Educational Codeforces Round 85 (Rated for Div. 2)

A. Level Statistics

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

Polycarp has recently created a new level in this cool new game Berlio Maker 85 and uploaded it online. Now players from all over the world can try his level.

All levels in this game have two stats to them: the number of plays and the number of clears. So when a player attempts the level, the number of plays increases by 1. If he manages to finish the level successfully then the number of clears increases by 1 as well. **Note that both of the statistics update at the same time** (so if the player finishes the level successfully then the number of plays will increase at the same time as the number of clears).

Polycarp is very excited about his level, so he keeps peeking at the stats to know how hard his level turns out to be.

So he peeked at the stats n times and wrote down n pairs of integers — $(p_1, c_1), (p_2, c_2), \dots, (p_n, c_n)$, where p_i is the number of plays at the i-th moment of time and c_i is the number of clears at the same moment of time. **The stats are given in chronological order** (i.e. the order of given pairs is exactly the same as Polycarp has written down).

Between two consecutive moments of time Polycarp peeked at the stats many players (but possibly zero) could attempt the level.

Finally, Polycarp wonders if he hasn't messed up any records and all the pairs are correct. If there could exist such a sequence of plays (and clears, respectively) that the stats were exactly as Polycarp has written down, then he considers his records correct.

Help him to check the correctness of his records.

For your convenience you have to answer multiple independent test cases.

Input

The first line contains a single integer T $(1 \le T \le 500)$ — the number of test cases.

The first line of each test case contains a single integer n ($1 \le n \le 100$) — the number of moments of time Polycarp peeked at the stats.

Each of the next n lines contains two integers p_i and c_i ($0 \le p_i, c_i \le 1000$) — the number of plays and the number of clears of the level at the i-th moment of time.

Note that the stats are given in chronological order.

Output

For each test case print a single line.

If there could exist such a sequence of plays (and clears, respectively) that the stats were exactly as Polycarp has written down, then print "YES".

Otherwise, print "NO".

You can print each letter in any case (upper or lower).

Example

input

```
6
3
0 0
1 1
1 2
2
1 0
1000 3
4
10 1
15 2
10 2
15 2
1
765 432
2
4 4
4 3
5
0 0
1 0
```

```
1 0
1 0
1 0
```

output

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

Note

In the first test case at the third moment of time the number of clears increased but the number of plays did not, that couldn't have happened.

The second test case is a nice example of a Super Expert level.

In the third test case the number of plays decreased, which is impossible.

The fourth test case is probably an auto level with a single jump over the spike.

In the fifth test case the number of clears decreased, which is also impossible.

Nobody wanted to play the sixth test case; Polycarp's mom attempted it to make him feel better, however, she couldn't clear it.

B. Middle Class

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

Many years ago Berland was a small country where only n people lived. Each person had some savings: the i-th one had a_i burles.

The government considered a person as wealthy if he had at least x burles. To increase the number of wealthy people Berland decided to carry out several reforms. Each reform looked like that:

- the government chooses some subset of people (maybe all of them);
- the government takes all savings from the chosen people and redistributes the savings among the chosen people equally.

For example, consider the savings as list $[5, 1, 2, 1]$: if the government chose the 1-st and the 3-rd persons then it, at first, will take all $5 + 2 = 7$ burles and after that will return 3.5 burles to the chosen people. As a result, the savings will become $[3.5, 1, 3.5, 1]$.

A lot of data was lost from that time, so we don't know how many reforms were implemented and to whom. All we can do is ask you to calculate the maximum possible number of wealthy people after several (maybe zero) reforms.

Input

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

Next $2T$ lines contain the test cases — two lines per test case. The first line contains two integers n and x ($1 \le n \le 10^5$, $1 \le x \le 10^9$) — the number of people and the minimum amount of money to be considered as wealthy.

The second line contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le 10^9$) — the initial savings of each person.

It's guaranteed that the total sum of n doesn't exceed 10^5.

Output

Print T integers — one per test case. For each test case print the maximum possible number of wealthy people after several (maybe zero) reforms.

Example

input

```
4
4 3
5 1 2 1
4 10
11 9 11 9
2 5
4 3
3 7
9 4 9
```

output

```
2
4
0
3
```

Note

The first test case is described in the statement.

In the second test case, the government, for example, could carry out two reforms: $[\underline{11}, \underline{9}, 11, 9] \rightarrow [10, 10, \underline{11}, \underline{9}] \rightarrow [10, 10, 10, 10]$.

In the third test case, the government couldn't make even one person wealthy.

In the fourth test case, the government could choose all people to carry out a reform: $[\underline{9}, \underline{4}, \underline{9}] \rightarrow [7\frac{1}{3}, 7\frac{1}{3}, 7\frac{1}{3}]$.

C. Circle of Monsters

time limit per test: 1 second

memory limit per test: 256 megabytes

input: standard input

output: standard output

You are playing another computer game, and now you have to slay n monsters. These monsters are standing in a circle, numbered clockwise from 1 to n. Initially, the i-th monster has a_i health.

You may shoot the monsters to kill them. Each shot requires exactly one bullet and decreases the health of the targeted monster by 1 (deals 1 damage to it). Furthermore, when the health of some monster i becomes 0 or less than 0, it dies and explodes, dealing b_i damage to the next monster (monster $i + 1$, if $i < n$, or monster 1, if $i = n$). If the next monster is already dead, then nothing happens. If the explosion kills the next monster, it explodes too, damaging the monster after it and possibly triggering another explosion, and so on.

You have to calculate the minimum number of bullets you have to fire to kill all n monsters in the circle.

Input

The first line contains one integer T ($1 \le T \le 150000$) — the number of test cases.

Then the test cases follow, each test case begins with a line containing one integer n ($2 \le n \le 300000$) — the number of monsters. Then n lines follow, each containing two integers a_i and b_i ($1 \le a_i, b_i \le 10^{12}$) — the parameters of the i-th monster in the circle.

It is guaranteed that the total number of monsters in all test cases does not exceed 300000.

Output

For each test case, print one integer — the minimum number of bullets you have to fire to kill all of the monsters.

Example

input

```
1
3
7 15
2 14
5 3
```

output

```
6
```

D. Minimum Euler Cycle

time limit per test: 2 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

You are given a *complete directed* graph K_n with n vertices: each pair of vertices $u \neq v$ in K_n have both directed edges (u, v) and (v, u); there are no self-loops.

You should find such a cycle in K_n that visits every directed edge exactly once (allowing for revisiting vertices).

We can write such cycle as a list of $n(n-1)+1$ vertices $v_1, v_2, v_3, \ldots, v_{n(n-1)-1}, v_{n(n-1)}, v_{n(n-1)+1} = v_1$ — a visiting order, where each (v_i, v_{i+1}) occurs exactly once.

Find the **lexicographically smallest** such cycle. It's not hard to prove that the cycle always exists.

Since the answer can be too large print its $[l, r]$ segment, in other words, $v_l, v_{l+1}, \ldots, v_r$.

Input

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

Next T lines contain test cases — one per line. The first and only line of each test case contains three integers n, l and r ($2 \le n \le 10^5$, $1 \le l \le r \le n(n-1)+1$, $r - l + 1 \le 10^5$) — the number of vertices in K_n, and segment of the cycle to print.

It's guaranteed that the total sum of n doesn't exceed 10^5 and the total sum of $r - l + 1$ doesn't exceed 10^5.

Output

For each test case print the segment $v_l, v_{l+1}, \ldots, v_r$ of the lexicographically smallest cycle that visits every edge exactly once.

Example

input

```
3
2 1 3
3 3 6
99995 9998900031 9998900031
```

output

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

Note

In the second test case, the lexicographically minimum cycle looks like: $1, 2, 1, 3, 2, 3, 1$.

In the third test case, it's quite obvious that the cycle should start and end in vertex 1.

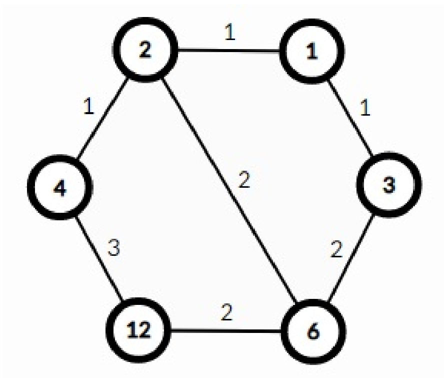
E. Divisor Paths

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

You are given a positive integer D. Let's build the following graph from it:

- each vertex is a divisor of D (not necessarily prime, 1 and D itself are also included);
- two vertices x and y ($x > y$) have an undirected edge between them if x is divisible by y and $\frac{x}{y}$ is a prime;
- the weight of an edge is the number of divisors of x that are not divisors of y.

For example, here is the graph for $D = 12$:



Edge $(4, 12)$ has weight 3 because 12 has divisors $[1, 2, 3, 4, 6, 12]$ and 4 has divisors $[1, 2, 4]$. Thus, there are 3 divisors of 12 that are not divisors of 4 — $[3, 6, 12]$.

There is no edge between 3 and 2 because 3 is not divisible by 2. There is no edge between 12 and 3 because $\frac{12}{3} = 4$ is not a prime.

Let the length of the path between some vertices v and u in the graph be the total weight of edges on it. For example, path $[(1, 2), (2, 6), (6, 12), (12, 4), (4, 2), (2, 6)]$ has length $1 + 2 + 2 + 3 + 1 + 2 = 11$. The empty path has length 0.

So the shortest path between two vertices v and u is the path that has the minimal possible length.

Two paths a and b are different if there is either a different number of edges in them or there is a position i such that a_i and b_i are different edges.

You are given q queries of the following form:

- v u — calculate the **number of the shortest paths** between vertices v and u.

The answer for each query might be large so print it modulo 998244353.

Input

The first line contains a single integer D ($1 \le D \le 10^{15}$) — the number the graph is built from.

The second line contains a single integer q ($1 \le q \le 3 \cdot 10^5$) — the number of queries.

Each of the next q lines contains two integers v and u ($1 \le v, u \le D$). It is guaranteed that D is divisible by both v and u (both v and u are divisors of D).

Output

Print q integers — for each query output the **number of the shortest paths** between the two given vertices modulo 998244353.

Examples

input

```
12
3
4 4
12 1
3 4
```

output

```
1
3
1
```

input

```
1
1
1 1
```

output

```
1
```

input

```
288807105787200
4
46 482955026400
12556830686400 897
414 12556830686400
4443186242880 325
```

output

```
547558588
277147129
457421435
702277623
```

Note

In the first example:

- The first query is only the empty path — length 0;
- The second query are paths $[(12,4),(4,2),(2,1)]$ (length $3+1+1=5$), $[(12,6),(6,2),(2,1)]$ (length $2+2+1=5$) and $[(12,6),(6,3),(3,1)]$ (length $2+2+1=5$).
- The third query is only the path $[(3,1),(1,2),(2,4)]$ (length $1+1+1=3$).

F. Strange Function

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

Let's denote the following function f. This function takes an array a of length n and returns an array. Initially the result is an empty array. For each integer i from 1 to n we add element a_i to the end of the resulting array if it is greater than all previous elements (more formally, if $a_i > \max\limits_{1 \le j < i} a_j$). Some examples of the function f:

1. if $a = [3,1,2,7,7,3,6,7,8]$ then $f(a) = [3,7,8]$;
2. if $a = [1]$ then $f(a) = [1]$;
3. if $a = [4,1,1,2,3]$ then $f(a) = [4]$;
4. if $a = [1,3,1,2,6,8,7,7,4,11,10]$ then $f(a) = [1,3,6,8,11]$.

You are given two arrays: array $a_1, a_2, \dots, a_n$ and array $b_1, b_2, \dots, b_m$. You can delete some elements of array a (possibly zero). To delete the element a_i, you have to pay p_i coins (the value of p_i can be negative, then you get $|p_i|$ coins, if you delete this element). Calculate the minimum number of coins (possibly negative) you have to spend for fulfilling equality $f(a) = b$.

Input

The first line contains one integer n $(1 \le n \le 5 \cdot 10^5)$ — the length of array a.

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

The third line contains n integers $p_1, p_2, \dots, p_n$ $(|p_i| \le 10^9)$ — the array p.

The fourth line contains one integer m $(1 \le m \le n)$ — the length of array b.

The fifth line contains m integers $b_1, b_2, \dots, b_m$ $(1 \le b_i \le n, b_{i-1} < b_i)$ — the array b.

Output

If the answer exists, in the first line print YES. In the second line, print the minimum number of coins you have to spend for fulfilling equality $f(a) = b$.

Otherwise in only line print `NO`.

Examples

input

```
11
4 1 3 3 7 8 7 9 10 7 11
3 5 0 -2 5 3 6 7 8 2 4
3
3 7 10
```

output

```
YES
20
```

input

```
6
2 1 5 3 6 5
3 -9 0 16 22 -14
4
2 3 5 6
```

output

```
NO
```

G. Substring Search

time limit per test: 1.25 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

You are given a permutation p consisting of exactly 26 integers from 1 to 26 (since it is a permutation, each integer from 1 to 26 occurs in p exactly once) and two strings s and t consisting of lowercase Latin letters.

A substring t' of string t is an **occurence** of string s if the following conditions are met:

1. $|t'| = |s|$;
2. for each $i \in [1, |s|]$, either $s_i = t'_i$, or $p_{idx(s_i)} = idx(t'_i)$, where $idx(c)$ is the index of character c in Latin alphabet ($idx(\text{a}) = 1$, $idx(\text{b}) = 2$, $idx(\text{z}) = 26$).

For example, if $p_1 = 2$, $p_2 = 3$, $p_3 = 1$, $s = \text{abc}$, $t = \text{abcaaba}$, then three substrings of t are occurences of s (they are $t' = \text{abc}$, $t' = \text{bca}$ and $t' = \text{aba}$).

For each substring of t having length equal to $|s|$, check if it is an **occurence** of s.

Input

The first line contains 26 integers p_1, p_2, ..., p_{26} ($1 \le p_i \le 26$, all these integers are pairwise distinct).

The second line contains one string s, and the third line contains one string t ($2 \le |s| \le |t| \le 2 \cdot 10^5$) both consisting of lowercase Latin letters.

Output

Print a string of $|t| - |s| + 1$ characters, each character should be either `0` or `1`. The i-th character should be `1` if and only if the substring of t starting with the i-th character and ending with the $(i + |s| - 1)$-th character (inclusive) is an **occurence** of s.

Example

input

```
2 3 1 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
abc
abcaaba
```

output

```
11001
```