Codeforces Round #658 (Div. 2)

A. Common Subsequence

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

You are given two arrays of integers $a_1, \ldots, a_n$ and $b_1, \ldots, b_m$.

Your task is to find a **non-empty** array $c_1, \ldots, c_k$ that is a subsequence of $a_1, \ldots, a_n$, and also a subsequence of $b_1, \ldots, b_m$. If there are multiple answers, find one of the **smallest** possible length. If there are still multiple of the smallest possible length, find any. If there are no such arrays, you should report about it.

A sequence a is a subsequence of a sequence b if a can be obtained from b by deletion of several (possibly, zero) elements. For example, $[3,1]$ is a subsequence of $[3,2,1]$ and $[4,3,1]$, but not a subsequence of $[1,3,3,7]$ and $[3,10,4]$.

Input

The first line contains a single integer t $(1 \le t \le 1000)$ — the number of test cases. Next $3t$ lines contain descriptions of test cases.

The first line of each test case contains two integers n and m $(1 \le n, m \le 1000)$ — the lengths of the two arrays.

The second line of each test case contains n integers $a_1, \ldots, a_n$ $(1 \le a_i \le 1000)$ — the elements of the first array.

The third line of each test case contains m integers $b_1, \ldots, b_m$ $(1 \le b_i \le 1000)$ — the elements of the second array.

It is guaranteed that the sum of n and the sum of m across all test cases does not exceed 1000 ($\sum_{i=1}^{t} n_i, \sum_{i=1}^{t} m_i \le 1000$).

Output

For each test case, output "YES" if a solution exists, or "NO" otherwise.

If the answer is "YES", on the next line output an integer k $(1 \le k \le 1000)$ — the length of the array, followed by k integers $c_1, \ldots, c_k$ $(1 \le c_i \le 1000)$ — the elements of the array.

If there are multiple solutions with the smallest possible k, output any.

Example

input

```
5
4 5
10 8 6 4
1 2 3 4 5
1 1
3
3
1 1
3
2
5 3
1000 2 2 2 3
3 1 5
5 5
1 2 3 4 5
1 2 3 4 5
```

output

```
YES
1 4
YES
1 3
NO
YES
1 3
YES
1 2
```

Note

In the first test case, $[4]$ is a subsequence of $[10,8,6,4]$ and $[1,2,3,4,5]$. This array has length 1, it is the smallest possible length of a subsequence of both a and b.

In the third test case, no non-empty subsequences of both $[3]$ and $[2]$ exist, so the answer is "NO".

B. Sequential Nim

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

There are n piles of stones, where the i-th pile has a_i stones. Two people play a game, where they take alternating turns removing stones.

In a move, a player may remove a positive number of stones from the **first non-empty pile** (the pile with the minimal index, that has at least one stone). The first player who cannot make a move (because all piles are empty) loses the game. If both players play optimally, determine the winner of the game.

Input

The first line contains a single integer t $(1 \le t \le 1000)$ — the number of test cases. Next $2t$ lines contain descriptions of test cases.

The first line of each test case contains a single integer n $(1 \le n \le 10^5)$ — the number of piles.

The second line of each test case contains n integers $a_1, \ldots, a_n$ $(1 \le a_i \le 10^9)$ — a_i is equal to the number of stones in the i-th pile.

It is guaranteed that the sum of n for all test cases does not exceed 10^5.

Output

For each test case, if the player who makes the first move will win, output "`First`". Otherwise, output "`Second`".

Example

input

```
7
3
2 5 4
8
1 1 1 1 1 1 1 1
6
1 2 3 4 5 6
6
1 1 2 1 2 2
1
1000000000
5
1 2 2 1 1
3
1 1 1
```

output

```
First
Second
Second
First
First
Second
First
```

Note

In the first test case, the first player will win the game. His winning strategy is:

1. The first player should take the stones from the first pile. He will take 1 stone. The numbers of stones in piles will be $[1, 5, 4]$.
2. The second player should take the stones from the first pile. He will take 1 stone because he can't take any other number of stones. The numbers of stones in piles will be $[0, 5, 4]$.
3. The first player should take the stones from the second pile because the first pile is empty. He will take 4 stones. The numbers of stones in piles will be $[0, 1, 4]$.
4. The second player should take the stones from the second pile because the first pile is empty. He will take 1 stone because he can't take any other number of stones. The numbers of stones in piles will be $[0, 0, 4]$.
5. The first player should take the stones from the third pile because the first and second piles are empty. He will take 4 stones. The numbers of stones in piles will be $[0, 0, 0]$.
6. The second player will lose the game because all piles will be empty.

C1. Prefix Flip (Easy Version)

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

This is the easy version of the problem. The difference between the versions is the constraint on n and the required number of operations. You can make hacks only if all versions of the problem are solved.

There are two binary strings a and b of length n (a binary string is a string consisting of symbols 0 and 1). In an operation, you select a prefix of a, and simultaneously invert the bits in the prefix (0 changes to 1 and 1 changes to 0) and reverse the order of the bits in the prefix.

For example, if $a = 001011$ and you select the prefix of length 3, it becomes 011011. Then if you select the entire string, it becomes 001001.

Your task is to transform the string a into b in at most $3n$ operations. It can be proved that it is always possible.

Input

The first line contains a single integer t $(1 \le t \le 1000)$ — the number of test cases. Next $3t$ lines contain descriptions of test cases.

The first line of each test case contains a single integer n $(1 \le n \le 1000)$ — the length of the binary strings.

The next two lines contain two binary strings a and b of length n.

It is guaranteed that the sum of n across all test cases does not exceed 1000.

Output

For each test case, output an integer k $(0 \le k \le 3n)$, followed by k integers $p_1, \ldots, p_k$ $(1 \le p_i \le n)$. Here k is the number of operations you use and p_i is the length of the prefix you flip in the i-th operation.

Example

input

```
5
2
01
10
5
01011
11100
2
01
01
10
0110011011
1000110100
1
0
1
```

output

```
3 1 2 1
6 5 2 5 3 1 2
0
9 4 1 2 10 4 1 2 1 5
1 1
```

Note

In the first test case, we have $01 \to 11 \to 00 \to 10$.

In the second test case, we have $01011 \to 00101 \to 11101 \to 01000 \to 10100 \to 00100 \to 11100$.

In the third test case, the strings are already the same. Another solution is to flip the prefix of length 2, which will leave a unchanged.

C2. Prefix Flip (Hard Version)

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

This is the hard version of the problem. The difference between the versions is the constraint on n and the required number of operations. You can make hacks only if all versions of the problem are solved.

There are two binary strings a and b of length n (a binary string is a string consisting of symbols 0 and 1). In an operation, you select a prefix of a, and simultaneously invert the bits in the prefix (0 changes to 1 and 1 changes to 0) and reverse the order of the bits in the prefix.

For example, if $a = 001011$ and you select the prefix of length 3, it becomes 011011. Then if you select the entire string, it becomes 001001.

Your task is to transform the string a into b in at most $2n$ operations. It can be proved that it is always possible.

Input

The first line contains a single integer t $(1 \le t \le 1000)$ — the number of test cases. Next $3t$ lines contain descriptions of test cases.

The first line of each test case contains a single integer n $(1 \le n \le 10^5)$ — the length of the binary strings.

The next two lines contain two binary strings a and b of length n.

It is guaranteed that the sum of n across all test cases does not exceed 10^5.

Output

For each test case, output an integer k ($0 \le k \le 2n$), followed by k integers $p_1, \ldots, p_k$ ($1 \le p_i \le n$). Here k is the number of operations you use and p_i is the length of the prefix you flip in the i-th operation.

Example

input

```
5
2
01
10
5
01011
11100
2
01
01
10
0110011011
1000110100
1
0
1
```

output

```
3 1 2 1
6 5 2 5 3 1 2
0
9 4 1 2 10 4 1 2 1 5
1 1
```

Note

In the first test case, we have $01 \to 11 \to 00 \to 10$.

In the second test case, we have $01011 \to 00101 \to 11101 \to 01000 \to 10100 \to 00100 \to 11100$.

In the third test case, the strings are already the same. Another solution is to flip the prefix of length 2, which will leave a unchanged.

D. Unmerge

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

Let a and b be two arrays of lengths n and m, respectively, with no elements in common. We can define a new array $\text{merge}(a, b)$ of length $n + m$ recursively as follows:

- If one of the arrays is empty, the result is the other array. That is, $\text{merge}(\emptyset, b) = b$ and $\text{merge}(a, \emptyset) = a$. In particular, $\text{merge}(\emptyset, \emptyset) = \emptyset$.
- If both arrays are non-empty, and $a_1 < b_1$, then $\text{merge}(a, b) = [a_1] + \text{merge}([a_2, \ldots, a_n], b)$. That is, we delete the first element a_1 of a, merge the remaining arrays, then add a_1 to the beginning of the result.
- If both arrays are non-empty, and $a_1 > b_1$, then $\text{merge}(a, b) = [b_1] + \text{merge}(a, [b_2, \ldots, b_m])$. That is, we delete the first element b_1 of b, merge the remaining arrays, then add b_1 to the beginning of the result.

This algorithm has the nice property that if a and b are sorted, then $\text{merge}(a, b)$ will also be sorted. For example, it is used as a subroutine in merge-sort. For this problem, however, we will consider the same procedure acting on non-sorted arrays as well. For example, if $a = [3, 1]$ and $b = [2, 4]$, then $\text{merge}(a, b) = [2, 3, 1, 4]$.

A permutation is an array consisting of n distinct integers from 1 to n in arbitrary order. For example, $[2, 3, 1, 5, 4]$ is a permutation, but $[1, 2, 2]$ is not a permutation (2 appears twice in the array) and $[1, 3, 4]$ is also not a permutation ($n = 3$ but there is 4 in the array).

There is a permutation p of length $2n$. Determine if there exist two arrays a and b, each of length n and with no elements in common, so that $p = \text{merge}(a, b)$.

Input

The first line contains a single integer t ($1 \le t \le 1000$) — the number of test cases. Next $2t$ lines contain descriptions of test cases.

The first line of each test case contains a single integer n ($1 \le n \le 2000$).

The second line of each test case contains $2n$ integers $p_1, \ldots, p_{2n}$ ($1 \le p_i \le 2n$). It is guaranteed that p is a permutation.

It is guaranteed that the sum of n across all test cases does not exceed 2000.

Output

For each test case, output "YES" if there exist arrays a, b, each of length n and with no common elements, so that $p = \text{merge}(a, b)$. Otherwise, output "NO".

Example

input

```
6
2
2 3 1 4
2
3 1 2 4
4
3 2 6 1 5 7 8 4
3
1 2 3 4 5 6
4
6 1 3 7 4 5 8 2
6
4 3 2 5 1 11 9 12 8 6 10 7
```

output

```
YES
NO
YES
YES
NO
NO
```

Note

In the first test case, $[2, 3, 1, 4] = \text{merge}([3, 1], [2, 4])$.

In the second test case, we can show that $[3, 1, 2, 4]$ is not the merge of two arrays of length 2.

In the third test case, $[3, 2, 6, 1, 5, 7, 8, 4] = \text{merge}([3, 2, 8, 4], [6, 1, 5, 7])$.

In the fourth test case, $[1, 2, 3, 4, 5, 6] = \text{merge}([1, 3, 6], [2, 4, 5])$, for example.

E. Mastermind

time limit per test: 1 second

memory limit per test: 256 megabytes

input: standard input

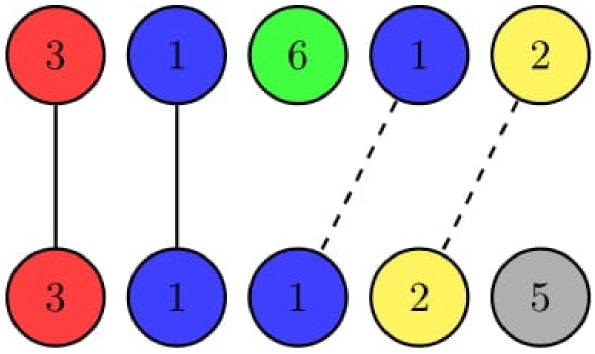
output: standard output

In the game of Mastermind, there are two players — Alice and Bob. Alice has a secret code, which Bob tries to guess. Here, a code is defined as a sequence of n colors. There are exactly $n+1$ colors in the entire universe, numbered from 1 to $n+1$ inclusive.

When Bob guesses a code, Alice tells him some information about how good of a guess it is, in the form of two integers x and y.

The first integer x is the number of indices where Bob's guess correctly matches Alice's code. The second integer y is the size of the intersection of the two codes as multisets. That is, if Bob were to change the order of the colors in his guess, y is the maximum number of indices he could get correct.

For example, suppose $n = 5$, Alice's code is $[3, 1, 6, 1, 2]$, and Bob's guess is $[3, 1, 1, 2, 5]$. At indices 1 and 2 colors are equal, while in the other indices they are not equal. So $x = 2$. And the two codes have the four colors $1, 1, 2, 3$ in common, so $y = 4$.



Solid lines denote a matched color for the same index. Dashed lines denote a matched color at a different index. x is the number of solid lines, and y is the total number of lines.

You are given Bob's guess and two values x and y. Can you find one possibility of Alice's code so that the values of x and y are correct?

Input

The first line contains a single integer t $(1 \le t \le 1000)$ — the number of test cases. Next $2t$ lines contain descriptions of test cases.

The first line of each test case contains three integers n, x, y $(1 \le n \le 10^5, 0 \le x \le y \le n)$ — the length of the codes, and two values Alice responds with.

The second line of each test case contains n integers $b_1, \ldots, b_n$ $(1 \le b_i \le n+1)$ — Bob's guess, where b_i is the i-th color of the guess.

It is guaranteed that the sum of n across all test cases does not exceed 10^5.

Output

For each test case, on the first line, output "YES" if there is a solution, or "NO" if there is no possible secret code consistent with the described situation. You can print each character in any case (upper or lower).

If the answer is "YES", on the next line output n integers $a_1, \ldots, a_n$ $(1 \le a_i \le n+1)$ — Alice's secret code, where a_i is the i-th color of the code.

If there are multiple solutions, output any.

Example

input

```
7
5 2 4
3 1 1 2 5
5 3 4
1 1 2 1 2
4 0 4
5 5 3 3
4 1 4
2 3 2 3
6 1 2
3 2 1 1 1 1
6 2 4
3 3 2 1 1 1
6 2 6
1 1 3 2 1 1
```

output

```
YES
3 1 6 1 2
YES
3 1 1 1 2
YES
3 3 5 5
NO
YES
4 4 4 4 3 1
YES
3 1 3 1 7 7
YES
2 3 1 1 1 1
```

Note

The first test case is described in the statement.

In the second test case, $x = 3$ because the colors are equal at indices $2, 4, 5$. And $y = 4$ because they share the colors $1, 1, 1, 2$.

In the third test case, $x = 0$ because there is no index where the colors are the same. But $y = 4$ because they share the colors $3, 3, 5, 5$.

In the fourth test case, it can be proved that no solution exists.