A. Strange Table

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

Polycarp found a rectangular table consisting of n rows and m columns. He noticed that each cell of the table has its number, obtained by the following algorithm **"by columns"**:

- cells are numbered starting from one;
- cells are numbered from left to right by columns, and inside each column from top to bottom;
- number of each cell is an integer one greater than in the previous cell.

For example, if $n = 3$ and $m = 5$, the table will be numbered as follows:

$$\begin{matrix} 1 & 4 & 7 & 10 & 13 \\ 2 & 5 & 8 & 11 & 14 \\ 3 & 6 & 9 & 12 & 15 \end{matrix}$$

However, Polycarp considers such numbering inconvenient. He likes the numbering **"by rows"**:

- cells are numbered starting from one;
- cells are numbered from top to bottom by rows, and inside each row from left to right;
- number of each cell is an integer one greater than the number of the previous cell.

For example, if $n = 3$ and $m = 5$, then Polycarp likes the following table numbering:

$$\begin{matrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{matrix}$$

Polycarp doesn't have much time, so he asks you to find out what would be the cell number in the numbering **"by rows"**, if in the numbering **"by columns"** the cell has the number x?

Input

The first line contains a single integer t ($1 \le t \le 10^4$). Then t test cases follow.

Each test case consists of a single line containing three integers n, m, x ($1 \le n, m \le 10^6$, $1 \le x \le n \cdot m$), where n and m are the number of rows and columns in the table, and x is the cell number.

Note that the numbers in some test cases do not fit into the 32-bit integer type, so you must use at least the 64-bit integer type of your programming language.

Output

For each test case, output the cell number in the numbering **"by rows"**.

Example

input

```
5
1 1 1
2 2 3
3 5 11
100 100 7312
1000000 1000000 1000000000000
```

output

```
1
2
9
1174
1000000000000
```

B. Partial Replacement

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

output: standard output

You are given a number k and a string s of length n, consisting of the characters '`.`' and '*'. You want to replace some of the '*' characters with 'x' characters so that the following conditions are met:

- The first character '*' in the original string should be replaced with 'x';
- The last character '*' in the original string should be replaced with 'x';
- The distance between two neighboring replaced characters 'x' must not exceed k (more formally, if you replaced characters at positions i and j ($i < j$) and at positions $[i+1, j-1]$ there is no "x" symbol, then $j - i$ must be no more than k).

For example, if $n = 7$, $s =$ `.**.***` and $k = 3$, then the following strings will satisfy the conditions above:

- `.xx.*xx`;
- `.x*.x*x`;
- `.xx.xxx`.

But, for example, the following strings will not meet the conditions:

- `.**.*xx` (the first character '*' should be replaced with 'x');
- `.x*.xx*` (the last character '*' should be replaced with 'x');
- `.x*.*xx` (the distance between characters at positions 2 and 6 is greater than $k = 3$).

Given n, k, and s, find the minimum number of '*' characters that must be replaced with 'x' in order to meet the above conditions.

Input

The first line contains one integer t ($1 \le t \le 500$). Then t test cases follow.

The first line of each test case contains two integers n and k ($1 \le k \le n \le 50$).

The second line of each test case contains a string s of length n, consisting of the characters '`.`' and '*'.

It is guaranteed that there is at least one '*' in the string s.

It is guaranteed that the distance between any two neighboring '*' characters does not exceed k.

Output

For each test case output the minimum number of '*' characters that must be replaced with 'x' characters in order to satisfy the conditions above.

Example

input

```
5
7 3
.**.***
5 1
..*..
5 2
*.*.*
3 2
*.*
1 1
*
```

output

```
3
1
3
2
1
```

C. Double-ended Strings

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

You are given the strings a and b, consisting of lowercase Latin letters. You can do any number of the following operations in any order:

- if $|a| > 0$ (the length of the string a is greater than zero), delete the first character of the string a, that is, replace a with $a_2 a_3 \ldots a_n$;
- if $|a| > 0$, delete the last character of the string a, that is, replace a with $a_1 a_2 \ldots a_{n-1}$;
- if $|b| > 0$ (the length of the string b is greater than zero), delete the first character of the string b, that is, replace b with $b_2 b_3 \ldots b_n$;
- if $|b| > 0$, delete the last character of the string b, that is, replace b with $b_1 b_2 \ldots b_{n-1}$.

Note that after each of the operations, the string a or b may become empty.

For example, if $a =$"`hello`" and $b =$"`icpc`", then you can apply the following sequence of operations:

- delete the first character of the string $a \Rightarrow a =$"`ello`" and $b =$"`icpc`";
- delete the first character of the string $b \Rightarrow a =$"`ello`" and $b =$"`cpc`";
- delete the first character of the string $b \Rightarrow a =$"`ello`" and $b =$"`pc`";
- delete the last character of the string $a \Rightarrow a =$"`ell`" and $b =$"`pc`";
- delete the last character of the string $b \Rightarrow a =$"`ell`" and $b =$"`p`".

For the given strings a and b, find the minimum number of operations for which you can make the strings a and b equal. Note that empty strings are also equal.

Input

The first line contains a single integer t ($1 \le t \le 100$). Then t test cases follow.

The first line of each test case contains the string a ($1 \le |a| \le 20$), consisting of lowercase Latin letters.

The second line of each test case contains the string b ($1 \le |b| \le 20$), consisting of lowercase Latin letters.

Output

For each test case, output the minimum number of operations that can make the strings a and b equal.

Example

input

```
5
a
a
abcd
bc
hello
codeforces
hello
helo
dhjakjsnasjhfksafasd
adjsnasjhfksvdafdser
```

output

```
0
2
13
3
20
```

D. Epic Transformation

time limit per test: 2 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

You are given an array a of length n consisting of integers. You can apply the following operation, consisting of several steps, on the array a **zero** or more times:

- you select two **different** numbers in the array a_i and a_j;
- you remove i-th and j-th elements from the array.

For example, if $n = 6$ and $a = [1, 6, 1, 1, 4, 4]$, then you can perform the following sequence of operations:

- select $i = 1, j = 5$. The array a becomes equal to $[6, 1, 1, 4]$;
- select $i = 1, j = 2$. The array a becomes equal to $[1, 4]$.

What can be the minimum size of the array after applying some sequence of operations to it?

Input

The first line contains a single integer t ($1 \le t \le 10^4$). Then t test cases follow.

The first line of each test case contains a single integer n ($1 \le n \le 2 \cdot 10^5$) is length of the array a.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$).

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output the minimum possible size of the array after applying some sequence of operations to it.

Example

input

```
5
6
```

```
1 6 1 1 4 4
2
1 2
2
1 1
5
4 5 4 5 4
6
2 3 2 1 3 1
```

output

```
0
0
2
1
0
```

E. Restoring the Permutation

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

A permutation is a sequence of n integers from 1 to n, in which all numbers occur exactly once. For example, $[1]$, $[3, 5, 2, 1, 4]$, $[1, 3, 2]$ are permutations, and $[2, 3, 2]$, $[4, 3, 1]$, $[0]$ are not.

Polycarp was presented with a permutation p of numbers from 1 to n. However, when Polycarp came home, he noticed that in his pocket, the permutation p had turned into an array q according to the following rule:

- $q_i = \max(p_1, p_2, \ldots, p_i)$.

Now Polycarp wondered what lexicographically minimal and lexicographically maximal permutations could be presented to him.

An array a of length n is lexicographically smaller than an array b of length n if there is an index i ($1 \le i \le n$) such that the first $i - 1$ elements of arrays a and b are the same, and the i-th element of the array a is less than the i-th element of the array b. For example, the array $a = [1, 3, 2, 3]$ is lexicographically smaller than the array $b = [1, 3, 4, 2]$.

For example, if $n = 7$ and $p = [3, 2, 4, 1, 7, 5, 6]$, then $q = [3, 3, 4, 4, 7, 7, 7]$ and the following permutations could have been as p initially:

- $[3, 1, 4, 2, 7, 5, 6]$ (lexicographically minimal permutation);
- $[3, 1, 4, 2, 7, 6, 5]$;
- $[3, 2, 4, 1, 7, 5, 6]$;
- $[3, 2, 4, 1, 7, 6, 5]$ (lexicographically maximum permutation).

For a given array q, find the lexicographically minimal and lexicographically maximal permutations that could have been originally presented to Polycarp.

Input

The first line contains one integer t ($1 \le t \le 10^4$). Then t test cases follow.

The first line of each test case contains one integer n ($1 \le n \le 2 \cdot 10^5$).

The second line of each test case contains n integers $q_1, q_2, \ldots, q_n$ ($1 \le q_i \le n$).

It is guaranteed that the array q was obtained by applying the rule from the statement to some permutation p.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output two lines:

- on the first line output n integers — lexicographically **minimal** permutation that could have been originally presented to Polycarp;
- on the second line print n integers — lexicographically **maximal** permutation that could have been originally presented to Polycarp;

Example

input

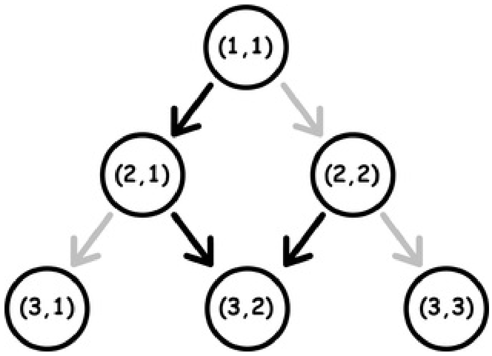
```
4
7
3 3 4 4 7 7 7
4
1 2 3 4
7
3 4 5 5 5 7 7
1
1
```

output
3 1 4 2 7 5 6 3 2 4 1 7 6 5 1 2 3 4 1 2 3 4 3 4 5 1 2 7 6 3 4 5 2 1 7 6 1 1

F. Triangular Paths

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

Consider an infinite triangle made up of layers. Let's number the layers, starting from one, from the top of the triangle (from top to bottom). The k-th layer of the triangle contains k points, numbered from left to right. Each point of an infinite triangle is described by a pair of numbers (r, c) ($1 \le c \le r$), where r is the number of the layer, and c is the number of the point in the layer. From each point (r, c) there are two **directed** edges to the points $(r+1, c)$ and $(r+1, c+1)$, but only one of the edges is activated. If $r+c$ is even, then the edge to the point $(r+1, c)$ is activated, otherwise the edge to the point $(r+1, c+1)$ is activated. Look at the picture for a better understanding.



Activated edges are colored in black. Non-activated edges are colored in gray.

From the point (r_1, c_1) it is possible to reach the point (r_2, c_2), if there is a path between them only from **activated** edges. For example, in the picture above, there is a path from $(1, 1)$ to $(3, 2)$, but there is no path from $(2, 1)$ to $(1, 1)$.

Initially, you are at the point $(1, 1)$. For each turn, you can:

- Replace activated edge for point (r, c). That is if the edge to the point $(r+1, c)$ is activated, then **instead of it**, the edge to the point $(r+1, c+1)$ becomes activated, otherwise if the edge to the point $(r+1, c+1)$, then **instead if it**, the edge to the point $(r+1, c)$ becomes activated. This action increases the cost of the path by 1;
- Move from the current point to another by following the activated edge. This action **does not increase** the cost of the path.

You are given a sequence of n points of an infinite triangle $(r_1, c_1), (r_2, c_2), \ldots, (r_n, c_n)$. Find the minimum cost path from $(1, 1)$, passing through all n points in **arbitrary** order.

Input

The first line contains one integer t ($1 \le t \le 10^4$) is the number of test cases. Then t test cases follow.

Each test case begins with a line containing one integer n ($1 \le n \le 2 \cdot 10^5$) is the number of points to visit.

The second line contains n numbers $r_1, r_2, \ldots, r_n$ ($1 \le r_i \le 10^9$), where r_i is the number of the layer in which i-th point is located.

The third line contains n numbers $c_1, c_2, \ldots, c_n$ ($1 \le c_i \le r_i$), where c_i is the number of the i-th point in the r_i layer.

It is guaranteed that all n points are distinct.

It is guaranteed that there is always at least one way to traverse all n points.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output the minimum cost of a path passing through all points in the corresponding test case.

Example

input
4 3 1 4 2 1 3 1 2 2 4 2 3

```
2
1 1000000000
1 1000000000
4
3 10 5 8
2 5 2 4
```

output

```
0
1
999999999
2
```

G. Maximize the Remaining String

time limit per test: 2.5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a string s, consisting of lowercase Latin letters. While there is at least one character in the string s that is **repeated at least twice**, you perform the following operation:

- you choose the index i ($1 \le i \le |s|$) such that the character at position i occurs **at least two** times in the string s, and delete the character at position i, that is, replace s with $s_1 s_2 \ldots s_{i-1} s_{i+1} s_{i+2} \ldots s_n$.

For example, if $s =$"`codeforces`", then you can apply the following sequence of operations:

- $i = 6 \Rightarrow s =$"`codefrces`";
- $i = 1 \Rightarrow s =$"`odefrces`";
- $i = 7 \Rightarrow s =$"`odefrcs`";

Given a given string s, find the lexicographically maximum string that can be obtained after applying a certain sequence of operations after which all characters in the string **become unique**.

A string a of length n is lexicographically less than a string b of length m, if:

- there is an index i ($1 \le i \le \min(n, m)$) such that the first $i-1$ characters of the strings a and b are the same, and the i-th character of the string a is less than i-th character of string b;
- **or** the first $\min(n, m)$ characters in the strings a and b are the same and $n < m$.

For example, the string $a =$"`aezakmi`" is lexicographically less than the string $b =$"`aezus`".

Input

The first line contains one integer t ($1 \le t \le 10^4$). Then t test cases follow.

Each test case is characterized by a string s, consisting of lowercase Latin letters ($1 \le |s| \le 2 \cdot 10^5$).

It is guaranteed that the sum of the lengths of the strings in all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output the lexicographically maximum string that can be obtained after applying a certain sequence of operations after which all characters in the string **become unique**.

Example

input

```
6
codeforces
aezakmi
abacaba
convexhull
swflldjgpaxs
myneeocktxpqjpz
```

output

```
odfrces
ezakmi
cba
convexhul
wfldjgpaxs
myneocktxqjpz
```