C7 - Statement

A Otter256

题目背景

水獭学院的每位水獭都拥有唯一的学号，格式为 `OTTERXXX`，其中前 5 个字符是固定的大写字母 `OTTER`，后三个字符是从 `000` 到 `256` 的数字（含 `000` 和 `256` 需要包含前导零）。

现在，小水獭需要编写一个程序，帮助验证某个学号是否为合法的学号。

输入格式

不定组输入，每组输入一行，包含一个长度为 8 的字符串 S，保证数据不超过 1000 组，保证字符串只有大小写字母和数字组成。

输出格式

如果 S 是合法的学号，输出 `Valid`；如果 S 不是合法的学号，输出 `Invalid`。

输入样例

```
OTTER003
OTTER257
OTTER12a
```

输出样例

```
Valid
Invalid
Invalid
```

Hint

可以参考下面的代码片段：

```
char str[10];
while(scanf("%s", str) != EOF) {
    if(str 前五位是 OTTER 且后三位是数字) {
        // 拆分出后三位表示的数字 num
        int num = (str[5] - '0') * 100 + (str[6] - '0') * 10 + (str[7] - '0');
        if(num 在 [0, 256] 范围内)
            printf("Valid\n");
        else
            printf("Invalid\n");
    } else
        printf("Invalid\n");
}
```

在判断前五位是否为 `OTTER` 时，可使用库函数 `int strncmp(const char *str1, const char *str2, size_t n)`，它可以比较两个字符串的前 n 个字符是否相等，相等时返回 `0`，不相等时依据字典序关系返回正数或者负数。比如我们可以用 `strncmp(str, "OTTER", 5) == 0` 来判断前五位是不是 `OTTER`。当然也可以选择逐位检查前五个字符。

B 爱吃糖葫芦

题目描述：

> 阿呦阿呦，爱吃甜甜的糖葫芦

> 阿呦阿呦，偶尔也犯点小迷糊

在某个热闹街角，有一家小摊因其独特的糖葫芦而备受欢迎。这里的糖葫芦不仅味道独特，还有一个特别的传统：每串糖葫芦都对应着一句神秘的“密语”。只有真正的老顾客才能凭借对摊主的了解，说出这句密语，从而触发一个特别的隐藏事件——免单。

阿呦作为糖葫芦的忠实爱好者，自然了解这些密语的构造规则：**摊主会按照一个固定的顺序将3种食材递给顾客，当顾客依次将食材串到竹签上时，密语就是所有食材名称沿竹签自顶向下组成的符号串的后半部分。**

现在，阿呦又一次来到了小摊前，他已经迫不及待想要品尝今天的糖葫芦了。请你帮助阿呦，根据摊主递给他的食材顺序，大声地说出那句密语，确保他能够准确地触发免单的优惠。

输入格式：

总共 3 行，每行包含一个由小写英文字母组成的字符串 s $(2 \leq |s| \leq 1000)$，且$|s|$为偶数，代表一种食材的名称。

输出格式：

输出一行，一个字符串，表示这串糖葫芦对应的密语（**将三种食材的名称按给定顺序的倒序拼接在一起，后半部分即为密语**）。

输入样例：

```
hawthorn
strawberry
pear
```

输出样例：

```
rryhawthorn
```

样例解释

先按顺序串联起所有的字符串，得到一个字符串`pearstrawberryhawthorn`，再截取字符串的后半段，即可得到密语`rryhawthorn`。

Hint

怎么才能得到触发免单的密语呢？不妨试试`char *strcat(char *dest, const char *src)`函数吧！

字符串追加函数：strcat, strncat

(cat, concatenate, 连接)

```
char *strcat(char dest[ ], const char src[ ]);
```

将字符串 src 复制到字符串 dest 已有字符串后面（追加），src 的第一个字符重定义 dest 的\0终止符，返回 dest []。

```
char *strncat(char dest[ ], const char src[ ], size_t n);
```

将字符串 src 中最多前 n 个字符添加到字符串 dest 后面，src 的第一个字符重定义 dest 的\0终止符，返回 dest[]。

应用：今天的作业没有写完，明天接着写

c 林士谔算法 信类2024ver

题目描述

林士谔（1913-1987）先生是中国著名的自动控制学家和航空教育专家，广东平远人。他于1935年毕业于上海交通大学电机系，随后于1939年获得美国麻省理工学院航空系航空工程博士学位。林士谔先生是北京航空航天大学的建校元老之一，对中国的航空自动控制学科和陀螺惯导学科有着奠基性的贡献。

林士谔算法，是林士谔先生在上世纪三十年代首次提出的一种求解高次方程近似解的方法。林士谔算法可以对一个次数大于等于 3 的多项式 $f(x)=a_nx^n+\cdots+a_1x+a\_0$，近似地求出一个形如 x^2+px+q 的二次因子。这个二次因子对应的方程 $x^2+px+q=0$ 的解就是方程 $f(x)=0$ 的近似解。因此，如果我们得知了 $f(x)$ 的各项系数，我们可以通过求出 p,q，进而求出 $f(x)=0$ 的近似解。

对于输入的 n 和 $a_n,a_{n-1},\ldots,a_1,a_0$，通过调用函数 `Shie`，可以计算得到多项式的近似解。函数 `Shie` 的定义如下：

```
#include<stdio.h>
#include<string.h>
double a[20];
void Shie(int n, double a[], double *p, double *q);

int main()
{
    /*------下面根据指引写你自己的代码------*/
    //定义 int 型变量 n，double 型变量 p 和 q
    //读取n
    //从第n项开始，一直到第0项，读取全局数组a
    //调用Shie函数
    //输出p q
    return 0;
}

/*--下面是Shie函数，只要调用即可，无需理解原理--*/
void Shie(int n, double a[], double *p, double *q)
{
    double eps = 1e-12;
    double b[20];
    double c[20];
    // 数组 b 是多项式 a 除以当前迭代二次三项式的商
    memset(b, 0, sizeof(b));
    // 数组 c 是多项式 b 乘以 x 平方再除以当前迭代二次三项式的商
    memset(c, 0, sizeof(c));
    *p = 0;
    *q = 0;
    double dp = 1;
    double dq = 1;
    while (dp > eps || dp < -eps || dq > eps || dq < -eps)  // eps 自行设定
    {
        double p0 = *p;
        double q0 = *q;
        b[n - 2] = a[n];
        c[n - 2] = b[n - 2];
        b[n - 3] = a[n - 1] - p0 * b[n - 2];
        c[n - 3] = b[n - 3] - p0 * b[n - 2];
        int j;
        for (j = n - 4; j >= 0; j--) {
            b[j] = a[j + 2] - p0 * b[j + 1] - q0 * b[j + 2];
            c[j] = b[j] - p0 * c[j + 1] - q0 * c[j + 2];
        }
        double r = a[1] - p0 * b[0] - q0 * b[1];
        double s = a[0] - q0 * b[0];
        double rp = c[1];
        double sp = b[0] - q0 * c[2];
        double rq = c[0];
        double sq = -q0 * c[1];
        dp = (rp * s - r * sp) / (rp * sq - rq * sp);
        dq = (r * sq - rq * s) / (rp * sq - rq * sp);
        *p += dp;
        *q += dq;
    }
    return;
}
```

函数利用了指针类型的参数来保存多个返回结果（相当于利用指针将函数里对参数造成的影响传递到函数外），`*p` 和 `*q` 分别表示二次因子 x^2+px+q 中的一次项系数 p 和常数项系数 q。

请你根据上面的函数，以及输入的变量 n 和 $a_n,a_{n-1},\ldots,a_1,a_0$，求出参数 p,q。

输入

两行，第一行一个整数 n，保证 $3\le n\le 10$；

第二行 $n+1$ 个保留一位小数的浮点数 $a_n,a_{n-1},\ldots,a_1,a_0$，保证其绝对值均不大于 50。

注意数组的读入顺序（按降幂顺序读取），从 a_n 开始读取（a_n 是 $n+1$ 个数的第一个），最后一个读取 a_0，**不要读取反了。**

输出

一行两个浮点数 p, q，保留 6 位小数。

输入样例 1

```
6
1.0 -4.0 11.0 -18.0 22.0 -16.0 8.0
```

输出样例 1

```
-1.000000 2.000000
```

输入样例 2

```
10
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
```

输出样例 2

```
0.284630 1.000000
```

Hint

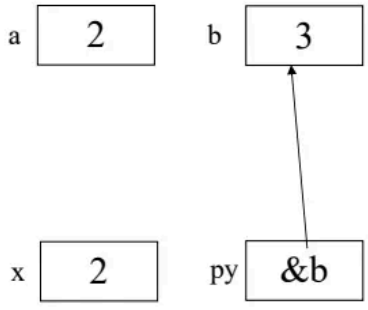
可以参考 PPT 7.2.2:

7.2.2 作为函数参数的指针

- C语言中的函数参数是值传递的，函数调用不会改变实参变量的值，一般通过返回值或者修改全局变量的方式实现。
- 定义指针作为函数参数，可以以指针的方式改变函数外部的变量值。
- 当一个函数需要同时向外部传递多个数值而又不希望改变全局变量时，可使用指针类型的函数参数。

```
int a = 2, b = 3, c = 4;
c = sum(a, &b);
...


int sum(int x, int *py)
{
    x += *py;
    *py = x;
    return x;
}
```



D 魔法师水獭与标准咒语指南

题目背景

魔法师水獭正在学习《标准咒语指南》，书中隐藏着许多强大的咒语。为了提高学习效率，魔法师水獭希望能找到某些咒语在书中的位置。

现在，请你写一个程序帮助魔法师水獭找到魔法书中第一个匹配的咒语。如果找到了咒语，就输出它在魔法书中首次出现的起始位置（从 0 开始计数）；如果没有找到，就输出 `Spell Not Found!`。

输入格式

第一行是一个字符串，表示魔法书的内容（长度不超过 1000）。

第二行一个正整数 n，表示要寻找 n 个咒语，保证 n 不超过 100。

接下来 n 行，每行一个字符串，表示要寻找的咒语关键词（长度不超过 1000），保证不以空格开头或结尾，但是中间可能含有空格。

保证字符串中的字符 ASCII 值在 $[32, 126]$ 范围内。

输出格式

- 如果在魔法书中找到了关键词，输出关键词在字符串中第一次出现的起始位置（从 0 开始）。
- 如果没有找到，输出 `Spell Not Found!`。

输入样例

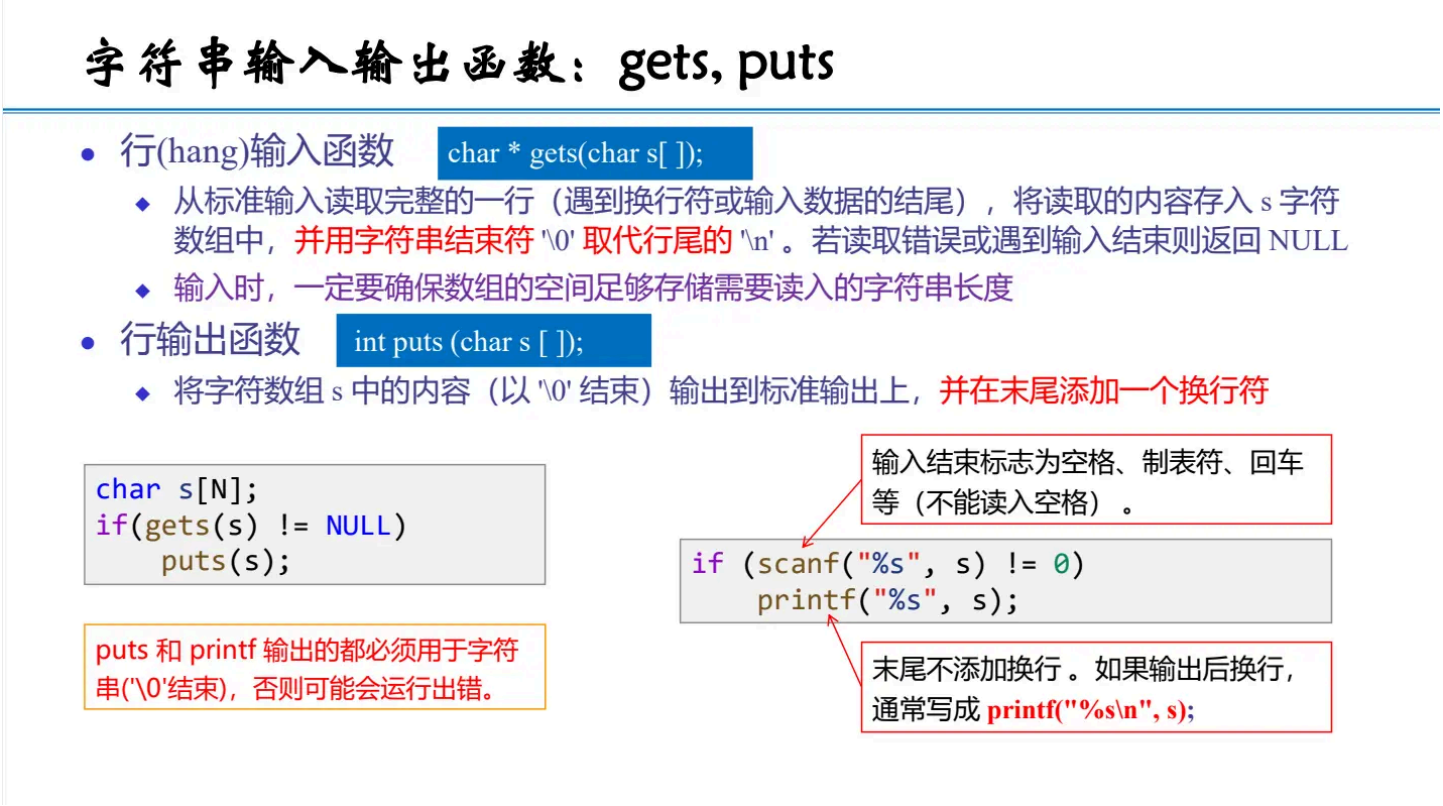
```
Expecto Patronum! Otter whispered into the darkness.
4
Patronum
to
Avada Kedavra
Otterwhispered
```

输出样例

```
8
5
Spell Not Found!
Spell Not Found!
```

Hint

在读入时，由于输入的字符串中含有空格，可以使用 `gets` 来进行读入，同时别忘了**处理整数后面的换行符**:



可以使用 `<string.h>` 中的函数 `char *strstr(char *s, char *sub_str);` 在字符串中查找子串，具体用法可参考例 7 - 9：

- 核心操作是需要在 buf 中定位 str 的位置，可以使用标准库函数：

```
char *strstr(const char *_Str, const char *_SubStr);
```

strstr 函数在字符串_Str 中寻找子串_SubStr，如果找到则返回在_Str 中指向_SubStr 子串的指针，否则返回空指针。

E 摩卡与指针

题目背景

Moca 今天学习了指针，并发现指针和地址有着紧密的联系。通过学习，他了解到数组的元素在内存中是连续存储的，因此可以根据数组的起始地址、元素类型的大小以及数组下标，计算出数组中每个元素的内存起始地址。

假设有一个数组 A，它的起始地址是 $base$，数组的元素类型大小为 $size$。数组中第 i 个元素的内存起始地址可以通过以下公式计算：

$$\text{address}(i) = \text{base} + (i \times \text{size})$$

请你帮助 Moca 完成一个程序，查询数组中某元素的起始地址（假设数组的起始地址为 `0x00000000`）。

输入格式

第一行包含一个字符串 $type$，表示数组的 **元素类型**，$type$ 的可能取值为：

- `int`：表示整型，大小为 4 字节。
- `char`：表示字符型，大小为 1 字节。
- `float`：表示浮点型，大小为 4 字节。
- `double`：表示双精度浮点型，大小为 8 字节。
- `long long`：表示长整型，大小为 8 字节。

第二行包含一个整数 n ($1 \leq n \leq 10^5$) 和一个整数 q ($1 \leq q \leq 10000$)，分别表示数组长度和查询次数。

第三行包含 q 个整数 $i_1, i_2, \ldots, i_q$ ($0 \leq i_k \leq 10^6$)，表示每次查询的数组下标。

输出格式

对于每次查询，输出一行如果下标 i_k 在合法范围内 ($0 \leq i_k < n$)，输出数组中第 i_k 个元素的内存起始地址，格式为 `0x` + 对应的八位十六进制数（不足八位用前导 0 来补位，字母使用**小写字母**）；如果下标 i_k 超出合法范围，输出 `Careless Otter!`。

输入样例 1

```
int
5 3
0 3 5
```

输出样例 1

```
0x00000000
0x0000000c
Careless Otter!
```

输入样例 2

```
char
10 4
0 9 10 15
```

输出样例 2

```
0x00000000
0x00000009
Careless Otter!
Careless Otter!
```

Hint

由于数据类型 `long long` 中间含有空格，你可能需要使用 `gets` 来进行字符串的读入；可以用 `printf("0x%08x", num)` 实现题目中要求的以 `0x` 开头的八位十六进制输出，如 `printf("0x%08x", 26)` 会输出 `0x0000001a`。

F ddz 点亮矿洞

题目描述

ddz 在玩 $Minecraft$ 时在自己的家下面开了一条 n 格的矿道，由于担心刷怪， ddz 就在矿道里放置了 k 个铜灯来点亮矿洞。但由于没有蜜脾给铜灯涂蜡，一些铜灯被氧化了，氧化的铜灯的亮度降低，就可能导致一些地方会刷怪，请你帮 ddz 数一下矿洞中有多少格有可能刷怪。（当一个格子的亮度**小于等于** 7 的时候会刷怪，部分数值不一定与游戏相符，以题面为准）

根据氧化程度铜灯分为**普通**、**斑驳**、**锈蚀**和**氧化**四个等级，它们的亮度分别是 15，12，8，4 。最开始没有铜灯时每个格子的亮度是 0 ，亮度传播时每经过一格会减一，多个铜灯同时照到一个格子时取其中最大的亮度，例如对于一条长 14 格的矿道，在第 3 格有一个锈蚀的铜灯，在第 10 格有一个氧化的铜灯时，整个矿道的亮度为 $[6, 7, 8, 7, 6, 5, 4, 3, 3, 4, 3, 2, 1, 0]$ ，一共有 13 个格子会刷怪。

输入

第一个数为数据组数 t $(1 \leq t \leq 100)$ 。

接下来 t 组数据，每组数据第一行为 2 个整数 n，k $(1 \leq k \leq n \leq 5 \times 10^5)$，接下来 k 行，每行两个数 a，b 表示第 a 格有一个氧化程度为 b 的铜灯（$b = 0,\ 1,\ 2,\ 3$分别对应**普通**、**斑驳**、**锈蚀**和**氧化**）。

数据保证 n 的总和不超过 5×10^5 。

输出

对于每组数据，输出一行一个整数 x，表示有 x 格会刷怪。

输入样例

```
2
14 2
3 2
10 3
30 4
3 0
25 1
16 3
20 2
```

输出样例

```
13
10
```

样例解释

1. 第一组数据即题面
2. 矿洞亮度为 $[13, 14, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 4, 5, 6, 7, 8, 8, 9, 10, 11, 12, 11, 10, 9, 8, 7]$

Hint

由于是多组数据，你可能需要用 for 循环先将数组的前 n 个数清零再进行计算

Author: ddz

G 盟主的挑战--逆序真言

题目描述

> 工欲善其事，必先利其器

在武林的激烈角逐中，T盟盟主正准备迎接即将到来的武林争霸赛。这场赛事不仅是武技的较量，更是内功心法的比拼。盟主在T盟的秘藏中发现了一本古老的内功秘籍，其中记载着一种神秘的内功修炼法门——“逆序真言”。这种法门能够激发武者体内的潜能，但修炼过程极为复杂和神秘，需要按照特定的规则来解读和逆置口诀中的字符。

对于每句功法口诀（每k个字符为一句），按照以下规则处理后，输出最终口诀串联得到的字符串：

1. 如果包含秘籍的名称G(保证一句口诀内最多包含一个G子串)，那么**在逆置过程中，G这个子串，相对于这句口诀的位置和顺序不能变动，只逆置其他字符。**

2. 首先判断其逆置后的字典序是否大于原始顺序的字典序。
    - 如果逆置后的字典序大于原始顺序的字典序，则直接逆置这句口诀。
    - 如果逆置后的字典序不大于原始顺序的字典序，则将原口诀内的每个字母颠倒（即a变为z，b变为y，以此类推）。

输入

- 第一行是一个正整数$k(1 \leq k \leq 10^4)$，表示每k个字符为一句。
- 第二行是一个字符串$G(1 \leq |G| \leq 10^3)$，只包含小写英文字母，表示秘籍的名字。
- 第三行是一个字符串$S(1 \leq |S| \leq 10^4)$，只包含小写英文字母，表示所有功法口诀。(保证$|S|$可以被k整除)

输出

输出共一行，一个字符串，表示最终的功法口诀。

输入样例

```
7
jug
zheshiyijugongfakouju
```

输出样例

```
asvhsrbrqftlmtujuokaf
```

样例解释

将S按照每7个字符分割成3句口诀，得到 [`zheshiy` , `ijugong` , `fakouju`]。

- 第一句 `zheshiy` ，不包含 `jug` 子串，因此逆置结果为 `yishehz` ， `yishehz` 字典序小于 `zheshiy` ，因此将原口决 `zheshiy` 每个字母颠倒得到 `asvhsrb` 。
- 第二句 `ijugong` ，包含 `jug` 子串， `jug` 字串在变换前后的位置和顺序不变，因此逆置结果为 `gjugnoi` ， `gjugnoi` 字典序小于 `ijugong` ，因此将原口决 `ijugong` 每个字母颠倒得到 `rqftlmt` 。
- 第三句 `fakouju` ，不包含 `jug` 子串，因此逆置结果为 `ujuokaf` ， `ujuokaf` 字典序大于 `fakouju` ，因此保留此次逆置结果 `ujuokaf` 。
- 将三句最终口诀拼接在一起，即得到最终的功法 `asvhsrbrqftlmtujuokaf`

Hint

测评机没有内置 `strrev` 函数哦，请参考课件【例6-5】实现自己的 `strrev` 吧!

字符数组示例

【例6-5】一行字符串倒置

```
#include <stdio.h>
#include <string.h>
void str_rev(char []);

int main()
{
    char a[100];
    int i, hi=0, low=0;
    gets(a);

    puts(a);
    str_rev(a);
    puts(a);
    return 0;
}
```

gets: 读入一行字符串（回车结束，支持空格读入）
scanf: 空格不能读入
如果写成 scanf("%s", a); 结果怎么样？
puts vs printf ？

```
void str_rev(char s[])
{
    int hi = 0, low = 0;
    char temp;
    while (s[hi] != '\0')
        hi++;
    for (hi--; hi > low; low++, hi--)
    {
        temp = s[low];
        s[low] = s[hi];
        s[hi] = temp;
    }
}
```

H 盟主的挑战--不战而胜

题目描述

> 百战百胜，非善之善者也；不战而屈人之兵，善之善者也。

武林争霸赛的战火愈演愈烈，T盟盟主的雄心壮志无人能挡。在成功击败了一批擂主之后，局势发生了一些变化：擂主们听闻盟主的威名，纷纷选择避战。在这场智谋与力量的较量中，盟主的威慑力成为了关键因素。已知在x轴上分布着n个擂台，每个擂台的位置用a_i表示。盟主的战斗力为$\mathbb{MAX}$，威慑力为m，**当位于a_i的擂台认输时，所有在a_i周围m距离以内的擂主都会感受到盟主的威慑力，选择认输**。已知盟主战无不胜，时间紧任务重，为了使所有擂主都屈服于他的威慑之下，至少需要发起多少次挑战。

输入格式如下：

- 第一行是一个正整数 T （$1 \leq T \leq 50$），表示数据组数。
- 接下来是T组数据：
  - 第一行，是两个正整数$n, m (1 \leq n \leq 10^3,\ 1 \leq m \leq 10^9)$，分别擂台数量和盟主的威慑力。
  - 第二行，n个正整数 $a_1, a_2, \ldots, a_n (1 \leq a_i \leq 10^9)$，表示擂台所在位置。

输出格式如下：

对于每组数据，输出一行一个正整数，表示盟主最少需要发起多少次挑战。

输入样例：

```
2
5 2
1 5 9 3 7
4 3
8 2 6 1
```

输出样例：

```
1
2
```

样例解释

解释第一组样例：擂台在x轴上的位置分别是1、3、5、7、9。盟主的威慑力是2。我们可以通过以下战斗顺序来让所有的擂台认输：

1. 首先，盟主选择挑战位于x=1位置的擂台。由于盟主的威慑力范围是2，这个擂台认输后，所有在x=1±2范围内的擂台也会受到影响，即x=-1到x=3的区间。因此，位于x=3的擂台也会认输。
2. 接下来，由于x=3的擂台已经被认输，盟主的威慑力进而会覆盖x=3±2的范围，即x=1到x=5的区间，因此，位于x=5的擂台也认输。
3. 通过这种方式，盟主仅通过挑战位于x=1的擂台一次，就使得所有在威慑力范围内的擂台都认输了。

综上所述，盟主最少只需要发起1次挑战，就能让所有擂主都屈服于他的威慑力之下。

I Moca 与水獭排队 3

题目描述

现在有 n 只水獭站成一排。Moca 希望每次能找到**不低于**某身高 h 的第一只水獭的编号（编号从 1 开始），请你编写一个程序完成这个任务。

输入格式

第一行两个正整数 n 和 m，其中 n 表示小水獭的个数，保证 $1 \leq n \leq 5 \times 10^5$；$m$ 表示接下来询问的次数，保证 $1 \leq m \leq 5 \times 10^5$。

接下来一行，输入 n 个正整数，第 i 个正整数表示第 i 只小水獭的身高 a_i，保证 $1 \leq a_i \leq 10^7$ 。

接下来 m 行，每行一个正整数 h_i，表示要查找的值，保证 $1 \leq h_i \leq 10^7$。

输出格式

输出 m 行，对每次询问，输出对应的不低于对应身高 h 的第一只水獭的编号；如果不存在这样的水獭输出 `-1`。

样例输入

```
5 5
2 7 10 1 13
1
8
10
12
14
```

样例输出

```
1
3
3
5
-1
```

Author：Moca, Saisyc

J Baymax 的困难字符串

题目描述

如果一个字符串包含两个相邻的重复子串，则称它是「容易的串」；其他串称为「困难的串」。例如，`BB`、`ABCABC`、`ABCDACABCAB` 都是容易的串，而 `D`、`DC`、`ABDAB`、`CBABCBA` 都是困难的串。

对于正整数 n 和 L， Baymax 想知道由前 L 个大写字母组成的、字典序第 n 小的困难的串是什么，请你帮帮他吧~

输入

共一行，两个正整数 n 和 L。保证 $n \leq 10000$，$L \leq 26$ 。

输出

共一行，表示由前 L 个大写字母组成的、字典序第 n 小的困难的串。保证输出不超过 10000 个字符。

输入样例 1

```
7 3
```

输出样例 1

```
ABACABA
```

输入样例 2

```
30 3
```

输出样例 2

```
ABACABCACBABCABACABCACBACABA
```

样例解释

对于样例 1，前 7 个困难的串分别为 `A`、`AB`、`ABA`、`ABAC`、`ABACA`、`ABACAB`、`ABACABA`。