Lesson4

容斥原理

例子

- 韦恩图的面积计算S = S1 + S2 + S3 - (S1∩S2) - (S2∩S3) - (S1∩S3) + (S1∩S2∩S3)

一般形式：n个圆组合，则其面积为

- **S = 1(所有一个圆的组合)-2(所有两个圆的组合)+3-4+5-...+(-1)^(n-1)*n(所有n个圆的组合)**

从集合角度考虑，则

- I S1US2US3 I = IS1I+IS2I+IS3I - IS1∩S2I - IS2∩S3I - IS1∩S3I + I S1∩S2∩S3 I
- 推广形式...
- 组合数性质：C(n, 0)+C(n,1)+...C(n, n) = 2^n ，所以I S1 U S2 U S3... U Sn I 用容斥原理展开有(2^n)-1项（除去I 0 I的情况），即时间复杂度为**O(2^n)**

证明：

I S1US2US3...USn I 中，对于数x，假设其出现k(0<=k<=n) 次。则对于容斥原理右侧等式，x会被计算

C(k, 1) - C(k, 2) + C(k, 3) -....+(-1)^(k-1)C(k, k)，而该式=1（组合恒等式）。即x在右侧统计时，只会被统计一次，所以容斥原理正确

实例：能被整除的数（应用容斥原理降低时间复杂度）

思路：

- 能被2整除的数S2 = {2, 4, 6, 8, 10}
- S3 = {3, 6, 9}

  I S2US3 I = IS1I + IS2I - IS2∩S3I = 5 + 3 -1 = 7
- 1~n中， 1 ~n 中是 p1* p2 * ... * pn的倍数个数为 n / (p1 * p2 * ... * pn)

  所以1~n中**I Sp1∩Sp2∩...∩Spn I = n / (p1 * p2 * ... * pn)**
- 因此本题时间复杂度O(2^m * m)，即O(2^16*2^4) 运算次数100w，满足时间要求
- 本题公式I Sp1USp2U...Spm I (能至少被p1、p2、...、pm中一个数整除的个数)，即可利用**容斥原理**进行计算

具体实现：使用**位运算枚举所有方式**进行优化

```java
static int[] p = new int[20];

public static void main(String[] args) throws Exception {
    ins.nextToken(); int n = (int)ins.nval;
    ins.nextToken(); int m = (int)ins.nval;

    for (int i=0; i<m; i++) { ins.nextToken(); p[i] = (int)ins.nval; }

    long res = 0;
```

```java
    for (int i=1; i<1<<m; i++) {
        long t = 1, s = 0;

        for (int j=0; j<m; j++) {
            if ((i>>j & 1) == 1) {
                s++;

                if (t*p[j] > n) {
                    t = -1;
                    break;
                }

                t *= p[j];
            }
        }

        if (t != -1) {
            if (s % 2 == 0) res -= n/t;
            else res += n/t;
        }
    }

    out.println(res);

    out.flush();
}
```

博弈论

公平游戏组合ICG，若一个游戏满足

- 两名玩家交替行动
- 游戏任意时刻，玩家可以进行的合法行动和轮到谁无关
- 不能行动的玩家判负

  NIM博弈属于公平组合游戏。围棋，五子棋不是公平组合游戏，围棋交战双方只能落黑子与白子，胜负判定也比较复杂，不满足第二点与第三点

有向图游戏

- 给定一个有向无环图，图中有一个唯一的起点，在起点上放有一枚棋子，两名玩家交替地把这枚棋子沿有向边进行移动，每次可以移动一步，无法移动者判负。任何一个公平组合游戏都可以转化为有向图游戏。具体方法是，把每个局面看成图中的一个结点，并且把这个局面沿着合法行动能够到达的下一个局面连有向边

NIM游戏

性质：

- 先手必胜状态：可以走到一个必败状态

  先手必败状态：不管怎么操作，剩下的状态都是必胜状态，例如(0, 0)，等价于走不到任何一个必败状态

- 结论：**若有a1, a2, .., an堆石子，若a1^a2^...^an = 0，则先手必败；若a1^a2^...^an != 0，则先手必胜。**

  证明：

  首先，0^0^...^0 = 0，即终点(0, 0, ..., 0)是必败状态

  若a1^a2^..^an = x != 0，则一定可以通过某种方式，让其异或值变为0。

  方式从某一堆里拿走若干石子，让剩下的石堆异或值为0

  假设**x**的二进制表示中，最高一位1在第k位，则a1~an中必然存在一个数ai，且ai二进制的第k位是1，显然**ai ^ x < ai**，进而可以从ai石堆拿走**(ai - (ai^x))**个石子，所以**ai - > ai- (ai - (ai^x)) = ai ^ x**，所以剩下**a1 ^ a2 ^ ... ^ (ai ^ x) ^ ai+1 ^ ... ^ an = x ^ x = 0**

  若a1^a2^...^an = 0，不管怎么拿，剩下所有数异或值一定不是0，用反证法进行证明

  假设从ai'中拿掉某些石子（不能不拿），剩下异或值仍为0，即a1^a2^...ai'^a(i+1)^...^an = 0;

  将其与a1^a2^...^an = 0进行异或，则有(ai^ai' = 0)，即ai = ai'，与假设矛盾，所以原结论成立。

- 所以若a1^a2^..^an = x != 0，两方在采取最优策略时，先手手里一定不是0，后手手里一定是0，且游戏一定会结束，故最终一定是先手必胜。若a1^a2^..^an = x = 0，则一定先手必败。
- 扩展：K-NIM游戏（每次最多拿K个）

NIM游戏具体实现

```java
public static void main(String[] args) throws Exception {
    ins.nextToken(); int n = (int)ins.nval;

    int res = 0;
    while (n-- > 0) {
        ins.nextToken(); int a = (int)ins.nval;
        res ^= a;
    }

    if (res != 0) out.println("Yes");
    else out.println("No");

    out.flush();
}
```

台阶-NIM游戏

性质

- 若所有奇数级台阶上的石子异或值为0，即a1^a3^...^an(n为奇数) = x !=0，则先手必胜；反之，若a1^a3^...^an(n为奇数) = x =0，则先手必败。

  证明：(参见NIM游戏的证明)　当先手a1^a3^...^an(n为奇数) = x !=0时的情况

  若x!=0，则一定可以拿走某一堆石子里的若干个使异或值为0

若x=0，若对手拿的偶数级i级台阶石子放到i-1级上，我们则从i-1级上拿同样多石子放到i-2级上，从而使奇数级台阶上石子个数不变，所以x仍等于0；若对手拿的奇数级j级台阶上的石子，则操作之后，a1^a3^...^an(n为奇数) = x !=0，回到上述第一种局面。

终止局面没有石子异或值为0，且该局面一定会被对手遇到，所以对手必败。

```java
public static void main(String[] args) throws Exception {
    ins.nextToken(); int n = (int)ins.nval;

    int res = 0;
    for (int i=1; i<=n; i++) {
        ins.nextToken(); int a = (int)ins.nval;
        if ((i & 1) == 1) res ^= a;      //二进制优化
    }

    if (res != 0) out.println("Yes");
    else out.println("No");

    out.flush();
}
```

集合-NIM游戏

SG函数

- **MEX运算**：设S表示一个非负整数集合，则mex(S)为求出**不属于**集合S的**最小非负整数**

  即mex(S) = min(x)，x属于自然数，且**不属于**S。mex({1, 2, 3}) = 0（自然数从0开始），mex({0, 2}) = 1

- **SG函数**：在有向图游戏中，对于每个结点x，设从x出发共有k条有向边，分别到达结点y1, y2, .., yk，定义SG(x)为x的后继节点y1, y2, .., yk的SG函数值构成的集合S再执行mex(S)运算的结果，即：

  SG(x) = mex({SG(y1), SG(y2), ..., SG(yk)})，且定义SG(终点) = 0

  特别的，整个有向图游戏G的 SG函数值被定义为有向图游戏起点s的SG函数值，即SG(G) = SG(s);

性质：若只有一个有向图

- 若SG(x) = 0，则为必败状态
- 若SG(x) ≠ 0，则为必胜状态
- **当有多个有向图时，若SG(总)=SG(x1)^SG(x2)^...^SG(xn) = 0，则为必败状态，反之为必胜状态**

  使用SG函数可以有效降低运算复杂度。

  证明：（参见NIM游戏结论的证明）

  首先，所有局面都不能走，SG(xi) = 0，则SG(x1)^SG(x2)^...^SG(xn) = 0，为必败状态

  若SG(x1)^SG(x2)^...^SG(xn) != 0，则一定可以找到某个局面SG(xi)^x<SG(xi)，并把

  SG(xi) -> SG(xi)^x，即可把SG(x1)^SG(x2)^...^SG(xn) 变为0

  若SG(x1)^SG(x2)^...^SG(xn) = 0，不管怎么拿，剩下所有SG(xi)异或值一定不为0

思路：

- n堆石子，视为有n个有向图，由SG定理，对每个有向图起点的SG值进行异或，若结果为0，则为必败状态，若结果不为0，则为必胜状态。

具体实现：**求SG函数一般使用记忆化搜索以降低时间复杂度**

```java
static int N = 110, M = 10010;

static int n, m;
static int[] s = new int[N], f = new int[M];    //s存储S数组，f记忆化搜索

static int sg(int x) {
    if (f[x] != -1) return f[x];

    Set<Integer> S = new HashSet<Integer>();

    for (int i=0; i<m; i++) {
        int ss = s[i];
        if (x >= ss) S.add(sg(x-ss));    //dfs求子节点SG集合
    }

    //求SG(x)
    for (int i=0; ; i++) {
        if (!S.contains(i))
            return f[x] = i;
    }
}

public static void main(String[] args) throws Exception {
    Arrays.fill(f, -1);

    ins.nextToken(); m = (int)ins.nval;
    for (int i=0; i<m; i++) { ins.nextToken(); s[i] = (int)ins.nval; }

    ins.nextToken(); n = (int)ins.nval;
    int res = 0;
    while (n-- > 0) {
        ins.nextToken(); int a = (int)ins.nval;
        res ^= sg(a);
    }

    if (res != 0) out.println("Yes");
    else out.println("No");

    out.flush();
}
```

拆分-NIM游戏

性质

- 一定可以结束，最大值不断减小，最终最大值一定会变为0，即一定可以结束

- 把每堆石子视为一个独立的局面，分别求出SG(a1), SG(a2), ..., SG(an)，并进行异或，即可判断输赢

  如何求出每一堆的SG(ai)

  a->(b1, b2), (c1, c2), ...

  性质：**SG(b1, b2) = SG(b1) ^ SG(b2)**，并以此进行**记忆化搜索**

```java
static int N = 110;

static int[] f = new int[N];    //记忆化搜索

static int sg(int x) {
    if (f[x] != -1) return f[x];

    Set<Integer> S = new HashSet<Integer>();
    for (int i=0; i<x; i++)
        for (int j=0; j<=i; j++)
            S.add(sg(i)^sg(j));

    //求SG
    for (int i=0; ; i++)
        if (!S.contains(i))
            return f[x] = i;
}

public static void main(String[] args) throws Exception {
    Arrays.fill(f, -1);

    ins.nextToken(); int n = (int)ins.nval;

    int res = 0;
    while (n-- > 0) {
        ins.nextToken(); int a = (int)ins.nval;
        res ^= sg(a);
    }

    if (res != 0) out.println("Yes");
    else out.println("No");

    out.flush();
}
```