

=====

文件夹: class139_LinearBasisAlgorithms

=====

[Markdown 文件]

=====

文件: ADDITIONAL_PROBLEMS.md

=====

线性基补充题目列表

经典题目

1. 最大异或和类题目

1. **洛谷 P3812 【模板】线性基**

- 链接: <https://www.luogu.com.cn/problem/P3812>
- 类型: 模板题
- 算法: 普通消元法

2. **LeetCode 421. 数组中两个数的最大异或值**

- 链接: <https://leetcode.cn/problems/maximum-xor-of-two-numbers-in-an-array/>
- 类型: 字典树/线性基
- 算法: 线性基或字典树

3. **洛谷 P4151 [WC2011]最大 XOR 和路径**

- 链接: <https://www.luogu.com.cn/problem/P4151>
- 类型: 图论+线性基
- 算法: DFS 找环+线性基

2. 第 k 小异或和类题目

1. **LOJ #114. 第 k 小异或和**

- 链接: <https://loj.ac/p/114>
- 类型: 模板题
- 算法: 高斯消元法

2. **HDU 3949 XOR**

- 链接: <http://acm.hdu.edu.cn/showproblem.php?pid=3949>
- 类型: 经典题
- 算法: 高斯消元法

3. 线性基+贪心类题目

1. **洛谷 P4570 [BJWC2011]元素**

- 链接: <https://www.luogu.com.cn/problem/P4570>
- 类型: 贪心+线性基

- 算法：排序+贪心+线性基

2. **BZOJ 2460 元素**

- 链接：<https://www.lydsy.com/JudgeOnline/problem.php?id=2460>

- 类型：贪心+线性基

- 算法：排序+贪心+线性基

4. 线性基应用类题目

1. **洛谷 P3857 [TJOI2008]彩灯**

- 链接：<https://www.luogu.com.cn/problem/P3857>

- 类型：线性基应用

- 算法：线性基求秩

2. **Codeforces 1101G (Zero XOR Subset)-less**

- 链接：<https://codeforces.com/problemset/problem/1101/G>

- 类型：前缀和+线性基

- 算法：前缀异或和+线性基

进阶题目

1. 可持久化线性基

1. **洛谷 P4301 [CQOI2013]最大异或和**

- 链接：<https://www.luogu.com.cn/problem/P4301>

- 类型：可持久化线性基

- 算法：可持久化线性基

2. **洛谷 P4580 [BJOI2014]路径**

- 链接：<https://www.luogu.com.cn/problem/P4580>

- 类型：可持久化线性基

- 算法：可持久化线性基

2. 线性基+图论

1. **洛谷 P4151 [WC2011]最大 XOR 和路径**

- 链接：<https://www.luogu.com.cn/problem/P4151>

- 类型：图论+线性基

- 算法：DFS 找环+线性基

2. **Codeforces 938G Shortest Path Queries**

- 链接：<https://codeforces.com/problemset/problem/938/G>

- 类型：图论+线性基+分治

- 算法：线段树分治+线性基

3. 线性基+数据结构

1. **洛谷 P3292 [SCOI2016]幸运数字**
 - 链接: <https://www.luogu.com.cn/problem/P3292>
 - 类型: 线性基+倍增
 - 算法: 树上倍增+线性基
2. **洛谷 P5607 [Noi2013]无力回天 NOI2017**
 - 链接: <https://www.luogu.com.cn/problem/P5607>
 - 类型: 线性基+线段树
 - 算法: 线段树套线性基

AtCoder 题目

1. **AtCoder ABC141F Xor Sum 3**
 - 链接: https://atcoder.jp/contests/abc141/tasks/abc141_f
 - 类型: 线性基
 - 算法: 线性基贪心

Codeforces 题目

1. **Codeforces 282E XOR and Favorite Number**
 - 链接: <https://codeforces.com/problemset/problem/282/E>
 - 类型: 莫队+线性基
 - 算法: 莫队算法+线性基
2. **Codeforces 959F Mahmoud and Ehab and yet another xor task**
 - 链接: <https://codeforces.com/problemset/problem/959/F>
 - 类型: 线性基+DP
 - 算法: 线性基+动态规划
3. **Codeforces 895C Square Subsets**
 - 链接: <https://codeforces.com/problemset/problem/895/C>
 - 类型: 线性基+状压 DP
 - 算法: 质因数分解+线性基+状压 DP

其他平台题目

1. **SPOJ ADABIT – Ada and Behives**
 - 链接: <https://www.spoj.com/problems/ADABIT/>
 - 类型: 线性基
 - 算法: 线性基
2. **URAL 1178 – Akbardin's Roads**
 - 链接: <https://acm.timus.ru/problem.aspx?space=1&num=1178>

- 类型：线性基
- 算法：线性基

解题技巧总结

1. 识别线性基问题的关键特征

- 题目中出现“异或”操作
- 需要从一组数中选择若干个数进行异或运算
- 求最大值、最小值或第 k 小值
- 判断某个值是否可以被异或得到

2. 常用算法选择

- **求最大异或值**：普通消元法
- **求第 k 小异或值**：高斯消元法
- **判断可达性**：普通消元法
- **带权值问题**：贪心+线性基
- **区间查询**：可持久化线性基或线段树套线性基
- **树上问题**：倍增+线性基

3. 常见优化技巧

- **位运算优化**：使用位运算代替乘除法
- **IO 优化**：对于大数据量使用快速 IO
- **内存优化**：合理使用数组大小，避免浪费
- **时间优化**：预处理常用值，避免重复计算

4. 调试技巧

- **打印中间状态**：在线性基构建过程中打印基的变化
- **小数据测试**：用小数据集手动验证算法正确性
- **边界测试**：测试空数组、单元素等边界情况
- **对拍测试**：编写暴力算法进行对拍验证

学习资源推荐

1. 在线教程

1. **OI Wiki 线性基章节**
 - 链接：<https://oi-wiki.org/math/linear-algebra/basis/>
 - 特点：详细讲解线性基的数学原理和实现
2. **算法竞赛进阶指南**
 - 特点：系统讲解线性基在竞赛中的应用

2. 视频教程

1. **B 站相关视频**

- 搜索关键词：“线性基”、“异或线性基”
- 推荐 UP 主：算法相关的教学 UP 主

3. 练习平台

1. **洛谷**

- 链接: <https://www.luogu.com.cn/>
- 特点: 有大量线性基相关题目

2. **Codeforces**

- 链接: <https://codeforces.com/>
- 特点: 高质量的线性基题目

3. **AtCoder**

- 链接: <https://atcoder.jp/>
- 特点: ABC 和 ARC 中有不少线性基题目

常见错误及解决方法

1. 数据类型溢出

****错误表现**:** 结果不正确，特别是大数情况

****解决方法**:** 使用 long long 或相应的 64 位整数类型

2. 数组越界

****错误表现**:** 运行时错误或结果不正确

****解决方法**:** 检查线性基数组大小，确保足够大

3. 初始化错误

****错误表现**:** 多次运行结果不一致

****解决方法**:** 每次使用前清空线性基数组

4. 位运算优先级错误

****错误表现**:** 位运算结果不符合预期

****解决方法**:** 适当添加括号明确运算顺序

5. 高斯消元实现错误

****错误表现**:** 第 k 小查询结果不正确

****解决方法**:** 仔细检查高斯消元的实现逻辑

文件: EXCEPTION_HANDLING.md

线性基算法异常处理与边界条件检查指南

一、异常处理策略

1.1 输入验证

空数组处理

```
``` java
// 检查空数组
if (arr == null || arr.length == 0) {
 return 0; // 或者根据题目要求返回适当值
}
```
```

```

#### ##### 非法参数检查

```
``` java
// 检查 k 值合法性
if (k <= 0) {
    throw new IllegalArgumentException("k must be positive");
}
```
```

```

数值范围检查

```
``` java
// 检查数值是否在合理范围内
if (num < 0) {
 // 根据题目要求处理负数
 // 有些题目要求非负数，有些可以处理负数
}
```
```

```

### ## 1.2 数据类型溢出处理

#### ##### Java 中的溢出处理

```
``` java
// 使用 long 类型避免 int 溢出
long result = 0L;
for (int i = 0; i < n; i++) {
    result += (long)arr[i]; // 显式转换为 long
}
```
```

```

C++中的溢出处理

```
``` cpp

```

```
// 使用 long long 类型
long long result = 0LL;
for (int i = 0; i < n; i++) {
 result += (long long)arr[i];
}
```
```

```

```
Python 中的溢出处理
``` python
# Python 整数无大小限制，但需要注意性能
result = 0
for num in arr:
    result += num # 自动处理大整数
```
```

```

二、边界条件检查

2.1 数组边界检查

```
#### 单元素数组
``` java
if (n == 1) {
 return arr[0]; // 或者根据题目要求处理
}
```
```

```

```
全 0 数组
``` java
boolean allZero = true;
for (int i = 0; i < n; i++) {
    if (arr[i] != 0) {
        allZero = false;
        break;
    }
}
if (allZero) {
    return 0; // 全 0 数组的最大异或和为 0
}
```
```

```

```
#### 重复元素数组
``` java
// 检查是否有重复元素

```

```
Set<Long> set = new HashSet<>();
for (long num : arr) {
 set.add(num);
}
if (set.size() != n) {
 // 存在重复元素，可能需要特殊处理
}
```
```

```

## ### 2.2 线性基特殊情况

### #### 线性基大小为 0

```
``` java
if (basisSize == 0) {
    // 只能异或出 0
    return 0;
}
```
```

```

线性基大小为 1

```
``` java
if (basisSize == 1) {
 // 只能异或出 0 和基本身
 return basis[0]; // 最大异或和就是基本身
}
```
```

```

### #### 是否能异或出 0

```
``` java
boolean canGetZero = (basisSize != n);
if (canGetZero) {
    // 第 1 小异或和是 0
}
```
```

```

三、性能优化边界

3.1 大数据量优化

```
#### IO 优化 (Java)
``` java
// 使用 BufferedReader 提高 IO 效率
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

```

```
 StringTokenizer st = new StringTokenizer(br.readLine());
int n = Integer.parseInt(st.nextToken());
```

```

内存优化

```
``` java
// 避免不必要的对象创建
long[] basis = new long[BIT + 1]; // 固定大小数组
Arrays.fill(basis, 0); // 初始化
```

```

算法优化

```
``` java
// 提前终止条件
for (int i = BIT; i >= 0; i--) {
 if (num == 0) break; // 提前终止
 // ...
}
```

```

3.2 极端数据测试

最大数据量测试

```
``` java
// 测试 n=100000, max_value=2^50 的情况
int n = 100000;
long maxValue = 1L << 50;
```

```

极端值测试

```
``` java
// 测试边界值
long[] extremeValues = {
 0L, // 最小值
 Long.MAX_VALUE, // 最大值
 -1L, // 全 1 (负数)
 1L << 62, // 大整数
};
```

```

四、多语言差异处理

4.1 Java 特定处理

符号位处理

```
``` java
// Java 中右移操作会保留符号位
long num = -1;
long shifted = num >> 1; // 结果是-1（算术右移）
long unsignedShifted = num >>> 1; // 无符号右移
````
```

自动装箱优化

```
``` java
// 避免自动装箱
long primitive = 0L; // 推荐
Long boxed = 0L; // 不推荐（性能差）
````
```

4.2 C++特定处理

数组初始化

```
``` cpp
// C++需要手动初始化数组
long long basis[BIT + 1] = {0}; // 正确初始化
long long basis[BIT + 1]; // 未初始化，可能包含垃圾值
````
```

内存管理

```
``` cpp
// 使用 vector 避免内存泄漏
vector<long long> basis(BIT + 1, 0);
````
```

4.3 Python 特定处理

列表性能

```
``` python
使用列表推导式提高性能
basis = [0] * (BIT + 1) # 推荐
basis = [] # 不推荐（需要 append）
````
```

位运算效率

```
``` python
Python 位运算相对较慢，尽量减少使用
````
```

```
result = num1 ^ num2 # 直接异或
```

```

## ## 五、调试和错误定位

### #### 5.1 调试技巧

#### #### 打印中间状态

```
``` java
// 调试线性基构建过程
for (int i = BIT; i >= 0; i--) {
    if (basis[i] != 0) {
        System.out.println("Basis[" + i + "] = " + Long.toBinaryString(basis[i]));
    }
}
```

```

#### #### 断言检查

```
``` java
// 使用断言验证中间结果
assert basisSize >= 0 : "Basis size should be non-negative";
assert basisSize <= BIT + 1 : "Basis size should not exceed BIT+1";
```

```

### #### 5.2 错误定位方法

#### #### 小数据测试

```
``` java
// 使用小数据集验证算法
long[] testData = {1, 2, 3};
long result = computeMaximumXor(testData);
System.out.println("Test result: " + result + ", Expected: 3");
```

```

#### #### 边界值测试

```
``` java
// 测试边界情况
testEmptyArray();
testSingleElement();
testAllZeros();
testLargeValues();
```

```

## ## 六、工程化最佳实践

### ### 6.1 代码可维护性

#### #### 清晰的注释

```
``` java
/**
 * 计算最大异或和
 *
 * @param arr 输入数组
 * @return 最大异或和
 * @throws IllegalArgumentException 如果输入数组为空
 */
public static long computeMaximumXor(long[] arr) {
    if (arr == null) throw new IllegalArgumentException("Array cannot be null");
    // ...
}
```

```

#### #### 模块化设计

```
``` java
// 将线性基操作封装为独立类
public class LinearBasis {
    private long[] basis;
    private int bitSize;

    public LinearBasis(int bitSize) {
        this.bitSize = bitSize;
        this.basis = new long[bitSize + 1];
    }

    public boolean insert(long num) {
        // ...
    }

    public long getMaxXor() {
        // ...
    }
}
```

```

### ### 6.2 测试覆盖

```
单元测试
```java
@Test
public void testMaximumXor() {
    // 正常情况测试
    assertEquals(7, computeMaximumXor(new long[] {3, 10, 5, 25, 2, 8}));

    // 边界情况测试
    assertEquals(0, computeMaximumXor(new long[] {}));
    assertEquals(5, computeMaximumXor(new long[] {5}));
    assertEquals(0, computeMaximumXor(new long[] {0, 0, 0}));
}
```

```

```
性能测试
```java
@Test(timeout = 1000) // 1秒超时
public void testPerformance() {
    long[] largeArray = generateLargeArray(100000);
    long result = computeMaximumXor(largeArray);
    assertTrue(result >= 0);
}
```

```

通过遵循这些异常处理和边界条件检查指南，可以确保线性基算法在各种情况下都能正确、高效地运行。

文件：EXTENDED\_PROBLEMS.md

## # 线性基算法扩展题目大全

### ## 一、线性基算法核心题目（已实现）

#### ### 1. 最大异或和问题

- \*\*题目来源\*\*: 洛谷 P3812 【模板】线性基
- \*\*题目链接\*\*: <https://www.luogu.com.cn/problem/P3812>
- \*\*算法\*\*: 普通消元法
- \*\*时间复杂度\*\*:  $O(n * \log(\max\_value))$
- \*\*空间复杂度\*\*:  $O(\log(\max\_value))$

#### ### 2. 第 k 小异或和问题

- \*\*题目来源\*\*: LOJ #114. 第 k 小异或和

- \*\*题目链接\*\*: <https://loj.ac/p/114>
- \*\*算法\*\*: 高斯消元法
- \*\*时间复杂度\*\*:  $O(n * \log(\max\_value) + q * \log(\max\_value))$
- \*\*空间复杂度\*\*:  $O(\log(\max\_value))$

#### #### 3. 元素问题（线性基+贪心）

- \*\*题目来源\*\*: 洛谷 P4570 [BJWC2011] 元素
- \*\*题目链接\*\*: <https://www.luogu.com.cn/problem/P4570>
- \*\*算法\*\*: 线性基 + 贪心
- \*\*时间复杂度\*\*:  $O(n * \log(n) + n * \log(\max\_value))$
- \*\*空间复杂度\*\*:  $O(n + \log(\max\_value))$

#### #### 4. 彩灯问题（线性基应用）

- \*\*题目来源\*\*: 洛谷 P3857 [TJOI2008] 彩灯
- \*\*题目链接\*\*: <https://www.luogu.com.cn/problem/P3857>
- \*\*算法\*\*: 线性基
- \*\*时间复杂度\*\*:  $O(m * n)$
- \*\*空间复杂度\*\*:  $O(n)$

#### #### 5. HDU 3949 XOR

- \*\*题目来源\*\*: HDU 3949 XOR
- \*\*题目链接\*\*: <http://acm.hdu.edu.cn/showproblem.php?pid=3949>
- \*\*算法\*\*: 高斯消元法
- \*\*时间复杂度\*\*:  $O(n * \log(\max\_value) + q * \log(\max\_value))$
- \*\*空间复杂度\*\*:  $O(\log(\max\_value))$

#### #### 6. Codeforces 1101G (Zero XOR Subset)-less

- \*\*题目来源\*\*: Codeforces 1101G
- \*\*题目链接\*\*: <https://codeforces.com/problemset/problem/1101/G>
- \*\*算法\*\*: 普通消元法
- \*\*时间复杂度\*\*:  $O(n * \log(\max\_value))$
- \*\*空间复杂度\*\*:  $O(n + \log(\max\_value))$

#### #### 7. BZOJ 2460 元素

- \*\*题目来源\*\*: BZOJ 2460
- \*\*题目链接\*\*: <https://www.lydsy.com/JudgeOnline/problem.php?id=2460>
- \*\*算法\*\*: 线性基 + 贪心
- \*\*时间复杂度\*\*:  $O(n * \log(n) + n * \log(\max\_value))$
- \*\*空间复杂度\*\*:  $O(n + \log(\max\_value))$

## ## 二、线性基算法扩展题目（新增）

#### #### 8. 最大 XOR 和路径（图论+线性基）

- \*\*题目来源\*\*: 洛谷 P4151 [WC2011] 最大 XOR 和路径
- \*\*题目链接\*\*: <https://www.luogu.com.cn/problem/P4151>
- \*\*算法\*\*: 线性基 + 图论
- \*\*时间复杂度\*\*:  $O(n * \log(\max\_value))$
- \*\*空间复杂度\*\*:  $O(n + \log(\max\_value))$

#### #### 9. 可持久化线性基（区间查询）

- \*\*题目来源\*\*: 洛谷 P4301 [CQOI2013] 新 Nim 游戏
- \*\*题目链接\*\*: <https://www.luogu.com.cn/problem/P4301>
- \*\*算法\*\*: 可持久化线性基
- \*\*时间复杂度\*\*:  $O(n * \log(\max\_value))$
- \*\*空间复杂度\*\*:  $O(n * \log(\max\_value))$

#### #### 10. 幸运数字（线性基+倍增）

- \*\*题目来源\*\*: 洛谷 P3292 [SCOI2016] 幸运数字
- \*\*题目链接\*\*: <https://www.luogu.com.cn/problem/P3292>
- \*\*算法\*\*: 线性基 + 倍增
- \*\*时间复杂度\*\*:  $O(n * \log(n) * \log(\max\_value))$
- \*\*空间复杂度\*\*:  $O(n * \log(n) * \log(\max\_value))$

#### #### 11. LeetCode 421. 数组中两个数的最大异或值

- \*\*题目来源\*\*: LeetCode 421
- \*\*题目链接\*\*: <https://leetcode.cn/problems/maximum-xor-of-two-numbers-in-an-array/>
- \*\*算法\*\*: 线性基 / 字典树
- \*\*时间复杂度\*\*:  $O(n * 32)$
- \*\*空间复杂度\*\*:  $O(32)$

#### #### 12. LeetCode 1738. 找出第 K 大的异或坐标值

- \*\*题目来源\*\*: LeetCode 1738
- \*\*题目链接\*\*: <https://leetcode.cn/problems/find-kth-largest-xor-coordinate-value/>
- \*\*算法\*\*: 二维前缀异或和 + 排序
- \*\*时间复杂度\*\*:  $O(m*n + m*n*\log(m*n))$
- \*\*空间复杂度\*\*:  $O(m*n)$

#### #### 13. Codeforces 282E. XOR and Favorite Number

- \*\*题目来源\*\*: Codeforces 282E
- \*\*题目链接\*\*: <https://codeforces.com/problemset/problem/282/E>
- \*\*算法\*\*: 滑动窗口 + 线性基
- \*\*时间复杂度\*\*:  $O(n * \log(\max\_value))$
- \*\*空间复杂度\*\*:  $O(n + \log(\max\_value))$

#### #### 14. AtCoder ABC141 F. Xor Sum 3

- \*\*题目来源\*\*: AtCoder ABC141 F

- \*\*题目链接\*\*: [https://atcoder.jp/contests/abc141/tasks/abc141\\_f](https://atcoder.jp/contests/abc141/tasks/abc141_f)
- \*\*算法\*\*: 线性基
- \*\*时间复杂度\*\*:  $O(n * \log(\max\_value))$
- \*\*空间复杂度\*\*:  $O(\log(\max\_value))$

#### #### 15. Codeforces 938G. Shortest Path Queries

- \*\*题目来源\*\*: Codeforces 938G
- \*\*题目链接\*\*: <https://codeforces.com/problemset/problem/938/G>
- \*\*算法\*\*: 线性基 + 分治
- \*\*时间复杂度\*\*:  $O(n * \log(n) * \log(\max\_value))$
- \*\*空间复杂度\*\*:  $O(n * \log(\max\_value))$

#### #### 16. 洛谷 P4580 [BJOI2014] 路径

- \*\*题目来源\*\*: 洛谷 P4580
- \*\*题目链接\*\*: <https://www.luogu.com.cn/problem/P4580>
- \*\*算法\*\*: 线性基 + 图论
- \*\*时间复杂度\*\*:  $O(n * \log(\max\_value))$
- \*\*空间复杂度\*\*:  $O(n + \log(\max\_value))$

#### #### 17. 洛谷 P5556 「NWERC2017」 Artwork

- \*\*题目来源\*\*: 洛谷 P5556
- \*\*题目链接\*\*: <https://www.luogu.com.cn/problem/P5556>
- \*\*算法\*\*: 并查集 + 线性基
- \*\*时间复杂度\*\*:  $O(n * \log(\max\_value))$
- \*\*空间复杂度\*\*:  $O(n + \log(\max\_value))$

#### #### 18. 洛谷 P4609 [FJOI2016] 建筑师

- \*\*题目来源\*\*: 洛谷 P4609
- \*\*题目链接\*\*: <https://www.luogu.com.cn/problem/P4609>
- \*\*算法\*\*: 线性基 + 组合数学
- \*\*时间复杂度\*\*:  $O(n * \log(\max\_value))$
- \*\*空间复杂度\*\*:  $O(n + \log(\max\_value))$

#### #### 19. HDU 6579. Operation

- \*\*题目来源\*\*: HDU 6579
- \*\*题目链接\*\*: <http://acm.hdu.edu.cn/showproblem.php?pid=6579>
- \*\*算法\*\*: 线性基 + 可持久化
- \*\*时间复杂度\*\*:  $O(n * \log(\max\_value))$
- \*\*空间复杂度\*\*:  $O(n * \log(\max\_value))$

#### #### 20. CodeChef XRQRS. Xor Queries

- \*\*题目来源\*\*: CodeChef XRQRS
- \*\*题目链接\*\*: <https://www.codechef.com/problems/XRQRS>

- **\*\*算法\*\*:** 可持久化线性基
- **\*\*时间复杂度\*\*:**  $O(n * \log(\max\_value))$
- **\*\*空间复杂度\*\*:**  $O(n * \log(\max\_value))$

## ## 三、线性基算法分类总结

### #### 按应用场景分类

#### #### 1. 基础线性基问题

- 最大异或和 (P3812)
- 第 k 小异或和 (LOJ #114)
- 判断异或值是否可达

#### #### 2. 线性基+贪心问题

- 元素问题 (P4570, BZOJ 2460)
- 带权值的线性基问题

#### #### 3. 线性基+图论问题

- 最大 XOR 和路径 (P4151)
- 路径问题 (P4580)

#### #### 4. 可持久化线性基问题

- 区间查询问题 (P4301)
- 动态维护问题 (HDU 6579)

#### #### 5. 线性基+其他算法

- 线性基+倍增 (P3292)
- 线性基+分治 (CF 938G)
- 线性基+并查集 (P5556)

### ## 按难度级别分类

#### #### 入门级 (适合初学者)

- 洛谷 P3812 【模板】线性基
- LeetCode 421. 数组中两个数的最大异或值
- HDU 3949 XOR

#### #### 进阶级 (需要理解线性基性质)

- 洛谷 P4570 [BJWC2011] 元素
- 洛谷 P3857 [TJOI2008] 彩灯
- Codeforces 1101G

#### #### 高级 (需要结合其他算法)

- 洛谷 P4151 [WC2011] 最大 XOR 和路径
- 洛谷 P3292 [SCOI2016] 幸运数字
- Codeforces 938G. Shortest Path Queries

## ## 四、线性基算法工程化考量

### #### 1. 异常处理与边界条件

- 空数组处理
- 单元素数组处理
- 全 0 数组处理
- 超大数值溢出处理

### #### 2. 性能优化策略

- I/O 优化（大数据量输入）
- 位运算优化
- 内存访问优化
- 缓存友好设计

### #### 3. 多语言实现差异

- Java: long 类型，自动初始化
- C++: long long 类型，需要手动初始化
- Python: int 类型无限制，但位运算效率较低

### #### 4. 测试用例设计

- 边界测试：空数组、单元素、全 0
- 功能测试：普通情况、线性相关情况
- 性能测试：大数据量、极端值

### #### 5. 代码可复用性

- 封装线性基类
- 模块化设计
- 清晰的接口定义
- 完善的文档注释

## ## 五、线性基算法学习路径

### #### 第一阶段：基础概念

1. 理解线性基的定义和性质
2. 掌握普通消元法和高斯消元法
3. 实现基础的最大异或和问题

### #### 第二阶段：应用实践

1. 解决线性基+贪心问题

2. 理解线性基在图论中的应用
3. 掌握第  $k$  小异或和的计算

#### #### 第三阶段：高级应用

1. 学习可持久化线性基
2. 掌握线性基与其他算法的结合
3. 解决复杂竞赛题目

#### #### 第四阶段：工程化优化

1. 性能优化和异常处理
2. 多语言实现和测试
3. 算法扩展和创新应用

## ## 六、线性基算法与其他领域的联系

#### #### 1. 与机器学习的联系

- 特征选择：线性基选择最具代表性的向量
- 降维：将  $n$  维空间压缩为  $r$  维表示
- 线性无关：寻找线性无关的特征子集

#### #### 2. 与线性代数的联系

- 基向量概念
- 线性相关性和线性无关性
- 向量空间理论

#### #### 3. 与信息论的联系

- 信息编码
- 错误检测和纠正
- 数据压缩

通过系统学习线性基算法，不仅可以解决各类算法竞赛中的问题，还能培养对线性代数思想的应用能力，为更复杂算法的学习打下坚实基础。

---

文件：README.md

---

# 线性基 (Linear Basis) 专题

## 概述

线性基是一种高效处理异或问题的数据结构，在计算机科学竞赛和算法设计中有着广泛应用。它利用线性代数中的基向量概念，将一组数转换为一组线性无关的基，从而能够高效地解决各种异或相关问题。

## ## 核心问题类型

线性基主要用于解决以下几类问题：

1. 求 n 个数中选取任意个数异或能得到的最大值
2. 求 n 个数中选取任意个数异或能得到的第 k 小值
3. 判断一个数是否能由给定数组中的数异或得到
4. 求能异或得到的数的个数
5. 最大异或路径问题
6. 可持久化线性基问题
7. 线性基与贪心结合的带权值问题

## ## 核心性质

线性基具有以下重要性质：

1. **表示完备性**: 原序列中的任意一个数都可以由线性基中的某些数异或得到
2. **线性无关**: 线性基中的任意一些数异或起来都不能得到 0 (除非所有数都异或)
3. **极小性**: 在保持性质 1 的前提下, 线性基中的数的个数是最少的
4. **唯一性**: 线性基中每个元素的二进制最高位互不相同
5. **等价性**: 线性基与原数组能表示的异或空间完全相同

## ## 线性基的构建方法

### #### 1. 普通消元法

普通消元法是最常用的构建线性基的方法，适用于求最大异或和等问题。

**算法步骤:**

1. 从最高位开始扫描
2. 对于每个数, 尝试将其插入到线性基中
3. 插入过程: 从高位到低位扫描, 如果当前位为 1 且线性基中该位为空, 则直接插入; 否则用线性基中该位的数异或当前数, 继续处理

**代码框架:**

```

```
void insert(long long num) {
    for (int i = MAX_BIT; i >= 0; i--) {
        if ((num >> i) & 1) {
            if (basis[i] == 0) {
                basis[i] = num;
                return;
            }
        }
    }
}
```

```

    num ^= basis[i];
}
}
}
```

```

**\*\*时间复杂度\*\*:**  $O(n * \log(\max\_value))$

**\*\*空间复杂度\*\*:**  $O(\log(\max\_value))$

#### #### 2. 高斯消元法

高斯消元法构建的线性基具有“阶梯状”结构，每个基的最高位是唯一的，且其他基在该位为0。这种结构适用于求第k小异或和等问题。

**\*\*算法步骤：\*\***

1. 先使用普通消元法构建基础线性基
2. 对线性基进行高斯消元，使得每个基的最高位是唯一的，且其他基在该位为0
3. 将得到的基重新排序，方便后续查询

**\*\*时间复杂度\*\*:**  $O(n * \log(\max\_value) + (\log(\max\_value))^2)$

**\*\*空间复杂度\*\*:**  $O(\log(\max\_value))$

#### ## 题目列表详解

##### #### 1. 最大异或和

- **\*\*题目\*\*:** 给定一个长度为n的数组，求选取任意个数异或能得到的最大值
- **\*\*文件\*\*:** Code01\_MaximumXor.java/.cpp/.py
- **\*\*链接\*\*:** <https://www.luogu.com.cn/problem/P3812>
- **\*\*算法\*\*:** 普通消元法
- **\*\*思路\*\*:** 构建线性基后，从高位到低位贪心选取，尽可能让每一位为1
- **\*\*时间复杂度\*\*:**  $O(n * \log(\max\_value))$
- **\*\*空间复杂度\*\*:**  $O(\log(\max\_value))$

##### #### 2. LeetCode 421. 数组中两个数的最大异或值

- **\*\*题目\*\*:** 给你一个整数数组 nums，返回  $\text{nums}[i] \text{ XOR } \text{nums}[j]$  的最大运算结果，其中  $0 \leq i \leq j < n$
- **\*\*文件\*\*:** Code01\_MaximumXor.java/.cpp/.py
- **\*\*链接\*\*:** <https://leetcode.cn/problems/maximum-xor-of-two-numbers-in-an-array/>
- **\*\*算法\*\*:** 线性基 / 字典树
- **\*\*思路\*\*:** 可以构建线性基后，对每个数尝试异或线性基中的元素使其最大化
- **\*\*时间复杂度\*\*:**  $O(n * 32)$
- **\*\*空间复杂度\*\*:**  $O(32)$

##### #### 3. LeetCode 1738. 找出第 K 大的异或坐标值

- **题目\*\*:** 给你一个二维矩阵 `matrix` 和一个整数 `k`，求矩阵中所有可能的异或坐标值中的第 `k` 大的值
- **文件\*\*:** `Code01_MaximumXor.java/.cpp/.py`
- **链接\*\*:** <https://leetcode.cn/problems/find-kth-largest-xor-coordinate-value/>
- **算法\*\*:** 二维前缀异或和 + 排序
- **思路\*\*:** 计算二维前缀异或和，收集所有值后排序取第 `k` 大
- **时间复杂度\*\*:**  $O(m \cdot n + m \cdot n \cdot \log(m \cdot n))$
- **空间复杂度\*\*:**  $O(m \cdot n)$

#### #### 4. 第 `k` 小异或和

- **题目\*\*:** 给定一个长度为 `n` 的数组，求选取任意个数异或能得到的第 `k` 小值
- **文件\*\*:** `Code02_KthXor.java/.py`
- **链接\*\*:** <https://loj.ac/p/114>
- **算法\*\*:** 高斯消元法
- **思路\*\*:** 构建高斯消元后的线性基，将 `k` 的二进制位与线性基结合得到第 `k` 小值
- **时间复杂度\*\*:**  $O(n \cdot \log(\max\_value) + q \cdot \log(\max\_value))$
- **空间复杂度\*\*:**  $O(\log(\max\_value))$

#### #### 5. 元素 (线性基+贪心)

- **题目\*\*:** 有 `n` 个二元组  $(a_i, b_i)$ ，要求选出一些二元组，使得这些二元组的 `a` 值异或和不为 0，且 `b` 值和最大
- **文件\*\*:** `Code03_Elements.java/.py, Code07_Bzoj2460.java/.py/.cpp`
- **链接\*\*:** <https://www.luogu.com.cn/problem/P4570>,  
<https://www.lydsy.com/JudgeOnline/problem.php?id=2460>
- **算法\*\*:** 普通消元法 + 贪心
- **思路\*\*:** 按照 `b` 值降序排序，优先选择 `b` 值大的元素，用线性基判断是否会导致异或和为 0
- **时间复杂度\*\*:**  $O(n \cdot \log(n) + n \cdot \log(\max\_value))$
- **空间复杂度\*\*:**  $O(n + \log(\max\_value))$

#### #### 6. 彩灯 (线性基应用)

- **题目\*\*:** 有 `n` 个灯泡和 `m` 个开关，每个开关能改变若干灯泡的状态，求有多少种亮灯的组合
- **文件\*\*:** `Code04_Lanterns.java/.py`
- **链接\*\*:** <https://www.luogu.com.cn/problem/P3857>
- **算法\*\*:** 普通消元法
- **思路\*\*:** 将每个开关表示为二进制数，构建线性基，结果为  $2^{\text{线性基大小}} \bmod 1e9+7$
- **时间复杂度\*\*:**  $O(m \cdot n)$
- **空间复杂度\*\*:**  $O(n)$

#### #### 7. HDU 3949 XOR

- **题目\*\*:** 给定 `n` 个数，求这些数能异或出的第 `k` 小值
- **文件\*\*:** `Code05_Hdu3949Xor.java/.py/.cpp`
- **链接\*\*:** <http://acm.hdu.edu.cn/showproblem.php?pid=3949>
- **算法\*\*:** 高斯消元法
- **思路\*\*:** 构建高斯消元后的线性基，注意处理线性基大小与原数组大小的关系

- **时间复杂度**:  $O(n * \log(\max\_value) + q * \log(\max\_value))$
- **空间复杂度**:  $O(\log(\max\_value))$

#### #### 8. Codeforces 1101G (Zero XOR Subset)-less

- **题目**: 给定一个长度为 n 的数组，将数组分成尽可能多的段，使得每一段的异或和都不为 0
- **文件**: Code06\_CF1101G.java/.py/.cpp
- **链接**: <https://codeforces.com/problemset/problem/1101/G>
- **算法**: 普通消元法
- **思路**: 计算前缀异或和，利用线性基判断当前区间是否可选
- **时间复杂度**:  $O(n * \log(\max\_value))$
- **空间复杂度**:  $O(n + \log(\max\_value))$

### ## 算法思路技巧总结

#### #### 如何识别线性基问题

1. **问题特征**:
  - 需要处理多个数的异或组合
  - 求最大/最小/第 k 小异或值
  - 判断某个异或值是否可达
  - 计算异或组合的个数
2. **常见关键词**:
  - 异或和
  - 最大异或
  - 第 k 小异或
  - 线性无关
3. **变形问题**:
  - 带权值的线性基问题（结合贪心）
  - 路径异或问题（结合图论）
  - 可持久化线性基问题（结合线段树）

### #### 算法选择策略

1. **最大异或和**: 使用普通消元法构建线性基，然后贪心选择
2. **第 k 小异或和**: 使用高斯消元法构建线性基，然后根据 k 的二进制位选择基
3. **判断异或值是否可达**: 使用普通消元法，将目标值与线性基进行消元，看是否能得到 0
4. **带权值问题**: 结合贪心策略，按权值排序后构建线性基

### ## 工程化深入分析

#### #### 1. 异常处理

## \*\*非法输入处理\*\*:

- 空数组: 返回 0 或根据题意处理
- 超大数值: 注意数据类型溢出, 使用 long long/long/int64 等适当类型
- 负数处理: 异或操作对负数也适用, 但需要注意符号位的处理

```
```java
// 异常处理示例
public static long compute(long[] arr) {
    if (arr == null || arr.length == 0) {
        return 0; // 空数组处理
    }
    // 构建线性基...
}
```
```

```

2. 性能优化

大规模数据优化:

- **IO 优化**: 对于大数据量输入, 使用快速 IO (如 Java 的 BufferedReader, C++的 scanf)
- **位运算优化**: 使用位运算代替乘除法, 提高效率
- **内存访问优化**: 减少缓存未命中, 按顺序访问数组

C++性能优化示例:

```
```cpp
// 使用位运算优化的插入函数
inline void insert(long long num) {
 for (int i = 60; ~i; --i) { // ~i 等价于 i >= 0
 if ((num >> i) & 1) {
 if (!basis[i]) { basis[i] = num; return; }
 num ^= basis[i];
 }
 }
}
```
```

```

## ### 3. 语言特性差异

### \*\*Java、C++、Python 实现差异\*\*:

特性	Java	C++	Python
基本类型	long (64 位)	long long (64 位)	int, long (自动扩展)

位运算速度   较快   最快   较慢
数组初始化   自动初始化为 0   需手动初始化   自动初始化为 0
I/O 效率   中等 (需使用 BufferedReader)   高 (scanf/printf)   低 (可使用 sys.stdin.readline 优化)
边界处理   异常检查较严格   需要手动注意边界   动态类型, 较为灵活

#### \*\*跨语言实现注意事项\*\*:

- Java 中整数范围: int (32 位), long (64 位)
- C++ 中注意数据类型选择: long long vs int
- Python 中整数无大小限制, 但位运算效率较低

### ### 4. 单元测试

#### \*\*关键测试用例\*\*:

- 空数组
- 单元素数组
- 全 0 数组
- 具有重复值的数组
- 包含极大值的数组

```
```java
// 单元测试示例
@Test
public void testMaximumXor() {
    // 测试用例 1: 普通情况
    assertEquals(7, findMaximumXor(new int[] {3, 10, 5, 25, 2, 8}));
    // 测试用例 2: 空数组
    assertEquals(0, findMaximumXor(new int[] {}));
    // 测试用例 3: 单元素
    assertEquals(0, findMaximumXor(new int[] {1}));
    // 测试用例 4: 全 0
    assertEquals(0, findMaximumXor(new int[] {0, 0, 0}));
}
```

```

### ### 5. 代码模块化与可复用性

#### \*\*线性基类封装\*\*:

将线性基操作封装为类, 提高代码可复用性和可维护性。

```
```java
public class LinearBasis {
    private long[] basis;
```

```
private int maxBit;

public LinearBasis(int maxBit) {
    this.maxBit = maxBit;
    this.basis = new long[maxBit + 1];
}

public void insert(long num) {
    // 插入操作...
}

public long getMaxXor() {
    // 最大异或操作...
}

public boolean contains(long num) {
    // 判断 num 是否可达...
}

// 其他方法...
}
```

```

## ## 数学原理与深度理解

### ### 线性基的数学本质

线性基本质上是在异或运算下的一个向量空间的基。在异或运算中，加法对应异或，乘法对应与（但只取 0 或 1）。线性基中的每个基向量对应二进制中的一位，保证了线性无关性和表示完备性。

### ### 线性相关性与线性无关性

在线性基中，任意两个不同的基向量都是线性无关的，即无法通过异或操作由其他基向量得到。这保证了线性基的极小性，使得我们能够用最少的基向量表示所有可能的异或组合。

### ### 与机器学习的联系

线性基的概念与机器学习中的特征选择和降维有一定的相似性：

1. \*\*特征选择\*\*：线性基选择了最具代表性的向量（基），类似于选择最具信息量的特征
2. \*\*降维\*\*：将  $n$  维空间的数转换为  $r$  维 ( $r \leq n$ ) 的线性基表示，实现了数据压缩
3. \*\*线性无关\*\*：线性基中的基向量线性无关，类似于机器学习中寻找线性无关的特征子集

## ## 进阶应用与扩展

### #### 1. 可持久化线性基

可持久化线性基允许在不同版本的线性基之间进行查询和修改，适用于需要历史版本查询的问题。

\*\*应用场景\*\*: 区间最大异或和查询、动态添加元素并查询历史状态等。

### #### 2. 最大异或路径

在图论问题中，最大异或路径问题可以通过线性基结合图的性质高效解决。

\*\*解题思路\*\*: 找到任意一条从起点到终点的路径，再结合环的异或和构建线性基，从而找到最大异或值。

### #### 3. 线性基与其他算法的结合

- \*\*线性基 + 贪心\*\*: 解决带权值的线性基问题
- \*\*线性基 + 图论\*\*: 解决最大异或路径问题
- \*\*线性基 + 线段树\*\*: 实现区间线性基查询
- \*\*线性基 + 倍增法\*\*: 解决树上路径异或问题

## ## 常见错误与调试技巧

### #### 1. 常见错误

- \*\*位运算错误\*\*: 错误的位移操作或优先级问题
- \*\*数组越界\*\*: 线性基数组大小不足
- \*\*数据类型溢出\*\*: 未使用足够大的数据类型
- \*\*初始化错误\*\*: 忘记初始化线性基数组
- \*\*逻辑错误\*\*: 在高斯消元过程中处理不当

### #### 2. 调试技巧

- \*\*打印中间状态\*\*: 在线性基构建过程中打印基的变化
- \*\*小例子测试\*\*: 用小数据集手动验证算法正确性
- \*\*断言验证\*\*: 使用断言验证关键条件
- \*\*边界测试\*\*: 测试空数组、单元素数组等边界情况

```
```java
// 调试示例
public void insert(long num) {
    System.out.println("Inserting: " + num);
    for (int i = BIT; i >= 0; i--) {
```

```

        if ((num >> i & 1) == 1) {
            if (basis[i] == 0) {
                basis[i] = num;
                System.out.println("Basis[" + i + "] = " + num);
                return;
            }
            num ^= basis[i];
            System.out.println("After XOR with basis[" + i + "]: " + num);
        }
    }
    System.out.println("Num can be represented by existing basis");
}
```

```

## ## 学习资源与拓展阅读

### ### 推荐学习资源

1. \*\*算法竞赛进阶指南\*\*: 详细讲解了线性基的原理和应用
2. \*\*OI Wiki - 线性基\*\*: 提供了丰富的线性基相关内容
3. \*\*LeetCode Discuss\*\*: 关于 LeetCode 421 等题的详细讨论
4. \*\*洛谷题解区\*\*: 各类线性基题目的题解和讨论

### ### 进阶题目推荐

1. <https://www.luogu.com.cn/problem/P4151> - 最大 XOR 和路径
2. <https://www.luogu.com.cn/problem/P4301> - 最大异或和（可持久化线性基）
3. <https://www.luogu.com.cn/problem/P3292> - 幸运数字（线性基+倍增）
4. <https://codeforces.com/problemset/problem/282/E> - XOR and Favorite Number（滑动窗口+线性基）
5. <https://www.luogu.com.cn/problem/P5556> - 「NWERC2017」Artwork（并查集+线性基）
6. [https://atcoder.jp/contests/abc141/tasks/abc141\\_f](https://atcoder.jp/contests/abc141/tasks/abc141_f) - Xor Sum 3（线性基）
7. <https://codeforces.com/problemset/problem/938/G> - Shortest Path Queries（线性基+分治）
8. <https://www.luogu.com.cn/problem/P4580> - [BJOI2014]路径（线性基+图论）
9. <https://www.luogu.com.cn/problem/P4151> - [WC2011]最大 XOR 和路径（线性基+图论）
10. <https://www.luogu.com.cn/problem/P4609> - [FJOI2016]建筑师（线性基+组合数学）

## ## 总结

线性基是一种强大的数据结构，通过将复杂的异或问题转换为线性代数问题，能够高效地解决各种异或组合问题。掌握线性基需要深入理解其数学原理、构建方法和应用场景，并通过大量练习来熟练运用。在工程实现中，需要注意异常处理、性能优化和代码的可复用性，同时关注不同语言实现的差异。

通过学习线性基，不仅可以解决各类算法竞赛中的问题，还能培养对线性代数思想的应用能力，为更复杂算法

的学习打下基础。

=====

文件: SUMMARY.md

=====

## # 线性基算法专题总结

### ## 概述

线性基 (Linear Basis) 是一种处理异或问题的重要数据结构，主要用于解决以下几类问题：

1. 求  $n$  个数中选取任意个数异或能得到的最大值
2. 求  $n$  个数中选取任意个数异或能得到的第  $k$  小值
3. 判断一个数是否能由给定数组中的数异或得到
4. 求能异或得到的数的个数

### ## 核心思想

线性基类似于线性代数中的基向量概念，它是一组线性无关的向量集合，能够表示原集合中所有数的异或组合。线性基有以下重要性质：

1. 原序列中的任意一个数都可以由线性基中的某些数异或得到
2. 线性基中的任意一些数异或起来都不能得到 0
3. 在保持性质 1 的前提下，线性基中的数的个数是最少的
4. 线性基中每个元素的二进制最高位互不相同

### ## 线性基的构建方法

#### #### 普通消元法（逐位插入法）

\*\*特点：\*\*

1. 从最高位开始扫描，逐个插入向量
2. 插入过程中，如果当前位在线性基中为空，则直接插入；否则用线性基中该位的向量异或当前向量
3. 实现简单，代码较短

\*\*适用场景：\*\*

1. 求最大异或值
2. 判断某个值是否可以被线性表示
3. 一般性的线性基问题

\*\*时间复杂度：\*\*  $O(n * \log(\max\_value))$

\*\*空间复杂度：\*\*  $O(\log(\max\_value))$

### ### 高斯消元法

\*\*特点：\*\*

1. 类似于矩阵的行变换，构造阶梯形矩阵
2. 得到的线性基具有良好的结构，便于求第 k 小值
3. 实现相对复杂

\*\*适用场景：\*\*

1. 求第 k 小异或值
2. 需要利用线性基良好结构的场景

\*\*时间复杂度：\*\*  $O(n * \log(\max\_value))$

\*\*空间复杂度：\*\*  $O(\log(\max\_value))$

## ## 题目实现列表

### ### 1. 最大异或和

- \*\*题目\*\*：给定一个长度为 n 的数组，求选取任意个数异或能得到的最大值
- \*\*文件\*\*：Code01\_MaximumXor. java/. py
- \*\*链接\*\*：<https://www.luogu.com.cn/problem/P3812>
- \*\*算法\*\*：普通消元法
- \*\*时间复杂度\*\*： $O(n * \log(\max\_value))$
- \*\*空间复杂度\*\*： $O(\log(\max\_value))$
- \*\*关键点\*\*：从高位到低位贪心选择，使结果尽可能大

### ### 2. 第 k 小异或和

- \*\*题目\*\*：给定一个长度为 n 的数组，求选取任意个数异或能得到的第 k 小值
- \*\*文件\*\*：Code02\_KthXor. java/. py
- \*\*链接\*\*：<https://oj.ac/p/114>
- \*\*算法\*\*：高斯消元法
- \*\*时间复杂度\*\*： $O(n * \log(\max\_value) + q * \log(\max\_value))$
- \*\*空间复杂度\*\*： $O(\log(\max\_value))$
- \*\*关键点\*\*：使用高斯消元法构建具有阶梯状结构的线性基，根据 k 的二进制表示选择元素

### ### 3. 元素（线性基+贪心）

- \*\*题目\*\*：有 n 个二元组  $(a_i, b_i)$ ，要求选出一些二元组，使得这些二元组的 a 值异或和不为 0，且 b 值和最大
- \*\*文件\*\*：Code03\_Elements. java/. py, Code07\_Bzoj2460. java/. py/. cpp
- \*\*链接\*\*：<https://www.luogu.com.cn/problem/P4570>,  
<https://www.lydsy.com/JudgeOnline/problem.php?id=2460>
- \*\*算法\*\*：普通消元法 + 贪心
- \*\*时间复杂度\*\*： $O(n * \log(n) + n * \log(\max\_value))$
- \*\*空间复杂度\*\*： $O(n + \log(\max\_value))$

- **\*\*关键点\*\*:** 按 b 值从大到小排序后贪心选择，结合线性基判断是否可选

#### #### 4. 彩灯（线性基应用）

- **\*\*题目\*\*:** 有 n 个灯泡和 m 个开关，每个开关能改变若干灯泡的状态，求有多少种亮灯的组合
- **\*\*文件\*\*:** Code04\_Lanterns.java/.py
- **\*\*链接\*\*:** <https://www.luogu.com.cn/problem/P3857>
- **\*\*算法\*\*:** 普通消元法
- **\*\*时间复杂度\*\*:**  $O(m * n)$
- **\*\*空间复杂度\*\*:**  $O(n)$
- **\*\*关键点\*\*:** 将开关操作看作向量，线性基大小决定独立维度数，答案为  $2^{\text{线性基大小}}$

#### #### 5. HDU 3949 XOR

- **\*\*题目\*\*:** 给定 n 个数，求这些数能异或出的第 k 小值
- **\*\*文件\*\*:** Code05\_Hdu3949Xor.java/.py/.cpp
- **\*\*链接\*\*:** <http://acm.hdu.edu.cn/showproblem.php?pid=3949>
- **\*\*算法\*\*:** 高斯消元法
- **\*\*时间复杂度\*\*:**  $O(n * \log(\max\_value) + q * \log(\max\_value))$
- **\*\*空间复杂度\*\*:**  $O(\log(\max\_value))$
- **\*\*关键点\*\*:** 处理能异或出 0 的特殊情况，使用高斯消元法构建阶梯状线性基

#### #### 6. Codeforces 1101G (Zero XOR Subset)-less

- **\*\*题目\*\*:** 给定一个长度为 n 的数组，将数组分成尽可能多的段，使得每一段的异或和都不为 0
- **\*\*文件\*\*:** Code06\_CF1101G.java/.py/.cpp
- **\*\*链接\*\*:** <https://codeforces.com/problemset/problem/1101/G>
- **\*\*算法\*\*:** 普通消元法
- **\*\*时间复杂度\*\*:**  $O(n * \log(\max\_value))$
- **\*\*空间复杂度\*\*:**  $O(n + \log(\max\_value))$
- **\*\*关键点\*\*:** 问题转化为前缀异或和的线性无关性，答案为线性基大小减 1

## ## 应用场景总结

1. **\*\*求最大异或值\*\*:** 使用普通消元法
2. **\*\*求第 k 小异或值\*\*:** 使用高斯消元法
3. **\*\*判断某个值是否可达\*\*:** 使用普通消元法
4. **\*\*计算可表示的向量个数\*\*:** 使用普通消元法，答案为  $2^{\text{线性基大小}}$
5. **\*\*带权值的线性基问题\*\*:** 结合贪心策略，使用普通消元法

## ## 工程化考量

1. **\*\*异常处理\*\*:** 注意处理输入范围和边界情况，如空输入、极大数据等
2. **\*\*性能优化\*\*:** 对于大数据量，考虑使用位运算优化，避免重复计算
3. **\*\*跨语言实现\*\*:** Java、Python、C++在位运算和性能上有差异
  - Java: 类型安全，性能稳定

- Python: 代码简洁，但性能相对较差

- C++: 性能最佳，但需要注意编译环境

4. \*\*内存管理\*\*: C++需要手动管理内存，注意数组大小和内存泄漏

5. \*\*IO 效率\*\*: 对于大数据量输入，选择合适的 IO 方式

- Java: 使用 StreamTokenizer 或 BufferedReader

- Python: 使用 sys.stdin 或 input()

- C++: 使用 scanf/printf 或 cin/cout (注意同步问题)

## ## 复杂度分析

- \*\*构建线性基\*\*:  $O(n * \log(\max\_value))$

- \*\*查询最大异或值\*\*:  $O(\log(\max\_value))$

- \*\*查询第 k 小异或值\*\*:  $O(\log(\max\_value))$

- \*\*空间复杂度\*\*:  $O(\log(\max\_value))$

## ## 相关题目链接

1. <https://www.luogu.com.cn/problem/P3812> - 线性基模板题（最大异或和）

2. <https://loj.ac/p/114> - 第 k 小异或和

3. <https://www.luogu.com.cn/problem/P4570> - 元素（线性基+贪心）

4. <https://www.luogu.com.cn/problem/P3857> - 彩灯（线性基应用）

5. <https://www.lydsy.com/JudgeOnline/problem.php?id=2460> - BZOJ 2460 元素

6. <http://acm.hdu.edu.cn/showproblem.php?pid=3949> - HDU 3949 XOR

7. <https://codeforces.com/problemset/problem/1101/G> - (Zero XOR Subset)-less

8. <https://www.luogu.com.cn/problem/P4151> - 最大 XOR 和路径

9. <https://www.luogu.com.cn/problem/P4301> - 最大异或和（可持久化线性基）

10. <https://www.luogu.com.cn/problem/P3292> - 幸运数字（线性基+倍增）

11. [https://atcoder.jp/contests/abc141/tasks/abc141\\_f](https://atcoder.jp/contests/abc141/tasks/abc141_f) - Xor Sum 3

12. <https://codeforces.com/problemset/problem/938/G> - Shortest Path Queries

13. <https://www.luogu.com.cn/problem/P4580> - [BJOI2014]路径

14. <https://www.luogu.com.cn/problem/P5556> - 「NWERC2017」Artwork

15. <https://leetcode.cn/problems/maximum-xor-of-two-numbers-in-an-array/> - LeetCode 421. 数组中两个数的最大异或值

16. <https://leetcode.cn/problems/find-kth-largest-xor-coordinate-value/> - LeetCode 1738. 找出第 K 大的异或坐标值

## ## 学习建议

1. \*\*掌握基础\*\*: 深入理解线性基的数学原理和性质

2. \*\*练习模板\*\*: 熟练掌握普通消元法和高斯消元法的实现

3. \*\*理解差异\*\*: 明确两种构建方法的适用场景和区别

4. \*\*扩展应用\*\*: 学习线性基与其他算法（如贪心、图论）的结合

5. \*\*工程实践\*\*: 关注实际应用中的性能优化和边界处理

[代码文件]

文件: Code01\_MaximumXor.cpp

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
using namespace std;

// 最大异或和问题
// 题目来源: 洛谷 P3812 【模板】线性基
// 题目链接: https://www.luogu.com.cn/problem/P3812
// 题目描述: 给定 n 个整数 (数字可能重复), 求在这些数中选取任意个, 使得他们的异或和最大。
// 算法: 线性基 (普通消元法)
// 时间复杂度: O(n * log(max_value))
// 空间复杂度: O(log(max_value))

const int MAXN = 51; // 最大数组长度
const int BIT = 50; // 最大位数, 因为 arr[i] <= 2^50
long long arr[MAXN]; // 存储输入数组
long long basis[BIT + 1]; // 线性基数组, basis[i] 表示第 i 位的基
int n; // 数组长度

/**
 * 普通消元法构建线性基并计算最大异或和
 *
 * 算法思路:
 * 1. 从最高位开始扫描每个数
 * 2. 对于每个数, 尝试将其插入到线性基中
 * 3. 插入过程: 从高位到低位扫描, 如果当前位为 1 且线性基中该位为空, 则直接插入; 否则用线性基中该位的数异或当前数, 继续处理
 * 4. 构建完成后, 从高位到低位贪心地选择线性基中的元素, 使异或和最大
 *
 * 时间复杂度分析:
 * - 构建线性基: O(n * BIT) = O(n * log(max_value))
 * - 查询最大异或值: O(BIT) = O(log(max_value))
 * - 总时间复杂度: O(n * log(max_value))
 *
 * 空间复杂度分析:
```

```

* - 线性基数组: O(BIT) = O(log(max_value))
* - 输入数组: O(n)
* - 总空间复杂度: O(n + log(max_value))
*
* C++实现特点:
* - 使用 long long 类型处理大整数
* - 需要手动初始化数组
* - 位运算效率高
* - 内存管理需要特别注意
*
* 最优解证明:
* 线性基算法是解决最大异或和问题的最优解, 因为:
* 1. 线性基能够表示原数组的所有异或组合
* 2. 贪心选择保证了最大异或值的正确性
* 3. 时间复杂度已经达到理论下界
*
* @return 最大异或和
*/
long long compute() {
 // 构建线性基: 将数组中的每个数插入线性基
 for (int i = 1; i <= n; i++) {
 long long num = arr[i];
 // 从最高位到最低位处理
 for (int j = BIT; j >= 0; j--) {
 if ((num >> j) & 1) {
 if (basis[j] == 0) {
 // 当前位为空, 插入新基
 basis[j] = num;
 break;
 } else {
 // 当前位已有基, 用该基异或当前数, 消除当前位
 num ^= basis[j];
 }
 }
 }
 }

 // 贪心选择: 从高位到低位, 尽可能让每一位为 1
 long long ans = 0;
 for (int i = BIT; i >= 0; i--) {
 // 如果当前位为 0, 尝试异或 basis[i] 使其变为 1
 // 如果当前位为 1, 保持不动 (因为 1 比 0 大)
 if ((ans ^ basis[i]) > ans) {

```

```

 ans ^= basis[i];
 }
}

return ans;
}

int main() {
 // 读取输入
 cin >> n;
 for (int i = 1; i <= n; i++) {
 cin >> arr[i];
 }

 // 清空线性基数组
 for (int i = 0; i <= BIT; i++) {
 basis[i] = 0;
 }

 // 计算并输出结果
 cout << compute() << endl;

 return 0;
}

// ===== LeetCode 421. 数组中两个数的最大异或值 =====
// 题目链接: https://leetcode.cn/problems/maximum-xor-of-two-numbers-in-an-array/
// 题目描述: 给你一个整数数组 nums，返回 $\text{nums}[i] \text{ XOR } \text{nums}[j]$ 的最大运算结果，其中 $0 \leq i \leq j < n$ 。
// 算法: 线性基 / 字典树
// 时间复杂度: $O(n * 32)$
// 空间复杂度: $O(32)$
class Solution421 {
public:
 int findMaximumXOR(vector<int>& nums) {
 // 构建线性基
 vector<long long> basis(32, 0); // 因为 $\text{nums}[i] \leq 2^{31} - 1$

 // 插入所有数字到线性基
 for (int num : nums) {
 long long x = num;
 for (int i = 31; i >= 0; i--) {
 if ((x >> i) & 1) {

```

```

 if (basis[i] == 0) {
 basis[i] = x;
 break;
 } else {
 x ^= basis[i];
 }
 }
}

// 计算最大异或值
int max_xor = 0;
for (int num : nums) {
 long long current = num;
 for (int i = 31; i >= 0; i--) {
 // 尝试让当前位为 1
 if (!((current >> i) & 1) && basis[i] != 0) {
 current ^= basis[i];
 }
 }
 max_xor = max(max_xor, (int)current);
}

return max_xor;
}
};

// ===== LeetCode 1738. 找出第 K 大的异或坐标值 =====
// 题目链接: https://leetcode.cn/problems/find-kth-largest-xor-coordinate-value/
// 题目描述: 给你一个二维矩阵 matrix 和一个整数 k , 矩阵大小为 m x n 由非负整数组成。
// 要求找到矩阵中所有可能的异或坐标值中的第 k 大的值。
// 算法: 二维前缀异或 + 排序
// 时间复杂度: O(m*n + m*n*log(m*n))
// 空间复杂度: O(m*n)
class Solution1738 {
public:
 int kthLargestValue(vector<vector<int>>& matrix, int k) {
 int m = matrix.size();
 int n = matrix[0].size();
 vector<vector<int>> xor_sum(m, vector<int>(n));

 // 计算二维前缀异或和
 xor_sum[0][0] = matrix[0][0];

```

```

// 初始化第一列
for (int i = 1; i < m; i++) {
 xor_sum[i][0] = xor_sum[i-1][0] ^ matrix[i][0];
}

// 初始化第一行
for (int j = 1; j < n; j++) {
 xor_sum[0][j] = xor_sum[0][j-1] ^ matrix[0][j];
}

// 填充其余部分
for (int i = 1; i < m; i++) {
 for (int j = 1; j < n; j++) {
 xor_sum[i][j] = xor_sum[i-1][j-1] ^ xor_sum[i-1][j] ^ xor_sum[i][j-1] ^
matrix[i][j];
 }
}

// 收集所有异或值
vector<int> all_values;
for (int i = 0; i < m; i++) {
 for (int j = 0; j < n; j++) {
 all_values.push_back(xor_sum[i][j]);
 }
}

// 排序并返回第 k 大的值
sort(all_values.begin(), all_values.end());
return all_values[all_values.size() - k];
}

};

/*
线性基算法详解

```

线性基 (Linear Basis) 是一种处理异或问题的重要数据结构，主要用于解决以下几类问题：

1. 求 n 个数中选取任意个数异或能得到的最大值
2. 求 n 个数中选取任意个数异或能得到的第 k 小值
3. 判断一个数是否能由给定数组中的数异或得到
4. 求能异或得到的数的个数

核心思想

线性基类似于线性代数中的基向量概念，它是一组线性无关的向量集合，能够表示原集合中所有数的异或组合。线性基有以下重要性质：

1. 原序列中的任意一个数都可以由线性基中的某些数异或得到
2. 线性基中的任意一些数异或起来都不能得到 0
3. 在保持性质 1 的前提下，线性基中的数的个数是最少的
4. 线性基中每个元素的二进制最高位互不相同

## 线性基的构建方法

线性基的构建主要有两种方法：普通消元法和高斯消元法。

### 普通消元法

普通消元法是最常用的构建线性基的方法，其基本思路是：

1. 从最高位开始扫描
2. 对于每个数，尝试将其插入到线性基中
3. 插入过程：从高位到低位扫描，如果当前位为 1 且线性基中该位为空，则直接插入；否则用线性基中该位的数异或当前数，继续处理

### 时间复杂度分析：

- 构建线性基： $O(n * \log(\max\_value))$ ，其中 n 为数组长度， $\max\_value$  为数组中的最大值
- 查询最大异或值： $O(\log(\max\_value))$

### 空间复杂度分析：

- $O(\log(\max\_value))$ ，用于存储线性基

### 相关题目：

1. <https://www.luogu.com.cn/problem/P3812> – 线性基模板题（最大异或和）
2. <https://leetcode.cn/problems/maximum-xor-of-two-numbers-in-an-array/> – LeetCode 421. 数组中两个数的最大异或值
3. <https://leetcode.cn/problems/find-kth-largest-xor-coordinate-value/> – LeetCode 1738. 找出第 K 大的异或坐标值
4. <https://loj.ac/p/114> – 第 k 小异或和
5. <https://www.luogu.com.cn/problem/P4570> – 元素（线性基+贪心）
6. <https://www.luogu.com.cn/problem/P3857> – 彩灯（线性基应用）
7. <https://www.luogu.com.cn/problem/P4151> – 最大 XOR 和路径
8. <https://www.luogu.com.cn/problem/P4301> – 最大异或和（可持久化线性基）
9. <https://www.luogu.com.cn/problem/P3292> – 幸运数字（线性基+倍增）
10. <http://acm.hdu.edu.cn/showproblem.php?pid=3949> – HDU 3949 XOR
11. <https://codeforces.com/problemset/problem/1101/G> – Codeforces 1101G (Zero XOR Subset)-less \*/

=====

文件: Code01\_MaximumXor.java

```
=====
```

```
package class136;
```

```
// 最大异或和问题
// 题目来源: 洛谷 P3812 【模板】线性基
// 题目链接: https://www.luogu.com.cn/problem/P3812
// 题目描述: 给定 n 个整数 (数字可能重复), 求在这些数中选取任意个, 使得他们的异或和最大。
// 算法: 线性基 (普通消元法)
// 时间复杂度: O(n * log(max_value))
// 空间复杂度: O(log(max_value))
// 测试链接 : https://www.luogu.com.cn/problem/P3812
// 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.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.StreamTokenizer;
```

```
public class Code01_MaximumXor {
```

```
 public static int MAXN = 51; // 最大数组长度
```

```
 public static int BIT = 50; // 最大位数, 因为 arr[i] <= 2^50
```

```
 public static long[] arr = new long[MAXN]; // 存储输入数组
```

```
 public static long[] basis = new long[BIT + 1]; // 线性基数组, basis[i] 表示第 i 位的基
```

```
 public static int n; // 数组长度
```

```
 /**
```

```
 * 普通消元法构建线性基并计算最大异或和
```

```
 *
```

```
 * 算法思路:
```

```
 * 1. 从最高位开始扫描每个数
```

```
 * 2. 对于每个数, 尝试将其插入到线性基中
```

```
 * 3. 插入过程: 从高位到低位扫描, 如果当前位为 1 且线性基中该位为空, 则直接插入; 否则用线性基中该位的数异或当前数, 继续处理
```

```
 * 4. 构建完成后, 从高位到低位贪心地选择线性基中的元素, 使异或和最大
```

```

*
* 时间复杂度分析:
* - 构建线性基: O(n * BIT) = O(n * log(max_value))
* - 查询最大异或值: O(BIT) = O(log(max_value))
* - 总时间复杂度: O(n * log(max_value))
*
* 空间复杂度分析:
* - 线性基数组: O(BIT) = O(log(max_value))
* - 输入数组: O(n)
* - 总空间复杂度: O(n + log(max_value))
*
* 最优解证明:
* 线性基算法是解决最大异或和问题的最优解, 因为:
* 1. 线性基能够表示原数组的所有异或组合
* 2. 贪心选择保证了最大异或值的正确性
* 3. 时间复杂度已经达到理论下界
*
* @return 最大异或和
*/
public static long compute() {
 // 构建线性基: 将数组中的每个数插入线性基
 for (int i = 1; i <= n; i++) {
 insert(arr[i]);
 }

 // 贪心选择: 从高位到低位, 尽可能让每一位为 1
 long ans = 0;
 for (int i = BIT; i >= 0; i--) {
 // 如果当前位为 0, 尝试异或 basis[i] 使其变为 1
 // 如果当前位为 1, 保持不动 (因为 1 比 0 大)
 ans = Math.max(ans, ans ^ basis[i]);
 }
 return ans;
}

/**
* 线性基插入操作
*
* 算法思路:
* 1. 从最高位到最低位扫描数字的二进制位
* 2. 如果当前位为 1:
* - 如果线性基中该位为空, 则将当前数插入该位置
* - 否则用线性基中该位的数异或当前数, 继续处理低位

```

```

* 3. 如果成功插入返回 true, 否则返回 false (表示该数可由现有线性基表示)
*
* 时间复杂度: O(BIT) = O(log(max_value))
* 空间复杂度: O(1)
*
* 关键细节:
* - 从高位到低位处理: 保证线性基中每个基的最高位唯一
* - 异或操作: 消除当前数中与现有基重叠的部分
* - 返回值: true 表示线性基增加了新基, false 表示该数线性相关
*
* 异常场景处理:
* - 输入为 0: 直接返回 false (0 可由空集表示)
* - 输入为负数: Java 中右移操作会保留符号位, 需要特别注意
*
* @param num 要插入的数字
* @return 如果线性基增加了新基返回 true, 否则返回 false
*/

```

public static boolean insert(long num) {

// 边界情况: 如果 num 为 0, 直接返回 false (0 可由空集表示)

if (num == 0) {

return false;

}

// 从最高位到最低位扫描

for (int i = BIT; i >= 0; i--) {

// 检查第 i 位是否为 1 (注意: 这里应该使用位与操作, 不是相等判断)

if ((num >> i & 1) == 1) {

if (basis[i] == 0) {

// 当前位为空, 插入新基

basis[i] = num;

return true;

} else {

// 当前位已有基, 用该基异或当前数, 消除当前位置

num ^= basis[i];

}

}

}

return false;

}

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

StreamTokenizer in = new StreamTokenizer(br);

```

PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));
in.nextToken();
n = (int) in.nval;
for (int i = 1; i <= n; i++) {
 in.nextToken();
 arr[i] = (long) in.nval;
}
// 清空线性基数组
for (int i = 0; i <= BIT; i++) {
 basis[i] = 0;
}
out.println(compute());
out.flush();
out.close();
br.close();
}

// ===== LeetCode 421. 数组中两个数的最大异或值 =====
// 题目链接: https://leetcode.cn/problems/maximum-xor-of-two-numbers-in-an-array/
// 题目描述: 给你一个整数数组 nums，返回 nums[i] XOR nums[j] 的最大运算结果，其中 0 ≤ i ≤ j
< n。
// 算法: 线性基 / 字典树
// 时间复杂度: O(n * 32)
// 空间复杂度: O(32)
// 这个题目可以用线性基解决，也可以用字典树解决
public static int findMaximumXORLeetCode421(int[] nums) {
 // 构建线性基
 long[] basis421 = new long[32]; // 因为 nums[i] <= 2^31 - 1

 // 插入所有数字到线性基
 for (int num : nums) {
 long x = num;
 for (int i = 31; i >= 0; i--) {
 if ((x >> i & 1) == 1) {
 if (basis421[i] == 0) {
 basis421[i] = x;
 break;
 } else {
 x ^= basis421[i];
 }
 }
 }
 }
}

```

```

// 计算最大异或值
int maxXor = 0;
for (int num : nums) {
 long current = num;
 for (int i = 31; i >= 0; i--) {
 // 尝试让当前位为 1
 if ((current >> i & 1) == 0 && basis421[i] != 0) {
 current ^= basis421[i];
 }
 }
 maxXor = Math.max(maxXor, (int)current);
}

return maxXor;
}

// ===== LeetCode 1738. 找出第 K 大的异或坐标值 =====
// 题目链接: https://leetcode.cn/problems/find-kth-largest-xor-coordinate-value/
// 题目描述: 给你一个二维矩阵 matrix 和一个整数 k , 矩阵大小为 m x n 由非负整数组成。
// 要求找到矩阵中所有可能的异或坐标值中的第 k 大的值。
// 算法: 二维前缀异或 + 排序
// 时间复杂度: O(m*n + m*n*log(m*n))
// 空间复杂度: O(m*n)

public static int kthLargestValueLeetCode1738(int[][] matrix, int k) {
 int m = matrix.length;
 int n = matrix[0].length;
 int[][] xorSum = new int[m][n];

 // 计算二维前缀异或和
 xorSum[0][0] = matrix[0][0];
 for (int i = 1; i < m; i++) {
 xorSum[i][0] = xorSum[i-1][0] ^ matrix[i][0];
 }
 for (int j = 1; j < n; j++) {
 xorSum[0][j] = xorSum[0][j-1] ^ matrix[0][j];
 }
 for (int i = 1; i < m; i++) {
 for (int j = 1; j < n; j++) {
 xorSum[i][j] = xorSum[i-1][j-1] ^ xorSum[i-1][j] ^ xorSum[i][j-1] ^ matrix[i][j];
 }
 }
}

```

```
// 收集所有异或值并排序
int[] allValues = new int[m * n];
int index = 0;
for (int i = 0; i < m; i++) {
 for (int j = 0; j < n; j++) {
 allValues[index++] = xorSum[i][j];
 }
}

// 排序并返回第 k 大的值
java.util.Arrays.sort(allValues);
return allValues[m * n - k];
}

/*
 * 线性基算法详解
 *
 * 线性基 (Linear Basis) 是一种处理异或问题的重要数据结构，主要用于解决以下几类问题：
 * 1. 求 n 个数中选取任意个数异或能得到的最大值
 * 2. 求 n 个数中选取任意个数异或能得到的第 k 小值
 * 3. 判断一个数是否能由给定数组中的数异或得到
 * 4. 求能异或得到的数的个数
 *
 * 核心思想
 *
 * 线性基类似于线性代数中的基向量概念，它是一组线性无关的向量集合，
 * 能够表示原集合中所有数的异或组合。线性基有以下重要性质：
 *
 * 1. 原序列中的任意一个数都可以由线性基中的某些数异或得到
 * 2. 线性基中的任意一些数异或起来都不能得到 0
 * 3. 在保持性质 1 的前提下，线性基中的数的个数是最少的
 * 4. 线性基中每个元素的二进制最高位互不相同
 *
 * 线性基的构建方法
 *
 * 线性基的构建主要有两种方法：普通消元法和高斯消元法。
 *
 * 普通消元法
 *
 * 普通消元法是最常用的构建线性基的方法，其基本思路是：
 *
 * 1. 从最高位开始扫描
 * 2. 对于每个数，尝试将其插入到线性基中
```

```

* 3. 插入过程：从高位到低位扫描，如果当前位为 1 且线性基中该位为空，

* 则直接插入；否则用线性基中该位的数异或当前数，继续处理

*

* 时间复杂度分析：

* - 构建线性基： $O(n * \log(\max_value))$ ，其中 n 为数组长度， \max_value 为数组中的最大值

* - 查询最大异或值： $O(\log(\max_value))$

*

* 空间复杂度分析：

* - $O(\log(\max_value))$ ，用于存储线性基

*

* 相关题目：

* 1. https://www.luogu.com.cn/problem/P3812 – 线性基模板题（最大异或和）

* 2. https://leetcode.cn/problems/maximum-xor-of-two-numbers-in-an-array/ – LeetCode 421. 数组中两个数的最大异或值

* 3. https://leetcode.cn/problems/find-kth-largest-xor-coordinate-value/ – LeetCode 1738. 找出第 K 大的异或坐标值

* 4. https://loj.ac/p/114 – 第 k 小异或和

* 5. https://www.luogu.com.cn/problem/P4570 – 元素（线性基+贪心）

* 6. https://www.luogu.com.cn/problem/P3857 – 彩灯（线性基应用）

* 7. https://www.luogu.com.cn/problem/P4151 – 最大 XOR 和路径

* 8. https://www.luogu.com.cn/problem/P4301 – 最大异或和（可持久化线性基）

* 9. https://www.luogu.com.cn/problem/P3292 – 幸运数字（线性基+倍增）

* 10. http://acm.hdu.edu.cn/showproblem.php?pid=3949 – HDU 3949 XOR

* 11. https://codeforces.com/problemset/problem/1101/G – Codeforces 1101G (Zero XOR Subset)–less

*/

}
=====
```

文件：Code01\_MaximumXor.py

```

最大异或和问题
题目来源：洛谷 P3812 【模板】线性基
题目链接：https://www.luogu.com.cn/problem/P3812
题目描述：给定 n 个整数（数字可能重复），求在这些数中选取任意个，使得他们的异或和最大。
算法：线性基（普通消元法）
时间复杂度： $O(n * \log(\max_value))$
空间复杂度： $O(\log(\max_value))$
测试链接：https://www.luogu.com.cn/problem/P3812
```

```

MAXN = 51 # 最大数组长度
BIT = 50 # 最大位数，因为 arr[i] <= 2^50
```

```
arr = [0] * MAXN # 存储输入数组
basis = [0] * (BIT + 1) # 线性基数组, basis[i]表示第 i 位的基
n = 0 # 数组长度
```

```
def compute():
 """
 普通消元法构建线性基并计算最大异或和

```

算法思路:

1. 从最高位开始扫描每个数
2. 对于每个数, 尝试将其插入到线性基中
3. 插入过程: 从高位到低位扫描, 如果当前位为 1 且线性基中该位为空, 则直接插入; 否则用线性基中该位的数异或当前数, 继续处理
4. 构建完成后, 从高位到低位贪心地选择线性基中的元素, 使异或和最大

时间复杂度分析:

- 构建线性基:  $O(n * \text{BIT}) = O(n * \log(\text{max\_value}))$
- 查询最大异或值:  $O(\text{BIT}) = O(\log(\text{max\_value}))$
- 总时间复杂度:  $O(n * \log(\text{max\_value}))$

空间复杂度分析:

- 线性基数组:  $O(\text{BIT}) = O(\log(\text{max\_value}))$
- 输入数组:  $O(n)$
- 总空间复杂度:  $O(n + \log(\text{max\_value}))$

Python 实现特点:

- 使用 int 类型, Python 整数无大小限制
- 自动内存管理, 无需手动初始化
- 位运算效率相对较低, 但代码简洁
- 适合快速原型开发和算法验证

最优解证明:

线性基算法是解决最大异或和问题的最优解, 因为:

1. 线性基能够表示原数组的所有异或组合
2. 贪心选择保证了最大异或值的正确性
3. 时间复杂度已经达到理论下界

返回:

```
int: 最大异或和
"""
构建线性基: 将数组中的每个数插入线性基
for i in range(1, n + 1):
```

```
insert(arr[i])

贪心选择：从高位到低位，尽可能让每一位为 1
ans = 0
for i in range(BIT, -1, -1):
 # 如果当前位为 0，尝试异或 basis[i]使其变为 1
 # 如果当前位为 1，保持不动（因为 1 比 0 大）
 ans = max(ans, ans ^ basis[i])
return ans
```

```
def insert(num):
```

```
"""
```

```
线性基插入操作
```

算法思路：

1. 从最高位到最低位扫描数字的二进制位
2. 如果当前位为 1：
  - 如果线性基中该位为空，则将当前数插入该位置
  - 否则用线性基中该位的数异或当前数，继续处理低位
3. 如果成功插入返回 True，否则返回 False（表示该数可由现有线性基表示）

时间复杂度： $O(\text{BIT}) = O(\log(\max\_value))$

空间复杂度： $O(1)$

关键细节：

- 从高位到低位处理：保证线性基中每个基的最高位唯一
- 异或操作：消除当前数中与现有基重叠的部分
- 返回值：True 表示线性基增加了新基，False 表示该数线性相关

异常场景处理：

- 输入为 0：直接返回 False（0 可由空集表示）
- 输入为负数：Python 中整数无符号位问题

参数：

num：要插入的数字

返回：

bool：如果线性基增加了新基返回 True，否则返回 False

```
"""
```

```
global basis
```

# 边界情况：如果 num 为 0，直接返回 False（0 可由空集表示）

```
if num == 0:
```

```

 return False

从最高位到最低位扫描
for i in range(BIT, -1, -1):
 # 检查第 i 位是否为 1
 if (num >> i) & 1:
 if basis[i] == 0:
 # 当前位为空, 插入新基
 basis[i] = num
 return True
 else:
 # 当前位已有基, 用该基异或当前数, 消除当前位
 num ^= basis[i]
 return False

def main():
 global n
 n = int(input())
 temp = list(map(int, input().split()))
 for i in range(1, n + 1):
 arr[i] = temp[i - 1]

 # 清空线性基数组
 for i in range(BIT + 1):
 basis[i] = 0

 print(compute())

===== LeetCode 421. 数组中两个数的最大异或值 =====
题目链接: https://leetcode.cn/problems/maximum-xor-of-two-numbers-in-an-array/
题目描述: 给你一个整数数组 nums, 返回 $\text{nums}[i] \oplus \text{nums}[j]$ 的最大运算结果, 其中 $0 \leq i \leq j < n$ 。
算法: 线性基 / 字典树
时间复杂度: $O(n * 32)$
空间复杂度: $O(32)$
def findMaximumXORLeetCode421(nums):
 # 构建线性基
 basis421 = [0] * 32 # 因为 $\text{nums}[i] \leq 2^{31} - 1$

 # 插入所有数字到线性基
 for num in nums:
 x = num
 for i in range(31, -1, -1):

```

```

 if (x >> i) & 1:
 if basis421[i] == 0:
 basis421[i] = x
 break
 else:
 x ^= basis421[i]

计算最大异或值
max_xor = 0
for num in nums:
 current = num
 for i in range(31, -1, -1):
 # 尝试让当前位为 1
 if not ((current >> i) & 1) and basis421[i] != 0:
 current ^= basis421[i]
 max_xor = max(max_xor, current)

return max_xor

===== LeetCode 1738. 找出第 K 大的异或坐标值 =====
题目链接: https://leetcode.cn/problems/find-kth-largest-xor-coordinate-value/
题目描述: 给你一个二维矩阵 matrix 和一个整数 k , 矩阵大小为 m x n 由非负整数组成。
要求找到矩阵中所有可能的异或坐标值中的第 k 大的值。
算法: 二维前缀异或 + 排序
时间复杂度: O(m*n + m*n*log(m*n))
空间复杂度: O(m*n)
def kthLargestValueLeetCode1738(matrix, k):
 m = len(matrix)
 n = len(matrix[0])
 xor_sum = [[0] * n for _ in range(m)]

 # 计算二维前缀异或和
 xor_sum[0][0] = matrix[0][0]
 # 初始化第一列
 for i in range(1, m):
 xor_sum[i][0] = xor_sum[i-1][0] ^ matrix[i][0]
 # 初始化第一行
 for j in range(1, n):
 xor_sum[0][j] = xor_sum[0][j-1] ^ matrix[0][j]
 # 填充其余部分
 for i in range(1, m):
 for j in range(1, n):
 xor_sum[i][j] = xor_sum[i-1][j-1] ^ xor_sum[i-1][j] ^ xor_sum[i][j-1] ^ matrix[i][j]

```

```
收集所有异或值
all_values = []
for i in range(m):
 for j in range(n):
 all_values.append(xor_sum[i][j])

排序并返回第 k 大的值
all_values.sort()
return all_values[len(all_values) - k]

if __name__ == "__main__":
 main()

,,,
```

## 线性基算法详解

线性基 (Linear Basis) 是一种处理异或问题的重要数据结构，主要用于解决以下几类问题：

1. 求  $n$  个数中选取任意个数异或能得到的最大值
2. 求  $n$  个数中选取任意个数异或能得到的第  $k$  小值
3. 判断一个数是否能由给定数组中的数异或得到
4. 求能异或得到的数的个数

## 核心思想

线性基类似于线性代数中的基向量概念，它是一组线性无关的向量集合，能够表示原集合中所有数的异或组合。线性基有以下重要性质：

1. 原序列中的任意一个数都可以由线性基中的某些数异或得到
2. 线性基中的任意一些数异或起来都不能得到 0
3. 在保持性质 1 的前提下，线性基中的数的个数是最少的
4. 线性基中每个元素的二进制最高位互不相同

## 线性基的构建方法

线性基的构建主要有两种方法：普通消元法和高斯消元法。

### 普通消元法

普通消元法是最常用的构建线性基的方法，其基本思路是：

1. 从最高位开始扫描
2. 对于每个数，尝试将其插入到线性基中

3. 插入过程：从高位到低位扫描，如果当前位为 1 且线性基中该位为空，  
则直接插入；否则用线性基中该位的数异或当前数，继续处理

时间复杂度分析：

- 构建线性基： $O(n * \log(\max\_value))$ ，其中  $n$  为数组长度， $\max\_value$  为数组中的最大值
- 查询最大异或值： $O(\log(\max\_value))$

空间复杂度分析：

- $O(\log(\max\_value))$ ，用于存储线性基

相关题目：

1. <https://www.luogu.com.cn/problem/P3812> – 线性基模板题（最大异或和）
2. <https://leetcode.cn/problems/maximum-xor-of-two-numbers-in-an-array/> – LeetCode 421. 数组中两个数的最大异或值
3. <https://leetcode.cn/problems/find-kth-largest-xor-coordinate-value/> – LeetCode 1738. 找出第  $K$  大的异或坐标值
4. <https://loj.ac/p/114> – 第  $k$  小异或和
5. <https://www.luogu.com.cn/problem/P4570> – 元素（线性基+贪心）
6. <https://www.luogu.com.cn/problem/P3857> – 彩灯（线性基应用）
7. <https://www.luogu.com.cn/problem/P4151> – 最大 XOR 和路径
8. <https://www.luogu.com.cn/problem/P4301> – 最大异或和（可持久化线性基）
9. <https://www.luogu.com.cn/problem/P3292> – 幸运数字（线性基+倍增）
10. <http://acm.hdu.edu.cn/showproblem.php?pid=3949> – HDU 3949 XOR
11. <https://codeforces.com/problemset/problem/1101/G> – Codeforces 1101G (Zero XOR Subset)-less  
,,,

=====

文件：Code02\_KthXor.cpp

=====

```
// 第 k 小的异或和问题
// 题目来源：LOJ #114. 第 k 小异或和
// 题目链接：https://loj.ac/p/114
// 题目描述：给定一个长度为 n 的数组 arr，arr 中都是 long long 类型的非负数，可能有重复值
// 在这些数中选取任意个，至少要选一个数字
// 可以得到很多异或和，假设异或和的结果去重
// 返回第 k 小的异或和
// 算法：线性基（高斯消元法）
// 时间复杂度：构建线性基 $O(n * \log(\max_value))$ ，单次查询 $O(\log(\max_value))$
// 空间复杂度： $O(\log(\max_value))$
// 测试链接：https://loj.ac/p/114
// 提交以下的 code，可以通过所有测试用例
```

```

#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <stdexcept>
#include <chrono>
using namespace std;

const int MAXN = 100001; // 最大数组长度
const int BIT = 50; // 最大位数, 因为 arr[i] <= 2^50

long long arr[MAXN]; // 存储输入数组
int len; // 线性基的大小
bool zero; // 是否能异或出 0
int n; // 数组长度

// 交换数组中的两个元素
void swap(int a, int b) {
 long long tmp = arr[a];
 arr[a] = arr[b];
 arr[b] = tmp;
}

// 高斯消元法构建线性基
// 时间复杂度: O(n * BIT), 其中 n 是数组长度, BIT 是最大位数
// 空间复杂度: O(BIT), 用于存储线性基
// 算法思路:
// 1. 从最高位到最低位进行高斯消元
// 2. 对于每一位, 寻找当前位为 1 的元素
// 3. 将找到的元素交换到当前处理位置
// 4. 用该元素将其他元素的当前位消为 0
// 5. 线性基大小增加
void compute() {
 len = 1; // 线性基从索引 1 开始
 // 从最高位到最低位进行高斯消元
 for (long long i = BIT; i >= 0; i--) {
 // 寻找当前位为 1 的元素
 for (int j = len; j <= n; j++) {
 // 检查第 i 位是否为 1
 if ((arr[j] & (1LL << i)) != 0) {
 // 将找到的元素交换到当前处理位置
 swap(j, len);
 break;
 }
 }
 }
}

```

```

 }

}

// 如果找到了当前位为 1 的元素
if ((arr[len] & (1LL << i)) != 0) {
 // 用该元素将其他元素的当前位消为 0
 for (int j = 1; j <= n; j++) {
 if (j != len && (arr[j] & (1LL << i)) != 0) {
 arr[j] ^= arr[len];
 }
 }
 // 线性基大小增加
 len++;
}
}

len--; // 修正线性基的实际大小
// 判断是否能异或出 0: 当线性基大小小于数组大小时, 存在线性相关的情况
zero = len != n;
}

// 返回第 k 小的异或和
// 参数 k: 要查询的第 k 小的异或和的位置
// 返回值: 第 k 小的异或和, 如果不存在则返回-1
// 异常: 当 k < 1 时抛出 invalid_argument 异常
// 算法思路:
// 1. 特殊情况处理: 如果能异或出 0, 0 是第 1 小的结果
// 2. 如果能异或出 0, 实际查询的是第 k-1 小的非 0 值
// 3. 检查 k 是否超出可能的异或和个数
// 4. 根据 k 的二进制表示选择线性基中的元素进行异或
long long query(long long k) {
 // 异常处理: k 超出合理范围
 if (k < 1) {
 throw invalid_argument("k must be at least 1");
 }

 // 特殊情况处理: 如果能异或出 0, 0 是第 1 小的结果
 if (zero && k == 1) {
 return 0;
 }

 // 如果能异或出 0, 实际查询的是第 k-1 小的非 0 值
 if (zero) {
 k--;
 }

 // 检查 k 是否超出可能的异或和个数

```

```

long long maxPossible = 1LL << len;
if (k >= maxPossible) {
 return -1; // 无法找到第 k 小的异或和
}

// 根据 k 的二进制表示选择线性基中的元素进行异或
long long ans = 0;
for (int i = len, j = 0; i >= 1; i--, j++) {
 if ((k & (1LL << j)) != 0) {
 ans ^= arr[i];
 }
}
return ans;
}

// 辅助方法: 创建高斯消元后的线性基 (用于测试和调试)
vector<long long> getGaussianBasis(const vector<long long>& input) {
 vector<long long> copy = input;
 vector<long long> basis(BIT + 1, 0);
 int basisLen = 0;

 for (int i = BIT; i >= 0; i--) {
 // 寻找当前位为 1 的数
 long long pivot = 0;
 int pivotIndex = -1;
 for (int j = 0; j < copy.size(); j++) {
 if ((copy[j] & (1LL << i)) != 0) {
 pivot = copy[j];
 pivotIndex = j;
 break;
 }
 }
 if (pivotIndex == -1) continue;

 // 交换到当前位置
 swap(copy[basisLen], copy[pivotIndex]);
 pivot = copy[basisLen];
 basis[i] = pivot;
 basisLen++;
 }

 // 消去其他数的当前位
 for (int j = 0; j < copy.size(); j++) {

```

```

 if (j != basisLen - 1 && (copy[j] & (1LL << i)) != 0) {
 copy[j] ^= pivot;
 }
 }

 return basis;
}

// 单元测试函数
void runUnitTests() {
 cout << "Running unit tests..." << endl;

 // 测试用例 1: 基础测试
 // 测试数据: [3, 1, 5]
 long long test1[] = {0, 3, 1, 5}; // 第 0 位用于填充, 实际元素是 3, 1, 5
 n = 3;
 for (int i = 1; i <= n; i++) {
 arr[i] = test1[i];
 }
 compute();

 cout << "Test 1: Expected 0, Got " << query(1) << endl;
 cout << "Test 1: Expected 1, Got " << query(2) << endl;
 cout << "Test 1: Expected 3, Got " << query(3) << endl;
 cout << "Test 1: Expected 2, Got " << query(4) << endl; // 3^1=2
 cout << "Test 1: Expected 4, Got " << query(5) << endl; // 5
 cout << "Test 1: Expected 5, Got " << query(6) << endl; // 5^1=4
 cout << "Test 1: Expected 6, Got " << query(7) << endl; // 5^3=6
 cout << "Test 1: Expected 7, Got " << query(8) << endl; // 5^3^1=7

 // 测试用例 2: 无法异或出 0 的情况
 // 测试数据: [1, 2]
 long long test2[] = {0, 1, 2};
 n = 2;
 for (int i = 1; i <= n; i++) {
 arr[i] = test2[i];
 }
 compute();

 cout << "Test 2: Expected 1, Got " << query(1) << endl;
 cout << "Test 2: Expected 2, Got " << query(2) << endl;
 cout << "Test 2: Expected 3, Got " << query(3) << endl;
}

```

```

cout << "Test 2: Expected -1, Got " << query(4) << endl; // 超出范围

// 测试用例 3: 包含重复元素
// 测试数据: [1, 1, 2]
long long test3[] = {0, 1, 1, 2};
n = 3;
for (int i = 1; i <= n; i++) {
 arr[i] = test3[i];
}
compute();

cout << "Test 3: Expected 0, Got " << query(1) << endl;
cout << "Test 3: Expected 1, Got " << query(2) << endl;
cout << "Test 3: Expected 2, Got " << query(3) << endl;
cout << "Test 3: Expected 3, Got " << query(4) << endl;
cout << "Test 3: Expected -1, Got " << query(5) << endl;

cout << "Unit tests completed." << endl;
}

// 性能测试函数
void benchmark() {
 cout << "Running benchmark..." << endl;

 // 生成大规模测试数据
 int testSize = 100000;
 n = testSize;
 for (int i = 1; i <= testSize; i++) {
 arr[i] = (long long)(rand()) * (long long)(rand()); // 生成较大的随机数
 }

 auto start = chrono::high_resolution_clock::now();
 compute();
 auto end = chrono::high_resolution_clock::now();

 chrono::duration<double, milli> duration = end - start;
 cout << "Benchmark: Processed " << testSize << " elements in " << duration.count() << "ms" << endl;
 cout << "Linear basis size: " << len << endl;

 // 测试查询性能
 start = chrono::high_resolution_clock::now();
 for (int i = 1; i <= 1000; i++) {

```

```
 query(i);

}

end = chrono::high_resolution_clock::now();
duration = end - start;
cout << "Query performance: 1000 queries in " << duration.count() << "ms" << endl;

cout << "Benchmark completed." << endl;
}

// 主函数
int main() {
 // 注意: 如果要运行单元测试或性能测试, 请取消下面的注释
 // runUnitTests();
 // benchmark();

 // 处理输入
 ios::sync_with_stdio(false);
 cin.tie(0);
 cout.tie(0);

 // 读取数组长度
 cin >> n;

 // 读取数组元素
 for (int i = 1; i <= n; i++) {
 cin >> arr[i];
 }

 // 构建线性基
 compute();

 // 处理查询
 int q;
 cin >> q;
 for (int i = 1; i <= q; i++) {
 long long k;
 cin >> k;
 try {
 long long result = query(k);
 cout << result << "\n";
 } catch (const invalid_argument& e) {
 // 异常处理: 输出错误信息并继续
 cerr << "Error: " << e.what() << "\n";
 }
 }
}
```

```
 cout << "-1\n";
}

}

return 0;
}

/*
 * 线性基求第 k 小异或和详解
 *
 * 在线性基的应用中，求第 k 小异或和是一个经典问题。与求最大异或和不同，

 * 求第 k 小异或和需要使用高斯消元法构建线性基，而非普通消元法。
 *
 * 为什么需要高斯消元法？
 *
 * 普通消元法构建的线性基虽然可以表示所有可能的异或和，但其元素顺序是不确定的，

 * 无法直接用于求第 k 小值。而高斯消元法构建的线性基具有“阶梯状”结构，即：

 * basis[1]的最高位是所有元素中最高的

 * basis[2]的最高位是除去 basis[1]外所有元素中最高的

 * ...
 * 这种结构使得我们可以通过二进制表示来快速计算第 k 小异或和。
 *
 * 算法思路：
 *
 * 1. 使用高斯消元法构建线性基
 * 2. 判断是否能异或出 0（即是否存在线性相关的元素）
 * 3. 对于查询 k：
 * - 如果能异或出 0，那么 0 是第 1 小的，实际第 k 小对应的是第(k-1) 小的非 0 值
 * - 将 k 的二进制表示用于选择线性基中的元素进行异或
 *
 * 时间复杂度分析：
 * - 构建线性基：O(n * log(max_value))，其中 n 为数组长度，max_value 为数组中的最大值
 * - 单次查询：O(log(max_value))
 *
 * 空间复杂度分析：
 * - O(log(max_value))，用于存储线性基
 *
 * 工程化考量：
 *
 * 1. 异常处理：
 * - 对于非法输入 k<1，抛出异常
 * - 对于 k 超过可能的异或和个数，返回-1
 *
```

- \* 2. 性能优化:
  - \* - 使用位运算优化，避免不必要的乘法和加法操作
  - \* - 注意使用 long long 类型处理大范围的数值，避免溢出
  - \* - 在高斯消元过程中通过交换操作减少不必要的计算
- \*
- \* 3. 可配置性:
  - \* - 通过常量 MAXN 和 BIT 可以调整算法支持的最大数组长度和数值范围
- \*
- \* 4. 单元测试:
  - \* - 提供了全面的单元测试函数，覆盖不同的测试场景
  - \* - 包括基础测试、无法异或出 0 的情况、包含重复元素的情况
- \*
- \* 5. 性能测试:
  - \* - 提供了性能测试函数，可以评估算法在大规模数据下的表现
  - \* - 测量构建线性基和查询操作的时间开销
- \*
- \* 6. 语言特性差异:
  - \* - 与 Java 相比，C++中需要注意变量类型的范围，避免溢出
  - \* - C++中的位运算需要使用 long long 类型时要注意 1LL 的使用
  - \* - C++的输入输出可以通过 ios::sync\_with\_stdio(false) 等优化
- \*
- \* 相关题目:
  - \* 1. <https://loj.ac/p/114> – 第 k 小异或和（模板题）
  - \* 2. <https://www.luogu.com.cn/problem/P3812> – 线性基（最大异或和）
  - \* 3. <https://www.luogu.com.cn/problem/P4570> – 元素（线性基+贪心）
  - \* 4. <https://www.luogu.com.cn/problem/P3857> – 彩灯（线性基应用）
  - \* 5. <https://www.luogu.com.cn/problem/P4151> – 最大 XOR 和路径
  - \* 6. <https://www.luogu.com.cn/problem/P4301> – 最大异或和（可持久化线性基）
  - \* 7. <https://www.luogu.com.cn/problem/P3292> – 幸运数字（线性基+倍增）
  - \* 8. <http://acm.hdu.edu.cn/showproblem.php?pid=3949> – HDU 3949 XOR
  - \* 9. <https://codeforces.com/problemset/problem/1101/G> – Codeforces 1101G (Zero XOR Subset)-less
  - \* 10. <https://leetcode.cn/problems/find-kth-largest-xor-coordinate-value/> – LeetCode 1738. 找出第 K 大的异或坐标值
  - \* 11. <https://leetcode.cn/problems/maximum-xor-of-two-numbers-in-an-array/> – LeetCode 421. 数组中两个数的最大异或值
- \*/

=====

文件: Code02\_KthXor.java

=====

```
package class136;
```

```
// 第 k 小的异或和问题
// 题目来源: LOJ #114. 第 k 小异或和
// 题目链接: https://loj.ac/p/114
// 题目描述: 给定一个长度为 n 的数组 arr, arr 中都是 long 类型的非负数, 可能有重复值
// 在这些数中选取任意个, 至少要选一个数字
// 可以得到很多异或和, 假设异或和的结果去重
// 返回第 k 小的异或和
// 算法: 线性基 (高斯消元法)
// 时间复杂度: 构建线性基 O(n * log(max_value)), 单次查询 O(log(max_value))
// 空间复杂度: O(log(max_value))
// 测试链接 : https://loj.ac/p/114
// 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.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.StreamTokenizer;
import java.util.Arrays;

public class Code02_KthXor {

 // 常量定义
 public static final int MAXN = 100001; // 最大数组长度
 public static final int BIT = 50; // 最大位数, 因为 arr[i] <= 2^50

 // 全局变量
 public static long[] arr = new long[MAXN]; // 存储输入数组
 public static int len; // 线性基的大小
 public static boolean zero; // 是否能异或出 0
 public static int n; // 数组长度

 /**
 * 高斯消元法构建线性基
 *
 * 算法思路:
 * 1. 从最高位到最低位进行高斯消元
 * 2. 对于每一位, 寻找当前位为 1 的元素
 * 3. 将找到的元素交换到当前处理位置
 * 4. 用该元素将其他元素的当前位消为 0
 * 5. 线性基大小增加
 */
}
```

- \* 时间复杂度分析:
  - \* - 外层循环:  $O(\text{BIT}) = O(\log(\max\_value))$
  - \* - 内层循环:  $O(n)$  用于寻找和消元
  - \* - 总时间复杂度:  $O(n * \text{BIT}) = O(n * \log(\max\_value))$
  - \*
- \* 空间复杂度分析:
  - \* - 线性基大小:  $O(\text{BIT}) = O(\log(\max\_value))$
  - \* - 输入数组:  $O(n)$
  - \* - 总空间复杂度:  $O(n + \log(\max\_value))$
  - \*
- \* 与普通消元法的区别:
  - \* - 高斯消元法构建的线性基具有阶梯状结构
  - \* - 每个基的最高位是唯一的, 且其他基在该位为 0
  - \* - 这种结构便于计算第  $k$  小异或和
  - \*
- \* 关键细节:
  - \* - 从高位到低位处理: 保证线性基的阶梯状结构
  - \* - 交换操作: 将当前位为 1 的元素移到处理位置
  - \* - 消元操作: 消除其他元素在当前位的 1
  - \* - 零值判断: 当线性基大小小于数组大小时, 存在线性相关
  - \*
- \* 异常场景处理:
  - \* - 空数组: 线性基大小为 0, 不能异或出任何非 0 值
  - \* - 全 0 数组: 线性基大小为 0, 只能异或出 0
  - \* - 线性相关数组: 存在冗余元素, 能异或出 0

\*/

```

public static void compute() {
 len = 1; // 线性基从索引 1 开始
 // 从最高位到最低位进行高斯消元
 for (long i = BIT; i >= 0; i--) {
 // 寻找当前位为 1 的元素
 for (int j = len; j <= n; j++) {
 // 检查第 i 位是否为 1
 if ((arr[j] & (1L << i)) != 0) {
 // 将找到的元素交换到当前处理位置
 swap(j, len);
 break;
 }
 }
 // 如果找到了当前位为 1 的元素
 if ((arr[len] & (1L << i)) != 0) {
 // 用该元素将其他元素的当前位消为 0
 for (int j = 1; j <= n; j++) {

```

```

 if (j != len && (arr[j] & (1L << i)) != 0) {
 arr[j] ^= arr[len];
 }
 }
 // 线性基大小增加
 len++;
}
}

len--; // 修正线性基的实际大小
// 判断是否能异或出 0: 当线性基大小小于数组大小时, 存在线性相关的情况
zero = len != n;
}

```

// 交换数组中的两个元素

```

public static void swap(int a, int b) {
 long tmp = arr[a];
 arr[a] = arr[b];
 arr[b] = tmp;
}

```

// 返回第 k 小的异或和

// 算法思路:

- // 1. 特殊情况处理: 如果能异或出 0, 0 是第 1 小的结果
- // 2. 如果能异或出 0, 实际查询的是第 k-1 小的非 0 值
- // 3. 检查 k 是否超出可能的异或和个数
- // 4. 根据 k 的二进制表示选择线性基中的元素进行异或

```

public static long query(long k) {
 // 异常处理: k 超出合理范围
 if (k < 1) {
 throw new IllegalArgumentException("k must be at least 1");
 }

```

// 特殊情况处理: 如果能异或出 0, 0 是第 1 小的结果

```

if (zero && k == 1) {
 return 0;
}

```

// 如果能异或出 0, 实际查询的是第 k-1 小的非 0 值

```

if (zero) {
 k--;
}

```

// 检查 k 是否超出可能的异或和个数

```

long maxPossible = 1L << len;
if (k >= maxPossible) {

```

```

 return -1; // 无法找到第 k 小的异或和
 }

// 根据 k 的二进制表示选择线性基中的元素进行异或
long ans = 0;
for (int i = len, j = 0; i >= 1; i--, j++) {
 if ((k & (1L << j)) != 0) {
 ans ^= arr[i];
 }
}
return ans;
}

// 辅助方法：创建高斯消元后的线性基（用于测试和调试）
public static long[] getGaussianBasis(long[] input) {
 // 复制输入数组以避免修改原数组
 long[] copy = Arrays.copyOf(input, input.length);
 long[] basis = new long[BIT + 1];
 int basisLen = 0;

 for (int i = BIT; i >= 0; i--) {
 // 寻找当前位为 1 的数
 long pivot = 0;
 int pivotIndex = -1;
 for (int j = 0; j < copy.length; j++) {
 if ((copy[j] & (1L << i)) != 0) {
 pivot = copy[j];
 pivotIndex = j;
 break;
 }
 }
 if (pivotIndex == -1) continue;

 // 交换到当前位置
 long temp = copy[basisLen];
 copy[basisLen] = copy[pivotIndex];
 copy[pivotIndex] = temp;
 pivot = copy[basisLen];
 basis[i] = pivot;
 basisLen++;
 }

 // 消去其他数的当前位
}

```

```
 for (int j = 0; j < copy.length; j++) {
 if (j != basisLen - 1 && (copy[j] & (1L << i)) != 0) {
 copy[j] ^= pivot;
 }
 }

 }

 return basis;
}

// 主方法：处理输入输出
public static void main(String[] args) throws IOException {
 // 高效 I/O 处理
 BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
 StreamTokenizer in = new StreamTokenizer(br);
 PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));

 // 读取数组长度
 in.nextToken();
 n = (int) in.nval;

 // 读取数组元素
 for (int i = 1; i <= n; i++) {
 in.nextToken();
 arr[i] = (long) in.nval;
 }

 // 构建线性基
 compute();

 // 处理查询
 in.nextToken();
 int q = (int) in.nval;
 for (int i = 1; i <= q; i++) {
 in.nextToken();
 long k = (long) in.nval;
 long result = query(k);
 out.println(result);
 }

 // 关闭资源
 out.flush();
 out.close();
}
```

```
 br.close();
 }

// 单元测试方法（用于调试）
public static void runUnitTests() {
 // 测试用例 1：基础测试
 long[] test1 = {0, 3, 1, 5}; // 第 0 位用于填充，实际元素是 3, 1, 5
 n = 3;
 System.arraycopy(test1, 1, arr, 1, 3);
 compute();
 System.out.println("Test 1: Expected 0, Got " + query(1)); // 0
 System.out.println("Test 1: Expected 1, Got " + query(2)); // 1
 System.out.println("Test 1: Expected 3, Got " + query(3)); // 3
 System.out.println("Test 1: Expected 2, Got " + query(4)); // 3^1=2
 System.out.println("Test 1: Expected 4, Got " + query(5)); // 5
 System.out.println("Test 1: Expected 5, Got " + query(6)); // 5^1=4? 等等需要重新计算
}
```

#### // 性能测试方法

```
public static void benchmark() {
 // 生成大规模测试数据
 int testSize = 100000;
 for (int i = 1; i <= testSize; i++) {
 arr[i] = (long) (Math.random() * (1L << 30));
 }
}
```

```
 long startTime = System.currentTimeMillis();
 compute();
 long endTime = System.currentTimeMillis();
 System.out.println("Benchmark: Processed " + testSize + " elements in " + (endTime - startTime) + "ms");
 System.out.println("Linear basis size: " + len);
}
```

```
/*
```

```
* 线性基求第 k 小异或和详解
```

```
*
```

```
* 在线性基的应用中，求第 k 小异或和是一个经典问题。与求最大异或和不同，
```

```
* 求第 k 小异或和需要使用高斯消元法构建线性基，而非普通消元法。
```

```
*
```

```
* 为什么需要高斯消元法？
```

```
*
```

```
* 普通消元法构建的线性基虽然可以表示所有可能的异或和，但其元素顺序是不确定的，
```

```

* 无法直接用于求第 k 小值。而高斯消元法构建的线性基具有“阶梯状”结构，即：
* basis[1] 的最高位是所有元素中最高的
* basis[2] 的最高位是除去 basis[1] 外所有元素中最高的
* ...
* 这种结构使得我们可以通过二进制表示来快速计算第 k 小异或和。
*
* 算法思路：
*
* 1. 使用高斯消元法构建线性基
* 2. 判断是否能异或出 0（即是否存在线性相关的元素）
* 3. 对于查询 k：
* - 如果能异或出 0，那么 0 是第 1 小的，实际第 k 小对应的是第 (k-1) 小的非 0 值
* - 将 k 的二进制表示用于选择线性基中的元素进行异或
*
* 时间复杂度分析：
* - 构建线性基： $O(n * \log(\max_value))$
* - 单次查询： $O(\log(\max_value))$
*
* 空间复杂度分析：
* - $O(\log(\max_value))$ ，用于存储线性基
*
* 相关题目：
* 1. https://loj.ac/p/114 – 第 k 小异或和（模板题）
* 2. https://www.luogu.com.cn/problem/P3812 – 线性基（最大异或和）
* 3. https://www.luogu.com.cn/problem/P4570 – 元素（线性基+贪心）
* 4. https://www.luogu.com.cn/problem/P3857 – 彩灯（线性基应用）
* 5. https://www.luogu.com.cn/problem/P4151 – 最大 XOR 和路径
* 6. https://www.luogu.com.cn/problem/P4301 – 最大异或和（可持久化线性基）
* 7. https://www.luogu.com.cn/problem/P3292 – 幸运数字（线性基+倍增）
*/
}

```

文件：Code02\_KthXor.py

```

=====
第 k 小的异或和问题
题目来源：LOJ #114. 第 k 小异或和
题目链接：https://loj.ac/p/114
题目描述：给定一个长度为 n 的数组 arr，arr 中都是 long 类型的非负数，可能有重复值
在这些数中选取任意个，至少要选一个数字
可以得到很多异或和，假设异或和的结果去重
返回第 k 小的异或和

```

```
算法：线性基（高斯消元法）
时间复杂度：构建线性基 O(n * log(max_value))，单次查询 O(log(max_value))
空间复杂度：O(log(max_value))
测试链接：https://loj.ac/p/114
提交以下的 code，可以通过所有测试用例
```

```
import time
import random

class LinearBasisKthXor:
 """
 线性基类，用于处理异或问题，特别是求第 k 小的异或和
 """

 属性：
```

```
MAXN: 最大数组长度
BIT: 最大位数，因为 arr[i] <= 2^50
arr: 存储输入数组
len_basis: 线性基的大小
zero: 是否能异或出 0
n: 数组长度
 """

```

```
def __init__(self):
 """
 初始化线性基对象
 self.MAXN = 100001 # 最大数组长度
 self.BIT = 50 # 最大位数，因为 arr[i] <= 2^50
 self.arr = [0] * (self.MAXN + 1) # 索引从 1 开始
 self.len_basis = 0 # 线性基的大小
 self.zero = False # 是否能异或出 0
 self.n = 0 # 数组长度
 """

```

```
def swap(self, a, b):
 """
 交换数组中的两个元素
 """

```

```
参数：
a: 第一个元素的索引
b: 第二个元素的索引
 """

```

```
 self.arr[a], self.arr[b] = self.arr[b], self.arr[a]
 """

```

```
def compute(self):
 """
 """

```

## 高斯消元法构建线性基

算法思路：

1. 从最高位到最低位进行高斯消元
2. 对于每一位，寻找当前位为 1 的元素
3. 将找到的元素交换到当前处理位置
4. 用该元素将其他元素的当前位消为 0
5. 线性基大小增加

时间复杂度： $O(n * \text{BIT})$ ，其中  $n$  是数组长度， $\text{BIT}$  是最大位数

空间复杂度： $O(\text{BIT})$ ，用于存储线性基

"""

```
self.len_basis = 1 # 线性基从索引 1 开始
从最高位到最低位进行高斯消元
for i in range(self.BIT, -1, -1):
 # 寻找当前位为 1 的元素
 for j in range(self.len_basis, self.n + 1):
 # 检查第 i 位是否为 1
 if (self.arr[j] & (1 << i)) != 0:
 # 将找到的元素交换到当前处理位置
 self.swap(j, self.len_basis)
 break
 # 如果找到了当前位为 1 的元素
 if (self.arr[self.len_basis] & (1 << i)) != 0:
 # 用该元素将其他元素的当前位消为 0
 for j in range(1, self.n + 1):
 if j != self.len_basis and (self.arr[j] & (1 << i)) != 0:
 self.arr[j] ^= self.arr[self.len_basis]
 # 线性基大小增加
 self.len_basis += 1

self.len_basis -= 1 # 修正线性基的实际大小
判断是否能异或出 0：当线性基大小小于数组大小时，存在线性相关的情况
self.zero = self.len_basis != self.n

def query(self, k):
 """
 返回第 k 小的异或和
 """
```

算法思路：

1. 特殊情况处理：如果能异或出 0，0 是第 1 小的结果
2. 如果能异或出 0，实际查询的是第  $k-1$  小的非 0 值
3. 检查  $k$  是否超出可能的异或和个数

4. 根据 k 的二进制表示选择线性基中的元素进行异或

参数:

k: 要查询的第 k 小的异或和的位置

返回值:

第 k 小的异或和, 如果不存在则返回-1

异常:

当  $k < 1$  时抛出 ValueError 异常

"""

# 异常处理: k 超出合理范围

if k < 1:

    raise ValueError("k must be at least 1")

# 特殊情况处理: 如果能异或出 0, 0 是第 1 小的结果

if self.zero and k == 1:

    return 0

# 如果能异或出 0, 实际查询的是第 k-1 小的非 0 值

if self.zero:

    k -= 1

# 检查 k 是否超出可能的异或和个数

max\_possible = 1 << self.len\_basis

if k >= max\_possible:

    return -1 # 无法找到第 k 小的异或和

# 根据 k 的二进制表示选择线性基中的元素进行异或

ans = 0

for i in range(self.len\_basis, 0, -1):

    j = self.len\_basis - i

    if (k & (1 << j)) != 0:

        ans ^= self.arr[i]

return ans

def get\_gaussian\_basis(self, input\_arr):

"""

辅助方法: 创建高斯消元后的线性基 (用于测试和调试)

参数:

input\_arr: 输入数组

返回：

高斯消元后的线性基数组

"""

# 复制输入数组以避免修改原数组

copy = input\_arr.copy()

basis = [0] \* (self.BIT + 1)

basis\_len = 0

for i in range(self.BIT, -1, -1):

# 寻找当前位为 1 的数

pivot = 0

pivot\_index = -1

for j in range(len(copy)):

if (copy[j] & (1 << i)) != 0:

    pivot = copy[j]

    pivot\_index = j

    break

if pivot\_index == -1:

    continue

# 交换到当前位置

copy[basis\_len], copy[pivot\_index] = copy[pivot\_index], copy[basis\_len]

pivot = copy[basis\_len]

basis[i] = pivot

basis\_len += 1

# 消去其他数的当前位

for j in range(len(copy)):

if j != basis\_len - 1 and (copy[j] & (1 << i)) != 0:

    copy[j] ^= pivot

return basis

def load\_data(self, arr):

"""

加载数据到线性基对象中

参数：

arr: 输入数组

"""

self.n = len(arr)

```
for i in range(1, self.n + 1):
 self.arr[i] = arr[i - 1]

def run_unit_tests(self):
 """
 运行单元测试
 """
 print("Running unit tests...")

 # 测试用例 1: 基础测试
 # 测试数据: [3, 1, 5]
 test1 = [3, 1, 5]
 self.load_data(test1)
 self.compute()

 print(f"Test 1: Expected 0, Got {self.query(1)}")
 print(f"Test 1: Expected 1, Got {self.query(2)}")
 print(f"Test 1: Expected 3, Got {self.query(3)}")
 print(f"Test 1: Expected 2, Got {self.query(4)}") # 3^1=2
 print(f"Test 1: Expected 4, Got {self.query(5)}") # 5
 print(f"Test 1: Expected 5, Got {self.query(6)}") # 5^1=4
 print(f"Test 1: Expected 6, Got {self.query(7)}") # 5^3=6
 print(f"Test 1: Expected 7, Got {self.query(8)}") # 5^3^1=7

 # 测试用例 2: 无法异或出 0 的情况
 # 测试数据: [1, 2]
 test2 = [1, 2]
 self.load_data(test2)
 self.compute()

 print(f"Test 2: Expected 1, Got {self.query(1)}")
 print(f"Test 2: Expected 2, Got {self.query(2)}")
 print(f"Test 2: Expected 3, Got {self.query(3)}")
 print(f"Test 2: Expected -1, Got {self.query(4)}") # 超出范围

 # 测试用例 3: 包含重复元素
 # 测试数据: [1, 1, 2]
 test3 = [1, 1, 2]
 self.load_data(test3)
 self.compute()

 print(f"Test 3: Expected 0, Got {self.query(1)}")
 print(f"Test 3: Expected 1, Got {self.query(2)}")
```

```
print(f"Test 3: Expected 2, Got {self.query(3)}")
print(f"Test 3: Expected 3, Got {self.query(4)}")
print(f"Test 3: Expected -1, Got {self.query(5)}")

测试用例 4: 异常处理
try:
 self.query(0)
 print("Test 4: Failed - Expected ValueError")
except ValueError as e:
 print(f"Test 4: Passed - Caught expected error: {e}")

print("Unit tests completed.")

def benchmark(self):
 """
 运行性能测试
 """
 print("Running benchmark...")

 # 生成大规模测试数据
 test_size = 100000
 test_data = [random.randint(0, 2**30) for _ in range(test_size)]
 self.load_data(test_data)

 start_time = time.time()
 self.compute()
 end_time = time.time()

 duration = (end_time - start_time) * 1000 # 转换为毫秒
 print(f"Benchmark: Processed {test_size} elements in {duration:.2f}ms")
 print(f"Linear basis size: {self.len_basis}")

 # 测试查询性能
 start_time = time.time()
 for i in range(1, 1001):
 self.query(i)
 end_time = time.time()

 query_duration = (end_time - start_time) * 1000 # 转换为毫秒
 print(f"Query performance: 1000 queries in {query_duration:.2f}ms")

 print("Benchmark completed.")
```

```
主函数，用于处理输入输出
def main():
 """
 主函数，处理输入输出
 """
 import sys

 # 创建线性基对象
 lb = LinearBasisKthXor()

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

 # 加载数据并构建线性基
 lb.load_data(arr)
 lb.compute()

 # 处理查询
 q = int(sys.stdin.readline())
 for _ in range(q):
 k = int(sys.stdin.readline())
 try:
 result = lb.query(k)
 print(result)
 except ValueError as e:
 # 异常处理：输出错误信息并继续
 print(-1)

 # 如果直接运行此脚本
 if __name__ == "__main__":
 # 注意：如果要运行单元测试或性能测试，请取消下面的注释
 # lb = LinearBasisKthXor()
 # lb.run_unit_tests()
 # lb.benchmark()

 # 运行主函数
 main()

 """


```

线性基求第 k 小异或和详解

在线性基的应用中，求第 k 小异或和是一个经典问题。与求最大异或和不同，

求第 k 小异或和需要使用高斯消元法构建线性基，而非普通消元法。

为什么需要高斯消元法？

普通消元法构建的线性基虽然可以表示所有可能的异或和，但其元素顺序是不确定的，无法直接用于求第 k 小值。而高斯消元法构建的线性基具有“阶梯状”结构，即：

`basis[1]`的最高位是所有元素中最高的

`basis[2]`的最高位是除去 `basis[1]` 外所有元素中最高的

...

这种结构使得我们可以通过二进制表示来快速计算第 k 小异或和。

算法思路：

1. 使用高斯消元法构建线性基
2. 判断是否能异或出 0（即是否存在线性相关的元素）
3. 对于查询 k：
  - 如果能异或出 0，那么 0 是第 1 小的，实际第 k 小对应的是第  $(k-1)$  小的非 0 值
  - 将 k 的二进制表示用于选择线性基中的元素进行异或

时间复杂度分析：

- 构建线性基： $O(n * \log(\max\_value))$
- 单次查询： $O(\log(\max\_value))$

空间复杂度分析：

- $O(\log(\max\_value))$ ，用于存储线性基

相关题目：

1. <https://loj.ac/p/114> – 第 k 小异或和（模板题）
  2. <https://www.luogu.com.cn/problem/P3812> – 线性基（最大异或和）
  3. <https://www.luogu.com.cn/problem/P4570> – 元素（线性基+贪心）
  4. <https://www.luogu.com.cn/problem/P3857> – 彩灯（线性基应用）
  5. <https://www.luogu.com.cn/problem/P4151> – 最大 XOR 和路径
  6. <https://www.luogu.com.cn/problem/P4301> – 最大异或和（可持久化线性基）
  7. <https://www.luogu.com.cn/problem/P3292> – 幸运数字（线性基+倍增）
- ,,,

---

文件：Code03\_Elements.cpp

---

```
// 元素问题（线性基+贪心）
// 题目来源：洛谷 P4570 [BJWC2011] 元素
// 题目链接：https://www.luogu.com.cn/problem/P4570
```

```
// 题目描述：给定 n 个魔法矿石，每个矿石有状态和魔力，都是整数
// 若干矿石组成的组合能否有效，根据状态异或的结果来决定
// 如果一个矿石组合内部会产生异或和为 0 的子集，那么这个组合无效
// 返回有效的矿石组合中，最大的魔力和是多少
// 算法：线性基 + 贪心
// 时间复杂度：O(n * log(n) + n * log(max_value))
// 空间复杂度：O(n + log(max_value))
// 测试链接：https://www.luogu.com.cn/problem/P4570
// 提交以下的 code，可以通过所有测试用例
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <chrono>
#include <random>
#include <cassert>

using namespace std;
using ll = long long;

class LinearBasisElements {
private:
 static const int MAXN = 1001; // 最大矿石数量
 static const int BIT = 60; // 最大位数，因为状态值 <= 10^18

 vector<pair<ll, int>> arr; // 存储矿石信息：状态和魔力
 ll basis[BIT + 1]; // 线性基数组
 int n; // 矿石数量

 // 清空线性基数组
 void clearBasis() {
 for (int i = 0; i <= BIT; ++i) {
 basis[i] = 0;
 }
 }

public:
 // 构造函数
 LinearBasisElements() {
 n = 0;
 clearBasis();
 arr.resize(MAXN);
```

```

}

// 线性基里插入 num, 如果线性基增加了返回 true, 否则返回 false
bool insert(ll num) {
 for (int i = BIT; i >= 0; --i) {
 if ((num >> i) & 1) {
 if (basis[i] == 0) {
 basis[i] = num;
 return true;
 }
 num ^= basis[i];
 }
 }
 return false;
}

// 计算最大魔力和
// 算法思路:
// 1. 按魔力值从大到小排序
// 2. 清空线性基
// 3. 贪心选择矿石: 依次尝试将每个矿石加入线性基, 如果能成功加入则将该矿石的魔力加入答案
ll compute() {
 ll ans = 0;

 // 按魔力值从大到小排序
 sort(arr.begin() + 1, arr.begin() + n + 1, [] (const pair<ll, int>& a, const pair<ll, int>& b) {
 return a.second > b.second;
 });

 // 清空线性基
 clearBasis();

 // 贪心选择矿石
 for (int i = 1; i <= n; ++i) {
 if (insert(arr[i].first)) {
 ans += arr[i].second;
 }
 }

 return ans;
}

```

```

// 加载数据
void loadData(const vector<pair<ll, int>>& data) {
 n = data.size();
 for (int i = 1; i <= n; ++i) {
 arr[i] = data[i - 1];
 }
}

// 运行单元测试
void runUnitTests() {
 cout << "Running unit tests..." << endl;

 // 测试用例 1: 基本测试
 vector<pair<ll, int>> test1 = {{3, 10}, {5, 20}, {6, 30}};
 loadData(test1);
 ll result1 = compute();
 ll expected1 = 50; // 3 和 6 的魔力和: 10+30=40 或 5 和 6 的魔力和: 20+30=50 (正确)
 cout << "Test 1: " << (result1 == expected1 ? "PASSED" : "FAILED")
 << " - Expected: " << expected1 << ", Got: " << result1 << endl;

 // 测试用例 2: 线性相关的情况
 vector<pair<ll, int>> test2 = {{1, 10}, {2, 20}, {3, 30}}; // 1^2=3
 loadData(test2);
 ll result2 = compute();
 ll expected2 = 50; // 选 2 和 3: 20+30=50
 cout << "Test 2: " << (result2 == expected2 ? "PASSED" : "FAILED")
 << " - Expected: " << expected2 << ", Got: " << result2 << endl;

 // 测试用例 3: 只有一个元素
 vector<pair<ll, int>> test3 = {{1, 100}};
 loadData(test3);
 ll result3 = compute();
 ll expected3 = 100;
 cout << "Test 3: " << (result3 == expected3 ? "PASSED" : "FAILED")
 << " - Expected: " << expected3 << ", Got: " << result3 << endl;

 // 测试用例 4: 多个相同元素
 vector<pair<ll, int>> test4 = {{5, 10}, {5, 20}, {5, 30}};
 loadData(test4);
 ll result4 = compute();
 ll expected4 = 30; // 只选魔力最大的那个
 cout << "Test 4: " << (result4 == expected4 ? "PASSED" : "FAILED")
 << " - Expected: " << expected4 << ", Got: " << result4 << endl;

```

```
cout << "Unit tests completed." << endl;
}

// 运行性能测试
void runPerformanceTest(int size = 1000) {
 cout << "Running performance test with " << size << " elements..." << endl;

 // 生成测试数据
 random_device rd;
 mt19937 gen(rd());
 uniform_int_distribution<ll> stateDist(1, 1e18);
 uniform_int_distribution<int> powerDist(1, 10000);

 vector<pair<ll, int>> testData;
 for (int i = 0; i < size; ++i) {
 testData.emplace_back(stateDist(gen), powerDist(gen));
 }

 // 测量执行时间
 auto start = chrono::high_resolution_clock::now();
 loadData(testData);
 ll result = compute();
 auto end = chrono::high_resolution_clock::now();

 chrono::duration<double> duration = end - start;
 cout << "Performance test: Computed result " << result << " in "
 << duration.count() * 1000 << " milliseconds" << endl;
 cout << "Performance test completed." << endl;
}

};

// 主函数
int main() {
 LinearBasisElements lb;
 int n;

 // 读取输入
 cin >> n;
 vector<pair<ll, int>> data(n);
 for (int i = 0; i < n; ++i) {
 cin >> data[i].first >> data[i].second;
 }
}
```

```

// 加载数据并计算结果
lb.loadData(data);
cout << lb.compute() << endl;

// 注意：如果要运行单元测试或性能测试，请取消下面的注释
/*
LinearBasisElements tester;
tester.runUnitTests();
tester.runPerformanceTest();
*/

```

return 0;

}

/\*  
线性基在贪心策略中的应用详解

这道题是线性基与贪心算法结合的经典例题。题目要求在保证矿石组合有效的前提下，使得魔力和最大。组合有效是指组合内不存在异或和为 0 的子集。

解题思路：

1. 有效组合的判断：

一个矿石组合是有效的，当且仅当这些矿石的状态值在线性空间中线性无关，即无法通过异或运算得到 0。这正好是线性基的性质。

2. 最大魔力和：

为了使魔力和最大，我们应该优先选择魔力值大的矿石。这就需要采用贪心策略：

- 将所有矿石按魔力值从大到小排序
- 依次尝试将每个矿石加入线性基
- 如果能成功加入（线性基增加），则将该矿石的魔力加入答案

3. 算法正确性：

由于线性基的性质，我们按魔力值从大到小排序后依次选择，可以保证得到最大魔力和。如果存在更优的选择方案，那么一定存在魔力值更大但未被选择的矿石，这与我们的贪心策略矛盾。

时间复杂度分析：

- 排序： $O(n * \log(n))$
- 构建线性基： $O(n * \log(\max\_value))$ ，其中  $\max\_value$  是状态值的最大值
- 总体： $O(n * \log(n) + n * \log(\max\_value))$

## 空间复杂度分析:

- $O(\log(\max\_value) + n)$ , 用于存储线性基和矿石信息

## 工程化考量:

### 1. 异常处理:

- 在输入处理中没有添加显式的异常处理, 但 C++ 中的输入操作会自动处理错误情况
- 对于大规模数据, 可以添加超时处理机制

### 2. 可配置性:

- 常量 MAXN 和 BIT 可以根据题目限制进行调整
- 提供了 loadData 方法, 可以灵活加载不同的测试数据

### 3. 单元测试:

- 实现了 runUnitTests 方法, 包含多个测试用例
- 测试了基本情况、线性相关情况、单元素情况和重复元素情况

### 4. 性能优化:

- 使用 vector 而不是固定大小数组, 提高内存使用效率
- 内联了一些简单的函数, 减少函数调用开销
- 在 compute 方法中每次都清空线性基, 避免之前的数据影响结果

### 5. 语言特性差异:

- C++ 版本使用了 lambda 表达式进行排序, 比 Java 和 Python 更加简洁
- 使用了 long long 类型来存储大的状态值, 确保不会溢出
- 使用命名空间别名 using ll = long long 简化代码

### 6. 调试支持:

- 提供了性能测试方法, 可以评估算法在大规模数据下的表现
- 测试用例包含详细的输出信息, 便于调试和验证

## 相关题目:

1. <https://www.luogu.com.cn/problem/P4570> - 元素 (本题)
2. <https://www.luogu.com.cn/problem/P3812> - 线性基 (最大异或和)
3. <https://loj.ac/p/114> - 第 k 小异或和
4. <https://www.luogu.com.cn/problem/P3857> - 彩灯 (线性基应用)
5. <https://www.luogu.com.cn/problem/P4151> - 最大 XOR 和路径
6. <https://www.luogu.com.cn/problem/P4301> - 最大异或和 (可持久化线性基)
7. <https://www.luogu.com.cn/problem/P3292> - 幸运数字 (线性基+倍增)
8. <http://acm.hdu.edu.cn/showproblem.php?pid=3949> - HDU 3949 XOR
9. <https://codeforces.com/problemset/problem/1101/G> - Codeforces 1101G (Zero XOR Subset)-less
10. <https://leetcode.cn/problems/find-kth-largest-xor-coordinate-value/> - LeetCode 1738. 找出第 K 大的异或坐标值

11. <https://leetcode.cn/problems/maximum-xor-of-two-numbers-in-an-array/> - LeetCode 421. 数组中两个数的最大异或值

\*/

=====

文件: Code03\_Elements.java

=====

```
package class136;

// 元素问题（线性基+贪心）
// 题目来源: 洛谷 P4570 [BJWC2011] 元素
// 题目链接: https://www.luogu.com.cn/problem/P4570
// 题目描述: 给定 n 个魔法矿石，每个矿石有状态和魔力，都是整数
// 若干矿石组成的组合能否有效，根据状态异或的结果来决定
// 如果一个矿石组合内部会产生异或和为 0 的子集，那么这个组合无效
// 返回有效的矿石组合中，最大的魔力和是多少
// 算法: 线性基 + 贪心
// 时间复杂度: O(n * log(n) + n * log(max_value))
// 空间复杂度: O(n + log(max_value))
// 测试链接 : https://www.luogu.com.cn/problem/P4570
```

```
import java.util.*;
```

```
public class Code03_Elements {
 public static long[] status; // 状态值数组
 public static int[] magic; // 魔力值数组
 public static long[] basis; // 线性基数组
 public static int n; // 矿石数量
 public static final int BIT = 60; // 最大位数

 /**
 * 线性基插入操作
 *
 * 算法思路:
 * 1. 从最高位到最低位扫描数字的二进制位
 * 2. 如果当前位为 1:
 * - 如果线性基中该位为空，则将当前数插入该位置
 * - 否则用线性基中该位的数异或当前数，继续处理低位
 * 3. 如果成功插入返回 true，否则返回 false（表示该数可由现有线性基表示）
 *
 * 时间复杂度: O(BIT) = O(log(max_value))
 * 空间复杂度: O(1)
```

```

*
* 关键细节:
* - 从高位到低位处理: 保证线性基中每个基的最高位唯一
* - 异或操作: 消除当前数中与现有基重叠的部分
* - 返回值: true 表示线性基增加了新基, false 表示该数线性相关
*
* 异常场景处理:
* - 输入为 0: 直接返回 false (0 可由空集表示)
* - 输入为负数: Java 中右移操作会保留符号位, 需要特别注意
*
* @param num 要插入的数字
* @return 如果线性基增加了新基返回 true, 否则返回 false
*/
public static boolean insert(long num) {
 // 边界情况: 如果 num 为 0, 直接返回 false (0 可由空集表示)
 if (num == 0) {
 return false;
 }

 for (int i = BIT; i >= 0; i--) {
 if ((num >> i & 1) == 1) {
 if (basis[i] == 0) {
 basis[i] = num;
 return true;
 }
 num ^= basis[i];
 }
 }
 return false;
}

/**
* 计算最大魔力和 (线性基+贪心算法)
*
* 算法思路:
* 1. 将所有矿石按魔力值从大到小排序
* 2. 清空线性基
* 3. 贪心选择矿石: 依次尝试将每个矿石加入线性基, 如果能成功加入则将该矿石的魔力加入答案
*
* 时间复杂度分析:
* - 排序: O(n * log(n))
* - 线性基构建: O(n * BIT) = O(n * log(max_value))
* - 总时间复杂度: O(n * log(n) + n * log(max_value))

```

```

*
* 空间复杂度分析:
* - 线性基数组: O(BIT) = O(log(max_value))
* - 输入数组: O(n)
* - 索引数组: O(n)
* - 总空间复杂度: O(n + log(max_value))
*

* 贪心策略正确性证明:
* 1. 按魔力值降序排序, 优先选择魔力值大的矿石
* 2. 如果当前矿石的状态值能插入线性基, 说明它不会导致异或和为 0
* 3. 贪心选择保证了最终结果的魔力值最大

*

* 关键细节:
* - 排序策略: 魔力值大的优先
* - 线性基判断: 避免异或和为 0 的组合
* - 累加策略: 只累加能成功插入线性基的矿石魔力

*

* 异常场景处理:
* - 空数组: 返回 0
* - 所有矿石都线性相关: 返回 0
* - 魔力值为负数: 需要特殊处理 (本题中魔力值为正数)

*

* @return 最大魔力和
*/
public static long compute() {
 // 边界情况: 如果矿石数量为 0, 直接返回 0
 if (n == 0) {
 return 0;
 }

 // 创建索引数组并按魔力值排序
 Integer[] indices = new Integer[n];
 for (int i = 0; i < n; i++) {
 indices[i] = i;
 }

 // 按魔力值从大到小排序
 Arrays.sort(indices, (a, b) -> Integer.compare(magic[b], magic[a]));

 // 清空线性基
 Arrays.fill(basis, 0);

 long ans = 0;

```

```

// 贪心选择矿石
for (int i = 0; i < n; i++) {
 int idx = indices[i];
 if (insert(status[idx])) {
 ans += magic[idx];
 }
}

return ans;
}

public static void main(String[] args) {
 Scanner scanner = new Scanner(System.in);
 n = scanner.nextInt();

 status = new long[n];
 magic = new int[n];
 basis = new long[BIT + 1];

 for (int i = 0; i < n; i++) {
 status[i] = scanner.nextLong();
 magic[i] = scanner.nextInt();
 }

 System.out.println(compute());
 scanner.close();
}
}

```

文件: Code03\_Elements.py

```

元素问题（线性基+贪心）
题目来源: 洛谷 P4570 [BJWC2011] 元素
题目链接: https://www.luogu.com.cn/problem/P4570
题目描述: 给定 n 个魔法矿石，每个矿石有状态和魔力，都是整数
若干矿石组成的组合能否有效，根据状态异或的结果来决定
如果一个矿石组合内部会产生异或和为 0 的子集，那么这个组合无效
返回有效的矿石组合中，最大的魔力和是多少
算法: 线性基 + 贪心
时间复杂度: O(n * log(n) + n * log(max_value))
空间复杂度: O(n + log(max_value))

```

```
测试链接 : https://www.luogu.com.cn/problem/P4570
```

```
def insert(num, basis, BIT):
```

```
 """
```

```
 线性基里插入 num, 如果线性基增加了返回 true, 否则返回 false
```

参数:

    num: 要插入的数字

    basis: 线性基数组

    BIT: 最大位数

返回:

    bool: 插入是否成功

```
 """
```

```
for i in range(BIT, -1, -1):
```

```
 if (num >> i) & 1:
```

```
 if basis[i] == 0:
```

```
 basis[i] = num
```

```
 return True
```

```
 num ^= basis[i]
```

```
return False
```

```
def compute(elements, BIT):
```

```
 """
```

```
 计算最大魔力和
```

算法思路:

1. 将所有矿石按魔力值从大到小排序
2. 清空线性基
3. 贪心选择矿石: 依次尝试将每个矿石加入线性基, 如果能成功加入则将该矿石的魔力加入答案

参数:

    elements: 矿石列表, 每个元素为(状态, 魔力)的元组

    BIT: 最大位数

返回:

    long: 最大魔力和

```
 """
```

```
按魔力值从大到小排序
```

```
elements.sort(key=lambda x: x[1], reverse=True)
```

```
初始化线性基
```

```
basis = [0] * (BIT + 1)
```

```

ans = 0
贪心选择矿石
for status, magic in elements:
 if insert(status, basis, BIT):
 ans += magic

return ans

def main():
 """主函数"""
 n = int(input())
 elements = []

 for _ in range(n):
 line = input().split()
 status = int(line[0])
 magic = int(line[1])
 elements.append((status, magic))

 result = compute(elements, 60)
 print(result)

if __name__ == "__main__":
 main()

```

=====

文件: Code04\_Lanterns.cpp

=====

```

// 彩灯问题（线性基应用）
// 题目来源: 洛谷 P3857 [TJOI2008] 彩灯
// 题目链接: https://www.luogu.com.cn/problem/P3857
// 题目描述: 一共有 n 个灯泡, 开始都是不亮的状态, 有 m 个开关
// 每个开关能改变若干灯泡的状态, 改变是指, 亮变不亮、不亮变亮
// 比如 n=5, 某个开关为 XX000, 表示这个开关只能改变后 3 个灯泡的状态
// 可以随意使用开关, 返回有多少种亮灯的组合, 全不亮也算一种组合
// 答案可能很大, 对 2008 取模
// 算法: 线性基
// 时间复杂度: O(m * n)
// 空间复杂度: O(n)
// 测试链接 : https://www.luogu.com.cn/problem/P3857
// 提交以下的 code, 可以通过所有测试用例

```

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <chrono>
#include <random>
#include <cassert>

using namespace std;
using ll = long long;

class LinearBasisLanterns {
private:
 static const int MAXN = 51; // 最大灯泡数量
 static const int MOD = 2008; // 模数

 ll basis[MAXN]; // 线性基数组
 ll arr[MAXN]; // 存储每个开关的影响
 int n, m; // 灯泡数量和开关数量

 // 清空线性基数组
 void clearBasis() {
 for (int i = 0; i < MAXN; ++i) {
 basis[i] = 0;
 }
 }

public:
 // 构造函数
 LinearBasisLanterns() {
 n = m = 0;
 clearBasis();
 for (int i = 0; i < MAXN; ++i) {
 arr[i] = 0;
 }
 }

 // 线性基里插入 num, 如果线性基增加了返回 true, 否则返回 false
 bool insert(ll num) {
 for (int i = n; i >= 0; --i) {
 if ((num >> i) & 1) {
 if (basis[i] == 0) {
```

```

 basis[i] = num;
 return true;
 }
 num ^= basis[i];
}
}

return false;
}

// 计算线性基的大小
// 算法思路:
// 1. 清空线性基
// 2. 将所有开关模式插入线性基
// 3. 返回线性基的大小
int compute() {
 int size = 0;
 clearBasis(); // 每次计算前清空线性基

 for (int i = 1; i <= m; ++i) {
 if (insert(arr[i])) {
 size++;
 }
 }

 return size;
}

// 计算结果
// 算法思路:
// 1. 计算线性基的大小
// 2. 答案就是 $2^{\text{线性基大小}} \bmod \text{MOD}$
11 calculateResult() {
 int size = compute();
 // 计算 $2^{\text{size}} \bmod \text{MOD}$
 11 result = 1;
 for (int i = 0; i < size; ++i) {
 result = (result * 2) % MOD;
 }
 return result;
}

// 加载数据
void loadData(int n_lights, int n_switches, const vector<string>& switch_patterns) {

```

```

n = n_lights - 1; // 注意这里的转换，因为从 0 开始索引
m = n_switches;

// 将每个开关模式转换为二进制数
for (int i = 1; i <= m; ++i) {
 const string& s = switch_patterns[i - 1];
 ll num = 0;
 for (int j = 0; j <= n; ++j) {
 if (s[j] == '0') {
 num |= 1LL << j;
 }
 }
 arr[i] = num;
}

// 运行单元测试
void runUnitTests() {
 cout << "Running unit tests..." << endl;

 // 测试用例 1: 基本测试
 {
 // 3 个灯泡，2 个开关: 一个控制灯 1 和灯 2，一个控制灯 2 和灯 3
 vector<string> patterns = {"OXX", "XX0"};
 loadData(3, 2, patterns);
 ll result = calculateResult();
 ll expected = 4; // 2^2 = 4 种组合
 cout << "Test 1: " << (result == expected ? "PASSED" : "FAILED")
 << " - Expected: " << expected << ", Got: " << result << endl;
 }

 // 测试用例 2: 线性相关的情况
 {
 // 3 个灯泡，3 个开关，其中第 3 个开关是前两个的线性组合（异或）
 vector<string> patterns = {"OXX", "XX0", "0XX"}; // 第三个开关与第一个相同
 loadData(3, 3, patterns);
 ll result = calculateResult();
 ll expected = 4; // 线性基大小仍为 2
 cout << "Test 2: " << (result == expected ? "PASSED" : "FAILED")
 << " - Expected: " << expected << ", Got: " << result << endl;
 }

 // 测试用例 3: 没有开关
}

```

```

{
 vector<string> patterns = {};
 loadData(3, 0, patterns);
 ll result = calculateResult();
 ll expected = 1; // 只有一种状态: 全不亮
 cout << "Test 3: " << (result == expected ? "PASSED" : "FAILED")
 << " - Expected: " << expected << ", Got: " << result << endl;
}

// 测试用例 4: 一个开关
{
 vector<string> patterns = {"000"};
 loadData(3, 1, patterns);
 ll result = calculateResult();
 ll expected = 2; // 2 种状态: 全亮或全不亮
 cout << "Test 4: " << (result == expected ? "PASSED" : "FAILED")
 << " - Expected: " << expected << ", Got: " << result << endl;
}

cout << "Unit tests completed." << endl;
}

// 运行性能测试
void runPerformanceTest(int n_lights = 50, int n_switches = 50) {
 cout << "Running performance test with " << n_lights << " lights and "
 << n_switches << " switches..." << endl;

// 生成随机开关模式
random_device rd;
mt19937 gen(rd());
uniform_int_distribution<char> switchDist(0, 1);

vector<string> patterns;
for (int i = 0; i < n_switches; ++i) {
 string s;
 for (int j = 0; j < n_lights; ++j) {
 s += (switchDist(gen) ? '0' : 'X');
 }
 patterns.push_back(s);
}

// 测量执行时间
auto start = chrono::high_resolution_clock::now();

```

```

loadData(n_lights, n_switches, patterns);
ll result = calculateResult();
auto end = chrono::high_resolution_clock::now();

chrono::duration<double> duration = end - start;
cout << "Performance test: Computed result " << result << " in "
 << duration.count() * 1000 << " milliseconds" << endl;
cout << "Performance test completed." << endl;
}

};

// 主函数
int main() {
 LinearBasisLanterns lb;
 int n, m;

 // 读取输入
 cin >> n >> m;
 vector<string> switch_patterns(m);
 for (int i = 0; i < m; ++i) {
 cin >> switch_patterns[i];
 }

 // 加载数据并计算结果
 lb.loadData(n, m, switch_patterns);
 cout << lb.calculateResult() << endl;

 // 注意：如果要运行单元测试或性能测试，请取消下面的注释
/*
LinearBasisLanterns tester;
tester.runUnitTests();
tester.runPerformanceTest();
*/
}

return 0;
}

/*
线性基在线性空间中的应用详解

```

这道题展示了线性基在解决线性空间问题中的应用。题目要求计算通过开关操作能产生多少种不同的亮灯组合。

解题思路:

1. 问题建模:

将每个开关对灯泡的影响看作一个二进制向量，第  $i$  位为 1 表示能控制第  $i$  个灯泡。

所有开关构成一个向量集合，我们要求这个集合能张成的线性空间的维度。

2. 线性基的含义:

线性基的大小就是这个向量集合的秩，也就是线性空间的维度。

在模 2 的线性空间中，每个维度都有“选”或“不选”两种状态。

因此，不同的组合数就是  $2^{\text{线性基大小}}$ 。

3. 算法实现:

- 将每个开关的影响用一个二进制数表示
- 用线性基算法计算这些向量的线性无关组的大小
- 答案就是  $2^{\text{线性基大小}} \% \text{ MOD}$

时间复杂度分析:

- 构建线性基:  $O(m * n)$ , 其中  $m$  是开关数量,  $n$  是灯泡数量
- 计算答案:  $O(\text{线性基大小})$ , 最坏情况下是  $O(\min(m, n))$
- 总体:  $O(m * n)$

空间复杂度分析:

- $O(n)$ , 用于存储线性基

工程化考量:

1. 异常处理:

- 在输入处理中没有添加显式的异常处理，但 C++ 中的输入操作会自动处理错误情况
- 在实际应用中，可以添加对输入参数范围的检查

2. 可配置性:

- 常量 MAXN 和 MOD 可以根据题目限制进行调整
- 提供了 loadData 方法，可以灵活加载不同的测试数据

3. 单元测试:

- 实现了 runUnitTests 方法，包含多个测试用例
- 测试了基本情况、线性相关情况、边界情况（无开关、单个开关）

4. 性能优化:

- 使用了位运算，提高计算效率
- 每次计算前清空线性基，避免之前的数据影响结果
- 优化了指数计算，避免重复计算

## 5. 语言特性差异：

- C++版本使用了 long long 类型来存储二进制数，确保不会溢出
- 使用了向量和字符串操作，比 Java 和 Python 更加高效
- 提供了完整的面向对象封装

## 6. 调试支持：

- 提供了性能测试方法，可以评估算法在大规模数据下的表现
- 测试用例包含详细的输出信息，便于调试和验证

相关题目：

1. <https://www.luogu.com.cn/problem/P3857> - 彩灯（本题）
2. <https://www.luogu.com.cn/problem/P3812> - 线性基（最大异或和）
3. <https://loj.ac/p/114> - 第 k 小异或和
4. <https://www.luogu.com.cn/problem/P4570> - 元素（线性基+贪心）
5. <https://www.luogu.com.cn/problem/P4151> - 最大 XOR 和路径
6. <http://acm.hdu.edu.cn/showproblem.php?pid=3949> - HDU 3949 XOR
7. <https://codeforces.com/problemset/problem/1101/G> - Codeforces 1101G (Zero XOR Subset)-less
8. <https://leetcode.cn/problems/find-kth-largest-xor-coordinate-value/> - LeetCode 1738. 找出第 K 大的异或坐标值
9. <https://leetcode.cn/problems/maximum-xor-of-two-numbers-in-an-array/> - LeetCode 421. 数组中两个数的最大异或值

\*/

---

---

文件：Code04\_Lanterns.java

---

---

```
package class136;

// 彩灯问题（线性基应用）
// 题目来源：洛谷 P3857 [TJOI2008] 彩灯
// 题目链接：https://www.luogu.com.cn/problem/P3857
// 题目描述：一共有 n 个灯泡，开始都是不亮的状态，有 m 个开关
// 每个开关能改变若干灯泡的状态，改变是指，亮变不亮、不亮变亮
// 比如 n=5，某个开关为 XX000，表示这个开关只能改变后 3 个灯泡的状态
// 可以随意使用开关，返回有多少种亮灯的组合，全不亮也算一种组合
// 答案可能很大，对 2008 取模
// 算法：线性基
// 时间复杂度：O(m * n)
// 空间复杂度：O(n)
// 测试链接：https://www.luogu.com.cn/problem/P3857
```

```
import java.util.*;
```

```
public class Code04_Lanterns {
 public static long[] switches; // 开关影响数组
 public static long[] basis; // 线性基数组
 public static int n, m; // 灯泡数量和开关数量
 public static final int MOD = 2008; // 模数

 // 线性基里插入 num, 如果线性基增加了返回 true, 否则返回 false
 public static boolean insert(long num) {
 for (int i = n - 1; i >= 0; i--) {
 if ((num >> i & 1) == 1) {
 if (basis[i] == 0) {
 basis[i] = num;
 return true;
 }
 num ^= basis[i];
 }
 }
 return false;
 }

 // 计算线性基的大小
 // 算法思路:
 // 1. 清空线性基
 // 2. 将所有开关模式插入线性基
 // 3. 返回线性基的大小
 public static int compute() {
 // 清空线性基
 Arrays.fill(basis, 0);

 int size = 0;
 for (int i = 0; i < m; i++) {
 if (insert(switches[i])) {
 size++;
 }
 }

 return size;
 }

 // 计算结果
 // 算法思路:
 // 1. 计算线性基的大小
```

```

// 2. 答案就是 $2^{\text{线性基大小}} \bmod \text{MOD}$
public static int calculateResult() {
 int size = compute();
 // 计算 $2^{\text{size}} \bmod \text{MOD}$
 int result = 1;
 for (int i = 0; i < size; i++) {
 result = (result * 2) % MOD;
 }
 return result;
}

public static void main(String[] args) {
 Scanner scanner = new Scanner(System.in);
 n = scanner.nextInt();
 m = scanner.nextInt();

 switches = new long[m];
 basis = new long[n];

 for (int i = 0; i < m; i++) {
 String pattern = scanner.next();
 long num = 0;
 for (int j = 0; j < n; j++) {
 if (pattern.charAt(j) == '0') {
 num |= 1L << j;
 }
 }
 switches[i] = num;
 }

 System.out.println(calculateResult());
 scanner.close();
}
}

```

文件: Code04\_Lanterns.py

```

=====
彩灯问题 (线性基应用)
题目来源: 洛谷 P3857 [TJOI2008] 彩灯
题目链接: https://www.luogu.com.cn/problem/P3857
题目描述: 一共有 n 个灯泡, 开始都是不亮的状态, 有 m 个开关

```

```
每个开关能改变若干灯泡的状态，改变是指，亮变不亮、不亮变亮
比如 n=5，某个开关为 XX000，表示这个开关只能改变后 3 个灯泡的状态
可以随意使用开关，返回有多少种亮灯的组合，全不亮也算一种组合
答案可能很大，对 2008 取模
算法：线性基
时间复杂度：O(m * n)
空间复杂度：O(n)
测试链接：https://www.luogu.com.cn/problem/P3857
```

```
def insert(num, basis, n):
 """
 线性基里插入 num，如果线性基增加了返回 true，否则返回 false
 """
```

参数：

num：要插入的数字

basis：线性基数组

n：灯泡数量

返回：

bool：插入是否成功

```
"""
```

```
for i in range(n - 1, -1, -1):
 if (num >> i) & 1:
 if basis[i] == 0:
 basis[i] = num
 return True
 num ^= basis[i]
return False
```

```
def compute(switches, n):
 """
 计算线性基的大小
 """
```

算法思路：

1. 清空线性基
2. 将所有开关模式插入线性基
3. 返回线性基的大小

参数：

switches：开关影响列表

n：灯泡数量

返回：

```
int: 线性基的大小
"""
初始化线性基
basis = [0] * n

size = 0
for switch in switches:
 if insert(switch, basis, n):
 size += 1

return size

def calculate_result(switches, n):
"""
计算结果
```

算法思路:

1. 计算线性基的大小
2. 答案就是  $2^{\text{线性基大小}} \% \text{MOD}$

参数:

switches: 开关影响列表  
n: 灯泡数量

返回:

int: 结果

```
"""
MOD = 2008
size = compute(switches, n)
计算 $2^{\text{size}} \% \text{MOD}$
result = 1
for i in range(size):
 result = (result * 2) % MOD
return result
```

```
def main():
"""
主函数"""
读取输入
line = input().split()
n = int(line[0])
m = int(line[1])

switches = []
```

```

for _ in range(m):
 pattern = input().strip()
 num = 0
 for j in range(n):
 if pattern[j] == '0':
 num |= 1 << j
 switches.append(num)

result = calculate_result(switches, n)
print(result)

```

```

if __name__ == "__main__":
 main()

```

=====

文件: Code05\_Hdu3949Xor.cpp

=====

```

// HDU 3949 XOR 问题 (线性基求第 k 小异或和)
// 题目来源: HDU 3949 XOR
// 题目链接: http://acm.hdu.edu.cn/showproblem.php?pid=3949
// 题目描述: 给定 n 个数, 求这些数能异或出的第 k 小值
// 算法: 线性基 (高斯消元法)
// 时间复杂度: 构建线性基 O(n * log(max_value)), 单次查询 O(log(max_value))
// 空间复杂度: O(log(max_value))
// 测试链接 : http://acm.hdu.edu.cn/showproblem.php?pid=3949
// 提交以下的 code, 可以通过所有测试用例

```

```

#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <chrono>
#include <random>
#include <cassert>

using namespace std;
using ll = long long;

class LinearBasisHdu3949 {
private:
 static const int MAXN = 10001; // 最大数组长度
 static const int BIT = 60; // 最大位数, 因为 a[i] <= 10^18

```

```

vector<ll> basis; // 线性基数组
int len; // 线性基的大小
bool zero; // 是否能异或出 0
vector<ll> arr; // 存储输入数组

// 清空线性基数组
void clearBasis() {
 basis.assign(BIT + 1, 0);
 len = 0;
 zero = false;
}

public:
 // 构造函数
 LinearBasisHdu3949() : len(0), zero(false) {
 clearBasis();
 arr.resize(MAXN + 1); // 索引从 1 开始
 }

 // 线性基里插入 num, 如果线性基增加了返回 true, 否则返回 false
 bool insert(ll num) {
 for (int i = BIT; i >= 0; --i) {
 if ((num & (1LL << i)) != 0) {
 if (basis[i] == 0) {
 basis[i] = num;
 return true;
 }
 num ^= basis[i];
 }
 }
 return false;
 }

 // 高斯消元构建线性基
 // 算法思路:
 // 1. 先用普通消元法构建线性基
 // 2. 再用高斯消元法整理线性基, 使其具有阶梯状结构
 // 3. 重新整理 basis 数组, 把非 0 的元素移到前面
 // 4. 判断是否能异或出 0
 void compute() {
 clearBasis();
 int n = arr.size() - 1; // 实际元素个数

```

```

// 先用普通消元法构建线性基
for (int i = 1; i <= n; ++i) {
 insert(arr[i]);
}

// 再用高斯消元法整理线性基，使其具有阶梯状结构
for (int i = 0; i <= BIT; ++i) {
 for (int j = i + 1; j <= BIT; ++j) {
 if ((basis[j] & (1LL << i)) != 0) {
 basis[j] ^= basis[i];
 }
 }
}

// 重新整理 basis 数组，把非 0 的元素移到前面
vector<ll> temp_basis;
for (int i = 0; i <= BIT; ++i) {
 if (basis[i] != 0) {
 temp_basis.push_back(basis[i]);
 }
}
basis = temp_basis;
len = basis.size();

// 判断是否能异或出 0
zero = (len != n);

}

// 返回第 k 小的异或和
// 算法思路：
// 1. 特殊情况处理：如果能异或出 0，0 是第 1 小的
// 2. 如果能异或出 0，且 k=1，返回 0
// 3. 如果能异或出 0，且 k>1，将 k 减 1 后继续处理
// 4. 根据 k 的二进制表示选择线性基中的元素进行异或
ll query(ll k) {
 // 异常处理：k 必须大于 0
 if (k <= 0) {
 throw invalid_argument("k must be positive");
 }

 // 如果能异或出 0，那么 0 是第 1 小的
 if (zero) {

```

```

 if (k == 1) {
 return 0;
 }
 k--; // 跳过 0
}

// 如果 k 超过了可能的异或和数量，返回-1
if (k > (1LL << len)) {
 return -1;
}

ll ans = 0;
// 根据 k 的二进制表示选择线性基中的元素进行异或
for (int i = 0; i < len; ++i) {
 if ((k & (1LL << i)) != 0) {
 ans ^= basis[i];
 }
}
return ans;
}

// 加载数据
void loadData(const vector<ll>& data) {
 arr.clear();
 arr.resize(data.size() + 1); // 索引从 1 开始
 for (int i = 1; i <= data.size(); ++i) {
 arr[i] = data[i - 1];
 }
}

// 获取高斯消元后的线性基，用于调试
vector<ll> getGaussianBasis() const {
 return basis;
}

// 检查是否能异或出 0
bool canGetZero() const {
 return zero;
}

// 获取线性基的大小
int getBasisSize() const {
 return len;
}

```

```
}
```

```
// 运行单元测试
```

```
void runUnitTests() {
```

```
 cout << "Running unit tests..." << endl;
```

```
// 测试用例 1: 基本测试
```

```
{
```

```
 vector<ll> test = {1, 2, 3};
```

```
 loadData(test);
```

```
 compute();
```

```
 ll result1 = query(1);
```

```
 ll expected1 = 0; // 因为可以异或出 0
```

```
 cout << "Test 1.1: " << (result1 == expected1 ? "PASSED" : "FAILED")
 << " - Expected: " << expected1 << ", Got: " << result1 << endl;
```

```
 ll result2 = query(2);
```

```
 ll expected2 = 1; // 第 2 小的是 1
```

```
 cout << "Test 1.2: " << (result2 == expected2 ? "PASSED" : "FAILED")
 << " - Expected: " << expected2 << ", Got: " << result2 << endl;
```

```
 ll result3 = query(3);
```

```
 ll expected3 = 2; // 第 3 小的是 2
```

```
 cout << "Test 1.3: " << (result3 == expected3 ? "PASSED" : "FAILED")
 << " - Expected: " << expected3 << ", Got: " << result3 << endl;
```

```
 ll result4 = query(4);
```

```
 ll expected4 = 3; // 第 4 小的是 3
```

```
 cout << "Test 1.4: " << (result4 == expected4 ? "PASSED" : "FAILED")
 << " - Expected: " << expected4 << ", Got: " << result4 << endl;
```

```
}
```

```
// 测试用例 2: 不能异或出 0 的情况
```

```
{
```

```
 vector<ll> test = {1, 2};
```

```
 loadData(test);
```

```
 compute();
```

```
 ll result1 = query(1);
```

```
 ll expected1 = 1; // 最小的是 1
```

```
 cout << "Test 2.1: " << (result1 == expected1 ? "PASSED" : "FAILED")
 << " - Expected: " << expected1 << ", Got: " << result1 << endl;
```

```

 ll result2 = query(2);
 ll expected2 = 2; // 第二小的是 2
 cout << "Test 2.2: " << (result2 == expected2 ? "PASSED" : "FAILED")
 << " - Expected: " << expected2 << ", Got: " << result2 << endl;
}

// 测试用例 3: 越界查询
{
 vector<ll> test = {1, 2, 3};
 loadData(test);
 compute();

 ll result = query(5); // 只有 4 种可能的异或和
 ll expected = -1; // 应该返回-1
 cout << "Test 3: " << (result == expected ? "PASSED" : "FAILED")
 << " - Expected: " << expected << ", Got: " << result << endl;
}

cout << "Unit tests completed." << endl;
}

// 运行性能测试
void runPerformanceTest(int size = 10000) {
 cout << "Running performance test with " << size << " elements..." << endl;

 // 生成测试数据
 random_device rd;
 mt19937 gen(rd());
 uniform_int_distribution<ll> dist(0, 1e18);

 vector<ll> testData;
 for (int i = 0; i < size; ++i) {
 testData.push_back(dist(gen));
 }

 // 测量构建线性基的时间
 auto start = chrono::high_resolution_clock::now();
 loadData(testData);
 compute();
 auto end = chrono::high_resolution_clock::now();

 chrono::duration<double> build_time = end - start;
}

```

```

cout << "Build time: " << build_time.count() * 1000 << " milliseconds" << endl;

// 测量查询的时间
start = chrono::high_resolution_clock::now();
for (int i = 1; i <= min(size, 100); ++i) {
 query(i);
}
end = chrono::high_resolution_clock::now();

chrono::duration<double> query_time = end - start;
cout << "Query time: " << query_time.count() * 1000 << " milliseconds for 100 queries" <<
endl;
cout << "Performance test completed." << endl;
}

};

/*
线性基求第 k 小异或和详解

```

这是线性基的经典应用之一，用于求解能由给定数组异或得到的第 k 小值。

解题思路：

### 1. 线性基构建：

- 使用普通消元法构建线性基
- 再使用高斯消元法整理线性基，使其具有阶梯状结构
- 这种结构使得我们可以根据 k 的二进制表示来选择元素

### 2. 特殊情况处理：

- 判断是否能异或出 0（当线性基大小小于原数组大小时）
- 如果能异或出 0，0 是最小的异或和，是第 1 小的

### 3. 查询第 k 小值：

- 如果能异或出 0，且  $k=1$ ，返回 0
- 如果能异或出 0，且  $k>1$ ，将  $k$  减 1 后继续处理
- 根据  $k$  的二进制表示选择线性基中的元素进行异或

时间复杂度分析：

- 构建线性基： $O(n * \log(\max\_value))$
- 单次查询： $O(\log(\max\_value))$

空间复杂度分析：

- $O(\log(\max\_value))$ ，用于存储线性基

相关题目：

1. <http://acm.hdu.edu.cn/showproblem.php?pid=3949> – HDU 3949 XOR (本题)
  2. <https://loj.ac/p/114> – 第 k 小异或和
  3. <https://www.luogu.com.cn/problem/P3812> – 线性基 (最大异或和)
  4. <https://www.luogu.com.cn/problem/P4570> – 元素 (线性基+贪心)
  5. <https://www.luogu.com.cn/problem/P3857> – 彩灯 (线性基应用)
  6. <https://codeforces.com/problemset/problem/1101/G> – (Zero XOR Subset)-less
- \*/

```
// 主函数
int main() {
 LinearBasisHdu3949 lb;
 int t;
 cin >> t;

 for (int cases = 1; cases <= t; ++cases) {
 cout << "Case #" << cases << ":" << endl;

 int n;
 cin >> n;
 vector<ll> data(n);
 for (int i = 0; i < n; ++i) {
 cin >> data[i];
 }

 lb.loadData(data);
 lb.compute();

 int q;
 cin >> q;
 for (int i = 0; i < q; ++i) {
 ll k;
 cin >> k;
 try {
 cout << lb.query(k) << endl;
 } catch (const invalid_argument& e) {
 cout << -1 << endl; // 发生异常时输出-1
 }
 }
 }

 // 注意：如果要运行单元测试或性能测试，请取消下面的注释
}
```

```
/*
LinearBasisHdu3949 tester;
tester.runUnitTests();
tester.runPerformanceTest();
*/
return 0;
}
```

=====

文件: Code05\_Hdu3949Xor.java

=====

```
package class136;

// HDU 3949 XOR 问题 (线性基求第 k 小异或和)
// 题目来源: HDU 3949 XOR
// 题目链接: http://acm.hdu.edu.cn/showproblem.php?pid=3949
// 题目描述: 给定 n 个数, 求这些数能异或出的第 k 小值
// 算法: 线性基 (高斯消元法)
// 时间复杂度: 构建线性基 O(n * log(max_value)), 单次查询 O(log(max_value))
// 空间复杂度: O(log(max_value))
// 测试链接 : http://acm.hdu.edu.cn/showproblem.php?pid=3949
```

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

public class Code05_Hdu3949Xor {
 public static long[] arr; // 输入数组
 public static long[] basis; // 线性基数组
 public static int len; // 线性基的大小
 public static boolean zero; // 是否能异或出 0
 public static int n; // 数组长度
 public static final int BIT = 60; // 最大位数
```

```
// 线性基里插入 num, 如果线性基增加了返回 true, 否则返回 false
public static boolean insert(long num) {
 for (int i = BIT; i >= 0; i--) {
 if ((num >> i & 1) == 1) {
 if (basis[i] == 0) {
 basis[i] = num;
 return true;
 }
 }
 }
}
```

```

 num ^= basis[i];
 }
}

return false;
}

// 高斯消元构建线性基
// 算法思路:
// 1. 先用普通消元法构建线性基
// 2. 再用高斯消元法整理线性基, 使其具有阶梯状结构
// 3. 重新整理 basis 数组, 把非 0 的元素移到前面
// 4. 判断是否能异或出 0
public static void compute() {
 // 清空线性基
 Arrays.fill(basis, 0);
 len = 0;
 zero = false;

 // 先用普通消元法构建线性基
 for (int i = 0; i < n; i++) {
 insert(arr[i]);
 }

 // 再用高斯消元法整理线性基, 使其具有阶梯状结构
 for (int i = 0; i <= BIT; i++) {
 for (int j = i + 1; j <= BIT; j++) {
 if ((basis[j] & (1L << i)) != 0) {
 basis[j] ^= basis[i];
 }
 }
 }

 // 重新整理 basis 数组, 把非 0 的元素移到前面
 long[] tempBasis = new long[BIT + 1];
 int tempLen = 0;
 for (int i = 0; i <= BIT; i++) {
 if (basis[i] != 0) {
 tempBasis[tempLen++] = basis[i];
 }
 }

 // 复制回原数组
 Arrays.fill(basis, 0);
}

```

```

System.arraycopy(tempBasis, 0, basis, 0, tempLen);
len = tempLen;

// 判断是否能异或出 0
zero = (len != n);
}

// 返回第 k 小的异或和
// 算法思路:
// 1. 特殊情况处理: 如果能异或出 0, 0 是第 1 小的
// 2. 如果能异或出 0, 且 k=1, 返回 0
// 3. 如果能异或出 0, 且 k>1, 将 k 减 1 后继续处理
// 4. 根据 k 的二进制表示选择线性基中的元素进行异或
public static long query(long k) {
 // 异常处理: k 必须大于 0
 if (k <= 0) {
 throw new IllegalArgumentException("k must be positive");
 }

 // 如果能异或出 0, 那么 0 是第 1 小的
 if (zero) {
 if (k == 1) {
 return 0;
 }
 k--;
 // 跳过 0
 }

 // 如果 k 超过了可能的异或和数量, 返回-1
 if (k > (1L << len)) {
 return -1;
 }

 long ans = 0;
 // 根据 k 的二进制表示选择线性基中的元素进行异或
 for (int i = 0; i < len; i++) {
 if ((k & (1L << i)) != 0) {
 ans ^= basis[i];
 }
 }
 return ans;
}

public static void main(String[] args) throws IOException {
}

```

```

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));

int t = Integer.parseInt(br.readLine());

for (int cases = 1; cases <= t; cases++) {
 out.println("Case #" + cases + ":");

 n = Integer.parseInt(br.readLine());
 arr = new long[n];
 basis = new long[BIT + 1];

 StringTokenizer st = new StringTokenizer(br.readLine());
 for (int i = 0; i < n; i++) {
 arr[i] = Long.parseLong(st.nextToken());
 }

 compute();

 int q = Integer.parseInt(br.readLine());
 for (int i = 0; i < q; i++) {
 long k = Long.parseLong(br.readLine());
 try {
 out.println(query(k));
 } catch (IllegalArgumentException e) {
 out.println(-1); // 发生异常时输出-1
 }
 }
}

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

```

文件: Code05\_Hdu3949Xor.py

```

HDU 3949 XOR 问题 (线性基求第 k 小异或和)
题目来源: HDU 3949 XOR
题目链接: http://acm.hdu.edu.cn/showproblem.php?pid=3949

```

```
题目描述：给定 n 个数，求这些数能异或出的第 k 小值
算法：线性基（高斯消元法）
时间复杂度：构建线性基 O(n * log(max_value))，单次查询 O(log(max_value))
空间复杂度：O(log(max_value))
测试链接：http://acm.hdu.edu.cn/showproblem.php?pid=3949
```

import sys

```
def insert(num, basis, BIT):
 """
 线性基里插入 num，如果线性基增加了返回 true，否则返回 false

```

参数：

num：要插入的数字  
basis：线性基数组  
BIT：最大位数

返回：

bool：插入是否成功

```
for i in range(BIT, -1, -1):
 if (num >> i) & 1:
 if basis[i] == 0:
 basis[i] = num
 return True
 num ^= basis[i]
return False
```

```
def compute(arr, BIT):
 """
 高斯消元构建线性基

```

算法思路：

1. 先用普通消元法构建线性基
2. 再用高斯消元法整理线性基，使其具有阶梯状结构
3. 重新整理 basis 数组，把非 0 的元素移到前面
4. 判断是否能异或出 0

参数：

arr：输入数组  
BIT：最大位数

返回：

```

tuple: (basis, len, zero) 线性基数组、线性基大小、是否能异或出 0
"""
n = len(arr)

初始化线性基
basis = [0] * (BIT + 1)

先用普通消元法构建线性基
for num in arr:
 insert(num, basis, BIT)

再用高斯消元法整理线性基，使其具有阶梯状结构
for i in range(BIT + 1):
 for j in range(i + 1, BIT + 1):
 if (basis[j] & (1 << i)) != 0:
 basis[j] ^= basis[i]

重新整理 basis 数组，把非 0 的元素移到前面
temp_basis = [b for b in basis if b != 0]

判断是否能异或出 0
zero = (len(temp_basis) != n)

return temp_basis, len(temp_basis), zero

```

```
def query(k, basis, len_basis, zero):
```

```
"""
返回第 k 小的异或和
```

算法思路：

1. 特殊情况处理：如果能异或出 0，0 是第 1 小的
2. 如果能异或出 0，且  $k=1$ ，返回 0
3. 如果能异或出 0，且  $k>1$ ，将  $k$  减 1 后继续处理
4. 根据  $k$  的二进制表示选择线性基中的元素进行异或

参数：

k: 查询的第  $k$  小值  
 basis: 线性基数组  
 len\_basis: 线性基大小  
 zero: 是否能异或出 0

返回：

long: 第  $k$  小的异或和，如果不存在则返回-1

```
"""
异常处理: k 必须大于 0
if k <= 0:
 raise ValueError("k must be positive")

如果能异或出 0, 那么 0 是第 1 小的
if zero:
 if k == 1:
 return 0
 k -= 1 # 跳过 0

如果 k 超过了可能的异或和数量, 返回-1
if k > (1 << len_basis):
 return -1

ans = 0
根据 k 的二进制表示选择线性基中的元素进行异或
for i in range(len_basis):
 if (k & (1 << i)) != 0:
 ans ^= basis[i]

return ans

def main():
 """主函数"""
 input = sys.stdin.read
 data = input().split()

 idx = 0
 t = int(data[idx])
 idx += 1

 for cases in range(1, t + 1):
 print(f"Case #{cases}: ")

 n = int(data[idx])
 idx += 1

 arr = []
 for i in range(n):
 arr.append(int(data[idx]))
 idx += 1
```

```
basis, len_basis, zero = compute(arr, 60)

q = int(data[idx])
idx += 1

for i in range(q):
 k = int(data[idx])
 idx += 1
 try:
 result = query(k, basis, len_basis, zero)
 print(result)
 except ValueError:
 print(-1) # 发生异常时输出-1

if __name__ == "__main__":
 main()

,,,
```

文件: Code06\_CF1101G.cpp

```
=====
// Codeforces 1101G (Zero XOR Subset)-less 问题 (线性基应用)
// 题目来源: Codeforces 1101G (Zero XOR Subset)-less
// 题目链接: https://codeforces.com/problemset/problem/1101/G
// 题目描述: 给定一个长度为 n 的数组, 将数组分成尽可能多的段,
// 使得每一段的异或和都不为 0, 求最多能分成多少段
// 算法: 线性基
// 时间复杂度: O(n * log(max_value))
// 空间复杂度: O(n + log(max_value))
// 测试链接 : https://codeforces.com/problemset/problem/1101/G

// 由于编译环境限制, 不使用标准库头文件和 IO 函数
// 此代码仅展示算法实现逻辑
```

```
const int MAXN = 200001;
const int BIT = 30;

int arr[MAXN];
```

```

int prefix[MAXN];
int basis[BIT + 1];
int n;

// 线性基里插入 num, 如果线性基增加了返回 true, 否则返回 false
bool insert(int num) {
 for (int i = BIT; i >= 0; i--) {
 if ((num & (1 << i)) != 0) {
 if (basis[i] == 0) {
 basis[i] = num;
 return true;
 }
 num ^= basis[i];
 }
 }
 return false;
}

// 普通消元法构建线性基
// 返回线性基的大小
// 算法思路:
// 1. 清空线性基
// 2. 将所有前缀异或和插入线性基
// 3. 返回线性基的大小
int compute() {
 // 清空线性基
 for (int i = 0; i <= BIT; i++) {
 basis[i] = 0;
 }

 int size = 0;
 // 将所有前缀异或和插入线性基
 for (int i = 0; i <= n; i++) {
 if (insert(prefix[i])) {
 size++;
 }
 }
 return size;
}

/*
线性基在数组分割问题中的应用

```

这是一道线性基的经典应用题，考察如何将数组分割成最多的段，使得每段异或和不为 0。

解题思路：

1. 问题转化：

- 要使每段异或和不为 0，等价于不存在一个子数组的异或和为 0
- 一个子数组  $[i, j]$  的异或和为 0，等价于  $\text{prefix}[j] \wedge \text{prefix}[i-1] = 0$ ，即  $\text{prefix}[j] = \text{prefix}[i-1]$
- 因此，问题转化为：选择最多的分割点，使得不存在两个前缀异或和相等

2. 线性基应用：

- 将所有前缀异或和看作向量，构建线性基
- 线性基的大小就是线性无关的前缀异或和的个数
- 这些线性无关的前缀异或和可以构成最多分割段数

3. 特殊情况处理：

- 如果整个数组的异或和为 0，则无法分割，返回 -1
- 否则答案为线性基大小减 1（因为线性基中包含 0）

时间复杂度分析：

- 构建前缀异或和： $O(n)$
- 构建线性基： $O(n * \log(\max\_value))$
- 总体： $O(n * \log(\max\_value))$

空间复杂度分析：

- $O(n + \log(\max\_value))$ ，用于存储前缀异或和和线性基

相关题目：

1. <https://codeforces.com/problemset/problem/1101/G> – (Zero XOR Subset)-less (本题)
  2. <http://acm.hdu.edu.cn/showproblem.php?pid=3949> – HDU 3949 XOR
  3. <https://loj.ac/p/114> – 第 k 小异或和
  4. <https://www.luogu.com.cn/problem/P3812> – 线性基（最大异或和）
  5. <https://www.luogu.com.cn/problem/P4570> – 元素（线性基+贪心）
  6. <https://www.luogu.com.cn/problem/P3857> – 彩灯（线性基应用）
- \*/

=====

文件：Code06\_CF1101G.java

=====

```
// Codeforces 1101G (Zero XOR Subset)-less 问题 (线性基应用)
// 题目来源: Codeforces 1101G (Zero XOR Subset)-less
// 题目链接: https://codeforces.com/problemset/problem/1101/G
// 题目描述: 给定一个长度为 n 的数组, 将数组分成尽可能多的段,
```

```
// 使得每一段的异或和都不为 0，求最多能分成多少段
// 算法：线性基
// 时间复杂度：O(n * log(max_value))
// 空间复杂度：O(n + log(max_value))
// 测试链接：https://codeforces.com/problemset/problem/1101/G
```

```
package class136;
```

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

public class Code06_CF1101G {
 public static int[] arr; // 输入数组
 public static int[] prefix; // 前缀异或和数组
 public static long[] basis; // 线性基数组
 public static int n; // 数组长度
 public static final int BIT = 30; // 最大位数
```

```
// 线性基里插入 num，如果线性基增加了返回 true，否则返回 false
public static boolean insert(long num) {
```

```
 for (int i = BIT; i >= 0; i--) {
 if ((num >> i & 1) == 1) {
 if (basis[i] == 0) {
 basis[i] = num;
 return true;
 }
 num ^= basis[i];
 }
 }
 return false;
}
```

```
// 普通消元法构建线性基
```

```
// 返回线性基的大小
```

```
// 算法思路：
```

```
// 1. 清空线性基
// 2. 将所有前缀异或和插入线性基
// 3. 返回线性基的大小
```

```
public static int compute() {
```

```
 // 清空线性基
 Arrays.fill(basis, 0);
```

```
 int size = 0;
```

```

// 将所有前缀异或插入线性基
for (int i = 0; i <= n; i++) {
 if (insert(prefix[i])) {
 size++;
 }
}
return size;
}

public static void main(String[] args) throws IOException {
 BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
 PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));

 n = Integer.parseInt(br.readLine());
 arr = new int[n];
 prefix = new int[n + 1];
 basis = new long[BIT + 1];

 StringTokenizer st = new StringTokenizer(br.readLine());
 for (int i = 0; i < n; i++) {
 arr[i] = Integer.parseInt(st.nextToken());
 }

 // 计算前缀异或和
 prefix[0] = 0;
 for (int i = 1; i <= n; i++) {
 prefix[i] = prefix[i - 1] ^ arr[i - 1];
 }

 // 特殊情况处理：如果整个数组的异或和为 0，则无法分割，返回-1
 if (prefix[n] == 0) {
 out.println(-1);
 } else {
 // 否则答案为线性基大小减 1（因为线性基中包含 0）
 int result = compute() - 1;
 out.println(result);
 }

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

```

文件: Code06\_CF1101G.py

```
=====
Codeforces 1101G (Zero XOR Subset)-less 问题 (线性基应用)
题目来源: Codeforces 1101G (Zero XOR Subset)-less
题目链接: https://codeforces.com/problemset/problem/1101/G
题目描述: 给定一个长度为 n 的数组, 将数组分成尽可能多的段,
使得每一段的异或和都不为 0, 求最多能分成多少段
算法: 线性基
时间复杂度: O(n * log(max_value))
空间复杂度: O(n + log(max_value))
测试链接 : https://codeforces.com/problemset/problem/1101/G
```

```
def insert(num, basis, BIT):
```

```
 """
```

线性基里插入 num, 如果线性基增加了返回 true, 否则返回 false

参数:

num: 要插入的数字

basis: 线性基数组

BIT: 最大位数

返回:

bool: 插入是否成功

```
 """
```

```
for i in range(BIT, -1, -1):
```

```
 if (num >> i) & 1:
```

```
 if basis[i] == 0:
```

```
 basis[i] = num
```

```
 return True
```

```
 num ^= basis[i]
```

```
return False
```

```
def compute(prefix, BIT):
```

```
 """
```

普通消元法构建线性基

返回线性基的大小

算法思路:

1. 清空线性基
2. 将所有前缀异或和插入线性基

### 3. 返回线性基的大小

参数:

prefix: 前缀异或和数组

BIT: 最大位数

返回:

int: 线性基的大小

"""

# 初始化线性基

basis = [0] \* (BIT + 1)

size = 0

# 将所有前缀异或和插入线性基

for p in prefix:

if insert(p, basis, BIT):

size += 1

return size

def main():

"""主函数"""

n = int(input())

arr = list(map(int, input().split()))

# 计算前缀异或和

prefix = [0] \* (n + 1)

for i in range(1, n + 1):

prefix[i] = prefix[i - 1] ^ arr[i - 1]

# 特殊情况处理: 如果整个数组的异或和为 0, 则无法分割, 返回-1

if prefix[n] == 0:

print(-1)

else:

# 否则答案为线性基大小减 1 (因为线性基中包含 0)

result = compute(prefix, 30) - 1

print(result)

if \_\_name\_\_ == "\_\_main\_\_":

main()

=====

文件: Code07\_Bzoj2460.cpp

```
=====

// BZOJ 2460 元素问题 (线性基+贪心)
// 题目来源: BZOJ 2460 元素
// 题目链接: https://www.lydsy.com/JudgeOnline/problem.php?id=2460
// 题目描述: 有 n 个二元组 (ai, bi), 要求选出一些二元组,
// 使得这些二元组的 a 值异或和不为 0, 且 b 值和最大
// 算法: 线性基 + 贪心
// 时间复杂度: O(n * log(n) + n * log(max_value))
// 空间复杂度: O(n + log(max_value))
// 测试链接 : https://www.lydsy.com/JudgeOnline/problem.php?id=2460
```

```
// 由于编译环境限制, 不使用标准库头文件和 IO 函数
// 此代码仅展示算法实现逻辑
```

```
const int MAXN = 1001;
const int BIT = 60;
```

```
long long arr[MAXN][2]; // [a, b]
long long basis[BIT + 1];
int n;
```

```
// 线性基里插入 num, 如果线性基增加了返回 true, 否则返回 false
bool insert(long long num) {
 for (int i = BIT; i >= 0; i--) {
 if ((num & (1LL << i)) != 0) {
 if (basis[i] == 0) {
 basis[i] = num;
 return true;
 }
 num ^= basis[i];
 }
 }
 return false;
}
```

```
// 普通消元法构建线性基
// 返回最大 b 值和
// 算法思路:
// 1. 按 b 值从大到小排序, 贪心选择
// 2. 清空线性基
// 3. 依次尝试插入每个元素的 a 值
// 4. 如果能成功插入, 则选择该二元组
```

```

long long compute() {
 long long ans = 0;
 // 清空线性基
 for (int i = 0; i <= BIT; i++) {
 basis[i] = 0;
 }

 // 按 b 值从大到小排序，贪心选择
 // 这里省略排序实现，实际应用中需要实现排序算法

 // 依次尝试插入每个元素
 for (int i = 1; i <= n; i++) {
 if (insert(arr[i][0])) {
 ans += arr[i][1];
 }
 }
 return ans;
}

/*
线性基与贪心算法结合

```

这是线性基与贪心算法结合的经典例题，要求在保证异或和不为 0 的前提下，选择二元组使得 b 值和最大。

解题思路：

1. 贪心策略：
  - 为了使 b 值和最大，应该优先选择 b 值大的二元组
  - 将所有二元组按 b 值从大到小排序
2. 线性基应用：
  - 依次尝试将每个二元组的 a 值插入线性基
  - 如果能成功插入（线性基增加），则选择该二元组
  - 如果不能插入，说明该 a 值可以由已选的 a 值异或得到，不能选择
3. 算法正确性：
  - 由于线性基的性质，我们按 b 值从大到小排序后依次选择，可以保证得到最大 b 值和
  - 如果存在更优的选择方案，那么一定存在 b 值更大但未被选择的二元组，这与我们的贪心策略矛盾

时间复杂度分析：

- 排序:  $O(n * \log(n))$
- 构建线性基:  $O(n * \log(\max\_value))$
- 总体:  $O(n * \log(n) + n * \log(\max\_value))$

空间复杂度分析:

- $O(n + \log(\max\_value))$ , 用于存储二元组和线性基

相关题目:

1. <https://www.lydsy.com/JudgeOnline/problem.php?id=2460> - BZOJ 2460 元素 (本题)
  2. <https://www.luogu.com.cn/problem/P4570> - 元素 (线性基+贪心)
  3. <http://acm.hdu.edu.cn/showproblem.php?pid=3949> - HDU 3949 XOR
  4. <https://loj.ac/p/114> - 第 k 小异或和
  5. <https://www.luogu.com.cn/problem/P3812> - 线性基 (最大异或和)
  6. <https://www.luogu.com.cn/problem/P3857> - 彩灯 (线性基应用)
- \*/

=====

文件: Code07\_Bzoj2460.java

```
// BZOJ 2460 元素问题 (线性基+贪心)
// 题目来源: BZOJ 2460 元素
// 题目链接: https://www.lydsy.com/JudgeOnline/problem.php?id=2460
// 题目描述: 有 n 个二元组 (ai, bi), 要求选出一些二元组,
// 使得这些二元组的 a 值异或和不为 0, 且 b 值和最大
// 算法: 线性基 + 贪心
// 时间复杂度: O(n * log(n) + n * log(max_value))
// 空间复杂度: O(n + log(max_value))
// 测试链接 : https://www.lydsy.com/JudgeOnline/problem.php?id=2460
```

```
package class136;
```

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

public class Code07_Bzoj2460 {
 public static long[][] arr; // [a, b]
 public static long[] basis; // 线性基数组
 public static int n; // 数组长度
 public static final int BIT = 60; // 最大位数
```

```
// 线性基里插入 num, 如果线性基增加了返回 true, 否则返回 false
public static boolean insert(long num) {
```

```

 for (int i = BIT; i >= 0; i--) {
 if ((num >> i & 1) == 1) {
 if (basis[i] == 0) {
 basis[i] = num;
 return true;
 }
 num ^= basis[i];
 }
 }
 return false;
 }

// 普通消元法构建线性基
// 返回最大 b 值和
// 算法思路:
// 1. 按 b 值从大到小排序, 贪心选择
// 2. 清空线性基
// 3. 依次尝试插入每个元素的 a 值
// 4. 如果能成功插入, 则选择该二元组
public static long compute() {
 // 按 b 值从大到小排序
 Arrays.sort(arr, 0, n, (a, b) -> Long.compare(b[1], a[1]));

 // 清空线性基
 Arrays.fill(basis, 0);

 long ans = 0;
 // 依次尝试插入每个元素
 for (int i = 0; i < n; i++) {
 if (insert(arr[i][0])) {
 ans += arr[i][1];
 }
 }
 return ans;
}

public static void main(String[] args) throws IOException {
 BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
 PrintWriter out = new PrintWriter(new OutputStreamWriter(System.out));

 n = Integer.parseInt(br.readLine());
 arr = new long[n][2];
 basis = new long[BIT + 1];
}

```

```

for (int i = 0; i < n; i++) {
 StringTokenizer st = new StringTokenizer(br.readLine());
 arr[i][0] = Long.parseLong(st.nextToken()); // a 值
 arr[i][1] = Long.parseLong(st.nextToken()); // b 值
}

out.println(compute());

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

```

=====

文件: Code07\_Bzoj2460.py

=====

```

BZOJ 2460 元素问题（线性基+贪心）
题目来源: BZOJ 2460 元素
题目链接: https://www.lydsy.com/JudgeOnline/problem.php?id=2460
题目描述: 有 n 个二元组 (ai, bi), 要求选出一些二元组,
使得这些二元组的 a 值异或和不为 0, 且 b 值和最大
算法: 线性基 + 贪心
时间复杂度: O(n * log(n) + n * log(max_value))
空间复杂度: O(n + log(max_value))
测试链接 : https://www.lydsy.com/JudgeOnline/problem.php?id=2460

```

def insert(num, basis, BIT):

"""

线性基里插入 num, 如果线性基增加了返回 true, 否则返回 false

参数:

num: 要插入的数字

basis: 线性基数组

BIT: 最大位数

返回:

bool: 插入是否成功

"""

```

for i in range(BIT, -1, -1):
 if (num >> i) & 1:

```

```
 if basis[i] == 0:
 basis[i] = num
 return True
 num ^= basis[i]
return False
```

```
def compute(elements, BIT):
```

```
 """
```

```
 普通消元法构建线性基
```

```
 返回最大 b 值和
```

算法思路：

1. 按 b 值从大到小排序，贪心选择
2. 清空线性基
3. 依次尝试插入每个元素的 a 值
4. 如果能成功插入，则选择该二元组

参数：

elements: 二元组列表，每个元素为(a, b)的元组

BIT: 最大位数

返回：

long: 最大 b 值和

```
"""
```

```
按 b 值从大到小排序
```

```
elements.sort(key=lambda x: x[1], reverse=True)
```

```
初始化线性基
```

```
basis = [0] * (BIT + 1)
```

```
ans = 0
```

```
依次尝试插入每个元素
```

```
for a, b in elements:
```

```
 if insert(a, basis, BIT):
```

```
 ans += b
```

```
return ans
```

```
def main():
```

```
 """主函数"""

```

```
n = int(input())
```

```
elements = []
```

```
for _ in range(n):
 line = input().split()
 a = int(line[0])
 b = int(line[1])
 elements.append((a, b))

result = compute(elements, 60)
print(result)

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

=====

文件: COMPREHENSIVE\_TEST.java

=====

```
package class136;

import java.util.*;

/**
 * 线性基算法综合测试类
 *
 * 功能:
 * 1. 单元测试: 验证所有线性基算法的正确性
 * 2. 性能测试: 测试算法在大数据量下的表现
 * 3. 边界测试: 测试各种边界情况
 * 4. 异常测试: 测试异常输入的处理
 *
 * 测试覆盖:
 * - 最大异或和问题
 * - 第 k 小异或和问题
 * - 线性基+贪心问题
 * - 线性基应用问题
 *
 * 测试策略:
 * - 小数据集: 验证基本功能
 * - 大数据集: 验证性能表现
 * - 边界数据: 验证鲁棒性
 * - 随机数据: 验证通用性
 */
public class COMPREHENSIVE_TEST {
```

```
// ===== 单元测试方法 =====

/**
 * 测试最大异或和算法
 */
public static void testMaximumXor() {
 System.out.println("==> 测试最大异或和算法 ==>");

 // 测试用例 1: 普通情况
 long[] arr1 = {3, 10, 5, 25, 2, 8};
 long expected1 = 28; // 5^25 = 28
 long result1 = testMaximumXorHelper(arr1);
 System.out.println("测试用例 1 - 普通情况: " + (result1 == expected1 ? "通过" : "失败"));

 // 测试用例 2: 线性相关情况
 long[] arr2 = {1, 2, 3}; // 1^2 = 3, 线性相关
 long expected2 = 3;
 long result2 = testMaximumXorHelper(arr2);
 System.out.println("测试用例 2 - 线性相关: " + (result2 == expected2 ? "通过" : "失败"));

 // 测试用例 3: 空数组
 long[] arr3 = {};
 long expected3 = 0;
 long result3 = testMaximumXorHelper(arr3);
 System.out.println("测试用例 3 - 空数组: " + (result3 == expected3 ? "通过" : "失败"));

 // 测试用例 4: 单元素数组
 long[] arr4 = {5};
 long expected4 = 5;
 long result4 = testMaximumXorHelper(arr4);
 System.out.println("测试用例 4 - 单元素: " + (result4 == expected4 ? "通过" : "失败"));

 // 测试用例 5: 全 0 数组
 long[] arr5 = {0, 0, 0};
 long expected5 = 0;
 long result5 = testMaximumXorHelper(arr5);
 System.out.println("测试用例 5 - 全 0 数组: " + (result5 == expected5 ? "通过" : "失败"));

 System.out.println();
}

private static long testMaximumXorHelper(long[] arr) {
 // 使用 Code01_MaximumXor 的算法逻辑进行测试
}
```

```

int n = arr.length;
long[] basis = new long[51]; // BIT=50

// 构建线性基
for (int i = 0; i < n; i++) {
 long num = arr[i];
 for (int j = 50; j >= 0; j--) {
 if ((num >> j & 1) == 1) {
 if (basis[j] == 0) {
 basis[j] = num;
 break;
 } else {
 num ^= basis[j];
 }
 }
 }
}

// 计算最大异或值
long ans = 0;
for (int i = 50; i >= 0; i--) {
 ans = Math.max(ans, ans ^ basis[i]);
}
return ans;
}

/***
 * 测试第 k 小异或和算法
 */
public static void testKthXor() {
 System.out.println("==> 测试第 k 小异或和算法 ==<");

 // 测试用例 1: 普通情况
 long[] arr1 = {1, 2, 3};
 long[] expected1 = {1, 2, 3, 0}; // 第 1 小:1, 第 2 小:2, 第 3 小:3, 第 4 小:0
 boolean pass1 = true;
 for (int k = 1; k <= 4; k++) {
 long result = testKthXorHelper(arr1, k);
 if (result != expected1[k-1]) {
 pass1 = false;
 break;
 }
 }
}

```

```

System.out.println("测试用例 1 - 普通情况: " + (pass1 ? "通过" : "失败"));

// 测试用例 2: 线性无关情况
long[] arr2 = {1, 4};
long[] expected2 = {1, 3, 4, 5}; // 第 1 小:1, 第 2 小:3, 第 3 小:4, 第 4 小:5
boolean pass2 = true;
for (int k = 1; k <= 4; k++) {
 long result = testKthXorHelper(arr2, k);
 if (result != expected2[k-1]) {
 pass2 = false;
 break;
 }
}
System.out.println("测试用例 2 - 线性无关: " + (pass2 ? "通过" : "失败"));

System.out.println();
}

private static long testKthXorHelper(long[] arr, long k) {
 // 简化的第 k 小异或和算法
 int n = arr.length;

 // 构建线性基
 long[] basis = new long[51];
 int basisSize = 0;

 for (int i = 0; i < n; i++) {
 long num = arr[i];
 for (int j = 50; j >= 0; j--) {
 if ((num >> j & 1) == 1) {
 if (basis[j] == 0) {
 basis[j] = num;
 basisSize++;
 break;
 } else {
 num ^= basis[j];
 }
 }
 }
 }

 // 判断是否能异或出 0
 boolean canGetZero = (basisSize != n);
}

```

```

// 计算第 k 小值
if (k == 1 && canGetZero) {
 return 0;
}

// 简化实现：收集所有可能的异或值
Set<Long> xorValues = new HashSet<>();
collectXorValues(basis, 0, 0, xorValues);

List<Long> sortedValues = new ArrayList<>(xorValues);
Collections.sort(sortedValues);

if (k <= sortedValues.size()) {
 return sortedValues.get((int)k - 1);
} else {
 return -1; // 超出范围
}
}

private static void collectXorValues(long[] basis, int index, long current, Set<Long> result)
{
 if (index == basis.length) {
 if (current != 0) {
 result.add(current);
 }
 return;
 }

 // 不选择当前基
 collectXorValues(basis, index + 1, current, result);

 // 选择当前基（如果存在）
 if (basis[index] != 0) {
 collectXorValues(basis, index + 1, current ^ basis[index], result);
 }
}

// ===== 性能测试方法 =====

/**
 * 性能测试：测试算法在大数据量下的表现
 */

```

```
public static void performanceTest() {
 System.out.println("== 性能测试 ==");

 // 生成测试数据
 int[] sizes = {1000, 5000, 10000};

 for (int size : sizes) {
 long[] testData = generateRandomArray(size, 1000000);

 long startTime = System.currentTimeMillis();
 long result = testMaximumXorHelper(testData);
 long endTime = System.currentTimeMillis();

 System.out.println("数据量: " + size + ", 耗时: " + (endTime - startTime) + "ms");
 }

 System.out.println();
}

private static long[] generateRandomArray(int size, int maxValue) {
 long[] arr = new long[size];
 Random random = new Random();
 for (int i = 0; i < size; i++) {
 arr[i] = Math.abs(random.nextLong()) % maxValue;
 }
 return arr;
}

// ====== 边界测试方法 ======

/**
 * 边界测试: 测试各种边界情况
 */
public static void boundaryTest() {
 System.out.println("== 边界测试 ==");

 // 测试超大数值
 long[] largeValues = {Long.MAX_VALUE, Long.MAX_VALUE - 1};
 long result1 = testMaximumXorHelper(largeValues);
 System.out.println("超大数值测试: " + (result1 >= 0 ? "通过" : "失败"));

 // 测试负数 (需要特殊处理)
 long[] negativeValues = {-1, -2}; // 负数在异或运算中需要特殊处理
}
```

```
try {
 long result2 = testMaximumXorHelper(negativeValues);
 System.out.println("负数测试: 通过");
} catch (Exception e) {
 System.out.println("负数测试: 失败 - " + e.getMessage());
}

// 测试重复数据
long[] duplicateValues = {1, 1, 1, 1};
long result3 = testMaximumXorHelper(duplicateValues);
System.out.println("重复数据测试: " + (result3 == 1 ? "通过" : "失败"));

System.out.println();
}

// ====== 异常测试方法 ======

/***
 * 异常测试: 测试异常输入的处理
 */
public static void exceptionTest() {
 System.out.println("== 异常测试 ==");

 // 测试空指针
 try {
 testMaximumXorHelper(null);
 System.out.println("空指针测试: 失败 - 应该抛出异常");
 } catch (NullPointerException e) {
 System.out.println("空指针测试: 通过");
 } catch (Exception e) {
 System.out.println("空指针测试: 通过 (抛出其他异常)");
 }
}

// 测试非法 k 值
try {
 testKthXorHelper(new long[]{1, 2}, 0);
 System.out.println("非法 k 值测试: 失败 - 应该抛出异常");
} catch (IllegalArgumentException e) {
 System.out.println("非法 k 值测试: 通过");
} catch (Exception e) {
 System.out.println("非法 k 值测试: 通过 (抛出其他异常)");
}
```

```
 System.out.println();
 }

// ====== 主测试方法 ======

public static void main(String[] args) {
 System.out.println("线性基算法综合测试");
 System.out.println("=====");
 System.out.println();

 // 运行所有测试
 testMaximumXor();
 testKthXor();
 performanceTest();
 boundaryTest();
 exceptionTest();

 System.out.println("所有测试完成!");
}

}
```

文件: ShowDetails.java

```
=====
package class136;

// 课上讲述普通消元、高斯消元构建异或空间线性基的例子

import java.util.Arrays;
import java.util.HashSet;

public class ShowDetails {

 public static int MAXN = 101;

 // 最高位从哪开始
 public static int BIT = 60;

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

 public static int n;
```

```
// 普通消元
public static long[] basis1 = new long[BIT + 1];

public static boolean zero1;

public static void compute1() {
 zero1 = false;
 for (int i = 1; i <= n; i++) {
 if (!insert(arr[i])) {
 zero1 = true;
 }
 }
}
```

```
// 线性基里插入 num, 如果线性基增加了返回 true, 否则返回 false
public static boolean insert(long num) {
 for (int i = BIT; i >= 0; i--) {
 if (num >> i == 1) {
 if (basis1[i] == 0) {
 basis1[i] = num;
 return true;
 }
 num ^= basis1[i];
 }
 }
 return false;
}
```

```
// 高斯消元
// 因为不需要维护主元和自由元的依赖关系
// 所以高斯消元的写法可以得到简化
public static long[] basis2 = new long[MAXN];
```

```
public static int len;

public static boolean zero2;

public static void compute2() {
 len = 1;
 for (long i = BIT; i >= 0; i--) {
 for (int j = len; j <= n; j++) {
 if ((basis2[j] & (1L << i)) != 0) {
 swap(j, len);
 }
 }
 }
}
```

```

 break;
 }
}
if ((basis2[len] & (1L << i)) != 0) {
 for (int j = 1; j <= n; j++) {
 if (j != len && (basis2[j] & (1L << i)) != 0) {
 basis2[j] ^= basis2[len];
 }
 }
 len++;
}
len--;
zero2 = len != n;
}

```

```

public static void swap(int a, int b) {
 long tmp = basis2[a];
 basis2[a] = basis2[b];
 basis2[b] = tmp;
}

```

```

public static void main(String[] args) {
 // 课上讲的普通消元，例子 1
 // 12, 9, 14, 11
 System.out.println("例子 1");
 Arrays.fill(basis1, 0);
 arr[1] = 12;
 arr[2] = 9;
 arr[3] = 14;
 arr[4] = 11;
 n = 4;
 System.out.println("原始数组得到的异或结果如下");
 printXor(arr, n);
}

```

```

System.out.println("=====");
System.out.println("普通消元得到的线性基 : ");
compute1();
long[] b1 = new long[MAXN];
int s1 = 0;
for (int i = BIT; i >= 0; i--) {
 if (basis1[i] != 0) {
 System.out.print(basis1[i] + " ");
 }
}

```

```

 b1[++s1] = basis1[i];
 }
}

System.out.println();
System.out.println("是否能异或出 0 : " + zero1);
System.out.println("普通消元得到的异或结果如下");
printXor(b1, s1);
System.out.println("=====");

System.out.println();
System.out.println();

// 课上讲的普通消元，例子 2
// 2, 5, 11, 6
System.out.println("例子 2");
Arrays.fill(basis1, 0);
arr[1] = 2;
arr[2] = 5;
arr[3] = 11;
arr[4] = 6;
n = 4;
System.out.println("原始数组得到的异或结果如下");
printXor(arr, n);
System.out.println("=====");
System.out.println("普通消元得到的线性基 : ");
compute1();
long[] b2 = new long[MAXN];
int s2 = 0;
for (int i = BIT; i >= 0; i--) {
 if (basis1[i] != 0) {
 System.out.print(basis1[i] + " ");
 b2[++s2] = basis1[i];
 }
}
System.out.println();
System.out.println("是否能异或出 0 : " + zero1);
System.out.println("普通消元得到的异或结果如下");
printXor(b2, s2);
System.out.println("=====");

System.out.println();
System.out.println();

```

```
// 课上讲的高斯消元的例子，例子 3
// 6, 37, 35, 33
System.out.println("例子 3");
Arrays.fill(basis2, 0);
arr[1] = basis2[1] = 6;
arr[2] = basis2[2] = 37;
arr[3] = basis2[3] = 35;
arr[4] = basis2[4] = 33;
n = 4;
System.out.println("原始数组得到的异或结果如下");
printXor(arr, n);
System.out.println("=====");
System.out.println("高斯消元得到的线性基 : ");
compute2();
for (int i = 1; i <= len; i++) {
 System.out.print(basis2[i] + " ");
}
System.out.println();
System.out.println("是否能异或出 0 : " + zero2);
System.out.println("高斯消元得到的异或结果如下");
printXor(basis2, len);
System.out.println("=====");

System.out.println();
System.out.println();

// 课上讲的高斯消元的例子，例子 4
// 如果想得到第 k 小的非 0 异或和
// 必须用高斯消元，不能用普通消元，比如
// arr = { 7, 10, 4 }
// 普通消元得到的异或空间线性基是 : 10 7 3
// 第三小异或和是 4，第四小异或和是 10，这是错误的
// 高斯消元可以得到正确结果
// 高斯消元得到的异或空间线性基是 : 9 4 3
// 第三小异或和是 7，第四小异或和是 9，这是正确的
System.out.println("例子 4");
Arrays.fill(basis1, 0);
Arrays.fill(basis2, 0);
arr[1] = basis2[1] = 7;
arr[2] = basis2[2] = 10;
arr[3] = basis2[3] = 4;
n = 3;
System.out.println("原始数组得到的异或结果如下");
```

```

printXor(arr, n);
System.out.println("=====");
System.out.println("普通消元");
compute1();
for (int i = BIT; i >= 0; i--) {
 if (basis1[i] != 0) {
 System.out.print(basis1[i] + " ");
 }
}
System.out.println();
System.out.println("是否能异或出 0 : " + zero1);

System.out.println("=====");

System.out.println("高斯消元");
compute2();
for (int i = 1; i <= len; i++) {
 System.out.print(basis2[i] + " ");
}
System.out.println();
System.out.println("是否能异或出 0 : " + zero2);
}

// nums 中 1~n 的范围
// 所有可能的异或和打印出来
// 要求去重并且至少选择一个数字
public static void printXor(long[] nums, int n) {
 HashSet<Long> set = new HashSet<>();
 dfs(nums, n, 1, false, 0, set);
 System.out.println("至少选择一个数字所有可能的异或和:");
 for (Long s : set) {
 System.out.print(s + ", ");
 }
 System.out.println();
}

// 当前来到 i 位置
// 之前是否选择过数字用 pick 表示
// 之前做的决定形成的异或和是 path
// 当前 i 位置的数字要或者不要全决策
// 收集所有可能的异或和
public static void dfs(long[] nums, int n, int i, boolean pick, long path, HashSet<Long> set)
{

```

```

if (i > n) {
 if (pick) {
 set.add(path);
 }
} else {
 dfs(nums, n, i + 1, pick, path, set);
 dfs(nums, n, i + 1, true, path ^ nums[i], set);
}
}

/*
* 线性基构建方法详解与对比
*
* 线性基有两种主要的构建方法：普通消元法和高斯消元法。两种方法各有特点和适用场景。
*
* 普通消元法（逐位插入法）：
*
* 特点：
* 1. 从最高位开始扫描，逐个插入向量
* 2. 插入过程中，如果当前位在线性基中为空，则直接插入；否则用线性基中该位的向量异或当前向量
* 3. 实现简单，代码较短
*
* 适用场景：
* 1. 求最大异或值
* 2. 判断某个值是否可以被线性表示
* 3. 一般性的线性基问题
*
* 高斯消元法：
*
* 特点：
* 1. 类似于矩阵的行变换，构造阶梯形矩阵
* 2. 得到的线性基具有良好的结构，便于求第 k 小值
* 3. 实现相对复杂
*
* 适用场景：
* 1. 求第 k 小异或值
* 2. 需要利用线性基良好结构的场景
*
* 两种方法的对比：
*
* | 特性 | 普通消元法 | 高斯消元法 |
* |-----|-----|-----|
* | 实现难度 | 简单 | 复杂 |

```

```
* | 代码长度 | 短 | 长 |
* | 结构特性 | 无特定结构 | 阶梯状结构 |
* | 适用问题 | 最大值、判断可达性 | 第 k 小值 |
* | 时间复杂度 | $O(n \log(\max_value))$ | $O(n \log(\max_value))$ |
*
* 线性基的重要性质：
*
* 1. 线性基中的向量线性无关
* 2. 原向量集合中的任何向量都可以由线性基线性表示
* 3. 线性基中的任何非空子集异或和都不为 0
* 4. 线性基的大小是固定的，等于原向量集合的秩
*
* 应用场景总结：
*
* 1. 求最大异或值：使用普通消元法
* 2. 求第 k 小异或值：使用高斯消元法
* 3. 判断某个值是否可达：使用普通消元法
* 4. 计算可表示的向量个数：使用普通消元法，答案为 2^r (r 为线性基大小)
* 5. 带权值的线性基问题：结合贪心策略，使用普通消元法
*
* 相关题目：
*
* 1. https://www.luogu.com.cn/problem/P3812 - 线性基模板（最大异或和）
* 2. https://loj.ac/p/114 - 第 k 小异或和（高斯消元）
* 3. https://www.luogu.com.cn/problem/P4570 - 元素（线性基+贪心）
* 4. https://www.luogu.com.cn/problem/P3857 - 彩灯（线性基应用）
* 5. https://www.luogu.com.cn/problem/P4151 - 最大 XOR 和路径
* 6. https://www.luogu.com.cn/problem/P3292 - 幸运数字（线性基+倍增）
* 7. http://acm.hdu.edu.cn/showproblem.php?pid=3949 - HDU 3949 XOR（第 k 小异或和）
* 8. https://codeforces.com/problemset/problem/1101/G - (Zero XOR Subset)-less（线性基应用）
* 9. https://www.lydsy.com/JudgeOnline/problem.php?id=2460 - BZOJ 2460 元素（线性基+贪心）
*/
}
```

文件：TestLinearBasis.java

```
=====
// 线性基测试类
// 包含 Java、Python、C++三种语言实现的测试用例

package class136;

import java.util.*;
```

```
public class TestLinearBasis {
 // 测试用例 1: 最大异或和
 public static void testMaximumXor() {
 System.out.println("==> 测试最大异或和 ==>");
 long[] arr1 = {3, 10, 5, 25, 2, 8};
 long result1 = findMaximumXor(arr1);
 System.out.println("输入: [3, 10, 5, 25, 2, 8]");
 System.out.println("期望输出: 28 (5^25)");
 System.out.println("实际输出: " + result1);
 System.out.println("测试结果: " + (result1 == 28 ? "通过" : "失败"));
 System.out.println();
 }

 // 测试用例 2: 线性相关情况
 public static void testLinearDependent() {
 System.out.println("==> 测试线性相关情况 ==>");
 long[] arr2 = {1, 2, 3}; // 1^2 = 3, 线性相关
 long result2 = findMaximumXor(arr2);
 System.out.println("输入: [1, 2, 3]");
 System.out.println("期望输出: 3");
 System.out.println("实际输出: " + result2);
 System.out.println("测试结果: " + (result2 == 3 ? "通过" : "失败"));
 System.out.println();
 }

 // 测试用例 3: 空数组
 public static void testEmptyArray() {
 System.out.println("==> 测试空数组 ==>");
 long[] arr3 = {};
 long result3 = findMaximumXor(arr3);
 System.out.println("输入: []");
 System.out.println("期望输出: 0");
 System.out.println("实际输出: " + result3);
 System.out.println("测试结果: " + (result3 == 0 ? "通过" : "失败"));
 System.out.println();
 }

 // 测试用例 4: 单元素数组
 public static void testSingleElement() {
 System.out.println("==> 测试单元素数组 ==>");
 long[] arr4 = {5};
 long result4 = findMaximumXor(arr4);
```

```

System.out.println("输入: [5]");
System.out.println("期望输出: 5");
System.out.println("实际输出: " + result4);
System.out.println("测试结果: " + (result4 == 5 ? "通过" : "失败"));
System.out.println();
}

// 最大异或和实现（普通消元法）
public static long findMaximumXor(long[] nums) {
 if (nums == null || nums.length == 0) {
 return 0;
 }

 long[] basis = new long[64]; // 线性基数组

 // 构建线性基
 for (long num : nums) {
 insert(num, basis);
 }

 // 计算最大异或值
 long result = 0;
 for (int i = 63; i >= 0; i--) {
 if (basis[i] != 0) {
 result = Math.max(result, result ^ basis[i]);
 }
 }

 return result;
}

// 线性基插入操作
public static void insert(long num, long[] basis) {
 for (int i = 63; i >= 0; i--) {
 if ((num & (1L << i)) != 0) {
 if (basis[i] == 0) {
 basis[i] = num;
 return;
 }
 num ^= basis[i];
 }
 }
}
}

```

```

// 线性基查询第 k 小值
public static long queryKthXor(long[] nums, long k) {
 if (k <= 0) {
 throw new IllegalArgumentException("k must be positive");
 }

 long[] basis = new long[64];
 int basisSize = 0;

 // 构建线性基
 for (long num : nums) {
 if (insertWithReturn(num, basis)) {
 basisSize++;
 }
 }

 // 高斯消元
 for (int i = 0; i < 64; i++) {
 for (int j = i + 1; j < 64; j++) {
 if ((basis[j] & (1L << i)) != 0) {
 basis[j] ^= basis[i];
 }
 }
 }

 // 重新整理
 long[] gaussianBasis = new long[basisSize];
 int idx = 0;
 for (int i = 0; i < 64; i++) {
 if (basis[i] != 0) {
 gaussianBasis[idx++] = basis[i];
 }
 }

 // 判断是否能异或出 0
 boolean canGetZero = (basisSize != nums.length);

 // 查询第 k 小
 if (canGetZero) {
 if (k == 1) {
 return 0;
 }
 }
}

```

```

 k--;
 }

 if (k > (1L << basisSize)) {
 return -1;
 }

 long result = 0;
 for (int i = 0; i < basisSize; i++) {
 if ((k & (1L << i)) != 0) {
 result ^= gaussianBasis[i];
 }
 }

 return result;
}

// 带返回值的线性基插入操作
public static boolean insertWithReturn(long num, long[] basis) {
 for (int i = 63; i >= 0; i--) {
 if ((num & (1L << i)) != 0) {
 if (basis[i] == 0) {
 basis[i] = num;
 return true;
 }
 num ^= basis[i];
 }
 }
 return false;
}

public static void main(String[] args) {
 System.out.println("线性基算法测试");
 System.out.println("=====");
 // 运行所有测试用例
 testMaximumXor();
 testLinearDependent();
 testEmptyArray();
 testSingleElement();

 // 测试第 k 小异或和
 System.out.println("== 测试第 k 小异或和 ==");
}

```

```

long[] arr5 = {1, 2, 3};
System.out.println("输入: [1, 2, 3]");
for (int k = 1; k <= 4; k++) {
 long result = queryKthXor(arr5, k);
 System.out.println("第" + k + "小异或和: " + result);
}
System.out.println();

System.out.println("所有测试完成!");
}
}

=====

```

文件: test\_linear\_basis.cpp

```

// 线性基测试文件 (C++版本)

#include <iostream>
#include <vector>
#include <algorithm>
#include <cassert>

using namespace std;

const int BIT = 63;

// 线性基插入操作
void insert(long long num, long long basis[]) {
 for (int i = BIT; i >= 0; i--) {
 if ((num & (1LL << i)) != 0) {
 if (basis[i] == 0) {
 basis[i] = num;
 return;
 }
 num ^= basis[i];
 }
 }
}

// 带返回值的线性基插入操作
bool insertWithReturn(long long num, long long basis[]) {
 for (int i = BIT; i >= 0; i--) {

```

```

 if ((num & (1LL << i)) != 0) {
 if (basis[i] == 0) {
 basis[i] = num;
 return true;
 }
 num ^= basis[i];
 }
}
return false;
}

```

// 计算最大异或和

```

long long findMaximumXor(const vector<long long>& nums) {
 if (nums.empty()) {
 return 0;
 }
}

```

```
long long basis[BIT + 1] = {0}; // 初始化线性基
```

// 构建线性基

```

for (long long num : nums) {
 insert(num, basis);
}

```

// 计算最大异或值

```

long long result = 0;
for (int i = BIT; i >= 0; i--) {
 if (basis[i] != 0) {
 result = max(result, result ^ basis[i]);
 }
}

```

```
return result;
```

```
}
```

// 查询第 k 小异或和

```

long long queryKthXor(const vector<long long>& nums, long long k) {
 if (k <= 0) {
 throw invalid_argument("k must be positive");
 }
}

```

```
long long basis[BIT + 1] = {0}; // 初始化线性基
int basisSize = 0;
```

```

// 构建线性基
for (long long num : nums) {
 if (insertWithReturn(num, basis)) {
 basisSize++;
 }
}

// 高斯消元
for (int i = 0; i <= BIT; i++) {
 for (int j = i + 1; j <= BIT; j++) {
 if ((basis[j] & (1LL << i)) != 0) {
 basis[j] ^= basis[i];
 }
 }
}

// 重新整理
vector<long long> gaussianBasis;
for (int i = 0; i <= BIT; i++) {
 if (basis[i] != 0) {
 gaussianBasis.push_back(basis[i]);
 }
}

// 判断是否能异或出 0
bool canGetZero = (gaussianBasis.size() != nums.size());

// 查询第 k 小
if (canGetZero) {
 if (k == 1) {
 return 0;
 }
 k--;
}

if (k > (1LL << gaussianBasis.size())) {
 return -1;
}

long long result = 0;
for (int i = 0; i < gaussianBasis.size(); i++) {
 if ((k & (1LL << i)) != 0) {

```

```

 result ^= gaussianBasis[i];
 }

}

return result;
}

// 测试用例 1: 最大异或和
void testMaximumXor() {
 cout << "==== 测试最大异或和 ===" << endl;
 vector<long long> arr1 = {3, 10, 5, 25, 2, 8};
 long long result1 = findMaximumXor(arr1);
 cout << "输入: [3, 10, 5, 25, 2, 8]" << endl;
 cout << "期望输出: 28 (5^25)" << endl;
 cout << "实际输出: " << result1 << endl;
 cout << "测试结果: " << (result1 == 28 ? "通过" : "失败") << endl;
 cout << endl;
}

// 测试用例 2: 线性相关情况
void testLinearDependent() {
 cout << "==== 测试线性相关情况 ===" << endl;
 vector<long long> arr2 = {1, 2, 3}; // 1^2 = 3, 线性相关
 long long result2 = findMaximumXor(arr2);
 cout << "输入: [1, 2, 3]" << endl;
 cout << "期望输出: 3" << endl;
 cout << "实际输出: " << result2 << endl;
 cout << "测试结果: " << (result2 == 3 ? "通过" : "失败") << endl;
 cout << endl;
}

// 测试用例 3: 空数组
void testEmptyArray() {
 cout << "==== 测试空数组 ===" << endl;
 vector<long long> arr3 = {};
 long long result3 = findMaximumXor(arr3);
 cout << "输入: []" << endl;
 cout << "期望输出: 0" << endl;
 cout << "实际输出: " << result3 << endl;
 cout << "测试结果: " << (result3 == 0 ? "通过" : "失败") << endl;
 cout << endl;
}

```

```

// 测试用例 4: 单元素数组
void testSingleElement() {
 cout << "==== 测试单元素数组 ===" << endl;
 vector<long long> arr4 = {5};
 long long result4 = findMaximumXor(arr4);
 cout << "输入: [5]" << endl;
 cout << "期望输出: 5" << endl;
 cout << "实际输出: " << result4 << endl;
 cout << "测试结果: " << (result4 == 5 ? "通过" : "失败") << endl;
 cout << endl;
}

// 测试第 k 小异或和
void testKthXor() {
 cout << "==== 测试第 k 小异或和 ===" << endl;
 vector<long long> arr5 = {1, 2, 3};
 cout << "输入: [1, 2, 3]" << endl;
 for (int k = 1; k <= 4; k++) {
 long long result = queryKthXor(arr5, k);
 cout << "第" << k << "小异或和: " << result << endl;
 }
 cout << endl;
}

int main() {
 cout << "线性基算法测试 (C++版本)" << endl;
 cout << "====="
 // 运行所有测试用例
 testMaximumXor();
 testLinearDependent();
 testEmptyArray();
 testSingleElement();
 testKthXor();

 cout << "所有测试完成!" << endl;

 return 0;
}
=====
```

文件: test\_linear\_basis.py

```
=====
```

```
线性基测试文件 (Python 版本)
```

```
def insert(num, basis):
```

```
 """
```

```
 线性基插入操作
```

```
参数:
```

```
 num: 要插入的数字
```

```
 basis: 线性基数组
```

```
返回:
```

```
 None
```

```
 """
```

```
for i in range(63, -1, -1):
```

```
 if (num >> i) & 1:
```

```
 if basis[i] == 0:
```

```
 basis[i] = num
```

```
 return
```

```
 num ^= basis[i]
```

```
def insert_with_return(num, basis):
```

```
 """
```

```
带返回值的线性基插入操作
```

```
参数:
```

```
 num: 要插入的数字
```

```
 basis: 线性基数组
```

```
返回:
```

```
 bool: 是否插入成功
```

```
 """
```

```
for i in range(63, -1, -1):
```

```
 if (num >> i) & 1:
```

```
 if basis[i] == 0:
```

```
 basis[i] = num
```

```
 return True
```

```
 num ^= basis[i]
```

```
return False
```

```
def find_maximum_xor(nums):
```

```
 """
```

```
计算最大异或和
```

参数:

    nums: 数字列表

返回:

    long: 最大异或和

"""

if not nums:

    return 0

# 初始化线性基

basis = [0] \* 64

# 构建线性基

for num in nums:

    insert(num, basis)

# 计算最大异或值

result = 0

for i in range(63, -1, -1):

    if basis[i] != 0:

        result = max(result, result ^ basis[i])

return result

def query\_kth\_xor(nums, k):

"""

查询第 k 小异或和

参数:

    nums: 数字列表

    k: 查询位置

返回:

    long: 第 k 小异或和

"""

if k <= 0:

    raise ValueError("k must be positive")

# 初始化线性基

basis = [0] \* 64

basis\_size = 0

```

构建线性基
for num in nums:
 if insert_with_return(num, basis):
 basis_size += 1

高斯消元
for i in range(64):
 for j in range(i + 1, 64):
 if (basis[j] & (1 << i)) != 0:
 basis[j] ^= basis[i]

重新整理
gaussian_basis = [b for b in basis if b != 0]

判断是否能异或出 0
can_get_zero = (len(gaussian_basis) != len(nums))

查询第 k 小
if can_get_zero:
 if k == 1:
 return 0
 k -= 1

 if k > (1 << len(gaussian_basis)):
 return -1

 result = 0
 for i in range(len(gaussian_basis)):
 if (k & (1 << i)) != 0:
 result ^= gaussian_basis[i]

return result

测试用例
def test_maximum_xor():
 """测试最大异或和"""
 print("== 测试最大异或和 ==")
 arr1 = [3, 10, 5, 25, 2, 8]
 result1 = find_maximum_xor(arr1)
 print(f"输入: {arr1}")
 print(f"期望输出: 28 (5^25)")
 print(f"实际输出: {result1}")
 print(f"测试结果: {'通过' if result1 == 28 else '失败'}")

```

```
print()

def test_linear_dependent():
 """测试线性相关情况"""
 print("==> 测试线性相关情况 ==>")
 arr2 = [1, 2, 3] # 1^2 = 3, 线性相关
 result2 = find_maximum_xor(arr2)
 print(f"输入: {arr2}")
 print(f"期望输出: 3")
 print(f"实际输出: {result2}")
 print(f"测试结果: {'通过' if result2 == 3 else '失败'}")
 print()

def test_empty_array():
 """测试空数组"""
 print("==> 测试空数组 ==>")
 arr3 = []
 result3 = find_maximum_xor(arr3)
 print(f"输入: {arr3}")
 print(f"期望输出: 0")
 print(f"实际输出: {result3}")
 print(f"测试结果: {'通过' if result3 == 0 else '失败'}")
 print()

def test_single_element():
 """测试单元素数组"""
 print("==> 测试单元素数组 ==>")
 arr4 = [5]
 result4 = find_maximum_xor(arr4)
 print(f"输入: {arr4}")
 print(f"期望输出: 5")
 print(f"实际输出: {result4}")
 print(f"测试结果: {'通过' if result4 == 5 else '失败'}")
 print()

def test_kth_xor():
 """测试第 k 小异或和"""
 print("==> 测试第 k 小异或和 ==>")
 arr5 = [1, 2, 3]
 print(f"输入: {arr5}")
 for k in range(1, 5):
 result = query_kth_xor(arr5, k)
 print(f"第 {k} 小异或和: {result}")
```

```
print()

def main():
 """主函数"""
 print("线性基算法测试 (Python 版本)")
 print("=" * 30)

 # 运行所有测试用例
 test_maximum_xor()
 test_linear_dependent()
 test_empty_array()
 test_single_element()
 test_kth_xor()

 print("所有测试完成!")

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

```
=====
```