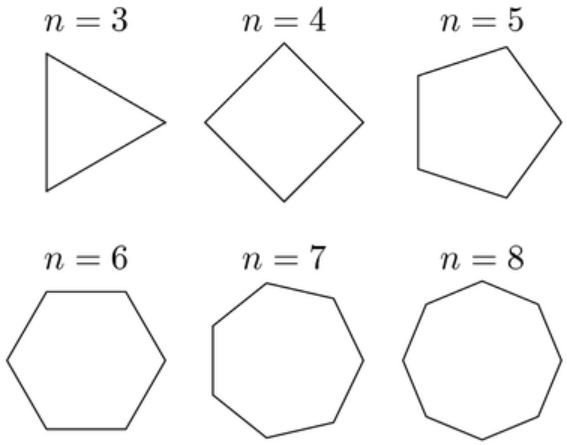
Educational Codeforces Round 83 (Rated for Div. 2)

A. Two Regular Polygons

1 second, 256 megabytes

You are given two integers n and m ($m < n$). Consider a **convex** regular polygon of n vertices. Recall that a regular polygon is a polygon that is equiangular (all angles are equal in measure) and equilateral (all sides have the same length).



Examples of convex regular polygons

Your task is to say if it is possible to build another **convex** regular polygon with m vertices such that its center coincides with the center of the initial polygon and each of its vertices is some vertex of the initial polygon.

You have to answer t independent test cases.

Input

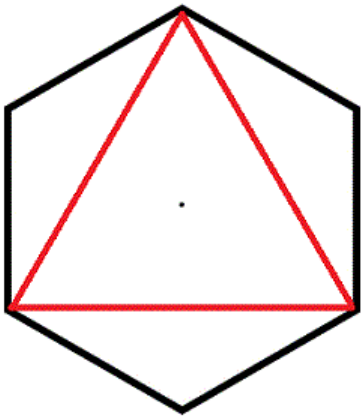
The first line of the input contains one integer t ($1 \le t \le 10^4$) — the number of test cases.

The next t lines describe test cases. Each test case is given as two space-separated integers n and m ($3 \le m < n \le 100$) — the number of vertices in the initial polygon and the number of vertices in the polygon you want to build.

Output

For each test case, print the answer — "YES" (without quotes), if it is possible to build another **convex** regular polygon with m vertices such that its center coincides with the center of the initial polygon and each of its vertices is some vertex of the initial polygon and "NO" otherwise.

input
2 6 3 7 3
output
YES NO



The first test case of the example

It can be shown that the answer for the second test case of the example is "NO".

B. Bogosort

2 seconds, 256 megabytes

You are given an array $a_1, a_2, \ldots, a_n$. Array is good if for each pair of indexes $i < j$ the condition $j - a_j \neq i - a_i$ holds. Can you shuffle this array so that it becomes good? To shuffle an array means to reorder its elements arbitrarily (leaving the initial order is also an option).

For example, if $a = [1, 1, 3, 5]$, then shuffled arrays $[1, 3, 5, 1]$, $[3, 5, 1, 1]$ and $[5, 3, 1, 1]$ are good, but shuffled arrays $[3, 1, 5, 1]$, $[1, 1, 3, 5]$ and $[1, 1, 5, 3]$ aren't.

It's guaranteed that it's always possible to shuffle an array to meet this condition.

Input

The first line contains one integer t ($1 \le t \le 100$) — the number of test cases.

The first line of each test case contains one integer n ($1 \le n \le 100$) — the length of array a.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 100$).

Output

For each test case print the shuffled version of the array a which is good.

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

C. Adding Powers

2 seconds, 256 megabytes

Suppose you are performing the following algorithm. There is an array $v_1, v_2, \ldots, v_n$ filled with zeroes at start. The following operation is applied to the array several times — at i-th step (0-indexed) you can:

- either choose position pos ($1 \le pos \le n$) and increase v_{pos} by k^i;
- or not choose any position and skip this step.

You can choose how the algorithm would behave on each step and when to stop it. The question is: can you make array v equal to the given array a ($v_j = a_j$ for each j) after some step?

Input
The first line contains one integer T ($1 \le T \le 1000$) — the number of test cases. Next $2T$ lines contain test cases — two lines per test case.

The first line of each test case contains two integers n and k ($1 \le n \le 30, 2 \le k \le 100$) — the size of arrays v and a and value k used in the algorithm.

The second line contains n integers $a_1, a_2, \dots, a_n$ ($0 \le a_i \le 10^{16}$) — the array you'd like to achieve.

Output
For each test case print `YES` (case insensitive) if you can achieve the array a after some step or `NO` (case insensitive) otherwise.

input
5 4 100 0 0 0 0 1 2 1 3 4 1 4 1 3 2 0 1 3 3 9 0 59049 810
output
YES YES NO NO YES

In the first test case, you can stop the algorithm before the 0-th step, or don't choose any position several times and stop the algorithm.

In the second test case, you can add k^0 to v_1 and stop the algorithm.

In the third test case, you can't make two 1 in the array v.

In the fifth test case, you can skip 9^0 and 9^1, then add 9^2 and 9^3 to v_3, skip 9^4 and finally, add 9^5 to v_2.

D. Count the Arrays

2 seconds, 512 megabytes

Your task is to calculate the number of arrays such that:

- each array contains n elements;
- each element is an integer from 1 to m;
- for each array, there is **exactly** one pair of equal elements;
- for each array a, there exists an index i such that the array is **strictly ascending** before the i-th element and **strictly descending** after it (formally, it means that $a_j < a_{j+1}$, if $j < i$, and $a_j > a_{j+1}$, if $j \ge i$).

Input
The first line contains two integers n and m ($2 \le n \le m \le 2 \cdot 10^5$).

Output
Print one integer — the number of arrays that meet all of the aforementioned conditions, taken modulo 998244353.

input
3 4
output
6

input
3 5
output
10

input
42 1337
output
806066790

input
100000 200000
output
707899035

The arrays in the first example are:

- $[1, 2, 1]$;
- $[1, 3, 1]$;
- $[1, 4, 1]$;
- $[2, 3, 2]$;
- $[2, 4, 2]$;
- $[3, 4, 3]$.

E. Array Shrinking

2 seconds, 256 megabytes

You are given an array $a_1, a_2, \dots, a_n$. You can perform the following operation any number of times:

- Choose a pair of two neighboring equal elements $a_i = a_{i+1}$ (if there is at least one such pair).
- Replace them by one element with value $a_i + 1$.

After each such operation, the length of the array will decrease by one (and elements are renumerated accordingly). What is the minimum possible length of the array a you can get?

Input
The first line contains the single integer n ($1 \le n \le 500$) — the initial length of the array a.

The second line contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le 1000$) — the initial array a.

Output
Print the only integer — the minimum possible length you can get after performing the operation described above any number of times.

input
5 4 3 2 2 3
output
2

input
7 3 3 4 4 4 3 3
output
2

input
3 1 3 5
output
3

input
1 1000
output
1

In the first test, this is one of the optimal sequences of operations: $4\ 3\ 2\ 2 \rightarrow 4\ 3\ 3\ 3 \rightarrow 4\ 4\ 3 \rightarrow 5\ 3$.

In the second test, this is one of the optimal sequences of operations: $3\ 3\ 4\ 4\ 4\ 3\ 3 \rightarrow 4\ 4\ 4\ 4\ 3\ 3 \rightarrow 4\ 4\ 4\ 4\ 4 \rightarrow 5\ 4\ 4\ 4 \rightarrow 5\ 5\ 4 \rightarrow 6\ 4$.

In the third and fourth tests, you can't perform the operation at all.

F. Attack on Red Kingdom

3 seconds, 512 megabytes

The Red Kingdom is attacked by the White King and the Black King!

The Kingdom is guarded by n castles, the i-th castle is defended by a_i soldiers. To conquer the Red Kingdom, the Kings have to eliminate all the defenders.

Each day the White King launches an attack on one of the castles. Then, at night, the forces of the Black King attack a castle (possibly the same one). Then the White King attacks a castle, then the Black King, and so on. The first attack is performed by the White King.

Each attack must target a castle with **at least one** alive defender in it. There are three types of attacks:

- a *mixed attack* decreases the number of defenders in the targeted castle by x (or sets it to 0 if there are already less than x defenders);
- an *infantry attack* decreases the number of defenders in the targeted castle by y (or sets it to 0 if there are already less than y defenders);
- a *cavalry attack* decreases the number of defenders in the targeted castle by z (or sets it to 0 if there are already less than z defenders).

The *mixed attack* can be launched at any valid target (at any castle with at least one soldier). However, the *infantry attack* cannot be launched if the **previous attack on the targeted castle** had the same type, no matter when and by whom it was launched. The same applies to the *cavalry attack*. A castle that was not attacked at all can be targeted by any type of attack.

The King who launches the last attack will be glorified as the conqueror of the Red Kingdom, so both Kings want to launch the last attack (and they are wise enough to find a strategy that allows them to do it no matter what are the actions of their opponent, if such strategy exists). The White King is leading his first attack, and you are responsible for planning it. Can you calculate the number of possible options for the first attack that allow the White King to launch the last attack? Each option for the first attack is represented by the targeted castle and the type of attack, and two options are different if the targeted castles or the types of attack are different.

Input

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

Then, the test cases follow. Each test case is represented by two lines.

The first line contains four integers n, x, y and z ($1 \le n \le 3 \cdot 10^5$, $1 \le x, y, z \le 5$).

The second line contains n integers a_1, a_2, ..., a_n ($1 \le a_i \le 10^{18}$).

It is guaranteed that the sum of values of n over all test cases in the input does not exceed $3 \cdot 10^5$.

Output

For each test case, print the answer to it: the number of possible options for the first attack of the White King (or 0, if the Black King can launch the last attack no matter how the White King acts).

input
3 2 1 3 4 7 6 1 1 2 3 1 1 1 2 2 3
output
2 3 0

input
10 6 5 4 5 2 3 2 3 1 3 1 5 2 3 10 4 4 2 3 8 10 8 5 2 2 1 4 8 5 3 5 3 5 9 2 10 4 5 5 5 2 10 4 2 2 3 1 4 1 10 3 1 5 3 9 8 7 2 5 4 5 8 8 3 5 1 4 5 5 10
output
0 2 1 2 5 12 5 0 0 2

G. Autocompletion

7 seconds, 512 megabytes

You are given a set of strings S. Each string consists of lowercase Latin letters.

For each string in this set, you want to calculate the minimum number of seconds required to type this string. To type a string, you have to start with an empty string and transform it into the string you want to type using the following actions:

- if the current string is t, choose some lowercase Latin letter c and append it to the back of t, so the current string becomes $t + c$. This action takes 1 second;
- use autocompletion. When you try to autocomplete the current string t, a list of all strings $s \in S$ such that t is a prefix of s is shown to you. **This list includes t itself, if t is a string from S**, and the strings are ordered lexicographically. You can transform t into the i-th string from this list in i seconds. Note that you may choose any string from this list you want, it is not necessarily the string you are trying to type.

What is the minimum number of seconds that you have to spend to type each string from S?

Note that the strings from S are given in an unusual way.

Input

The first line contains one integer n ($1 \le n \le 10^6$).

Then n lines follow, the i-th line contains one integer p_i ($0 \le p_i < i$) and one lowercase Latin character c_i. These lines form some set of strings such that S is its subset as follows: there are $n + 1$ strings, numbered from 0 to n; the 0-th string is an empty string, and the i-th string ($i \ge 1$) is the result of appending the character c_i to the string p_i. **It is guaranteed that all these strings are distinct**.

The next line contains one integer k ($1 \le k \le n$) — the number of strings in S.

The last line contains k integers a_1, a_2, ..., a_k ($1 \le a_i \le n$, all a_i are pairwise distinct) denoting the indices of the strings generated by above-mentioned process that form the set S — formally, if we denote the i-th generated string as s_i, then $S = s_{a_1}, s_{a_2}, \ldots, s_{a_k}$.

Output

Print k integers, the i-th of them should be equal to the minimum number of seconds required to type the string s_{a_i}.

input

```
10
0 i
1 q
2 g
0 k
1 e
5 r
4 m
5 h
3 p
3 e
5
8 9 1 10 6
```

output

```
2 4 1 3 3
```

input

```
8
0 a
1 b
2 a
2 b
4 a
4 b
5 c
6 d
5
2 3 4 7 8
```

output

```
1 2 2 4 4
```

In the first example, S consists of the following strings: `ieh`, `iqgp`, `i`, `iqge`, `ier`.