Codeforces Round #726 (Div. 2)

A. Arithmetic Array

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

An array b of length k is called good if its arithmetic mean is equal to 1. More formally, if

$$\frac{b_1 + \cdots + b_k}{k} = 1.$$

Note that the value $\frac{b_1+\cdots+b_k}{k}$ is not rounded up or down. For example, the array $[1,1,1,2]$ has an arithmetic mean of 1.25, which is not equal to 1.

You are given an integer array a of length n. In an operation, you can append a **non-negative** integer to the end of the array. What's the minimum number of operations required to make the array good?

We have a proof that it is always possible with finitely many operations.

Input

The first line contains a single integer t ($1 \le t \le 1000$) — the number of test cases. Then t test cases follow.

The first line of each test case contains a single integer n ($1 \le n \le 50$) — the length of the initial array a.

The second line of each test case contains n integers $a_1, \ldots, a_n$ ($-10^4 \le a_i \le 10^4$), the elements of the array.

Output

For each test case, output a single integer — the minimum number of non-negative integers you have to append to the array so that the arithmetic mean of the array will be exactly 1.

Example

input

```
4
3
1 1 1
2
1 2
4
8 4 6 2
1
-2
```

output

```
0
1
16
1
```

Note

In the first test case, we don't need to add any element because the arithmetic mean of the array is already 1, so the answer is 0.

In the second test case, the arithmetic mean is not 1 initially so we need to add at least one more number. If we add 0 then the arithmetic mean of the whole array becomes 1, so the answer is 1.

In the third test case, the minimum number of elements that need to be added is 16 since only non-negative integers can be added.

In the fourth test case, we can add a single integer 4. The arithmetic mean becomes $\frac{-2+4}{2}$ which is equal to 1.

B. Bad Boy

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

Riley is a very bad boy, but at the same time, he is a yo-yo master. So, he decided to use his yo-yo skills to annoy his friend Anton.

Anton's room can be represented as a grid with n rows and m columns. Let (i, j) denote the cell in row i and column j. Anton is currently standing at position (i, j) in his room. To annoy Anton, Riley decided to throw exactly **two** yo-yos in cells of the room (they can be in the same cell).

Because Anton doesn't like yo-yos thrown on the floor, he has to pick up both of them and return back to the initial position. The distance travelled by Anton is the shortest path that goes through the positions of both yo-yos and returns back to (i, j) by travelling only to adjacent by side cells. That is, if he is in cell (x, y) then he can travel to the cells $(x+1, y)$, $(x-1, y)$, $(x, y+1)$ and $(x, y-1)$ in one step (if a cell with those coordinates exists).

Riley is wondering where he should throw these two yo-yos so that the distance travelled by Anton is **maximized**. But because he is very busy, he asked you to tell him.

Input

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

The only line of each test case contains four integers n, m, i, j ($1 \le n, m \le 10^9$, $1 \le i \le n$, $1 \le j \le m$) — the dimensions of the room, and the cell at which Anton is currently standing.

Output

For each test case, print four integers x_1, y_1, x_2, y_2 ($1 \le x_1, x_2 \le n$, $1 \le y_1, y_2 \le m$) — the coordinates of where the two yo-yos should be thrown. They will be thrown at coordinates (x_1, y_1) and (x_2, y_2).

If there are multiple answers, you may print any.

Example

input

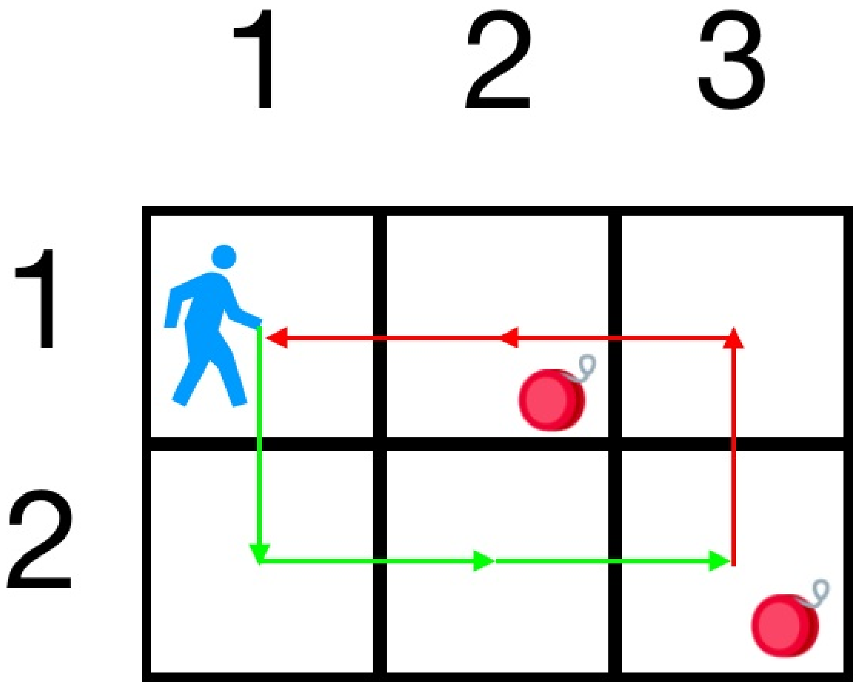
```
7
2 3 1 1
4 4 1 2
3 5 2 2
5 1 2 1
3 1 3 1
1 1 1 1
1000000000 1000000000 1000000000 50
```

output

```
1 2 2 3
4 1 4 4
3 1 1 5
5 1 1 1
1 1 2 1
1 1 1 1
50 1 1 1000000000
```

Note

Here is a visualization of the first test case.



C. Challenging Cliffs

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

You are a game designer and want to make an obstacle course. The player will walk from left to right. You have n heights of mountains already selected and want to arrange them so that the absolute difference of the heights of the first and last mountains is as small as possible.

In addition, you want to make the game difficult, and since walking uphill or flat is harder than walking downhill, the difficulty of the level will be the number of mountains i ($1 \le i < n$) such that $h_i \le h_{i+1}$ where h_i is the height of the i-th mountain. You don't want to waste any of the mountains you modelled, so you have to use all of them.

From all the arrangements that minimize $|h_1 - h_n|$, find one that is the most difficult. If there are multiple orders that satisfy these requirements, you may find any.

Input

The first line will contain a single integer t ($1 \le t \le 100$) — the number of test cases. Then t test cases follow.

The first line of each test case contains a single integer n ($2 \le n \le 2 \cdot 10^5$) — the number of mountains.

The second line of each test case contains n integers $h_1, \ldots, h_n$ ($1 \le h_i \le 10^9$), where h_i is the height of the i-th mountain.

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 n integers — the given heights in an order that maximizes the difficulty score among all orders that minimize $|h_1 - h_n|$.

If there are multiple orders that satisfy these requirements, you may output any.

Example

input

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

output

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

Note

In the first test case:

The player begins at height 2, next going up to height 4 increasing the difficulty by 1. After that he will go down to height 1 and the difficulty doesn't change because he is going downhill. Finally the player will go up to height 2 and the difficulty will increase by 1. The absolute difference between the starting height and the end height is equal to 0 and it's minimal. The difficulty is maximal.

In the second test case:

The player begins at height 1, next going up to height 3 increasing the difficulty by 1. The absolute difference between the starting height and the end height is equal to 2 and it's minimal as they are the only heights. The difficulty is maximal.

D. Deleting Divisors

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

Alice and Bob are playing a game.

They start with a positive integer n and take alternating turns doing operations on it. Each turn a player can subtract from n one of its divisors that isn't 1 or n. The player who cannot make a move on his/her turn loses. Alice always moves first.

Note that they subtract a divisor of the **current** number in each turn.

You are asked to find out who will win the game if both players play optimally.

Input

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

Each test case contains a single integer n ($1 \le n \le 10^9$) — the initial number.

Output

For each test case output "`Alice`" if Alice will win the game or "`Bob`" if Bob will win, if both players play optimally.

Example

input

```
4
1
4
```

```
12
69
```

output

```
Bob
Alice
Alice
Bob
```

Note

In the first test case, the game ends immediately because Alice cannot make a move.

In the second test case, Alice can subtract 2 making $n = 2$, then Bob cannot make a move so Alice wins.

In the third test case, Alice can subtract 3 so that $n = 9$. Bob's only move is to subtract 3 and make $n = 6$. Now, Alice can subtract 3 again and $n = 3$. Then Bob cannot make a move, so Alice wins.

E1. Erase and Extend (Easy Version)

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

This is the easy version of the problem. The only difference is the constraints on n and k. You can make hacks only if all versions of the problem are solved.

You have a string s, and you can do two types of operations on it:

- Delete the last character of the string.
- Duplicate the string: $s := s + s$, where $+$ denotes concatenation.

You can use each operation any number of times (possibly none).

Your task is to find the lexicographically smallest string of length exactly k that can be obtained by doing these operations on string s.

A string a is lexicographically smaller than a string b if and only if one of the following holds:

- a is a prefix of b, but $a \neq b$;
- In the first position where a and b differ, the string a has a letter that appears earlier in the alphabet than the corresponding letter in b.

Input

The first line contains two integers n, k $(1 \leq n, k \leq 5000)$ — the length of the original string s and the length of the desired string.

The second line contains the string s, consisting of n lowercase English letters.

Output

Print the lexicographically smallest string of length k that can be obtained by doing the operations on string s.

Examples

input

```
8 16
dbcadabc
```

output

```
dbcadabcdbcadabc
```

input

```
4 5
abcd
```

output

```
aaaaa
```

Note

In the first test, it is optimal to make one duplication: "dbcadabc" $\rightarrow$ "dbcadabcdbcadabc".

In the second test it is optimal to delete the last 3 characters, then duplicate the string 3 times, then delete the last 3 characters to make the string have length k.

"abcd" $\rightarrow$ "abc" $\rightarrow$ "ab" $\rightarrow$ "a" $\rightarrow$ "aa" $\rightarrow$ "aaaa" $\rightarrow$ "aaaaaaaa" $\rightarrow$ "aaaaaaa" $\rightarrow$ "aaaaaa" $\rightarrow$ "aaaaa".

E2. Erase and Extend (Hard Version)

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

input: standard input
output: standard output

This is the hard version of the problem. The only difference is the constraints on n and k. You can make hacks only if all versions of the problem are solved.

You have a string s, and you can do two types of operations on it:

- Delete the last character of the string.
- Duplicate the string: $s := s + s$, where $+$ denotes concatenation.

You can use each operation any number of times (possibly none).

Your task is to find the lexicographically smallest string of length exactly k that can be obtained by doing these operations on string s.

A string a is lexicographically smaller than a string b if and only if one of the following holds:

- a is a prefix of b, but $a \ne b$;
- In the first position where a and b differ, the string a has a letter that appears earlier in the alphabet than the corresponding letter in b.

Input

The first line contains two integers n, k ($1 \le n, k \le 5 \cdot 10^5$) — the length of the original string s and the length of the desired string.

The second line contains the string s, consisting of n lowercase English letters.

Output

Print the lexicographically smallest string of length k that can be obtained by doing the operations on string s.

Examples

input
```
8 16
dbcadabc
```
output
```
dbcadabcdbcadabc
```

input
```
4 5
abcd
```
output
```
aaaaa
```

Note

In the first test, it is optimal to make one duplication: "dbcadabc" $\to$ "dbcadabcdbcadabc".

In the second test it is optimal to delete the last 3 characters, then duplicate the string 3 times, then delete the last 3 characters to make the string have length k.

"abcd" $\to$ "abc" $\to$ "ab" $\to$ "a" $\to$ "aa" $\to$ "aaaa" $\to$ "aaaaaaaa" $\to$ "aaaaaaa" $\to$ "aaaaaa" $\to$ "aaaaa".

F. Figure Fixing

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

You have a connected undirected graph made of n nodes and m edges. The i-th node has a value v_i and a target value t_i.

In an operation, you can choose an edge (i, j) and add k to both v_i and v_j, where k can be any **integer**. In particular, k can be negative.

Your task to determine if it is possible that by doing some finite number of operations (possibly zero), you can achieve for every node i, $v_i = t_i$.

Input

The first line contains a single integer t ($1 \le t \le 1000$), the number of test cases. Then the test cases follow.

The first line of each test case contains two integers n, m ($2 \le n \le 2 \cdot 10^5$, $n-1 \le m \le \min(2 \cdot 10^5, \frac{n(n-1)}{2})$) — the number of nodes and edges respectively.

The second line contains n integers $v_1 \ldots, v_n$ ($-10^9 \le v_i \le 10^9$) — initial values of nodes.

The third line contains n integers $t_1 \ldots, t_n$ ($-10^9 \leq t_i \leq 10^9$) — target values of nodes.

Each of the next m lines contains two integers i and j representing an edge between node i and node j ($1 \leq i, j \leq n, i \neq j$).

It is guaranteed that the graph is connected and there is at most one edge between the same pair of nodes.

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

Output

For each test case, if it is possible for every node to reach its target after some number of operations, print "YES". Otherwise, print "NO".

Example

input

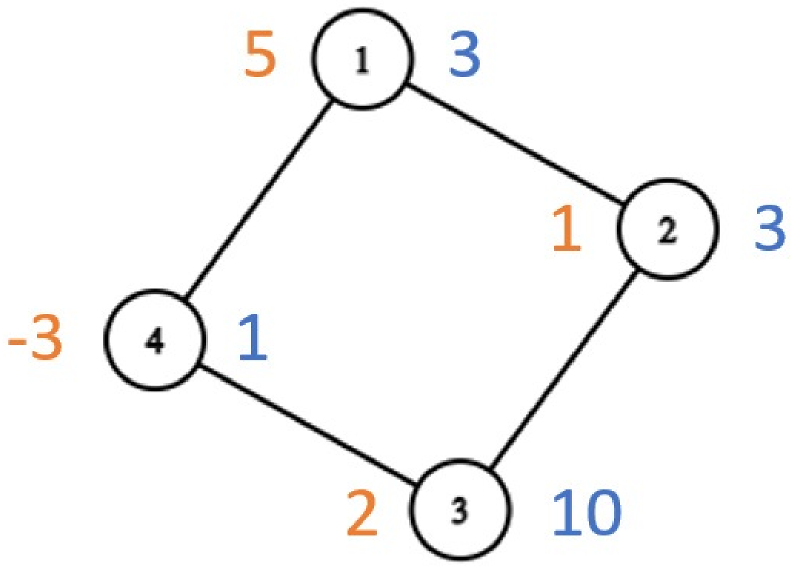
```
2
4 4
5 1 2 -3
3 3 10 1
1 2
1 4
3 2
3 4
4 4
5 8 6 6
-3 1 15 4
1 2
1 4
3 2
3 4
```

output

```
YES
NO
```

Note

Here is a visualization of the first test case (the orange values denote the initial values and the blue ones the desired values):



One possible order of operations to obtain the desired values for each node is the following:

- Operation 1: Add 2 to nodes 2 and 3.
- Operation 2: Add -2 to nodes 1 and 4.
- Operation 3: Add 6 to nodes 3 and 4.

Now we can see that in total we added -2 to node 1, 2 to node 2, 8 to node 3 and 4 to node 4 which brings each node exactly to it's desired value.

For the graph from the second test case it's impossible to get the target values.