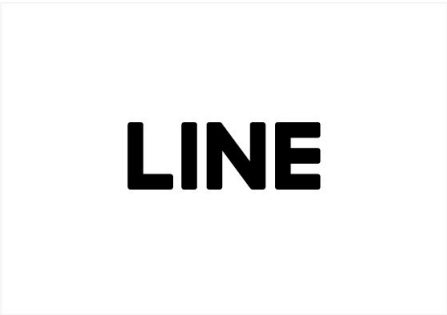
Programming Design And Optimization

Saturday, April 19, 2025

DO NOT turn to next page unless instructed to do so

CONTEST OVERVIEW

- The contest runs from 12:00 to 17:00.
- There are 14 problems in total, not ordered from easiest to hardest. However, the problems are equally weighted regardless of difficulty.
- Teams are ranked according to the most problems solved. Teams who solve the same number of problems are ranked by least total time. The total time is the sum of the time consumed for each problem solved.
- The time consumed for a solved problem is the time elapsed from the beginning of the contest to the submission of the first accepted run plus 20 penalty minutes times the number of any other submission to that problem regardless of submission time. There is no time consumed for a problem that is not solved. If two teams have the same number of problems solved and least total time, then the winner goes to the team who made their first accepted submission on their lastly solved problem earlier.
- Each team can only use a single computer with at most one mouse, one screen and one keyboard, so your team members cannot code simultaneously. Allocate your time wisely!
- **Online Judge**: You have to login to **PDOGS** with the username and password provided via email to make submissions.
- Communicating with any creature other than your teammates is strictly forbidden. Violating this rule may result in a disqualification.
- If you have questions, request a clarification via the Google form link in **PDOGS info**. The judges will clarify with either “Yes”, “No”, “No Comment” or “Pardon”.



PDAO 2025 is sponsored by

This page is intentionally left blank.

Problem A. The Mischievous Elves and the Magic Sorting Spell

Input file:	stdin
Output file:	stdout
Time limit:	1 second
Memory limit:	64 MB

Deep in an enchanted forest, a group of mischievous elves love playing a game called **Number Gathering**. During the game, they arrange a row of magic number stones in a completely random order!

The Elf King, however, has established a **sacred rule** for how these number stones should be arranged:

- At least one of the **largest** number stones must be placed at the **rightmost** position.
- At least one of the **smallest** number stones must be placed at the **leftmost** position.

The elves have a special power – the **Adjacent Swap Spell**, which allows them to swap two neighboring stones. Now, the Elf King wants to know:

What is the **minimum number of swapping** needed to rearrange the stones to satisfy his rule?

Input

Line 1 contains a single integer n, denoting the number of stones.

Line 2 contains n integers v_i, denoting the value of each stone.

- $1 \leq n \leq 100000$
- $1 \leq v_i \leq 100000$

Output

Print a single integer, the minimum number of adjacent swaps required to place:

- One of the smallest number stones at the leftmost position.
- One of the largest number stones at the rightmost position.

Example

stdin	stdout
6 3 4 1 5 5 3	3

Explanation

Given the initial arrangement: {3, 4, 1, 5, 5, 3}

1. Swap index 1 and 2: {3, 1, 4, 5, 5, 3}
2. Swap index 0 and 1: {1, 3, 4, 5, 5, 3} (smallest value is now at the leftmost position)
3. Swap index 4 and 5: {1, 3, 4, 5, 3, 5} (largest value is now at the rightmost position)

Thus, the minimum number of swaps required is **3**.

Problem B. Stock Price Prediction Adjustment

Input file:	stdin
Output file:	stdout
Time limit:	1 second
Memory limit:	64 MB

HappyCorp, a top investment firm, predicts stock prices daily. However, when compared to actual prices, the prediction error is too high! To save their reputation, they can adjust one prediction by swapping it with another from their forecast.

Analysts have access to two positive integer sequences of stock prices:

- `predicted_prices`: The predicted stock prices over a period of n days.
- `actual_prices`: The actual stock prices observed over the same period of n days.

The **total prediction error** is defined as the sum:

$$\sum_{i=0}^{n-1} |\texttt{actual\_prices}[i] - \texttt{predicted\_prices}[i]|$$

You are allowed to replace **at most one** element of `predicted_prices` with **any** other element in `predicted_prices` to **minimize** the total prediction error.

Your task is to determine the **minimum possible total prediction error** after making at most one such replacement.

Input

Line 1 contains a single integer n, denoting the number of days.

Line 2 contains n space-separated positive integers representing the **predicted** stock prices.

Line 3 contains n space-separated positive integers representing the **actual** stock prices.

Constraints:

- $1 \leq n \leq 10^5$
- $n =$ `predicted_prices.length`
- $n =$ `actual_prices.length`
- $1 \leq \texttt{predicted\_prices}[i], \texttt{actual\_prices}[i] \leq 10^5$

Output

Print a single integer, the minimum possible total prediction error after at most one replacement.

Example

stdin	stdout
3 1 7 5 2 3 5	3

Explanation

Given:

$$\texttt{predicted\_prices} = [1, 7, 5], \quad \texttt{actual\_prices} = [2, 3, 5]$$

The initial absolute sum difference is:

$$|1 - 2| + |7 - 3| + |5 - 5| = 1 + 4 + 0 = 5.$$

The optimal solution is:

- Replace the second element (7) with the first element (1), yielding:

  $$[1, \mathbf{1}, 5]$$

  The new sum difference is:

  $$|1 - 2| + |1 - 3| + |5 - 5| = 1 + 2 + 0 = 3.$$

- Alternatively, replace the second element (7) with the third element (5), yielding:

  $$[1, \mathbf{5}, 5]$$

  The new sum difference is:

  $$|1 - 2| + |5 - 3| + |5 - 5| = 1 + 2 + 0 = 3.$$

Since both cases yield a minimum absolute sum difference of 3, the output is: 3

Problem C. Xiangqi

Input file:	stdin
Output file:	stdout
Time limit:	1 second
Memory limit:	64 MB

Two Xiangqi masters Tokugawa and Gashuin said they've beaten every other master in the world. However, some said that there are some players with sharper skills. This brought them here to challenge master Junpi and Mahiro. There will be an award if they win or be a punishment. Therefore, they're careful about their play and clench every chance to win.

Now let's introduce the rules of Xiangqi:
Xiangqi is a game represents a battle between two armies with the goal of capturing the enemy's **general** piece. In this problem, you are given a situation of later stages in the game, you need to check whether you've already win after your move, i.e. you can't be **checkmated** while black has **no available moves** to prevent themself from being **checked** or **stalemate**.

Xiangqi is played on a 10×9 board and pieces are placed on the points. The upper left point is (1,1) and the lower right point is (10,9). There are two groups of pieces marked by black or red Chinese characters, belonging to the two players separately.

Each player, in turn, moves one piece from its own point to another point. No two pieces can occupy the same point at the same time. A piece can be moved to a point occupied by an enemy piece, in which case the enemy piece is **captured** and removed from the board.

When the **general** is in danger of captured by the enemy player on the enemy player's next move, the enemy player is said to have **delivered a check**. If the general's player can make no move to prevent the general's capture by next enemy move, the situation called **checkmate**.

Each piece's introduction is as follows:

General: the generals can move and capture one point either vertically or horizontally and cannot leave the **palace** unless the situation called **flying general**. **Flying general** means that one general can **fly** across the board to capture the enemy general if they stand on the same line without intervening pieces.

Advisor: the advisors can move and capture one point diagonally and may not leave the palace, which confines them to five points on the board.

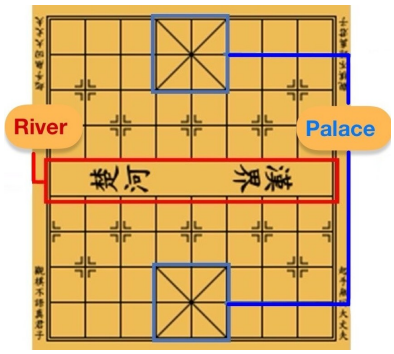
Elephant: the elephants move and capture exactly two points diagonally and may not jump over the intervening pieces; the move like the character 田, in reference to the board's squares. However, it is possible to block an elephant with a diagonally adjacent piece, known as **blocking elephant's eye** (see the figure below). When this happens, it cannot move or capture in that direction. The elephant cannot cross the river.


Chariot: the chariots can move and capture vertically and horizontally by any distance, but may not jump over the intervening pieces.


Cannon: the cannons move like the chariots, horizontally and vertically, but capture by jumping exactly one piece (whether it is friendly or enemy) over to its target.

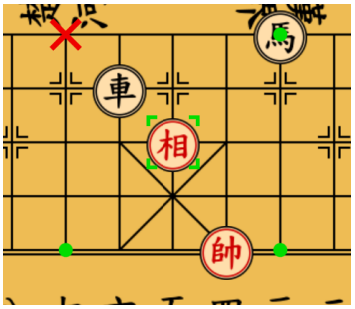

Horse: the horses have 8 kinds of jumps to move and capture shown in the left figure. However, if there is any pieces lying on a point away from the horse horizontally or vertically it cannot move or capture in that direction (see the figure below), called **hobbling horse's leg**. Now you are given a situation only containing a black general, a red general and several red chariots, cannons and horses, and the red side has delivered a check. Now it turns to black side's move. Your job is to determine that whether this situation is **checkmate**.


Soldier: the solders move and capture by advancing one point. Once they have crossed the river, they may also move and capture one point horizontally. Soldiers cannot move backward, and therefore cannot retreat; after advancing to the last rank of the board, however, a soldier may still move sideways at the enemy's edge.



Game Board


hobbling horse's leg


blocking elephant's eye

Please be noted that only the soldier, horse, canon and chariot can cross the river.
Now give you a table of later game, you need to check if you have already won the game. It's not necessary mean checkmate (Stalemate also indicates a win in Xiangqi). You always play red (i.e., the one on the bottom). You have already made your move while the other hasn't.

Input

Given a 10×9 array full of characters, where the general, advisor, elephant, chariot, horse, canon, soldier, and empty space are represented by G, S, E, R, H, C, P, and . (a single dot) respectively. Upper cases indicate Black player while lower cases indicate red player.

Output

A single line of "Yes" or "No"

Examples

stdin	stdout
...G..... ..r..... ....P. ....g....	No

stdin	stdout
...G..... ..r..... ....g....	Yes

Explanation

Notice that in Sample 1 and Sample 2, both of the black generals (G) are unable to move since all of the spaces near them are being attacked, which indicates potential stalemate. However, what makes Sample 2 a real stalemate and Sample 1 not one is that the black player has one pawn in Sample 1. Therefore, they are able to move the pawn and the game shall go on. Not like in Sample 2 where they don't have any choice other than move their general (G), which leads to their general (G) being capture by their opponent.

Notes

- The situation presented will always be achievable in normal game play, and you have already made your move while your opponent hasn't.
- Only the soldier, horse, canon and chariot can cross the river.
- You cannot be checked.

This page is intentionally left blank.

Problem D. Pirate Code: The Black Spot Game

Input file:	stdin
Output file:	stdout
Time limit:	1 second
Memory limit:	64 MB

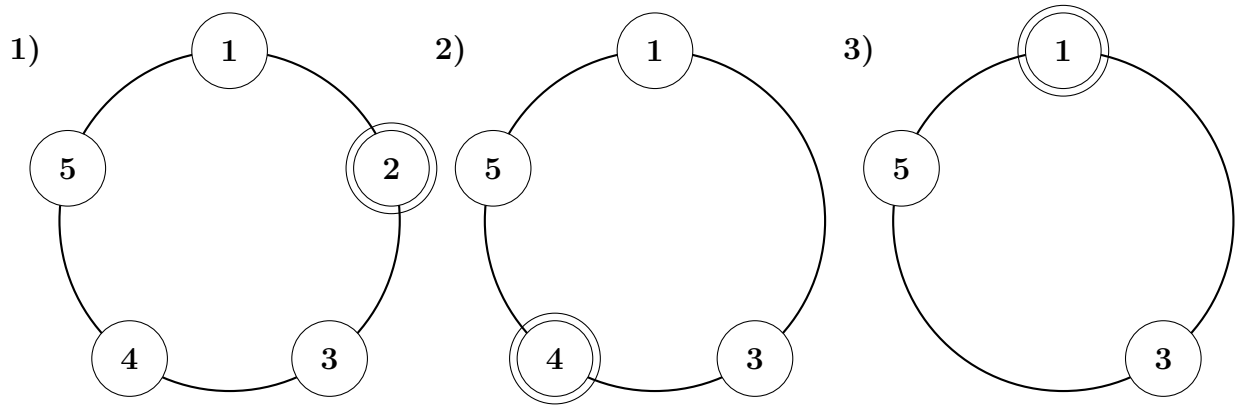
During the Golden Age of Piracy, Captain Henry Ironhook Vance and his crew of N pirates had just seized the legendary Bloodstone Compass-a mystical artifact said to guide its owner to untold riches. But greed is a pirate's curse, and the crew fell into a deadly dispute over who would claim it.

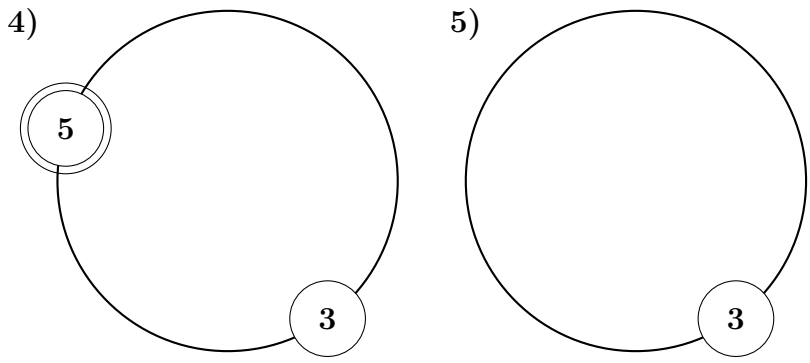
Following the ancient Pirate Code, they resorted to the dreaded Black Spot Game to decide. Here, we will first introduce the famous Black Spot Game: forming a circle, the m pirates passed around the cursed medallion. Every k^{th} pirate to receive the medallion was marked for doom and thrown overboard. The process repeated until only one pirate remained.

Here is a detailed description of the Black Spot Game: Before the game starts, there will be m pirates in total and an elimination number k will be pre-decided.

1. A total of m pirates (numbered 1 to m) stand in a circle.
2. The game starts at pirate 1.
3. Counting proceeds **clockwise**, eliminating every k^{th} pirate.
4. The pirate who is eliminated is **removed** from the circle.
5. The next round starts with the pirate immediately clockwise to the eliminated one.
6. The process continues until only one pirate remains.

For a more detailed illustration of how Black Spot Game proceeds, below is an illustration of the Black Spot Game with a total of $m = 5$ pirates and elimination number $k = 2$.





Each pirate is labeled with a number from 1 to 5 in clockwise direction. Each round, one pirate will be eliminated. The eliminated pirate is labeled in a double-circle. The winner is the last remaining pirate, which is pirate number 3.

However, after listening to the explanation of the game, the pirates thought that it was too plain and simple. To make the game more complex and sophisticated, the pirates decide to elevate the game to another level. The N pirates will be divided into n **non-identical groups**, each group $i \in \{1, 2, ..., n\}$ will consist of m_i pirates, and the group will be assigned an elimination number k_i. **Every group will play the Black Spot Game until only one pirate remain in the group.** After all groups have sort out a final survivor, the n remaining pirates (one from each group) will gather to **play one final round** of the Black Spot Game, and their numbers in the last round are their own group number.The last pirate to survive this round will be the true heir to the untold treasures.

Your task is to determine which pirate will outlast the curse and claim their fortune. You will need to determine which group the winning pirate was originally in, and the pirate's number in that original group.

Input

The first line of input consists of two integers n and k_f:

- $1 \leq n \leq 500$ Representing the number of groups to split the pirates in,
- $1 \leq k_f \leq 500$ Denoting the elimination number for the **final round**.

The following n lines (line 2 to line $n + 1$) will each represent every group of pirates, each line consist of two integers m_i and k_i, i denotes the group number.

- $1 \leq m_i \leq 500$ Denoting the number of pirates in group i,
- $1 \leq k_i \leq m_i$ Denoting the assigned elimination number for group i.

Note that we did not explicitely input the total number of pirates onboard N, since $\Sigma_{i=1}^{n} = N$ (number of pirates in each group will naturally sum up to N).

Please also note that the groups are non-identical, so the number of pirates in each group varies, the elimination number for each group also varies.

Output

You should output two integers to tell us who the winner is:

1. The group the winner was initially in.
2. The number of the winner in the group.

For example, the winner was in the 1^{st} group, and was the 3^{rd} pirate in the group, output 1 3.

Example 1

stdin	stdout
5 2 6 3 7 4 10 3 3 2 5 2	3 4

Example 2

stdin	stdout
7 3 10 2 5 3 4 3 5 2 12 4 21 10 8 3	4 3

This page is intentionally left blank.

Problem E. The Museum's Security Grid

Input file:	stdin
Output file:	stdout
Time limit:	500 milliseconds
Memory limit:	64 MB

The prestigious National Museum of Technology operates a high-tech security system based on an $m \times n$ grid of laser sensors. Each sensor has a **unique** power level used to detect intruders. However, the chief security officer has noticed that the power levels are unnecessarily high, leading to excessive energy consumption.

To optimize energy efficiency, the power levels must be reconfigured while preserving the system's effectiveness. The new configuration must satisfy the following conditions:

- If a sensor originally had a higher power level than another sensor in the **same row or column**, it must remain more powerful in the new configuration.
- The highest power level in the new configuration must be **as low as possible**.

As the museum's security consultant, your task is to determine the optimal power levels for each sensor in the grid while ensuring that the conditions above are met.

Input

Line 1 contains two integers m and n, denoting the number of rows and columns in security grid.

The next m lines each contain n space-separated integers representing the power levels of the sensors in the grid and denote the power levels of the sensors as p. Each power level in the initial configuration is unique.

- $1 \leq m, n \leq 1000$
- $1 \leq p \leq 10^5$
- p is **unique**
- $m \cdot n \leq 10^5$

Output

Output m lines, each containing n space-separated integers, representing the reconfigured security grid.

Example 1

Explanation

The original grid is:

stdin	stdout
2 2 3 1 2 5	2 1 1 2

$$\begin{bmatrix} 3 & 1 \\ 2 & 5 \end{bmatrix}$$

After reconfiguration, the new grid is:

$$\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

The maximum power level in this new grid is 5, and it can be shown that no smaller power level configuration satisfies the conditions.

Example 2

stdin	stdout
1 1 10	1

Explanation

Since there is only one sensor, it must be assigned the minimum power level of 1.

Problem F. The Ancient Formula

Input file:	stdin
Output file:	stdout
Time limit:	1 second
Memory limit:	64 MB

In an archaeological dig, Dr. Yee uncovered a collection of ancient tablets that contained strange chemical formulas written in an old script. After careful translation by his research team, the formulas were finally decoded.

As a modern chemist, Dr. Yee recognized some of the symbols as elements, but the structure seemed unusual. Was this an early attempt at chemical notation, or could it represent something valuable?

Upon further investigation, Dr. Yee noticed that some of the formulas appeared similar to those in his notes. However, the ancient formulas used parentheses and had different groupings, raising the question: Could they be describing the same compound, just with a different notation system?

Can you help Dr. Yee verify whether the ancient formula matches the modern one?

Input

Line 1 contains two integers n_1 and n_2, denoting the length of following two formulas.

Line 2 - 3 each contains a string represents ancient and modern formula respectively.

A formula follows these rules:

- An atom is denoted as a uppercase letter and may followed by a lowercase letter.
- An element can be a group of atoms, elements, or both.
- Digits (ranging from 1 to 100) following an atom or element indicate the quantity. If no digit is present, the quantity is assumed to be 1.
- Parentheses are used to group elements and can be followed by a number, indicating how many times the group should be repeated.
- Parentheses may be used to group elements, with a maximum nesting depth of 10.
- The number of every atom in each formula will not exceed 10^4.
- $1 \leq n_1, n_2 \leq 10^4$

Note: The atoms and elements in these formulas may not necessarily exist on Earth, as Dr. Yee lives in a different universe.

Output

Two formulas are considered to represent the same compound if they contain exactly the same atoms in identical quantities, regardless of how they are grouped. Therefore, the output will be either "Yes" or "No".

Example 1

stdin	stdout
10 8 (X3(OR)2)3 X8(OR2)3	No

Explanation

Expanding the formulas:

- `(X3(OR)2)3` expands as:
  - `X3` is repeated 3 times → `X9`
  - `(OR)2` expands to `O2R2`, and repeating this 3 times gives `O6R6`.

  So, the full expansion is `X9O6R6`.
- `X8(OR2)3` expands as:
  - `X8` remains as is.
  - `(OR2)3` expands to `O3R6`.

  So, the full expansion is `X8O3R6`.

Since the number of atoms does not match, the two formulas represent different compounds.

Example 2

stdin	stdout
9 15 SO2(ORe)4 (ORe)2S(ORe)2O2	Yes

Explanation

Expanding both formulas:

- `SO2(ORe)4` expands to `S1O6Re4`.
- `(ORe)2S(ORe)2O2` expands to `S1O6Re4`.

Since both formulas contain the same elements in the same quantities, they represent the same compound.

Problem G. Magic Potion Battle

Input file:	stdin
Output file:	stdout
Time limit:	500 milliseconds
Memory limit:	64 MB

In the dark ages of wizardry, the legendary sorcerers **Merlin** and **Morgana** engaged in a fierce **Magic Potion Battle**. In this battle, they took turns casting spells to destroy potion bottles infused with magical essence.

The game follows these rules:

- **Merlin** and **Morgana** take turns destroying exactly one potion bottle per turn.
- **Merlin** always plays first.
- The game ends immediately if the total amount of magical essence destroyed becomes **divisible by 3**. The player who made that move **loses**.
- If all potions are destroyed and the total essence is **not divisible by 3**, **Morgana automatically wins**.

Both players play **optimally**, meaning they make the best possible moves to maximize their chances of winning.

Your task is to determine whether **Merlin** can guarantee a win, or if **Morgana** will inevitably win. Print “Yes“ if **Merlin** wins and “No“ if **Morgana** wins.

Input

Line 1 contains a single integer n ($1 \leq n \leq 100000$), representing the number of potion bottles.

Line 2 contains n space-separated integers, where the i^{th} integer represents the magical essence in the i^{th} potion bottle. Each integer k satisfies $1 \leq k \leq 10000$.

Output

Print ”Yes” if **Merlin** can force a win; otherwise, print ”No”.

Example 1

stdin	stdout
2 2 1	Yes

Explanation

The game proceeds as follows:

- **Turn 1: Merlin** removes either potion bottle.
- **Turn 2: Morgana** removes the remaining bottle.

The sum of the destroyed potion bottles is $2 + 1 = 3$, which is divisible by 3. Since **Morgana** made the move that resulted in this, she **loses**, and **Merlin** wins.

Example 2

stdin	stdout
5 5 1 2 4 3	No

Explanation

One possible sequence of moves for **Morgana** to win:

- **Turn 1: Merlin** removes the potion with value 1. Sum = 1.
- **Turn 2: Morgana** removes the potion with value 3. Sum = $1 + 3 = 4$.
- **Turn 3: Merlin** removes the potion with value 4. Sum = $1 + 3 + 4 = 8$.
- **Turn 4: Morgana** removes the potion with value 2. Sum = $1 + 3 + 4 + 2 = 10$.
- **Turn 5: Merlin** removes the potion with value 5. Sum = $1 + 3 + 4 + 2 + 5 = 15$.

Since 15 is divisible by 3, **Merlin loses**, meaning **Morgana** wins. Thus, the output is ”No”.

Problem H. Ingenious Missions

Input file:	stdin
Output file:	stdout
Time limit:	1 second
Memory limit:	64 MB

You are the mastermind behind a covert operations unit tasked with executing high-stakes missions. Your team has a fixed number of agents and a list of m planned missions.

Your goal is to select a subset of missions to achieve **at least** a total benefit of r while using **at most** n agents in total.

Each mission has specific requirements:

- A mission requires a certain number of agents and yields a corresponding benefit.
- Each mission can be executed at most once.
- Each agent can be assigned to at most one mission.

The order in which missions are selected does not matter; only the set of missions chosen determines the outcome.

Input

Line 1 contains three integers n, r, m, denoting the **number of agents available**, the **minimum required benefit**, and the **number of missions**, respectively.

The next m lines each contain two integers g and b, denoting the **agents required to execute the mission**, the **benefit received from the mission**, respectively.

- $1 \leq n, r, m \leq 100$
- $1 \leq g, b \leq 100$

Output

Print a single integer, the total number of valid mission schemes modulo $10^9 + 7$.

Example

stdin	stdout
5 3 3 2 2 2 3 1 1	5

Explanation

There are 3 missions:

- **Mission 0**: Requires 2 agents, yields a benefit of 2.
- **Mission 1**: Requires 2 agents, yields a benefit of 3.
- **Mission 2**: Requires 1 agent, yields a benefit of 1.

Your objective is to achieve at least a total benefit of 3, using at most 5 agents.
Possible valid schemes:

1. **Select Mission 1 only**:
    - Agents used: 2, total benefit: 3.
    - **Valid**.
2. **Select Missions 0 and 1**:
    - Agents used: 4, total benefit: 5.
    - **Valid**.
3. **Select Missions 1 and 2**:
    - Agents used: 3, total benefit: 4.
    - **Valid**.
4. **Select Missions 0 and 2**:
    - Agents used: 3, total benefit: 3.
    - **Valid**.
5. **Select Missions 0, 1, and 2**:
    - Agents used: 5, total benefit: 6.
    - **Valid**.

Invalid schemes:

1. - **Select Mission 0 only**: Agents used = 2, total benefit = 2.
    - **Invalid** (benefit < 3).
2. - **Select Mission 2 only**: Agents used = 1, total benefit = 1.
    - **Invalid** (benefit < 3).

In conclusion, there are **5** valid schemes. The output is:

5

Side Note: A scheme where no missions are selected is always invalid, as it yields no benefit.

Problem I. Block Builder's Challenge: The Rainfall Simulator

Input file:	stdin
Output file:	stdout
Time limit:	1 second
Memory limit:	64 MB

In the popular game **"Block Builder's Challenge"**, players design landscapes using cubic blocks arranged in a grid. After completing a creation, the game enters the **"Rainfall Simulation"** phase, where water falls from the sky and fills the terrain's low points. The objective is to calculate how much water is trapped within the block structure.

Your task is to determine the **total volume of trapped water** after rainfall, following these rules:

- The landscape is represented as an $m \times n$ grid.
- Each grid cell contains a non-negative integer representing the height of a block.
- Water flows downward and seeks the lowest available space.
- Water can only flow horizontally or vertically (not diagonally).
- Water **can escape** if it reaches the edges of the grid.
- One unit of water fills one cubic unit of space.

Input

Line 1 contains two integers m and n, representing the grid dimensions.

The next m lines contain n space-separated integers, where each integer represents the height of a block. The constraints are:

- $1 \leq m, n \leq 200$
- $0 \leq \text{height} \leq 10^4$

Output

Print a single integer, the total volume of water (in cubic units) that is trapped after rainfall.

Example 1

stdin	stdout
3 6 1 4 3 1 3 2 3 2 1 3 2 4 2 3 3 2 3 1	4

Explanation

After the rainfall:

- Water collects in low areas but **can escape at the edges**.
- The total trapped water volume is **4** cubic units.

Example 2

stdin	stdout
5 5 3 3 3 3 3 3 2 2 2 3 3 2 1 2 3 3 2 2 2 3 3 3 3 3 3	10

Explanation

This terrain forms a **closed basin** with walls of height 3:

- Water fills up to level 3 inside the basin.
- The total volume of trapped water is **10** cubic units.

Problem J. The Interplanetary Network

Input file:	stdin
Output file:	stdout
Time limit:	1 second
Memory limit:	64 MB

The year is 2184, and humanity has successfully colonized multiple planets in the solar system. The Interplanetary Communication Network (ICN) consists of N space stations connected by M bidirectional communication channels. Each channel has a specific bandwidth capacity.

The ICN Administration seeks to establish a **minimum-cost network** that connects all space stations while ensuring its reliability. However, given the harsh space environment and potential meteor strikes, they must classify the communication channels into two types:

- **Critical Channels**: If any of these channels are damaged, the total cost of maintaining a connected network will increase.
- **Backup-Ready Channels**: These channels can be seamlessly integrated into a minimum-cost network if a critical channel fails, without increasing the total cost.

Your task is to:

- Identify all **critical channels** that, if damaged, would increase the minimum network maintenance cost.
- Identify all **backup-ready channels** that can replace critical channels without increasing the total cost.

Input

Line 1 contains two integers N and M, denoting the number of space stations and communication channels:

$$2 \leq N \leq 100, \quad 1 \leq M \leq \min(200, \frac{N(N-1)}{2}).$$

The next M lines each contain three integers u, v, w, representing a communication channel:

- u and v are space stations connected by the channel ($0 \leq u, v < N$).
- w is the bandwidth capacity of the channel ($1 \leq w \leq 1000$).
- Each channel connects **two distinct** space stations.
- There is at most **one direct channel** between any pair of stations.
- All space stations are **initially connected** through at least one path.

Output

The output consists of two lines:

- The first line contains the set of **critical channels** (space-separated indices).
- The second line contains the set of **backup-ready channels** (space-separated indices).
- If either set is empty, print `no channels` for that line.

Example 1

stdin	stdout
5 7 0 1 1 1 2 1 2 3 2 0 3 2 0 4 3 3 4 3 1 4 6	0 1 2 3 4 5

Explanation

The **minimum-cost network** has a total maintenance cost of **7 units**.

- **Channels 0 and 1** are critical because removing either would increase the minimum network cost.
- **Channels 2, 3, 4, and 5** are backup-ready since they can replace a critical channel without increasing the cost.

Example 2

stdin	stdout
4 4 0 1 1 1 2 1 2 3 1 0 3 1	no channels 0 1 2 3

Problem K. NTU Class Picking

Input file:	stdin
Output file:	stdout
Time limit:	1 second
Memory limit:	64 MB

At NTU, students must use an online system to register for their classes each semester. However, the system has a significant limitation: students cannot insert a class between existing ones directly. Instead, they must manually shift other classes to make space.

The rules for class selection are as follows:

- Classes are arranged in a sequential order, with each class occupying a unique position.
- To insert a class at position P, all existing classes at position P and beyond must be shifted down by one position.
- To delete a class at position P, the position remains empty, leaving a gap.
- Each move consists of shifting a class exactly one position up or down.

Given a sequence of insertion and deletion commands, determine the minimum number of moves required to achieve the final class arrangement.

Input

Line 1 contains an integer **N** ($1 \leq N \leq 1000$), denoting the number of operations.

Next N lines, each contains a command:

- **I P** $\rightarrow$ Insert a class at position P ($1 \leq P \leq 1000$).
- **D P** $\rightarrow$ Delete a class at position P ($1 \leq P \leq 1000$).

Output

Print a single integer, the minimum number of moves required to execute all operations.

Example 1

Explanation

1. Insert class at **position 1** $\rightarrow \{\underline{A}\}$.
2. Insert class at **position 2** $\rightarrow \{A, \underline{B}\}$.
3. Insert class at **position 3** $\rightarrow \{A, B, \underline{C}\}$.
4. Insert class at **position 1**: Shift A, B, C down by 1 $\{\underline{D}, A, B, C\}$ (**3 moves**).
5. Delete class at **position 1** $\rightarrow \{A, B, C\}$ (removal itself is not counted).

Total moves: **8**.

stdin	stdout
5 I 1 I 2 I 3 I 1 D 1	8

Example 2

stdin	stdout
5 I 1 I 2 I 3 D 2 I 1	6

Explanation

1. Insert class at **position 1** $\rightarrow \{\underline{A}\}$.
2. Insert class at **position 2** $\rightarrow \{A, \underline{B}\}$.
3. Insert class at **position 3** $\rightarrow \{A, B, \underline{C}\}$.
4. Delete class at **position 2** $\rightarrow \{A, \_, C\}$ (creates a gap).
5. Insert class at **position 1**: Shift A down by 1 $\{\underline{D}, A, C\}$ (**1 move**).

Total moves: **6**.

Notes

- Each class occupies exactly one position and cannot overlap unless deleted.
- Empty spaces may remain after deletions.
- Moves are counted only when shifting existing classes up or down.

Problem L. NTU I'm Forest Master

Input file:	stdin
Output file:	stdout
Time limit:	1.5 seconds
Memory limit:	64 MB

Your uncle, Roger, a humble lumberyard farmer, sells the best part of his forest to earn a living. However, there is one big problem: he has absolutely no idea how to handle it!

For years, he has been growing trees and selling them blindly, never knowing whether he was cutting down the most valuable part of the forest. But this year, during the Lunar New Year, something changed. As you were trying to survive the usual storm of nosy relatives asking about your studies, uncle Roger overheard that you were studying at **NTUIM**.

With his eyes full of hope (and possibly tears), he asked for your help, being the good niece and nephew(haiyaa), you agreed. Now, it's time to use your programming skills to help uncle Roger maximize his profits!

You need to help uncle Roger manage his forest by keeping track of the growth and changes in tree heights over time. The forest consists of N **trees** planted in a straight line, each with an initial height. Over time, various events such as fertilization or natural disasters will alter the trees' heights.

And sometimes, uncle Roger may wonder which part is the **most valuable segment** of the forest, of fixed length LN, to determine which part to sell. The value of a segment is determined by the following formula:

$$\text{Value} = (\max(H_L, H_{L+1}, \ldots, H_{L+LN-1}) \times P_1) + (\sum_{i=L}^{L+LN-1} H_i \times P_2)$$

Where:

- $\max(H_L, H_{L+1}, \ldots, H_{L+LN-1})$, $\sum_{i=L}^{L+LN-1} H_i$ is the **maximum and sum of tree height** in the segment, respectively.
- P_1 and P_2 are the given price coefficients.

Your task is to track the changes of the forest and help your uncle.

Input

The first line contains two integers:

$$N \quad Q$$

- N ($1 \leq N \leq 20000$) - Number of trees in the forest.
- Q ($1 \leq Q \leq 100000$) - Number of operations to be performed.

The second line contains N integers:

$$H_1 \quad H_2 \quad \dots \quad H_N$$

- H_i $(0 \leq H_i \leq 100)$ - The initial height of the i^{th} tree.

The 3 to $Q+2$ lines contain operation in one of the following formats:

1. **Fertilization/Natural Disaster**

$$1/2 \quad L \quad R \quad v$$

   - Increase/Decrease the height of all trees in range $[L, R]$ by v.
   - Constraints: $1 \leq L \leq R \leq N$, $1 \leq v \leq 10$.

2. **Query of Most Valuable Segment**

$$3 \quad LN \quad P_1 \quad P_2$$

   - Find the best value segment of length LN.
   - Constraints: $1 \leq LN \leq min(1000, N)$, $1 \leq P_1,\ P_2 \leq 1000$.
   - There will be at most 100 query operation.

Output

If there is a query operation, print the segment of length LN that has the highest value. Print the best segment in the format:

$$L\ R\ V$$

Where:

- L, R is the starting index and ending index of the best segment.
- V is the maximum value of the segment.

If multiple segments have the same maximum value, print the one with the **smallest starting index**. If there is no segment with positive value, print "**Oiiaioiiiai**".

Example 1

stdin	stdout
5 3 1 3 2 5 4 1 1 3 2 2 2 4 1 3 4 2 1	2 5 23

Explanation

After operation 1-2 , the forest heights are:

$$[3, 4, 3, 4, 4]$$

Finding the best segment of length $LN = 4$:

- Segment $[1, 4]$: $\max(3, 4, 3, 4) = 4$, $\sum = 3 + 4 + 3 + 4 = 14$

$$\text{Value} = (4 \times 2) + (14 \times 1) = 8 + 14 = 22$$

- Segment $[2, 5]$: $\max(4, 3, 4, 4) = 4$, $\sum = 4 + 3 + 4 + 4 = 15$

$$\text{Value} = (4 \times 2) + (15 \times 1) = 8 + 15 = 23$$

Thus, the best segment is:

$$2\ 5\ 23$$

Example 2

stdin	stdout
5 3 1 2 1 3 0 2 1 5 8 3 4 2 1 1 1 3 2	0 0 0

Explanation

After operation 1, the forest heights are:

$$[-7, -6, -7, -5, -8]$$

Finding the best segment of length $LN = 4$:

- Segment $[1, 4]$: $\max(-7, -6, -7, -5) = -5$, $\sum = -7 + (-6) + (-7) + (-5) = -25$

$$\text{Value} = (-5 \times 2) + (-25 \times 1) = -10 + (-25) = -35$$

- Segment $[2, 5]$: $\max(-6, -7, -5, -8) = -5$, $\sum = -6 + (-7) + (-5) + (-8) = -26$

$$\text{Value} = (-5 \times 2) + (-26 \times 1) = -10 + (-26) = -36$$

Since there is no any segment has positive value, the output is:

Oiiaioiiiai

This page is intentionally left blank.

Problem M. Cursed Envoys and Race for Awakening

Input file:	stdin
Output file:	stdout
Time limit:	1 second
Memory limit:	64 MB

In a mysterious world, multiple ancient kingdoms exist, each ruled by powerful monarchs. Every few centuries, the kingdoms send their wisest envoys to participate in the legendary Moonlight Trial–a test designed to break an ancient curse.

- Each of you has a shadow that is either circular or triangular.
- There are at least one person with triangular shadow in each kingdom.
- You can see the shadows of others, but you cannot see your own.
- Every night, during the Moonlight Ritual, you may declare the shape of your shadow. If you are correct, you will awaken; if you are wrong, you will be forever lost in darkness.
- The chosen individuals cannot communicate with each other and must deduce their shadow's shape over time.

The trial's objective is clear: The first kingdom whose envoys all successfully identify their own shadow shapes and break the curse will claim ultimate victory.

Input

Line 1 contains an integer K, denoting the number of countries participating the Trial.

Line 2 to $K + 1$ each contains two interger, N_i and M_i, denoting the number of chosen individuals and the number of people with triangular shadows in country i, respectively.

- $1 \leq K \leq 10000$
- $1 \leq M_i \leq N_i \leq 100000$

Output

Print a single integer, the number of country that will win the Moonlight Trial.

If there are more than one country that will win, print the country with the most number of people since they have more power.

But if there still are more than one country with the same number of people, print the one with the smaller number since they are lucky.

stdin	stdout
2 10 2 2 1	2

Example 1

Explanation

In country 2, there is only one person with a triangular shadow, the one with triangular shadow will find everyone is circular shadow. But there must be one person with triangular shadow, thus itself can determine its shape and awaken. And the second night everyone will find they are all circular shadow.

And in country 1, there are 2 people with triangular shadow, thus they definitely will need more days than country 2 to determine their shape.

Thus, country 2 will win the Moonlight Trial.

Example 2

stdin	stdout
2 3 1 2 1	1

Explanation

Since we know those two country will all identify their shadows in the same night, but country 1 has more people.

Thus, country 1 will win the Moonlight Trial.

Problem N. Signal Synchronization in the Sky Port

Input file:	stdin
Output file:	stdout
Time limit:	1 second
Memory limit:	64 MB

In the futuristic Sky Port, thousands of aerial drones arrive and depart every second. Each drone is equipped with a synchronization chip that ensures it remains in alignment with two different transmission frequencies used by the control tower.

To ensure flight safety, a master synchronization array c must be generated. This array serves as an encrypted signal. Each drone will decode this signal using their own decoding window.

Here is how the synchronization system operates:

- The control tower provides two sequences a and b, each of length N, representing the plaintext message that drones of type-A and type-B should be able to recover.
- Each type-A drone extracts its message by XOR-ing every contiguous subarray of k_1 elements (in circular fashion) from the array c.
- Each type-B drone extracts its message by XOR-ing every contiguous subarray of k_2 elements (also circular) from the array c.

Your job is to generate such an array c of N non-negative integers, so that:

- XOR of every k_1-length contiguous subsequence (circular) of c equals the value in a.
- XOR of every k_2-length contiguous subsequence (circular) of c equals the value in b.

It is promised that there exists at least one valid c extracted to recover both messages.

XOR Explanation

XOR, denoted $\oplus$, is a binary operation. For two bits a and b, the operation is defined as:

$$a \oplus b = \begin{cases} 1, & \text{if } a \neq b; \\ 0, & \text{if } a = b. \end{cases}$$

When doing XOR on integers, it is performed bitwise on their binary. For example, computing the XOR of two integers, 12 and 10. We first convert 12 and 10 to binary representation:

$$12_{10} = 1100_2 \quad \text{and} \quad 10_{10} = 1010_2.$$

And recall that for two bits, the XOR operation $\oplus$ in binary operation.

Bit:	1	1	0	0
$\oplus$	1	0	1	0
	0	1	1	0

Thus the result in binary and its decimal of $12 \oplus 10$ is:

$$(0\,1\,1\,0)_2 = 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 0 + 4 + 2 + 0 = (6)_{10} \quad \Rightarrow 12 \oplus 10 = 6.$$

Input

Line 1 contains three integers N, k_1, and k_2.

Line 2 contains N integers $a_1, a_2, \ldots, a_N$ the message to be extracted by type-A drones.

Line 3 contains N integers $b_1, b_2, \ldots, b_N$ the message to be extracted by type-B drones.

- $1 \leq N \leq 10^5$
- $1 \leq k_1, k_2 \leq N \wedge k_1 \neq k_2$
- $0 \leq a_i, b_i < 2^{30}$

Output

Print N non-negative integers $c_1, c_2, \ldots, c_N$ representing the encrypted signal such that the XOR constraints allow recovery of both messages.

If there are multiple valid arrays, print any one of them. any number you write to the output must be in the range of $[0, 2^{30})$, or you'll get a "Wrong Answer".

Example 1

stdin	stdout
5 2 4 2 2 1 2 3 3 0 2 0 1	2 0 2 3 1

Explanation

To recover each a_i, type-A drones perform XOR on every window of $k_1 = 2$ elements from c:

- $c_1 \oplus c_2 = 3 \oplus 1 = 2 = a_1$
- $c_2 \oplus c_3 = 1 \oplus 0 = 1 = a_2 \ldots$

Type-B drones apply XOR on $k_2 = 4$-element windows, and the values match perfectly with b.

Thus, the encrypted array is valid.

Example 2

stdin	stdout
7 2 5 2 1 5 0 1 0 7 2 5 0 3 7 2 3	3 1 0 5 5 4 4