



# 历年 CSP 题目解析

仅为参考练习所用

作者: Ionlyn

组织: Shanxi University Algorithm Group

时间: December 30, 2021

版本: 1.0

## 特别声明

该书仅供内部学习使用，如果有侵权请联系作者。

信息学竞赛的发展，吸引越来越多的人加入了“卷”的行列。CCF CSP 的历年题解在网上也是随处可见，但题解质量参差不齐。很多题解只有标准答案，缺少题目分析；更有甚者无法通过答案，充满了分号大小写问题等错误。

本书的目的是为了实现以下几点：

- 提供规范的代码程序。这里的规范，既要具有程序的可读性，也要具备考场的简易性。
- 提供多样的解题思路。有些时候，网上的大佬往往一语道破问题求解的思路，但怎么想到的却往往不提。这里力求从部分分开始，逐渐深入，汇集众人智慧，逐步解决难题。
- 提供筛选的额外补充。做一道题的目的不是只做一道题，而是可以做到举一反三，但我们常常忽略这一点。

感谢 [Elegant<sup>La</sup>T<sub>E</sub>X](#) 提供如此精美的模板，希望这本书能够给大家带来帮助。

lonlyn

December 30, 2021

# 目录

1	CCF CSP 认证总览	1
2	第 24 次认证 (2021 年 12 月)	2
2.1	题目及涉及知识点	2
2.2	202112-1 序列查询	3
2.2.1	50% 数据——模拟	4
2.2.1.1	思路	4
2.2.1.2	C++ 实现	4
2.2.2	100% 数据——利用 $f(x)$ 单调性	4
2.2.2.1	思路	4
2.2.2.2	C++ 实现	5
2.2.3	100% 数据——阶段求和	5
2.2.3.1	思路	5
2.2.3.2	C++ 实现	6
2.3	202112-2 序列查询新解	7
2.3.1	与上一题的比较	9
2.3.2	70% 数据——计算出每个 $f(x), g(x)$ 的值	9
2.3.2.1	思路	9
2.3.2.2	C++ 实现	9
2.3.3	100% 数据——对 $f(x), g(x)$ 都相同的区间进行求和处理	9
2.3.3.1	思路	9
2.3.3.2	C++ 实现	9
2.3.4	100% 思路——以 $f(x)$ 为单位, 讨论内部 $g(x)$ 求和	10
2.3.4.1	思路	10
2.3.4.2	C++ 实现	11
2.4	202112-3 登机牌条码	14
2.4.1	40% 数据——直接模拟	18
2.4.1.1	思路	18
2.4.1.2	C++ 实现	18
2.4.2	100% 数据——模拟 + 多项式除法	20
2.4.2.1	思路	20
2.4.2.2	C++ 实现	21
2.5	202112-4 磁盘文件操作	25
2.6	202112-5 极差路径	28

# 第 1 章 CCF CSP 认证总览

待补充。

## 第 2 章 第 24 次认证（2021 年 12 月）

### 2.1 题目及涉及知识点

题目编号	题目名称	知识点
1	序列查询	数学
2	序列查询新解	数学
3	登机牌条码	模拟，多项式除法
4	磁盘文件操作	线段树
5	极差路径	树分治

## 2.2 202112-1 序列查询

### 题目背景

西西艾弗岛的购物中心里店铺林立，商品琳琅满目。为了帮助游客根据自己的预算快速选择心仪的商品，IT 部门决定研发一套商品检索系统，支持对任意给定的预算  $x$ ，查询在该预算范围内 ( $\leq x$ ) 价格最高的商品。如果没有商品符合该预算要求，便向游客推荐可以免费领取的西西艾弗岛定制纪念品。

假设购物中心里有  $n$  件商品，价格从低到高依次为  $A_1, A_2, \dots, A_n$ ，则根据预算  $x$  检索商品的过程可以抽象为如下序列查询问题。

### 题目描述

$A = [A_0, A_1, A_2, \dots, A_n]$  是一个由  $n+1$  个  $[0, N)$  范围内整数组成的序列，满足  $0 = A_0 < A_1 < A_2 < \dots < A_n < N$ 。（这个定义中蕴含了  $n$  一定小于  $N$ 。）

基于序列  $A$ ，对于  $[0, N)$  范围内任意的整数  $x$ ，查询  $f(x)$  定义为：序列  $A$  中小于等于  $x$  的整数里最大的数的下标。具体来说有以下两种情况：

1. 存在下标  $0 \leq i < n$  满足  $A_i \leq x < A_{i+1}$ ，此时序列  $A$  中从  $A_0$  到  $A_i$  均小于等于  $x$ ，其中最大的数为  $A_i$ ，其下标为  $i$ ，故  $f(x) = i$ 。
2.  $A_n \leq x$ ，此时序列  $A$  中左右的数都小于等于  $x$ ，其中最大的数是  $A_n$ ，故  $f(x) = n$ 。

令  $sum(A)$  表示  $f(0)$  到  $f(N-1)$  的总和，即：

$$sum(A) = \sum_{i=0}^{N-1} f(i) = f(0) + f(1) + f(2) + \dots + f(N-1)$$

对于给定的序列  $A$ ，试计算  $sum(A)$ 。

### 输入格式

从标准输入读入数据。

输入的第一行包含空格分隔的两个正整数  $n$  和  $N$ 。

输入的第二行包含  $n$  个用空格分隔的整数  $A_1, A_2, \dots, A_n$ 。

注意  $A_0$  固定为 0，因此输入数据中不包括  $A_0$ 。

### 输出格式

输出到标准输出。

仅输出一个整数，表示  $sum(A)$  的值。

### 样例

输入 #1:

```
3 10
2 5 8
```

输出 #1:

```
15
```

解释 #1:

$A = [0, 2, 5, 8]$

$i$	0	1	2	3	4	5	6	7	8	9
$f(i)$	0	0	1	1	1	2	2	2	3	3

如上表所示,  $sum(A) = f(0) + f(1) + \dots + f(9) = 15$ 。

考虑到  $f(0) = f(1)$ 、 $f(2) = f(3) = f(4)$ 、 $f(5) = f(6) = f(7)$  以及  $f(8) = f(9)$ , 亦可通过如下算式计算  $sum(A)$ :

$$sum(A) = f(0) \times 2 + f(2) \times 3 + f(5) \times 3 + f(8) \times 2$$

输入 #2:

```
9 10
1 2 3 4 5 6 7 8 9
```

输出 #2:

```
45
```

## 子任务

50 % 的测试数据满足  $1 \leq n \leq 200$  且  $n \leq N \leq 1000$ ;

全部的测试数据满足  $1 \leq n \leq 200$  且  $n \leq N \leq 10^7$ 。

## 提示

若存在区间  $[i, j]$  满足  $f(i) = f(i+1) = \dots = f(j-1)$ , 使用乘法运算  $f(i) \times (j-i)$  代替将  $f(i)$  到  $f(j-1)$  逐个相加, 或可大幅提高算法效率。

### 2.2.1 50% 数据——模拟

#### 2.2.1.1 思路

模拟一下这个过程, 计算出每一个  $f(i)$  后加起来即可。

考虑针对确定的  $x$ , 如何求解  $f(x)$ 。我们可以从小到大枚举  $A$  中的数, 枚举到第一个大于等于  $x$  的数即可。注意末尾的判断。

枚举  $x$  时间复杂度  $O(N)$ , 计算  $f(x)$  时间复杂度  $O(n)$ , 整体时间复杂度  $O(nN)$ 。

#### 2.2.1.2 C++ 实现

待补充。

### 2.2.2 100% 数据——利用 $f(x)$ 单调性

#### 2.2.2.1 思路

为了方便, 设  $f(n+1) = \infty$ 。

通过模拟, 可以得到一个显然的结论:

**定理 2.1 ( $f(x)$  的单调性)**

对于  $x, y \in [0, N]$ , 若  $x \leq y$ , 则  $f(x) \leq f(y)$ 。



那么, 我们可以从小到大枚举  $x$ , 同时记录目前  $f(x)$  的值, 设为  $y$ , 那么  $A_{y+1}$  是第一个大于  $x$  的数。当需要计算  $f(x+1)$  的时候, 我们从小到大依次判断  $A_{y+1}, A_{y+2}, \dots$  是否满足条件, 直到遇到第一个大于  $f(x+1)$  的数  $A_z$ , 那么  $f(x+1) = z - 1$ 。之后, 在  $f(x+1)$  的基础上以同样的步骤求  $f(x+2)$ , 直到求完所有的值。

考虑该算法的时间复杂度, 枚举  $x$  的复杂度是  $O(N)$ , 而  $A$  数组中每个数对多被枚举一次, 枚举所有  $x$  的整体复杂度  $O(n)$ , 可以得到整体复杂度  $O(N + n)$ 。

**2.2.2.2 C++ 实现**

```
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
using namespace std;
#define ll long long
#define il inline
const int maxn = 210;
int n, N;
int a[maxn];
ll ans = 0;
int main() {
    scanf("%d%d", &n, &N);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &a[i]);
    }
    int cur = 0;
    for (int i = 0; i < N; ++i) {
        while (cur < n && a[cur + 1] <= i)
            ++cur;
        ans += cur;
    }
    printf("%lld\n", ans);
    return 0;
}
```

**2.2.3 100% 数据——阶段求和****2.2.3.1 思路**

在提示中, 指出了可以将  $f(x)$  相同的值一起计算。现在需要解决的问题是如何快速确定  $f(x)$  值相等的区间。

通过观察和模拟可以发现, 随着  $x$  增大,  $f(x)$  只会在等于某个  $A$  数组的值时发生变化。更具体的说, 对于某个属于  $A$  数组的值  $A_i$  来说,  $[A_i, A_{i+1} - 1]$  间的  $f(x)$  值是相同的, 这样的数共有  $A_{i+1} - A_i$  个。



也可以以另一种方式理解：对于一个值  $y$ ，考虑有多少  $x$  满足  $f(x) = y$ 。当  $x < A_y$  时， $f(x) < y$ ，当  $x \geq A_{y+1}$  时， $f(x) > y$ 。只有  $x \in [A_y, A_{y+1}]$  时才能得到  $f(x) = y$ 。

得到范围后，我们就可以根据  $A$  数组来进行求和计算。

考虑  $f(x) = n$  的处理：我们可以得知满足  $f(x) = n$  的  $x$  共有  $N - A_n$  个，根据上文推算，我们可以将  $A_{n+1}$  设置为  $A_n + (N - A_n) = N$  即可等效替代。

时间复杂度  $O(n)$ 。

### 2.2.3.2 C++ 实现

```
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
using namespace std;
#define ll long long
#define il inline
const int maxn = 210;
int n, N;
int a[maxn];
ll ans = 0;
int main() {
    scanf("%d%d", &n, &N);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &a[i]);
    }
    a[n + 1] = N;
    for (int i = 1; i <= n + 1; ++i) {
        // 处理区间 [A(i-1), A(i)] 的 f(x) 值的和
        ans += 1ll * (a[i] - a[i - 1]) * (i - 1);
    }
    printf("%lld\n", ans);
    return 0;
}
```

## 2.3 202112-2 序列查询新解

### 题目背景

上一题“序列查询”中说道： $A = [A_0, A_1, A_2, \dots, A_n]$  是一个由  $n+1$  个  $[0, N)$  范围内整数组成的序列，满足  $0 = A_0 < A_1 < A_2 < \dots < A_n < N$ 。基于序列  $A$ ，对于  $[0, N)$  范围内任意的整数  $x$ ，查询  $f(x)$  定义为：序列  $A$  中小于等于  $x$  的整数里最大的数的下标。

对于给定的序列  $A$  和整数  $x$ ，查询  $f(x)$  是一个很经典的问题，可以使用二分搜索在  $O(\log n)$  的时间复杂度内轻松解决。但在 IT 部门讨论如何实现这一功能时，小 P 同学提出了些新的想法。

### 题目描述

小 P 同学认为，如果事先知道了序列  $A$  中整数的分布情况，就能直接估计出其中小于等于  $x$  的最大整数的大致位置。接着从这一估计位置开始线性查找，锁定  $f(x)$ 。如果估计得足够准确，线性查找的时间开销可能比二分查找算法更小。

比如说，如果  $A_1, A_2, \dots, A_n$  均匀分布在  $(0, N)$  的区间，那么就可以估算出：

$$f(x) \approx \frac{(n+1) \cdot x}{N}$$

为了方便计算，小 P 首先定义了比例系数  $r = \lfloor \frac{N}{n+1} \rfloor$

，其中  $\lfloor \cdot \rfloor$  表示下取整，即  $r$  等于  $N$  除以  $n+1$  的商。进一步地，小 P 用  $g(x) = \lfloor \frac{x}{r} \rfloor$

表示自己估算出的  $f(x)$  的大小，这里同样使用了下取整来保证  $g(x)$  是一个整数。

显然，对于任意的询问  $x \in [0, N)$ ， $g(x)$  和  $f(x)$  越接近则说明小 P 的估计越准确，后续进行线性查找的时间开销也越小。因此，小 P 用两者差的绝对值  $|g(x) - f(x)|$  来表示处理询问  $x$  时的误差。

为了整体评估小 P 同学提出的方法在序列  $A$  上的表现，试计算：

$$error(A) = \sum_{i=0}^{N-1} |g(i) - f(i)| = |g(0) - f(0)| + \dots + |g(N-1) - f(N-1)|$$

### 输入格式

从标准输入读入数据。

输入的第一行包含空格分隔的两个正整数  $n$  和  $N$ 。

输入的第二行包含  $n$  个用空格分隔的整数  $A_1, A_2, \dots, A_n$ 。

注意  $A_0$  固定为 0，因此输入数据中不包括  $A_0$ 。

### 输出格式

输出到标准输出。

仅输出一个整数，表示  $error(A)$  的值。

### 样例

输入 #1:

```
3 10
2 5 8
```

输出 #1:

5

解释 #1:

$$A = [0, 2, 5, 8]$$

$$r = \lfloor \frac{N}{n+1} \rfloor = \lfloor \frac{10}{3+1} \rfloor = 2$$

$i$	0	1	2	3	4	5	6	7	8	9
$f(i)$	0	0	1	1	1	2	2	2	3	3
$g(i)$	0	0	1	1	2	2	3	3	4	4
$ g(i) - f(i) $	0	0	0	0	1	0	1	1	1	1

输入 #2:

```
9 10
1 2 3 4 5 6 7 8 9
```

输出 #2:

0

输入 #3:

```
2 10
1 3
```

输出 #3:

6

解释 #3:

$$A = [0, 1, 3]$$

$$r = \lfloor \frac{N}{n+1} \rfloor = \lfloor \frac{10}{2+1} \rfloor = 3$$

$i$	0	1	2	3	4	5	6	7	8	9
$f(i)$	0	1	1	2	2	2	2	2	2	2
$g(i)$	0	0	0	1	1	1	2	2	2	3
$ g(i) - f(i) $	0	1	1	1	1	1	0	0	0	1

## 子任务

70 % 的测试数据满足  $1 \leq n \leq 200$  且  $n \leq N \leq 1000$ ;

全部的测试数据满足  $1 \leq n \leq 10^5$  且  $n \leq N \leq 10^9$ 。

## 提示

需要注意，输入数据  $[A_1 \cdots A_n]$  并不一定均匀分布在  $(0, N)$  区间，因此总误差  $error(A)$  可能很大。

### 2.3.1 与上一题的比较

1. 上一题是求和，而本题要求求绝对值的和，无法转化为两者求差的形式。
2.  $f(x), g(x)$  的变化是各自独立的，当  $f(x)$  改变时， $g(x)$  可能不变，也可能改变； $g(x)$  对  $f(x)$  也是如此。
3. 对于所有数据点， $n$  和  $N$  都增大了许多。如果复杂度涉及到  $n$ ，则最多预计为  $O(n \log n)$  级别；如果涉及到  $N$ ，则必须是亚线性级别。

### 2.3.2 70% 数据——计算出每个 $f(x), g(x)$ 的值

#### 2.3.2.1 思路

由于 1,2 条限制，我们无法直接对  $f(x), g(x)$  分别进行处理。但我们可以求出每个  $f(x), g(x)$  的值，再计算求和即可。

$f(x)$  的计算同第一问，任意方法皆可。单个  $g(x)$  的值可以直接  $O(1)$  求得。

#### 2.3.2.2 C++ 实现

待补充


### 2.3.3 100% 数据——对 $f(x), g(x)$ 都相同的区间进行求和处理

#### 2.3.3.1 思路

注：为了防止混淆，将题目中的  $r$  改为  $ratio$ 。

假设  $f(x)$  一共有  $x$  种取值， $g(x)$  一共有  $y$  种取值。直接来看  $f(x), g(x)$  的组合一共有  $xy$  种，但注意到  $f(x), g(x)$  都是单调不递减函数，所以真正的组合只有  $x + y$  种。

在第一题中已经说明  $f(x)$  的取值范围为  $[0, n]$ ，在  $O(n)$  级别。考虑  $g(x)$  的取值情况，将  $ratio$  的公式带入可以得到  $g(x) = \lfloor \frac{x}{ratio} \rfloor = \lfloor \frac{x}{\frac{N}{n+1}} \rfloor$ 。由于  $x$  取值有  $N$  种，所以  $g(x)$  的取值是  $O(\frac{N}{\frac{N}{n+1}}) = O(n)$  级别的。所以，整体复杂度为  $O(n + n) = O(n)$ 。

 **提示** 有些时候，题目给出的某些量的值会比较特殊（如本题  $ratio = \lfloor \frac{N}{n+1} \rfloor$ ），代表着出题人可能想要隐藏某些做法，但不得不为了让时间复杂度正确而妥协。在没有思路的时候，可以作为突破口。

考虑范围问题：假设当前左端点为  $l$ ，如何找到右端点  $r$ ，满足  $f(l) = f(l+1) = \dots = f(r), g(l) = g(l+1) = \dots = g(r)$  且  $f(l) \neq f(r+1)$  or  $g(l) \neq g(r+1)$ 。我们可以对  $f(x), g(x)$  分别考虑：

1. 对于  $f(x)$  而言，第一个满足  $f(x) = f(l) + 1$  的  $x$  值为  $A_{f_l+1}$ 。
2. 对于  $g(x)$  而言，因为分母  $ratio$  是固定的，所以值相同的区间长度也是固定为  $ratio$ 。我们不妨将  $g(x)$  值相同的数字为一组，则可以得到  $[0, ratio - 1], [ratio, 2 \cdot ratio - 1], \dots, [n \cdot ratio, (n+1) \cdot ratio - 1], \dots$  这样的分组序列，每组的  $g(x)$  取值为  $0, 1, \dots, n, \dots$ 。可以发现，对于一个数  $l$ ，其所属的分组是  $\lfloor \frac{l}{ratio} \rfloor$ ，也即  $g(l)$ ；而下一组开始的第一个数为  $ratio \cdot (g(l) + 1)$ ，从而可以得到右端点  $r = ratio \cdot (g(l) + 1) - 1$ 。
3. 在  $f(x), g(x)$  计算得到的右端点中，选择较小的一个作为计算的右端点。

计算完一段后，设  $l = r + 1$  继续计算下一段，直到结束。时间复杂度  $O(n)$ 。

#### 2.3.3.2 C++ 实现

```
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
```

```

using namespace std;
#define ll long long
const int maxn = 100010;
int n, N;
int a[maxn];
int rat, f, g;
ll ans = 0;
int main() {
    scanf("%d%d", &n, &N);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &a[i]);
    }
    rat = N / (n + 1); // 为了防止冲突，题目中 r 改为 rat
    int cur = 0; // 用来计算 f(x)
    bool flag = false; // 如果需要更新 f(x) 值，则 flag = true
    for (int l = 0, r; l < N; l = r + 1) {
        flag = false;
        // 利用 f(x) 的值确定 r 的范围
        if (cur < n)
            r = a[cur + 1] - 1;
        else
            r = N - 1;
        // 判断 f(x), g(x) 谁先变化，选择较小的区间
        if ((l / rat + 1) * rat - 1 < r) {
            // 如果 g(x) 先变化，则改为选择 g(x)
            r = (l / rat + 1) * rat - 1;
        } else {
            // 如果 f(x) 先变化，则确定选择 f(x)，计算后更新 f(x)
            flag = true;
        }
        // [l, r] 区间内的值是相等的，可以求和
        ans += 1ll * (r - l + 1) * abs(l / rat - cur);
        // 更新 f(x) 的值
        if (flag)
            ++cur;
    }
    printf("%lld\n", ans);
    return 0;
}

```

### 2.3.4 100% 思路——以 $f(x)$ 为单位，讨论内部 $g(x)$ 求和

感谢 DoctorLazy 提供的宝贵思路，原文可以查看 [第二题 100 分题解 by DoctorLazy.md](#)。

#### 2.3.4.1 思路

和上文同样的思路，我们需要进行区间求和来降低复杂度。一种思路是，整体上对  $f(x)$  进行求和，而在内部对  $g(x)$  的情况进行分类讨论。

我们单独考虑每一个  $f(x)$  的区间，每个区间上  $f(x)$  的值相同。可以观察到，对于一个区间上的下标  $i$ ，可能存在  $g(i) \geq f(i)$ ，也可能存在  $g(i) < f(i)$ 。求绝对值时，前者用  $g(x) - f(x)$ ，后者用  $f(x) - g(x)$ 。

观察到，由于  $g(x)$  单调不减的性质，我们可以得到：对于该区间，一定存在一个下标  $p$ ，如同一个分界线，当  $i \geq p$  时，有  $g(i) \geq f(i)$ ，当  $i < p$ ，有  $g(i) < f(i)$ 。这样，就把该区间分成了两个“小区间”。我们就可以用“乘法思想”来加速两个“小区间”的求解了。

更规范些，用  $contribution(i)$  代表区间  $[A_i, A_{i+1})$  对答案的贡献，用  $len(l, r) = r - l + 1$  代表区间长度，用公式可以表达为：

$$contribution(i) = len(A_i, p - 1) \times f(x) - \sum_{x=A[i]}^{p-1} g(x) \\ + \sum_{x=p}^{A_{i+1}-1} g(x) - len(p, A_{i+1} - 1) \times f(x)$$

上式中， $f(x)$  是一个常数，所以乘以“小区间”的长度即可； $g(x)$  的求和，大家可以发挥数学思维：因为  $g(x)$  其实非常规律，它的每一块是定长的，我们可以通过除法和取余来确定相同值的数量，再利用乘法思想求和，灵活实现，在  $O(n)$  时间内求出即可。 $p$  的具体值可以通过在  $g(x)$  中二分查找， $O(\log n)$  时间内求出， $n$  为区间的长度。

一个例子：

$x$	...	4	5	6	7	...
$f(x)$	...	2	2	2	2	...
$g(x)$	...	1	1	2	2	...

上面的表格截取了一个小区间， $f(x)$  的值固定 2， $p = 6$ ，那么  $p$  的左边用  $f(x) - g(x)$ ， $p$  的右边用  $g(x) - f(x)$ 。

当然，有一个特殊的边界情况，那就是该区间上有可能所有的  $g(x)$  都绝对大于或小于  $f(x)$ ，这时候  $p$  可能会在区间外。该情况大家可以对  $p$  设置初值，然后在写完二分后加以判断即可。

### 2.3.4.2 C++ 实现

```
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <map>
#include <queue>
#include <vector>

using namespace std;
typedef long long LL;
typedef unsigned long long uLL;
typedef pair<int, int> pii;
const int mod = 1e9 + 7;
const int maxn = 1e5 + 5;

LL N, n;
```

```

LL arr[maxn]; // 题中 A 数组
vector<LL> f; // 存储每个区间上f的值
vector<pii> pos; // 存储每个区间的边界，是左闭右闭
LL r, ans; // 题中的 r, ans为计算的答案

// 下面的函数用于计算g(x)在区间上的和
// 这一步比较细，具体可以灵活实现
// 下面的思路还是比较冗杂的
LL totG(LL be, LL ed) {
    // 右边界小于左边界，返回0
    if (ed < be) {
        return 0;
    }
    // 两边界重合，返回一个g值
    if (be == ed) {
        return be / r;
    }
    // 如果两边界g值相同，返回该值乘以区间长度
    if (be / r == ed / r) {
        return (be / r) * (ed - be + 1);
    }
    // 将区间分为三部分，分别累计
    LL tot = 0;
    // 对于左边界，其值为be/r,数目为 r - be % r
    tot += (r - (be % r)) * (be / r);
    // 对于右边界，其值为ed/r, 数目为 ed % r + 1
    tot += (ed % r + 1) * (ed / r);
    // 对于不在边界上的g值，我们用等差数列求和公式
    if (ed / r - be / r > 1) {
        be = be / r + 1;
        ed = ed / r - 1;
        tot += r * ((be + ed) * (ed - be + 1) / 2);
    }
    return tot;
}

void solve() {
    // 输入
    scanf("%lld%lld", &n, &N);
    r = N / (n + 1);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &arr[i]);
    }
    // 根据数组，生成f(x)的每个区间，值存入f，区间边界存入pos
    LL last = 0; // 记录上一个边界
    // 这里的逻辑参考第一题
    for (int i = 1; i <= n; i++) {
        if (arr[i] > arr[last]) {
            f.push_back(last);

```

```

        pos.push_back({arr[last], arr[i] - 1});
        last = i;
    }
}
// 单独处理下最后一个区间，即[A[n],N-1]
f.push_back(n);
pos.push_back({arr[last], N - 1});
// 对于每个f区间，将g分成两个小区间
int si = f.size();
for (int i = 0; i < si; i++) {
    LL num = f[i];
    LL be = pos[i].first;
    LL ed = pos[i].second;
    LL length = ed - be + 1;
    // 因为be和ed在二分过程其值发生变化，所以下面再存一份
    LL bbe = be, eed = ed;
    // 下面使用二分，在g中寻找分界p
    LL pin = -1;
    while (be <= ed) {
        LL mid = (be + ed) / 2;
        LL cur = mid / r;
        if (cur >= num) {
            pin = mid;
            ed = mid - 1;
        } else {
            be = mid + 1;
        }
    }
    // 如果f的值一直大于g，p值不会被二分的过程赋值，所以还是初值
    if (pin == -1) {
        ans += num * length - totG(bbe, eed);
    } else {
        // 左边的用f-g，右边用g-f。就算g的值一直大于f，即左边的部分长度为0
        ans += num * (pin - bbe) - totG(bbe, pin - 1);
        ans += totG(pin, eed) - num * (eed - pin + 1);
    }
}
}
printf("%lld", ans);
}

int main() {
    int t;
    t = 1;
    while (t--) {
        solve();
    }
    return 0;
}

```



## 2.4 202112-3 登机牌条码

### 题目背景

西西艾弗岛景色优美，游人如织。但是，由于和外界的交通只能靠渡船，交通的不便严重制约了岛上旅游业的发展。西西艾弗岛管委会经过努力，争取到了一笔投资，建设了一个通用航空机场。在三年紧锣密鼓的主体建设后，西西艾弗岛通用航空机场终于开始进行航站楼内部软硬件系统的安装和调试工程了。小 C 是机场运营公司信息部的研发工程师，最近，信息部门的一项重要任务是，研发登机牌自助打印系统。如图所示的是设计部门根据国际民航组织的行业标准设计的登机牌样张。

 西西艾弗岛通航机场

登机牌 BOARDING PASS

 西西艾弗

姓名 NAME DU/XIAOXI

航班号 FLIGHT CP 024    日期 DATE DEC 05    到达站 DEST PKX

登机时间 BOD TIME 1330    登机口 GATE A5    座位号 SEAT 45A    舱位 CLASS Y



登机口于起飞前 10 分钟关闭 GATES CLOSE 10 MINUTES BEFORE DEPARTURE TIME

航班号 FLIGHT CP 024

日期 DATE DEC 05

到达站 DEST PKX

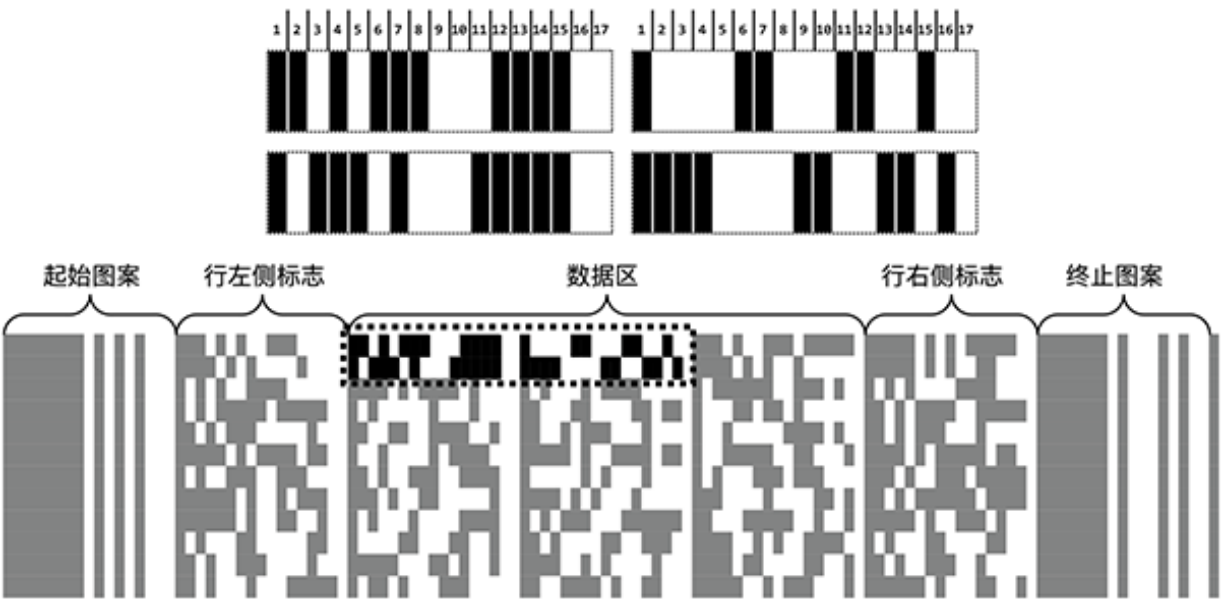
姓名 NAME DU/XIAOXI

座位号 SEAT 45A

登机牌上最重要的部分就是最下方的机读条形码了。小 C 承担了生成机读条形码算法的开发工作。从被编码的数据到条形码，中间有好多步骤要走。小 C 请你来帮忙，让你帮忙处理一下数据编码的问题。

### 题目描述

登机牌上的条形码，是 PDF417 码。PDF417 码的结构如下图所示。



PDF417 码组成的基本元素是码元 (Module)，所有的码元都是等大的矩形，填充有黑色或白色。码元先组成行，若干行堆叠组成整个 PDF417 码。每一行中，每 17 个码元表示一个码字 (Code word)。码字是 PDF417 编码中的最小数据单位。每个码字图案中，有交替排列的四个黑色矩形和四个白色矩形，这便是“417”的由来。每行开始和结尾有固定的起始和中止图案。与他们相邻的是行左侧和右侧标志，表示行号、行内码字个数等信息。中间的是有效数据区。编码的步骤是：先按照编码规则，将被编码的数据转换为码字；接着根据选定 PDF417 码的宽度（即每行码字的数目）以及冗余程度计算校验码字；最后将码字按规则转换为对应的图案，并按照从左至右，从上至下的顺序填入有效数据区，并与起始终止图案和行左右标志拼合，形成完整的 PDF417 码。

每个码字是一个 0 至 928 之间的数字，每个码字可以编码两个输入字符。对于输入的被编码的数据，按照下表进行编码。编码器共有三种模式：大写字母模式、小写字母模式和数字模式。在编码开始时，编码器处于大写字母模式。编码器处于某种模式时，仅能编码对应类型的字符，如果需要编码其它类型的字符，需要通过特殊值切换到对应模式下。要进行模式切换，可以有多种切换方法。例如，要从大写模式切入小写模式，可以直接用 27 切入，也可以先用 28 切入数字模式后立刻再用 27 切入小写模式。你需要选择最短的方式进行切换，因此只有前一种方法是正确的。需要注意的是，从小写模式不能直接切入大写模式，必须要经过数字模式过渡。

值	大写模式	小写模式	数字模式
0	A	a	0
1	B	b	1
2	C	c	2
3	D	d	3
4	E	e	4
5	F	f	5
6	G	g	6
7	H	h	7
8	I	i	8
9	J	j	9
10	K	k	
11	L	l	
12	M	m	
13	N	n	
14	O	o	
15	P	p	
16	Q	q	
17	R	r	
18	S	s	
19	T	t	
20	U	u	
21	V	v	
22	W	w	
23	X	x	
24	Y	y	
25	Z	z	
27	小写		小写
28	数字	数字	大写
29	填充	填充	填充

按照这个方法可以得到一系列的不超过 30 的数字。如果有奇数个这样的数字，则在最后补充一个 29，使之成为偶数个。将它们两两成组，假设  $H$  和  $L$  是一组中连续出现的两个数字，那么可以得到一个码字是：

$$30 \times H + L$$

例如，要编码 “HEllo”，首先根据字母表，产生数字序列：

H E    1    1    o  
7 4 28 1 27 11 14

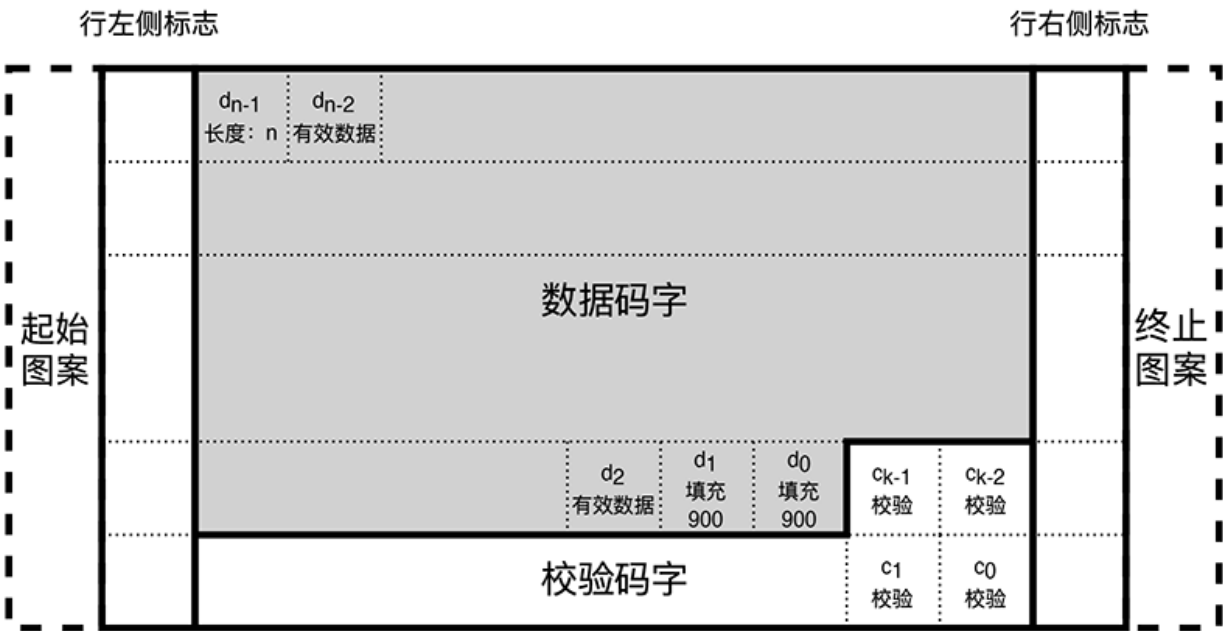
由于只有奇数个数字，需要在末尾补充 29，然后将它们两两成组：

(7, 4), (28, 1), (27, 11), (14, 29)

最后计算码字，例如： $30 \times 7 + 4 = 214$ ，以此类推，可以得到码字为：

214, 841, 821, 449

接下来要计算校验码。校验码字的数目，由校验级别确定。假设校验级别为  $s(0 \leq s \leq 8)$ ，则校验码字的数目为  $k = 2^{s+1}$ 。特别地，如果指定了  $s = -1$ ，则表示不需要计算校验码字。要计算校验码字，首先要确定数据码字。数据码字由以下数据按顺序拼接而成（如图所示）：



- 一个长度码字，表示全部数据码字的个数  $n$ ，包括该长度码字、有效数据码字、填充码字；
- 若干有效数据码字，是此前计算的码字序列；
- 零个或多个由重复的 900 组成的填充码字，使得包括校验码字在内的码字总数恰能被有效数据区的行宽度整除。

设全部数据码字依次为  $d_{n-1}, d_{n-2}, \dots, d_0$ ；校验码字依次为  $c_{k-1}, c_{k-2}, \dots, c_0$ 。那么校验码字按照如下方式计算：

取  $k$  次多项式  $g(x) = (x - 3)(x - 3^2) \cdots (x - 3^k)$ ， $(n - 1)$  次多项式  $d(x) = d_{n-1}x^{n-1} + d_{n-2}x^{n-2} + \cdots + d_1x + d_0$ ，找到多项式  $q(x)$  和不超过  $k - 1$  次的多项式  $r(x)$ ，使得

$$x^k d(x) \equiv q(x)g(x) - r(x)$$

那么多项式  $r(x)$  中  $x$  的  $i$  次项系数对 929 取模后（取正值）的数字即为校验码字  $c_i$ 。

例如，如果要将 `HEllo` 编码为 PDF417 条码，且有效数据区的行宽是 4 码字（即 68 码元），校验级别为 0。此时校验码字有两个。根据此前的编码结果，有效数据码字有 4 个。再加上一个长度码字，共有 7 个码字。因此需要补充一个填充码字，使包括校验码字在内的总码字数能够被 4 整除。这样，用于计算校验码字的数据码字有 6 个，分别是：

6, 214, 841, 821, 449, 900

因此有  $g(x) = x^2 - 12x + 27$ ， $d(x) = 6x^5 + 214x^4 + 841x^3 + 821x^2 + 449x + 900$ ，不难得到  $r(x) = -32902164x + 98246277$ ，因此相应可以计算出  $c_1 = 299 \equiv -32902164 \pmod{929}$ ， $c_0 = 811 \equiv 98246277 \pmod{929}$ 。这样，全部码字序列即为：

6, 214, 841, 821, 449, 900, 229, 811

在本题中，你需要帮助小 C 完成的任务是，给定被编码的数据，计算出需要填入有效数据区的码字序列。被处理的数据中只含有大写字母、小写字母和数字。

## 输入格式

从标准输入读入数据。

输入的第一行包含两个用空格分隔的整数  $w$ 、 $s$ ，分别表示有效数据区每行能容纳的码字数和校验级别。保证  $0 < w < 929$ ， $-1 \leq s \leq 8$ 。特别地，当  $s = -1$  时，表示不需要计算校验码字。

输入的第二行是一个非空字符串，仅包含大小写字母和数字，长度保证编码后全部数据码字的个数少于 929。

## 输出格式

输出到标准输出。

输出若干行，每行一个数字，表示编码后的全部码字序列。

## 样例

输入 #1：

5 -1  
HELLO

输出 #1：

5  
214  
341  
449  
900

解释 #1：

要求编码数据是 `HELLO`，首先查表将其对应成数字。注意，由于编码器在开始时就处于大写字母模式，因此不需要额外的模式切换。因此对应成的数字为：7, 4, 11, 11, 14。由于只有奇数个数字，因此补充 29，形成序列 7, 4, 11, 11, 14, 29。然后两两成组计算码字： $7 \times 30 + 4 = 214$ ，以此类推，得到 214, 341, 449。本输入不要求产生校验码，且有效数据区的宽度是 5 码字。目前有效数据的码字是 3 个，加上开头要添加的长度码字，共有 4 个码字。因此，需要补充一个填充码字，使得总码字数达到 5 个，充满一行。注意，长度码字中的长度数据包括所有数据码字，因此长度码字是 5 而不是 4。最终可以得到码字序列 5, 214, 341, 449, 900。

输入 #2：

```
4 0
HE11o
```

输出 #2:

```
6
214
841
821
449
900
229
811
```

解释 #2:

本组数据即为此前用于说明编码过程的示例。

## 子任务

对于 20% 的数据，有  $s = -1$ ，且输入字符串中仅含有大写字母或小写字母；

对于 40% 的数据，有  $s = -1$ ；

对于 80% 的数据，有  $s \leq 2$ ；

对于 100% 的数据，满足全部对于输入的要求。

### 2.4.1 40% 数据——直接模拟

#### 2.4.1.1 思路

这一部分数据满足  $s = -1$ ，即校验码为空。我们按照题目要求进行对应操作即可，大体分为以下几个步骤：

1. 得到数字序列，注意不同模式的切换以及最后的补全。
2. 将得到的数字转换为码字。
3. 根据有效数据区每行能容纳的码字数  $w$  及目前码字数，在末尾补充码字。注意不要忽略长度码字。
4. 输出结果。

#### 2.4.1.2 C++ 实现

```
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <vector>
using namespace std;
const int maxn = 210;
const int mod = 929;
int w, lev;
char s[100010];
int n;
vector<int> numberList; // 数字序列
```

```

vector<int> codeWord; // 码字序列
int currentMode; // 目前编码器模式。0:大写模式 1:小写模式 2:数字模式
void checkmode(char c) {
    /*
        检查将要输出的下个字符与目前模式是否匹配，
        若不匹配，则输出对应更改模式步骤。
    */
    if (c >= '0' && c <= '9') {
        if (currentMode != 2) {
            numberList.push_back(28);
            currentMode = 2;
        }
    } else if (c >= 'a' && c <= 'z') {
        if (currentMode != 1) {
            numberList.push_back(27);
            currentMode = 1;
        }
    } else if (c >= 'A' && c <= 'Z') {
        if (currentMode == 1) {
            numberList.push_back(28);
            numberList.push_back(28);
            currentMode = 0;
        }
        if (currentMode == 2) {
            numberList.push_back(28);
            currentMode = 0;
        }
    }
}

int main() {
    scanf("%d%d", &w, &lev); // lev 表示校验级别
    scanf("%s", s);
    n = strlen(s);
    // 步骤一：得到数字序列
    currentMode = 0; // 初始为大写模式
    for (int i = 0; i < n; ++i) {
        checkmode(s[i]);
        if (s[i] >= '0' && s[i] <= '9') {
            numberList.push_back(s[i] - '0');
        } else if (s[i] >= 'a' && s[i] <= 'z') {
            numberList.push_back(s[i] - 'a');
        } else if (s[i] >= 'A' && s[i] <= 'Z') {
            numberList.push_back(s[i] - 'A');
        }
    }
    if (numberList.size() % 2)
        numberList.push_back(29);
    // 步骤二：转换为码字
    for (int i = 0; i < numberList.size(); i += 2) {

```

```

        codeWord.push_back(30 * numberList[i] + numberList[i + 1]);
    }
    // 步骤三：补充码字
    while ((1 + codeWord.size()) % w != 0) {
        codeWord.push_back(900);
    }
    // 步骤四：输出结果
    codeWord.insert(codeWord.begin(), codeWord.size() + 1);
    for (int i = 0; i < codeWord.size(); ++i) {
        printf("%d\\n", codeWord[i]);
    }
    return 0;
}

```

## 2.4.2 100% 数据——模拟 + 多项式除法

### 2.4.2.1 思路

这部分数据要求我们对校验码进行处理，所以步骤变为：

1. 得到数字序列，注意不同模式的切换以及最后的补全。
2. 将得到的数字转换为码字。
3. 根据有效数据区每行能容纳的码字数  $w$ 、目前码字个数以及校验码的位数，在末尾补充码字。注意不要忽略长度码字。
4. 输出数据码部分结果。
5. 计算得出校验码，并输出。

校验码的位数能比较方便得出，关键在于校验码的计算。考虑关键公式：

$$x^k d(x) \equiv q(x)g(x) - r(x)$$

其中  $d(x)$  是  $n-1$  次多项式（已知）， $g(x)$  是  $k$  次多项式（已知），未知项有  $q(x), r(x)$ ，其中  $r(x)$  为所求。考虑消去  $q(x)$  的影响：可以在两端同时对  $g(x)$  取余，则  $q(x)g(x)$  项会被直接消去，可以化所求式为：

$$x^k d(x) \equiv -r(x) \pmod{q(x)}$$

所以目前问题转化为求解  $x^k d(x) \pmod{q(x)}$ 。

#### 定义 2.1 (多项式带余除法)

若  $f(x)$  和  $g(x)$  是两个多项式，且  $g(x)$  不等于 0，则存在唯一的多项式  $q(x)$  和  $r(x)$ ，满足：

$$f(x) = q(x)g(x) + r(x)$$

其中  $r(x)$  的次数小于  $g(x)$  的次数。此时  $q(x)$  称为  $g(x)$  除  $f(x)$  的商式， $r(x)$  称为余式。

#### 定义 2.2 (多项式长除法)

求解多项式带余除法的一种方法，步骤如下：

1. 把被除式、除式按某个字母作降幂排列，并把所缺的项用零补齐；
2. 用被除式的第一项除以除式第一项，得到商式的第一项；

3. 用商式的第一项去乘除式，把积写在被除式下面（同类项对齐），消去相等项，把不相等的项结合起来；
4. 把减得的差当作新的被除式，再按照上面的方法继续演算，直到余式为零或余式的次数低于除式的次数时为止。

下面展示的是一个多项式长除法的例子：

$$\begin{array}{r}
 \phantom{x^2 - 12x + 27) } 6x^5 \phantom{+ 286x^4} \phantom{+ 4111x^3} \phantom{+ 42431x^2} \phantom{+ 398624x} \phantom{+ 3638751} \\
 \hline
 x^2 - 12x + 27) \phantom{6x^5} 6x^7 + 214x^6 + 841x^5 + 821x^4 + 449x^3 + 900x^2 \\
 \phantom{x^2 - 12x + 27) } - 6x^7 + 72x^6 - 162x^5 \\
 \hline
 \phantom{x^2 - 12x + 27) } \phantom{6x^7} 286x^6 + 679x^5 + 821x^4 \\
 \phantom{x^2 - 12x + 27) } \phantom{6x^7} - 286x^6 + 3432x^5 - 7722x^4 \\
 \hline
 \phantom{x^2 - 12x + 27) } \phantom{6x^7} \phantom{286x^6} 4111x^5 - 6901x^4 + 449x^3 \\
 \phantom{x^2 - 12x + 27) } \phantom{6x^7} \phantom{286x^6} - 4111x^5 + 49332x^4 - 110997x^3 \\
 \hline
 \phantom{x^2 - 12x + 27) } \phantom{6x^7} \phantom{286x^6} \phantom{4111x^5} 42431x^4 - 110548x^3 + 900x^2 \\
 \phantom{x^2 - 12x + 27) } \phantom{6x^7} \phantom{286x^6} \phantom{4111x^5} - 42431x^4 + 509172x^3 - 1145637x^2 \\
 \hline
 \phantom{x^2 - 12x + 27) } \phantom{6x^7} \phantom{286x^6} \phantom{4111x^5} \phantom{42431x^4} 398624x^3 - 1144737x^2 \\
 \phantom{x^2 - 12x + 27) } \phantom{6x^7} \phantom{286x^6} \phantom{4111x^5} \phantom{42431x^4} - 398624x^3 + 4783488x^2 - 10762848x \\
 \hline
 \phantom{x^2 - 12x + 27) } \phantom{6x^7} \phantom{286x^6} \phantom{4111x^5} \phantom{42431x^4} \phantom{398624x^3} 3638751x^2 - 10762848x \\
 \phantom{x^2 - 12x + 27) } \phantom{6x^7} \phantom{286x^6} \phantom{4111x^5} \phantom{42431x^4} \phantom{398624x^3} - 3638751x^2 + 43665012x - 98246277 \\
 \hline
 \phantom{x^2 - 12x + 27) } \phantom{6x^7} \phantom{286x^6} \phantom{4111x^5} \phantom{42431x^4} \phantom{398624x^3} \phantom{3638751x^2} 32902164x - 98246277
 \end{array}$$

得到求解多项式带余除法的步骤后，考虑求解  $r(x)$  的步骤：

1. 计算  $g(x) = (x-3)(x-3^2)\cdots(x-3^k)$ ；
2. 计算  $x^k d(x)$ ；
3. 计算  $x^k d(x) \bmod g(x)$ ，得到  $-r(x)$ ；
4. 对得到的每一项取反即可得到  $r(x)$ 。

计算  $g(x)$ ：考虑到每一次多项式乘以的因子都是  $(x-a)$  的格式，所以可以把  $A(x-a)$  的多项式相乘转化为  $xA - aA$  的格式。 $xA$  可以通过整体移项实现：在移项后，原本在  $x^i$  的系数成为  $x^{i+1}$  的系数，所以可以在一个数组上，从低位到高位依次计算，得到结果。

计算  $x^k d(x)$ ：这部分比较简单，将低  $k$  位的系数赋 0，再将已计算出的数据位放入对应位置即可。

计算  $x^k d(x) \bmod g(x)$ ：利用上文提到的多项式长除法即可。本题  $g(x)$  的最高位系数恒为 1，简化了计算。

### 2.4.2.2 C++ 实现

```

#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <vector>
using namespace std;
const int maxn = 210;
const int mod = 929;
int w, lev;
char s[100010];

```



```

int n;
vector<int> numberList;
vector<int> codeWord;
int verifyCodeLen; // 校验码长度
int currentMode;
void checkmode(char c) {
    /*
        检查将要输出的下个字符与目前模式是否匹配，
        若不匹配，则输出对应更改模式步骤。
    */
    if (c >= '0' && c <= '9') {
        if (currentMode != 2) {
            numberList.push_back(28);
            currentMode = 2;
        }
    } else if (c >= 'a' && c <= 'z') {
        if (currentMode != 1) {
            numberList.push_back(27);
            currentMode = 1;
        }
    } else if (c >= 'A' && c <= 'Z') {
        if (currentMode == 1) {
            numberList.push_back(28);
            numberList.push_back(28);
            currentMode = 0;
        }
        if (currentMode == 2) {
            numberList.push_back(28);
            currentMode = 0;
        }
    }
}

vector<int> get_gx(int k) {
    // 根据 k 计算 g(x)
    vector<int> res;
    res.push_back(mod - 3);
    res.push_back(1);
    int a0 = 3;
    for (int i = 2; i <= k; ++i) {
        a0 = (a0 * 3) % mod;
        res.insert(res.begin(), 0); // 在最低位插入 1，即整体次数 +1
        for (int j = 0; j < i; ++j) {
            res[j] = (res[j] - (a0 * res[j + 1]) % mod + mod) % mod;
        }
    }
    return res;
}

void get_verify_code() {

```

```

// 计算校验码并输出
vector<int> tmp;
vector<int> g = get_gx(verifyCodeLen);
// 初始化  $x^{kd}(x)$ 
for (int i = 1; i <= verifyCodeLen; ++i) {
    tmp.push_back(0);
}
for (int i = codeWord.size() - 1; i >= 0; --i) {
    tmp.push_back(codeWord[i]);
}
// 多项式长除法计算结果
for (int i = tmp.size() - 1; i >= verifyCodeLen; --i) {
    int val = tmp[i];
    for (int j = 0; j < g.size(); ++j) {
        tmp[i - j] =
            (tmp[i - j] - (val * g[g.size() - 1 - j]) % mod + mod) % mod;
    }
}
// 将  $-r(x)$  转化为  $r(x)$ 
for (int i = 0; i < verifyCodeLen; ++i) {
    // 注意: 不能直接  $\text{mod} - \text{tmp}[i]$ , 因为  $\text{tmp}[i]$  可能为 0
    tmp[i] = (mod - tmp[i]) % mod;
}
// 输出结果
for (int i = verifyCodeLen - 1; i >= 0; --i) {
    printf("%d\n", tmp[i]);
}
}

int main() {
    scanf("%d%d", &w, &lev);
    scanf("%s", s);
    n = strlen(s);
    // 步骤一: 得到数字序列
    currentMode = 0;
    for (int i = 0; i < n; ++i) {
        checkmode(s[i]);
        if (s[i] >= '0' && s[i] <= '9') {
            numberList.push_back(s[i] - '0');
        } else if (s[i] >= 'a' && s[i] <= 'z') {
            numberList.push_back(s[i] - 'a');
        } else if (s[i] >= 'A' && s[i] <= 'Z') {
            numberList.push_back(s[i] - 'A');
        }
    }
    if (numberList.size() % 2)
        numberList.push_back(29);
    // 步骤二: 转换为码字
    for (int i = 0; i < numberList.size(); i += 2) {
        codeWord.push_back(30 * numberList[i] + numberList[i + 1]);
    }
}

```

```
}
if (lev == -1)
    verifyCodeLen = 0;
else {
    verifyCodeLen = 1;
    for (int i = 0; i <= lev; ++i) {
        verifyCodeLen *= 2;
    }
}
// 步骤三：补充码字
while ((1 + verifyCodeLen + codeWord.size()) % w != 0) {
    codeWord.push_back(900);
}
codeWord.insert(codeWord.begin(), codeWord.size() + 1);
// 步骤四：输出数据码结果
for (int i = 0; i < codeWord.size(); ++i) {
    printf("%d\\n", codeWord[i]);
}
// 步骤五：如果有校验码，则计算并输出
if (verifyCodeLen != 0) {
    get_verify_code();
}
return 0;
}
```

## 2.5 202112-4 磁盘文件操作

### 题目背景

小 C 对计算机运行的原理很感兴趣，经常进行一些研究和实验。

有一天，他在尝试删除一个好几 GB 大小的文件时，惊奇地发现删除操作几乎在一瞬间就完成了！这让他很是纳闷：如果计算机在每次删除文件时都直接在磁盘上把对应的数据抹掉，不是应该要花挺长时间吗？

于是他找来了小 S 和小 P 一起讨论。小 S 说，或许计算机是一个很“懒”的体系，在删除时不会真的去抹除数据吧？而小 P 则更见多识广一些，他当即找来了一个号称能“恢复磁盘数据”的软件，当场把小 C 刚刚删除的文件恢复了！

这让小 C 有了更强的好奇心，于是他们决定设计一个模型来模拟一个磁盘文件的写入、删除及恢复过程。但是在他们生活的西西艾弗岛上没有合适的条件来运行他们的模型，于是他们联系了带着一台算力超强的电脑来西西艾弗岛旅游的你来帮助他们。

### 题目描述

在小 C、小 S 和小 P 设计的模型中，计算机中有  $n$  段程序（编号为  $1 \sim n$ ），它们共享一块大小为  $m$  的磁盘空间（编号为  $1 \sim m$ ），磁盘上的每个位置可以写入一个整数。

最初，磁盘上每个位置上的数都是 0，并不被任何程序占用。

现在，这  $n$  段程序同时执行，在某一时刻，某段程序可能对磁盘数据进行读写等操作。

操作共  $k$  个，按时间先后顺序给出，具体操作如下：

- $0\ id\ l\ r\ x$ ：编号为  $id$  的程序尝试向磁盘空间中  $[l, r]$  位置上每个位置都写入一个整数  $x$ 。
  - 操作执行过程中，程序  $id$  会尝试从最左端  $l$  开始向右顺次写入数据。
  - 对于每个位置，若目前不被任何程序占用，则成功写入整数  $x$ ，并将其视为被程序  $id$  占用；
  - 若该位置目前正被程序  $id$  自己占用，则这次写入的  $x$  可以覆盖之前写入的结果，此后该位置仍被程序  $id$  占用；
  - 直到成功向  $r$  位置写入数据，或遇到第一个正在被其他程序占用的位置为止，此时该操作立刻中断。
- $1\ id\ l\ r$ ：程序  $id$  尝试删除磁盘中  $[l, r]$  位置上的所有数据。
  - 这一操作当且仅当  $[l, r]$  区间内所有位置都正在被程序  $id$  占用时才能成功执行。
  - 执行效果为将其中所有位置都解除占用，即恢复到可以被任意程序写入的状态。但为了便于恢复数据，不会立即将全部位置重新覆盖成 0。
  - 否则，认为此操作执行失败，不进行任何修改。
- $2\ id\ l\ r$ ：程序  $id$  尝试恢复磁盘中  $[l, r]$  位置上的所有数据。
  - 这一操作当且仅当  $[l, r]$  区间内所有位置都未被占用，且上一次被占用是被程序  $id$  占用时才能成功执行。
  - 执行效果为将其中所有位置恢复为被程序  $id$  占用的状态，同时由于之前删除操作并未改变其存储的值，因此本次操作也不需要改变每个位置上的值。
  - 否则，认为此操作执行失败，不进行任何修改。
- $3\ p$ ：尝试读取磁盘中  $p$  位置的数据，返回结果为两个整数。
  - 如果该位置当前正被程序  $id$  占用且存储的值为  $p$ ，返回结果为  $id\ p$ 。
  - 如果该位置当前没有被任何程序占用，返回 0 0。

你需要实现一个程序，帮助小 C、小 S 和小 P 来模拟实现上述过程，并对于每个操作输出操作结果。

### 输入格式

从标准输入读入数据。

第一行：3 个正整数  $n, m, k$ 。

接下来  $k$  行，每行若干个整数描述一个操作，格式如上所述。

## 输出格式

输出到标准输出。

输出共  $k$  行，对于每个操作输出一行。

对于每个写入操作，输出一个整数表示此次操作写入成功的最右位置；特别地如果该操作一个位置也没有写入成功，输出  $-1$ 。

对于每个删除、恢复操作，若该操作成功，输出一个字符串 **OK**，否则输出一个字符串 **FAIL**。

对于每个读取操作，输出两个整数表示此次查询的结果。

## 样例

输入 #1:

```
3 15 12
0 1 1 5 -1
0 2 10 13 2
0 1 4 14 6
1 1 2 8
3 1
3 3
3 14
2 1 3 5
0 3 7 8 -4
2 1 6 8
1 3 6 7
0 2 5 7 3
```

输出 #1:

```
5
13
9
OK
1 -1
0 0
0 0
OK
8
FAIL
FAIL
-1
```

## 子任务

对于 25% 的数据， $n, k \leq 2000, m \leq 10000$ ；

对于另外 15% 的数据，没有删除、恢复操作；

对于另外 20% 的数据，没有恢复操作；

对于另外 15% 的数据， $n = 1$ 。

对于 100% 的数据， $1 \leq n, k \leq 2 \times 10^5, 1 \leq m \leq 10^9, 1 \leq id \leq n, 1 \leq l \leq r \leq m, 1 \leq p \leq m, |x| \leq 10^9$ 。

## 2.6 202112-5 极差路径

### 题目背景

众所周知，西西艾弗岛是一个旅游胜地，但是由于兴建机场，西西艾弗岛最近的财务状况有点紧张。

### 题目描述

为了从游客手中获取更多的经济利润，岛上仅有的三个小学生小 C、小 S 和小 P 建立了  $n$  个景点，编号依次从 1 到  $n$ 。编号为  $i$  的景点是第  $i$  个被修建的。由于越到后期经费越是不足，所以编号更大的景点通常更令人不满意——方便起见，假定编号为  $i$  的景点的令人不满意程度是  $i$ 。

有些景点之间修有双向可通行的道路，但是出于减少经费的考虑，他们只修了能使得所有景点连通的最少数量的道路，从而这些景点和其间的道路形成一棵树的结构。

对于每个游客而言，由于只修了  $n-1$  条道路，所以他只能沿着树上的边参观，并且由于他不可能重复参观一个景点，所以他的游览路径一定是树上的一条简单路径。

现在西西艾弗岛希望制定一些推荐游览路径，但并非所有树上的路径都是合意的，因为这条路径上的景点令人不满意程度的极差可能过大，使游客产生这些景点质量不稳定的错觉。由于最开始的景点和最后的景点令人印象比较深刻，所以游客通常会把游览路径上的景点和这两个景点作比较。因此，最令人不满意的景点不能比这两个景点差太多，最优秀的景点也不能比这两个景点优秀太多。

具体来说，一条从  $x$  到  $y$  的游览路径（记作  $(x, y)$ ）是推荐的，当且仅当下式成立：

$$\min\{x, y - k_1\} \leq \min\{P(x, y)\} \leq \max\{P(x, y)\} \leq \max\{x, y + k_2\}$$

其中  $P(x, y)$  表示一条从  $x$  出发到达  $y$  的简单路径上的点的令人不满意程度的数值的集合（包括  $x$  和  $y$ ，也就是  $x$  到  $y$  的简单路径上的点的编号的集合）， $\min S$  和  $\max S$  分别表示集合  $S$  中的最小和最大值， $k_1, k_2$  是西西艾弗岛经过数次尝试后选取的两个给定的参数，保证  $k_1, k_2 > 0$ 。

特别的，容易验证  $x = y$  时  $(x, y)$  总是推荐的。

现在西西艾弗岛想知道，一共有多少树上的简单路径作为游览路径是被推荐的？这里我们认为  $(x, y)$  和  $(y, x)$  是同一条路径。

### 输入格式

从标准输入读入数据。

第一行输入三个非负整数  $n, k_1, k_2$ 。

接下来  $n-1$  行，每行两个正整数  $x, y$  表示树上的一条边。

### 输出格式

输出到标准输出。

输出一行一个非负整数表示答案。

### 样例

输入 #1:

```
5 0 1
5 4
4 2
```

4 1  
2 3

输出 #1:

12

解释 #1:

容易验证  $(1, 1), (1, 4), (1, 5), (1, 3), (2, 2), (2, 4), (2, 5), (2, 3), (3, 3), (4, 4), (4, 5), (5, 5)$  都是推荐的游览路径, 因此答案是 12。

## 子任务

测试点	$n \leq$	$k_1$	$k_2$	树的形态	堆性质
1	5000	$\leq n$	$\leq n$	$A_3$	无
2	5000	$\leq n$	$\leq n$	$A_3$	无
3	5000	$\leq n$	$\leq n$	$A_3$	无
4	$5 \times 10^5$	$= 0$	$= 0$	$A_1$	有
5	$5 \times 10^5$	$= 0$	$= 0$	$A_1$	无
6	$5 \times 10^5$	$\leq n$	$= 0$	$A_1$	有
7	$5 \times 10^5$	$\leq n$	$= 0$	$A_1$	无
8	$5 \times 10^5$	$\leq n$	$\leq n$	$A_1$	有
9	$5 \times 10^5$	$\leq n$	$\leq n$	$A_1$	无
10	$5 \times 10^5$	$= 0$	$= 0$	$A_2$	无
11	$5 \times 10^5$	$\leq n$	$= 0$	$A_2$	无
12	$5 \times 10^5$	$\leq n$	$\leq n$	$A_2$	无
13	$5 \times 10^5$	$= 0$	$= 0$	$A_3$	有
14	$5 \times 10^5$	$= 0$	$= 0$	$A_3$	无
15	$5 \times 10^5$	$= 0$	$= 0$	$A_3$	无
16	$5 \times 10^5$	$= 0$	$= 0$	$A_3$	无
17	$5 \times 10^5$	$\leq n$	$= 0$	$A_3$	有
18	$5 \times 10^5$	$\leq n$	$= 0$	$A_3$	无
19	$5 \times 10^5$	$\leq n$	$= 0$	$A_3$	无
20	$5 \times 10^5$	$\leq n$	$= 0$	$A_3$	无
21	$5 \times 10^5$	$\leq n$	$\leq n$	$A_3$	有
22	$5 \times 10^5$	$\leq n$	$\leq n$	$A_3$	有
23	$5 \times 10^5$	$\leq n$	$\leq n$	$A_3$	无
24	$5 \times 10^5$	$\leq n$	$\leq n$	$A_3$	无
25	$5 \times 10^5$	$\leq n$	$\leq n$	$A_3$	无

对于 100% 的数据,  $1 \leq n \leq 5 \times 10^5, 0 \leq k_1, k_2 \leq n$ , 保证输入的  $n - 1$  条边一定构成一棵树。

树的形态:

- $A_1$ : 树是一条链;
- $A_2$ : 存在一个度数为  $n - 1$  的点;
- $A_3$ : 树的形态无特殊性质。

堆性质: 若取编号为 1 的点为根, 则除 1 号点外, 每个点的编号都比其父节点的编号大。



## 提示

请注意答案可能的取值范围。