A. Destroying Bridges

Input file: standard input
Output file: standard output
Time limit: 1 second
Memory limit: 256 megabytes

There are n islands, numbered $1, 2, \ldots, n$. Initially, every pair of islands is connected by a bridge. Hence, there are a total of $\frac{n(n-1)}{2}$ bridges.

Everule lives on island 1 and enjoys visiting the other islands using bridges. Dominater has the power to destroy at most k bridges to minimize the number of islands that Everule can reach using (possibly multiple) bridges.

Find the minimum number of islands (including island 1) that Everule can visit if Dominater destroys bridges optimally.

Input

Each test contains multiple test cases. The first line contains a single integer t ($1 \leq t \leq 10^3$) — the number of test cases. The description of the test cases follows.

The first and only line of each test case contains two integers n and k ($1 \leq n \leq 100$, $0 \leq k \leq \frac{n \cdot (n-1)}{2}$).

Output

For each test case, output the minimum number of islands that Everule can visit if Dominater destroys bridges optimally.

Standard Input	Standard Output
6 2 0 2 1 4 1 5 10 5 3 4 4	2 1 4 1 5 1

Note

In the first test case, since no bridges can be destroyed, all the islands will be reachable.

In the second test case, you can destroy the bridge between islands 1 and 2. Everule will not be able to visit island 2 but can still visit island 1. Therefore, the total number of islands that Everule can visit is 1.

In the third test case, Everule always has a way of reaching all islands despite what Dominater does. For example, if Dominater destroyed the bridge between islands 1 and 2, Everule can still visit island 2 by traveling by $1 \to 3 \to 2$ as the bridges between 1 and 3, and between 3 and 2 are not destroyed.

In the fourth test case, you can destroy all bridges since $k = \frac{n \cdot (n-1)}{2}$. Everule will be only able to visit 1 island (island 1).

,

B. Equal XOR

Input file: standard input
Output file: standard output
Time limit: 1 second
Memory limit: 256 megabytes

You are given an array a of length $2n$, consisting of each integer from 1 to n exactly **twice**.

You are also given an integer k ($1 \le k \le \lfloor \frac{n}{2} \rfloor$).

You need to find two arrays l and r each of length $\mathbf{2k}$ such that:

- l is a subset† of $[a_1, a_2, \ldots a_n]$
- r is a subset of $[a_{n+1}, a_{n+2}, \ldots a_{2n}]$
- bitwise XOR of elements of l is equal to the bitwise XOR of elements of r; in other words, $l_1 \oplus l_2 \oplus \ldots \oplus l_{2k} = r_1 \oplus r_2 \oplus \ldots \oplus r_{2k}$

It can be proved that at least one pair of l and r always exists. If there are multiple solutions, you may output any one of them.

† A sequence x is a subset of a sequence y if x can be obtained by deleting several (possibly none or all) elements of y and rearranging the elements in any order. For example, $[3, 1, 2, 1]$, $[1, 2, 3]$, $[1, 1]$ and $[3, 2]$ are subsets of $[1, 1, 2, 3]$ but $[4]$ and $[2, 2]$ are not subsets of $[1, 1, 2, 3]$.

Input

Each test contains multiple test cases. The first line contains a single integer t ($1 \le t \le 5000$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains 2 integers n and k ($2 \le n \le 5 \cdot 10^4$, $1 \le k \le \lfloor \frac{n}{2} \rfloor$).

The second line contains $2n$ integers $a_1, a_2, \ldots, a_{2n}$ ($1 \le a_i \le n$). It is guaranteed that every integer from 1 to n occurs exactly twice in a.

It is guaranteed that the sum of n over all test cases does not exceed $5 \cdot 10^4$.

Output

For each test case, output two lines.

On the first line of output, output $2k$ integers $l_1, l_2, \ldots, l_{2k}$.

On the second line of output, output $2k$ integers $r_1, r_2, \ldots r_{2k}$.

If there are multiple solutions, you may output any one of them.

Standard Input	Standard Output
4 2 1 1 2 2 1 6 1 6 4 2 1 2 3 1 6 3 5 5 4 4 1 1 2 3 4 1 2 3 4	2 1 2 1 6 4 1 3 1 2 1 2

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

Note

In the first test case, we choose $l = [2, 1]$ and $r = [2, 1]$. $[2, 1]$ is a subset of $[a_1, a_2]$ and $[2, 1]$ is a subset of $[a_3, a_4]$, and $2 \oplus 1 = 2 \oplus 1 = 3$.

In the second test case, $6 \oplus 4 = 1 \oplus 3 = 2$.

,

C. MEX Game 1

Input file: standard input
Output file: standard output
Time limit: 2 seconds
Memory limit: 256 megabytes

Alice and Bob play yet another game on an array a of size n. Alice starts with an empty array c. Both players take turns playing, with Alice starting first.

On Alice's turn, she picks one element from a, appends that element to c, and then deletes it from a.

On Bob's turn, he picks one element from a, and then deletes it from a.

The game ends when the array a is empty. Game's score is defined to be the MEX† of c. Alice wants to maximize the score while Bob wants to minimize it. Find game's final score if both players play optimally.

† The MEX (minimum excludant) of an array of integers is defined as the smallest non-negative integer which does not occur in the array. For example:

- The MEX of $[2, 2, 1]$ is 0, because 0 does not belong to the array.
- The MEX of $[3, 1, 0, 1]$ is 2, because 0 and 1 belong to the array, but 2 does not.
- The MEX of $[0, 3, 1, 2]$ is 4, because 0, 1, 2 and 3 belong to the array, but 4 does not.

Input

Each test contains multiple test cases. The first line contains a single integer t ($1 \le t \le 2 \cdot 10^4$) — the number of test cases. The description of the test cases follows.

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

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i < n$).

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

Output

For each test case, find game's score if both players play optimally.

Standard Input	Standard Output
3 4 0 0 1 1 4 0 1 2 3 2 1 1	2 1 0

Note

In the first test case, a possible game with a score of 2 is as follows:

1. Alice chooses the element 1. After this move, $a = [0, 0, 1]$ and $c = [1]$.
2. Bob chooses the element 0. After this move, $a = [0, 1]$ and $c = [1]$.

3. Alice chooses the element 0. After this move, $a = [1]$ and $c = [1, 0]$.
4. Bob chooses the element 1. After this move, $a = [\,]$ and $c = [1, 0]$.

At the end, $c = [1, 0]$, which has a MEX of 2. Note that this is an example game and does not necessarily represent the optimal strategy for both players.

,

D. Non-Palindromic Substring

```
Input file:     standard input
Output file:    standard output
Time limit:     3 seconds
Memory limit:   256 megabytes
```

A string t is said to be k-good if there exists at least one substring[†] of length k which is not a palindrome[‡]. Let $f(t)$ denote the sum of all values of k such that the string t is k-good.

You are given a string s of length n. You will have to answer q of the following queries:

- Given l and r ($l < r$), find the value of $f(s_l s_{l+1} \ldots s_r)$.

[†] A substring of a string z is a contiguous segment of characters from z. For example, "`defor`", "`code`" and "`o`" are all substrings of "`codeforces`" while "`codes`" and "`aaa`" are not.

[‡] A palindrome is a string that reads the same backwards as forwards. For example, the strings "`z`", "`aa`" and "`tacocat`" are palindromes while "`codeforces`" and "`ab`" are not.

Input

Each test contains multiple test cases. The first line contains a single integer t ($1 \le t \le 2 \cdot 10^4$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains two integers n and q ($2 \le n \le 2 \cdot 10^5, 1 \le q \le 2 \cdot 10^5$), the size of the string and the number of queries respectively.

The second line of each test case contains the string s. It is guaranteed the string s only contains lowercase English characters.

The next q lines each contain two integers, l and r ($1 \le l < r \le n$).

It is guaranteed the sum of n and the sum of q both do not exceed $2 \cdot 10^5$.

Output

For each query, output $f(s_l s_{l+1} \ldots s_r)$.

Standard Input	Standard Output
5 4 4 aaab 1 4 1 3 3 4 2 4 3 2 abc 1 3 1 2 5 4 pqpcc 1 5	9 0 2 5 5 2 14 0 2 5 0 65

4 5 1 3 2 4 2 1 aa 1 2 12 1 steponnopets 1 12	

Note

In the first query of the first test case, the string is **aaab**. **aaab**, **aab** and **ab** are all substrings that are not palindromes, and they have lengths 4, 3 and 2 respectively. Thus, the string is 2-good, 3-good and 4-good. Hence, $f(\texttt{aaab}) = 2 + 3 + 4 = 9$.

In the second query of the first test case, the string is **aaa**. There are no non-palindromic substrings. Hence, $f(\texttt{aaa}) = 0$.

In the first query of the second test case, the string is **abc**. **ab**, **bc** and **abc** are all substrings that are not palindromes, and they have lengths 2, 2 and 3 respectively. Thus, the string is 2-good and 3-good. Hence, $f(\texttt{abc}) = 2 + 3 = 5$. Note that even though there are 2 non-palindromic substrings of length 2, we count it only once.

,

E. Tree Compass

Input file: standard input
Output file: standard output
Time limit: 2 seconds
Memory limit: 256 megabytes

You are given a tree with n vertices numbered $1, 2, \ldots, n$. Initially, all vertices are colored white.

You can perform the following two-step operation:

1. Choose a vertex v ($1 \le v \le n$) and a distance d ($0 \le d \le n - 1$).
2. For all vertices u ($1 \le u \le n$) such that $\mathrm{dist}^{\dagger}(u, v) = d$, color u black.

Construct a sequence of operations to color all the nodes in the tree black using the minimum possible number of operations. It can be proven that it is always possible to do so using at most n operations.

† $\mathrm{dist}(x, y)$ denotes the number of edges on the (unique) simple path between vertices x and y on the tree.

Input

Each test contains multiple test cases. The first line contains a single integer t ($1 \le t \le 200$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer n ($1 \le n \le 2 \cdot 10^3$) — the number of vertices of the tree.

The following $n - 1$ lines of each test case describe the edges of the tree. The i-th of these lines contains two integers u_i and v_i ($1 \le u_i, v_i \le n, u_i \ne v_i$), the indices of the vertices connected by the i-th edge.

It is guaranteed that the given edges form a tree.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^3$.

Output

For each test case, first output a single integer op ($1 \le op \le n$), the minimum number of operations needed to color all vertices of the tree black.

Then, output op lines, each containing 2 integers. The i-th line should contain the values of v and d chosen for the i-th operation ($1 \le v \le n, 0 \le d \le n - 1$)

You must guarantee that at the end of op operations, all vertices are colored black.

If there are multiple solutions, you may output any one of them.

Standard Input	Standard Output
4 1 2 1 2 4 1 2 1 3 1 4	1 1 0 2 1 1 2 1 2 1 1 2 1

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

Note

In the first test case, there is only one possible operation, and performing it gives us a valid answer.

In the second test case, the first operation colors vertex 2 black, and the second operation colors vertex 1 black. It can be shown that it is impossible to color both vertices black in one operation, so the minimum number of operations needed is 2. Another possible solution is to use the 2 operations: $(u, r) = (1, 0)$ and $(u, r) = (2, 0)$.

In the third test case, the first operation colors vertices 2, 3 and 4 black, and the second operation colors vertex 1 black. Again, it can be shown that it is impossible to color all vertices black in 1 operation, so the minimum number of operations needed is 2.

In the fourth test case, the first operation colors vertices 4, 1 and 7 black, the second operation colors vertices 2, 5 and 6 black while the third operation colors vertices 3 and 7 black. Notice that it is allowed to color vertex 7 black twice.

Thus, each node was marked at least once, with node 7 marked twice. It can be shown that it is impossible to color all vertices black in fewer than 3 moves.

,

F1. Counting Is Fun (Easy Version)

Input file: standard input
Output file: standard output
Time limit: 3 seconds
Memory limit: 1024 megabytes

This is the easy version of the problem. The only difference between the two versions is the constraint on n. You can make hacks only if both versions of the problem are solved.

An array b of m non-negative integers is said to be *good* if all the elements of b can be made equal to 0 using the following operation some (possibly, zero) times:

- Select two **distinct** indices l and r ($1 \leq l < r \leq m$) and subtract 1 from all b_i such that $l \leq i \leq r$.

You are given two positive integers n, k and a prime number p.

Over all $(k+1)^n$ arrays of length n such that $0 \leq a_i \leq k$ for all $1 \leq i \leq n$, count the number of good arrays.

Since the number might be too large, you are only required to find it modulo p.

Input

Each test contains multiple test cases. The first line contains a single integer t ($1 \leq t \leq 10^3$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains three positive integers n, k and p ($3 \leq n \leq 400$, $1 \leq k \leq n$, $10^8 < p < 10^9$) — the length of the array a, the upper bound on the elements of a and modulus p.

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

Output

For each test case, on a new line, output the number of good arrays modulo p.

Standard Input	Standard Output
4 3 1 998244853 4 1 998244353 3 2 998244353 343 343 998244353	4 7 10 456615865

Note

In the first test case, the 4 good arrays a are:

- $[0, 0, 0]$;
- $[0, 1, 1]$;
- $[1, 1, 0]$;
- $[1, 1, 1]$.

,

F2. Counting Is Fun (Hard Version)

```
Input file:     standard input
Output file:    standard output
Time limit:     3 seconds
Memory limit:   1024 megabytes
```

This is the hard version of the problem. The only difference between the two versions is the constraint on n. You can make hacks only if both versions of the problem are solved.

An array b of m non-negative integers is said to be *good* if all the elements of b can be made equal to 0 using the following operation some (possibly, zero) times:

- Select two **distinct** indices l and r ($1 \leq l < r \leq m$) and subtract 1 from all b_i such that $l \leq i \leq r$.

You are given two positive integers n, k and a prime number p.

Over all $(k+1)^n$ arrays of length n such that $0 \leq a_i \leq k$ for all $1 \leq i \leq n$, count the number of good arrays.

Since the number might be too large, you are only required to find it modulo p.

Input

Each test contains multiple test cases. The first line contains a single integer t ($1 \leq t \leq 10^3$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains three positive integers n, k and p ($3 \leq n \leq 3000$, $1 \leq k \leq n$, $10^8 < p < 10^9$) — the length of the array a, the upper bound on the elements of a and modulus p.

It is guaranteed that the sum of n^2 over all test cases does not exceed 10^7, and p is prime.

Output

For each test case, on a new line, output the number of good arrays modulo p.

Standard Input	Standard Output
4 3 1 998244853 4 1 998244353 3 2 998244353 343 343 998244353	4 7 10 456615865

Note

In the first test case, the 4 good arrays a are:

- $[0, 0, 0]$;
- $[0, 1, 1]$;
- $[1, 1, 0]$;
- $[1, 1, 1]$.