Codeforces Round #490 (Div. 3)

A. Mishka and Contest

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Mishka started participating in a programming contest. There are $$$n$$$ problems in the contest. Mishka's problem-solving skill is equal to $$$k$$$.

Mishka arranges all problems from the contest into a list. Because of his weird principles, Mishka only solves problems from one of the ends of the list. Every time, he chooses which end (left or right) he will solve the next problem from. Thus, each problem Mishka solves is either the leftmost or the rightmost problem in the list.

Mishka cannot solve a problem with difficulty greater than $$$k$$$. When Mishka solves the problem, it disappears from the list, so the length of the list decreases by $$$1$$$. Mishka stops when he is unable to solve any problem from any end of the list.

How many problems can Mishka solve?

Input

The first line of input contains two integers $$$n$$$ and $$$k$$$ ($$$1 \le n, k \le 100$$$) — the number of problems in the contest and Mishka's problem-solving skill.

The second line of input contains $$$n$$$ integers $$$a_1, a_2, \dots, a_n$$$ ($$$1 \le a_i \le 100$$$), where $$$a_i$$$ is the difficulty of the $$$i$$$-th problem. The problems are given in order from the leftmost to the rightmost in the list.

Output

Print one integer — the maximum number of problems Mishka can solve.

Examples

input
8 4 4 2 3 1 5 1 6 4
output
5

input
5 2 3 1 2 1 3
output
0

input
5 100 12 34 55 43 21
output
5

Note

In the first example, Mishka can solve problems in the following order: $$$[4, 2, 3, 1, 5, 1, 6, 4] \rightarrow [2, 3, 1, 5, 1, 6, 4] \rightarrow [2, 3, 1, 5, 1, 6] \rightarrow [3, 1, 5, 1, 6] \rightarrow [1, 5, 1, 6] \rightarrow [5, 1, 6]$$$, so the number of solved problems will be equal to $$$5$$$.

In the second example, Mishka can't solve any problem because the difficulties of problems from both ends are greater than $$$k$$$.

In the third example, Mishka's solving skill is so amazing that he can solve all the problems.

B. Reversing Encryption

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

A string $$$s$$$ of length $$$n$$$ can be encrypted by the following algorithm:

- iterate over all divisors of $$$n$$$ in decreasing order (i.e. from $$$n$$$ to $$$1$$$),
- for each divisor $$$d$$$, reverse the substring $$$s[1 \dots d]$$$ (i.e. the substring which starts at position $$$1$$$ and ends at position $$$d$$$).

For example, the above algorithm applied to the string $$$s$$$="`codeforces`" leads to the following changes: "`codeforces`" $$$\to$$$ "`secrofedoc`" $$$\to$$$ "`orcesfedoc`" $$$\to$$$ "`rocesfedoc`" $$$\to$$$ "`rocesfedoc`" (obviously, the last reverse operation doesn't change the string because $$$d=1$$$).

You are given the encrypted string $$$t$$$. Your task is to decrypt this string, i.e., to find a string $$$s$$$ such that the above algorithm results in string $$$t$$$. It can be proven that this string $$$s$$$ always exists and is unique.

Input
The first line of input consists of a single integer $$$n$$$ ($$$1 \le n \le 100$$$) — the length of the string $$$t$$$. The second line of input consists of the string $$$t$$$. The length of $$$t$$$ is $$$n$$$, and it consists only of lowercase Latin letters.

Output
Print a string $$$s$$$ such that the above algorithm results in $$$t$$$.

Examples

input
10 rocesfedoc
output
codeforces

input
16 plmaetwoxesisiht
output
thisisexampletwo

input
1 z
output
z

Note
The first example is described in the problem statement.

C. Alphabetic Removals

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a string $$$s$$$ consisting of $$$n$$$ lowercase Latin letters. Polycarp wants to remove exactly $$$k$$$ characters ($$$k \le n$$$) from the string $$$s$$$. Polycarp uses the following algorithm $$$k$$$ times:

- if there is at least one letter '`a`', remove the leftmost occurrence and stop the algorithm, otherwise go to next item;
- if there is at least one letter '`b`', remove the leftmost occurrence and stop the algorithm, otherwise go to next item;
- ...
- remove the leftmost occurrence of the letter '`z`' and stop the algorithm.

This algorithm removes a single letter from the string. Polycarp performs this algorithm exactly $$$k$$$ times, thus removing exactly $$$k$$$ characters.

Help Polycarp find the resulting string.

Input
The first line of input contains two integers $$$n$$$ and $$$k$$$ ($$$1 \le k \le n \le 4 \cdot 10^5$$$) — the length of the string and the number of letters Polycarp will remove.

The second line contains the string $$$s$$$ consisting of $$$n$$$ lowercase Latin letters.

Output
Print the string that will be obtained from $$$s$$$ after Polycarp removes exactly $$$k$$$ letters using the above algorithm $$$k$$$ times.

If the resulting string is empty, print nothing. It is allowed to print nothing or an empty line (line break).

Examples

input

```
15 3
cccaababababccbc
```

output

```
cccbbabaccbc
```

input

```
15 9
cccaababababccbc
```

output

```
cccccc
```

input

```
1 1
u
```

output

D. Equalize the Remainders

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given an array consisting of $$$n$$$ integers $$$a_1, a_2, \dots, a_n$$$, and a positive integer $$$m$$$. It is guaranteed that $$$m$$$ is a divisor of $$$n$$$.

In a single move, you can choose any position $$$i$$$ between $$$1$$$ and $$$n$$$ and increase $$$a_i$$$ by $$$1$$$.

Let's calculate $$$c_r$$$ ($$$0 \le r \le m-1$$$ — the number of elements having remainder $$$r$$$ when divided by $$$m$$$. In other words, for each remainder, let's find the number of corresponding elements in $$$a$$$ with that remainder.

Your task is to change the array in such a way that $$$c_0 = c_1 = \dots = c_{m-1} = \frac{n}{m}$$$.

Find the minimum number of moves to satisfy the above requirement.

Input

The first line of input contains two integers $$$n$$$ and $$$m$$$ ($$$1 \le n \le 2 \cdot 10^5, 1 \le m \le n$$$). It is guaranteed that $$$m$$$ is a divisor of $$$n$$$.

The second line of input contains $$$n$$$ integers $$$a_1, a_2, \dots, a_n$$$ ($$$0 \le a_i \le 10^9$$$), the elements of the array.

Output

In the first line, print a single integer — the minimum number of moves required to satisfy the following condition: for each remainder from $$$0$$$ to $$$m - 1$$$, the number of elements of the array having this remainder equals $$$\frac{n}{m}$$$.

In the second line, print **any** array satisfying the condition and can be obtained from the given array **with the minimum number of moves**. The values of the elements of the resulting array must not exceed $$$10^{18}$$$.

Examples

input

```
6 3
3 2 0 6 10 12
```

output

```
3
3 2 0 7 10 14
```

input

```
4 2
0 1 2 3
```

output

```
0
0 1 2 3
```

E. Reachability from the Capital

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input

output: standard output

There are $$$n$$$ cities and $$$m$$$ roads in Berland. Each road connects a pair of cities. The roads in Berland are one-way.

What is the minimum number of new roads that need to be built to make all the cities reachable from the capital?

New roads will also be one-way.

Input

The first line of input consists of three integers $$$n$$$, $$$m$$$ and $$$s$$$ ($$$1 \le n \le 5000, 0 \le m \le 5000, 1 \le s \le n$$$) — the number of cities, the number of roads and the index of the capital. Cities are indexed from $$$1$$$ to $$$n$$$.

The following $$$m$$$ lines contain roads: road $$$i$$$ is given as a pair of cities $$$u_i$$$, $$$v_i$$$ ($$$1 \le u_i, v_i \le n$$$, $$$u_i \ne v_i$$$). For each pair of cities $$$(u, v)$$$, there can be at most one road from $$$u$$$ to $$$v$$$. Roads in opposite directions between a pair of cities are allowed (i.e. from $$$u$$$ to $$$v$$$ and from $$$v$$$ to $$$u$$$).

Output

Print one integer — the minimum number of extra roads needed to make all the cities reachable from city $$$s$$$. If all the cities are already reachable from $$$s$$$, print `0`.

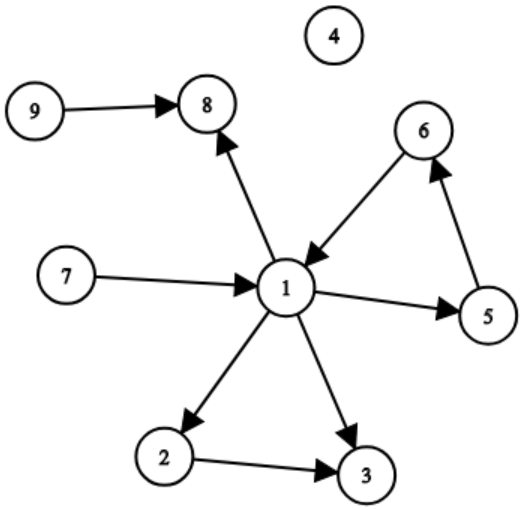
Examples

input
9 9 1 1 2 1 3 2 3 1 5 5 6 6 1 1 8 9 8 7 1
output
3

input
5 4 5 1 2 2 3 3 4 4 1
output
1

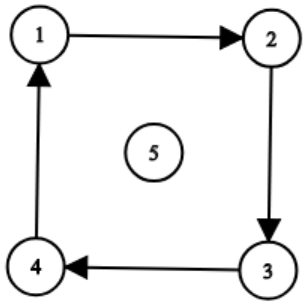
Note

The first example is illustrated by the following:



For example, you can add roads ($$$6, 4$$$), ($$$7, 9$$$), ($$$1, 7$$$) to make all the cities reachable from $$$s = 1$$$.

The second example is illustrated by the following:



In this example, you can add any one of the roads ($$$5, 1$$$), ($$$5, 2$$$), ($$$5, 3$$$), ($$$5, 4$$$) to make all the cities reachable from $$$s = 5$$$.

F. Cards and Joy

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

There are $$$n$$$ players sitting at the card table. Each player has a favorite number. The favorite number of the $$$j$$$-th player is $$$f_j$$$.

There are $$$k \cdot n$$$ cards on the table. Each card contains a single integer: the $$$i$$$-th card contains number $$$c_i$$$. Also, you are given a sequence $$$h_1, h_2, \dots, h_k$$$. Its meaning will be explained below.

The players have to distribute all the cards in such a way that each of them will hold exactly $$$k$$$ cards. After all the cards are distributed, each player counts the number of cards he has that contains his favorite number. The joy level of a player equals $$$h_t$$$ if the player holds $$$t$$$ cards containing his favorite number. If a player gets no cards with his favorite number (i.e., $$$t=0$$$), his joy level is $$$0$$$.

Print the maximum possible total joy levels of the players after the cards are distributed. Note that the sequence $$$h_1, \dots, h_k$$$ is the same for all the players.

Input

The first line of input contains two integers $$$n$$$ and $$$k$$$ ($$$1 \le n \le 500, 1 \le k \le 10$$$) — the number of players and the number of cards each player will get.

The second line contains $$$k \cdot n$$$ integers $$$c_1, c_2, \dots, c_{k \cdot n}$$$ ($$$1 \le c_i \le 10^5$$$) — the numbers written on the cards.

The third line contains $$$n$$$ integers $$$f_1, f_2, \dots, f_n$$$ ($$$1 \le f_j \le 10^5$$$) — the favorite numbers of the players.

The fourth line contains $$$k$$$ integers $$$h_1, h_2, \dots, h_k$$$ ($$$1 \le h_t \le 10^5$$$), where $$$h_t$$$ is the joy level of a player if he gets exactly $$$t$$$ cards with his favorite number written on them. It is guaranteed that the condition $$$h_{t - 1} < h_t$$$ holds for each $$$t \in [2..k]$$$.

Output

Print one integer — the maximum possible total joy levels of the players among all possible card distributions.

Examples

input
4 3 1 3 2 8 5 5 8 2 2 8 5 2 1 2 2 5 2 6 7
output
21

input
3 3 9 9 9 9 9 9 9 9 9 1 2 3 1 2 3
output

```
0
```

Note

In the first example, one possible optimal card distribution is the following:

- Player $$$1$$$ gets cards with numbers $$$[1, 3, 8]$$$;
- Player $$$2$$$ gets cards with numbers $$$[2, 2, 8]$$$;
- Player $$$3$$$ gets cards with numbers $$$[2, 2, 8]$$$;
- Player $$$4$$$ gets cards with numbers $$$[5, 5, 5]$$$.

Thus, the answer is $$$2 + 6 + 6 + 7 = 21$$$.

In the second example, no player can get a card with his favorite number. Thus, the answer is $$$0$$$.