Bangladesh Artificial Intelligence Olympiad (BdAIO) 2025 - Preliminary Round

https://toph.co/c/bdaio-2025-preliminary

Toph

Schedule

The contest will run for **5h0m0s**.

Rules

This contest is formatted as per the official rules of ICPC Regional Programming Contests.

You can use PyPy 7.3 (3.10), and Python 3.12 in this contest.

Be fair, be honest. Plagiarism will result in disqualification. Judges' decisions will be final.

Notes

There are 11 challenges in this contest.

Please make sure this booklet contains all of the pages.

If you find any discrepencies between the printed copy and the problem statements in Toph Arena, please rely on the later.

A. Dora the Explorer

Dora and her friend Blue Monkey love to explore. Now that they are on their summer vacation, they have decided to go for exploration. Dora wants to go south, and Blue Monkey wants to go west. But Dora's parent forbade them from being more than d kilometers away. So they decided to move according to their preferred directions and agreed to stop moving when they are d kilometers away from each other.

The Blue Monkey and Dora move at speeds of m and n kilometers per hour, respectively. Now, before starting, they want your help to know that after how long they should stop moving.

Input

The first line of the input contains the number of test t $(1 \leq t \leq 10^5)$. The next $t$lines contain three integers m, n and d $(1 \leq m, n, d \leq 10^9)$.

Output

For each test case, print the number of minutes Dora and Blue Monkey continue their exploration. Round up the answer to two decimal points.

Samples

Input	Output
2 3 4 5 8 9 10	60.0 49.827288

Your answer will be considered correct if its absolute or relative error doesn't exceed 10^{-6}.

Tip: You can use the following code syntax to take input of m, n and d in one line.

```
m, n, d = map(int, input().split())
```

B. Kaif the Conqueror

Kaif the Conqueror has decided to conquer all states of the USN (United States of Nokali). To do so, he has launched an operation named KCN (Kaif Conquers Nokali). The USN can be represented as a grid of $m * n$ size. Each cell in the grid is either a state or a wetland. Besides, each cell can have one of the following values:

- 0 represents a wetland that cannot be conquered.
- 1 represents that the state is not under the rule of Kaif the Conqueror and can be conquered.
- 2 represents that the state is under the rule of Kaif the Conqueror.

Every day, Kaif the Conqueror takes control of any new state that is 4-directionally adjacent to a land that is under his rule.

Since Kaif the Conqueror is super busy ruling and setting up strategies for taking up new states, he wants your help to find out:

1. If he can take control of all the available states.
2. If he can take control of all the available states, then on the kth day of the operation KCN, how many new states will be added under his rulings?

Input

The first line of the test case contains three integers m, n and $k (1 \leq m, n, k \leq 1000)$. Next each m line contains n integers ranging $(0$ to $2)$.

NB: There will be at least one grid cell that has 2 in it.

Output

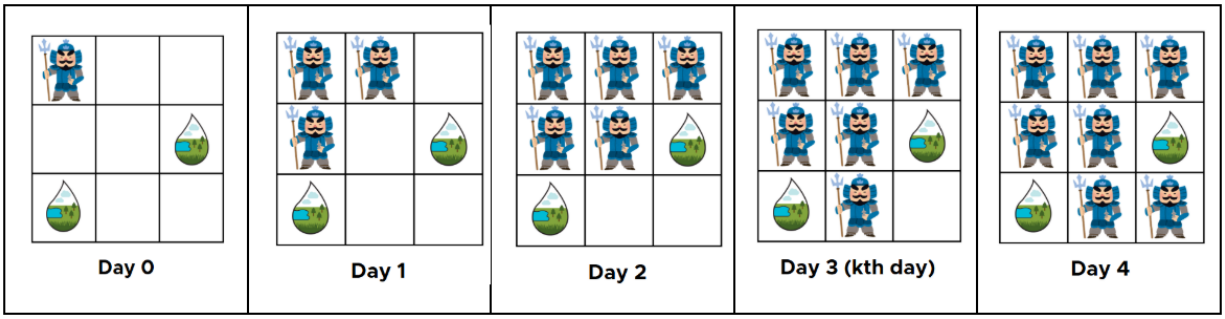
Print 0 if Kaif the Conqueror fails to conquer all the lands except wetlands, else print 1 followed by the number of states that are added under Kaif the Conqueror's rule on k th day.

NB: If all the available states are conquered before the kth day, print 0.

Samples

Input	Output
3 3 3 2 1 1 1 1 0 0 1 1	1 1

Visualization:



Input	Output
2 3 1 0 2 1 1 0 1	0

Since the state on the lowest left corner can never be conquered, the answer is 0.

Input	Output
3 3 7 2 1 1 1 1 0 0 1 1	1 0

All the states are conquered by day 4. So for 7th day the answer is 0.

C. Parrot AI

You have been hired by a futuristic pet robotics company to build a prototype of their newest product — the **Parrot AI**. This isn't just any parrot; it's a highly intelligent AI trained to **repeat whatever it hears**!

As the first step, your task is to create a basic version of this AI parrot. It takes a single input — a line of text spoken by the user — and simply repeats it back exactly as heard.

Can you build this prototype?

Input

A single line of text S ($1 \le |S| \le 1000$), consisting of printable ASCII characters, including spaces, digits, and punctuation.

Output

Output the exact same line of text.

Samples

Input	Output
`Polly wants a cracker!`	`Polly wants a cracker!`

E. Nandan's Problem

Nandan is planning to buy a new painting for his living room. He has a space on the wall where he wants to hang the painting, and the space has enough free area so that the width of the painting should not exceed **a** and the height should not exceed **b**. Nandan has a certain aspect ratio for the painting: if the width of the painting is **w** and the height is **h**, the following condition should hold: **w/h = x/y**, where **x** and **y** are fixed integers representing the desired aspect ratio.

There are many different paintings available for sale, and Nandan is sure that for any pair of positive integers **w** and **h**, there is a painting with width **w** and height **h** in the shop.

Nandan isn't sure which painting to pick yet. He wants to calculate the number of possible combinations of the painting's width and height that fit on the wall, where the width does not exceed **a** and the height does not exceed **b**, and the aspect ratio condition is satisfied.

In other words, Nandan wants to determine the number of different width and height combinations for the painting that satisfy:

- $w \le a$
- $h \le b$
- $w/h = x/y$

Input

The first line contains four integers: **a**, **b**, **x**, and **y** ($1 \le a, b, x, y \le 10^{18}$) — the constraints on the painting's width and height, and on the aspect ratio.

Output

Print one integer - the number of different width and height combinations for the painting that satisfy the aforementioned conditions.

Samples

Input	Output
17 15 5 3	3

Input	Output
14 16 7 22	0

Input	Output
4 2 6 4	1

Input	Output
1000000000000000000 1000000000000000000 999999866000004473 999999822000007597	1000000063

F. Electric Odyssey

Story

After years of saving up, Arin finally bought his dream electric car — the VoltRunner X. To celebrate, he decided to take a scenic trip through the winding hills of Zylon Ridge, a road famous for its constantly changing elevation.

But the VoltRunner has one quirk: it doesn't have regenerative braking! That means while climbing up, it uses battery power, but while descending, it recharges an equivalent amount of energy. To ensure he doesn't get stranded mid-journey, Arin wants to calculate the minimum amount of battery charge he must have in the car at the start of the trip to complete it without ever running out of power.

Can you help him?

Problem Statement

You are given an array of integers $h[1...n]$ representing the elevations (in meters) of n consecutive points on the road. Arin starts at the first point with **0 battery charge**.

He drives from the first point to the second, then to the third, and so on, until the last one.

- If the next point is **higher**, the car consumes energy equal to the height difference.
- If the next point is **lower or equal**, the car gains energy equal to the height difference (or zero if flat).

At any point, if the energy falls below zero, the car breaks down. So Arin wants to know the **minimum amount of battery charge he should have initially** so that the energy level **never goes below zero** during the entire journey.

Input

- The first line contains an integer n — the number of points on the road, where ($1 \le n \le 10^5$)
- The second line contains n space-separated integers h[1], h[2], ..., h[n] — the heights of the road points where ($0 \le h[i] \le 10^9$)

Output

Output a single integer — the minimum initial battery charge required so that the car can complete the journey safely.

Samples

Input	Output
5 10 20 15 25 10	15

G. SmartSum AI

In a highly advanced, AI-driven, futuristic economy, resources are allocated using digital coins of varying values. Each AI agent, named as SmartSum AI, is responsible for managing a set of n digital coins, each with a distinct value. The AI's primary task is to calculate all possible sums of these coins at most once to ensure optimal resource distribution across various sectors such as energy, healthcare, and technology.

The challenge lies in determining every unique monetary sum that can be formed by combining any number of coins, from none to all. These sums represent potential budgets for different initiatives, enabling precise allocation without wastage. The AI must account for the fact that coins can only be used once in each combination.

Given the coin values as input, your goal is to develop an algorithm that efficiently computes all possible distinct sums and presents them in ascending order. The solution should work for large datasets, as the futuristic economy demands rapid and accurate decision-making.

This problem is not only a test of computational ability but also a step toward designing efficient systems that mimic real-world scenarios of dynamic resource allocation. How can you assist the AI in mastering this challenge to unlock the economy's full potential?

Input

The first input line has an integer **n**: the number of coins.
The next line has n integers **x_1**, **x_2,** …, **x_n**: the values of the coins.

Output

First, print an integer **k**: the number of distinct money sums. After this, print all possible sums in increasing order.

Samples

Input	Output
3 2 4 3	7 2 3 4 5 6 7 9

Constraints

- 1 <= n <= 100
- 1 <= xi <= 1000

H. The Lost Explorer's Riddle

An ancient kingdom has unearthed a mysterious map containing **N** hidden chambers, numbered from 1 to **N**, connected by **M** secret tunnels. Each tunnel connects two chambers bidirectionally.

A brave explorer intends to navigate this labyrinth, but she fears becoming trapped in an endless loop—traveling through tunnels only to return to the starting chamber without ever reaching an exit.

Your task is to determine whether the labyrinth contains a **cycle**—a path that starts and ends at the same chamber, traversing distinct tunnels, and never revisiting any chamber in between (except for the start/end).

If a cycle exists, the explorer must prepare to block tunnels to avoid getting lost. Help her by analyzing each map and reporting whether a cycle exists.

Input

- The first line contains a single integer **T** (1 ≤ T ≤ 1000), the number of maps to analyze.
- The description of each map follows:
  - The first line of each map contains two space-separated integers N and M:
    - 1 ≤ **N** ≤ 100000 — number of chambers
    - 0 ≤ **M** ≤ min(100000, N × (N – 1) / 2) — number of tunnels
  - The next **M** lines each contain two space-separated integers **u** and **v** (1 ≤ u, v ≤ N, u ≠ v), indicating a bidirectional tunnel between chambers **u** and **v**.

Output

For each map, print a single line:

- `YES` If the map contains at least one cycle.
- `NO` If the map is acyclic.

Samples

Input	Output
3 4 4 1 2 2 3 3 4 4 1 4 3 1 2 1 3 1 4 5 4 1 2 2 3 3 4 4 5	YES NO NO

Explanation:

- **Map 1**: A closed loop (1–2-2–3-4-4-1). Has cycle.
- **Map 2:** Forms a tree structure with no loops. No cycles.

Additional Constraints:

- There are no self-loops (u ≠ v).
- No more than one tunnel exists between any pair of chambers.
- The total sum of **N** across all maps does not exceed **1,000,000.**
- The total sum of **M** across all maps does not exceed **1,000,000.**

I. Manage GPUs

At a leading artificial intelligence research company, you are responsible for managing the training of a state-of-the-art deep learning model. This model will power next-generation autonomous systems, but to be production-ready, it must complete **exactly *t* training iterations**.

Your company uses ***n*** high-performance GPU servers, each with different hardware capabilities. The **ith** server takes k_i seconds to complete a single training iteration. Since this is a distributed training setup, all servers can run in **parallel**, allowing multiple iterations to be processed at the same time.

The project deadline is tight, and every second of extra training time translates to thousands of dollars in cloud computing costs. Missing the deadline could result in investor losses and delayed deployment, affecting the entire company's roadmap.

As the **Lead AI Engineer**, you must determine the **minimum time** required to complete **exactly *t* iterations** using the available servers.

Input

1. The first line contains two integers ***n*** and ***t*** — the number of GPU servers and the number of training iterations required.
2. The second line contains *n* space-separated integers $k_1,k_2,k_3,\ldots,k_n$, where k_i represents the number of seconds required for the *ith* server to process a single iteration.

Output

- Print one integer — the **minimum time** required to complete exactly *t* training iterations.

Samples

Input	Output
3 7 3 2 5	8

Input	Output
Explanation: Server 1 process two iteration, Server 2 process four iterations and Server 3 process one iteration.	

Constraints

- $1 \le n \le 200{,}000$
- $1 \le t \le 10^9$
- $1 \le k_i \le 10^9$

J. intahAI

We are developing a powerful AI bot named intahAI, and to support it, we have two competing AI models:

1. m1, developed by **Alice**
2. m2, developed by **Bob**

Each model has a type - either **even** or **odd** - which defines how it processes data.

You are given a list of context strings to train models. You have to conduct some operations. We have two types of operations - either **modification** or **test**. Each modification replaces the context string list at any position. Each test will evaluate both models on a specific range of context strings. Models compute a **score** based on a special **"AI distance"** metric, and your job is to determine which model performs better on each test. Operation will be executed in any order.

> **AI distance:** The AI distance between two strings A and B is the **minimum number of steps** required to make them the same using the following operations:
>
> 1. Append a character at the end of the string.
> 2. Replace any single character in the string with another character.
>
> **Note:** Deletion is not allowed.

Score Calculation:

Each model maintains its own score, initialized to 0. For each test, input - `TEST l r`:

1. For each context string from index l to r (inclusive), calculate the AI distance to the model's name.
2. Filter only those distances that match the model's parity (even/odd).
3. If at least one valid distance exists, take the maximum of them. Otherwise, use 0 as the max distance.
4. New score = `previous_score ⊕ max_valid_distance`. (Update the model's `previous_score`.)

Operations

You will receive a series of interactive commands:

- `MODIFY i <new_string>` : Update the `i`-th context string to `new_string` (1-based index).
- `TEST l r` : Evaluate both models on the subarray `[l..r]`.

After each **TEST**, print the result:

- `Alice <score_diff>` if Alice has the higher score
- `Bob <score_diff>` if Bob has the higher score
- `Draw` if scores are equal

Input Format

```
<model_name1> <model_name2>
<model_parity1> <model_parity2>
n
<context_1>
<context_2>
...
<context_n>
q
<operations_1>
<operations_2>
...
<operations_q>
```

Output Format

For each TEST, output:

- `Alice <score_diff>` or `Bob <score_diff>` or `Draw`

Constraints

$1 \le n \le 100000$

$1 \le |context_i| \le 100000$

$1 \le |model\_name| \le 100000$

$1 \le q \le 100000$

All strings consist of lowercase English letters

Total length of all the initial context string ≤ 1000000

Total length of all the MODIFY operations string ≤ 1000000

Samples

Input	Output
gpt sonnet odd even 3 abc xyz gcd 4 TEST 1 2 TEST 2 3 MODIFY 2 sonnetgpt TEST 1 3	BOB 3 Draw ALICE 3

Input	Output
chatgpt gemini odd odd 10 zebwrncblo hlwjxvdkkq aocuauobnv hzupqnafcy kvlserfwug jbzmbdlmmh faufxjwbgc luqsiazawt xhufwnamud pzdosycsdf 5 TEST 1 10 MODIFY 2 txhrnmwzvt MODIFY 8 hfiivhyrms TEST 3 10 TEST 2 8	Draw Draw BOB 9

Explanation (Sample I):

The Initial score of gpt and sonnet is 0.

Operation 1: `TEST 1 2`
Context range: `["abc", "xyz"]`
For **`gpt (odd)`**:

- `ai_distance("gpt", "abc")` = 3 (odd) # Replace all 3 characters to context string
- `ai_distance("gpt", "xyz")` = 3 (odd) # Replace all 3 characters to context string
- Valid distances: [3, 3] → max = 3
- Score update: `0 ⊕ 3` = 3

For **`sonnet (even)`**:

- `ai_distance("sonnet", "abc")` = 6 (even) # Need to replace 3 and append 3 to context string
- `ai_distance("sonnet", "xyz")` = 6 (even) # Need to replace 3 and append 3 to context string
- Valid distances: [6, 6] → max = 6
- Score update: `0 ⊕ 6` = 6

Output: `Bob 3`

Operation 2: `TEST 2 3`
Context: `["xyz", "gcd"]`
For **gpt**:

- xyz → distance = 3 (odd)
- gcd → distance = 2 (even)
- Valid: [3, 0] → Max = 3
- New score: 3 ⊕ 3 = 0

For **sonnet**:

- xyz → distance = 6 (even)

- gcd → distance = 6 (even)
- Valid: [6, 6] → Max = 6
- New score: 6 ⊕ 6 = 0

Result: **Draw**

Operation 3: `MODIFY 2 sob`
Update index 2 → context becomes: `["abc", "sonnetgpt", "acd"]`

Operation 4: `TEST 1 3`
Context: `["abc", "sonnetgpt", "acd"]`

For **gpt**:

- abc → distance = 3 (odd)
- sonnetgpt → distance = 9 (odd)
- gcd → distance = 2 (even)
- Valid: [9, 3, 0] → max = 9
- Score: 0 ⊕ 9 = 9

For **sonnet**:

- abc → distance = 6 (even)
- sonnetgpt → distance = 3 (odd)
- gcd → distance = 6 (even)
- Valid: [6, 0, 6] → max = 6
- Score: 0 ⊕ 6 = 6

Result: **Alice 3**

K. Build The Stairs

You're tasked with building a **perfect staircase ramp** using `n` steps. Each step already has some height (in stair pieces), but the staircase isn't quite right.

The requirement for this project is strict:

- **Each step must be exactly 1 higher** than the step before it.

You **can only add stair pieces** to steps (you cannot remove any). Each stair piece costs **1 coin**, and you can add as many as needed.

Your job is to calculate the **minimum number of stair pieces** you'll need to add so that the staircase follows the rule:

- If the first step is height `h`, then the steps must be `h`, `h+1`, `h+2`,..., `h+n-1`.

You're free to choose any starting height `h` as long as you reach the goal using the fewest extra stair pieces.

Input

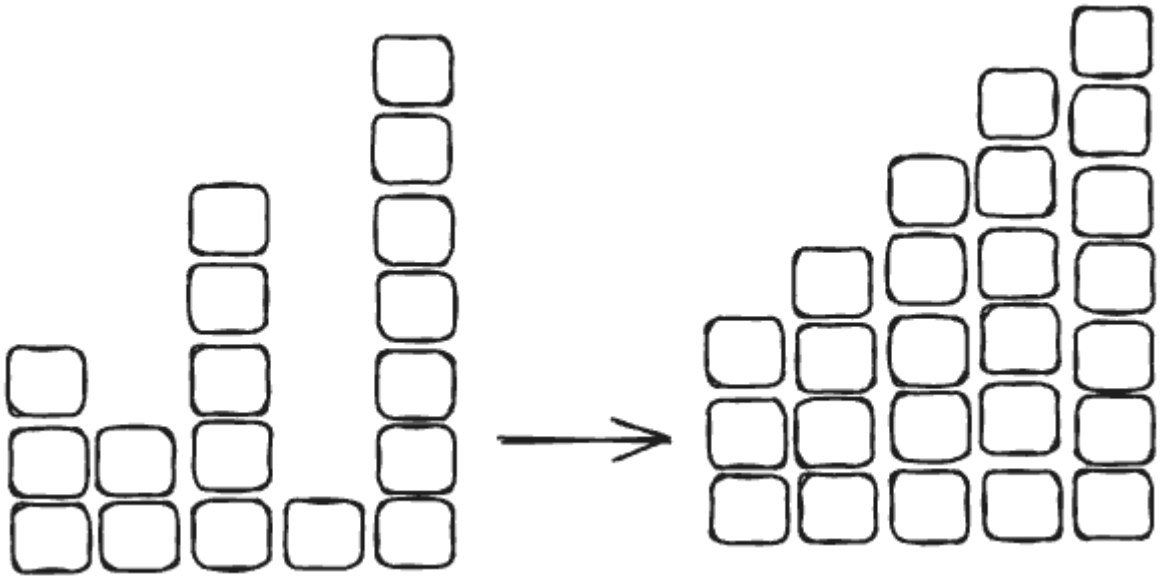
1. First line: an integer **n** — number of steps where $1 <= n <= 10^5$
2. Second line: **n** integers $a_1, a_2, a_3, \ldots a_n$ integers — the current heights of the steps, where $0 <= a_i <= 10^5$ for $1 <= i <= n$

Output

A single integer: the minimum number of stair pieces you need to add.

Samples

Input	Output
5 3 2 5 1 7	7



L. Unbalanced

A string **t** is called unbalanced if and only if:

- The length of **t** is at least 2, and
- More than half of the letters in **t** are the same.

You are given a string **s** consisting of lowercase letters. Determine if there exists an unbalanced contiguous substring of **s**. If yes, output any one such position.

Input

A single string s 2 ≤ |s| ≤ 10^5 where |s| is the length of the string s.

Output

If there exists no unbalanced substring, print: **-1 -1**

If there exists an unbalanced substring s[a...b] (1 ≤ a < b ≤ |s|), print: a b

Samples

Input	Output
needed	2 3

Input	Output
me	-1 -1

In the first example, the substring s[2...3] = "ee" is unbalanced.
In the second example, no unbalanced substring exists.