# Problem A
# Angle Problem

Time limit: 2 seconds
Memory limit: 1024 megabytes

## Problem Description

You are given $n$ points $p_1, p_2, \ldots, p_n$ in 2D space. Each point $p_i$ has coordinates $(x_i, y_i)$. You are to process $q$ queries. Each query consists of four numbers: $L, R, X, Y$, where:

- $[L, R]$ denotes a subarray of points $p_L, p_{L+1}, \ldots, p_R$.
- $h = (X, Y)$ is a fixed observation point.

It is guaranteed that:

$$Y > \max\{y_1, y_2, \ldots, y_n\}.$$

Your task is to calculate the minimum angle (in degrees) that point $h$ must cover in order to see all points from $L$ to $R$ inclusive.

Formally, find the smallest angle $\theta$ (in degrees) so that if you are at point $h$ and start looking in one direction, turning your head by $\theta$ degrees will let you see all the points in $q_L, q_{L+1} ... q_R$.

## Input Format

The first line contains two integers $n$ and $q$.

The next $n$ lines each contain two integers $x_i$ and $y_i$, describing point $p_i$.

The next $q$ lines each contain four integers $L$, $R$, $X$, and $Y$, describing a query.

## Output Format

For each query, output the smallest angle $\theta$ in degrees.

The answer is considered correct if it has an absolute or relative error of at most $10^{-6}$.

## Technical Specification

- $1 \leq n, q \leq 10^5$
- $-10^9 \leq x_i, y_i \leq 10^9$ for $i \in [1, n]$
- $-10^9 \leq X, Y \leq 10^9$
- $Y > \max\{y_1, \ldots, y_n\}$
- $1 \leq L \leq R \leq n$

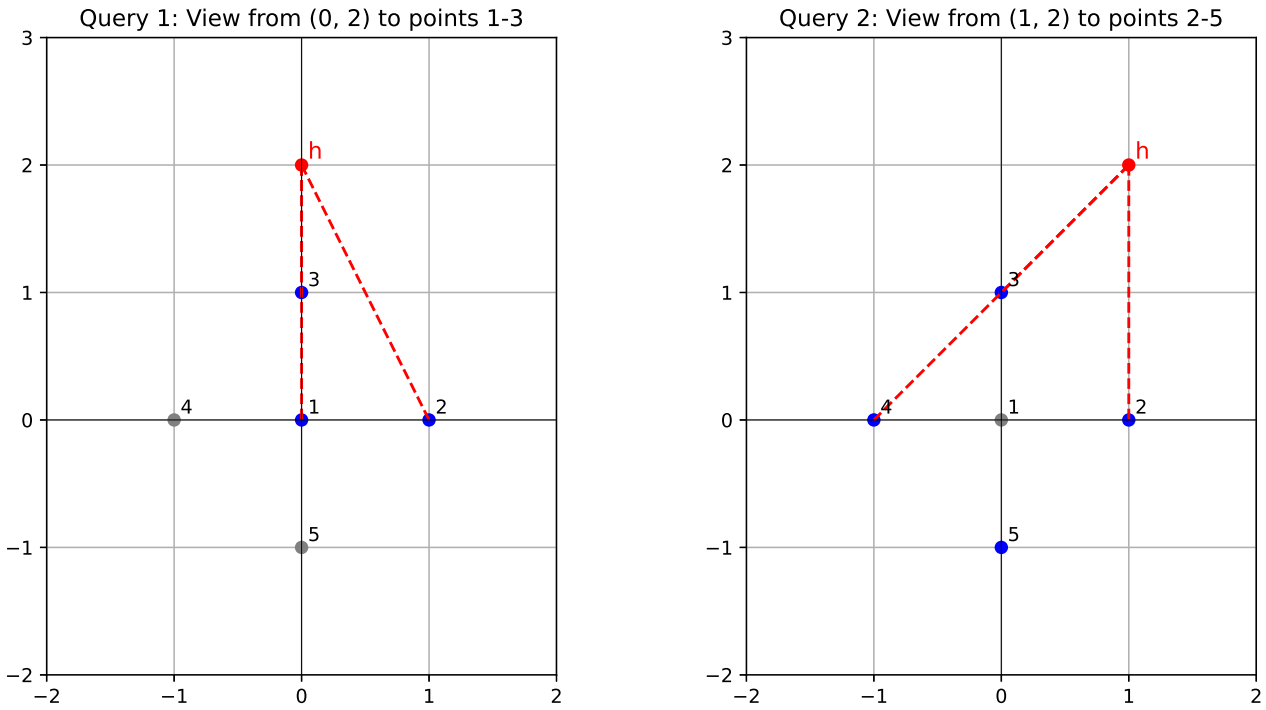
## Sample Input 1

```
5 2
0 0
1 0
0 1
-1 0
0 -1
1 3 0 2
2 5 1 2
```

## Sample Output 1

```
26.565051177
45.000000000
```

## Note




Sample Input 1

# Problem B
# Binary Palindromes

Time limit: 2 seconds
Memory limit: 1024 megabytes

## Problem Description

You are given two binary strings $s$ and $t$, where $|s| = n$ and $|t| = m$. You must perform the following process:

1. Choose an integer $k$ such that $1 \leq k \leq n$, and split the string $s$ into $k$ contiguous substrings $s_1, s_2, \ldots, s_k$. Each character of $s$ must belong to exactly one of these substrings. There are $2^{n-1}$ such ways to split $s$.
2. For each substring $s_i$ ($1 \leq i \leq k$), perform the following transformation:
   - If $s_i$ is a prefix of the string $t$, replace $s_i$ with the character `1`.
   - Otherwise, replace $s_i$ with the character `0`.
3. Concatenate the resulting characters (either `0` or `1` for each $s_i$) to form a new binary string $s' = s'_1 s'_2 \ldots s'_k$.

Your task is to count how many of the $2^{n-1}$ possible splits of $s$ will result in a string $s'$ that is a palindrome. Output the answer modulo $998244353$.

A binary string is called a palindrome if it reads the same forwards and backwards.

## Input Format

Each test contains multiple test cases. The first line contains the number of test cases $t$. The description of the test cases follows.

The first line of each test case contains two integers $n$ and $m$ —the length of the string $s$ and $t$ respectively.

The second line of each test case contains the binary string $s$.

The third line of each test case contains the binary string $t$.

## Output Format

For each test case, output a single integer denoting the number of possible splits of $s$ that result in a string $s'$ that is a palindrome. Output the answer modulo $998244353$.

## Technical Specification

- $1 \leq t \leq 2000$
- $1 \leq n \leq 2000$
- $1 \leq m \leq 2000$

- It is guaranteed that the sum of $n$ over all test cases does not exceed 2000.
- It is guaranteed that the sum of $m$ over all test cases does not exceed 2000.

## Sample Input 1

```
4
3 3
100
010
6 3
101101
101
2 3
01
001
4 1
1000
1
```

## Sample Output 1

```
2
15
1
4
```

# Problem C
# Chess Pieces

Time limit: 2 seconds
Memory limit: 1024 megabytes

## Problem Description

You are given three chess pieces placed on a two-dimensional plane, with initial positions $(x_1, y_1)$, $(x_2, y_2)$, and $(x_3, y_3)$, all distinct.

In one operation, you may select any piece and move it to a new position on the plane, subject to the following condition: the angle at the moved piece, formed by the lines connecting it to the other two pieces, must remain unchanged. The new position may have any real coordinates, as long as this angle condition is satisfied.

Formally, suppose a piece moves from $X$ to $X'$, while the other two pieces remain fixed at $A$ and $B$. Then the following must hold:

$$\angle AXB = \angle AX'B$$

where angles are measured as the smaller angle in $[0, \pi]$.

For example:

- For positions $(1, 0), (0, 1), (-1, 0)$, you can move the piece at $(0, 1)$ to $(0, -1)$, since both angles equal $\frac{\pi}{2}$.
- For positions $(1, 0), (0, 1), (-1, 0)$, you **cannot** move the piece at $(0, 1)$ to $(-1, -2)$, since the angle changes from $\frac{\pi}{2}$ to $\frac{\pi}{4}$.
- For positions $(1, 0), (0, 0), (-1, 0)$, you **cannot** move the piece at $(0, 0)$ to $(2, 0)$, since the angle changes from $\pi$ to $0$.

Given the target positions $(x'_1, y'_1)$, $(x'_2, y'_2)$, and $(x'_3, y'_3)$, determine whether it is possible to move the $i$-th piece to $(x'_i, y'_i)$ **simultaneously**, using any number of operations described above.

## Input Format

Each test contains multiple test cases. The first line contains the number of test cases $t$. The description of the test cases follows.

The first line of each test case contains six integers $x_1, y_1, x_2, y_2, x_3, y_3$.

The second line of each test case contains six integers $x'_1, y'_1, x'_2, y'_2, x'_3, y'_3$.

## Output Format

For each test case, output `Yes` if it is possible to move the $i$-th chess piece to $(x'_i, y'_i)$ at the same time after any number of operations mentioned above. Otherwise, output `No`.

## Technical Specification

- $1 \leq t \leq 10^4$
- $-100 \leq x_1, y_1, x_2, y_2, x_3, y_3 \leq 100$
- $-100 \leq x'_1, y'_1, x'_2, y'_2, x'_3, y'_3 \leq 100$
- Coordinates $(x_1, y_1)$, $(x_2, y_2)$, and $(x_3, y_3)$ are all distinct.
- Coordinates $(x'_1, y'_1)$, $(x'_2, y'_2)$, and $(x'_3, y'_3)$ are all distinct.

## Sample Input 1

```
4
1 0 0 1 -1 0
1 0 0 -1 -1 0
3 1 4 1 5 9
2 7 1 8 2 8
0 0 1 1 2 2
0 0 2 2 1 1
0 0 -1 3 1 5
10 10 13 9 9 5
```

## Sample Output 1

```
Yes
No
No
Yes
```

# Problem D
# Data Transmission

Time limit: 2 seconds
Memory limit: 1024 megabytes

## Problem Description

Consider the following network. There are $n$ nodes, labeled as 1 to $n$. In the network, each edge is undirected. For any two nodes, $u$ and $v$, there exists a unique path between $u$ and $v$. This implies that the network forms a **tree**.

There are some users who want to transmit data through the network. To transmit data from node $u$ to node $v$, all nodes along the path from $u$ to $v$ must be **activated**.

To limit the network load, we allow at most **two** users transmitting data simultaneously.

Given $q$ queries, each describing a set of users and their respective paths, your task is to compute the **number of distinct nodes that need to be activated** for each query.

## Input Format

The first line contains two integers $n$ and $q$, the number of nodes and queries.

The next $n-1$ lines each contain two integers $u$ and $v$ indicating that there exists an edge between $u$ and $v$.

The following $q$ lines each contain four integers $a$, $b$, $c$, and $d$, indicating that a user wants to transmit data from $a$ to $b$ and the other one wants to transmit data from $c$ to $d$.

## Output Format

For each query, output a single integer. Each line represents the number of nodes that need to be active.

## Technical Specification

- $1 \leq n, q \leq 10^5$
- $1 \leq u, v, a, b, c, d \leq n$

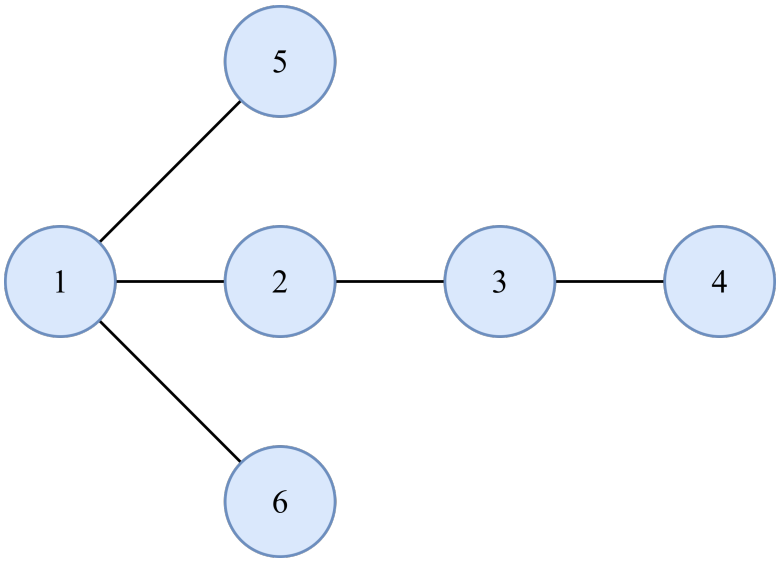
## Sample Input 1

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

## Sample Output 1

```
6
4
5
```

## Note




Sample Input 1

In first query:
Path $6 \to 5$ goes through nodes $6, 1, 5$.
Path $2 \to 4$ goes through nodes $2, 3, 4$.
We need to activate nodes $\{1, 2, 3, 4, 5, 6\}$.

In second query:
Path $1 \to 6$ goes through nodes $1, 6$.
Path $2 \to 5$ goes through nodes $2, 1, 5$.
We need to activate nodes $\{1, 2, 5, 6\}$.

In third query:
Path $4 \to 1$ goes through nodes $4, 3, 2, 1$.
Path $3 \to 6$ goes through nodes $3, 2, 1, 6$.
We need to activate nodes $\{1, 2, 3, 4, 6\}$.

# Problem E
# Echoes on the Endless Line

Time limit: 2 seconds
Memory limit: 1024 megabytes

## Problem Description

On an endless number line called the Meridian, humanity has retreated into scattered watchposts after a century of twilight. From the east blow cold *echoes*—not sounds, but drifting distortions that birth *Evil Entities*. These evil entities do not think or sleep; they drift, they feed, and they are strangely faithful to the Meridian, choosing positions along the line like stars pinned to a single axis.

To survive, the Watchers of the Meridian have trained their senses and their courage separately. Sight is not the same as valor. Each Watcher stationed has a field of sight: any evil entity within distance $R$ flickers into view as a shimmering silhouette. Yet only those who steel themselves within distance $L$ can step forward to strike. The elders crafted this tradition after learning a cruel lesson: seeing too far without the will to fight breaks the heart — but fighting beyond one's reach breaks the line.

A convoy moves nightly, stationing $q$ Watchers at their posts for a single hour before the world slides one notch darker. The *Evil Entities*, $n$ in total this cycle, have already taken their places at coordinates $a_1, \ldots, a_n$. The commander records two numbers for every Watcher: (1) how many evil entities they *see but cannot fight*, and (2) the *total distance* to those very entities. These two measures decide where to send reinforcements and how to ration courage: a Watcher who sees many untouchable threats may need a partner; one who is haunted by few but distant shapes might be reassigned to steadier ground.

Tonight, the convoy asks you to produce those numbers quickly and without error. The Meridian is long, the echoes are patient, and dawn is never guaranteed.

You are given $n$ positions of *evil entities* on a number line, and $q$ positions of *Watchers*. For each Watcher at position $b_j$:

- They can **see** any entity whose distance from $b_j$ is at most $R$.
- They can **fight** any entity whose distance from $b_j$ is at most $L$.

For each Watcher, compute:

1. the number of evil entities that are **seen but not fightable**, i.e. whose distance $d$ from $b_j$ satisfies $L < d \leq R$; and
2. the **sum of distances** to all such evil entities.

## Input Format

- The first line contains four integers $n$, $q$, $L$, and $R$ with $0 \leq L \leq R$.
- The second line contains $n$ integers $a_1, \ldots, a_n$—the positions of the evil entities.

- The third line contains $q$ integers $b_1, \ldots, b_q$—the positions of the Watcher.

## Output Format

Output $q$ lines. For each $j = 1, \ldots, q$, print two integers:

- the number of evil entities with distance in $(L, R]$ from $b_j$, and
- the sum of their distances to $b_j$.

Separate the two numbers by a single space.

## Technical Specification

- $1 \leq n, q \leq 2 \cdot 10^5$
- $0 \leq L < R \leq 10^9$
- $|a_i|, |b_j| \leq 10^9$
- The $a_i$ and $b_j$ are not necessarily distinct and may appear in any order.

## Sample Input 1

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

## Sample Output 1

```
1 5
1 5
```

## Sample Input 2

```
5 4 1 3
1 3 5 6 9
2 4 7 8
```

## Sample Output 2

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

## Note

**Distance Definition.** The distance between positions $x$ and $y$ on the line is $|x - y|$.

**Boundary Conventions.** "Within distance $T$" means distance $\leq T$. Therefore, "seen but not fightable" is the strict annulus $(L, R]$.

# Problem F
# Forbidden Spell Sequence

Time limit: 2 seconds
Memory limit: 1024 megabytes

## Problem Description

In the world of Arcana, spellcasters compose magical incantations using a fixed set of 8 ancient runes. These runes are labeled with the lowercase letters from `a` to `g`, and only these 7 characters may be used when constructing a spell.

A spell is defined as a sequence of $n$ runes, where each rune is one of the allowed characters from `a` to `g`, and runes may appear multiple times. That is, each spell is a string of length $n$ over the alphabet `{a, b, c, d, e, f, g}`.

However, some combinations of runes are unstable and must be avoided. The Magic Council has issued $m$ forbidden rules. Each rule specifies a group of $k_i$ distinct runes. A spell is considered invalid if it contains all of the runes specified in any forbidden rule.

In other words, for every forbidden rule, the constructed spell must omit at least one of the runes listed in that rule.

Your task is to determine the number of valid spells of length $n$ that satisfy all of the forbidden rules.

Since the answer may be large, output the result modulo $10^9 + 7$.

## Input Format

The first line contains two integers $n$ and $m$, representing the length of the spell and the number of forbidden rules.

Each of the next $m$ lines describes a forbidden rule. The $i$-th line starts with an integer $k_i$, followed by $k_i$ distinct lowercase letters $c_1, c_2, \ldots, c_{k_i}$, indicating that the spell must not contain all of the runes $c_1$ to $c_{k_i}$ simultaneously.

## Output Format

Print a single integer. This number represents the total number of valid spell sequences of length $n$ that do not violate any of the forbidden rules. Since the answer may be large, output it modulo $10^9 + 7$.

## Technical Specification

- $1 \leq n \leq 10^9$
- $0 \leq m < 2^7$
- $1 \leq k_i \leq 7$
- Each rune $c_i$ is a lowercase letter from `a` to `g`

## Sample Input 1

```
3 2
2 a b
3 b c d
```

## Sample Output 1

```
301
```

## Note

For the first test case:

**Some valid spells**

- `aac` (does not contain both a and b, and does not contain b, c, d together)
- `cef` (does not contain both a and b, and does not contain b, c, d together)
- `bef` (contains b but not a, and does not contain c and d together)

**Some invalid spells**

- `abe` (contains both a and b $\Rightarrow$ violates rule $\{a, b\}$)
- `bcd` (contains b, c, d $\Rightarrow$ violates rule $\{b, c, d\}$)
- `abd` (contains both a and b $\Rightarrow$ violates rule $\{a, b\}$)

This illustrates how to check each spell against the forbidden rules.

# Problem G
# Graph Orientation

Time limit: 2 seconds
Memory limit: 1024 megabytes

## Problem Description

You are given an undirected, simple, connected, bipartite graph $G$ with $n$ vertices and $m$ edges. The vertices are numbered 1 through $n$, and each vertex $i$ has a non-negative integer weight $w_i$.

An *orientation* of $G$ assigns a direction to each edge, producing a directed graph $\overrightarrow{G}$.

Let $T$ be the set of the vertices with an outdegree 0 in $\overrightarrow{G}$. For each vertex $i$ in $\overrightarrow{G}$, let $d_i$ be the minimum number of directed edges on a directed path from $i$ to any vertex in $T$. If no such path exists, set $d_i = 10^9$.

We then define the *cost* of $\overrightarrow{G}$ as

$$\text{cost}(\overrightarrow{G}) = \sum_{i=1}^{n} d_i \cdot w_i$$

Your task is is to determine an orientation of $G$ that minimizes $\text{cost}(\overrightarrow{G})$, and to output the resulting orientation of all edges.

## Input Format

The first line of each test case contains two integers $n$ and $m$ — the number of vertices and the number of edges of the graph.

The second line of each test case contains $n$ non-negative integers $w_1, w_2, \ldots, w_n$ — the weights of the vertices.

The $i$-th of the following $m$ lines contains two integers $u_i$ and $v_i$ — the ends of the $i$-th edge.

## Output Format

Output the answer in the following format:

Output an integer $c$ in the first line — the minimum total cost.

Then each of the following $m$ lines should contain two integers $u_i$ and $v_i$ — the directed edge $u_i \to v_i$.

You can output the directed edges in any order. If there are multiple ways to direct the edges with the cost minimized, you can output any of them.

## Technical Specification

- $1 \le n \le 100$
- $n - 1 \le m \le \frac{n(n-1)}{2}$
- $1 \le w_i \le 10^4$ for $i = 1, 2, \ldots, n$
- $1 \le u_i, v_i \le n$ and $u_i \ne v_i$ for $i = 1, 2, \ldots, m$.

- It is guaranteed that the given graph is undirected, simple, bipartite and connected.

## Sample Input 1

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

## Sample Output 1

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

## Sample Input 2

```
4 4
6 9 8 4
1 2
2 3
3 4
4 1
```

## Sample Output 2

```
13
2 1
2 3
4 3
4 1
```

# Problem H
# Huge Subsets

Time limit: 2 seconds
Memory limit: 1024 megabytes

## Problem Description

You are given an array of $n$ positive integers $a_1, a_2, \ldots, a_n$.

For each integer $k$ such that $1 \leq k \leq n$, define the multiset $S_k$ as the collection of all possible sums obtained by selecting exactly $k$ elements from the array $a$. That is, $S_k$ contains the sums of all $\binom{n}{k}$ subsets of size $k$ from $a$. For example, if $a = [1, 2, 3, 4]$, the sets $S_1$, $S_2$, $S_3$, and $S_4$ are:

- $S_1 = \{1, 2, 3, 4\}$
- $S_2 = \{3, 4, 5, 5, 6, 7\}$
- $S_3 = \{6, 7, 8, 9\}$
- $S_4 = \{10\}$

Your task is to find $\gcd(S_1), \gcd(S_2), \ldots, \gcd(S_n)$, where $\gcd(S)$ denotes the greatest common divisor (GCD) of all elements in multiset $S$. Specifically, if $S$ contains only a single element, then $\gcd(S)$ is the element itself.

## Input Format

Each test contains multiple test cases. The first line contains the number of test cases $t$. The description of the test cases follows.

The first line of each test case contains a single integer $n$ — the length of the array $a$.

The second line of each test case contains $n$ positive integers $a_1, a_2, \ldots, a_n$ — the elements of the array $a$.

## Output Format

For each test case, output $n$ integers in a new line — the value of $\gcd(S_1), \gcd(S_2), \ldots, \gcd(S_n)$ respectively.

## Technical Specification

- $1 \leq t \leq 10^4$
- $1 \leq n \leq 10^5$
- $1 \leq a_i \leq 10^{18}$
- It is guaranteed that the sum of $n$ over all test cases does not exceed $10^5$.

## Sample Input 1

```
3
4
```

## Sample Output 1

```
1 1 1 10
1000000000000000000
```

```
1 2 3 4
1
1000000000000000000
3
6 6 6
```

```
6 12 18
```

# Problem I
# Ice Sliding

Time limit: 2 seconds
Memory limit: 1024 megabytes

## Problem Description

You are given an $n \times m$ grid consisting of either empty ice cells (represented by `.`) or walls (represented by `#`). You will also be given $Q$ queries. In each query, you are given a starting cell and a target cell.

At the beginning of each query, you may choose to slide in any of the four directions (up, down, left, right). While moving on the ice, you will slide continuously in the chosen direction until you hit a wall (`#`) or the boundary of the grid. When you hit a wall or the boundary, you immediately turn $90°$ to the right (clockwise) and continue sliding in the new direction. This process repeats until you reach the target cell.

There are $Q$ queries to process. For each query, you are given a starting cell and a target cell. The player begins at the starting cell and may initially choose any of the four directions to move, stopping at the target cell. You need to compute the minimum number of cells required to reach the target, or $-1$ if it is impossible to reach. Note that if you pass through the same cell multiple times, each visit is counted separately. Turning itself does not count as a step.

## Input Format

The first line contains two integers $n, m$, representing the size of the grid. The following $n$ lines each contain $m$ characters, where each character is either `#` (a wall) or `.` (ice). The next line contains an integer $Q$, representing the number of queries. The following $Q$ lines each contain four integers $r_s$, $c_s$, $r_t$, $c_t$, representing the row and column of the starting cell, and the row and column of the target cell.

## Output Format

For each query, output a single integer - the number of steps (cells traversed, including the starting cell) it takes to reach the destination cell for the first time. If it is impossible to reach the destination, output $-1$.

## Technical Specification

- $1 \leq n \times m \leq 10^5$
- $1 \leq Q \leq 10^5$
- $1 \leq r_s, r_t \leq n$
- $1 \leq c_s, c_t \leq m$
- The grid consists only of `.` and `#`.
- The start and target cells are ice cells.

## Sample Input 1

```
3 3
...
.#.
...
1
1 1 3 3
```

## Sample Output 1

```
5
```

## Sample Input 2

```
3 5
....#
#....
.....
2
2 2 1 1
1 1 3 5
```

## Sample Output 2

```
-1
13
```

# Problem J
# Jigsaw of Perfect Squares

Time limit: 2 seconds
Memory limit: 1024 megabytes

## Problem Description

In the ancient archives of the New York City University, there is a curious puzzle known as the *Jigsaw of Perfect Squares*. It is said that many centuries ago, a group of mathematicians created this puzzle as both a test of intelligence and a riddle for future generations. Legends describe how the puzzle was used to decide who would be granted access to a hidden library, where countless manuscripts of mathematical secrets were stored. To gain entry, one had to solve this riddle of numbers and squares.

The puzzle works as follows. You are given an array of $n$ integers:

$$A = [a_1, a_2, \ldots, a_n].$$

You must determine whether it is possible to rearrange these integers into a new sequence

$$(b_1, b_2, \ldots, b_n)$$

such that for every position $i \in [1, n]$, the sum $b_i + i$ forms a *perfect square*. A perfect square is an integer of the form $k^2$ for some integer $k$.

Imagine each element of the array as a *puzzle piece*. The position $i$ is like a slot in the puzzle board, and the condition $b_i + i = k^2$ is the rule that ensures the piece fits perfectly into the board. If every piece can be arranged so that the whole board is filled with perfect-square sums, then the ancient puzzle is solved, and the gates to the hidden library open.

On the other hand, if no such arrangement exists, then the puzzle remains unsolved, and the legend says the mathematicians would laugh at the challenger's futile effort.

Your task is to determine whether such a permutation of $A$ exists.

## Input Format

The first line contains a single integer $n$, the length of the array. The second line contains $n$ integers $a_1, a_2, \ldots, a_n$.

## Output Format

Print `YES` if such a permutation exists. Otherwise, print `NO`.

## Technical Specification

- $1 \leq n \leq 1000$
- $1 \leq a_i \leq 10^9$

## Sample Input 1

```
3
2 3 6
```

## Sample Output 1

```
YES
```

## Sample Input 2

```
6
4 4 4 4 4 4
```

## Sample Output 2

```
NO
```

# Problem K
# Karl's Dormitory Allocation

Time limit: 2 seconds
Memory limit: 1024 megabytes

## Problem Description

At New York Cheesecake University, the number of dormitory rooms is limited and cannot accommodate all students. Traditionally, a *lottery system* has been used: each student registers, and only those who win the lottery are granted the right to live in the dorms. While this system appears fair since everyone gets one chance, in reality, not all students receive equal treatment — some win a dormitory place while others do not.

Over time, many students who do not need a dormitory place have *resold* their rights to others, creating a black market. Despite decades of school policies aimed at banning such trades, enforcement has proven costly and ineffective.

One student, **Karl**, after winning a dormitory right, criticized this system. He compared it to awarding a prize many times more valuable than a scholarship through mere luck, while also fueling an exploitative black market. Karl proposed a new allocation mechanism:

- Suppose the school has $m$ dormitory beds and $n$ students needing accommodation.
- Each student $i$ declares the value $v_i$ (in dollars) they assign to a dormitory right.
- Sort all declared values in descending order. Let $v_m^*$ be the $m$-th highest value, and $v_{m+1}^*$ be the $(m+1)$-th highest.
- The dormitory right is priced at
$$v = \frac{v_m^* + v_{m+1}^*}{2}.$$
- The total value of the $m$ dormitory rights is $m \cdot v$. This total is distributed evenly among all $n$ students, so each student is entitled to receive
$$\frac{m \cdot v}{n}.$$

Thus:

- Each of the top $m$ students (the winners) pays
$$v - \frac{m \cdot v}{n}.$$
- Each of the other $n - m$ students receives
$$\frac{m \cdot v}{n}.$$

This ensures fairness: every student receives at least $\frac{m}{n}$ of their perceived value of a dormitory right.

However, the monetary system only supports transactions in **whole dollars**. To avoid accusations of

profiteering, the school adjusts as follows:

- Each winner pays
$$P = \left\lfloor v - \frac{m \cdot v}{n} \right\rfloor$$
- Each non-winner receives
$$Q = \left\lceil \frac{m \cdot v}{n} \right\rceil$$
- The school compensates for the mismatch by paying or absorbing the difference:
$$R = |\, m \cdot P - (n - m) \cdot Q \,|$$

## Input Format

The first line contains two integers $n$ and $m$: the number of students and the number of dormitory beds.

The second line contains $n$ integers $v_1, v_2, \ldots, v_n$: the declared values of each student.

## Output Format

Output three integers $P, Q, R$:

- $P$: the amount each winning student pays,
- $Q$: the amount each non-winning student receives,
- $R$: the amount the school pays to balance the system.

## Technical Specification

- $2 \leq n \leq 2 \times 10^5$
- $1 \leq m < n$
- $1 \leq v_i \leq 10^9$ for $i \in [1, n]$

## Sample Input 1

```
5 2
10 3 7 6 2
```

## Sample Output 1

```
3 3 3
```

## Sample Input 2

```
5 4
100 20 60 40 50
```

## Sample Output 2

```
6 24 0
```

## Hint

The following explains the first sample test case.

There are $n = 5$ students and $m = 2$ dormitory beds. The declared values are: $10, 3, 7, 6, 2$. Sorted in descending order: $10, 7, 6, 3, 2$.

- The $m$-th highest value is $v_3 = v_2^* = 7$, and the $(m+1)$-th highest is $v_4 = v_3^* = 6$.

- The price is
$$v = \frac{7+6}{2} = 6.5.$$
- The total value is $m \cdot v = 2 \times 6.5 = 13$.
- Each student's fair share is $\frac{13}{5} = 2.6$.
- Each winner should pay $v - \frac{13}{5} = 6.5 - 2.6 = 3.9$.
- Each non-winner should receive $2.6$.

After rounding:

- Each winner pays $P = \lfloor 3.9 \rfloor = 3$.
- Each non-winner receives $Q = \lceil 2.6 \rceil = 3$.

Now, $m \cdot P = 2 \times 3 = 6$ is collected from the winners, while $(n-m) \cdot Q = 3 \times 3 = 9$ is distributed to the non-winners.

The school must therefore pay the difference:

$$R = |6 - 9| = 3.$$

Almost blank page

# Problem L
# Lantern Festival

Time limit: 2 seconds
Memory limit: 1024 megabytes

## Problem Description

Every year, the village of Lumina celebrates the Lantern Festival. On this night, the villagers light up the sky with glowing lanterns arranged in a straight line along the riverbank.

There are $n$ lanterns in total. Each lantern can either be **lit (1)** or **unlit (0)**.

Lina, a little girl in the village, wonders how many lanterns are glowing. She asks for your help to count the total number of lit lanterns.

## Input Format

The first line contains an integer $n$ — the number of lanterns. The second line contains $n$ integers, each being either $0$ (unlit) or $1$ (lit).

## Output Format

Print a single integer — the number of lanterns that are lit.

## Technical Specification

$1 \leq n \leq 2 \times 10^5$

## Sample Input 1

```
7
1 0 1 1 0 0 1
```

## Sample Output 1

```
4
```

## Sample Input 2

```
5
0 1 0 0 1
```

## Sample Output 2

```
2
```

Almost blank page

# Problem M
# Median Replacement

Time limit: 2 seconds
Memory limit: 1024 megabytes

## Problem Description

For a given array of $m$ integers $b_1, b_2, \ldots, b_m$, define the following operation:

- Select an index $i$ uniformly at random from 1 to $m$ (inclusive). Then, replace $b_i$ with the median of the remaining $m - 1$ elements (i.e., all elements of $b$ except $b_i$).

Here, the median of an array of length $k$ is defined as the $\lceil \frac{k}{2} \rceil$-th smallest element in the sorted array. For example, the median of the array $[2, 1, 3, 4]$ is 2, and the median of the array $[4, 7, 9]$ is 7.

Let $f(b)$ denote the expected number of operations needed to make all elements of the array $b$ equal by repeatedly applying the operation described above. It can be proved that this expected value is finite for any initial array $b$.

You are given an array $a$ of $n$ positive integers $a_1, a_2, \ldots, a_n$. Your task is to find the value of $f(p)$ for every possible prefix $p$ of the array $a$. In other words, for every integer $i$ from 1 to $n$ (inclusive), find the value of $f([a_1, a_2, \ldots, a_i])$.

## Input Format

Each test contains multiple test cases. The first line contains the number of test cases $t$. The description of the test cases follows.

The first line of each test case contains a single integer $n$ — the length of the array $a$.

The second line of each test case contains $n$ non-negative integers $a_1, a_2, \ldots, a_n$ — the elements of the array $a$.

## Output Format

For each test case, output $n$ numbers in a new line — the value of $f(p)$ for prefix $p$ of length $1, 2, \ldots, n$ respectively.

It can be proved that the answers can be represented by a rational number $\frac{p}{q}$ where $q$ is not a multiple of $998244353$. Then you need to output $p \times q^{-1}$ modulo $998244353$, where $q^{-1}$ means the multiplicative inverse of $q$ modulo $998244353$.

## Technical Specification

- $1 \leq t \leq 2 \cdot 10^5$
- $1 \leq n \leq 2 \cdot 10^5$
- $1 \leq a_i \leq 10^9$
- It is guaranteed that the sum of $n$ over all test cases does not exceed $2 \cdot 10^5$.

## Sample Input 1

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

## Sample Output 1

```
0 1 4
0 1 4 6 166374068
0 0 0 0 0 0
```