# 7.2 线性基

**7.2.1.概述：**

线性基，是线性代数中的概念，在信息学竞赛中，前缀线性基是线性基的扩展，他们主要用于处理有关异或和的极值问题。

一组线性无关的向量即可作为一组基底，张起一个线性的向量空间，这个基底即称为线性基，利用线性基的基底进行线性运算，可表示向量空间内的所有向量，换句话说，所有向量都可以拆成基底的线性组合。

根据异或的原理，将一个数字拆成他的二进制形式，将二进制形式用向量来表示，由于一组线性无关的向量可以张起一个向量空间，因此可以考虑构造这样一组数字的二进制形式组成的线性基，在这个线性基中，通过基底的线性组合、异或运算，从而可以表达所有的异或结果。

简单来说，若一个数集 T 的值域范围为 ，那么 T 的线性基是 T 的一个子集 A={a1,a2,a3,...,an}，A 中元素相互异或而成的集合，等价于原数集 T 的元素相互异或形成异或集合

**7.2.2.预备知识**

交换变量位置：

a = a ^ b; b = a ^ b; a = a ^ b;

如果a^b==c 则a^c==b b^c==a a^b^c==0

**7.2.3.定义**

线性基是一个数的集合，并且每个序列都拥有至少一个线性基，取线性基中若干个数异或起来可以得到原序列中的任何一个数。

**7.2.4.二进制求线性基**

1. **void** init()//求非最简线性基
2. {
3. **for**(**int** i=1;i<=n;i++)//1-n遍历数组元素
4. {
5. ll x=num[i];
6. **for**(**int** j=63;j>=0;j--)//考察元素的每一位
7. {
8. **if**(x&(1LL<<j))//如果元素的第j位为1
9. {
10. **if**(p[j]) x^=p[j];//在线性基中已经出现第j位为1时，消去当前元素//的1，保证为下三角形式,化为下三角是为了更容易异//或得出某数
11. **else**
12. {
13. p[j]=x;//否则将第j位元素置1
14. **break**;
15. }
16. }
17. }
18. }
19. **return** ;
20. }

**7.2.5.判断某数能否由线性基表示**

**思路：将该数通过线性基，看结果是否为0即可**

**代码：**

1. **bool** check(**long** **long** x)
2. {
3. **for**(**int** j=60;j>=0;j--)
4. {
5. **if**(x&(1LL<<j))
6. {
7. **if**(d[j]==0)
8. {
9. **return** **false**;
10. }
11. **else** x^=d[j];
12. }
13. }
14. **return** x==0;
15. }

**7.2.6.求异或和最大**

**思路：求出线性基，**由于高位为1贡献更大，每次优先异或高位，遍历线性基即可

**代码：**

1. **long** **long** query\_max()
2. {
3. **long** **long** ret=0;
4. **for** (**int** i=60;i>=0;i--)
5. **if** ((ret^d[i])>ret)//贪心
6. ret^=d[i];
7. **return** ret;
8. }

**7.2.7.求异或和最小**

**思路：**求最小的线性基元素即可

**代码：**

1. **long** **long** query\_min()
2. {
3. **for** (**int** i=0;i<=60;i++)
4. **if** (d[i])  **return** d[i];
5. **return** 0;
6. }

**7.2.8.求异或和第K大**

**思路：**将线性基化为最简，K用二进制表示，选出1的位，将以其为下标的线性基元素相乘即可

**代码：**

1. **void** rebuild()//高斯消元化为最简线性基
2. {
3. **for** (**int** i=60;i>=0;i--)
4. **for** (**int** j=i-1;j>=0;j--)
5. **if** (d[i]&(1LL<<j))
6. d[i]^=d[j];
7. **for** (**int** i=0;i<=60;i++)
8. **if** (d[i])
9. p[cnt++]=d[i];
10. }
11. **long** **long** kthquery(**long** **long** k)
12. {
13. **int** ret=0;
14. **if** (k>=(1LL<<cnt))
15. **return** -1;
16. **for** (**int** i=60;i>=0;i--)
17. **if** (k&(1LL<<i))
18. ret^=p[i];
19. **return** ret;
20. }

**7.2.9.经典例题：**

**7.2.9.1．HDU 3949 XOR**

**Problem Description**

XOR is a kind of bit operator, we define that as follow: for two binary base number A and B, let C=A XOR B, then for each bit of C, we can get its value by check the digit of corresponding position in A and B. And for each digit, 1 XOR 1 = 0, 1 XOR 0 = 1, 0 XOR 1 = 1, 0 XOR 0 = 0. And we simply write this operator as ^, like 3 ^ 1 = 2,4 ^ 3 = 7. XOR is an amazing operator and this is a question about XOR. We can choose several numbers and do XOR operatorion to them one by one, then we get another number. For example, if we choose 2,3 and 4, we can get 2^3^4=5. Now, you are given N numbers, and you can choose some of them(even a single number) to do XOR on them, and you can get many different numbers. Now I want you tell me which number is the K-th smallest number among them.

**Input**

First line of the input is a single integer T(T<=30), indicates there are T test cases.  
For each test case, the first line is an integer N(1<=N<=10000), the number of numbers below. The second line contains N integers (each number is between 1 and 10^18). The third line is a number Q(1<=Q<=10000), the number of queries. The fourth line contains Q numbers(each number is between 1 and 10^18) K1,K2,......KQ.

**Output**

For each test case,first output Case #C: in a single line,C means the number of the test case which is from 1 to T. Then for each query, you should output a single line contains the Ki-th smallest number in them, if there are less than Ki different numbers, output -1.

**Sample input**

2

2

1 2

4

1 2 3 4

3

1 2 3

5

1 2 3 4 5

**Sanple output**

Case #1:

1

2

3

-1

Case #2:

0

1

2

3

-1

**题意：给出n个数，求他们任意组合的最小值**

**思路：求出这些数的线性基，将k二进制拆分，第i位为1，答案就异或第i个线性基，若线性基个数小于原个数，即最小值可为0，则用类似方法求k-1小即可**

**代码：**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **typedef** **long** **long** ll;
4. ll p[65];//存储线性基
5. ll np[65];//存储最简线性基
6. ll num[10005];//存储原数组
7. **int** n;
8. **int** k;
9. **void** init()//求非最简线性基
10. {
11. **for**(**int** i=1;i<=n;i++)//1-n遍历数组元素
12. {
13. ll x=num[i];
14. **for**(**int** j=63;j>=0;j--)//考察元素的每一位
15. {
16. **if**(x&(1LL<<j))//如果元素的第j位为1
17. {
18. **if**(p[j]) x^=p[j];//在线性基中已经出现第j位为1时，消去当前元素的1，保证为下三角形式
19. //化为下三角是为了更容易异或得出某数
20. **else**
21. {
22. p[j]=x;//否则将第j位元素置1
23. **break**;
24. }
25. }
26. }
27. }
28. **return** ;
29. }
30. **int** ninit()//求最简线性基   化为对角矩阵
31. {
32. **int** cnt=0;
33. **for**(**int** i=0;i<=63;i++)//从右下角开始
34. {
35. **for**(**int** j=i-1;j>=0;j--)//从左往右遍历
36. {
37. **if**(p[i]&(1LL<<j)) p[i]^=p[j];//如果p[i]在除第i位为1外有第j位为1，高斯消元消去第j位的1
38. //由于p[j]的第j位必为1，因此与p[J]异或
39. }
40. }
41. **for**(**int** i=0;i<=63;i++)
42. **if**(p[i])
43. {
44. np[cnt++]=p[i];//将最简的线性基存入新数组中
45. }
46. **return** cnt;
47. }
48. ll kth(ll k,**int** z)//查询异或和第k小的函数
49. {
50. ll ans=0;
51. **for**(**int** i=0;i<=62;i++)//将k用二进制表示，逐一搜索它的每一位，若第j位为1，则将答案异p[j]
52. {
53. **if**(k&(1LL<<i)) ans^=np[i];
54. }
55. **return** ans;
56. }
57. ll q[10005];
58. **int** main()
59. {
60. **int** t;
61. cin>>t;
62. **int** command;
63. **int** now=0;
64. **while**(t--)
65. {
66. now++;
67. memset(p,0,**sizeof**(p));
68. memset(num,0,**sizeof**(num));
69. memset(np,0,**sizeof**(np));
70. cin>>n;
71. **for**(**int** i=1;i<=n;i++) cin>>num[i];
72. cin>>command;
73. init();
74. k=ninit();
75. **for**(**int** i=1;i<=command;i++) cin>>q[i];
76. cout<<"Case #"<<now<<":"<<endl;
77. **for**(**int** i=1;i<=command;i++)
78. {
79. **if**(k<n&&q[i]==1)
80. {
81. cout<<"0"<<endl;
82. **continue**;
83. }
84. **if**(k<n) q[i]--;//有重复的，异或和最小为0，由于线性基无法组合出0，因此先减
85. **if**(q[i]>=(1LL<<k))//如果查询的位次比所有的组合都要多，则输出-1
86. {
87. cout<<"-1"<<endl;
88. **continue**;
89. }
90. cout<<kth(q[i],k)<<endl;
91. }
92. }
93. **return** 0;
94. }

**7.2.9.2．洛谷P3812 线性基**

**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。

**输入格式**

第一行一个数n，表示元素个数

接下来一行n个数

**输出格式**

仅一行，表示答案。

**输入样例**

2

1 1

**输出样例**

1

**思路：遍历线性基，每次异或高位为1的线性基元素**

**代码：**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **typedef** **long** **long** ll;
4. ll mp[65];
5. ll p[65];
6. ll np[65];
7. **int** n;
8. **void** init()//求非最简线性基（下三角）
9. {
10. memset(p,0,**sizeof**(p));
11. memset(np,0,**sizeof**(np));
12. **for**(**int** i=1;i<=n;i++)//遍历数组
13. {
14. ll x=mp[i];
15. **for**(**int** j=62;j>=0;j--)//遍历元素每一位
16. {
17. **if**(x&(1LL<<j))//若元素第j位为1
18. {
19. **if**(!p[j])//并且线性基中无对应的值
20. {
21. p[j]=x;//加入元素
22. **break**;
23. }
24. **else** x^=p[j];//否则消去第j位的1
25. }
26. }
27. }
28. }
29. **int** main()
30. {
31. cin>>n;
32. **for**(**int** i=1;i<=n;i++) cin>>mp[i];
33. init();
34. ll ans=0;
35. **for**(**int** i=62;i>=0;i--)//求最大值贪心
36. {
37. **if**((ans^p[i])>ans) ans^=p[i];
38. }
39. cout<<ans<<endl;
40. **return** 0;
41. }

**7.2.9.3．洛谷P4570[BJWC2011]元素**

**题目描述**

相传，在远古时期，位于西方大陆的 Magic Land 上，人们已经掌握了用魔法矿石炼制法杖的技术。那时人们就认识到，一个法杖的法力取决于使用的矿石。

一般地，矿石越多则法力越强，但物极必反：有时，人们为了获取更强的法力而使用了很多矿石，却在炼制过程中发现魔法矿石全部消失了，从而无法炼制出法杖，这个现象被称为“魔法抵消”。特别地，如果在炼制过程中使用超过一块同一种矿石，那么一定会发生“魔法抵消”。后来，随着人们认知水平的提高，这个现象得到了很好的解释。经过了大量的实验后，著名法师 Dmitri 发现：如果给现在发现的每一种矿石进行合理的编号（编号为正整数，称为该矿石的元素序号），那么，一个矿石组合会产生“魔法抵消”当且仅当存在一个非空子集，那些矿石的元素序号按位异或起来为零（如果你不清楚什么是异或，请参见下一页的名词解释 ）。

例如，使用两个同样的矿石必将发生“魔法抵消”，因为这两种矿石的元素序号相同，异或起来为零。并且人们有了测定魔力的有效途径，已经知道了：合成出来的法杖的魔力等于每一种矿石的法力之和。人们已经测定了现今发现的所有矿石的法力值，并且通过实验推算出每一种矿石的元素序号。

现在，给定你以上的矿石信息，请你来计算一下当时可以炼制出的法杖最多有多大的魔力。

**输入格式**

第一行包含一个正整数 N*N*，表示矿石的种类数。

接下来 N行，每行两个正整数Numberi​ 和 Magici，表示这种矿石的元素序号和魔力值。

**输出格式**

仅包含一行，一个整数代表最大的魔力值。

**输入样例**

3

1 10

2 20

3 30

**输出样例**

50

**思路：**由于线性基不唯一，本题要求最大魔力值，可优先选魔力值大的值生成线性基，贪心即可

**题解：**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **typedef** **long** **long** ll;
4. **struct** node
5. {
6. ll val,magic;
7. **friend** **bool** operator < (**const** node x,**const** node y)
8. {
9. **return** x.magic>y.magic;
10. }
11. }mp[1005];
12. ll p[65];
13. **int** init(**int** n)
14. {
15. ll ans=0;
16. **for**(**int** i=1;i<=n;i++)
17. {
18. ll x=mp[i].val;
19. **for**(**int** j=62;j>=0;j--)
20. {
21. **if**(x&(1LL<<j))
22. {
23. **if**(!p[j])
24. {
25. p[j]=x;
26. ans+=mp[i].magic;
27. **break**;
28. }
29. **else** x^=p[j];
30. }
31. }
32. }
33. **return** ans;
34. }
35. **int** main()
36. {
37. **int** n;
38. cin>>n;
39. **for**(**int** i=1;i<=n;i++)
40. {
41. cin>>mp[i].val>>mp[i].magic;
42. }
43. sort(mp+1,mp+n+1);
44. ll ans=init(n);
45. cout<<ans<<endl;
46. **return** 0;
47. }

**7.2.9.4 洛谷P3265 装备购买**

**题目描述**

有n件装备，每件装备有m个属性，用向量（a1，a2，a3……am）表示，需要花费c购买，现在规定如果一件装备能由其他装备组合出，那么就不需要购买（即可由其他向量线性表示），求在买下最多装备的情况下花费最少的钱

**输入格式**

第一行两个数n，m。接下来n行，每行m个数，其中第i行描述装备i的各项属性值。接下来一行n个数，其中ci表示购买第i件装备的花费。

**输出格式**

一行两个数，第一个数表示能够购买的最多装备数量，第二个数表示在购买最多数量的装备的情况下的最小花费

**输入样例**

3 3

1 2 3

3 4 5

2 3 4

1 1 2

**输出样例**

2 2

**思路：**实数线性基+贪心

题解：

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **typedef** **long** **long** ll;
4. **const** **double** eps=1e-6;
5. **int** n,m;
6. **int** p[505];
7. **struct** node\_
8. {
9. **double** mp[505];
10. **int** c;
11. **friend** **bool** operator < (**const** node\_ x,**const** node\_ y)
12. {
13. **return** x.c<y.c;
14. }
15. }node[505];
16. **int** cnt=0;
17. **int** ans=0;
18. **void** init()
19. {
20. **for**(**int** i=1;i<=n;i++)
21. {
22. **for**(**int** j=1;j<=m;j++)
23. {
24. **if**(fabs(node[i].mp[j])>eps)//卡精度
25. {
26. **if**(!p[j])
27. {
28. cnt++;
29. ans+=node[i].c;
30. p[j]=i;
31. **break**;
32. }
33. **else**
34. {
35. **long** **double** times=node[i].mp[j]\*1.0/node[p[j]].mp[j];
36. **for**(**int** k=j;k<=m;k++)//初等行变换，化为下三角
37. {
38. node[i].mp[k]-=times\*node[p[j]].mp[k];
39. }
40. }
41. }
42. }
43. }
44. }
45. **int** main()
46. {
47. cin>>n>>m;
48. **for**(**int** i=1;i<=n;i++)
49. **for**(**int** j=1;j<=m;j++)
50. cin>>node[i].mp[j];
51. **for**(**int** i=1;i<=n;i++)
52. cin>>node[i].c;
53. sort(node+1,node+n+1);
54. memset(p,0,**sizeof**(p));
55. init();
56. cout<<cnt<<" "<<ans<<endl;
57. **return** 0;
58. }