icpc International Collegiate Programming Contest




# The 2026 ICPC North America Southeast Regional Contest

## Division 1

# Official Problem Set

Google Cloud
Google DeepMind
icpc diamond global sponsor

JETBRAINS
icpc global sponsor programming tools

Jane Street
icpc multi-regional titanium sponsor

north america platinum sponsor

north america institutional sponsor

This page is intentionally left blank.

# Problem A
## Recursive Pattern

Time Limit: 7 seconds  Memory Limit: 1GB

Solving too many competitive programming problems can trick the mind into noticing patterns where they may not even exist. Today's sanity check involves looking for "recursive" $2 \times 2$ patterns in everyday objects.

The simplest recursive pattern is any $2 \times 2$ square of tiles, where each tile is either an '`X`' or an '`o`'.

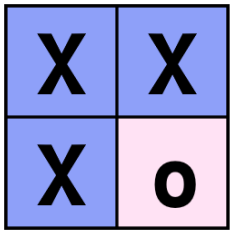



Figure A.1: One of the simplest recursive patterns.

To make your pattern more interesting, you can "apply" a second $2 \times 2$ square onto your existing pattern. This is done by replacing every '`X`' in the existing pattern with this new square and replacing every '`o`' in the existing pattern with a $2 \times 2$ square of '`o`' tiles. After this is done, the pattern is now twice as wide and twice as tall!

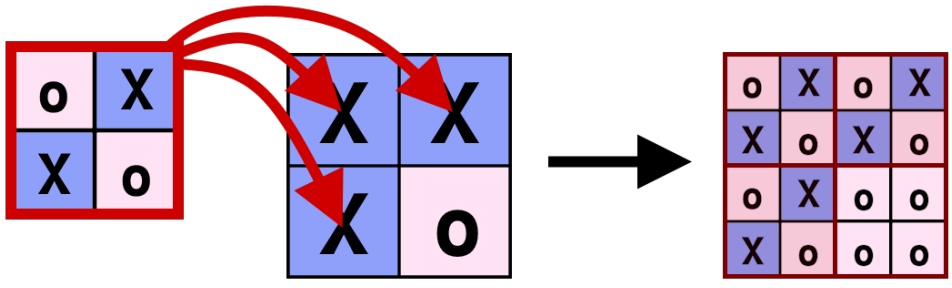



Figure A.2: "Applying" a second square onto the first pattern.

This can be done any number of times. Each time, you can apply any $2 \times 2$ square, not necessarily

the same one as before.

You found a 2D grid of 'X' and 'O' tiles in the wild that you suspect might be a subrectangle within a recursive pattern. Given this 2D grid, determine if there exists any recursive pattern that contains this grid as a subrectangle.

## Input

The first line of input contains two positive integers $r$ and $c$ ($1 \leq r \cdot c \leq 2 \cdot 10^5$), the number of rows and columns of your 2D grid.

The next $r$ lines describe the rows of the grid from top to bottom. Each of them contains $c$ characters, giving the tiles of its row from left to right. Every character is either 'X' or 'O'.

## Output

Output 1 if there is a recursive pattern containing the input grid as a subrectangle, and 0 otherwise.

| Sample Input 1 | Sample Output 1 |
|---|---|
| 2 2<br>XX<br>Xo | 1 |

| Sample Input 2 | Sample Output 2 |
|---|---|
| 2 3<br>oXo<br>Xoo | 1 |

| Sample Input 3 | Sample Output 3 |
|---|---|
| 6 45<br>ooXXXooXooXXXXoooooXXXooXXXooXoooXoXXXXoXooXoo<br>ooXooXoXoXXooooooXXoooXooXXoXoooXooXoooXooXoo<br>ooXXXooXoXooooooooXoooXoooXoXoooXooXoooXooXoo<br>ooXoXXoXoXooXXooooooXooXoooXoXoooXooXoooXXXXoo<br>ooXooXoXoXXooXooooooXXoXooXXoXXooXooXoooXooXoo<br>ooXXXooXooXXXXoooXXXoooXXXoooXXXoooXoooXooXoo | 0 |

# Problem B
## Castle-pult

Time Limit: 1 second  Memory Limit: 1GB

An alliance of three castles exists on a circle. Along the edge of this circle lie $n$ evenly spaced zones, three of which have a castle. The zones are numbered $1$ to $n$ in order around the circle.

These castles want to be able to quickly reinforce each other with troops should one be attacked. Once an attack begins, each other castle embarks troops to move to the castle under siege. Troops walk along the edge of the circle in either direction, taking one day to move to an adjacent zone.

However, because the distances between castles may be long, each castle has decided on a more *epic* way to embark its troops. Installed in each castle is a cannon that launches its troops exactly $x$ zones along the edge of the circle. A cannon launch takes no time. All castles use the same $x$.

Whenever troops set out, the cannon **always** launches them first: they choose whether it sends them $x$ zones clockwise or counter-clockwise, and then walk the shorter way to their destination. They take the fastest such route, and they cannot use another castle's cannon.

The alliance wants to choose a positive integer for $x$ such that the *maximum number of days* needed for troops to move between any pair of castles is *as small as possible*.

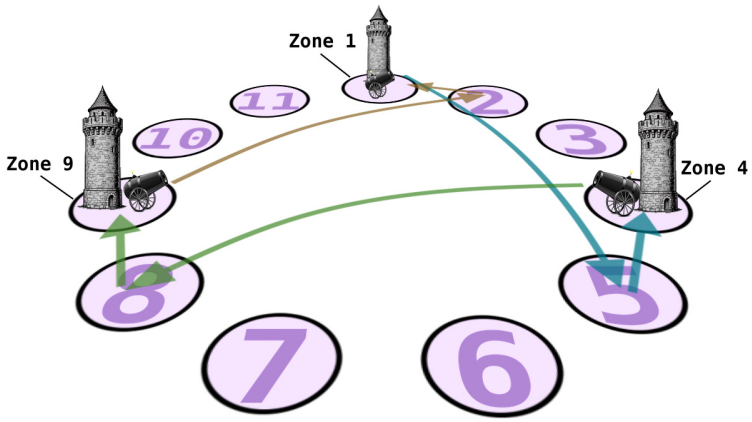



Figure B.1: Illustration of Sample 1. The alliance chooses $x = 4$. The diagram shows how troops can launch from a cannon and move at most one zone to get between any pair of castles.

## Input

The input consists of a single line containing four integers: the number of zones $n$ ($3 \leq n \leq 10^9$), followed by the positions of the three castles, each between $1$ and $n$, inclusive. The three positions

are distinct.

## Output

Output a single integer, the *smallest possible* maximum number of days needed for troops to move between any pair of castles, assuming the alliance picks $x$ optimally.

| Sample Input 1 | Sample Output 1 |
|---|---|
| 11 1 4 9 | 1 |

| Sample Input 2 | Sample Output 2 |
|---|---|
| 12 11 1 5 | 2 |

# Problem C
## Crystalline Conundrum

Time Limit: 2 seconds  Memory Limit: 1GB

You are given an $r \times c$ grid of tiles, where each tile is either a '`.`' or a '$\star$'. A '$\star$' represents a crystal and a '`.`' represents an empty space.

Your goal is to replicate this input by "growing" crystals on a different $r \times c$ grid that is initially empty. When you finish, your grid must match the input exactly, with crystals on all the '$\star$' tiles and none on the '`.`' tiles.

First, you choose $x$ tiles on this grid and place a crystal on each tile. There are no restrictions on which tiles you choose.

Then, you may do either of the following any number of times, in any order:

1. Shine a light on a column. A crystal grows directly above the highest existing crystal in the column. If no crystal exists in this column, or the highest one is at the top edge of the grid, no new crystal grows.
2. Shine a light on a row. A crystal grows directly to the left of the leftmost existing crystal in the row. If no crystal exists in this row, or the leftmost one is at the left edge of the grid, no new crystal grows.

You prefer growing crystals rather than simply placing them on the grid, so you want $x$ to be as small as possible. What is the smallest value of $x$ such that the input grid is still reconstructable using the method described above?

## Input

The first line of input contains two integers $r$ and $c$ ($1 \leq r, c \leq 800$), the number of rows and columns of the grid.

The next $r$ lines describe the rows of the grid from top to bottom. Each of them contains $c$ characters, giving the tiles of its row from left to right. Every character is either '$\star$' or '`.`'.

It is guaranteed that the grid contains at least one crystal.

## Output

Output a single integer, the smallest value of $x$, which is the number of crystals you must place at the beginning.

| Sample Input 1 | Sample Output 1 |
|---|---|
| 4 4<br>***.<br>....<br>.**.<br>..*. | 3 |

| Sample Input 2 | Sample Output 2 |
|---|---|
| 3 4<br>**..<br>*...<br>*.** | 4 |

# Problem D
## Daisy Chain

Time Limit: 6 seconds  Memory Limit: 1GB

You are hosting an event where $d$ devices need to be powered. Exactly $t$ of them have 3-prong plugs, and the rest have 2-prong plugs. The venue has $w$ wall outlets, all of them 3-prong. They may not be enough for all the devices, so you are going shopping for power strips.

The store has $n$ power strips for sale, and each of them can be purchased at most once. Each power strip has a price, a number of 2-prong outlets and a number of 3-prong outlets. Every power strip also has a 3-prong plug of its own.

Plugging works as follows:

- A 3-prong plug fits only a 3-prong outlet. A 2-prong plug fits *either* kind of outlet.
- Each outlet holds at most one plug.
- A power strip plugged into a wall outlet has power. A power strip also gets power if it is plugged into another power strip that already has power. A device has power if it is plugged into a wall outlet or into a power strip that has power.
- There is no current limit: every wall outlet and every power strip can power everything plugged into it.

Find the minimum total cost of the power strips you need to buy so that all $d$ devices have power.

## Input

The first line of input contains four integers $d$, $t$, $w$ and $n$ ($1 \le d \le 500$, $0 \le t \le d$, $1 \le w, n \le 500$): the number of devices, the number of devices with 3-prong plugs, the number of wall outlets, and the number of power strips for sale.

Each of the next $n$ lines contains three integers $p$, $a$ and $b$ ($1 \le p \le 10^6$, $0 \le a, b \le 500$, $a+b \ge 1$): the price of a power strip, its number of 2-prong outlets, and its number of 3-prong outlets.

## Output

Output a single integer, the minimum total cost of the power strips to buy so that all $d$ devices have power. If this is impossible, output $-1$ instead.

| Sample Input 1 | Sample Output 1 |
|---|---|
| 6 2 1 4<br>7 2 2<br>4 4 0<br>10 0 3<br>12 6 1 | 14 |

| Sample Input 2 | Sample Output 2 |
|---|---|
| 5 2 2 2<br>5 4 0<br>3 1 1 | -1 |

# Problem E
## Mall Escape

Time Limit: 3 seconds  Memory Limit: 1GB

It's the zombie apocalypse, and a zombie has chased you into a mall! The zombie has lost track of you for now, but you need to escape the mall without being detected and eaten.

The rooms and hallways of the mall can be represented by a connected graph. There are $n$ rooms, numbered from 1 to $n$, and some pairs of rooms are connected by a bidirectional hallway. You start at room 1 and the mall entrance is room $n$. The zombie starts at the mall entrance. A room's distance from the mall entrance is the number of hallways on the shortest route between them.

Every minute, the following happens, in order:

1. You may move from your current room to a neighboring room. You may also stay put, if you think doing so is safer.
2. Then, after your turn, the zombie shambles from its current room to a neighboring room that is *strictly* farther from the mall entrance than its current room. If there are multiple such rooms, it chooses one uniformly at random. If none of the neighboring rooms are farther from the mall entrance than its current room, the zombie stays put.

Every move, yours or the zombie's, goes through a single hallway.

If at any point in this process you are in the same room as the zombie, the zombie catches you and eats your brains. If you make it to the mall entrance (and the zombie isn't currently there!) you safely escape.

You know that the zombie starts at the mall entrance and that it moves according to the above, but you have no way to track the zombie once it starts moving. If you move according to your best possible strategy for escaping the mall, what is the probability of the zombie catching you?

## Input

The first line of input contains two integers $n$ and $e$ ($2 \leq n \leq 5 \cdot 10^5$, $n - 1 \leq e \leq 5 \cdot 10^5$), the number of rooms in the mall and the number of hallways between the rooms.

Each of the next $e$ lines contains two integers $a$ and $b$ ($1 \leq a < b \leq n$), denoting that a bidirectional hallway connects room $a$ and room $b$. There is at most one hallway between any pair of rooms. It is possible to reach the mall entrance from every room.

## Output

Output a single real number, the probability of the zombie catching you if you move optimally. Your answer will be accepted if its relative error does not exceed $10^{-6}$ or its absolute error does not exceed $10^{-200}$. Note that the answer **can be smaller than $10^{-200}$**. You may output the answer in scientific notation.

## Explanation of Sample Case 1

You can either move immediately into one of the rooms 2, 3, or 4, or wait one minute and then move into one of these rooms. Either way, the zombie has a $1/3$ chance of moving into your room (in the first case) or already being there (in the second case); if the zombie doesn't come/isn't there, you can escape the mall on your next turn. Staying put in room 1 for longer than one minute guarantees that you're caught by the zombie, so it isn't optimal.

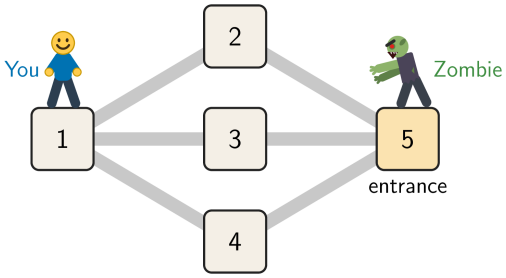



Figure E.1: Illustration of Sample 1. You start in room 1, and the zombie starts at the mall entrance, room 5.

## Explanation of Sample Case 2

An optimal strategy is to go straight to either room 4 or room 8, wait there for a minute, and then continue to the mall entrance. This strategy minimizes the probability of getting caught by the zombie ($1/4$). Note that making a beeline directly for the mall entrance, without waiting a minute in room 4 or 8, is worse as it increases your probability of getting caught to $1/2$.

| Sample Input 1 | Sample Output 1 |
|---|---|
| 5 6<br>1 2<br>1 3<br>1 4<br>2 5<br>3 5<br>4 5 | 0.3333333333333333 |

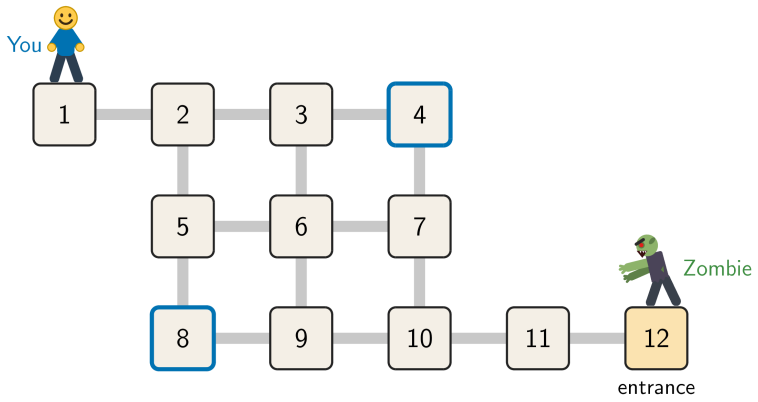



Figure E.2: Illustration of Sample 2. An optimal strategy waits in room 4 or 8, outlined in blue.

| Sample Input 2 | Sample Output 2 |
|---|---|
| 12 15<br>1 2<br>2 3<br>3 4<br>2 5<br>3 6<br>4 7<br>5 6<br>6 7<br>5 8<br>6 9<br>7 10<br>8 9<br>9 10<br>10 11<br>11 12 | 2.5e-01 |

This page is intentionally left blank.

# Problem F
## Ciphertext Count

Time Limit: 16 seconds  Memory Limit: 1GB

Agent Twilight became a spy because he wanted to make the world a better place. Besides, someone has to prove that human agents are better than AI ones. However, he didn't expect the amount of paperwork that would come with it. He has to send an encrypted report to his agency every day. To minimize the work he has to do, he wrote one report and decided to keep sending encrypted excerpts of it (each one is called a *ciphertext*). His agency will catch on if they receive the same ciphertext twice.

A *monoalphabetic substitution* assigns each of the 26 lowercase English letters a replacement letter, and no two letters get the same replacement. A letter may be its own replacement. It encrypts a string by replacing every letter in the string with that letter's replacement. For example, a substitution that replaces `p` with `t`, `a` with `i`, `e` with `l` and `r` with `e` turns `paper` into `title`.

Each day, Twilight chooses a non-empty substring of the report, that is, one or more consecutive letters, and encrypts it with a monoalphabetic substitution of his choice. He then sends the encrypted substring as the ciphertext for that day. On different days, he may reuse both substrings and substitutions, but he may not repeat a ciphertext. How many days can Twilight keep sending encrypted excerpts of his report before he has to write a second report?

## Input

The input consists of a single line containing the report, a string of lowercase English letters (`a`–`z`). The length of the string is between 1 and $3 \cdot 10^5$, inclusive.

## Output

Output a single integer, the maximum number of days for which Twilight can send a new ciphertext each day. Since the answer can be large, output it modulo $998\,244\,353$.

## Explanation of Sample Case 1

The string `aba` has five distinct non-empty substrings: `a`, `b`, `ab`, `ba` and `aba`. Encrypting `a` or `b` can produce any single letter, so these two substrings give 26 ciphertexts together. Encrypting `ab` or `ba` can produce any two different letters, so these two give $26 \cdot 25 = 650$ ciphertexts together. Encrypting `aba` can produce any three letters whose first and last are the same and differ from the middle one, which gives another 650. In total, there are $26+650+650 = 1\,326$ distinct ciphertexts.

| Sample Input 1 | Sample Output 1 |
|---|---|
| aba | 1326 |

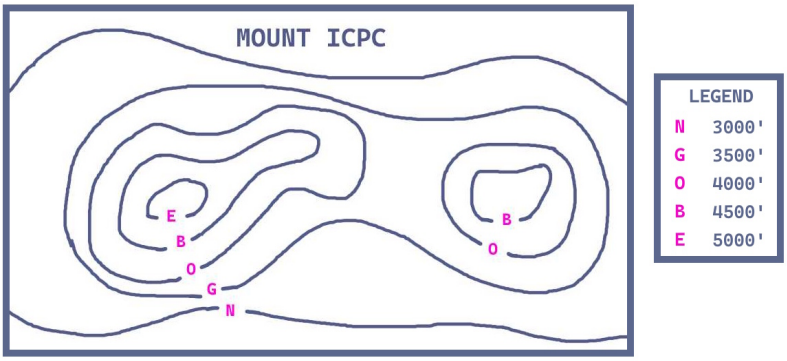
| Sample Input 2 | Sample Output 2 |
|---|---|
| decipher | 834038017 |

# Problem G
## Uncertain Topography

Time Limit: 1 second  Memory Limit: 1GB

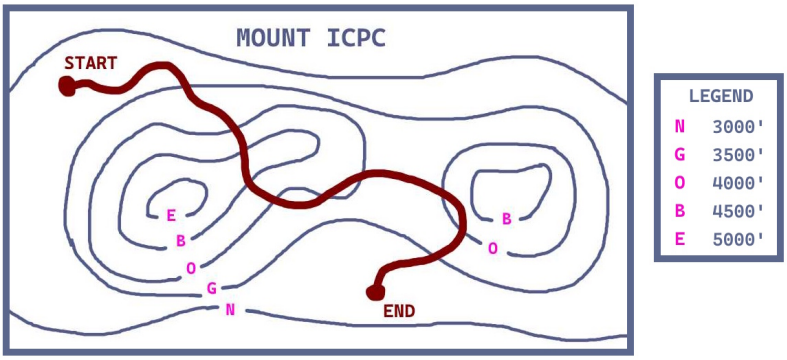
You went on an outdoor trek across a multi-peak mountain ridge. However, your topographic maps are a bit strange: they label contour lines with letters instead of elevations.

A topographic map describes a mountain range by drawing *contour lines* over it. Each contour line has a specified elevation, and every point on that line is of that elevation. An example is below:




In these maps, each letter represents a single, distinct elevation (divisible by `500'`) based on the legend attached to the map. Additionally, two adjacent contour lines with different elevations are always `500'` apart, where adjacent means you can walk from one line to the other without crossing a different contour line.

To understand your hike, you draw it on this contour map, and write down a contour line's letter every time your route reaches that line, whether it crosses the line or turns back. Your route only meets contour lines at single points and never follows one for any distance. An example is below:



For the example above, you would record `GOBBOOOOOG`.

Finally, once you have this record, you want to know which letter is the highest contour line your route reaches. Using the map's legend above, the answer is clearly `B`. Under a different valid legend, though, the answer could be different. For instance, if the legend mapped `E` to `3000'`, `B` to `3500'`, `O` to `4000'`, `G` to `4500'`, and `N` to `5000'`, the highest contour line your route reaches would be `G`.

Just after you recorded your route as described above, a strong gust of wind blew your map right out of your hands! Consequently, you could no longer use its legend to determine the highest contour line on your route. Given the record of your route, can you identify all letters that could have been the highest contour line your route reached, under some valid topographic map?

## Input

The input consists of a single line containing the record of your route: a string of uppercase English letters (`A`–`Z`), one for each time your route reaches a contour line, in order. The string has between 1 and $10^4$ letters, inclusive. It is guaranteed that some valid topographic map allows your route.

## Output

Output a single string with some number of distinct uppercase English letters sorted in *lexicographical* order. These letters are the ones that could be the highest contour line your route reaches, under some valid topographic map.

| Sample Input 1 | Sample Output 1 |
|---|---|
| GOBBOOOOOG | BG |

| Sample Input 2 | Sample Output 2 |
|---|---|
| AAAAAAA | A |

| Sample Input 3 | Sample Output 3 |
|---|---|
| ABBC | AC |

# Problem H
## Office Soundproofing

Time Limit: 1 second  Memory Limit: 1GB



Illustration of the panel installation in Sample 1's output. Each panel is drawn in white.

The Incessantly Chatty Programmers Company (ICPC) recently moved into a new office building. The floor is a grid of $n \times n$ square cells, and every two neighboring cells are separated by a wall. Each cell is either an office or part of the open space.

The company is soundproofing its offices. Every wall between two neighboring offices needs a soundproof panel, which can be mounted on either side of the wall, inside either of the two offices. A wall can hold at most one panel, on one side or the other. The doors and windows in the walls are no obstacle, since they are negligibly small and a panel never blocks them. Panels take up room, and it would be awkward for an office to hold more than one.

Can the company mount panels so that every wall between two neighboring offices has a panel and no office holds more than one panel? If it can, where should each panel be mounted?

## Input

The first line of input contains an integer $n$ ($2 \leq n \leq 50$), the size of the floor. The next $2n + 1$ lines draw the floor plan, with north at the top and west on the left. Each of them contains $2n + 1$ characters. On the first line of the floor plan, the third, and so on up to the last, the characters alternate between '+', a corner where walls meet, and '−', a wall running from west to east, starting and ending with '+'. On the lines between them, the characters alternate between '|', a wall running from north to south, and a cell, starting and ending with '|'. A cell is 'o' for an office or '.' for open space.

It is guaranteed that the floor has at least two offices.

## Output

Output the number of panels on the first line. Then output the floor plan in the same format, showing where to mount the panels. Draw a panel on every wall between two neighboring offices,

and leave every other character unchanged. A panel is drawn as an arrow pointing into the office that holds it. On a wall running from west to east, draw '^' for a panel in the office to the north and 'v' for a panel in the office to the south. On a wall running from north to south, draw '<' for a panel in the office to the west and '>' for a panel in the office to the east.

If there are multiple ways to mount the panels, you may output any of them.

If the panels cannot be mounted to meet the requirements, output `impossible` instead.

### Sample Input 1

```
2
+-+-+
|o|o|
+-+-+
|.|o|
+-+-+
```

### Sample Output 1

```
2
+-+-+
|o>o|
+-+v+
|.|o|
+-+-+
```

### Sample Input 2

```
4
+-+-+-+-+
|o|o|.|o|
+-+-+-+-+
|o|o|.|o|
+-+-+-+-+
|.|o|.|.|
+-+-+-+-+
|.|o|o|.|
+-+-+-+-+
```

### Sample Output 2

```
8
+-+-+-+-+
|o>o|.|o|
+^+v+-+v+
|o<o|.|o|
+-+v+-+-+
|.|o|.|.|
+-+v+-+-+
|.|o>o|.|
+-+-+-+-+
```

### Sample Input 3

```
3
+-+-+-+
|o|o|o|
+-+-+-+
|o|o|o|
+-+-+-+
|.|.|.|
+-+-+-+
```

### Sample Output 3

```
impossible
```

# Problem I
## Protein Shake Prognosis

Time Limit: 2 seconds  Memory Limit: 1GB

Before any big programming contest, you always make sure to pregame with some protein shakes. Unfortunately, right before the Big South regionals, you realize that some of your rival competitors have stolen most of the protein shakes your coach brought, in an effort to pregame themselves!

Determined to reclaim your rightful shakes, you plan to roust as many rival competitors as possible from the contest practice room, taking back your shakes in the process.

There are $n$ competitors. Each competitor has a quantifiable buffness and holds a number of stolen protein shakes. You also have a buffness and a number of protein shakes of your own. You can take any number of actions, one at a time. Each action is one of the following:

1. Consume a protein shake of yours. This increases your buffness by one.
2. Roust a competitor whose buffness must be strictly less than yours. You take all of that competitor's remaining protein shakes.

A rousted competitor leaves the room. However, immediately after you take an action, every competitor who is still in the room consumes one of their protein shakes, if they have at least one left. This increases their buffness by one and decreases their number of protein shakes by one.

You want to enact justice on as many competitors as possible. What is the maximum number of competitors you can roust, assuming you choose your actions optimally?

## Input

The first line of input contains three integers $n$, $b_u$ and $p_u$ ($1 \leq n \leq 2 \cdot 10^5$, $0 \leq b_u, p_u \leq 10^{12}$), where $n$ is the number of competitors, $b_u$ is your buffness, and $p_u$ is your number of protein shakes.

Each of the next $n$ lines contains two integers $b$ and $p$ ($0 \leq b, p \leq 10^{12}$), where $b$ is a competitor's buffness and $p$ is the number of stolen protein shakes they hold.

## Output

Output a single integer, the maximum number of competitors you can roust.

## Explanation of Sample Case 1

You can roust the first competitor and take their 3 protein shakes, then consume all of your protein shakes, then roust the second competitor. It is not possible to roust the third competitor.

| Sample Input 1 | Sample Output 1 |
|---|---|
| 3 4 3<br>3 3<br>6 2<br>10 3 | 2 |

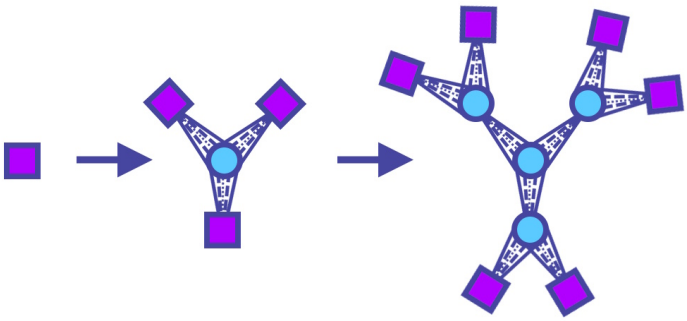
# Problem J
## Snow Many Ways

Time Limit: 1 second  Memory Limit: 1GB

As a scientist studying how snowflakes form, you make a breakthrough in describing the nucleation process with just a few simple rules!

Specifically, a snowflake can be modeled with two types of ice crystals: frozen and unfrozen crystals. When a snowflake is born, it begins as a single unfrozen ice crystal. Then, any number of times, the snowflake may "expand". During an expansion, every unfrozen ice crystal freezes into a frozen ice crystal and splits into exactly $x$ new unfrozen ice crystals. Here $x$ can be any integer strictly larger than one. The value of $x$ is the same for every unfrozen crystal within one expansion, but it does not have to be the same for every expansion.

The diagram below showcases an expansion of $x = 3$, then an expansion of $x = 2$ for a snowflake. The purple squares are unfrozen ice crystals, and the light blue circles are frozen ice crystals.



You're curious to understand all of the snowflakes that have exactly $u$ unfrozen crystals. There are *snow many ways* to make a snowflake with $u$ unfrozen crystals, but among all of those ways, what is the maximum number of frozen crystals that can exist on one of these snowflakes?

## Input

The input consists of a single line containing an integer $u$ ($2 \leq u \leq 10^9$), the number of unfrozen crystals on the snowflake.

## Output

Output a single integer, the maximum number of frozen crystals that can exist on a snowflake with exactly $u$ unfrozen crystals.

| Sample Input 1 | Sample Output 1 |
| --- | --- |
| 6 | 4 |

# Problem K
## 2SAT Queries

Time Limit: 1 second  Memory Limit: 1GB

You are playing a 2SAT guessing game with the judge. In each game, the judge has $n$ hidden boolean variables. A 2SAT clause over two variables $x$ and $y$ is one of the following:

- $x \vee y$
- $\neg x \vee y$
- $x \vee \neg y$
- $\neg x \vee \neg y$

The symbol $\neg$ indicates a negation. The negation $\neg x$ is true if $x$ is false, and false if $x$ is true. The symbol $\vee$ indicates a logical OR. A clause is true if at least one of its two parts is true.

In one query, you pick two **different variables** and form a 2SAT clause over them, and the judge replies with its value, true or false. You win the game by guessing every hidden value correctly.

You will play at most $1\,000$ games with the judge and want to guess right in all the games using no more than $10^4 + 400$ queries. The total number of hidden variables over all games is at most $10^4$.

## Interaction

This is an interactive problem.

At the start of the input, you will receive a single integer between $1$ and $1\,000$, inclusive, the number of games.

At the start of each game, you will receive a single integer $n$ ($2 \le n \le 10^4$), the number of hidden variables. The variables are labeled with integers from $1$ to $n$. The sum of $n$ over all games is at most $10^4$.

After reading $n$, your program should begin making queries. In each query, output a single line with two integers $a$ and $b$ ($1 \le |a|, |b| \le n$, $|a| \ne |b|$), the two literals of a clause. Let $x_i$ be the value of variable $i$. A positive integer $i$ represents $x_i$, and a negative integer $-i$ represents $\neg x_i$. For example, the query `-3 5` represents the clause $\neg x_3 \vee x_5$. You will then receive a single letter, which is `T` if the clause is true and `F` if it is false.

Once you have determined the value of every variable, output a single line with a string of $n$ letters, whose $i$-th letter is `T` if $x_i$ is true and `F` if $x_i$ is false. This answer does not count as a query. If every value is correct and there are more games, the judge sends $n$ for the next game, without a

separate acknowledgement. If the answer is invalid or any value is wrong, the judge will not send any further input, and you will receive a Wrong Answer verdict. After answering the last game, your program should terminate.

You may make at most $10^4 + 400$ queries in total over all games.

If a query is invalid or exceeds the query limit, you will receive the letter `X` instead of `T` or `F` in reply to that query. Upon receiving `X`, your program should terminate immediately, and you will receive a Wrong Answer verdict. The judge will not send any further input, and reading more input may cause your program to hang and receive a Time Limit Exceeded verdict.

The judge is non-adaptive: all hidden assignments are fixed before the interaction begins.

## Notes

Do not forget to flush the output after printing each query and each answer. Do not output anything extra after the interaction ends.

The sample interaction only shows the format and does not represent a solving strategy.

A local testing tool, `testing_tool.py`, is provided in the attachments. See its header for usage instructions and the input file format.

**Sample Interaction 1**

| Read | Write |
|---|---|
| 2<br>3 | |
| | 1 3 |
| T | |
| | -1 2 |
| F | |
| | TFT |
| 4 | |
| | -4 2 |
| T | |
| | 1 4 |
| F | |
| | -3 -2 |
| T | |
| | FFTF |

# Problem L
## Volleyball Scoring

Time Limit: 1 second  Memory Limit: 1GB

In volleyball, the game consists of a series of rallies, in which the team that wins the rally scores one point and the other team's score stays the same. Whichever team scored in the previous rally always serves next, but either team is always able to score in any rally. In this problem, there is no winning score: play continues no matter how many points either team has.

The game score is kept on the scoreboard. For awareness of the score and for brevity, before each serve, only the last digit of each team's score is called out, and the team that is about to serve always has their score called out first. For example, the score of a team with $9$ points serving against a team with $13$ points is called out as "$9$ $3$".

Given the current scores of the two teams, determine the minimum possible number of rallies played until the score called out is "$6$ $7$".

### Input

The input consists of a single line containing two integers between $0$ and $100$, inclusive: the score of the serving team and the score of the receiving team before the next serve.

### Output

Output a single integer, the minimum possible number of rallies played until the score called out is "$6$ $7$".

### Explanation of Sample Case 2

The answer is not $1$ because if the receiving team wins the next rally, they also serve next, so the score is called out as "$7$ $6$".

| Sample Input 1 | Sample Output 1 |
|---|---|
| 6 17 | 0 |

| Sample Input 2 | Sample Output 2 |
|---|---|
| 6 6 | 11 |

| Sample Input 3 | Sample Output 3 |
|---|---|
| 5 7 | 1 |

| Sample Input 4 | Sample Output 4 |
|---|---|
| 7 8 | 8 |

# Problem M
## Mushroom Island

Time Limit: 2 seconds  Memory Limit: 1GB

Many blocky adventurers often embark on long journeys looking for the rarest biome in the world: the Mushroom Island! This beautiful island is incredibly hard to find, lying alone in the middle of a vast ocean.

You are given a map of the world. On this map, every biome (excluding oceans) is marked with a *dot* at its center. Every dot is at most $100$ miles away from another dot, except the one marking the Mushroom Island, which is at least $10\,000$ miles away from every other dot! All distances are Euclidean.

All of the dots are indistinguishable, so you cannot immediately tell which dot corresponds to the Mushroom Island. However, given the coordinates of each dot and the distance rules above, can you identify which dot is the Mushroom Island?

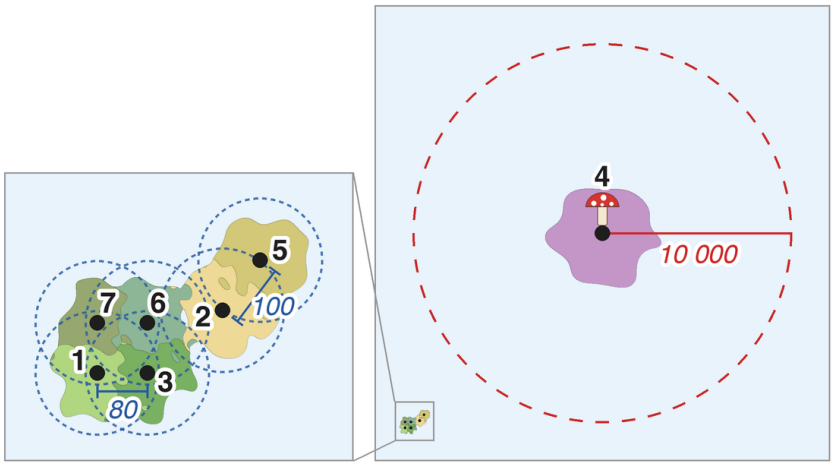



Figure M.1: Illustration of Sample 1. The dots near the origin are magnified on the left. The biome shapes are only decorative and do not matter. Every dot near the origin is at most $100$ miles away from another dot, which lies on or inside its dashed circle of radius $100$ miles, while dot $4$, the Mushroom Island, is at least $10\,000$ miles away from every other dot.

## Input

The first line of input contains an integer $n$ ($3 \le n \le 2 \cdot 10^5$), the number of dots. Each of the next $n$ lines contains two integers between $1$ and $10^5$, inclusive, the $x$- and $y$-coordinates of a dot in miles. No two dots are at the same location. The dots are numbered from $1$ to $n$ in the order they are given. It is guaranteed that the locations of the dots satisfy the conditions described above.

## Output

Output a single integer, the number of the dot that marks the Mushroom Island.

### Sample Input 1

```
7
10 10
210 110
90 10
10000 10000
270 190
90 90
10 90
```

### Sample Output 1

```
4
```

### Sample Input 2

```
5
100 190
150 170
49000 5960
99900 99950
99960 99910
```

### Sample Output 2

```
3
```

This page is intentionally left blank.

2026 ICPC Southeast USA Regional Division 1

Back Cover