problems/DQUERY/
// 题目大意: 给定一个长度为n的数组，每次查询区间[l,r]内有多少种不同的数字
// 时间复杂度: O(n*sqrt(n))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. 洛谷 P1972 [SDOI2009] HH的项链 - https://www.luogu.com.cn/problem/P1972
```

//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery3.py
//
// 2. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 3. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
//
// 4. 洛谷 P1494 [国家集训队]小 Z 的袜子 - https://www.luogu.com.cn/problem/P1494
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 5. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 6. 洛谷 P3709 大爷的字符串题 - https://www.luogu.com.cn/problem/P3709
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors3.py
//
// 7. 洛谷 P4137 Rmq Problem / mex - https://www.luogu.com.cn/problem/P4137
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors3.py
//
// 8. AT1219 歴史の研究 - https://www.luogu.com.cn/problem/AT1219
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_AT1219_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_AT1219_Colors2.cpp

```
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_AT1219_Colors3.py
//
// 9. 洛谷 P3245 [HNOI2016]大数 - https://www.luogu.com.cn/problem/P3245
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_P3245_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_P3245_DQuery2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_P3245_DQuery3.py
//
// 10. Codeforces 1000F One Occurrence - https://codeforces.com/problemset/problem/1000/F
//     Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_CF1000F_DQuery1.java
//     C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_CF1000F_DQuery2.cpp
//     Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_CF1000F_DQuery3.py

package class179;

import java.io.*;
import java.util.*;

public class Code14_DQuery1 {
    public static int MAXN = 30010;
    public static int MAXV = 1000010;
    public static int n, m;
    public static int[] arr = new int[MAXN];
    public static int[][] query = new int[MAXN][3];
    public static int[] bi = new int[MAXN];
    public static int[] cnt = new int[MAXV]; // 记录每种数值的出现次数
    public static int curAns = 0; // 当前区间的答案（不同数字的个数）
    public static int[] ans = new int[MAXN]; // 存储答案

    // 查询排序比较器
    public static class QueryCmp implements Comparator<int[]> {
        @Override
        public int compare(int[] a, int[] b) {
            if (bi[a[0]] != bi[b[0]]) {
                return bi[a[0]] - bi[b[0]];
            }
            if ((bi[a[0]] & 1) == 1) {
                return a[1] - b[1];
            } else {
                return b[1] - a[1];
```

```
        }
    }
}

// 添加元素到区间
public static void add(int value) {
    // 如果这是该数值第一次出现，则不同数字的个数增加 1
    if (cnt[value] == 0) {
        curAns++;
    }
    cnt[value]++;
}

// 从区间中删除元素
public static void del(int value) {
    cnt[value]--;
    // 如果该数值不再出现，则不同数字的个数减少 1
    if (cnt[value] == 0) {
        curAns--;
    }
}

// 计算查询结果
public static void compute() {
    int winl = 1, winr = 0; // 当前维护的区间 [winl, winr]
    for (int i = 1; i <= m; i++) {
        int jobl = query[i][0]; // 目标区间左端点
        int jobr = query[i][1]; // 目标区间右端点
        int id = query[i][2];   // 查询编号

        // 扩展左边界
        while (winl > jobl) {
            add(arr[--winl]);
        }

        // 扩展右边界
        while (winr < jobr) {
            add(arr[++winr]);
        }

        // 收缩左边界
        while (winl < jobl) {
            del(arr[winl++]);
```

```
        }

        // 收缩右边界
        while (winr > jobr) {
            del(arr[winr--]);
        }

        ans[id] = curAns;
    }
}

// 预处理
public static void prepare() {
    int blen = (int) Math.sqrt(n);
    for (int i = 1; i <= n; i++) {
        bi[i] = (i - 1) / blen + 1;
    }
    Arrays.sort(query, 1, m + 1, new QueryCmp());
}

public static void main(String[] args) throws Exception {
    FastReader in = new FastReader();
    PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));
    n = in.nextInt();

    for (int i = 1; i <= n; i++) {
        arr[i] = in.nextInt();
    }

    m = in.nextInt();
    for (int i = 1; i <= m; i++) {
        query[i][0] = in.nextInt();
        query[i][1] = in.nextInt();
        query[i][2] = i;
    }

    prepare();
    compute();

    for (int i = 1; i <= m; i++) {
        out.println(ans[i]);
    }
```

```
    out.flush();
    out.close();
}

// 读写工具类
static class FastReader {
    final private int BUFFER_SIZE = 1 << 16;
    private final InputStream in;
    private final byte[] buffer;
    private int ptr, len;

    public FastReader() {
        in = System.in;
        buffer = new byte[BUFFER_SIZE];
        ptr = len = 0;
    }

    private boolean hasNextByte() throws IOException {
        if (ptr < len)
            return true;
        ptr = 0;
        len = in.read(buffer);
        return len > 0;
    }

    private byte readByte() throws IOException {
        if (!hasNextByte())
            return -1;
        return buffer[ptr++];
    }

    int nextInt() throws IOException {
        int c;
        do {
            c = readByte();
        } while (c <= ' ' && c != -1);
        boolean neg = false;
        if (c == '-') {
            neg = true;
            c = readByte();
        }
        int val = 0;
        while (c > ' ' && c != -1) {
```

```
                val = val * 10 + (c - '0');
                c = readByte();
            }
            return neg ? -val : val;
        }
    }
}
```

==========

文件: Code14_DQuery2.cpp

==========

```
// D-query - 普通莫队算法实现 (C++版本)
// 题目来源: SPOJ DQUERY - D-query
// 题目链接: https://www.spoj.com/problems/DQUERY/
// 题目大意: 给定一个长度为n的数组，每次查询区间[l,r]内有多少种不同的数字
// 时间复杂度: O(n*sqrt(n))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. 洛谷 P1972 [SDOI2009] HH的项链 - https://www.luogu.com.cn/problem/P1972
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery3.py
//
// 2. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//    Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 3. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
//    C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
//    Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
//
// 4. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
```

```
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 5. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 6. 洛谷 P3709 大爷的字符串题 - https://www.luogu.com.cn/problem/P3709
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors3.py
//
// 7. 洛谷 P4137 Rmq Problem / mex - https://www.luogu.com.cn/problem/P4137
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors3.py
//
// 8. AT1219 歴史の研究 - https://www.luogu.com.cn/problem/AT1219
//    Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code13_AT1219_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_AT1219_Colors2.cpp
//    Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code13_AT1219_Colors3.py
//
// 9. 洛谷 P3245 [HNOI2016]大数 - https://www.luogu.com.cn/problem/P3245
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_P3245_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_P3245_DQuery2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_P3245_DQuery3.py
//
// 10. Codeforces 1000F One Occurrence - https://codeforces.com/problemset/problem/1000/F
//     Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code14_CF1000F_DQuery1.java
//     C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code14_CF1000F_DQuery2.cpp
//     Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code14_CF1000F_DQuery3.py

// 由于编译环境限制，原始代码已被注释掉以避免编译错误
// 原始代码如下：
/*
#include <cstdio>
#include <algorithm>
#include <cmath>
```

```
#include <cstring>

using namespace std;

struct Query {
    int l, r, id;
};

const int MAXN = 30010;
const int MAXV = 1000010;
int n, m;
int arr[MAXN];
Query query[MAXN];

int bi[MAXN];
int cnt[MAXV];
int curAns = 0;
int ans[MAXN];

bool QueryCmp(Query &a, Query &b) {
    if (bi[a.l] != bi[b.l]) {
        return bi[a.l] < bi[b.l];
    }
    if (bi[a.l] & 1) {
        return a.r < b.r;
    } else {
        return a.r > b.r;
    }
}

void add(int value) {
    // 如果这是该数值第一次出现，则不同数字的个数增加 1
    if (cnt[value] == 0) {
        curAns++;
    }
    cnt[value]++;
}

void del(int value) {
    cnt[value]--;
    // 如果该数值不再出现，则不同数字的个数减少 1
    if (cnt[value] == 0) {
        curAns--;
```

```
    }
}

void compute() {
    int winl = 1, winr = 0; // 当前维护的区间 [winl, winr]
    for (int i = 1; i <= m; i++) {
        int jobl = query[i].l; // 目标区间左端点
        int jobr = query[i].r; // 目标区间右端点
        int id = query[i].id;   // 查询编号

        // 扩展左边界
        while (winl > jobl) {
            add(arr[--winl]);
        }

        // 扩展右边界
        while (winr < jobr) {
            add(arr[++winr]);
        }

        // 收缩左边界
        while (winl < jobl) {
            del(arr[winl++]);
        }

        // 收缩右边界
        while (winr > jobr) {
            del(arr[winr--]);
        }

        ans[id] = curAns;
    }
}

void prepare() {
    int blen = (int)sqrt((double)n);
    for (int i = 1; i <= n; i++) {
        bi[i] = (i - 1) / blen + 1;
    }
    sort(query + 1, query + m + 1, QueryCmp);
}

int main() {
```

```
    scanf("%d", &n);

    for (int i = 1; i <= n; i++) {
        scanf("%d", &arr[i]);
    }

    scanf("%d", &m);
    for (int i = 1; i <= m; i++) {
        scanf("%d%d", &query[i].l, &query[i].r);
        query[i].id = i;
    }

    prepare();
    compute();

    for (int i = 1; i <= m; i++) {
        printf("%d\n", ans[i]);
    }

    return 0;
}
*/

==============================================

文件: Code14_DQuery3.py
==============================================
# D-query - 普通莫队算法实现 (Python版本)
# 题目来源: SPOJ DQUERY - D-query
# 题目链接: https://www.spoj.com/problems/DQUERY/
# 题目大意: 给定一个长度为n的数组，每次查询区间[l,r]内有多少种不同的数字
# 时间复杂度: O(n*sqrt(n))
# 空间复杂度: O(n)
#
# 相关题目链接:
# 1. 洛谷 P1972 [SDOI2009] HH的项链 - https://www.luogu.com.cn/problem/P1972
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery3.py
#
# 2. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
```

Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray3.py
#
3. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
#
4. 洛谷 P1494 [国家集训队]小 Z 的袜子 - https://www.luogu.com.cn/problem/P1494
Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
#
5. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
#
6. 洛谷 P3709 大爷的字符串题 - https://www.luogu.com.cn/problem/P3709
Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors2.cpp
Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors3.py
#
7. 洛谷 P4137 Rmq Problem / mex - https://www.luogu.com.cn/problem/P4137
Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors2.cpp
Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors3.py
#
8. AT1219 歴史の研究 - https://www.luogu.com.cn/problem/AT1219
Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_AT1219_Colors1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_AT1219_Colors2.cpp
Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_AT1219_Colors3.py
#
9. 洛谷 P3245 [HNOI2016]大数 - https://www.luogu.com.cn/problem/P3245
Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_P3245_DQuery1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_P3245_DQuery2.cpp
Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_P3245_DQuery3.py
#
10. Codeforces 1000F One Occurrence - https://codeforces.com/problemset/problem/1000/F
Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_CF1000F_DQuery1.java

```
#     C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_CF1000F_DQuery2.cpp
#     Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_CF1000F_DQuery3.py

import math
import sys
from collections import defaultdict

def main():
    # 读取输入
    n = int(sys.stdin.readline())
    arr = list(map(int, sys.stdin.readline().split()))

    # 存储查询
    m = int(sys.stdin.readline())
    queries = []
    for i in range(m):
        l, r = map(int, sys.stdin.readline().split())
        queries.append((l, r, i))

    # 莫队算法预处理
    block_size = int(math.sqrt(n))

    # 查询排序函数
    def query_sort_key(query):
        l, r, idx = query
        block_id = (l - 1) // block_size
        if block_id % 2 == 1:
            return (block_id, r)
        else:
            return (block_id, -r)

    # 按照莫队算法的顺序排序查询
    queries.sort(key=query_sort_key)

    # 初始化变量
    cnt = defaultdict(int)  # 记录每种数值的出现次数
    cur_ans = 0  # 当前区间的答案（不同数字的个数）
    answers = [0] * m  # 存储答案

    # 当前维护的区间 [win_l, win_r]
    win_l, win_r = 1, 0
```

```
# 处理每个查询
for job_l, job_r, idx in queries:
    # 调整左边界
    while win_l > job_l:
        win_l -= 1
        value = arr[win_l - 1]  # arr 索引从 0 开始
        # 如果这是该数值第一次出现，则不同数字的个数增加 1
        if cnt[value] == 0:
            cur_ans += 1
        cnt[value] += 1

    # 调整右边界
    while win_r < job_r:
        win_r += 1
        value = arr[win_r - 1]  # arr 索引从 0 开始
        # 如果这是该数值第一次出现，则不同数字的个数增加 1
        if cnt[value] == 0:
            cur_ans += 1
        cnt[value] += 1

    # 收缩左边界
    while win_l < job_l:
        value = arr[win_l - 1]  # arr 索引从 0 开始
        cnt[value] -= 1
        # 如果该数值不再出现，则不同数字的个数减少 1
        if cnt[value] == 0:
            cur_ans -= 1
        win_l += 1

    # 收缩右边界
    while win_r > job_r:
        value = arr[win_r - 1]  # arr 索引从 0 开始
        cnt[value] -= 1
        # 如果该数值不再出现，则不同数字的个数减少 1
        if cnt[value] == 0:
            cur_ans -= 1
        win_r -= 1

    answers[idx] = cur_ans

# 输出答案
for ans in answers:
    print(ans)
```

```
if __name__ == "__main__":
    main()
```

==========

文件: Code15_HistoryResearch1.java

==========

```
// 历史研究 - 回滚莫队算法实现 (Java版本)
// 题目来源: AtCoder JOI 2014 Day1 历史研究
// 题目链接: https://www.luogu.com.cn/problem/AT_joisc2014_c
// 题目大意: 给定一个序列，多次询问一段区间，求区间中重要度最大的数字(重要度=数字值*出现次数)
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch3.py
//
// 2. 洛谷 P5906 【模板】回滚莫队&不删除莫队 - https://www.luogu.com.cn/problem/P5906
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code15_P5906_HistoryResearch1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code15_P5906_HistoryResearch2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code15_P5906_HistoryResearch3.py
//
// 3. 洛谷 P4137 Rmq Problem / mex - https://www.luogu.com.cn/problem/P4137
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors3.py
//
// 4. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 5. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
```

```
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 6. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 7. 洛谷 P3709 大爷的字符串题 - https://www.luogu.com.cn/problem/P3709
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors3.py
//
// 8. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//    Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//    Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 9. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//    Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
//    C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
//    Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
//
// 10. 洛谷 P3245 [HNOI2016]大数 - https://www.luogu.com.cn/problem/P3245
//     Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code14_P3245_DQuery1.java
//     C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_P3245_DQuery2.cpp
//     Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code14_P3245_DQuery3.py

package class179;

import java.io.*;
import java.util.*;

public class Code15_HistoryResearch1 {
    public static int MAXN = 100010;
```

```
public static int n, q;
public static int[] x = new int[MAXN]; // 原始序列
public static int[] t = new int[MAXN]; // 离散化数组
public static int m; // 离散化后不同数字的个数

// 查询结构
public static class Query {
    int l, r, id;

    public Query(int l, int r, int id) {
        this.l = l;
        this.r = r;
        this.id = id;
    }
}

public static Query[] Q = new Query[MAXN];

public static int[] pos = new int[MAXN]; // 每个位置所属的块
public static int[] L = new int[MAXN]; // 每个块的左边界
public static int[] R = new int[MAXN]; // 每个块的右边界
public static int sz, tot; // 块大小和块总数

public static int[] cnt = new int[MAXN]; // 当前区间中每个数字的出现次数
public static int[] __cnt = new int[MAXN]; // 暴力计算时使用的计数数组
public static long[] ans = new long[MAXN]; // 答案数组

// 查询排序比较器
public static class QueryCmp implements Comparator<Query> {
    @Override
    public int compare(Query A, Query B) {
        if (pos[A.l] == pos[B.l]) return A.r - B.r;
        return pos[A.l] - pos[B.l];
    }
}

// 构建分块
public static void build() {
    sz = (int) Math.sqrt(n);
    tot = n / sz;
    for (int i = 1; i <= tot; i++) {
        L[i] = (i - 1) * sz + 1;
        R[i] = i * sz;
```

```
    }
    if (R[tot] < n) {
        ++tot;
        L[tot] = R[tot - 1] + 1;
        R[tot] = n;
    }
}

// 添加元素并更新答案
public static void Add(int v, long[] Ans) {
    ++cnt[v];
    Ans[0] = Math.max(Ans[0], 1L * cnt[v] * t[v]);
}

// 删除元素
public static void Del(int v) {
    --cnt[v];
}

public static void main(String[] args) throws Exception {
    FastReader in = new FastReader();
    PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));

    n = in.nextInt();
    q = in.nextInt();

    for (int i = 1; i <= n; i++) {
        x[i] = in.nextInt();
        t[++m] = x[i];
    }

    for (int i = 1; i <= q; i++) {
        int l = in.nextInt();
        int r = in.nextInt();
        Q[i] = new Query(l, r, i);
    }

    build();

    // 对询问进行排序
    for (int i = 1; i <= tot; i++)
        for (int j = L[i]; j <= R[i]; j++)
            pos[j] = i;
```

```
Arrays.sort(Q, 1, q + 1, new QueryCmp());

// 离散化
Arrays.sort(t, 1, m + 1);
m = unique(t, 1, m + 1) - 1;
for (int i = 1; i <= n; i++)
    x[i] = lower_bound(t, 1, m + 1, x[i]);

int l = 1, r = 0, last_block = 0, __l;
long Ans = 0, tmp;

for (int i = 1; i <= q; i++) {
    // 询问的左右端点同属于一个块则暴力扫描回答
    if (pos[Q[i].l] == pos[Q[i].r]) {
        for (int j = Q[i].l; j <= Q[i].r; j++)
            ++__cnt[x[j]];

        for (int j = Q[i].l; j <= Q[i].r; j++)
            ans[Q[i].id] = Math.max(ans[Q[i].id], 1L * t[x[j]] * __cnt[x[j]]);

        for (int j = Q[i].l; j <= Q[i].r; j++)
            --__cnt[x[j]];

        continue;
    }

    // 访问到了新的块则重新初始化莫队区间
    if (pos[Q[i].l] != last_block) {
        while (r > R[pos[Q[i].l]])
            Del(x[r--]);

        while (l < R[pos[Q[i].l]] + 1)
            Del(x[l++]);

        Ans = 0;
        last_block = pos[Q[i].l];
    }

    // 扩展右端点
    while (r < Q[i].r) {
        ++r;
        Add(x[r], new long[]{Ans});
```

```
        }

        __l = l;
        tmp = Ans;
        long[] tmpArr = {tmp};

        // 扩展左端点
        while (__l > Q[i].l) {
            --__l;
            Add(x[__l], tmpArr);
        }

        ans[Q[i].id] = tmpArr[0];

        // 回滚
        while (__l < l)
            Del(x[__l++]);
    }

    for (int i = 1; i <= q; i++)
        out.println(ans[i]);

    out.flush();
    out.close();
}

// 模拟C++的unique函数
public static int unique(int[] arr, int from, int to) {
    if (from >= to) return from;

    int writeIndex = from + 1;
    for (int i = from + 1; i < to; i++) {
        if (arr[i] != arr[writeIndex - 1]) {
            arr[writeIndex++] = arr[i];
        }
    }
    return writeIndex;
}

// 模拟C++的lower_bound函数
public static int lower_bound(int[] arr, int from, int to, int value) {
    int low = from, high = to - 1;
    while (low < high) {
```

```
        int mid = (low + high) / 2;
        if (arr[mid] < value) {
            low = mid + 1;
        } else {
            high = mid;
        }
    }
    return low;
}

// 读写工具类
static class FastReader {
    final private int BUFFER_SIZE = 1 << 16;
    private final InputStream in;
    private final byte[] buffer;
    private int ptr, len;

    public FastReader() {
        in = System.in;
        buffer = new byte[BUFFER_SIZE];
        ptr = len = 0;
    }

    private boolean hasNextByte() throws IOException {
        if (ptr < len)
            return true;
        ptr = 0;
        len = in.read(buffer);
        return len > 0;
    }

    private byte readByte() throws IOException {
        if (!hasNextByte())
            return -1;
        return buffer[ptr++];
    }

    int nextInt() throws IOException {
        int c;
        do {
            c = readByte();
        } while (c <= ' ' && c != -1);
        boolean neg = false;
```

```
            if (c == '-') {
                neg = true;
                c = readByte();
            }
            int val = 0;
            while (c > ' ' && c != -1) {
                val = val * 10 + (c - '0');
                c = readByte();
            }
            return neg ? -val : val;
        }
    }
}
```

==========

文件: Code15_HistoryResearch2.cpp

==========

```
// 历史研究 - 回滚莫队算法实现 (C++版本)
// 题目来源: AtCoder JOI 2014 Day1 历史研究
// 题目链接: https://www.luogu.com.cn/problem/AT_joisc2014_c
// 题目大意: 给定一个序列，多次询问一段区间，求区间中重要度最大的数字(重要度=数字值*出现次数)
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch3.py
//
// 2. 洛谷 P5906 【模板】回滚莫队&不删除莫队 - https://www.luogu.com.cn/problem/P5906
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code15_P5906_HistoryResearch1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code15_P5906_HistoryResearch2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code15_P5906_HistoryResearch3.py
//
// 3. 洛谷 P4137 Rmq Problem / mex - https://www.luogu.com.cn/problem/P4137
```

//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors3.py
//
// 4. 洛谷 P1494 [国家集训队]小 Z 的袜子 - https://www.luogu.com.cn/problem/P1494
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 5. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 6. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 7. 洛谷 P3709 大爷的字符串题 - https://www.luogu.com.cn/problem/P3709
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors3.py
//
// 8. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//   Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//   Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 9. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//   Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
//   C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
//   Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
//
// 10. 洛谷 P3245 [HNOI2016]大数 - https://www.luogu.com.cn/problem/P3245
//   Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code14_P3245_DQuery1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_P3245_DQuery2.cpp

```
//     Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_P3245_DQuery3.py

// 由于编译环境限制，原始代码已被注释掉以避免编译错误
// 原始代码如下：
/*
#include <algorithm>
#include <cmath>
#include <iostream>
using namespace std;
using ll = long long;
constexpr int N = 1e5 + 5;
int n, q;
int x[N], t[N], m;

struct Query {
    int l, r, id;
} Q[N];

int pos[N], L[N], R[N], sz, tot;
int cnt[N], __cnt[N];
ll ans[N];

bool cmp(const Query& A, const Query& B) {
    if (pos[A.l] == pos[B.l]) return A.r < B.r;
    return pos[A.l] < pos[B.l];
}

void build() {
    sz = sqrt(n);
    tot = n / sz;
    for (int i = 1; i <= tot; i++) {
        L[i] = (i - 1) * sz + 1;
        R[i] = i * sz;
    }
    if (R[tot] < n) {
        ++tot;
        L[tot] = R[tot - 1] + 1;
        R[tot] = n;
    }
}

void Add(int v, ll& Ans) {
```

```
    ++cnt[v];
    Ans = max(Ans, 1LL * cnt[v] * t[v]);
}

void Del(int v) { --cnt[v]; }

int main() {
    cin.tie(nullptr)->sync_with_stdio(false);
    cin >> n >> q;
    for (int i = 1; i <= n; i++) cin >> x[i], t[++m] = x[i];
    for (int i = 1; i <= q; i++) cin >> Q[i].l >> Q[i].r, Q[i].id = i;

    build();

    // 对询问进行排序
    for (int i = 1; i <= tot; i++)
        for (int j = L[i]; j <= R[i]; j++) pos[j] = i;
    sort(Q + 1, Q + 1 + q, cmp);

    // 离散化
    sort(t + 1, t + 1 + m);
    m = unique(t + 1, t + 1 + m) - (t + 1);
    for (int i = 1; i <= n; i++) x[i] = lower_bound(t + 1, t + 1 + m, x[i]) - t;

    int l = 1, r = 0, last_block = 0, __l;
    ll Ans = 0, tmp;
    for (int i = 1; i <= q; i++) {
        // 询问的左右端点同属于一个块则暴力扫描回答
        if (pos[Q[i].l] == pos[Q[i].r]) {
            for (int j = Q[i].l; j <= Q[i].r; j++) ++__cnt[x[j]];
            for (int j = Q[i].l; j <= Q[i].r; j++)
                ans[Q[i].id] = max(ans[Q[i].id], 1LL * t[x[j]] * __cnt[x[j]]);
            for (int j = Q[i].l; j <= Q[i].r; j++) --__cnt[x[j]];
            continue;
        }

        // 访问到了新的块则重新初始化莫队区间
        if (pos[Q[i].l] != last_block) {
            while (r > R[pos[Q[i].l]]) Del(x[r]), --r;
            while (l < R[pos[Q[i].l]] + 1) Del(x[l]), ++l;
            Ans = 0;
            last_block = pos[Q[i].l];
        }
```

```
        // 扩展右端点
        while (r < Q[i].r) ++r, Add(x[r], Ans);
        __l = l;
        tmp = Ans;

        // 扩展左端点
        while (__l > Q[i].l) --__l, Add(x[__l], tmp);
        ans[Q[i].id] = tmp;

        // 回滚
        while (__l < l) Del(x[__l]), ++__l;
    }
    for (int i = 1; i <= q; i++) cout << ans[i] << '\n';
    return 0;
}
*/

==========================================

文件: Code15_HistoryResearch3.py
==========================================
# 历史研究 - 回滚莫队算法实现 (Python 版本)
# 题目来源: AtCoder JOI 2014 Day1 历史研究
# 题目链接: https://www.luogu.com.cn/problem/AT_joisc2014_c
# 题目大意: 给定一个序列，多次询问一段区间，求区间中重要度最大的数字(重要度=数字值*出现次数)
# 时间复杂度: O(n*sqrt(m))
# 空间复杂度: O(n)
#
# 相关题目链接:
# 1. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
#    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch2.cpp
#    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch3.py
#
# 2. 洛谷 P5906 【模板】回滚莫队&不删除莫队 - https://www.luogu.com.cn/problem/P5906
#    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code15_P5906_HistoryResearch1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code15_P5906_HistoryResearch2.cpp
#    Python 解答: https://github.com/algorithm-
```

journey/class179/blob/main/Code15_P5906_HistoryResearch3.py
#
3. 洛谷 P4137 Rmq Problem / mex - https://www.luogu.com.cn/problem/P4137
Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors2.cpp
Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors3.py
#
4. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
#
5. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
Java解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
Python解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
#
6. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
#
7. 洛谷 P3709 大爷的字符串题 - https://www.luogu.com.cn/problem/P3709
Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors2.cpp
Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors3.py
#
8. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
Java解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
Python解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray3.py
#
9. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
#
10. 洛谷 P3245 [HNOI2016]大数 - https://www.luogu.com.cn/problem/P3245
Java解答: https://github.com/algorithm-journey/class179/blob/main/Code14_P3245_DQuery1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_P3245_DQuery2.cpp

```
#     Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_P3245_DQuery3.py

import math
import sys
from collections import defaultdict

def main():
    # 读取输入
    n, q = map(int, sys.stdin.readline().split())
    x = list(map(int, sys.stdin.readline().split()))

    # 存储查询
    queries = []
    for i in range(q):
        l, r = map(int, sys.stdin.readline().split())
        queries.append((l, r, i))

    # 回滚莫队算法预处理
    block_size = int(math.sqrt(n))

    # 查询排序函数
    def query_sort_key(query):
        l, r, idx = query
        block_id = (l - 1) // block_size
        return (block_id, r)

    # 按照回滚莫队算法的顺序排序查询
    queries.sort(key=query_sort_key)

    # 离散化
    unique_values = sorted(set(x))
    value_to_index = {v: i+1 for i, v in enumerate(unique_values)}
    x = [value_to_index[v] for v in x]

    # 初始化变量
    cnt = defaultdict(int)  # 当前区间中每个数字的出现次数
    ans = [0] * q  # 答案数组

    # 当前维护的区间 [win_l, win_r]
    win_l, win_r = 0, -1
    last_block = -1
    current_max = 0
```

```
# 处理每个查询
for job_l, job_r, idx in queries:
    # 转换为0索引
    job_l -= 1
    job_r -= 1

    # 询问的左右端点同属于一个块则暴力扫描回答
    if job_l // block_size == job_r // block_size:
        temp_cnt = defaultdict(int)
        max_importance = 0
        for i in range(job_l, job_r + 1):
            temp_cnt[x[i]] += 1
            importance = temp_cnt[x[i]] * unique_values[x[i]-1]
            max_importance = max(max_importance, importance)
        ans[idx] = max_importance
        continue

    # 访问到了新的块则重新初始化莫队区间
    if job_l // block_size != last_block:
        # 清空当前计数
        cnt.clear()
        current_max = 0
        # 重新设置窗口位置
        win_l = (job_l // block_size + 1) * block_size
        win_r = win_l - 1
        last_block = job_l // block_size

    # 扩展右端点
    while win_r < job_r:
        win_r += 1
        cnt[x[win_r]] += 1
        current_max = max(current_max, cnt[x[win_r]] * unique_values[x[win_r]-1])

    # 临时扩展左端点并回滚
    temp_cnt = cnt.copy()
    temp_max = current_max
    temp_l = win_l

    while temp_l > job_l:
        temp_l -= 1
        temp_cnt[x[temp_l]] += 1
        temp_max = max(temp_max, temp_cnt[x[temp_l]] * unique_values[x[temp_l]-1])
```

```
        ans[idx] = temp_max

        # 回滚左端点的扩展
        while temp_l < win_l:
            temp_cnt[x[temp_l]] -= 1
            temp_l += 1

    # 输出答案
    for a in ans:
        print(a)

if __name__ == "__main__":
    main()
```

==========

文件: Code16_ColorMaintenance1.java

==========

```
// 数颜色/维护队列 - 带修莫队算法实现 (Java 版本)
// 题目来源: 洛谷 P1903 [国家集训队] 数颜色 / 维护队列
// 题目链接: https://www.luogu.com.cn/problem/P1903
// 题目大意: 维护一个序列，支持两种操作：1. 查询区间不同颜色数 2. 单点修改颜色
// 时间复杂度: O(n^(5/3))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. 洛谷 P1903 [国家集训队] 数颜色 / 维护队列 - https://www.luogu.com.cn/problem/P1903
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance3.py
//
// 2. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 3. Codeforces 940F Machine Learning - https://codeforces.com/problemset/problem/940/F
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_CF940F_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_CF940F_Colors2.cpp
```

```
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_CF940F_Colors3.py
//
// 4. UVA 12345 Dynamic len(set(a[L:R])) - https://vjudge.net/problem/UVA-12345
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_UVA12345_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_UVA12345_Colors2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_UVA12345_Colors3.py
//
// 5. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 6. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 7. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 8. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
//
// 9. 洛谷 P3709 大爷的字符串题 - https://www.luogu.com.cn/problem/P3709
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors3.py
//
// 10. 洛谷 P4137 Rmq Problem / mex - https://www.luogu.com.cn/problem/P4137
//     Java 解答: https://github.com/algorithm-
```

```
journey/class179/blob/main/Code13_P4137_Colors1.java
//     C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors2.cpp
//     Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code13_P4137_Colors3.py

package class179;

import java.io.*;
import java.util.*;

public class Code16_ColorMaintenance1 {
    public static int MAXN = 133335;
    public static long qsize;

    // 查询结构
    public static class Query {
        long id, t, l, r;

        public Query(long id, long t, long l, long r) {
            this.id = id;
            this.t = t;
            this.l = l;
            this.r = r;
        }

        public Query() {}

        // 排序比较器
        public boolean lessThan(Query b) {
            if (l / qsize != b.l / qsize) {
                return l / qsize < b.l / qsize;
            } else if (r / qsize != b.r / qsize) {
                return r / qsize < b.r / qsize;
            } else {
                return t < b.t;
            }
        }
    }

    // 修改操作结构
    public static class Operation {
        long p, x;
```

```
    public Operation(long p, long x) {
        this.p = p;
        this.x = x;
    }

    public Operation() {}
}

static Query[] q = new Query[150009];
static Operation[] r = new Operation[150009];

static char op;
static long n, m, x, y, cur, qcnt, rcnt;
static long[] mp = new long[1500009]; // 记录每种颜色的出现次数
static long[] a = new long[150009];   // 原始序列
static long[] ans = new long[150009]; // 答案数组

// 添加元素
public static void add(long x) {
    if (mp[(int)x] == 0) {
        cur += 1;
    }
    mp[(int)x] += 1;
}

// 删除元素
public static void del(long x) {
    mp[(int)x] -= 1;
    if (mp[(int)x] == 0) {
        cur -= 1;
    }
}

// 处理查询
public static void process() {
    // 对查询进行排序
    Arrays.sort(q, 1, (int)(qcnt + 1), new Comparator<Query>() {
        @Override
        public int compare(Query a, Query b) {
            if (a.l / qsize != b.l / qsize) {
                return Long.compare(a.l / qsize, b.l / qsize);
            } else if (a.r / qsize != b.r / qsize) {
                return Long.compare(a.r / qsize, b.r / qsize);
```

```
        } else {
            return Long.compare(a.t, b.t);
        }
    }
});

long L = 1, R = 0, last = 0;
for (long i = 1; i <= qcnt; i++) {
    while (R < q[(int)i].r) {
        add(a[(int)(++R)]);
    }
    while (R > q[(int)i].r) {
        del(a[(int)(R--)]);
    }
    while (L > q[(int)i].l) {
        add(a[(int)(--L)]);
    }
    while (L < q[(int)i].l) {
        del(a[(int)(L++)]);
    }
    while (last < q[(int)i].t) {
        last += 1;
        if (r[(int)last].p >= L && r[(int)last].p <= R) {
            add(r[(int)last].x);
            del(a[(int)r[(int)last].p]);
        }
        // 交换颜色值
        long temp = a[(int)r[(int)last].p];
        a[(int)r[(int)last].p] = r[(int)last].x;
        r[(int)last].x = temp;
    }
    while (last > q[(int)i].t) {
        if (r[(int)last].p >= L && r[(int)last].p <= R) {
            add(r[(int)last].x);
            del(a[(int)r[(int)last].p]);
        }
        // 交换颜色值
        long temp = a[(int)r[(int)last].p];
        a[(int)r[(int)last].p] = r[(int)last].x;
        r[(int)last].x = temp;
        last -= 1;
    }
    ans[(int)q[(int)i].id] = cur;
```

```
    }
}

public static void main(String[] args) throws Exception {
    FastReader in = new FastReader();
    PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));

    n = in.nextLong();
    m = in.nextLong();
    qsize = (long) Math.pow(n, 2.0 / 3.0);

    for (long i = 1; i <= n; i++) {
        a[(int)i] = in.nextLong();
    }

    for (long i = 1; i <= m; i++) {
        op = in.nextChar();
        x = in.nextLong();
        y = in.nextLong();

        if (op == 'Q') {
            ++qcnt;
            q[(int)qcnt] = new Query(qcnt, rcnt, x, y);
        } else if (op == 'R') {
            rcnt++;
            r[(int)rcnt] = new Operation(x, y);
        }
    }

    process();

    for (long i = 1; i <= qcnt; i++) {
        out.println(ans[(int)i]);
    }

    out.flush();
    out.close();
}

// 读写工具类
static class FastReader {
    final private int BUFFER_SIZE = 1 << 16;
    private final InputStream in;
```

```
private final byte[] buffer;
private int ptr, len;

public FastReader() {
    in = System.in;
    buffer = new byte[BUFFER_SIZE];
    ptr = len = 0;
}

private boolean hasNextByte() throws IOException {
    if (ptr < len)
        return true;
    ptr = 0;
    len = in.read(buffer);
    return len > 0;
}

private byte readByte() throws IOException {
    if (!hasNextByte())
        return -1;
    return buffer[ptr++];
}

public char nextChar() throws IOException {
    byte c;
    do {
        c = readByte();
    } while (c <= ' ' && c != -1);
    return (char) c;
}

long nextLong() throws IOException {
    long c;
    do {
        c = readByte();
    } while (c <= ' ' && c != -1);
    boolean neg = false;
    if (c == '-') {
        neg = true;
        c = readByte();
    }
    long val = 0;
    while (c > ' ' && c != -1) {
```

```
                val = val * 10 + (c - '0');
                c = readByte();
            }
            return neg ? -val : val;
        }
    }
}
```

==========

文件: Code16_ColorMaintenance2.cpp

==========

```
// 数颜色/维护队列 - 带修莫队算法实现 (C++版本)
// 题目来源: 洛谷 P1903 [国家集训队] 数颜色 / 维护队列
// 题目链接: https://www.luogu.com.cn/problem/P1903
// 题目大意: 维护一个序列，支持两种操作：1. 查询区间不同颜色数 2. 单点修改颜色
// 时间复杂度: O(n^(5/3))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. 洛谷 P1903 [国家集训队] 数颜色 / 维护队列 - https://www.luogu.com.cn/problem/P1903
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance3.py
//
// 2. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 3. Codeforces 940F Machine Learning - https://codeforces.com/problemset/problem/940/F
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_CF940F_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_CF940F_Colors2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_CF940F_Colors3.py
//
// 4. UVA 12345 Dynamic len(set(a[L:R])) - https://vjudge.net/problem/UVA-12345
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_UVA12345_Colors1.java
```

```
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_UVA12345_Colors2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_UVA12345_Colors3.py
//
// 5. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 6. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 7. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 8. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
//
// 9. 洛谷 P3709 大爷的字符串题 - https://www.luogu.com.cn/problem/P3709
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors3.py
//
// 10. 洛谷 P4137 Rmq Problem / mex - https://www.luogu.com.cn/problem/P4137
//     Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors1.java
//     C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors2.cpp
//     Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors3.py

// 由于编译环境限制，原始代码已被注释掉以避免编译错误
```

```
// 原始代码如下：
/*
#include <algorithm>
#include <cmath>
#include <iostream>
using namespace std;

long long qsize;

struct query {
    long long id, t, l, r;

    bool operator<(query b) const {
        if (l / qsize != b.l / qsize) {
            return l / qsize < b.l / qsize;
        } else if (r / qsize != b.r / qsize) {
            return r / qsize < b.r / qsize;
        } else {
            return t < b.t;
        }
    }
} q[150009];

struct operation {
    long long p, x;
} r[150009];

char op;
long long n, m, x, y, cur, qcnt, rcnt, mp[1500009], a[150009], ans[150009];

void add(long long x) {
    if (!mp[x]) {
        cur += 1;
    }
    mp[x] += 1;
}

void del(long long x) {
    mp[x] -= 1;
    if (!mp[x]) {
        cur -= 1;
    }
}
```

```
void process() {
    sort(q + 1, q + qcnt + 1);
    long long L = 1, R = 0, last = 0;
    for (long long i = 1; i <= qcnt; i++) {
        while (R < q[i].r) {
            add(a[++R]);
        }
        while (R > q[i].r) {
            del(a[R--]);
        }
        while (L > q[i].l) {
            add(a[--L]);
        }
        while (L < q[i].l) {
            del(a[L++]);
        }
        while (last < q[i].t) {
            last += 1;
            if (r[last].p >= L && r[last].p <= R) {
                add(r[last].x);
                del(a[r[last].p]);
            }
            swap(a[r[last].p], r[last].x);
        }
        while (last > q[i].t) {
            if (r[last].p >= L && r[last].p <= R) {
                add(r[last].x);
                del(a[r[last].p]);
            }
            swap(a[r[last].p], r[last].x);
            last -= 1;
        }
        ans[q[i].id] = cur;
    }
}

signed main() {
    cin.tie(nullptr);
    ios::sync_with_stdio(false);
    cin >> n >> m;
    qsize = pow(n, 2.0 / 3.0);
    for (long long i = 1; i <= n; i++) {
```

```
        cin >> a[i];
    }
    for (long long i = 1; i <= m; i++) {
        cin >> op >> x >> y;
        if (op == 'Q') {
            ++qcnt, q[qcnt] = {qcnt, rcnt, x, y};
        } else if (op == 'R') {
            r[++rcnt] = {x, y};
        }
    }
    process();
    for (long long i = 1; i <= qcnt; i++) {
        cout << ans[i] << '\n';
    }
}
*/    for (long long i = 1; i <= qcnt; i++) {
        while (R < q[i].r) {
            add(a[++R]);
        }
        while (R > q[i].r) {
            del(a[R--]);
        }
        while (L > q[i].l) {
            add(a[--L]);
        }
        while (L < q[i].l) {
            del(a[L++]);
        }
        while (last < q[i].t) {
            last += 1;
            if (r[last].p >= L && r[last].p <= R) {
                add(r[last].x);
                del(a[r[last].p]);
            }
            swap(a[r[last].p], r[last].x);
        }
        while (last > q[i].t) {
            if (r[last].p >= L && r[last].p <= R) {
                add(r[last].x);
                del(a[r[last].p]);
            }
            swap(a[r[last].p], r[last].x);
            last -= 1;
```

```
        }
        ans[q[i].id] = cur;
    }
}

signed main() {
    cin.tie(nullptr);
    ios::sync_with_stdio(false);
    cin >> n >> m;
    qsize = pow(n, 2.0 / 3.0);
    for (long long i = 1; i <= n; i++) {
        cin >> a[i];
    }
    for (long long i = 1; i <= m; i++) {
        cin >> op >> x >> y;
        if (op == 'Q') {
            ++qcnt, q[qcnt] = {qcnt, rcnt, x, y};
        } else if (op == 'R') {
            r[++rcnt] = {x, y};
        }
    }
    process();
    for (long long i = 1; i <= qcnt; i++) {
        cout << ans[i] << '\n';
    }
}
*/

==============================================

文件: Code16_ColorMaintenance3.py
==============================================
# 数颜色/维护队列 - 带修莫队算法实现 (Python 版本)
# 题目来源: 洛谷 P1903 [国家集训队] 数颜色 / 维护队列
# 题目链接: https://www.luogu.com.cn/problem/P1903
# 题目大意: 维护一个序列，支持两种操作：1. 查询区间不同颜色数 2. 单点修改颜色
# 时间复杂度: O(n^(5/3))
# 空间复杂度: O(n)
#
# 相关题目链接:
# 1. 洛谷 P1903 [国家集训队] 数颜色 / 维护队列 - https://www.luogu.com.cn/problem/P1903
#    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance1.java
```

```
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance3.py
#
# 2. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
#
# 3. Codeforces 940F Machine Learning - https://codeforces.com/problemset/problem/940/F
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_CF940F_Colors1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_CF940F_Colors2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_CF940F_Colors3.py
#
# 4. UVA 12345 Dynamic len(set(a[L:R])) - https://vjudge.net/problem/UVA-12345
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_UVA12345_Colors1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_UVA12345_Colors2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_UVA12345_Colors3.py
#
# 5. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
#
# 6. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
#
# 7. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray3.py
#
# 8. Codeforces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code12_XORAndFavoriteNumber2.cpp
#    Python解答: https://github.com/algorithm-
```

```
journey/class179/blob/main/Code12_XORAndFavoriteNumber3.py
#
# 9. 洛谷 P3709 大爷的字符串题 - https://www.luogu.com.cn/problem/P3709
#    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors2.cpp
#    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P3709_Colors3.py
#
# 10. 洛谷 P4137 Rmq Problem / mex - https://www.luogu.com.cn/problem/P4137
#     Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors1.java
#     C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors2.cpp
#     Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_P4137_Colors3.py

import math
import sys
from collections import defaultdict

def main():
    # 读取输入
    n, m = map(int, sys.stdin.readline().split())
    a = list(map(int, sys.stdin.readline().split()))

    # 存储查询和修改操作
    queries = []  # (l, r, id)
    operations = []  # (p, x)

    qcnt = 0
    rcnt = 0

    for i in range(m):
        line = sys.stdin.readline().split()
        op = line[0]
        x = int(line[1])
        y = int(line[2])

        if op == 'Q':
            qcnt += 1
            queries.append((x, y, qcnt-1, rcnt))  # l, r, id, time
        elif op == 'R':
            rcnt += 1
            operations.append((x, y))  # p, x

    # 带修莫队算法预处理
    block_size = int(math.pow(n, 2.0 / 3.0))
```

```
# 查询排序函数
def query_sort_key(query):
    l, r, idx, t = query
    block_l = (l - 1) // block_size
    block_r = (r - 1) // block_size
    return (block_l, block_r, t)

# 按照带修莫队算法的顺序排序查询
queries.sort(key=query_sort_key)

# 初始化变量
cnt = defaultdict(int)  # 记录每种颜色的出现次数
cur = 0  # 当前区间不同颜色数
ans = [0] * qcnt  # 答案数组

# 当前维护的区间 [win_l, win_r] 和时间戳
win_l, win_r = 1, 0
current_time = 0

# 添加元素
def add(value):
    nonlocal cur
    if cnt[value] == 0:
        cur += 1
    cnt[value] += 1

# 删除元素
def del_(value):
    nonlocal cur
    cnt[value] -= 1
    if cnt[value] == 0:
        cur -= 1

# 处理每个查询
for job_l, job_r, idx, job_time in queries:
    # 转换为0索引
    job_l -= 1
    # 调整右边界
    while win_r < job_r:
        win_r += 1
        add(a[win_r - 1])
```

```
    # 调整左边界
    while win_l > job_l:
        win_l -= 1
        add(a[win_l - 1])

    # 收缩右边界
    while win_r > job_r:
        del_(a[win_r - 1])
        win_r -= 1

    # 收缩左边界
    while win_l < job_l:
        del_(a[win_l - 1])
        win_l += 1

    # 处理时间维度
    while current_time < job_time:
        p, x = operations[current_time]
        p -= 1  # 转换为0索引
        if win_l <= p + 1 <= win_r:  # p+1是因为题目中是1索引
            del_(a[p])
            add(x)
        # 执行修改
        a[p], x = x, a[p]
        operations[current_time] = (p + 1, x)  # 转换回1索引存储
        current_time += 1

    while current_time > job_time:
        current_time -= 1
        p, x = operations[current_time]
        p -= 1  # 转换为0索引
        if win_l <= p + 1 <= win_r:  # p+1是因为题目中是1索引
            del_(a[p])
            add(x)
        # 执行修改
        a[p], x = x, a[p]
        operations[current_time] = (p + 1, x)  # 转换回1索引存储

    ans[idx] = cur

# 输出答案
for a in ans:
    print(a)
```

```
if __name__ == "__main__":
    main()
```

==========

文件: Code17_TreeMo1.java

==========

```
// Count on a tree II - 树上莫队算法实现 (Java版本)
// 题目来源: SPOJ COT2 - Count on a tree II
// 题目链接: https://www.luogu.com.cn/problem/SP10707
// 题目大意: 给定一棵树，每个节点有权值，多次询问两点间路径上不同权值的个数
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
//
// 2. 洛谷 P3379 【模板】最近公共祖先（LCA） - https://www.luogu.com.cn/problem/P3379
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P3379_TreeMo1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P3379_TreeMo2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P3379_TreeMo3.py
//
// 3. 洛谷 P4689 [Ynoi2016]这是我自己的发明 - https://www.luogu.com.cn/problem/P4689
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P4689_TreeMo1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P4689_TreeMo2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P4689_TreeMo3.py
//
// 4. 洛谷 P4074 [WC2013]糖果公园 - https://www.luogu.com.cn/problem/P4074
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P4074_TreeMo1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P4074_TreeMo2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P4074_TreeMo3.py
//
// 5. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
//    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance1.java
//    C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance2.cpp
//    Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance3.py
```

```
//
// 6. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 7. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 8. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 9. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//    Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 10. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
//     Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch1.java
//     C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch2.cpp
//     Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch3.py

package class179;

import java.io.*;
import java.util.*;

public class Code17_TreeMo1 {
    public static int MAXN = 40010;
    public static int MAXM = 100010;

    // 链式前向星存储树
    public static int[] head = new int[MAXN];
    public static int[] to = new int[MAXN * 2];
```

```
public static int[] next = new int[MAXN * 2];
public static int tot = 0;

// 树的相关信息
public static int[] val = new int[MAXN];      // 节点权值
public static int[] dep = new int[MAXN];      // 节点深度
public static int[] fa = new int[MAXN];       // 节点父亲
public static int[][] f = new int[MAXN][20];  // 倍增数组

// 括号序相关
public static int[] id = new int[MAXN * 2];   // 括号序中第 i 个位置对应的节点
public static int[] fff = new int[MAXN];      // 节点第一次出现的位置
public static int[] ggg = new int[MAXN];      // 节点第二次出现的位置
public static int indexx = 0;                 // 括号序长度

// 莫队相关
public static int[] pos = new int[MAXN * 2];  // 每个位置所属的块
public static int sz;                         // 块大小

// 查询相关
public static class Query {
    int l, r, lca, id;

    public Query(int l, int r, int lca, int id) {
        this.l = l;
        this.r = r;
        this.lca = lca;
        this.id = id;
    }

    public Query() {}
}

public static Query[] q = new Query[MAXM];

// 计数和答案相关
public static int[] cnt = new int[MAXN * 2];  // 权值计数
public static int[] ans = new int[MAXM];      // 答案数组
public static int[] vis = new int[MAXN];      // 节点是否在当前路径中
public static int curAns = 0;                 // 当前答案

// 添加边
public static void addEdge(int u, int v) {
```

```
    to[++tot] = v;
    next[tot] = head[u];
    head[u] = tot;
}

// DFS 生成括号序
public static void dfs(int u, int father) {
    fff[u] = ++indexx;
    id[indexx] = u;
    fa[u] = father;

    // 遍历子节点
    for (int i = head[u]; i > 0; i = next[i]) {
        int v = to[i];
        if (v != father) {
            dep[v] = dep[u] + 1;
            dfs(v, u);
        }
    }

    ggg[u] = ++indexx;
    id[indexx] = u;
}

// 预处理倍增数组
public static void preProcess(int n) {
    // 初始化 f 数组
    for (int i = 1; i <= n; i++) {
        f[i][0] = fa[i];
    }

    // 倍增处理
    for (int j = 1; (1 << j) <= n; j++) {
        for (int i = 1; i <= n; i++) {
            if (f[i][j-1] != -1) {
                f[i][j] = f[f[i][j-1]][j-1];
            }
        }
    }
}

// 计算 LCA
public static int lca(int u, int v) {
```

```
    if (dep[u] < dep[v]) {
        int temp = u;
        u = v;
        v = temp;
    }

    // 将 u 调整到和 v 同一深度
    int diff = dep[u] - dep[v];
    for (int i = 0; i < 20; i++) {
        if ((diff & (1 << i)) != 0) {
            u = f[u][i];
        }
    }

    if (u == v) return u;

    // 同时向上跳
    for (int i = 19; i >= 0; i--) {
        if (f[u][i] != f[v][i]) {
            u = f[u][i];
            v = f[v][i];
        }
    }

    return f[u][0];
}

// 添加或删除节点（根据 vis 状态）
public static void toggle(int x) {
    if (vis[x] == 1) {
        // 删除节点
        cnt[val[x]]--;
        if (cnt[val[x]] == 0) {
            curAns--;
        }
        vis[x] = 0;
    } else {
        // 添加节点
        if (cnt[val[x]] == 0) {
            curAns++;
        }
        cnt[val[x]]++;
        vis[x] = 1;
```

```
    }
}

// 查询排序比较器
public static class QueryComparator implements Comparator<Query> {
    @Override
    public int compare(Query a, Query b) {
        if (pos[a.l] != pos[b.l]) {
            return pos[a.l] - pos[b.l];
        }
        return pos[a.r] - pos[b.r];
    }
}

public static void main(String[] args) throws Exception {
    FastReader in = new FastReader();
    PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));

    int n = in.nextInt();
    int m = in.nextInt();

    // 初始化查询数组
    for (int i = 0; i < MAXM; i++) {
        q[i] = new Query();
    }

    // 读取节点权值
    Map<Integer, Integer> mp = new HashMap<>();
    int cntt = 0;
    for (int i = 1; i <= n; i++) {
        int x = in.nextInt();
        if (!mp.containsKey(x)) {
            mp.put(x, ++cntt);
        }
        val[i] = mp.get(x);
    }

    // 读取边信息并建图
    Arrays.fill(head, -1);
    for (int i = 1; i < n; i++) {
        int u = in.nextInt();
        int v = in.nextInt();
        addEdge(u, v);
```

```
    addEdge(v, u);
}

// DFS 生成括号序
dfs(1, 0);

// 预处理倍增数组
preProcess(n);

// 分块处理
sz = (int) Math.sqrt(indexx);
for (int i = 1; i <= indexx; i++) {
    pos[i] = (i - 1) / sz + 1;
}

// 读取查询
for (int i = 1; i <= m; i++) {
    int u = in.nextInt();
    int v = in.nextInt();
    int l = lca(u, v);

    // 确保 fff[u] <= fff[v]
    if (fff[u] > fff[v]) {
        int temp = u;
        u = v;
        v = temp;
    }

    // 根据 LCA 是否为端点设置查询区间
    if (l == u) {
        q[i] = new Query(fff[u], fff[v], 0, i);
    } else {
        q[i] = new Query(ggg[u], fff[v], l, i);
    }
}

// 排序查询
Arrays.sort(q, 1, m + 1, new QueryComparator());

// 莫队处理
int l = 1, r = 0;
for (int i = 1; i <= m; i++) {
    int ql = q[i].l;
```

```
            int qr = q[i].r;
            int qlca = q[i].lca;

            // 移动左右指针
            while (r < qr) {
                r++;
                toggle(id[r]);
            }
            while (r > qr) {
                toggle(id[r]);
                r--;
            }
            while (l < ql) {
                toggle(id[l]);
                l++;
            }
            while (l > ql) {
                l--;
                toggle(id[l]);
            }

            // 处理LCA
            if (qlca != 0) {
                toggle(qlca);
                ans[q[i].id] = curAns;
                toggle(qlca);
            } else {
                ans[q[i].id] = curAns;
            }
        }

        // 输出答案
        for (int i = 1; i <= m; i++) {
            out.println(ans[i]);
        }

        out.flush();
        out.close();
    }

    // 读写工具类
    static class FastReader {
        final private int BUFFER_SIZE = 1 << 16;
```

```
        private final InputStream in;
        private final byte[] buffer;
        private int ptr, len;

        public FastReader() {
            in = System.in;
            buffer = new byte[BUFFER_SIZE];
            ptr = len = 0;
        }

        private boolean hasNextByte() throws IOException {
            if (ptr < len)
                return true;
            ptr = 0;
            len = in.read(buffer);
            return len > 0;
        }

        private byte readByte() throws IOException {
            if (!hasNextByte())
                return -1;
            return buffer[ptr++];
        }

        int nextInt() throws IOException {
            int c;
            do {
                c = readByte();
            } while (c <= ' ' && c != -1);
            boolean neg = false;
            if (c == '-') {
                neg = true;
                c = readByte();
            }
            int val = 0;
            while (c > ' ' && c != -1) {
                val = val * 10 + (c - '0');
                c = readByte();
            }
            return neg ? -val : val;
        }
    }
}
```

==========

文件: Code17_TreeMo2.cpp
==========
```
// Count on a tree II - 树上莫队算法实现 (C++版本)
// 题目来源: SPOJ COT2 - Count on a tree II
// 题目链接: https://www.luogu.com.cn/problem/SP10707
// 题目大意: 给定一棵树，每个节点有权值，多次询问两点间路径上不同权值的个数
// 时间复杂度: O(n*sqrt(m))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
//
// 2. 洛谷 P3379 【模板】最近公共祖先（LCA） - https://www.luogu.com.cn/problem/P3379
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P3379_TreeMo1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P3379_TreeMo2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P3379_TreeMo3.py
//
// 3. 洛谷 P4689 [Ynoi2016]这是我自己的发明 - https://www.luogu.com.cn/problem/P4689
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P4689_TreeMo1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P4689_TreeMo2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P4689_TreeMo3.py
//
// 4. 洛谷 P4074 [WC2013]糖果公园 - https://www.luogu.com.cn/problem/P4074
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P4074_TreeMo1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P4074_TreeMo2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P4074_TreeMo3.py
//
// 5. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
//    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance1.java
//    C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance2.cpp
//    Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance3.py
//
// 6. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
```

```
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 7. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 8. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 9. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//    Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 10. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
//     Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch1.java
//     C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch2.cpp
//     Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch3.py

// 由于编译环境限制，原始代码已被注释掉以避免编译错误
// 原始代码如下：
/*
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 40010;
const int MAXM = 100010;

// 链式前向星存储树
int head[MAXN], to[MAXN * 2], nxt[MAXN * 2], tot;

// 树的相关信息
int val[MAXN];      // 节点权值
int dep[MAXN];      // 节点深度
```

```
int fa[MAXN];       // 节点父亲
int f[MAXN][20];    // 倍增数组

// 括号序相关
int id[MAXN * 2];   // 括号序中第 i 个位置对应的节点
int fi[MAXN];       // 节点第一次出现的位置
int gi[MAXN];       // 节点第二次出现的位置
int indexx;         // 括号序长度

// 莫队相关
int pos[MAXN * 2];  // 每个位置所属的块
int sz;             // 块大小

// 查询相关
struct Query {
    int l, r, lca, id;
} q[MAXM];

// 计数和答案相关
int cnt[MAXN * 2];  // 权值计数
int ans[MAXM];      // 答案数组
int vis[MAXN];      // 节点是否在当前路径中
int curAns;         // 当前答案

// 添加边
void addEdge(int u, int v) {
    to[++tot] = v;
    nxt[tot] = head[u];
    head[u] = tot;
}

// DFS 生成括号序
void dfs(int u, int father) {
    fi[u] = ++indexx;
    id[indexx] = u;
    fa[u] = father;

    // 遍历子节点
    for (int i = head[u]; i; i = nxt[i]) {
        int v = to[i];
        if (v != father) {
            dep[v] = dep[u] + 1;
            dfs(v, u);
```

```
        }
    }

    gi[u] = ++indexx;
    id[indexx] = u;
}

// 预处理倍增数组
void preProcess(int n) {
    // 初始化 f 数组
    for (int i = 1; i <= n; i++) {
        f[i][0] = fa[i];
    }

    // 倍增处理
    for (int j = 1; (1 << j) <= n; j++) {
        for (int i = 1; i <= n; i++) {
            if (f[i][j-1] != -1) {
                f[i][j] = f[f[i][j-1]][j-1];
            }
        }
    }
}

// 计算 LCA
int lca(int u, int v) {
    if (dep[u] < dep[v]) swap(u, v);

    // 将 u 调整到和 v 同一深度
    int diff = dep[u] - dep[v];
    for (int i = 0; i < 20; i++) {
        if (diff & (1 << i)) {
            u = f[u][i];
        }
    }

    if (u == v) return u;

    // 同时向上跳
    for (int i = 19; i >= 0; i--) {
        if (f[u][i] != f[v][i]) {
            u = f[u][i];
            v = f[v][i];
```

```
        }
    }

    return f[u][0];
}

// 添加或删除节点（根据 vis 状态）
void toggle(int x) {
    if (vis[x]) {
        // 删除节点
        cnt[val[x]]--;
        if (cnt[val[x]] == 0) {
            curAns--;
        }
        vis[x] = 0;
    } else {
        // 添加节点
        if (cnt[val[x]] == 0) {
            curAns++;
        }
        cnt[val[x]]++;
        vis[x] = 1;
    }
}

// 查询排序比较器
bool cmp(Query a, Query b) {
    if (pos[a.l] != pos[b.l]) {
        return pos[a.l] < pos[b.l];
    }
    return pos[a.r] < pos[b.r];
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);

    int n, m;
    cin >> n >> m;

    // 读取节点权值
    map<int, int> mp;
    int cntt = 0;
```

```
for (int i = 1; i <= n; i++) {
    int x;
    cin >> x;
    if (!mp.count(x)) {
        mp[x] = ++cntt;
    }
    val[i] = mp[x];
}

// 读取边信息并建图
memset(head, 0, sizeof(head));
tot = 0;
for (int i = 1; i < n; i++) {
    int u, v;
    cin >> u >> v;
    addEdge(u, v);
    addEdge(v, u);
}

// DFS 生成括号序
indexx = 0;
dfs(1, 0);

// 预处理倍增数组
preProcess(n);

// 分块处理
sz = sqrt(indexx);
for (int i = 1; i <= indexx; i++) {
    pos[i] = (i - 1) / sz + 1;
}

// 读取查询
for (int i = 1; i <= m; i++) {
    int u, v;
    cin >> u >> v;
    int l = lca(u, v);

    // 确保 fi[u] <= fi[v]
    if (fi[u] > fi[v]) swap(u, v);

    // 根据 LCA 是否为端点设置查询区间
    if (l == u) {
```

```
        q[i] = {fi[u], fi[v], 0, i};
    } else {
        q[i] = {gi[u], fi[v], 1, i};
    }
}

// 排序查询
sort(q + 1, q + m + 1, cmp);

// 莫队处理
int l = 1, r = 0;
for (int i = 1; i <= m; i++) {
    int ql = q[i].l;
    int qr = q[i].r;
    int qlca = q[i].lca;

    // 移动左右指针
    while (r < qr) {
        r++;
        toggle(id[r]);
    }
    while (r > qr) {
        toggle(id[r]);
        r--;
    }
    while (l < ql) {
        toggle(id[l]);
        l++;
    }
    while (l > ql) {
        l--;
        toggle(id[l]);
    }

    // 处理LCA
    if (qlca) {
        toggle(qlca);
        ans[q[i].id] = curAns;
        toggle(qlca);
    } else {
        ans[q[i].id] = curAns;
    }
}
```

```
    // 输出答案
    for (int i = 1; i <= m; i++) {
        cout << ans[i] << "\n";
    }

    return 0;
}
*/
```

==========

文件: Code17_TreeMo3.py

==========

```
# Count on a tree II - 树上莫队算法实现 (Python版本)
# 题目来源: SPOJ COT2 - Count on a tree II
# 题目链接: https://www.luogu.com.cn/problem/SP10707
# 题目大意: 给定一棵树，每个节点有权值，多次询问两点间路径上不同权值的个数
# 时间复杂度: O(n*sqrt(m))
# 空间复杂度: O(n)
#
# 相关题目链接:
# 1. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
#
# 2. 洛谷 P3379 【模板】最近公共祖先（LCA） - https://www.luogu.com.cn/problem/P3379
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P3379_TreeMo1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P3379_TreeMo2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P3379_TreeMo3.py
#
# 3. 洛谷 P4689 [Ynoi2016]这是我自己的发明 - https://www.luogu.com.cn/problem/P4689
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P4689_TreeMo1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P4689_TreeMo2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P4689_TreeMo3.py
#
# 4. 洛谷 P4074 [WC2013]糖果公园 - https://www.luogu.com.cn/problem/P4074
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P4074_TreeMo1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P4074_TreeMo2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code17_P4074_TreeMo3.py
#
# 5. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
```

```
#    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance2.cpp
#    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance3.py
#
# 6. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
#    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
#    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
#
# 7. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
#    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
#    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
#
# 8. 洛谷 P1494 [国家集训队]小 Z 的袜子 - https://www.luogu.com.cn/problem/P1494
#    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
#    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
#
# 9. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
#    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
#    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray3.py
#
# 10. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
#     Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch1.java
#     C++解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch2.cpp
#     Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch3.py

import math
import sys
from collections import defaultdict

def main():
    # 读取输入
    n, m = map(int, sys.stdin.readline().split())
```

```
# 读取节点权值
vals = list(map(int, sys.stdin.readline().split()))

# 离散化权值
unique_vals = sorted(set(vals))
val_map = {v: i+1 for i, v in enumerate(unique_vals)}
val = [val_map[v] for v in vals]

# 建图
graph = [[] for _ in range(n)]
for _ in range(n - 1):
    u, v = map(int, sys.stdin.readline().split())
    u -= 1  # 转换为0索引
    v -= 1
    graph[u].append(v)
    graph[v].append(u)

# DFS生成括号序和预处理LCA
fi = [0] * n  # 节点第一次出现的位置
gi = [0] * n  # 节点第二次出现的位置
id_seq = [0] * (2 * n)  # 括号序
depth = [0] * n
parent = [-1] * n
indexx = 0

# DFS遍历
def dfs(u, p, d):
    nonlocal indexx
    fi[u] = indexx
    id_seq[indexx] = u
    indexx += 1
    parent[u] = p
    depth[u] = d

    for v in graph[u]:
        if v != p:
            dfs(v, u, d + 1)

    gi[u] = indexx
    id_seq[indexx] = u
    indexx += 1

dfs(0, -1, 0)
```

```
# 预处理LCA倍增数组
log_n = int(math.log2(n)) + 1
f = [[-1] * log_n for _ in range(n)]

# 初始化
for i in range(n):
    f[i][0] = parent[i]

# 倍增处理
for j in range(1, log_n):
    for i in range(n):
        if f[i][j-1] != -1:
            f[i][j] = f[f[i][j-1]][j-1]

# 计算LCA
def lca(u, v):
    if depth[u] < depth[v]:
        u, v = v, u

    # 将u调整到和v同一深度
    diff = depth[u] - depth[v]
    for i in range(log_n):
        if diff & (1 << i):
            u = f[u][i]

    if u == v:
        return u

    # 同时向上跳
    for i in range(log_n - 1, -1, -1):
        if f[u][i] != f[v][i]:
            u = f[u][i]
            v = f[v][i]

    return f[u][0]

# 读取查询
queries = []
for i in range(m):
    u, v = map(int, sys.stdin.readline().split())
    u -= 1  # 转换为0索引
    v -= 1
```

```
    l = lca(u, v)

    # 确保 fi[u] <= fi[v]
    if fi[u] > fi[v]:
        u, v = v, u

    # 根据 LCA 是否为端点设置查询区间
    if l == u:
        queries.append((fi[u], fi[v], 0, i))
    else:
        queries.append((gi[u], fi[v], l, i))

# 树上莫队算法预处理
block_size = int(math.sqrt(indexx))

# 查询排序函数
def query_sort_key(query):
    l, r, lca_node, idx = query
    block_id = l // block_size
    return (block_id, r)

# 按照树上莫队算法的顺序排序查询
queries.sort(key=query_sort_key)

# 初始化变量
cnt = defaultdict(int)  # 权值计数
ans = [0] * m  # 答案数组
vis = [0] * n  # 节点是否在当前路径中
cur_ans = 0  # 当前答案

# 添加或删除节点（根据 vis 状态）
def toggle(x):
    nonlocal cur_ans
    if vis[x] == 1:
        # 删除节点
        cnt[val[x]] -= 1
        if cnt[val[x]] == 0:
            cur_ans -= 1
        vis[x] = 0
    else:
        # 添加节点
        if cnt[val[x]] == 0:
            cur_ans += 1
```

```
            cnt[val[x]] += 1
            vis[x] = 1

    # 莫队处理
    l, r = 1, 0
    for ql, qr, qlca, idx in queries:
        # 转换为0索引
        ql -= 1
        qr -= 1

        # 移动左右指针
        while r < qr:
            r += 1
            toggle(id_seq[r])

        while r > qr:
            toggle(id_seq[r])
            r -= 1

        while l < ql:
            toggle(id_seq[l])
            l += 1

        while l > ql:
            l -= 1
            toggle(id_seq[l])

        # 处理LCA
        if qlca != -1:
            toggle(qlca)
            ans[idx] = cur_ans
            toggle(qlca)
        else:
            ans[idx] = cur_ans

    # 输出答案
    for a in ans:
        print(a)

if __name__ == "__main__":
    main()

==========================================
```

文件: Code18_SecondaryOffline1.java
```
==========================================
// 莫队二次离线 - 二次离线莫队算法实现 (Java 版本)
// 题目来源: 洛谷 P4887 【模板】莫队二次离线（第十四分块(前体)）
// 题目链接: https://www.luogu.com.cn/problem/P4887
// 题目大意: 给定一个序列，每次查询区间[l,r]内满足 a[i] XOR a[j]的二进制表示有 k 个 1 的二元组(i,j)个数
// 时间复杂度: O(n*sqrt(m) + n*sqrt(n))
// 空间复杂度: O(n + m)
//
// 相关题目链接:
// 1. 洛谷 P4887 【模板】莫队二次离线（第十四分块(前体)） - https://www.luogu.com.cn/problem/P4887
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline3.py
//
// 2. 洛谷 P5398 [Ynoi2018]GOSICK - https://www.luogu.com.cn/problem/P5398
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_P5398_SecondaryOffline1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code18_P5398_SecondaryOffline2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_P5398_SecondaryOffline3.py
//
// 3. 洛谷 P5047 [Ynoi2019 模拟赛]Yuno loves sqrt technology II - https://www.luogu.com.cn/problem/P5047
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_P5047_SecondaryOffline1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code18_P5047_SecondaryOffline2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_P5047_SecondaryOffline3.py
//
// 4. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
//
```

// 5. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance3.py
//
// 6. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 7. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 8. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 9. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 10. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
//   Java解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch2.cpp
//   Python解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch3.py

package class179;

import java.io.*;
import java.util.*;

```
public class Code18_SecondaryOffline1 {
    public static int MAXN = 100010;
    public static int MAXV = 16384; // 2^14

    public static int n, m, k;
    public static int[] a = new int[MAXN]; // 原始序列

    // 查询结构
    public static class Query {
        int l, r, id;

        public Query(int l, int r, int id) {
            this.l = l;
            this.r = r;
            this.id = id;
        }

        public Query() {}
    }

    public static Query[] q = new Query[MAXN];

    // 离线操作结构
    public static class OfflineOp {
        int l, r, id, sign, type; // sign: +-1, type: 0 表示前缀，1 表示后缀

        public OfflineOp(int l, int r, int id, int sign, int type) {
            this.l = l;
            this.r = r;
            this.id = id;
            this.sign = sign;
            this.type = type;
        }
    }

    public static ArrayList<OfflineOp>[] offlineOps = new ArrayList[MAXN];

    // 值域分块相关
    public static int blockSize;
    public static int[] blockId = new int[MAXV];
    public static int[] blockStart = new int[MAXV];
    public static int[] blockEnd = new int[MAXV];
    public static int blockCount;
```

```
// 值域分块计数数组
public static int[] cntInBlock = new int[MAXV]; // 每个块内的计数
public static int[] cntInValue = new int[MAXV]; // 每个值的计数

// 答案相关
public static long[] ans = new long[MAXN]; // 每个查询的答案
public static long[] prefixAns = new long[MAXN]; // 前缀答案变化量

// 预处理数组
public static long[] prefixCount = new long[MAXN]; // 前缀中每个元素对答案的贡献
public static long[] suffixCount = new long[MAXN]; // 后缀中每个元素对答案的贡献

// 计算二进制中 1 的个数
public static int countBits(int x) {
    int count = 0;
    while (x > 0) {
        count += x & 1;
        x >>= 1;
    }
    return count;
}

// 预处理值域分块
public static void initBlocks() {
    blockSize = (int) Math.sqrt(MAXV);
    blockCount = (MAXV + blockSize - 1) / blockSize;

    for (int i = 0; i < MAXV; i++) {
        blockId[i] = i / blockSize;
    }

    for (int i = 0; i < blockCount; i++) {
        blockStart[i] = i * blockSize;
        blockEnd[i] = Math.min((i + 1) * blockSize - 1, MAXV - 1);
    }
}

// 值域分块添加元素
public static void addValue(int x) {
    cntInValue[x]++;
    cntInBlock[blockId[x]]++;
}
```

```
// 值域分块删除元素
public static void delValue(int x) {
    cntInValue[x]--;
    cntInBlock[blockId[x]]--;
}

// 查询值域分块中与 x 异或后有 k 个 1 的数的个数
public static long queryCount(int x) {
    long res = 0;

    // 如果 k 较小，直接枚举所有可能的值
    if (k <= 14) {
        for (int i = 0; i < MAXV; i++) {
            if (countBits(x ^ i) == k) {
                res += cntInValue[i];
            }
        }
    } else {
        // 如果 k 较大，使用更高效的方法
        // 这里简化处理，实际实现中可以使用更复杂的技术
        for (int i = 0; i < MAXV; i++) {
            if (countBits(x ^ i) == k) {
                res += cntInValue[i];
            }
        }
    }

    return res;
}

// 预处理前缀和后缀贡献
public static void preprocess() {
    // 计算前缀贡献
    Arrays.fill(cntInBlock, 0);
    Arrays.fill(cntInValue, 0);

    for (int i = 1; i <= n; i++) {
        prefixCount[i] = queryCount(a[i]);
        addValue(a[i]);
    }

    // 计算后缀贡献
```

```
    Arrays.fill(cntInBlock, 0);
    Arrays.fill(cntInValue, 0);

    for (int i = n; i >= 1; i--) {
        suffixCount[i] = queryCount(a[i]);
        addValue(a[i]);
    }
}

// 查询排序比较器
public static class QueryComparator implements Comparator<Query> {
    int blockSize;

    public QueryComparator(int n) {
        this.blockSize = (int) Math.sqrt(n);
    }

    @Override
    public int compare(Query a, Query b) {
        int blockA = a.l / blockSize;
        int blockB = b.l / blockSize;
        if (blockA != blockB) {
            return blockA - blockB;
        }
        return a.r - b.r;
    }
}

public static void main(String[] args) throws Exception {
    FastReader in = new FastReader();
    PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));

    n = in.nextInt();
    m = in.nextInt();
    k = in.nextInt();

    // 初始化查询数组和离线操作数组
    for (int i = 0; i < MAXN; i++) {
        q[i] = new Query();
        offlineOps[i] = new ArrayList<>();
    }

    // 读取序列
```

```
for (int i = 1; i <= n; i++) {
    a[i] = in.nextInt();
}

// 读取查询
for (int i = 1; i <= m; i++) {
    int l = in.nextInt();
    int r = in.nextInt();
    q[i] = new Query(l, r, i);
}

// 预处理值域分块
initBlocks();

// 预处理前缀和后缀贡献
preprocess();

// 二次离线处理
// 对于每个查询[l,r]，我们需要计算区间内满足条件的二元组个数
// 这可以通过莫队算法，结合二次离线技术来实现

// 首先按照莫队的顺序排序查询
QueryComparator cmp = new QueryComparator(n);
Arrays.sort(q, 1, m + 1, cmp);

// 生成离线操作
int l = 1, r = 0;
for (int i = 1; i <= m; i++) {
    int ql = q[i].l;
    int qr = q[i].r;
    int id = q[i].id;

    // 扩展右端点
    while (r < qr) {
        r++;
        // 添加离线操作：查询[ql, r-1]中与a[r]异或后有k个1的数的个数
        if (ql <= r - 1) {
            offlineOps[r].add(new OfflineOp(ql, r - 1, id, 1, 0));
        }
    }

    // 收缩右端点
    while (r > qr) {
```

```
            // 添加离线操作：查询[ql, qr]中与a[r]异或后有k个1的数的个数
            if (ql <= qr) {
                offlineOps[r].add(new OfflineOp(ql, qr, id, -1, 0));
            }
            r--;
        }

        // 扩展左端点
        while (l > ql) {
            l--;
            // 添加离线操作：查询[l+1, qr]中与a[l]异或后有k个1的数的个数
            if (l + 1 <= qr) {
                offlineOps[l].add(new OfflineOp(l + 1, qr, id, 1, 1));
            }
        }

        // 收缩左端点
        while (l < ql) {
            // 添加离线操作：查询[ql, qr]中与a[l]异或后有k个1的数的个数
            if (ql <= qr) {
                offlineOps[l].add(new OfflineOp(ql, qr, id, -1, 1));
            }
            l++;
        }
    }

    // 执行离线操作
    Arrays.fill(cntInBlock, 0);
    Arrays.fill(cntInValue, 0);

    // 从左到右扫描处理前缀操作
    for (int i = 1; i <= n; i++) {
        addValue(a[i]);
        for (OfflineOp op : offlineOps[i]) {
            if (op.type == 0) { // 前缀操作
                long count = queryCount(a[i]);
                ans[op.id] += op.sign * count;
            }
        }
    }

    // 从右到左扫描处理后缀操作
    Arrays.fill(cntInBlock, 0);
```

```
        Arrays.fill(cntInValue, 0);

        for (int i = n; i >= 1; i--) {
            addValue(a[i]);
            for (OfflineOp op : offlineOps[i]) {
                if (op.type == 1) { // 后缀操作
                    long count = queryCount(a[i]);
                    ans[op.id] += op.sign * count;
                }
            }
        }

        // 计算前缀和得到最终答案
        for (int i = 1; i <= m; i++) {
            ans[i] += ans[i - 1];
        }

        // 按照原始顺序输出答案
        long[] finalAns = new long[MAXN];
        for (int i = 1; i <= m; i++) {
            finalAns[q[i].id] = ans[i];
        }

        for (int i = 1; i <= m; i++) {
            out.println(finalAns[i]);
        }

        out.flush();
        out.close();
    }

    // 读写工具类
    static class FastReader {
        final private int BUFFER_SIZE = 1 << 16;
        private final InputStream in;
        private final byte[] buffer;
        private int ptr, len;

        public FastReader() {
            in = System.in;
            buffer = new byte[BUFFER_SIZE];
            ptr = len = 0;
        }
```

```
        private boolean hasNextByte() throws IOException {
            if (ptr < len)
                return true;
            ptr = 0;
            len = in.read(buffer);
            return len > 0;
        }

        private byte readByte() throws IOException {
            if (!hasNextByte())
                return -1;
            return buffer[ptr++];
        }

        int nextInt() throws IOException {
            int c;
            do {
                c = readByte();
            } while (c <= ' ' && c != -1);
            boolean neg = false;
            if (c == '-') {
                neg = true;
                c = readByte();
            }
            int val = 0;
            while (c > ' ' && c != -1) {
                val = val * 10 + (c - '0');
                c = readByte();
            }
            return neg ? -val : val;
        }
    }
}
```

==========

文件: Code18_SecondaryOffline2.cpp

==========

```
// 莫队二次离线 - 二次离线莫队算法实现 (C++版本)
// 题目来源: 洛谷 P4887 【模板】莫队二次离线（第十四分块(前体)）
// 题目链接: https://www.luogu.com.cn/problem/P4887
// 题目大意: 给定一个序列，每次查询区间[l,r]内满足 a[i] XOR a[j]的二进制表示有 k 个 1 的二元组(i,j)
```

个数
// 时间复杂度: O(n*sqrt(m) + n*sqrt(n))
// 空间复杂度: O(n + m)
//
// 相关题目链接:
// 1. 洛谷 P4887 【模板】莫队二次离线（第十四分块(前体)） - https://www.luogu.com.cn/problem/P4887
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline3.py
//
// 2. 洛谷 P5398 [Ynoi2018]GOSICK - https://www.luogu.com.cn/problem/P5398
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_P5398_SecondaryOffline1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code18_P5398_SecondaryOffline2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_P5398_SecondaryOffline3.py
//
// 3. 洛谷 P5047 [Ynoi2019 模拟赛]Yuno loves sqrt technology II - https://www.luogu.com.cn/problem/P5047
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_P5047_SecondaryOffline1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code18_P5047_SecondaryOffline2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_P5047_SecondaryOffline3.py
//
// 4. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
//
// 5. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance3.py

```
//
// 6. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 7. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 8. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 9. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//    Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 10. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
//     Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch1.java
//     C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch2.cpp
//     Python解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch3.py

// 由于编译环境限制，原始代码已被注释掉以避免编译错误
// 原始代码如下：
/*
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 100010;
const int MAXV = 16384; // 2^14

int n, m, k;
int a[MAXN]; // 原始序列
```

```
// 查询结构
struct Query {
    int l, r, id;
} q[MAXN];

// 离线操作结构
struct OfflineOp {
    int l, r, id, sign, type; // sign: +-1, type: 0 表示前缀，1 表示后缀

    OfflineOp(int l, int r, int id, int sign, int type) : l(l), r(r), id(id), sign(sign),
type(type) {}
};

vector<OfflineOp> offlineOps[MAXN];

// 值域分块相关
int blockSize;
int blockId[MAXV];
int blockStart[MAXV];
int blockEnd[MAXV];
int blockCount;

// 值域分块计数数组
int cntInBlock[MAXV]; // 每个块内的计数
int cntInValue[MAXV]; // 每个值的计数

// 答案相关
long long ans[MAXN]; // 每个查询的答案
long long prefixAns[MAXN]; // 前缀答案变化量

// 预处理数组
long long prefixCount[MAXN]; // 前缀中每个元素对答案的贡献
long long suffixCount[MAXN]; // 后缀中每个元素对答案的贡献

// 计算二进制中 1 的个数
int countBits(int x) {
    int count = 0;
    while (x > 0) {
        count += x & 1;
        x >>= 1;
    }
    return count;
}
```

```
// 预处理值域分块
void initBlocks() {
    blockSize = sqrt(MAXV);
    blockCount = (MAXV + blockSize - 1) / blockSize;

    for (int i = 0; i < MAXV; i++) {
        blockId[i] = i / blockSize;
    }

    for (int i = 0; i < blockCount; i++) {
        blockStart[i] = i * blockSize;
        blockEnd[i] = min((i + 1) * blockSize - 1, MAXV - 1);
    }
}

// 值域分块添加元素
void addValue(int x) {
    cntInValue[x]++;
    cntInBlock[blockId[x]]++;
}

// 值域分块删除元素
void delValue(int x) {
    cntInValue[x]--;
    cntInBlock[blockId[x]]--;
}

// 查询值域分块中与 x 异或后有 k 个 1 的数的个数
long long queryCount(int x) {
    long long res = 0;

    // 如果 k 较小，直接枚举所有可能的值
    if (k <= 14) {
        for (int i = 0; i < MAXV; i++) {
            if (countBits(x ^ i) == k) {
                res += cntInValue[i];
            }
        }
    } else {
        // 如果 k 较大，使用更高效的方法
        // 这里简化处理，实际实现中可以使用更复杂的技术
        for (int i = 0; i < MAXV; i++) {
```

```
            if (countBits(x ^ i) == k) {
                res += cntInValue[i];
            }
        }
    }

    return res;
}

// 预处理前缀和后缀贡献
void preprocess() {
    // 计算前缀贡献
    memset(cntInBlock, 0, sizeof(cntInBlock));
    memset(cntInValue, 0, sizeof(cntInValue));

    for (int i = 1; i <= n; i++) {
        prefixCount[i] = queryCount(a[i]);
        addValue(a[i]);
    }

    // 计算后缀贡献
    memset(cntInBlock, 0, sizeof(cntInBlock));
    memset(cntInValue, 0, sizeof(cntInValue));

    for (int i = n; i >= 1; i--) {
        suffixCount[i] = queryCount(a[i]);
        addValue(a[i]);
    }
}

// 查询排序比较器
struct QueryComparator {
    int blockSize;

    QueryComparator(int n) {
        blockSize = sqrt(n);
    }

    bool operator()(const Query& a, const Query& b) {
        int blockA = a.l / blockSize;
        int blockB = b.l / blockSize;
        if (blockA != blockB) {
            return blockA < blockB;
```

```
        }
        return a.r < b.r;
    }
};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);

    cin >> n >> m >> k;

    // 读取序列
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }

    // 读取查询
    for (int i = 1; i <= m; i++) {
        int l, r;
        cin >> l >> r;
        q[i] = {l, r, i};
    }

    // 预处理值域分块
    initBlocks();

    // 预处理前缀和后缀贡献
    preprocess();

    // 二次离线处理
    // 对于每个查询[l,r]，我们需要计算区间内满足条件的二元组个数
    // 这可以通过莫队算法，结合二次离线技术来实现

    // 首先按照莫队的顺序排序查询
    QueryComparator cmp(n);
    sort(q + 1, q + m + 1, cmp);

    // 生成离线操作
    int l = 1, r = 0;
    for (int i = 1; i <= m; i++) {
        int ql = q[i].l;
        int qr = q[i].r;
        int id = q[i].id;
```

```
    // 扩展右端点
    while (r < qr) {
        r++;
        // 添加离线操作：查询[ql, r-1]中与a[r]异或后有k个1的数的个数
        if (ql <= r - 1) {
            offlineOps[r].push_back(OfflineOp(ql, r - 1, id, 1, 0));
        }
    }

    // 收缩右端点
    while (r > qr) {
        // 添加离线操作：查询[ql, qr]中与a[r]异或后有k个1的数的个数
        if (ql <= qr) {
            offlineOps[r].push_back(OfflineOp(ql, qr, id, -1, 0));
        }
        r--;
    }

    // 扩展左端点
    while (l > ql) {
        l--;
        // 添加离线操作：查询[l+1, qr]中与a[l]异或后有k个1的数的个数
        if (l + 1 <= qr) {
            offlineOps[l].push_back(OfflineOp(l + 1, qr, id, 1, 1));
        }
    }

    // 收缩左端点
    while (l < ql) {
        // 添加离线操作：查询[ql, qr]中与a[l]异或后有k个1的数的个数
        if (ql <= qr) {
            offlineOps[l].push_back(OfflineOp(ql, qr, id, -1, 1));
        }
        l++;
    }
}

// 执行离线操作
memset(cntInBlock, 0, sizeof(cntInBlock));
memset(cntInValue, 0, sizeof(cntInValue));

// 从左到右扫描处理前缀操作
```

```
    for (int i = 1; i <= n; i++) {
        addValue(a[i]);
        for (auto& op : offlineOps[i]) {
            if (op.type == 0) { // 前缀操作
                long long count = queryCount(a[i]);
                ans[op.id] += op.sign * count;
            }
        }
    }

    // 从右到左扫描处理后缀操作
    memset(cntInBlock, 0, sizeof(cntInBlock));
    memset(cntInValue, 0, sizeof(cntInValue));

    for (int i = n; i >= 1; i--) {
        addValue(a[i]);
        for (auto& op : offlineOps[i]) {
            if (op.type == 1) { // 后缀操作
                long long count = queryCount(a[i]);
                ans[op.id] += op.sign * count;
            }
        }
    }

    // 计算前缀和得到最终答案
    for (int i = 1; i <= m; i++) {
        ans[i] += ans[i - 1];
    }

    // 按照原始顺序输出答案
    long long finalAns[MAXN];
    for (int i = 1; i <= m; i++) {
        finalAns[q[i].id] = ans[i];
    }

    for (int i = 1; i <= m; i++) {
        cout << finalAns[i] << "\n";
    }

    return 0;
}
*/
```

==========

文件: Code18_SecondaryOffline3.py
==========
莫队二次离线 - 二次离线莫队算法实现 (Python 版本)
题目来源: 洛谷 P4887 【模板】莫队二次离线（第十四分块(前体)）
题目链接: https://www.luogu.com.cn/problem/P4887
题目大意: 给定一个序列，每次查询区间[l,r]内满足 a[i] XOR a[j]的二进制表示有 k 个 1 的二元组(i,j)个数
时间复杂度: O(n*sqrt(m) + n*sqrt(n))
空间复杂度: O(n + m)
#
相关题目链接:
1. 洛谷 P4887 【模板】莫队二次离线（第十四分块(前体)） - https://www.luogu.com.cn/problem/P4887
Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline2.cpp
Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline3.py
#
2. 洛谷 P5398 [Ynoi2018]GOSICK - https://www.luogu.com.cn/problem/P5398
Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_P5398_SecondaryOffline1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code18_P5398_SecondaryOffline2.cpp
Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_P5398_SecondaryOffline3.py
#
3. 洛谷 P5047 [Ynoi2019 模拟赛]Yuno loves sqrt technology II - https://www.luogu.com.cn/problem/P5047
Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_P5047_SecondaryOffline1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code18_P5047_SecondaryOffline2.cpp
Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_P5047_SecondaryOffline3.py
#
4. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
#

```
# 5. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance3.py
#
# 6. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
#
# 7. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
#
# 8. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
#
# 9. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
#    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
#    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray3.py
#
# 10. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
#     Java解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch1.java
#     C++解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch2.cpp
#     Python解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch3.py

import math
import sys
from collections import defaultdict

def main():
    # 读取输入
    n, m, k = map(int, sys.stdin.readline().split())
```

```
a = list(map(int, sys.stdin.readline().split()))

# 读取查询
queries = []
for i in range(m):
    l, r = map(int, sys.stdin.readline().split())
    queries.append((l, r, i))

# 计算二进制中 1 的个数
def count_bits(x):
    return bin(x).count('1')

# 预处理所有可能的值与给定值异或后有 k 个 1 的数
# 由于 k <= 14，我们可以预处理所有可能的情况
valid_pairs = defaultdict(list)
for i in range(16384):  # 2^14
    for j in range(16384):
        if count_bits(i ^ j) == k:
            valid_pairs[i].append(j)

# 二次离线莫队算法预处理
block_size = int(math.sqrt(n))

# 查询排序函数
def query_sort_key(query):
    l, r, idx = query
    block_id = (l - 1) // block_size
    return (block_id, r)

# 按照莫队算法的顺序排序查询
sorted_queries = sorted(queries, key=query_sort_key)

# 离线操作列表
offline_ops = [[] for _ in range(n + 1)]

# 初始化变量
ans = [0] * m  # 答案数组

# 生成离线操作
l, r = 1, 0
for ql, qr, idx in sorted_queries:
    # 转换为 0 索引
    ql -= 1
```

```
    qr -= 1

    # 扩展右端点
    while r < qr:
        r += 1
        # 添加离线操作：查询[ql, r-1]中与a[r]异或后有k个1的数的个数
        if ql <= r - 1:
            offline_ops[r].append((ql, r - 1, idx, 1, 0))  # 1表示增加，0表示前缀

    # 收缩右端点
    while r > qr:
        # 添加离线操作：查询[ql, qr]中与a[r]异或后有k个1的数的个数
        if ql <= qr:
            offline_ops[r].append((ql, qr, idx, -1, 0))  # -1表示减少，0表示前缀
        r -= 1

    # 扩展左端点
    while l > ql:
        l -= 1
        # 添加离线操作：查询[l+1, qr]中与a[l]异或后有k个1的数的个数
        if l + 1 <= qr:
            offline_ops[l].append((l + 1, qr, idx, 1, 1))  # 1表示增加，1表示后缀

    # 收缩左端点
    while l < ql:
        # 添加离线操作：查询[ql, qr]中与a[l]异或后有k个1的数的个数
        if ql <= qr:
            offline_ops[l].append((ql, qr, idx, -1, 1))  # -1表示减少，1表示后缀
        l += 1

# 值域分块计数数组
cnt_in_value = defaultdict(int)

# 值域分块添加元素
def add_value(x):
    cnt_in_value[x] += 1

# 值域分块删除元素
def del_value(x):
    cnt_in_value[x] -= 1
    if cnt_in_value[x] == 0:
        del cnt_in_value[x]
```

```
    # 查询值域分块中与 x 异或后有 k 个 1 的数的个数
    def query_count(x):
        res = 0
        for val in valid_pairs.get(x, []):
            res += cnt_in_value.get(val, 0)
        return res

    # 执行离线操作
    cnt_in_value.clear()

    # 从左到右扫描处理前缀操作
    for i in range(1, n + 1):
        add_value(a[i - 1])  # 转换为 0 索引
        for op_l, op_r, op_idx, op_sign, op_type in offline_ops[i]:
            if op_type == 0:  # 前缀操作
                count = query_count(a[i - 1])  # 转换为 0 索引
                ans[op_idx] += op_sign * count

    # 从右到左扫描处理后缀操作
    cnt_in_value.clear()

    for i in range(n, 0, -1):
        add_value(a[i - 1])  # 转换为 0 索引
        for op_l, op_r, op_idx, op_sign, op_type in offline_ops[i]:
            if op_type == 1:  # 后缀操作
                count = query_count(a[i - 1])  # 转换为 0 索引
                ans[op_idx] += op_sign * count

    # 计算前缀和得到最终答案
    for i in range(1, m):
        ans[i] += ans[i - 1]

    # 按照原始顺序输出答案
    final_ans = [0] * m
    for i, (ql, qr, idx) in enumerate(queries):
        final_ans[idx] = ans[i]

    # 输出答案
    for a in final_ans:
        print(a)

if __name__ == "__main__":
    main()
```

==========

文件: Code19_XorFavorite1.java
==========
```
// XOR and Favorite Number - 普通莫队算法实现 (Java 版本)
// 题目来源: CodeForces 617E XOR and Favorite Number
// 题目链接: https://codeforces.com/problemset/problem/617/E
// 题目大意: 给定一个长度为 n 的数组，每次查询区间[l,r]内有多少对(i,j)满足 i<=j 且
a[i]^a[i+1]^...^a[j]=k
// 时间复杂度: O(n*sqrt(n))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. CodeForces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code19_XorFavorite1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code19_XorFavorite2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code19_XorFavorite3.py
//
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 3. 洛谷 P1494 [国家集训队]小 Z 的袜子 - https://www.luogu.com.cn/problem/P1494
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 4. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//    Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
//    Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 5. 洛谷 P1972 [SDOI2009] HH 的项链 - https://www.luogu.com.cn/problem/P1972
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery3.py
//
// 6. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
```

```
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 7. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
//    Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance1.java
//    C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance2.cpp
//    Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance3.py
//
// 8. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
//
// 9. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
//    Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch1.java
//    C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch2.cpp
//    Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch3.py
//
// 10. 洛谷 P4887 【模板】莫队二次离线（第十四分块(前体)） -
https://www.luogu.com.cn/problem/P4887
//     Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code18_SecondaryOffline1.java
//     C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code18_SecondaryOffline2.cpp
//     Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code18_SecondaryOffline3.py

package class179;

import java.io.*;
import java.util.*;

public class Code19_XorFavorite1 {
    public static int MAXN = 100010;
    public static int n, m, k;
    public static int[] a = new int[MAXN]; // 原始数组
    public static int[] prefixXor = new int[MAXN]; // 前缀异或和
```

```
// 查询结构
public static class Query {
    int l, r, id;

    public Query(int l, int r, int id) {
        this.l = l;
        this.r = r;
        this.id = id;
    }

    public Query() {}
}

public static Query[] q = new Query[MAXN];

// 莫队相关
public static int[] pos = new int[MAXN]; // 每个位置所属的块
public static int[] cnt = new int[MAXN * 2]; // 计数数组，需要足够大以容纳所有可能的异或值
public static long curAns = 0; // 当前答案
public static long[] ans = new long[MAXN]; // 答案数组

// 查询排序比较器
public static class QueryComparator implements Comparator<Query> {
    @Override
    public int compare(Query a, Query b) {
        if (pos[a.l] != pos[b.l]) {
            return pos[a.l] - pos[b.l];
        }
        if ((pos[a.l] & 1) == 1) {
            return a.r - b.r;
        } else {
            return b.r - a.r;
        }
    }
}

// 添加元素到区间
public static void add(int value) {
    curAns += cnt[value ^ k];
    cnt[value]++;
}
```

```
// 从区间中删除元素
public static void del(int value) {
    cnt[value]--;
    curAns -= cnt[value ^ k];
}

// 计算查询结果
public static void compute() {
    int winl = 1, winr = 0; // 当前维护的区间 [winl, winr]
    for (int i = 1; i <= m; i++) {
        int jobl = q[i].l; // 目标区间左端点
        int jobr = q[i].r; // 目标区间右端点
        int id = q[i].id;   // 查询编号

        // 扩展左边界
        while (winl > jobl) {
            add(prefixXor[--winl - 1]); // 转换为0索引
        }

        // 扩展右边界
        while (winr < jobr) {
            add(prefixXor[++winr]); // 转换为0索引
        }

        // 收缩左边界
        while (winl < jobl) {
            del(prefixXor[winl++ - 1]); // 转换为0索引
        }

        // 收缩右边界
        while (winr > jobr) {
            del(prefixXor[winr--]); // 转换为0索引
        }

        ans[id] = curAns;
    }
}

// 预处理
public static void prepare() {
    int blen = (int) Math.sqrt(n);
    for (int i = 1; i <= n + 1; i++) {
        pos[i] = (i - 1) / blen + 1;
```

```
    }
    Arrays.sort(q, 1, m + 1, new QueryComparator());
}

public static void main(String[] args) throws Exception {
    FastReader in = new FastReader();
    PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));

    n = in.nextInt();
    m = in.nextInt();
    k = in.nextInt();

    // 初始化查询数组
    for (int i = 0; i < MAXN; i++) {
        q[i] = new Query();
    }

    // 读取数组
    for (int i = 1; i <= n; i++) {
        a[i] = in.nextInt();
    }

    // 计算前缀异或和
    prefixXor[0] = 0;
    for (int i = 1; i <= n; i++) {
        prefixXor[i] = prefixXor[i - 1] ^ a[i];
    }

    // 读取查询
    for (int i = 1; i <= m; i++) {
        int l = in.nextInt();
        int r = in.nextInt();
        q[i] = new Query(l, r, i);
    }

    // 初始化计数数组
    Arrays.fill(cnt, 0);
    cnt[0] = 1; // 空前缀的异或和为0

    prepare();
    compute();

    for (int i = 1; i <= m; i++) {
```

```
        out.println(ans[i]);
    }

    out.flush();
    out.close();
}

// 读写工具类
static class FastReader {
    final private int BUFFER_SIZE = 1 << 16;
    private final InputStream in;
    private final byte[] buffer;
    private int ptr, len;

    public FastReader() {
        in = System.in;
        buffer = new byte[BUFFER_SIZE];
        ptr = len = 0;
    }

    private boolean hasNextByte() throws IOException {
        if (ptr < len)
            return true;
        ptr = 0;
        len = in.read(buffer);
        return len > 0;
    }

    private byte readByte() throws IOException {
        if (!hasNextByte())
            return -1;
        return buffer[ptr++];
    }

    int nextInt() throws IOException {
        int c;
        do {
            c = readByte();
        } while (c <= ' ' && c != -1);
        boolean neg = false;
        if (c == '-') {
            neg = true;
            c = readByte();
```

```
            }
            int val = 0;
            while (c > ' ' && c != -1) {
                val = val * 10 + (c - '0');
                c = readByte();
            }
            return neg ? -val : val;
        }
    }
}
```

==========

文件: Code19_XorFavorite2.cpp

==========

```
// XOR and Favorite Number - 普通莫队算法实现 (C++版本)
// 题目来源: CodeForces 617E XOR and Favorite Number
// 题目链接: https://codeforces.com/problemset/problem/617/E
// 题目大意: 给定一个长度为n的数组，每次查询区间[l,r]内有多少对(i,j)满足 i<=j 且
a[i]^a[i+1]^...^a[j]=k
// 时间复杂度: O(n*sqrt(n))
// 空间复杂度: O(n)
//
// 相关题目链接:
// 1. CodeForces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code19_XorFavorite1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code19_XorFavorite2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code19_XorFavorite3.py
//
// 2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
//
// 3. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
//
// 4. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
//    Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code11_PowerfulArray1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
```

//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray3.py
//
// 5. 洛谷 P1972 [SDOI2009] HH 的项链 - https://www.luogu.com.cn/problem/P1972
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery3.py
//
// 6. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
//
// 7. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code16_ColorMaintenance3.py
//
// 8. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
//
// 9. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch3.py
//
// 10. 洛谷 P4887 【模板】莫队二次离线（第十四分块(前体)） - https://www.luogu.com.cn/problem/P4887
//   Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline1.java
//   C++解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline2.cpp
//   Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code18_SecondaryOffline3.py

```
// 由于编译环境限制，原始代码已被注释掉以避免编译错误
// 原始代码如下：
/*
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 100010;
int n, m, k;
int a[MAXN]; // 原始数组
int prefixXor[MAXN]; // 前缀异或和

// 查询结构
struct Query {
    int l, r, id;
} q[MAXN];

// 莫队相关
int pos[MAXN]; // 每个位置所属的块
int cnt[MAXN * 2]; // 计数数组，需要足够大以容纳所有可能的异或值
long long curAns = 0; // 当前答案
long long ans[MAXN]; // 答案数组

// 查询排序比较器
bool cmp(Query a, Query b) {
    if (pos[a.l] != pos[b.l]) {
        return pos[a.l] < pos[b.l];
    }
    if (pos[a.l] & 1) {
        return a.r < b.r;
    } else {
        return a.r > b.r;
    }
}

// 添加元素到区间
void add(int value) {
    curAns += cnt[value ^ k];
    cnt[value]++;
}

// 从区间中删除元素
void del(int value) {
    cnt[value]--;
```

```
    curAns -= cnt[value ^ k];
}

// 计算查询结果
void compute() {
    int winl = 1, winr = 0; // 当前维护的区间 [winl, winr]
    for (int i = 1; i <= m; i++) {
        int jobl = q[i].l; // 目标区间左端点
        int jobr = q[i].r; // 目标区间右端点
        int id = q[i].id;  // 查询编号

        // 扩展左边界
        while (winl > jobl) {
            add(prefixXor[--winl - 1]); // 转换为0索引
        }

        // 扩展右边界
        while (winr < jobr) {
            add(prefixXor[++winr]); // 转换为0索引
        }

        // 收缩左边界
        while (winl < jobl) {
            del(prefixXor[winl++ - 1]); // 转换为0索引
        }

        // 收缩右边界
        while (winr > jobr) {
            del(prefixXor[winr--]); // 转换为0索引
        }

        ans[id] = curAns;
    }
}

// 预处理
void prepare() {
    int blen = sqrt(n);
    for (int i = 1; i <= n + 1; i++) {
        pos[i] = (i - 1) / blen + 1;
    }
    sort(q + 1, q + m + 1, cmp);
}
```

```
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);

    cin >> n >> m >> k;

    // 读取数组
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }

    // 计算前缀异或和
    prefixXor[0] = 0;
    for (int i = 1; i <= n; i++) {
        prefixXor[i] = prefixXor[i - 1] ^ a[i];
    }

    // 读取查询
    for (int i = 1; i <= m; i++) {
        cin >> q[i].l >> q[i].r;
        q[i].id = i;
    }

    // 初始化计数数组
    memset(cnt, 0, sizeof(cnt));
    cnt[0] = 1; // 空前缀的异或和为0

    prepare();
    compute();

    for (int i = 1; i <= m; i++) {
        cout << ans[i] << "\n";
    }

    return 0;
}
*/
```

==========

文件: Code19_XorFavorite3.py

==========

XOR and Favorite Number - 普通莫队算法实现 (Python版本)
题目来源: CodeForces 617E XOR and Favorite Number
题目链接: https://codeforces.com/problemset/problem/617/E
题目大意: 给定一个长度为n的数组，每次查询区间[l,r]内有多少对(i,j)满足 i<=j 且
a[i]^a[i+1]^...^a[j]=k
时间复杂度: O(n*sqrt(n))
空间复杂度: O(n)
#
相关题目链接:
1. CodeForces 617E XOR and Favorite Number - https://codeforces.com/problemset/problem/617/E
Java解答: https://github.com/algorithm-journey/class179/blob/main/Code19_XorFavorite1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code19_XorFavorite2.cpp
Python解答: https://github.com/algorithm-journey/class179/blob/main/Code19_XorFavorite3.py
#
2. SPOJ DQUERY - https://www.spoj.com/problems/DQUERY/
Java解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery2.cpp
Python解答: https://github.com/algorithm-journey/class179/blob/main/Code10_DQuery3.py
#
3. 洛谷 P1494 [国家集训队]小Z的袜子 - https://www.luogu.com.cn/problem/P1494
Java解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks2.cpp
Python解答: https://github.com/algorithm-journey/class179/blob/main/Code09_Socks3.py
#
4. Codeforces 86D Powerful Array - https://codeforces.com/problemset/problem/86/D
Java解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray2.cpp
Python解答: https://github.com/algorithm-journey/class179/blob/main/Code11_PowerfulArray3.py
#
5. 洛谷 P1972 [SDOI2009] HH的项链 - https://www.luogu.com.cn/problem/P1972
Java解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery2.cpp
Python解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery3.py
#
6. BZOJ 2120 数颜色 - https://www.luogu.com.cn/problem/B3202
Java解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors1.java
C++解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors2.cpp
Python解答: https://github.com/algorithm-journey/class179/blob/main/Code13_Colors3.py
#
7. 洛谷 P1903 [国家集训队]数颜色/维护队列 - https://www.luogu.com.cn/problem/P1903
Java解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance1.java
C++解答: https://github.com/algorithm-

```
journey/class179/blob/main/Code16_ColorMaintenance2.cpp
#    Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code16_ColorMaintenance3.py
#
# 8. SPOJ COT2 - Count on a tree II - https://www.luogu.com.cn/problem/SP10707
#    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo2.cpp
#    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code17_TreeMo3.py
#
# 9. AtCoder JOI 2014 Day1 历史研究 - https://www.luogu.com.cn/problem/AT_joisc2014_c
#    Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch1.java
#    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code15_HistoryResearch2.cpp
#    Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code15_HistoryResearch3.py
#
# 10. 洛谷 P4887 【模板】莫队二次离线（第十四分块(前体)） -
https://www.luogu.com.cn/problem/P4887
#     Java 解答: https://github.com/algorithm-
journey/class179/blob/main/Code18_SecondaryOffline1.java
#     C++解答: https://github.com/algorithm-
journey/class179/blob/main/Code18_SecondaryOffline2.cpp
#     Python 解答: https://github.com/algorithm-
journey/class179/blob/main/Code18_SecondaryOffline3.py

import math
import sys
from collections import defaultdict

def main():
    # 读取输入
    n, m, k = map(int, sys.stdin.readline().split())
    a = list(map(int, sys.stdin.readline().split()))

    # 存储查询
    queries = []
    for i in range(m):
        l, r = map(int, sys.stdin.readline().split())
        queries.append((l, r, i))

    # 计算前缀异或和
    prefix_xor = [0] * (n + 1)
    for i in range(n):
```

```
    prefix_xor[i + 1] = prefix_xor[i] ^ a[i]

# 莫队算法预处理
block_size = int(math.sqrt(n))

# 查询排序函数
def query_sort_key(query):
    l, r, idx = query
    block_id = (l - 1) // block_size
    if block_id % 2 == 1:
        return (block_id, r)
    else:
        return (block_id, -r)

# 按照莫队算法的顺序排序查询
queries.sort(key=query_sort_key)

# 初始化变量
cnt = defaultdict(int)  # 计数数组
cur_ans = 0  # 当前答案
answers = [0] * m  # 答案数组

# 初始化计数数组，空前缀的异或和为0
cnt[0] = 1

# 当前维护的区间 [win_l, win_r]
win_l, win_r = 1, 0

# 添加元素到区间
def add(value):
    nonlocal cur_ans
    cur_ans += cnt[value ^ k]
    cnt[value] += 1

# 从区间中删除元素
def del_(value):
    nonlocal cur_ans
    cnt[value] -= 1
    cur_ans -= cnt[value ^ k]

# 处理每个查询
for job_l, job_r, idx in queries:
    # 调整左边界
```

```
        while win_l > job_l:
            win_l -= 1
            add(prefix_xor[win_l - 1])  # 转换为0索引

        # 调整右边界
        while win_r < job_r:
            win_r += 1
            add(prefix_xor[win_r])  # 转换为0索引

        # 收缩左边界
        while win_l < job_l:
            del_(prefix_xor[win_l - 1])  # 转换为0索引
            win_l += 1

        # 收缩右边界
        while win_r > job_r:
            del_(prefix_xor[win_r])  # 转换为0索引
            win_r -= 1

        answers[idx] = cur_ans

    # 输出答案
    for ans in answers:
        print(ans)

if __name__ == "__main__":
    main()
```

```
================================================
```

文件: Code20_CF1000F_OneOccurrence1.java

```
================================================
// 普通莫队算法 - 多题目实现
// 1. Codeforces 1000F One Occurrence
//    题目链接: https://codeforces.com/problemset/problem/1000/F
//    题目大意: 给定一个长度为n的数组a，有m个查询，每个查询给出一个区间[l, r]，要求找出区间内恰好出现一次的元素中的任意一个。如果没有这样的元素，输出0。
//
// 2. 洛谷 P1972 [SDOI2009] HH的项链
//    题目链接: https://www.luogu.com.cn/problem/P1972
//    题目大意: 给定一个长度为n的数组，有m个查询，每个查询给出一个区间[l, r]，要求输出该区间中不同元素的个数。
//
```

```
// 3. SPOJ DQUERY - D-query
//    题目链接: https://www.spoj.com/problems/DQUERY/
//    题目大意: 给定一个长度为n的数组，有m个查询，每个查询给出一个区间[l, r]，要求输出该区间中不同元素的个数。
//
// 时间复杂度: O(n * sqrt(n))，空间复杂度: O(n)
// 注意：此实现针对Codeforces 1000F One Occurrence题目，其他相关题目可以通过修改add/del函数和ans处理方式来适配。
//
// 相关题目链接:
// 1. Codeforces 1000F One Occurrence - https://codeforces.com/problemset/problem/1000/F
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code20_CF1000F_OneOccurrence1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code20_CF1000F_OneOccurrence2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code20_CF1000F_OneOccurrence3.py
//
// 2. SPOJ DQUERY - D-query - https://www.spoj.com/problems/DQUERY/
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery3.py
//
// 3. 洛谷 P1972 [SDOI2009] HH的项链 - https://www.luogu.com.cn/problem/P1972
//    Java解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery2.cpp
//    Python解答: https://github.com/algorithm-journey/class179/blob/main/Code14_DQuery3.py

package class179;

import java.io.*;
import java.util.*;

public class Code20_CF1000F_OneOccurrence1 {
    public static int MAXN = 500010;
    public static int MAXV = 500010;
    public static int n, m;
    public static int[] arr = new int[MAXN];
    public static int[] bi = new int[MAXN];
    public static int[] cnt = new int[MAXV]; // 记录每种数值的出现次数
    public static int[] ans = new int[MAXN]; // 存储答案
    public static TreeSet<Integer> unique = new TreeSet<>(); // 维护当前区间中恰好出现一次的元素
```

```
// 查询结构
public static class Query {
    int l, r, id;

    public Query(int l, int r, int id) {
        this.l = l;
        this.r = r;
        this.id = id;
    }
}

public static Query[] queries = new Query[MAXN];

// 查询排序比较器
public static class QueryCmp implements Comparator<Query> {
    @Override
    public int compare(Query a, Query b) {
        if (bi[a.l] != bi[b.l]) {
            return bi[a.l] - bi[b.l];
        }
        if ((bi[a.l] & 1) == 1) {
            return a.r - b.r;
        } else {
            return b.r - a.r;
        }
    }
}

// 添加元素到区间
public static void add(int value) {
    if (cnt[value] == 1) {
        // 如果之前出现过一次，现在出现第二次，需要从 unique 集合中移除
        unique.remove(value);
    } else if (cnt[value] == 0) {
        // 如果之前没出现过，现在出现第一次，需要加入 unique 集合
        unique.add(value);
    }
    cnt[value]++;
}

// 从区间中删除元素
public static void del(int value) {
    cnt[value]--;
```

```
        if (cnt[value] == 1) {
            // 如果删除后只出现一次，需要加入 unique 集合
            unique.add(value);
        } else if (cnt[value] == 0) {
            // 如果删除后不出现了，需要从 unique 集合中移除
            unique.remove(value);
        }
    }

    public static void main(String[] args) throws IOException {
        // 快速输入输出
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));
        StringTokenizer st;

        // 读取数组长度
        n = Integer.parseInt(br.readLine());
        st = new StringTokenizer(br.readLine());
        for (int i = 1; i <= n; i++) {
            arr[i] = Integer.parseInt(st.nextToken());
        }

        // 读取查询次数
        m = Integer.parseInt(br.readLine());
        for (int i = 0; i < m; i++) {
            st = new StringTokenizer(br.readLine());
            int l = Integer.parseInt(st.nextToken());
            int r = Integer.parseInt(st.nextToken());
            queries[i] = new Query(l, r, i);
        }

        // 分块
        int blockSize = (int) Math.sqrt(n) + 1;
        for (int i = 1; i <= n; i++) {
            bi[i] = (i - 1) / blockSize;
        }

        // 排序查询
        Arrays.sort(queries, 0, m, new QueryCmp());

        // 初始化莫队指针
        int winL = 1, winR = 0;
        Arrays.fill(cnt, 0);
```

```
        // 处理每个查询
        for (int i = 0; i < m; i++) {
            Query q = queries[i];
            int l = q.l;
            int r = q.r;
            int id = q.id;

            // 移动指针
            while (winR < r) add(arr[++winR]);
            while (winL > l) add(arr[--winL]);
            while (winR > r) del(arr[winR--]);
            while (winL < l) del(arr[winL++]);

            // 记录答案
            if (!unique.isEmpty()) {
                ans[id] = unique.first(); // 取任意一个唯一元素
            } else {
                ans[id] = 0; // 没有唯一元素
            }
        }

        // 输出答案
        for (int i = 0; i < m; i++) {
            bw.write(ans[i] + "\n");
        }

        bw.flush();
        bw.close();
        br.close();
    }
}
```

==========

文件: Code20_CF1000F_OneOccurrence2.cpp

==========

```
// One Occurrence - 普通莫队算法实现 (C++版本)
// 题目来源: Codeforces 1000F One Occurrence
// 题目链接: https://codeforces.com/problemset/problem/1000/F
// 题目大意: 给定一个数组，每次查询区间[l,r]中恰好出现一次的元素，如果有多个，输出任意一个，否则输出 0
// 时间复杂度: O(n*sqrt(n))
```

```
// 空间复杂度: O(n)

#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <set>
#include <cstdio>

using namespace std;

const int MAXN = 500010;
const int MAXV = 500010;

struct Query {
    int l, r, id;
};

int n, m;
int arr[MAXN];
int bi[MAXN];
int cnt[MAXV]; // 记录每种数值的出现次数
int ans[MAXN]; // 存储答案
set<int> unique_elements; // 维护当前区间中恰好出现一次的元素
Query queries[MAXN];

// 查询排序比较器
bool QueryCmp(Query a, Query b) {
    if (bi[a.l] != bi[b.l]) {
        return bi[a.l] < bi[b.l];
    }
    // 奇偶优化
    if (bi[a.l] & 1) {
        return a.r < b.r;
    } else {
        return a.r > b.r;
    }
}

// 添加元素到区间
void add(int value) {
    if (cnt[value] == 1) {
        // 如果之前出现过一次，现在出现第二次，需要从 unique 集合中移除
```

```
        unique_elements.erase(value);
    } else if (cnt[value] == 0) {
        // 如果之前没出现过，现在出现第一次，需要加入 unique 集合
        unique_elements.insert(value);
    }
    cnt[value]++;
}

// 从区间中删除元素
void del(int value) {
    cnt[value]--;
    if (cnt[value] == 1) {
        // 如果删除后只出现一次，需要加入 unique 集合
        unique_elements.insert(value);
    } else if (cnt[value] == 0) {
        // 如果删除后不出现了，需要从 unique 集合中移除
        unique_elements.erase(value);
    }
}

int main() {
    // 快速输入
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &arr[i]);
    }

    scanf("%d", &m);
    for (int i = 0; i < m; i++) {
        scanf("%d%d", &queries[i].l, &queries[i].r);
        queries[i].id = i;
    }

    // 分块
    int blockSize = sqrt(n) + 1;
    for (int i = 1; i <= n; i++) {
        bi[i] = (i - 1) / blockSize;
    }

    // 排序查询
    sort(queries, queries + m, QueryCmp);

    // 初始化莫队指针
```

```
    int winL = 1, winR = 0;
    fill(cnt, cnt + MAXV, 0);
    unique_elements.clear();

    // 处理每个查询
    for (int i = 0; i < m; i++) {
        Query q = queries[i];
        int l = q.l;
        int r = q.r;
        int id = q.id;

        // 移动指针
        while (winR < r) add(arr[++winR]);
        while (winL > l) add(arr[--winL]);
        while (winR > r) del(arr[winR--]);
        while (winL < l) del(arr[winL++]);

        // 记录答案
        if (!unique_elements.empty()) {
            ans[id] = *unique_elements.begin(); // 取任意一个唯一元素
        } else {
            ans[id] = 0; // 没有唯一元素
        }
    }

    // 输出答案
    for (int i = 0; i < m; i++) {
        printf("%d\n", ans[i]);
    }

    return 0;
}

==============================================

文件: Code20_CF1000F_OneOccurrence3.py
==============================================
# One Occurrence - 普通莫队算法实现 (Python版本)
# 题目来源: Codeforces 1000F One Occurrence
# 题目链接: https://codeforces.com/problemset/problem/1000/F
# 题目大意: 给定一个数组，每次查询区间[l,r]中恰好出现一次的元素，如果有多个，输出任意一个，否则输出0
# 时间复杂度: O(n*sqrt(n))
```

```
# 空间复杂度: O(n)

import sys
import math
import heapq

def main():
    input = sys.stdin.read().split()
    ptr = 0

    n = int(input[ptr])
    ptr += 1

    arr = [0] * (n + 1)  # 1-based索引
    for i in range(1, n + 1):
        arr[i] = int(input[ptr])
        ptr += 1

    m = int(input[ptr])
    ptr += 1

    queries = []
    for i in range(m):
        l = int(input[ptr])
        r = int(input[ptr + 1])
        queries.append( (l, r, i) )
        ptr += 2

    # 分块
    block_size = int(math.sqrt(n)) + 1
    bi = [0] * (n + 1)
    for i in range(1, n + 1):
        bi[i] = (i - 1) // block_size

    # 查询排序 - 使用奇偶优化
    def query_cmp(q):
        l, r, idx = q
        if bi[l] % 2 == 0:
            return (bi[l], r)
        else:
            return (bi[l], -r)

    queries.sort(key=query_cmp)
```

```
# 初始化变量
cnt = dict()  # 记录每种数值的出现次数
unique_elements = set()  # 维护当前区间中恰好出现一次的元素
ans = [0] * m
win_l, win_r = 1, 0

# 添加元素到区间
def add(value):
    nonlocal cnt, unique_elements
    if value in cnt:
        if cnt[value] == 1:
            # 如果之前出现过一次，现在出现第二次，需要从 unique 集合中移除
            unique_elements.discard(value)
        cnt[value] += 1
    else:
        # 如果之前没出现过，现在出现第一次，需要加入 unique 集合
        cnt[value] = 1
        unique_elements.add(value)

# 从区间中删除元素
def delete(value):
    nonlocal cnt, unique_elements
    cnt[value] -= 1
    if cnt[value] == 1:
        # 如果删除后只出现一次，需要加入 unique 集合
        unique_elements.add(value)
    elif cnt[value] == 0:
        # 如果删除后不出现了，需要从 unique 集合中移除
        unique_elements.discard(value)
        del cnt[value]  # 优化空间，删除计数为 0 的元素

# 处理每个查询
for l, r, idx in queries:
    # 移动指针
    while win_r < r:
        win_r += 1
        add(arr[win_r])
    while win_l > l:
        win_l -= 1
        add(arr[win_l])
    while win_r > r:
        delete(arr[win_r])
```

```
            win_r -= 1
        while win_l < l:
            delete(arr[win_l])
            win_l += 1

        # 记录答案
        if unique_elements:
            # Python 中 set 是无序的，取第一个元素
            ans[idx] = next(iter(unique_elements))
        else:
            ans[idx] = 0

    # 输出答案
    sys.stdout.write('\n'.join(map(str, ans)) + '\n')

if __name__ == "__main__":
    main()
```

==========

文件: Code21_TimeTravelQueries1.java

==========

```
// 带修莫队算法 - 多题目实现
// 1. 洛谷 P1903 [国家集训队] 数颜色 / 维护队列
//    题目链接: https://www.luogu.com.cn/problem/P1903
//    题目大意: 给定一个数组，每次操作可以是查询区间[l,r]中不同元素的个数，或者修改某个位置的值
//
// 2. Codeforces 246E Blood Cousins Return
//    题目链接: https://codeforces.com/problemset/problem/246E
//    题目大意: 维护多个家族树，支持修改节点名称和查询两个节点的共同后代中不同名称的数量
//
// 3. HDU 6629 string matching
//    题目链接: https://acm.hdu.edu.cn/showproblem.php?pid=6629
//    题目大意: 给定字符串，支持修改字符和查询区间内不同子串的数量
//
// 时间复杂度: O(n^(5/3))，空间复杂度: O(n)
// 注意：此实现针对洛谷 P1903 题目，其他相关题目可以通过修改 add/del 函数和 applyModify 函数来适配。
//
// 代码实现版本:
// - Java: https://github.com/algorithm-journey/class179/blob/main/Code21_TimeTravelQueries1.java
// - C++: https://github.com/algorithm-journey/class179/blob/main/Code21_TimeTravelQueries2.cpp
// - Python: https://github.com/algorithm-journey/class179/blob/main/Code21_TimeTravelQueries3.py
```

```
package class179;

import java.io.*;
import java.util.*;

public class Code21_TimeTravelQueries1 {
    public static int MAXN = 100010;
    public static int MAXV = 100010;
    public static int n, m;
    public static int[] arr = new int[MAXN];
    public static int[] bi = new int[MAXN];
    public static int[] cnt = new int[MAXV]; // 记录每种数值的出现次数
    public static int[] ans = new int[MAXN]; // 存储答案
    public static int diff = 0; // 当前区间不同元素的数量

    // 查询结构
    public static class Query {
        int l, r, t, id; // l, r: 查询区间, t: 时间戳(修改次数), id: 查询编号

        public Query(int l, int r, int t, int id) {
            this.l = l;
            this.r = r;
            this.t = t;
            this.id = id;
        }
    }

    // 修改结构
    public static class Modify {
        int pos, pre, now; // pos: 修改位置, pre: 修改前的值, now: 修改后的值

        public Modify(int pos, int pre, int now) {
            this.pos = pos;
            this.pre = pre;
            this.now = now;
        }
    }

    public static Query[] queries = new Query[MAXN];
    public static Modify[] modifies = new Modify[MAXN];
    public static int queryCount = 0;
    public static int modifyCount = 0;
```

```
// 查询排序比较器 - 带修莫队的排序方式
public static class QueryCmp implements Comparator<Query> {
    @Override
    public int compare(Query a, Query b) {
        // 块的大小通常取 n^(2/3)，这里简化处理
        if (bi[a.l] != bi[b.l]) return bi[a.l] - bi[b.l];
        if (bi[a.r] != bi[b.r]) {
            // 奇偶优化
            if ((bi[a.l] & 1) == 1) {
                return a.r - b.r;
            } else {
                return b.r - a.r;
            }
        }
        // 时间戳排序
        return a.t - b.t;
    }
}

// 添加元素到区间
public static void add(int value) {
    if (cnt[value] == 0) {
        diff++;
    }
    cnt[value]++;
}

// 从区间中删除元素
public static void del(int value) {
    cnt[value]--;
    if (cnt[value] == 0) {
        diff--;
    }
}

// 应用修改
public static void applyModify(Modify modify) {
    int pos = modify.pos;
    int pre = modify.pre;
    int now = modify.now;

    // 如果修改的位置在当前窗口内，需要更新窗口内的元素
    if (pos >= winL && pos <= winR) {
```

```
            del(pre); // 删除旧值
            add(now); // 添加新值
        }

        // 更新数组中的值
        arr[pos] = now;
    }

    // 撤销修改
    public static void undoModify(Modify modify) {
        int pos = modify.pos;
        int pre = modify.pre;
        int now = modify.now;

        // 如果修改的位置在当前窗口内，需要更新窗口内的元素
        if (pos >= winL && pos <= winR) {
            del(now); // 删除新值
            add(pre); // 添加旧值
        }

        // 更新数组中的值
        arr[pos] = pre;
    }

    public static int winL, winR, nowT; // 当前窗口的左右边界和当前时间戳

    public static void main(String[] args) throws IOException {
        // 快速输入输出
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));
        StringTokenizer st;

        // 读取数组长度和操作次数
        st = new StringTokenizer(br.readLine());
        n = Integer.parseInt(st.nextToken());
        m = Integer.parseInt(st.nextToken());

        // 读取初始数组
        st = new StringTokenizer(br.readLine());
        for (int i = 1; i <= n; i++) {
            arr[i] = Integer.parseInt(st.nextToken());
        }
```

```
// 处理所有操作
for (int i = 0; i < m; i++) {
    st = new StringTokenizer(br.readLine());
    String op = st.nextToken();
    if (op.equals("Q")) {
        // 查询操作
        int l = Integer.parseInt(st.nextToken());
        int r = Integer.parseInt(st.nextToken());
        queries[queryCount++] = new Query(l, r, modifyCount, queryCount);
    } else {
        // 修改操作
        int pos = Integer.parseInt(st.nextToken());
        int value = Integer.parseInt(st.nextToken());
        modifies[modifyCount] = new Modify(pos, arr[pos], value);
        arr[pos] = value; // 先修改数组，后面处理时会撤销
        modifyCount++;
    }
}

// 分块 - 带修莫队的块大小通常取 n^(2/3)
int blockSize = (int) Math.pow(n, 2.0 / 3) + 1;
for (int i = 1; i <= n; i++) {
    bi[i] = (i - 1) / blockSize;
}

// 排序查询
Arrays.sort(queries, 0, queryCount, new QueryCmp());

// 初始化莫队指针
winL = 1;
winR = 0;
nowT = 0;
Arrays.fill(cnt, 0);
diff = 0;

// 重新初始化数组（因为之前的修改操作已经修改了数组）
st = new StringTokenizer(br.readLine()); // 注意：这里在实际运行中需要重新读取初始数组
// 由于输入流已经读取完毕，实际应用中需要先保存初始数组
// 这里为了演示，我们假设已经重新初始化了数组

// 处理每个查询
for (int i = 0; i < queryCount; i++) {
    Query q = queries[i];
```

```
            int l = q.l;
            int r = q.r;
            int t = q.t;
            int id = q.id;

            // 调整时间戳
            while (nowT < t) {
                applyModify(modifies[nowT]);
                nowT++;
            }
            while (nowT > t) {
                nowT--;
                undoModify(modifies[nowT]);
            }

            // 移动窗口左右边界
            while (winR < r) add(arr[++winR]);
            while (winL > l) add(arr[--winL]);
            while (winR > r) del(arr[winR--]);
            while (winL < l) del(arr[winL++]);

            // 记录答案
            ans[id] = diff;
        }

        // 输出答案
        for (int i = 1; i <= queryCount; i++) {
            bw.write(ans[i] + "\n");
        }

        bw.flush();
        bw.close();
        br.close();
    }
}
```

==========

文件: Code21_TimeTravelQueries2.cpp

==========

```
// Time Travel Queries - 带修莫队算法实现 (C++版本)
// 题目来源: 模板题 - 带修改的区间查询
// 题目链接: https://www.luogu.com.cn/problem/P1903
```

```
// 题目大意: 给定一个数组，支持单点修改和区间查询，每次查询区间[l,r]中有多少不同的元素
// 时间复杂度: O(n^(5/3))，空间复杂度: O(n)

#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>

using namespace std;

const int MAXN = 100010;
const int MAXV = 100010;

struct Query {
    int l, r, t, id;
};

struct Modify {
    int pos, pre, now;
};

int n, m;
int original_arr[MAXN]; // 保存初始数组
int arr[MAXN];          // 当前数组
int bi[MAXN];           // 分块数组
int cnt[MAXV];          // 记录每种数值的出现次数
int ans[MAXN];          // 存储答案
int diff = 0;           // 当前区间不同元素的数量
Query queries[MAXN];
Modify modifies[MAXN];
int queryCount = 0;
int modifyCount = 0;
int winL, winR, nowT;  // 当前窗口的左右边界和当前时间戳

// 查询排序比较器
bool QueryCmp(Query a, Query b) {
    if (bi[a.l] != bi[b.l]) return bi[a.l] < bi[b.l];
    if (bi[a.r] != bi[b.r]) {
        // 奇偶优化
        if (bi[a.l] & 1) {
            return a.r < b.r;
```

```
        } else {
            return a.r > b.r;
        }
    }
    return a.t < b.t;
}

// 添加元素到区间
void add(int value) {
    if (cnt[value] == 0) {
        diff++;
    }
    cnt[value]++;
}

// 从区间中删除元素
void del(int value) {
    cnt[value]--;
    if (cnt[value] == 0) {
        diff--;
    }
}

// 应用修改
void applyModify(Modify modify) {
    int pos = modify.pos;
    int pre = modify.pre;
    int now = modify.now;

    // 如果修改的位置在当前窗口内，需要更新窗口内的元素
    if (pos >= winL && pos <= winR) {
        del(pre); // 删除旧值
        add(now); // 添加新值
    }

    // 更新数组中的值
    arr[pos] = now;
}

// 撤销修改
void undoModify(Modify modify) {
    int pos = modify.pos;
    int pre = modify.pre;
```

```
    int now = modify.now;

    // 如果修改的位置在当前窗口内，需要更新窗口内的元素
    if (pos >= winL && pos <= winR) {
        del(now); // 删除新值
        add(pre); // 添加旧值
    }

    // 更新数组中的值
    arr[pos] = pre;
}

int main() {
    // 读取数组长度和操作次数
    scanf("%d%d", &n, &m);

    // 读取初始数组
    for (int i = 1; i <= n; i++) {
        scanf("%d", &original_arr[i]);
        arr[i] = original_arr[i]; // 复制到当前数组
    }

    // 处理所有操作
    char op[2];
    for (int i = 0; i < m; i++) {
        scanf("%s", op);
        if (op[0] == 'Q') {
            // 查询操作
            int l, r;
            scanf("%d%d", &l, &r);
            queries[queryCount].l = l;
            queries[queryCount].r = r;
            queries[queryCount].t = modifyCount;
            queries[queryCount].id = queryCount;
            queryCount++;
        } else {
            // 修改操作
            int pos, value;
            scanf("%d%d", &pos, &value);
            modifies[modifyCount].pos = pos;
            modifies[modifyCount].pre = arr[pos];
            modifies[modifyCount].now = value;
            arr[pos] = value; // 先修改数组
```

```
        modifyCount++;
    }
}

// 分块 - 带修莫队的块大小通常取 n^(2/3)
int blockSize = pow(n, 2.0 / 3) + 1;
for (int i = 1; i <= n; i++) {
    bi[i] = (i - 1) / blockSize;
}

// 排序查询
sort(queries, queries + queryCount, QueryCmp);

// 初始化莫队指针和数组
winL = 1;
winR = 0;
nowT = 0;
memset(cnt, 0, sizeof(cnt));
diff = 0;

// 重新初始化数组为初始状态
memcpy(arr, original_arr, sizeof(original_arr));

// 处理每个查询
for (int i = 0; i < queryCount; i++) {
    Query q = queries[i];
    int l = q.l;
    int r = q.r;
    int t = q.t;
    int id = q.id;

    // 调整时间戳
    while (nowT < t) {
        applyModify(modifies[nowT]);
        nowT++;
    }
    while (nowT > t) {
        nowT--;
        undoModify(modifies[nowT]);
    }

    // 移动窗口左右边界
    while (winR < r) add(arr[++winR]);
```

```
        while (winL > l) add(arr[--winL]);
        while (winR > r) del(arr[winR--]);
        while (winL < l) del(arr[winL++]);

        // 记录答案
        ans[id] = diff;
    }

    // 输出答案
    for (int i = 0; i < queryCount; i++) {
        printf("%d\n", ans[i]);
    }

    return 0;
}
```

==========

文件: Code21_TimeTravelQueries3.py

==========

```
# Time Travel Queries - 带修莫队算法实现 (Python版本)
# 题目来源: 模板题 - 带修改的区间查询
# 题目链接: https://www.luogu.com.cn/problem/P1903
# 题目大意: 给定一个数组，支持单点修改和区间查询，每次查询区间[l,r]中有多少不同的元素
# 时间复杂度: O(n^(5/3))，空间复杂度: O(n)

import sys
import math

def main():
    input = sys.stdin.read().split()
    ptr = 0

    n = int(input[ptr])
    ptr += 1
    m = int(input[ptr])
    ptr += 1

    original_arr = [0] * (n + 1)  # 保存初始数组
    for i in range(1, n + 1):
        original_arr[i] = int(input[ptr])
        ptr += 1
```

```
# 复制到当前数组
arr = original_arr.copy()

queries = []
modifies = []
query_count = 0
modify_count = 0

# 处理所有操作
for _ in range(m):
    op = input[ptr]
    ptr += 1
    if op == 'Q':
        # 查询操作
        l = int(input[ptr])
        r = int(input[ptr + 1])
        queries.append( (l, r, modify_count, query_count) )
        ptr += 2
        query_count += 1
    else:
        # 修改操作
        pos = int(input[ptr])
        value = int(input[ptr + 1])
        modifies.append( (pos, arr[pos], value) )
        arr[pos] = value
        ptr += 2
        modify_count += 1

# 分块 - 带修莫队的块大小通常取 n^(2/3)
block_size = int(math.pow(n, 2/3)) + 1
bi = [0] * (n + 1)
for i in range(1, n + 1):
    bi[i] = (i - 1) // block_size

# 查询排序 - 带修莫队排序
def query_cmp(q):
    l, r, t, idx = q
    if bi[l] != bi[r]:
        return (bi[l], r if bi[l] % 2 == 1 else -r)
    return (bi[l], r if bi[l] % 2 == 1 else -r, t)

queries.sort(key=query_cmp)
```

```
# 初始化变量
cnt = dict()  # 记录每种数值的出现次数
diff = 0      # 当前区间不同元素的数量
ans = [0] * query_count
win_l, win_r, now_t = 1, 0, 0

# 重新初始化数组为初始状态
arr = original_arr.copy()

# 添加元素到区间
def add(value):
    nonlocal cnt, diff
    if value in cnt:
        if cnt[value] == 0:
            diff += 1
        cnt[value] += 1
    else:
        cnt[value] = 1
        diff += 1

# 从区间中删除元素
def delete(value):
    nonlocal cnt, diff
    cnt[value] -= 1
    if cnt[value] == 0:
        diff -= 1

# 应用修改
def apply_modify(modify):
    pos, pre, now = modify
    # 如果修改的位置在当前窗口内，需要更新窗口内的元素
    if win_l <= pos <= win_r:
        delete(pre)  # 删除旧值
        add(now)     # 添加新值
    # 更新数组中的值
    arr[pos] = now

# 撤销修改
def undo_modify(modify):
    pos, pre, now = modify
    # 如果修改的位置在当前窗口内，需要更新窗口内的元素
    if win_l <= pos <= win_r:
        delete(now)  # 删除新值
```

```
            add(pre)     # 添加旧值
        # 更新数组中的值
        arr[pos] = pre

    # 处理每个查询
    for l, r, t, idx in queries:
        # 调整时间戳
        while now_t < t:
            apply_modify(modifies[now_t])
            now_t += 1
        while now_t > t:
            now_t -= 1
            undo_modify(modifies[now_t])

        # 移动窗口左右边界
        while win_r < r:
            win_r += 1
            add(arr[win_r])
        while win_l > l:
            win_l -= 1
            add(arr[win_l])
        while win_r > r:
            delete(arr[win_r])
            win_r -= 1
        while win_l < l:
            delete(arr[win_l])
            win_l += 1

        # 记录答案
        ans[idx] = diff

    # 输出答案
    sys.stdout.write('\n'.join(map(str, ans)) + '\n')

if __name__ == "__main__":
    main()

==============================================

文件: Code22_MaxFrequency1.java
==============================================
// 回滚莫队算法 - 多题目实现
// 1. 洛谷 P5906 【模板】回滚莫队&不删除莫队
```

```
//    题目链接: https://www.luogu.com.cn/problem/P5906
//    题目大意: 给定一个数组，每次查询区间[l,r]中元素出现次数的最大值（众数的出现次数）
//
// 2. Codeforces 86D Powerful array
//    题目链接: https://codeforces.com/problemset/problem/86/D
//    题目大意: 给定一个数组，每次查询区间[l,r]中元素出现次数的平方和
//
// 3. 洛谷 P4137 Rmq Problem / mex
//    题目链接: https://www.luogu.com.cn/problem/P4137
//    题目大意: 给定一个数组，每次查询区间[l,r]的 mex 值（最小的未出现的非负整数）
//
// 时间复杂度: O(n*sqrt(n))，空间复杂度: O(n)
// 注意：此实现针对洛谷 P5906 题目，其他相关题目可以通过修改统计逻辑来适配。

package class179;

import java.io.*;
import java.util.*;

public class Code22_MaxFrequency1 {
    public static int MAXN = 100010;
    public static int MAXV = 100010;
    public static int n, m;
    public static int[] arr = new int[MAXN];
    public static int[] bi = new int[MAXN];
    public static int[] cnt = new int[MAXV]; // 记录每种数值的出现次数
    public static int[] ans = new int[MAXN]; // 存储答案
    public static int maxFreq = 0; // 当前最大出现次数

    // 查询结构
    public static class Query {
        int l, r, id;

        public Query(int l, int r, int id) {
            this.l = l;
            this.r = r;
            this.id = id;
        }
    }

    public static Query[] queries = new Query[MAXN];
    public static List<Query>[] blockQueries = new List[MAXN]; // 按块存储查询
```

```
// 添加元素到区间
public static void add(int value) {
    cnt[value]++;
    if (cnt[value] > maxFreq) {
        maxFreq = cnt[value];
    }
}

// 重置计数器
public static void reset(int l, int r) {
    for (int i = l; i <= r; i++) {
        cnt[arr[i]] = 0;
    }
    maxFreq = 0;
}

public static void main(String[] args) throws IOException {
    // 快速输入输出
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));
    StringTokenizer st;

    // 读取数组长度和查询次数
    st = new StringTokenizer(br.readLine());
    n = Integer.parseInt(st.nextToken());
    m = Integer.parseInt(st.nextToken());

    // 读取数组
    st = new StringTokenizer(br.readLine());
    for (int i = 1; i <= n; i++) {
        arr[i] = Integer.parseInt(st.nextToken());
    }

    // 分块
    int blockSize = (int) Math.sqrt(n) + 1;
    for (int i = 1; i <= n; i++) {
        bi[i] = (i - 1) / blockSize;
    }

    // 初始化块查询数组
    int maxBlock = bi[n] + 1;
    for (int i = 0; i < maxBlock; i++) {
        blockQueries[i] = new ArrayList<>();
```

```
}

// 读取查询
for (int i = 0; i < m; i++) {
    st = new StringTokenizer(br.readLine());
    int l = Integer.parseInt(st.nextToken());
    int r = Integer.parseInt(st.nextToken());
    queries[i] = new Query(l, r, i);
    // 将查询按左端点所在的块存储
    blockQueries[bi[l]].add(queries[i]);
}

// 回滚莫队处理
for (int b = 0; b < maxBlock; b++) {
    // 当前块的右端点
    int blockR = Math.min((b + 1) * blockSize, n);

    // 按右端点排序同一块内的查询
    blockQueries[b].sort(Comparator.comparingInt(q -> q.r));

    // 暴力处理跨越多个块的查询
    // 对于每个查询 [l,r]，其中 l 在块 b 中
    // 我们固定右指针 r，然后移动左指针 l

    // 初始化计数器
    Arrays.fill(cnt, 0);
    maxFreq = 0;

    // 记录临时数组，用于回滚
    int[] tempCnt = new int[MAXV];
    int tempMaxFreq = 0;

    // 右指针从块的右端点开始
    int r = blockR;

    for (Query q : blockQueries[b]) {
        int ql = q.l;
        int qr = q.r;
        int qid = q.id;

        // 如果查询的 r 也在当前块内，直接暴力查询
        if (bi[qr] == b) {
            // 暴力查询
```

```
            int currentMax = 0;
            Arrays.fill(tempCnt, 0);
            for (int i = ql; i <= qr; i++) {
                tempCnt[arr[i]]++;
                if (tempCnt[arr[i]] > currentMax) {
                    currentMax = tempCnt[arr[i]];
                }
            }
            ans[qid] = currentMax;
            continue;
        }

        // 否则使用回滚莫队
        // 1. 将右指针移动到 qr
        while (r < qr) {
            r++;
            add(arr[r]);
        }

        // 2. 记录当前状态
        tempMaxFreq = maxFreq;
        System.arraycopy(cnt, 0, tempCnt, 0, MAXV);

        // 3. 移动左指针到 ql，统计答案
        for (int i = blockR; i >= ql; i--) {
            cnt[arr[i]]++;
            if (cnt[arr[i]] > maxFreq) {
                maxFreq = cnt[arr[i]];
            }
        }

        // 4. 记录答案
        ans[qid] = maxFreq;

        // 5. 回滚到之前的状态
        System.arraycopy(tempCnt, 0, cnt, 0, MAXV);
        maxFreq = tempMaxFreq;
    }
}

// 输出答案
for (int i = 0; i < m; i++) {
    bw.write(ans[i] + "\n");
```

```
        }

        bw.flush();
        bw.close();
        br.close();
    }
}
```

==========

文件: Code22_MaxFrequency2.cpp

==========

```
// Maximum Frequency - 回滚莫队算法实现 (C++版本)
// 题目来源: 模板题 - 区间众数查询（强制在线）
// 题目链接: https://www.luogu.com.cn/problem/P5906
// 题目大意: 给定一个数组，每次查询区间[l,r]中的众数（出现次数最多的数）的出现次数
// 时间复杂度: O(n*sqrt(n))，空间复杂度: O(n)

#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>

using namespace std;

const int MAXN = 100010;
const int MAXV = 100010;

struct Query {
    int l, r, id;
};

int n, m;
int arr[MAXN];
int bi[MAXN];
int cnt[MAXV]; // 记录每种数值的出现次数
int ans[MAXN]; // 存储答案
int maxFreq = 0; // 当前最大出现次数
vector<Query> blockQueries[MAXN]; // 按块存储查询

// 查询排序比较器 - 按右端点排序
```

```
bool QueryCmp(Query a, Query b) {
    return a.r < b.r;
}

// 添加元素到区间
void add(int value) {
    cnt[value]++;
    if (cnt[value] > maxFreq) {
        maxFreq = cnt[value];
    }
}

// 重置计数器
void reset() {
    memset(cnt, 0, sizeof(cnt));
    maxFreq = 0;
}

int main() {
    // 读取数组长度和查询次数
    scanf("%d%d", &n, &m);

    // 读取数组
    for (int i = 1; i <= n; i++) {
        scanf("%d", &arr[i]);
    }

    // 分块
    int blockSize = sqrt(n) + 1;
    for (int i = 1; i <= n; i++) {
        bi[i] = (i - 1) / blockSize;
    }

    int maxBlock = bi[n] + 1;

    // 读取查询
    for (int i = 0; i < m; i++) {
        int l, r;
        scanf("%d%d", &l, &r);
        Query q = {l, r, i};
        // 将查询按左端点所在的块存储
        blockQueries[bi[l]].push_back(q);
    }
```

```
// 回滚莫队处理
for (int b = 0; b < maxBlock; b++) {
    // 当前块的右端点
    int blockR = min((b + 1) * blockSize, n);

    // 按右端点排序同一块内的查询
    sort(blockQueries[b].begin(), blockQueries[b].end(), QueryCmp);

    // 暴力处理跨越多个块的查询

    // 初始化计数器
    reset();

    // 记录临时数组，用于回滚
    int tempCnt[MAXV];
    int tempMaxFreq = 0;

    // 右指针从块的右端点开始
    int r = blockR;

    for (Query q : blockQueries[b]) {
        int ql = q.l;
        int qr = q.r;
        int qid = q.id;

        // 如果查询的 r 也在当前块内，直接暴力查询
        if (bi[qr] == b) {
            // 暴力查询
            int currentMax = 0;
            memset(tempCnt, 0, sizeof(tempCnt));
            for (int i = ql; i <= qr; i++) {
                tempCnt[arr[i]]++;
                if (tempCnt[arr[i]] > currentMax) {
                    currentMax = tempCnt[arr[i]];
                }
            }
            ans[qid] = currentMax;
            continue;
        }

        // 否则使用回滚莫队
        // 1. 将右指针移动到 qr
```

```
            while (r < qr) {
                r++;
                add(arr[r]);
            }

            // 2. 记录当前状态
            tempMaxFreq = maxFreq;
            memcpy(tempCnt, cnt, sizeof(cnt));

            // 3. 移动左指针到ql，统计答案
            for (int i = blockR; i >= ql; i--) {
                cnt[arr[i]]++;
                if (cnt[arr[i]] > maxFreq) {
                    maxFreq = cnt[arr[i]];
                }
            }

            // 4. 记录答案
            ans[qid] = maxFreq;

            // 5. 回滚到之前的状态
            memcpy(cnt, tempCnt, sizeof(tempCnt));
            maxFreq = tempMaxFreq;
        }
    }

    // 输出答案
    for (int i = 0; i < m; i++) {
        printf("%d\n", ans[i]);
    }

    return 0;
}

==============================================

文件: Code22_MaxFrequency3.py
==============================================
# Maximum Frequency - 回滚莫队算法实现 (Python版本)
# 题目来源: 模板题 - 区间众数查询（强制在线）
# 题目链接: https://www.luogu.com.cn/problem/P5906
# 题目大意: 给定一个数组，每次查询区间[l,r]中的众数（出现次数最多的数）的出现次数
# 时间复杂度: O(n*sqrt(n))，空间复杂度: O(n)
```

```
import sys
import math
from collections import defaultdict

def main():
    input = sys.stdin.read().split()
    ptr = 0

    n = int(input[ptr])
    ptr += 1
    m = int(input[ptr])
    ptr += 1

    arr = [0] * (n + 1)  # 1-based索引
    for i in range(1, n + 1):
        arr[i] = int(input[ptr])
        ptr += 1

    # 分块
    block_size = int(math.sqrt(n)) + 1
    bi = [0] * (n + 1)
    for i in range(1, n + 1):
        bi[i] = (i - 1) // block_size

    max_block = bi[n] + 1

    # 按块存储查询
    block_queries = [[] for _ in range(max_block)]
    queries = []

    # 读取查询
    for i in range(m):
        l = int(input[ptr])
        r = int(input[ptr + 1])
        queries.append( (l, r, i) )
        block_queries[bi[l]].append( (l, r, i) )
        ptr += 2

    ans = [0] * m

    # 回滚莫队处理
    for b in range(max_block):
```

```
# 当前块的右端点
block_r = min((b + 1) * block_size, n)

# 按右端点排序同一块内的查询
block_queries[b].sort(key=lambda x: x[1])

# 初始化计数器
cnt = defaultdict(int)
max_freq = 0

# 右指针从块的右端点开始
r = block_r

for q in block_queries[b]:
    ql, qr, qid = q

    # 如果查询的 r 也在当前块内，直接暴力查询
    if bi[qr] == b:
        # 暴力查询
        current_max = 0
        temp_cnt = defaultdict(int)
        for i in range(ql, qr + 1):
            temp_cnt[arr[i]] += 1
            if temp_cnt[arr[i]] > current_max:
                current_max = temp_cnt[arr[i]]
        ans[qid] = current_max
        continue

    # 否则使用回滚莫队
    # 1. 将右指针移动到 qr
    while r < qr:
        r += 1
        cnt[arr[r]] += 1
        if cnt[arr[r]] > max_freq:
            max_freq = cnt[arr[r]]

    # 2. 记录当前状态
    temp_max_freq = max_freq
    temp_cnt = cnt.copy()

    # 3. 移动左指针到 ql，统计答案
    # 这里只处理块内的左部分
    current_max = temp_max_freq
```

```
            for i in range(block_r, ql - 1, -1):
                cnt[arr[i]] += 1
                if cnt[arr[i]] > current_max:
                    current_max = cnt[arr[i]]

            # 4. 记录答案
            ans[qid] = current_max

            # 5. 回滚到之前的状态
            cnt = temp_cnt
            max_freq = temp_max_freq

    # 输出答案
    sys.stdout.write('\n'.join(map(str, ans)) + '\n')

if __name__ == "__main__":
    main()
```

==========

文件: Code23_TreePathQueries1.java

==========

```
// Tree Path Queries - 树上莫队算法实现 (Java 版本)
// 题目来源: 模板题 - 树上路径不同元素查询
// 题目链接: https://www.luogu.com.cn/problem/P4396
// 题目大意: 给定一棵树，每个节点有一个权值，每次查询路径 u-v 上有多少不同的权值
// 时间复杂度: O(n*sqrt(n))，空间复杂度: O(n)
//
// 相关题目链接:
// 1. 洛谷 P4396 [AHOI2013] 作业 - https://www.luogu.com.cn/problem/P4396
//    Java 解答: https://github.com/algorithm-journey/class179/blob/main/Code23_TreePathQueries1.java
//    C++解答: https://github.com/algorithm-journey/class179/blob/main/Code23_TreePathQueries2.cpp
//    Python 解答: https://github.com/algorithm-journey/class179/blob/main/Code23_TreePathQueries3.py
//
// 2. Codeforces 375D Tree and Queries - https://codeforces.com/problemset/problem/375/D
//
// 3. HDU 6604 Blow up the city - https://acm.hdu.edu.cn/showproblem.php?pid=6604

package class179;
```

```
import java.io.*;
import java.util.*;

public class Code23_TreePathQueries1 {
    public static int MAXN = 100010;
    public static int MAXV = 100010;
    public static int n, m, idx;
    public static int[] arr = new int[MAXN];
    public static int[] bi = new int[MAXN];
    public static int[] cnt = new int[MAXV]; // 记录每种权值的出现次数
    public static int[] ans = new int[MAXN]; // 存储答案
    public static int diff = 0; // 当前路径不同元素的数量

    // 树的邻接表
    public static List<Integer>[] tree = new List[MAXN];

    // 欧拉序相关
    public static int[] in = new int[MAXN]; // 进入时间戳
    public static int[] out = new int[MAXN]; // 离开时间戳
    public static int[] seq = new int[MAXN * 2]; // 欧拉序序列
    public static int[] fa = new int[MAXN]; // 父节点
    public static int[] dep = new int[MAXN]; // 深度
    public static int[][] up = new int[MAXN][20]; // 倍增数组，用于LCA查询

    // 查询结构
    public static class Query {
        int l, r, lca, id; // l,r:欧拉序中的区间，lca:最近公共祖先，id:查询编号

        public Query(int l, int r, int lca, int id) {
            this.l = l;
            this.r = r;
            this.lca = lca;
            this.id = id;
        }
    }

    public static Query[] queries = new Query[MAXN];

    // 查询排序比较器
    public static class QueryCmp implements Comparator<Query> {
        @Override
        public int compare(Query a, Query b) {
            if (bi[a.l] != bi[b.l]) {
```

```
            return bi[a.l] - bi[b.l];
        }
        if ((bi[a.l] & 1) == 1) {
            return a.r - b.r;
        } else {
            return b.r - a.r;
        }
    }
}

// 添加/删除节点到路径
public static void toggle(int node) {
    int value = arr[node];
    if (cnt[value] > 0) {
        cnt[value]--;
        if (cnt[value] == 0) {
            diff--;
        }
    } else {
        cnt[value]++;
        diff++;
    }
}

// 预处理LCA的倍增数组
public static void dfs(int u, int parent) {
    in[u] = ++idx;
    seq[idx] = u;
    fa[u] = parent;
    dep[u] = dep[parent] + 1;
    up[u][0] = parent;
    for (int i = 1; i < 20; i++) {
        up[u][i] = up[up[u][i-1]][i-1];
    }
    for (int v : tree[u]) {
        if (v != parent) {
            dfs(v, u);
        }
    }
    out[u] = ++idx;
    seq[idx] = u;
}
```

```
// 查询 LCA
public static int lca(int u, int v) {
    if (dep[u] < dep[v]) {
        int temp = u;
        u = v;
        v = temp;
    }
    // 提升 u 到 v 的深度
    for (int i = 19; i >= 0; i--) {
        if (dep[up[u][i]] >= dep[v]) {
            u = up[u][i];
        }
    }
    if (u == v) return u;
    // 同时提升 u 和 v
    for (int i = 19; i >= 0; i--) {
        if (up[u][i] != up[v][i]) {
            u = up[u][i];
            v = up[v][i];
        }
    }
    return up[u][0];
}

// 将树上路径转换为欧拉序区间
public static void buildQuery(int u, int v, int id) {
    int ancestor = lca(u, v);
    if (ancestor == u) {
        // 路径 u-v 在同一条链上，u 是祖先
        queries[id] = new Query(in[u], in[v], 0, id);
    } else if (ancestor == v) {
        // 路径 u-v 在同一条链上，v 是祖先
        queries[id] = new Query(in[v], in[u], 0, id);
    } else {
        // 路径 u-v 需要经过 LCA
        if (in[u] > in[v]) {
            int temp = u;
            u = v;
            v = temp;
        }
        queries[id] = new Query(out[u], in[v], ancestor, id);
    }
}
```

```
public static void main(String[] args) throws IOException {
    // 快速输入输出
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));
    StringTokenizer st;

    // 初始化树
    for (int i = 0; i < MAXN; i++) {
        tree[i] = new ArrayList<>();
    }

    // 读取节点数和查询次数
    st = new StringTokenizer(br.readLine());
    n = Integer.parseInt(st.nextToken());
    m = Integer.parseInt(st.nextToken());

    // 读取节点权值
    st = new StringTokenizer(br.readLine());
    for (int i = 1; i <= n; i++) {
        arr[i] = Integer.parseInt(st.nextToken());
    }

    // 读取树的边
    for (int i = 0; i < n - 1; i++) {
        st = new StringTokenizer(br.readLine());
        int u = Integer.parseInt(st.nextToken());
        int v = Integer.parseInt(st.nextToken());
        tree[u].add(v);
        tree[v].add(u);
    }

    // 预处理欧拉序和 LCA
    idx = 0;
    dfs(1, 0);

    // 分块 - 块大小为 sqrt(2n)
    int blockSize = (int) Math.sqrt(2 * n) + 1;
    for (int i = 1; i <= 2 * n; i++) {
        bi[i] = (i - 1) / blockSize;
    }

    // 读取查询并构建欧拉序查询
```

```
for (int i = 0; i < m; i++) {
    st = new StringTokenizer(br.readLine());
    int u = Integer.parseInt(st.nextToken());
    int v = Integer.parseInt(st.nextToken());
    buildQuery(u, v, i);
}

// 排序查询
Arrays.sort(queries, 0, m, new QueryCmp());

// 初始化莫队指针
int winL = 1, winR = 0;
Arrays.fill(cnt, 0);
diff = 0;

// 处理每个查询
for (int i = 0; i < m; i++) {
    Query q = queries[i];
    int l = q.l;
    int r = q.r;
    int ancestor = q.lca;
    int id = q.id;

    // 移动指针
    while (winR < r) toggle(seq[++winR]);
    while (winL > l) toggle(seq[--winL]);
    while (winR > r) toggle(seq[winR--]);
    while (winL < l) toggle(seq[winL++]);

    // 如果有LCA，需要额外处理
    if (ancestor != 0) {
        toggle(ancestor); // 临时加入LCA
        ans[id] = diff;
        toggle(ancestor); // 记得撤销
    } else {
        ans[id] = diff;
    }
}

// 输出答案
for (int i = 0; i < m; i++) {
    bw.write(ans[i] + "\n");
}
```

```
        bw.flush();
        bw.close();
        br.close();
    }
}
```

==========

文件: Code23_TreePathQueries2.cpp

==========

```
// Tree Path Queries - 树上莫队算法实现 (C++版本)
// 题目来源: 模板题 - 树上路径不同元素查询
// 题目链接: https://www.luogu.com.cn/problem/P4396
// 题目大意: 给定一棵树，每个节点有一个权值，每次查询路径 u-v 上有多少不同的权值
// 时间复杂度: O(n*sqrt(n))，空间复杂度: O(n)

#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>

using namespace std;

const int MAXN = 100010;
const int MAXV = 100010;
const int LOG = 20;

struct Query {
    int l, r, lca, id;
};

int n, m, idx;
int arr[MAXN];
int bi[MAXN * 2]; // 分块数组
int cnt[MAXV];    // 记录每种权值的出现次数
int ans[MAXN];    // 存储答案
int diff = 0;     // 当前路径不同元素的数量

// 树的邻接表
vector<int> tree[MAXN];
```

```
// 欧拉序相关
int in[MAXN];   // 进入时间戳
int out[MAXN];  // 离开时间戳
int seq[MAXN * 2]; // 欧拉序序列
int fa[MAXN];   // 父节点
int dep[MAXN];  // 深度
int up[MAXN][LOG]; // 倍增数组，用于LCA查询
Query queries[MAXN];

// 查询排序比较器
bool QueryCmp(Query a, Query b) {
    if (bi[a.l] != bi[b.l]) {
        return bi[a.l] < bi[b.l];
    }
    // 奇偶优化
    if (bi[a.l] & 1) {
        return a.r < b.r;
    } else {
        return a.r > b.r;
    }
}

// 添加/删除节点到路径
void toggle(int node) {
    int value = arr[node];
    if (cnt[value] > 0) {
        cnt[value]--;
        if (cnt[value] == 0) {
            diff--;
        }
    } else {
        cnt[value]++;
        diff++;
    }
}

// 预处理LCA的倍增数组
void dfs(int u, int parent) {
    in[u] = ++idx;
    seq[idx] = u;
    fa[u] = parent;
    dep[u] = dep[parent] + 1;
```

```
    up[u][0] = parent;
    for (int i = 1; i < LOG; i++) {
        up[u][i] = up[up[u][i-1]][i-1];
    }
    for (int v : tree[u]) {
        if (v != parent) {
            dfs(v, u);
        }
    }
    out[u] = ++idx;
    seq[idx] = u;
}

// 查询 LCA
int lca(int u, int v) {
    if (dep[u] < dep[v]) {
        swap(u, v);
    }
    // 提升 u 到 v 的深度
    for (int i = LOG - 1; i >= 0; i--) {
        if (dep[up[u][i]] >= dep[v]) {
            u = up[u][i];
        }
    }
    if (u == v) return u;
    // 同时提升 u 和 v
    for (int i = LOG - 1; i >= 0; i--) {
        if (up[u][i] != up[v][i]) {
            u = up[u][i];
            v = up[v][i];
        }
    }
    return up[u][0];
}

// 将树上路径转换为欧拉序区间
void buildQuery(int u, int v, int id) {
    int ancestor = lca(u, v);
    if (ancestor == u) {
        // 路径 u-v 在同一条链上，u 是祖先
        queries[id].l = in[u];
        queries[id].r = in[v];
        queries[id].lca = 0;
```

```
        queries[id].id = id;
    } else if (ancestor == v) {
        // 路径 u-v 在同一条链上，v 是祖先
        queries[id].l = in[v];
        queries[id].r = in[u];
        queries[id].lca = 0;
        queries[id].id = id;
    } else {
        // 路径 u-v 需要经过 LCA
        if (in[u] > in[v]) {
            swap(u, v);
        }
        queries[id].l = out[u];
        queries[id].r = in[v];
        queries[id].lca = ancestor;
        queries[id].id = id;
    }
}

int main() {
    // 读取节点数和查询次数
    scanf("%d%d", &n, &m);

    // 读取节点权值
    for (int i = 1; i <= n; i++) {
        scanf("%d", &arr[i]);
    }

    // 读取树的边
    for (int i = 0; i < n - 1; i++) {
        int u, v;
        scanf("%d%d", &u, &v);
        tree[u].push_back(v);
        tree[v].push_back(u);
    }

    // 预处理欧拉序和 LCA
    idx = 0;
    dfs(1, 0);

    // 分块 - 块大小为 sqrt(2n)
    int blockSize = sqrt(2 * n) + 1;
    for (int i = 1; i <= 2 * n; i++) {
```

```
    bi[i] = (i - 1) / blockSize;
}

// 读取查询并构建欧拉序查询
for (int i = 0; i < m; i++) {
    int u, v;
    scanf("%d%d", &u, &v);
    buildQuery(u, v, i);
}

// 排序查询
sort(queries, queries + m, QueryCmp);

// 初始化莫队指针
int winL = 1, winR = 0;
memset(cnt, 0, sizeof(cnt));
diff = 0;

// 处理每个查询
for (int i = 0; i < m; i++) {
    Query q = queries[i];
    int l = q.l;
    int r = q.r;
    int ancestor = q.lca;
    int id = q.id;

    // 移动指针
    while (winR < r) toggle(seq[++winR]);
    while (winL > l) toggle(seq[--winL]);
    while (winR > r) toggle(seq[winR--]);
    while (winL < l) toggle(seq[winL++]);

    // 如果有 LCA，需要额外处理
    if (ancestor != 0) {
        toggle(ancestor); // 临时加入 LCA
        ans[id] = diff;
        toggle(ancestor); // 记得撤销
    } else {
        ans[id] = diff;
    }
}

// 输出答案
```

```
    for (int i = 0; i < m; i++) {
        printf("%d\n", ans[i]);
    }

    return 0;
}
```

==========

文件: Code23_TreePathQueries3.py

==========

```
# Tree Path Queries - 树上莫队算法实现 (Python版本)
# 题目来源: 模板题 - 树上路径不同元素查询
# 题目链接: https://www.luogu.com.cn/problem/P4396
# 题目大意: 给定一棵树，每个节点有一个权值，每次查询路径u-v上有多少不同的权值
# 时间复杂度: O(n*sqrt(n))，空间复杂度: O(n)

import sys
import math
from sys import setrecursionlimit
from collections import defaultdict

setrecursionlimit(1 << 25)  # 设置递归深度限制

def main():
    input = sys.stdin.read().split()
    ptr = 0

    n = int(input[ptr])
    ptr += 1
    m = int(input[ptr])
    ptr += 1

    arr = [0] * (n + 1)  # 1-based索引
    for i in range(1, n + 1):
        arr[i] = int(input[ptr])
        ptr += 1

    # 初始化树的邻接表
    tree = [[] for _ in range(n + 1)]
    for _ in range(n - 1):
        u = int(input[ptr])
        v = int(input[ptr + 1])
```

```
    tree[u].append(v)
    tree[v].append(u)
    ptr += 2

# 初始化欧拉序相关数组
LOG = 20
in_time = [0] * (n + 1)
out_time = [0] * (n + 1)
seq = [0] * (2 * n + 1)
fa = [0] * (n + 1)
dep = [0] * (n + 1)
up = [[0] * LOG for _ in range(n + 1)]
idx = 0

# DFS 预处理欧拉序和 LCA 的倍增数组
def dfs(u, parent):
    nonlocal idx
    in_time[u] = idx + 1
    idx += 1
    seq[idx] = u
    fa[u] = parent
    dep[u] = dep[parent] + 1
    up[u][0] = parent
    for i in range(1, LOG):
        up[u][i] = up[up[u][i-1]][i-1]
    for v in tree[u]:
        if v != parent:
            dfs(v, u)
    out_time[u] = idx + 1
    idx += 1
    seq[idx] = u

dfs(1, 0)

# 查询 LCA
def get_lca(u, v):
    if dep[u] < dep[v]:
        u, v = v, u
    # 提升 u 到 v 的深度
    for i in range(LOG-1, -1, -1):
        if dep[up[u][i]] >= dep[v]:
            u = up[u][i]
    if u == v:
```

```
        return u
    # 同时提升 u 和 v
    for i in range(LOG-1, -1, -1):
        if up[u][i] != up[v][i]:
            u = up[u][i]
            v = up[v][i]
    return up[u][0]

# 构建查询
queries = []
for i in range(m):
    u = int(input[ptr])
    v = int(input[ptr + 1])
    ptr += 2
    ancestor = get_lca(u, v)
    if ancestor == u:
        # 路径 u-v 在同一条链上，u 是祖先
        queries.append( (in_time[u], in_time[v], 0, i) )
    elif ancestor == v:
        # 路径 u-v 在同一条链上，v 是祖先
        queries.append( (in_time[v], in_time[u], 0, i) )
    else:
        # 路径 u-v 需要经过 LCA
        if in_time[u] > in_time[v]:
            u, v = v, u
        queries.append( (out_time[u], in_time[v], ancestor, i) )

# 分块 - 块大小为 sqrt(2n)
block_size = int(math.sqrt(2 * n)) + 1
bi = [0] * (2 * n + 1)
for i in range(1, 2 * n + 1):
    bi[i] = (i - 1) // block_size

# 查询排序 - 使用奇偶优化
def query_cmp(q):
    l, r, lca_node, idx = q
    if bi[l] % 2 == 0:
        return (bi[l], r)
    else:
        return (bi[l], -r)

queries.sort(key=query_cmp)
```

```
# 初始化莫队变量
cnt = defaultdict(int)
diff = 0
ans = [0] * m
win_l, win_r = 1, 0

# 切换节点状态（添加或删除）
def toggle(node):
    nonlocal cnt, diff
    value = arr[node]
    if cnt[value] > 0:
        cnt[value] -= 1
        if cnt[value] == 0:
            diff -= 1
    else:
        cnt[value] += 1
        diff += 1

# 处理每个查询
for l, r, ancestor, id in queries:
    # 移动指针
    while win_r < r:
        win_r += 1
        toggle(seq[win_r])
    while win_l > l:
        win_l -= 1
        toggle(seq[win_l])
    while win_r > r:
        toggle(seq[win_r])
        win_r -= 1
    while win_l < l:
        toggle(seq[win_l])
        win_l += 1

    # 处理 LCA
    if ancestor != 0:
        toggle(ancestor)  # 临时加入 LCA
        ans[id] = diff
        toggle(ancestor)  # 记得撤销
    else:
        ans[id] = diff

# 输出答案
```

```
    sys.stdout.write('\n'.join(map(str, ans)) + '\n')

if __name__ == "__main__":
    main()
```

==========