A. Domino on Windowsill

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

You have a board represented as a grid with $2 \times n$ cells.

The first k_1 cells on the first row and first k_2 cells on the second row are colored in white. All other cells are colored in black.

You have w white dominoes (2×1 tiles, both cells are colored in white) and b black dominoes (2×1 tiles, both cells are colored in black).

You can place a white domino on the board if both board's cells are white and not occupied by any other domino. In the same way, you can place a black domino if both cells are black and not occupied by any other domino.

Can you place all $w + b$ dominoes on the board if you can place dominoes both horizontally and vertically?

Input

The first line contains a single integer t ($1 \le t \le 3000$) — the number of test cases.

The first line of each test case contains three integers n, k_1 and k_2 ($1 \le n \le 1000$; $0 \le k_1, k_2 \le n$).

The second line of each test case contains two integers w and b ($0 \le w, b \le n$).

Output

For each test case, print YES if it's possible to place all $w + b$ dominoes on the board and NO, otherwise.

You may print every letter in any case you want (so, for example, the strings yEs, yes, Yes and YES are all recognized as positive answer).

Example

input

```
5
1 0 1
1 0
1 1 1
0 0
3 0 0
1 3
4 3 1
2 2
5 4 3
3 1
```

output

```
NO
YES
NO
YES
YES
```

Note

In the first test case, $n = 1$, $k_1 = 0$ and $k_2 = 1$. It means that 2×1 board has black cell $(1, 1)$ and white cell $(2, 1)$. So, you can't place any white domino, since there is only one white cell.

In the second test case, the board of the same size 2×1, but both cell are white. Since $w = 0$ and $b = 0$, so you can place $0 + 0 = 0$ dominoes on the board.

In the third test case, board 2×3, but fully colored in black (since $k_1 = k_2 = 0$), so you can't place any white domino.

In the fourth test case, cells $(1, 1)$, $(1, 2)$, $(1, 3)$, and $(2, 1)$ are white and other cells are black. You can place 2 white dominoes at positions $((1, 1), (2, 1))$ and $((1, 2), (1, 3))$ and 2 black dominoes at positions $((1, 4), (2, 4))$ $((2, 2), (2, 3))$.

B. Binary Removals

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

You are given a string s, consisting only of characters '0' or '1'. Let $|s|$ be the length of s.

You are asked to choose some integer k ($k > 0$) and find a sequence a of length k such that:

- $1 \le a_1 < a_2 < \dots < a_k \le |s|$;
- $a_{i-1} + 1 < a_i$ for all i from 2 to k.

The characters at positions $a_1, a_2, \dots, a_k$ are removed, the remaining characters are concatenated without changing the order. So, in other words, the positions in the sequence a should not be adjacent.

Let the resulting string be s'. s' is called sorted if for all i from 2 to $|s'|$ $s'_{i-1} \le s'_i$.

Does there exist such a sequence a that the resulting string s' is sorted?

Input

The first line contains a single integer t ($1 \le t \le 1000$) — the number of testcases.

Then the descriptions of t testcases follow.

The only line of each testcase contains a string s ($2 \le |s| \le 100$). Each character is either '0' or '1'.

Output

For each testcase print "YES" if there exists a sequence a such that removing the characters at positions $a_1, a_2, \dots, a_k$ and concatenating the parts without changing the order produces a sorted string.

Otherwise, print "NO".

You may print every letter in any case you want (so, for example, the strings yEs, yes, Yes and YES are all recognized as positive answer).

Example

input

```
5
10101011011
0000
11111
110
1100
```

output

```
YES
YES
YES
YES
NO
```

Note

In the first testcase you can choose a sequence $a = [1, 3, 6, 9]$. Removing the underlined letters from "$\texttt{\underline{1}0\underline{1}01\underline{0}11\underline{0}11}$" will produce a string "0011111", which is sorted.

In the second and the third testcases the sequences are already sorted.

In the fourth testcase you can choose a sequence $a = [3]$. $s' =$ "11", which is sorted.

In the fifth testcase there is no way to choose a sequence a such that s' is sorted.

C. Minimum Grid Path

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

Let's say you are standing on the XY-plane at point $(0, 0)$ and you want to reach point (n, n).

You can move only in two directions:

- to the *right*, i. e. horizontally and in the direction that increase your x coordinate,
- or *up*, i. e. vertically and in the direction that increase your y coordinate.

In other words, your path will have the following structure:

- initially, you choose to go to the right or up;
- then you go some **positive integer** distance in the chosen direction (distances can be chosen independently);
- after that you change your direction (from right to up, or from up to right) and repeat the process.

You don't like to change your direction too much, so you will make no more than $n - 1$ direction changes.

As a result, your path will be a polygonal chain from $(0, 0)$ to (n, n), consisting of **at most** n line segments where each segment has positive integer length and vertical and horizontal segments alternate.

Not all paths are equal. You have n integers $c_1, c_2, \dots, c_n$ where c_i is the cost of the i-th segment.

Using these costs we can define the *cost of the path* as the sum of lengths of the segments of this path multiplied by their cost, i. e. if the path consists of k segments ($k \le n$), then the cost of the path is equal to $\sum\limits_{i=1}^{k} c_i \cdot length_i$ (segments are numbered from 1 to k in the order they are in the path).

Find the path of the minimum cost and print its cost.

Input

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

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

The second line of each test case contains n integers $c_1, c_2, \dots, c_n$ ($1 \le c_i \le 10^9$) — the costs of each segment.

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

Output

For each test case, print the minimum possible cost of the path from $(0, 0)$ to (n, n) consisting of at most n alternating segments.

Example

input

```
3
2
13 88
3
2 3 1
5
4 3 2 1 4
```

output

```
202
13
19
```

Note

In the first test case, to reach $(2, 2)$ you need to make at least one turn, so your path will consist of exactly 2 segments: one horizontal of length 2 and one vertical of length 2. The cost of the path will be equal to $2 \cdot c_1 + 2 \cdot c_2 = 26 + 176 = 202$.

In the second test case, one of the optimal paths consists of 3 segments: the first segment of length 1, the second segment of length 3 and the third segment of length 2.

The cost of the path is $1 \cdot 2 + 3 \cdot 3 + 2 \cdot 1 = 13$.

In the third test case, one of the optimal paths consists of 4 segments: the first segment of length 1, the second one — 1, the third one — 4, the fourth one — 4. The cost of the path is $1 \cdot 4 + 1 \cdot 3 + 4 \cdot 2 + 4 \cdot 1 = 19$.

D. The Number of Pairs

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

You are given three positive (greater than zero) integers c, d and x.

You have to find the number of pairs of positive integers (a, b) such that equality $c \cdot lcm(a, b) - d \cdot gcd(a, b) = x$ holds. Where $lcm(a, b)$ is the least common multiple of a and b and $gcd(a, b)$ is the greatest common divisor of a and b.

Input

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

Each test case consists of one line containing three integer c, d and x ($1 \le c, d, x \le 10^7$).

Output

For each test case, print one integer — the number of pairs (a, b) such that the above equality holds.

Example

input

```
4
1 1 3
4 2 6
3 3 7
2 7 25
```

output

```
4
3
0
8
```

Note

In the first example, the correct pairs are: $(1, 4)$, $(4, 1)$, $(3, 6)$, $(6, 3)$.

In the second example, the correct pairs are: $(1, 2)$, $(2, 1)$, $(3, 3)$.

E. Chaotic Merge

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

You are given two strings x and y, both consist only of lowercase Latin letters. Let $|s|$ be the length of string s.

Let's call a sequence a a merging sequence if it consists of exactly $|x|$ zeros and exactly $|y|$ ones in some order.

A merge z is produced from a sequence a by the following rules:

- if $a_i = 0$, then remove a letter from the beginning of x and append it to the end of z;
- if $a_i = 1$, then remove a letter from the beginning of y and append it to the end of z.

Two merging sequences a and b are different if there is some position i such that $a_i \neq b_i$.

Let's call a string z *chaotic* if for all i from 2 to $|z|$ $z_{i-1} \neq z_i$.

Let $s[l, r]$ for some $1 \leq l \leq r \leq |s|$ be a substring of consecutive letters of s, starting from position l and ending at position r inclusive.

Let $f(l_1, r_1, l_2, r_2)$ be the number of different merging sequences of $x[l_1, r_1]$ and $y[l_2, r_2]$ that produce *chaotic* merges. Note that only non-empty substrings of x and y are considered.

Calculate $\sum\limits_{\substack{1 \leq l_1 \leq r_1 \leq |x| \\ 1 \leq l_2 \leq r_2 \leq |y|}} f(l_1, r_1, l_2, r_2)$. Output the answer modulo $998\,244\,353$.

Input

The first line contains a string x ($1 \leq |x| \leq 1000$).

The second line contains a string y ($1 \leq |y| \leq 1000$).

Both strings consist only of lowercase Latin letters.

Output

Print a single integer — the sum of $f(l_1, r_1, l_2, r_2)$ over $1 \leq l_1 \leq r_1 \leq |x|$ and $1 \leq l_2 \leq r_2 \leq |y|$ modulo $998\,244\,353$.

Examples

input

```
aaa
bb
```

output

```
24
```

input

```
code
forces
```

output

```
1574
```

input

```
aaaaa
aaa
```

output

```
0
```

input

```
justamassivetesttocheck
howwellyouhandlemodulooperations
```

output

```
667387032
```

Note

In the first example there are:

- 6 pairs of substrings "a" and "b", each with valid merging sequences "`01`" and "`10`";
- 3 pairs of substrings "a" and "bb", each with a valid merging sequence "`101`";
- 4 pairs of substrings "aa" and "b", each with a valid merging sequence "`010`";
- 2 pairs of substrings "aa" and "bb", each with valid merging sequences "`0101`" and "`1010`";
- 2 pairs of substrings "aaa" and "b", each with no valid merging sequences;
- 1 pair of substrings "aaa" and "bb" with a valid merging sequence "`01010`";

Thus, the answer is $6 \cdot 2 + 3 \cdot 1 + 4 \cdot 1 + 2 \cdot 2 + 2 \cdot 0 + 1 \cdot 1 = 24$.

F. Diameter Cuts

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

You are given an integer k and an undirected tree, consisting of n vertices.

The length of a simple path (a path in which each vertex appears at most once) between some pair of vertices is the number of edges in this path. A diameter of a tree is the maximum length of a simple path between all pairs of vertices of this tree.

You are about to remove a set of edges from the tree. The tree splits into multiple smaller trees when the edges are removed. The set of edges is valid if all the resulting trees have diameter less than or equal to k.

Two sets of edges are different if there is an edge such that it appears in only one of the sets.

Count the number of valid sets of edges modulo $998\,244\,353$.

Input

The first line contains two integers n and k ($2 \le n \le 5000$, $0 \le k \le n-1$) — the number of vertices of the tree and the maximum allowed diameter, respectively.

Each of the next $n-1$ lines contains a description of an edge: two integers v and u ($1 \le v, u \le n$, $v \ne u$).

The given edges form a tree.

Output

Print a single integer — the number of valid sets of edges modulo $998\,244\,353$.

Examples

input

```
4 3
1 2
1 3
1 4
```

output

```
8
```

input

```
2 0
1 2
```

output

```
1
```

input

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

output

```
25
```

input

```
6 3
1 2
```

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

output

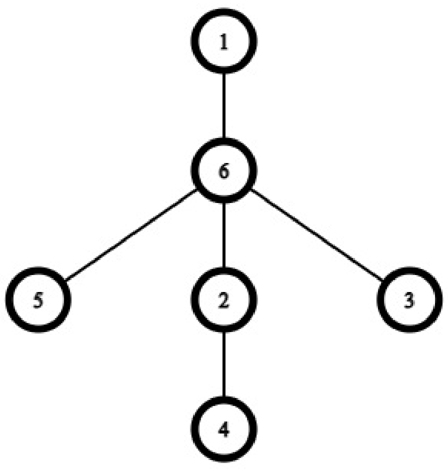
```
29
```

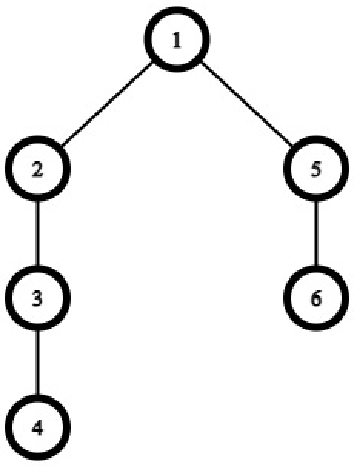
Note

In the first example the diameter of the given tree is already less than or equal to k. Thus, you can choose any set of edges to remove and the resulting trees will have diameter less than or equal to k. There are 2^3 sets, including the empty one.

In the second example you have to remove the only edge. Otherwise, the diameter will be 1, which is greater than 0.

Here are the trees for the third and the fourth examples:





G. Graph Coloring

time limit per test: 7 seconds
memory limit per test: 1024 megabytes
input: standard input
output: standard output

You are given a bipartite graph consisting of n_1 vertices in the first part, n_2 vertices in the second part, and m edges, numbered from 1 to m. You have to color each edge into one of two colors, red and blue. You have to minimize the following value: $\sum\limits_{v \in V} |r(v) - b(v)|$, where V is the set of vertices of the graph, $r(v)$ is the number of red edges incident to v, and $b(v)$ is the number of blue edges incident to v.

Sounds classical and easy, right? Well, you have to process q queries of the following format:

- $1\ v_1\ v_2$ — add a new edge connecting the vertex v_1 of the first part with the vertex v_2 of the second part. This edge gets a new index as follows: the first added edge gets the index $m+1$, the second — $m+2$, and so on. After adding the edge, you have to print the *hash* of the current optimal coloring (if there are multiple optimal colorings, print the *hash* of any of them). **Actually, this hash won't be verified, you may print any number as the answer to this query, but you may be asked to produce the coloring having this hash**;
- 2 — print the optimal coloring of the graph with the same *hash* you printed while processing the previous query. The query of this type will only be asked after a query of type 1, and there will be at most 10 queries of this type. If there are multiple optimal colorings corresponding to this *hash*, print any of them.

Note that if an edge was red or blue in some coloring, it may change its color in next colorings.

The *hash* of the coloring is calculated as follows: let R be the set of indices of red edges, then the *hash* is $(\sum\limits_{i \in R} 2^i) \bmod 998244353$.

Note that you should solve the problem in `online` mode. It means that you can't read the whole input at once. You can read each query only after writing the answer for the last query. Use functions `fflush` in C++ and `BufferedWriter.flush` in Java languages after each writing in your program.

Input

The first line contains three integers n_1, n_2 and m ($1 \le n_1, n_2, m \le 2 \cdot 10^5$).

Then m lines follow, the i-th of them contains two integers x_i and y_i ($1 \le x_i \le n_1$; $1 \le y_i \le n_2$) meaning that the i-th edge connects the vertex x_i from the first part and the vertex y_i from the second part.

The next line contains one integer q ($1 \le q \le 2 \cdot 10^5$) — the number of queries you have to process.

The next q lines contain the queries in the format introduced in the statement.

Additional constraints on the input:

- at any moment, the graph won't contain any multiple edges;
- the queries of type 2 are only asked if the previous query had type 1;
- there are at most 10 queries of type 2.

Output

To answer a query of type 1, print one integer — the *hash* of the optimal coloring.

To answer a query of type 2, print one line. It should begin with the integer k — the number of red edges. Then, k **distinct** integer should follow — the indices of **red** edges in your coloring, in any order. Each index should correspond to an existing edge, and the *hash* of the coloring you produce should be equal to the *hash* you printed as the answer to the previous query.

If there are multiple answers to a query, you may print any of them.

Example

input

```
3 4 2
1 2
3 4
10
1 1 3
1 2 3
2
1 3 3
2
1 2 4
2
1 2 1
1 1 1
2
```

output

```
8
8
1 3
40
2 3 5
104
3 5 6 3
104
360
4 5 6 3 8
```