A. Everyone Loves to Sleep

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

Vlad, like everyone else, loves to sleep very much.

Every day Vlad has to do n things, each at a certain time. For each of these things, he has an alarm clock set, the i-th of them is triggered on h_i hours m_i minutes every day ($0 \le h_i < 24, 0 \le m_i < 60$). Vlad uses the 24-hour time format, so after $h = 12, m = 59$ comes $h = 13, m = 0$ and after $h = 23, m = 59$ comes $h = 0, m = 0$.

This time Vlad went to bed at H hours M minutes ($0 \le H < 24, 0 \le M < 60$) and asks you to answer: how much he will be able to sleep until the next alarm clock.

If any alarm clock rings at the time when he went to bed, then he will sleep for a period of time of length 0.

Input

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

The first line of the case contains three integers n, H and M ($1 \le n \le 10, 0 \le H < 24, 0 \le M < 60$) — the number of alarms and the time Vlad went to bed.

The following n lines contain two numbers each h_i and m_i ($0 \le h_i < 24, 0 \le m_i < 60$) — the time of the i alarm. It is acceptable that two or more alarms will trigger at the same time.

Numbers describing time do not contain leading zeros.

Output

Output t lines, each containing the answer to the corresponding test case. As an answer, output two numbers — the number of hours and minutes that Vlad will sleep, respectively. If any alarm clock rings at the time when he went to bed, the answer will be `0 0`.

Example

input

```
3
1 6 13
8 0
3 6 0
12 30
14 45
6 0
2 23 35
20 15
10 30
```

output

```
1 47
0 0
10 55
```

B. Remove Prefix

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

Polycarp was presented with some sequence of integers a of length n ($1 \le a_i \le n$). A sequence can make Polycarp happy only if it consists of **different** numbers (i.e. distinct numbers).

In order to make his sequence like this, Polycarp is going to make some (possibly zero) number of moves.

In one move, he can:

- remove the first (leftmost) element of the sequence.

For example, in one move, the sequence $[3, 1, 4, 3]$ will produce the sequence $[1, 4, 3]$, which consists of different numbers.

Determine the minimum number of moves he needs to make so that in the remaining sequence all elements are different. In other words, find the length of the smallest prefix of the given sequence a, after removing which all values in the sequence will be unique.

Input

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

Each test case consists of two lines.

The first line contains an integer n ($1 \le n \le 2 \cdot 10^5$) — the length of the given sequence a.

The second line contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le n$) — elements of the given sequence a.

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

Output

For each test case print your answer on a separate line — the minimum number of elements that must be removed from the beginning of the sequence so that all remaining elements are different.

Example

input

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

output

```
1
4
0
0
5
```

Note

The following are the sequences that will remain after the removal of prefixes:

- $[1, 4, 3]$;
- $[1]$;
- $[1]$;
- $[6, 5, 4, 3, 2, 1]$;
- $[2, 1]$.

It is easy to see that all the remaining sequences contain only distinct elements. In each test case, the shortest matching prefix was removed.

C. Minimum Varied Number

time limit per test: 1 second

memory limit per test: 256 megabytes

input: standard input

output: standard output

Find the minimum number with the given sum of digits s such that **all** digits in it are distinct (i.e. all digits are unique).

For example, if $s = 20$, then the answer is 389. This is the minimum number in which all digits are different and the sum of the digits is 20 ($3 + 8 + 9 = 20$).

For the given s print the required number.

Input

The first line contains an integer t ($1 \le t \le 45$) — the number of test cases.

Each test case is specified by a line that contains the only integer s ($1 \le s \le 45$).

Output

Print t integers — the answers to the given test cases.

Example

input

```
4
20
8
45
10
```

output

```
389
8
123456789
19
```

D. Color with Occurrences

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

You are given some text t and a set of n strings $s_1, s_2, \dots, s_n$.

In one step, you can choose any occurrence of any string s_i in the text t and color the corresponding characters of the text in red. For example, if $t =$ **bababa** and $s_1 =$ **ba**, $s_2 =$ **aba**, you can get $t =$ **bababa**, $t =$ **bababa** or $t =$ **bababa** in one step.

You want to color all the letters of the text t in red. When you color a letter in red again, it stays red.

In the example above, three steps are enough:

- Let's color $t[2 \dots 4] = s_2 =$ **aba** in red, we get $t =$ **bababa**;
- Let's color $t[1 \dots 2] = s_1 =$ **ba** in red, we get $t =$ **bababa**;
- Let's color $t[4 \dots 6] = s_2 =$ **aba** in red, we get $t =$ **bababa**.

Each string s_i can be applied any number of times (or not at all). Occurrences for coloring can intersect arbitrarily.

Determine the minimum number of steps needed to color all letters t in red and how to do it. If it is impossible to color all letters of the text t in red, output -1.

Input

The first line of the input contains an integer q $(1 \le q \le 100)$ —the number of test cases in the test.

The descriptions of the test cases follow.

The first line of each test case contains the text t $(1 \le |t| \le 100)$, consisting only of lowercase Latin letters, where $|t|$ is the length of the text t.

The second line of each test case contains a single integer n $(1 \le n \le 10)$ — the number of strings in the set.

This is followed by n lines, each containing a string s_i $(1 \le |s_i| \le 10)$ consisting only of lowercase Latin letters, where $|s_i|$ — the length of string s_i.

Output

For each test case, print the answer on a separate line.

If it is impossible to color all the letters of the text in red, print a single line containing the number -1.

Otherwise, on the first line, print the number m — the minimum number of steps it will take to turn all the letters t red.

Then in the next m lines print pairs of indices: w_j and p_j $(1 \le j \le m)$, which denote that the string with index w_j was used as a substring to cover the occurrences starting in the text t from position p_j. The pairs can be output in any order.

If there are several answers, output any of them.

Example

input

```
6
bababa
2
ba
aba
caba
2
bac
acab
abacabaca
3
aba
bac
aca
baca
3
a
c
b
codeforces
4
def
code
efo
```

```
forces
aaaabbbbcccceeee
4
eeee
cccc
aaaa
bbbb
```

output

```
3
2 2
1 1
2 4
-1
4
1 1
2 6
3 3
3 7
4
3 1
1 2
2 3
1 4
2
4 5
2 1
4
3 1
4 5
2 9
1 13
```

Note

The first test case is explained in the problem statement.

In the second test case, it is impossible to color all the letters of the text in red.

E. Add Modulo 10

time limit per test: 2 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

You are given an array of n integers $a_1, a_2, \dots, a_n$

You can apply the following operation an arbitrary number of times:

- select an index i ($1 \le i \le n$) and replace the value of the element a_i with the value $a_i + (a_i \bmod 10)$, where $a_i \bmod 10$ is the remainder of the integer dividing a_i by 10.

For a single index (value i), this operation can be applied multiple times. If the operation is applied repeatedly to the same index, then the current value of a_i is taken into account each time. For example, if $a_i = 47$ then after the first operation we get $a_i = 47 + 7 = 54$, and after the second operation we get $a_i = 54 + 4 = 58$.

Check if it is possible to make **all** array elements equal by applying multiple (possibly zero) operations.

For example, you have an array $[6, 11]$.

- Let's apply this operation to the first element of the array. Let's replace $a_1 = 6$ with $a_1 + (a_1 \bmod 10) = 6 + (6 \bmod 10) = 6 + 6 = 12$. We get the array $[12, 11]$.
- Then apply this operation to the second element of the array. Let's replace $a_2 = 11$ with $a_2 + (a_2 \bmod 10) = 11 + (11 \bmod 10) = 11 + 1 = 12$. We get the array $[12, 12]$.

Thus, by applying 2 operations, you can make all elements of an array equal.

Input

The first line contains one integer t ($1 \le t \le 10^4$) — the number of test cases. What follows is a description of each test case.

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

The second line of each test case contains n integers a_i ($0 \le a_i \le 10^9$) — array elements.

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

Output

For each test case print:

- YES if it is possible to make all array elements equal;
- NO otherwise.

You can print YES and NO in any case (for example, the strings yEs, yes, Yes and YES will be recognized as a positive answer) .

Example

input

```
10
2
6 11
3
2 18 22
5
5 10 5 10 5
4
1 2 4 8
2
4 5
3
93 96 102
2
40 6
2
50 30
2
22 44
2
1 5
```

output

```
Yes
No
Yes
Yes
No
Yes
No
No
Yes
No
```

Note

The first test case is clarified above.

In the second test case, it is impossible to make all array elements equal.

In the third test case, you need to apply this operation once to all elements equal to 5.

In the fourth test case, you need to apply this operation to all elements until they become equal to 8.

In the fifth test case, it is impossible to make all array elements equal.

In the sixth test case, you need to apply this operation to all elements until they become equal to 102.

F. Build a Tree and That Is It

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

A tree is a connected undirected graph without cycles. Note that in this problem, we are talking about not rooted trees.

You are given four positive integers n, d_{12}, d_{23} and d_{31}. Construct a tree such that:

- it contains n vertices numbered from 1 to n,
- the distance (length of the shortest path) from vertex 1 to vertex 2 is d_{12},
- distance from vertex 2 to vertex 3 is d_{23},
- the distance from vertex 3 to vertex 1 is d_{31}.

Output any tree that satisfies all the requirements above, or determine that no such tree exists.

Input

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

This is followed by t test cases, each written on a separate line.

Each test case consists of four positive integers n, d_{12}, d_{23} and d_{31} ($3 \leq n \leq 2 \cdot 10^5; 1 \leq d_{12}, d_{23}, d_{31} \leq n - 1$).

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

Output

For each test case, print YES if the suitable tree exists, and NO otherwise.

If the answer is positive, print another $n - 1$ line each containing a description of an edge of the tree — a pair of positive integers x_i, y_i, which means that the ith edge connects vertices x_i and y_i.

The edges and vertices of the edges can be printed in any order. If there are several suitable trees, output any of them.

Example

input

```
9
5 1 2 1
5 2 2 2
5 2 2 3
5 2 2 4
5 3 2 3
4 2 1 1
4 3 1 1
4 1 2 3
7 1 4 1
```

output

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

G. Path Prefixes

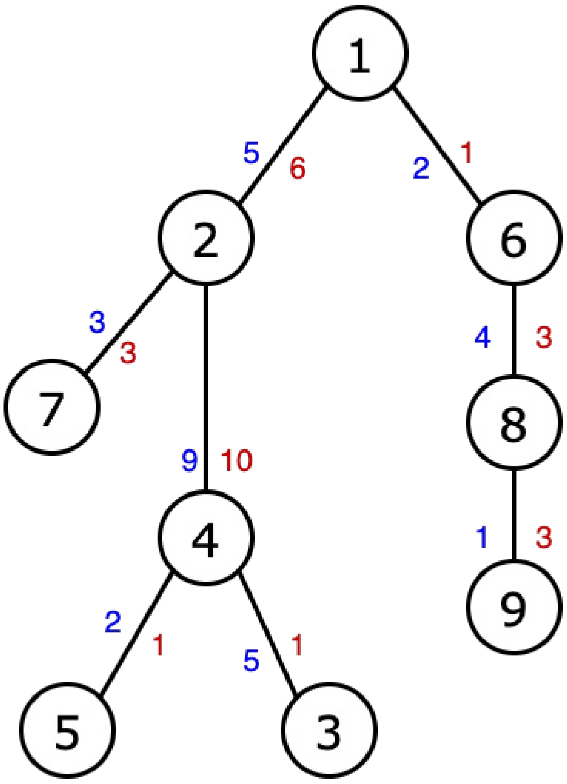
time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a rooted tree. It contains n vertices, which are numbered from 1 to n. The root is the vertex 1.

Each edge has two positive integer values. Thus, two positive integers a_j and b_j are given for each edge.

Output $n-1$ numbers $r_2, r_3, \dots, r_n$, where r_i is defined as follows.

Consider the path from the root (vertex 1) to i ($2 \le i \le n$). Let the sum of the costs of a_j along this path be A_i. Then r_i is equal to the length of the maximum prefix of this path such that the sum of b_j along this prefix does not exceed A_i.



Example for $n = 9$. The blue color shows the costs of a_j, and the red color shows the costs of b_j.

Consider an example. In this case:

- $r_2 = 0$, since the path to 2 has an amount of a_j equal to 5, only the prefix of this path of length 0 has a smaller or equal amount of b_j;
- $r_3 = 3$, since the path to 3 has an amount of a_j equal to $5 + 9 + 5 = 19$, the prefix of length 3 of this path has a sum of b_j equal to $6 + 10 + 1 = 17$ (the number is $17 \le 19$);
- $r_4 = 1$, since the path to 4 has an amount of a_j equal to $5 + 9 = 14$, the prefix of length 1 of this path has an amount of b_j equal to 6 (this is the longest suitable prefix, since the prefix of length 2 already has an amount of b_j equal to $6 + 10 = 16$, which is more than 14);
- $r_5 = 2$, since the path to 5 has an amount of a_j equal to $5 + 9 + 2 = 16$, the prefix of length 2 of this path has a sum of b_j equal to $6 + 10 = 16$ (this is the longest suitable prefix, since the prefix of length 3 already has an amount of b_j equal to $6 + 10 + 1 = 17$, what is more than 16);
- $r_6 = 1$, since the path up to 6 has an amount of a_j equal to 2, the prefix of length 1 of this path has an amount of b_j equal to 1;
- $r_7 = 1$, since the path to 7 has an amount of a_j equal to $5 + 3 = 8$, the prefix of length 1 of this path has an amount of b_j equal to 6 (this is the longest suitable prefix, since the prefix of length 2 already has an amount of b_j equal to $6 + 3 = 9$, which is more than 8);
- $r_8 = 2$, since the path up to 8 has an amount of a_j equal to $2 + 4 = 6$, the prefix of length 2 of this path has an amount of b_j equal to $1 + 3 = 4$;
- $r_9 = 3$, since the path to 9 has an amount of a_j equal to $2 + 4 + 1 = 7$, the prefix of length 3 of this path has a sum of b_j equal to $1 + 3 + 3 = 7$.

Input

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

The descriptions of test cases follow.

Each description begins with a line that contains an integer n ($2 \le n \le 2 \cdot 10^5$) — the number of vertices in the tree.

This is followed by $n - 1$ string, each of which contains three numbers p_j, a_j, b_j ($1 \le p_j \le n$; $1 \le a_j, b_j \le 10^9$) — the ancestor of the vertex j, the first and second values an edge that leads from p_j to j. The value of j runs through all values from 2 to n inclusive. It is guaranteed that each set of input data has a correct hanged tree with a root at the vertex 1.

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

Output

For each test case, output $n - 1$ integer in one line: $r_2, r_3, \ldots, r_n$.

Example

input

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

```
2 3 3
6 4 3
8 1 3
4
1 1 100
2 1 1
3 101 1
4
1 100 1
2 1 1
3 1 101
10
1 1 4
2 3 5
2 5 1
3 4 3
3 1 5
5 3 5
5 2 1
1 3 2
6 2 1
```

output

```
0 3 1 2 1 1 2 3
0 0 3
1 2 2
0 1 2 1 1 2 2 1 1
```

Note

The first example is parsed in the statement.

In the second example:

- $r_2 = 0$, since the path to 2 has an amount of a_j equal to 1, only the prefix of this path of length 0 has a smaller or equal amount of b_j;
- $r_3 = 0$, since the path to 3 has an amount of a_j equal to $1 + 1 = 2$, the prefix of length 1 of this path has an amount of b_j equal to 100 ($100 > 2$);
- $r_4 = 3$, since the path to 4 has an amount of a_j equal to $1 + 1 + 101 = 103$, the prefix of length 3 of this path has an amount of b_j equal to 102, .