信息学

线性基的应用

引入：子集异或和最大问题

给定n个整数（数据可重复），求在这些数中选择任意个，使得他们的异或和最大。

怎么搞？

Xor本质上是二进制位进行操作，所以直接从二进制位来考虑

贪心的角度来看，在所有数中找到一个二进制位最多且最高位为1的数，
例如$(1000000010)_2$ 他的最高位肯定对最终的答案有贡献。

那么次高位呢？

继续找到一个满足次高位的数呗。以此类推。其正确性是显然的。

引入：子集异或和最大问题

那跟线性基又有什么关系呢？

对于一组给定的数，我们可以求出一个数组p，记 p[i]=k 表示第i位二进制位为1且是最高位的数值为k

意这个k可以是原来给定的数，也可以是原来给定的数之间多次xor的产物。

可以通过这个数组p之间的异或生成原有给定的数的子集的异或值

本来有N个数的xor过程我们转化为了log_2(N) 个数的xor过程。可以极大缩小操作所需的查询次数。

线性基是一种特殊的数据结构，在解决XOR问题时非常好用

给定数列值域为[1…N] 那么我们可以用一个长度在log_2(N)的数组记录一组线性基。
从而降低操作的对象数量。

生成线性基

1.新拿到一个数 x
2.从高位数着看看第i位有没有1
3.如果有1，看看p[i]是不是被占了
4.如果p[i]是空的，执行p[i]=x先到先得, return 走人
5.如果p[i]不空，x^p[i]干掉x的最高位，然后继续循环。

注意下这行代码 `if(x&(1ll<<i))`

生成线性基p的过程，本质上是需要记录之前未出现过的二进制位，如果x中有未出现过的二进制位
即使第一位是出现过的，也要去标记

```
// register int i 建议编辑器将i变量放在CPU寄存器中
// 以获得更快的运行速度（优化并不显著）
typedef long long ll;
const int SIZE=62; //从最高进制位开始找
ll p[100],d[100];

void ins(ll x){
    for(reg int i=SIZE;~i;i--)
        if(x&(1ll<<i))
            if(!p[i]){p[i]=x;return;}
            else x^=p[i];
}
int main(){
    int n;ll x;
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        scanf("%lld",&x);
        ins(x); //将x放入线性基中
    }

    return 0;
}
```

生成线性基

一些性质：

插入x的过程中，要么x存在 当前线性基中并不存在的二进制位 然后被插入到p中，要么被^0成，即，原序列中每一个数，都能被线性基xor后所表示。

线性基中任意元素不可能通过xor得到0，因为如果有x^y^z = 0 那么x^y = z 根据ins函数的规则，z是不会被插入的。

因为先到先得的原因，同样的数列按不同输入的顺序输入后，线性基里的数可能不同，但是总数一定是相同的。

解决子集最大异或和问题


```
//基于之前的贪心思想+线性基生成思路，我们可以发现：
ll askmax() {
  ll ans=0;
  for(reg int i=SIZE;i>=0;i--)
    if((ans^p[i])>ans) ans^=p[i];
  return ans;
}
```

判断x是否能被生成

西南大学附属中学
High School Affiliated to Southwest University

```
bool check(ll x){
    for(reg int i=SIZE;~i;i--)
        if(x&(1ll<<i))
            if(!p[i])return false;
            else x^=p[i];
    return true;
} //判断是生成ins的逆过程，所以代码显然。
```

子集异或值最小问题

p[i]中的最小值就是子集xor最小

不是满秩 应该为0

```
ll askmin() {
  if(flag0) return 0;
  for(reg int i=0;i<=62;i++)
    if(p[i]) return p[i];
}
```

ins的时候记得打个flag　　线性基个数小于原序列个数

求第k小

假设有k位二进制位，那么其最大值为2^k

线性基中的每一个数像二进制位那样互不影响，求第k小则直接从k的二进制位入手选就行了。

构造底基 只负责第i位置的1

```
void rebuild(){
    cnt =0;top=0;
    for(reg int i=SIZE;i>=0;i--)
        for(reg int j=i-1;j>=0;j--)
            if(p[i]&(1ll<<j))
                p[i]^=p[j];
    for(reg int i=0;i<=SIZE;i++) if(p[i]) d[cnt++]=p[i];
}
```

不一定满秩，但不影响第k小

```
//调用kth函数前请先rebuild出d数组
ll kth(int k){
    if(k>=(1ll<<cnt))return -1; //超过了数域范围
    ll ans = 0;
    for(reg int i=SIZE;i>=0;i--){
        if(k&(1ll<<i)) ans^=d[i];
    }
    return ans;
}
```