分组加密算法语言

【题目背景】

在密码学中，分组加密(block cipher)，又称分块加密或块密码，是一种对称密钥算法。它将明文分成多个等长的模块(block)，使用确定的算法和对称密钥对每组分别加密解密。分组加密是极其重要的加密协议组成，其中典型的如 DES 和 AES 作为美国政府核定的标准加密算法，应用领域从电子邮件加密到银行交易转帐，非常广泛。(摘自维基百科)

【格式描述】

在学习了密码学之后，顿顿发现分组加密算法大都是由一些基本的操作组成，比如置换、S 盒等等。再加上一些简单的逻辑控制语句，就可以清晰地描述出该算法的内部结构。基于此，顿顿设计了一种简单的语言，用来描述分组加密算法的逻辑结构。

该语言的结构大致如下:

```
变量声明部分
BEGIN
加密算法部分
END
```

一、变量声明部分、该语言仅有两种变量，分别为**二进制串变量**和**循环控制变量**。在此处声明的变量均为**二进制串变量**，格式为:变量名(长度)，例如:

```
state(64)
key(10)
tmp(5)
beta(20)
```

在上面的例子中，我们声明了四个**二进制串变量**: state、key、tmp 和 beta，长度分别为 64、10、5 和 20。这里我们约定**长度**为 [1, 64] 范围内的整数，二进制串的 **变量名**由 2 到 10 个小写英文字母组成，且要求**变量名**互不相同。

在变量声明部分，每行声明一个**二进制串变量**，其中第一行和第二行固定为 state 和 key，第三行开始可以声明其它**二进制串变量**。state 在加密开始时存储明文，加密结束时存储密文。key 在加密开始时存储密钥。其余二进制串变量在加密开始时默认每一位均为 0。

循环控制变量为整数类型，变量名仅为一个小写英文字母，因此总共只有 26 个**循环控制变量**，不需要声明即可在加密算法部分直接使用。

二、加密算法部分

在这一部分顿顿设计了三种句法，分别为**赋值**、**循环**和**分组**，下面逐一进行介绍。

1. 赋值(异或、置换、S 盒)

```
二进制串变量 = 表达式
```

赋值语句将**表达式**的运算结果(二进制串)存入相应的**二进制串变量**，且运算结果需要与变量长度相同。简单地说，**表达式**就是若干个二进制串进行**异或**、**置换**和 **S 盒**三种运算产生的算式，运算结果仍为一个二进制串。其严格定义如下所示:

```
表达式 ::= 二进制串变量 | 二进制串常量
表达式 ::= 表达式 + 表达式          //异或运算
```

表达式 ::= 置换表 (表达式)　　//使用某个置换表对表达式的结果做置换运算
表达式 ::= S 盒 (表达式)　　//使用某个 S 盒对表达式的结果做 S 盒运算
置换表 ::= P[常数] | P[循环控制变量]　//常数、控制变量用来指明使用哪一个置换表
S 盒 ::= S[常数] | S[循环控制变量]　　//常数、控制变量用来指明使用哪一个 S 盒

在上面的定义中，::= 表示“定义为”，| 表示“或”的意思。下面则是一些表达式的示例：

state
"10100010"
state + "10100010"
P[0](state)
S[2](state + "10100010")
key + P[k](state) + S[1](state + "10100010")

这里顿顿使用双引号括起的 01 串来表示一个**二进制串常量**，并用 + 表示**异或**运算， 即两个等长的二进制串逐位异或。下面详细介绍一下**置换**和 **S 盒**运算。

置换运算简单来说就是用输入二进制串中的若干位，拼出一个新的二进制串，其中输入二进制串的每一位既可多次使用也可以不用。如果输入二进制串长度为 a，输出二进制串长度为 b，那么对应的置换表就是一个长度为 b 的数组，其中每一个元素取值范围$[0, a)$，表示输出二进制串中每一位的来源。假设第 i 个元素取值为 j，其含义为输出二进制串的第 i 位取自输入二进制串的第 j 位。输入、输出二进制串的长度同样满足 $1 \le a, b \le 64$。

输入二进制串：abcde
置换表：{0, 1, 2, 3, 4, 3, 2, 1}
输出二进制串：abcdedcb

输入二进制串：abcde
置换表：{1, 1, 1}
输出二进制串：bbb

S 盒实际上就是一个查找表，对于每一个输入给出一个相应的输出。如果输入二进制串长度为 c，输出二进制串长度为 d，那么对应的 S 盒就是一个长度为 2^c 的数组，每一个元素是一个长度为 d 的二进制串对应的十进制整数。这里二进制串转化成十进制数时默认左边为高位、右边为低位，例如 "1000" 对应十进制整数 8。假设第 i 个元素取值为 j，其含义为当 i 对应的二进制串做为输入时，输出为 j 对应的二进制串。因为 S 盒长度较大，这里约定输入、输出二进制串的长度满足 $1 \le c, d \le 8$。

输入二进制串长度：3
输出二进制串长度：2
S 盒：{0, 1, 2, 3, 3, 2, 1, 0}

输入		输出	
0	000	0	00
1	001	1	01
2	010	2	10
3	011	3	11
4	100	3	11

5 101 2 10
6 110 1 01
7 111 0 00

一个加密算法可能会使用多个置换表和 S 盒，因此需要用 P、S 后面方括号里的**常数**或**循环控制变量**来指明具体使用哪一个。

2. 循环（LOOP）

LOOP 循环控制变量 初值 终值
…
ENDLOOP

如上所示，循环控制变量从小到大取遍 [初值, 终值] 范围内的所有整数值。每取一个值，便按顺序执行一遍 LOOP 到相应 ENDLOOP 间的所有语句。循环中可以继续嵌套循环，但外层使用的**循环控制变量**内层不得重复使用。

由于循环结构的存在，一条语句可能会被执行很多次，这里我们约定每条语句执行的次数总和不会超 10000。这个约定只是为了方便估计程序的运行时间，下面的几个例子中给出了相应的执行次数分析。

初值和**终值**也只能是[0, 10000]范围内的**常数**，不能用循环控制变量代替，并且要求**初值**不能大于**终值**。此外，每一个**循环控制变量**都只能在相应的循环中使用。

```
//置换表 P[0]: {1, 2, 3, 0}
//tmp 初始为"0001"
LOOP i 1 3
    tmp = P[0](tmp)
ENDLOOP
//此时 tmp 为"1000"
//三条语句，循环 3 次，所以语句执行总次数为 9。

//置换表 P[0]: {1, 2, 3, 0}
//置换表 P[1]: {3, 0, 1, 2}
LOOP i 0 1
    LOOP j 1 999
        tmp = P[i](tmp)
   ENDLOOP
ENDLOOP
//循环前后 tmp 不变
//中间三条语句共循环 1998 次，首尾两条语句循环两次，总计 5998 次。
LOOP a 10000 10000
ENDLOOP
//一个循环什么都不做也是可以的
//两条语句，循环 1 次，所以语句执行总次数为 2。
//错误的例子
LOOP i 0 1
    tmp = P[i](tmp)
ENDLOOP
tmp = P[i](tmp)      //循环外部不能再使用 i
```

3. 分组(SPLIT/MERGE)

为了对二进制串进行更为细致的操作，顿顿使用 SPLIT 和 MERGE 来实现二进制串的拆分与合并。

```
SPLIT(二进制串变量, 常数)
... 二进制串变量 [常数] ...
... 二进制串变量 [循环控制变量] ...
MERGE(二进制串变量)
```

```
SPLIT(state,2)
... state[0] ...
... state[1] ...
... state[i] ...
MERGE(state)
```

SPLIT 和 MERGE 总是成对出现。SPLIT 把二进制串等分为若干份，在上面的例子中，state 被切成两半。在相应的 MERGE 语句之前，将使用 state[0] 和 state[1] 来表示 state 的左右两半。更一般地，如果将 state 分成 64 段，将使用 state[0] 到 state[63] 来表示其中每一部分(方括号中也可以使用**循环控制变量**)。为了保证可以等分，顿顿要求 SPLIT 语句中的**常数**必须能整除**二进制串变量**的长度且大于 1。而 MERGE 则是将拆分出来的若干段 state[...] 再重新合并回 state。

需要注意的是，在拆分期间 state[...] 将作为新的**二进制串变量**取代 state 存在，即在 MERGE(state) 之前都不能直接使用 state。并且，state[...] 作为 **二进制串变量**仍然可以继续进行拆分，下面是两个简单的例子。

```
state(8)
key(4)
BEGIN
SPLIT(state,2)
    state[0] = state[0] + "1100"
    state[1] = state[1] + key
MERGE(state)
state = state + "01010101" //MERGE 后才可以继续使用 state
END
//明文的左边四位异或"1100"，右边四位与密钥异或，
//最后整体异或"01010101" 得到密文。
```

```
//置换表 P[0]: {1, 2, 3, 0}
//置换表 P[1]: {3, 0, 1, 2}
state(64)
key(32)
BEGIN
SPLIT(state,8)
LOOP i 0 7
```

```
        SPLIT(state[i],2)
        LOOP j 0 1
            state[i][j] = P[j](state[i][j])
        ENDLOOP
        MERGE(state[i])
    ENDLOOP
    MERGE(state)
    END
    //把明文分成八块，每一块中的左四位使用 P[0] 进行置换，
    //右四位使用 P[1] 进行置换。
```

三、一些格式上的约定

该语言严格区分大小写，**保留字**均为大写字母，而变量名均为小写字母。这里**保留字**指的是有特定用途的字符串，包括 BEGIN、END、P、S、LOOP、ENDLOOP、SPLIT 和 MERGE。

变量声明部分每行声明一个变量，**加密算法部分**每行一条语句，再加上 BEGIN 和 END 代码总共不超过 50 行。

循环语句的各部分（LOOP、循环控制变量、初值 和 终值）之间至少有一个空格分隔，其余地方可以不用空格分隔。

在有具体含义的字母串（保留字、变量名）和数字串（常量）内部，不能插入空格和制表符\t;对于**二进制串常量**，双引号和 01 串同样视作一个整体不可分隔。除此之外，可以在代码中的任意地方添加空格和制表符，但要求每行不得超过 10^2 个字符。

试实现一个分组加密器，可以把顿顿的代码转化为相应的加密程序。即根据输入的 置换表、S 盒、代码，对若干组明文和密钥进行加密，输出相应的密文。保证输入的代 码满足上述所有要求。

【输入格式】

从标准输入读入数据。

输入第一行包含用空格分隔的两个整数 n 和 m，表示有 n 个置换表 和 m 个 S 盒，保证 $0 \le n,m \le 10$，然后依次输入这些置换表和 S 盒。

每个置换表占两行，第一行包含用空格分隔的两个正整数 a 和 b，分别表示该置换 表对应输入、输出二进制串的长度，保证 $1 \le a, b \le 64$;第二行 b 个用空格分隔的整数， 其中每个整数都在 $[0, a)$ 范围内，表示该置换表的内容。

每个 S 盒占两行，第一行包含用空格分隔的两个正整数 c 和 d，分别表示该 S 盒对 应输入、输出二进制串的长度，保证 $1 \le c, d \le 8$;第二行 2c 个用空格分隔的整数，其 中每个整数都在 $[0, 2d)$ 范围内，表示该 S 盒的内容。

从第 $2 + 2n + 2m$ 行开始，输入顿顿的代码。END 位于代码最后一行，可以借此判读是否读入了全部代码。

接下来一行包含一个正整数 k，表示有 k 组数据需要加密，保证 $k \le 10$。

在最后的 2k 行里，每两行包含一组加密数据，其中第一行为明文、第二行为密钥， 皆以 01 串的形式给出。

【输出格式】

输出到标准输出。

输出 k 行，每行一个 01 串表示相应的密文。

【样例 1 输入】

```
0 0
        state (    10    )
              key(1)
BEGIN
END
    1
    0000000000
    1
```

【样例 1 输出】

```
0000000000
```

【样例 1 解释】

实际上这个“加密算法”什么也没有做，所以将明文原样输出即可。

【样例 2 输入】

```
1 1
4 4
1 2 3 0
4 4
0 1 2 3 3 2 1 0 12 13 14 15 15 14 13 12
    state(4)
    key(4)
    tmp(4)
    BEGIN
    tmp = state
    LOOP i 1 3
        tmp = S[0](P[0](tmp)) + key
    ENDLOOP
 state = state + tmp + "1111"
 END
 1
 0101
 1100
```

【样例 2 输出】 1010

【样例 2 解释】 将明文取反。