IDI Open
Programming Contest
March 1st, 2025

Problem Set

Jury and Problem Writers

Jean Niklas L'orange (Head Judge)

Tobias Meyer Andersen

Dilawar Mahmood

Test Solvers

Johan Sokrates Wind

Rules

TL;DR: Teams of up to three persons try to solve as many problems as possible from a set, without external help.

Before the contest begins, you are allowed to log in on your assigned computer, and log in on the submission system. You may do nothing else with the computer (such as starting to write code). You may not touch the problem set before the contest has started.

Contestants are *only* allowed to communicate with members of their own team, and the organisers of the contest. You may not surf the web (except for allowed content), read e-mail, chat on Slack, or similar things. The only network traffic you may generate is from submitting problem solutions, and access to content specified by the local organisers.

- What you may bring to the contest floor:
  - Any written material (Books, manuals, handwritten notes, printed notes, etc).
  - Pens, pencils, blank paper, stapler and other useful non-electronic office equipment.
  - NO program using generative AI, such as Github Copilot or ChatGPT.
  - NO material in electronic form (CDs, USB pen and so on).
  - NO electronic devices (Cellular phone, PDA and so on).
- What you may use during the contest:
  - What you brought to the contest floor (see above).
  - Your assigned (single) computer.
  - The specified system for submitting solutions: `https://idio25.kattis.com`
  - Printers designated by the organiser.
  - The external documentation for your language of choice. This is documented in the tutorial for your language at `https://support.kattis.com`
  - All compilers and IDEs pre-installed on your assigned computer
  - Non-programmable tools which are a natural part of the working environment (such as `diff` and `less`).

The problem set consists of a number of problems (usually 8-12). The problem set will be in English, and given to the participating teams when the contest begins. For each of these problems, you are to write a program supported by the Kattis system. The jury guarantees that each problem is solvable in C, C++, Java and Python 3. No guarantees for other languages are given due to the large number of allowed languages, however the jury guarantees that for every language there is at least one problem solvable in that language. It has always been the case that several of the problems were solvable in all available languages, but there is no guarantee of this.

Your programs should read from standard in and write to standard out. See `https://support.kattis.com` for a tutorial and the compiler options for your language of choice.

The team that solves the most problems correctly wins. If two teams solve the same number of problems, the one with the lowest total time wins. If two top teams end up with the same number of problems solved and the same total time, then the team with the lowest time on a single problem is ranked higher. If two teams solve the same number of problems, with the same total time, and the same time on all problems, it is a draw. The time for a given problem is the time from the beginning of the contest to the time when the first correct solution was submitted, plus 20 minutes for each incorrect submission of that problem. The total time is the sum of the times for all solved problems, meaning you will not get extra time for a problem you never submit a correct solution to.

If you think some problem is ambiguous or underspecified, you may ask the judges for a clarification request through the Kattis system. The most likely response is “No comment, read problem statement”, indicating that the answer can be deduced by carefully reading the problem statement or by checking the sample test cases given in the problem, or that the answer to the question is simply irrelevant to solving the problem.

Input Validation

In general we are lenient with small formatting errors in the output, in particular whitespace errors within reason. But not printing any spaces at all (e.g. missing the space in the string “`1 2`” so that it becomes “`12`”) is typically not accepted. The safest way to get accepted is to follow the output format exactly.

For problems with floating point output, we only require that your output is correct up to some error tolerance. For example, if the problem requires the output to be within either absolute or relative error of 10^{-4}, this means that

- If the correct answer is 0.05, any answer between 0.0499 and .0501 will be accepted.
- If the correct answer is 500, any answer between 499.95 and 500.05 will be accepted.

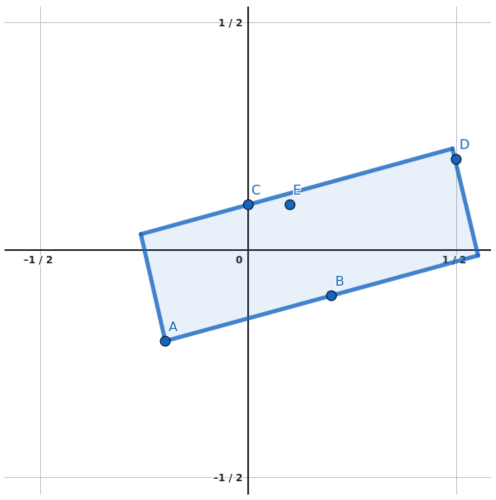
Any reasonable format for floating point numbers is acceptable. For instance, “`17.000000`”, “`0.17e2`”, and “`17`” are all acceptable ways of formatting the number 17. For the definition of reasonable, please use your common sense.

Tips

- Tear the problem set apart and share the problems among you.
- Problems are **not** sorted by difficulty.
- Try solving the easy problems first. Two problems in this set are tagged with “(Easy)” to help point you in the right direction.
- If your solution fails on a problem, you can print your program and debug it on paper while you let someone else work on a different problem on the computer.
- All problems are guaranteed to be solvable in C, C++, Java and Python 3
- Look at the scoreboard if you are unsure which problem to work on next.

Ant Attack
Problem ID: antattack

Gregor is out in the back yard studying all the insects moving around. In particular, he is inspecting the myriad of ants scattered out on a large flat stone. Gregor wants to collect all the ants to study them in his ant terrarium. Given where all the ants are placed right now, Gregor wants to know what is the smallest box that could capture all the ants he sees right now at the same time. Smallest is measured in the area of the bottom of the rectangular box. He will capture all the ants simulatenously, by placing the box on the stone, and may therefore rotate the box at any angle to minimize the size of the box. The solution to the sample input is illustrated in the figure.



Minimal rectangle for the sample input.

Input

The first line of input contains the number N, which is the number of ants on the stone. On the following N lines contain x_i and y_i which is coordinates of the i'th ant in a flat cartesian plane.

It is given that no points coincide, and that the solution area is non-zero.

Output

Output the smallest area of a rectangle that contains all the ants. Ants may be located arbitrarily close to an edge of the box, meaning they can effectively be on the edge of the box. The answer must be accurate to within an absolute and relative error of 10^{-5}.

Limits

- $1 \leq N \leq 10^5$
- $-1 \leq x_i, y_i \leq 1$

Sample Input 1	Sample Output 1

5
-0.2 -0.2
0.2 -0.1
0.0 0.1
0.5 0.2
0.1 0.1
	0.18823529411764706

Ballfoot Battles
Problem ID: ballfootbattles

You love the sport of ballfoot. It is the futuristic version of football where many teams can play at once! There is still only one ball, so one goal is scored at the time, and each team starts with zero goals. Being busy completing the mandatory assignments for your algorithms and datastructures course, you unfortunately do not have time to watch todays game between all of your T favorite teams!

The best you can do is check your futuristic phone from time to time and see what the current score is. You will therefore know the score at N arbitrary moments in the match, and you always check what the final result is. Having seen the scoreboard a few times throughout the match makes you wonder, in how many ways could the match have been played? This question is about how many ways the game could have gone in terms of the order of which team scores which goal. If one team wins four to zero, it must have been 4 goals in a row for that team. If a team wins two to one, there are three different sequences of goals that could reach that outcome.

Input

The first line contains two integers. T describes the number of teams participating in the match. N describes the number of times you check you phone and peek at the score. Then follows N lines, each of which containing the T integers $g_{t,n}$, which is the number of goals team t has at time point n in the match. The last observation is always the final scoreline.

Output

Output the number of ways the match could have given the observations. The answer can be very large, so output the result modulo $10^9 + 7$.

Limits

- $1 \leq T \leq 100$
- $1 \leq N \leq 1000$
- $1 \leq g_{t,n} \leq 50000$

Sample Input 1

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

Sample Output 1

```
3
```

Crate Chucking
Problem ID: cratechucking

You are playing a game on a grid. You must move a 2x1x1 crate from the starting cell where it starts in an upright position, to the target cell where it must also be in an upright position. The game is played by toppling over the crate in one of the four cardinal directions on a cartesian grid. Toppling over a standing create means it will occupy the two next cells in the direction you toppled it. A crate lying down can be toppled in two ways, either such that it keeps lying on its side, or in a way that makes it stand up. Toppling a standing crate will always make it lie down. When toppling the create over, it will occupy the neighboring squares in the direction it was pushed. In the example figure there are four consecutive valid states to illustrate how the crate moves on the grid.

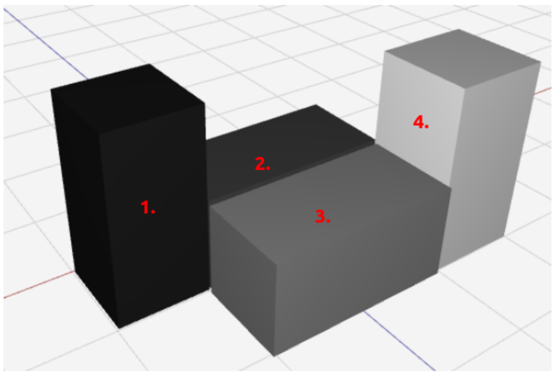


Illustration of four consecutive valid states of the crate.

To make the game more interesting, the board it is played on contains obstacles. Some cells are therefore blocked, meaning the crate may never occupy those squares.

In addition to reaching the final cell in an upright position, you want to minimize the amount of energy used. To topple a standing crate over only costs one unit of energy. Toppling over a lying crate such that is still lies sideways costs two units of energy. Taking it from a lying to standing position costs 3 units of energy.

You use your keyboard's four arrow keys to indicate in which direction you want to move the crate next. There is only one problem, your keyboard is broken! It is broken in a peculiar way where you may never press the same arrow key twice in a row. In the starting position you may assume no arrow key was the previous button press.

What is the least amount of energy you must use to reach the final position, given your faulty keyboard?

Input

The first line of input contains two numbers. H, which is the height of the cartesian grid, and W, which is the width. H lines follow which are strings of W characters each. The strings contain the following characters.

- ".", indicating that the cell free to move on
- "*B*", indicating that the cell is blocked and unavailable to the crate
- "*S*", indicating the starting position of the crate, where it stands at the start of the game
- "*E*", indicating the ending position of the crate, where it must stand at the end of the game

Output

Output the least amount of energy used to move from the starting to the ending position. If no solution exists under the given limitations you must output "*impossible*".

Limits

- $1 \leq H, W \leq 200$

Sample Input 1

```
4 4
S...
B...
B...
B..E
```

Sample Output 1

```
16
```

Sample Input 2

```
2 3
S..
E..
```

Sample Output 2

```
6
```

Divisor Detective
Problem ID: divisordetective

Ådne is a software engineer who loves solving algorithmic puzzles. His latest project is a tool called "Divisor Detective" that analyzes number patterns.
While testing the tool with his colleague, they encountered an interesting mathematical challenge:

> "For each integer from 1 to N, we need to count how many divisors it has, then sum up all those counts.
> But with these large inputs, we need an efficient algorithm to compute this quickly!"

Can you help them determine the sum of the divisor counts for all integers from 1 to N?

Input

The input consists of a single line containing one integer N.

Output

Output a single integer: the sum of the number of positive divisors of every integer from 1 up to N.

Limits

- $1 \leq N \leq 10^{10}$

Sample Input 1	Sample Output 1
`6`	`14`

Sample Input 2	Sample Output 2
`10`	`27`

Exclusive Equation

Problem ID: exclusiveequation

You and your friends are preparing for an exciting intellectual challenge this weekend. Each friend selects a number and writes it on multiple pieces of paper. You are given all these pieces of paper, along with a target number. Your task is to determine whether it is possible to produce the target number by combining the given numbers on the pieces of paper using only the exclusive-or (XOR) operator. You may use as many pieces of paper as you like, but only each piece once. Before using any pieces of paper, the current value produced is zero.

The XOR operator is a binary operation that takes two numbers and operates on their binary representations. It performs a logical XOR on each pair of corresponding bits: the result is 1 if the bits are different, and 0 if they are the same. For example, 5 XOR 3 = 6 because:

5 in binary is 101

3 in binary is 011

XOR result is 110, which is 6 in decimal.

Given the numbers on the pieces of paper and the target number, can you determine if the target can be achieved using the XOR operation?

Input

The first line contains two integers N (number of participants) and T (target number). Then follow N lines, each containing an integer N_i (participant's number) and K_i (number of copies available, $K_i \geq 1$).

Output

Output "possible", if it is possible to produce the target number, otherwise output "impossible".

Limits

- $1 \leq N \leq 35$
- $0 \leq T < 2^{64}$
- $1 \leq N_i < 2^{64}$
- $1 \leq K_i \leq 100$

Sample Input 1

```
4 12
1 2
5 2
11 2
8 2
```

Sample Output 1

```
possible
```

Sample Input 2

```
4 12
1 2
5 2
11 2
4 2
```

Sample Output 2

```
impossible
```

Flint Flinger
Problem ID: flintflinger

During medieval times, the catapult was a common siege weapon used to attack neighboring cities at a distance. Before they were invented, however, the placement of cities did not account for this future technology. Some cities therefore settled very close to each other, which now puts them under the constant threat of being launched large boulders of flint at!

Each city needs to evaluate how large the threat is. They do so by counting the number of other cities that can launch flint boulders at them. To calculate this correctly they also take into consideration how mature the local technology is, which causes some variation in the range of the catapults.

Input

The first line of input contains the number N, the number of cities. Then follows N lines, with three numbers each. The first two numbers are the x_i and y_i coordinates of the city, and the third number, r_i is the range of the catapult in that city. You may consider the cities as points in the cartesian plane, and it is guaranteed that no two cities coincide on the same coordinate. These three decimal numbers are separated by a single space and always have one digit after the decimal point.

Output

For each city, output the number of cities that can launch a flint boulder at it on a separate line.

Limits

- $1 \leq N \leq 100$
- $0 \leq x_i, y_i, r_i \leq 100$

Sample Input 1

```
4
10.0 10.0 11.0
20.0 10.0 2.0
50.0 50.0 100.0
90.0 90.0 10.0
```

Sample Output 1

```
1
2
0
1
```

Sample Input 2

```
2
0.2 0.0 0.9
1.1 0.0 0.9
```

Sample Output 2

```
1
1
```

Milk Mystery
Problem ID: milkmystery

Megatron, a mischievous orange kitten, has discovered N bowls of milk in the kitchen. Being a very picky cat, he has developed a sophisticated rating system for milk - each bowl has a "deliciousness score" that represents how tasty he thinks it will be. The problem is, Megatron's human has caught on to his milk-stealing habits and installed a security system. The system will only let Megatron drink from exactly K consecutive bowls before it activates. Megatron needs to figure out which sequence of K consecutive bowls will give him the maximum total deliciousness before the alarm goes off!

Input

The first line contains two integers N and K, where N is the total number of milk bowls and K is how many consecutive bowls Megatron can drink from. The second line contains N integers $d_1, d_2, \ldots, d_N$, where d_i represents the deliciousness score that Megatron assigns to the ith bowl of milk.

Output

Output a single integer - the maximum sum of deliciousness scores that Megatron can achieve by drinking from K consecutive bowls.

Limits

- $1 \leq K \leq N \leq 100$
- $0 \leq d_i \leq 100$

Sample Input 1	Sample Output 1
5 2 1 3 2 5 1	7

Sample Input 2	Sample Output 2
4 3 2 2 2 2	6

Hopping Haven
Problem ID: hoppinghaven

Oh no! The Harrowfall Horde has, once more, decided to launch their magic squares against us. And again, they've sealed us within a great rectangular wall!

While it won't hurt our people – magic squares never do – every hit will hurt our economy. They'll manage to damage houses or crops, and that is really annoying. Fortunately, a magic square will only deal damage the first time it hits the kingdom: Subsequent hits from the same square do no damage.

Since we captured some magic squares last time, we've managed to uncover their secrets. After magic square i appears, it teleports Δx_i units horizontally and Δy_i units vertically on every odd-numbered second. Also, instead of colliding with each other, it seems like multiple magic squares are able to occupy the same tile. And when a magic square moves beyond the boundaries of the great rectangular wall, it vanishes. Thanks to our spies, we also know when and where magic square i will appear: It seems to always be somewhere on top of the great rectangular wall.

This by itself won't help us much: After all, our rectangular kingdom is immovable...or is it? Thanks to the tireless efforts by the Institute of Dimensional Invocation (IDI), we have successfully reverse-engineered the magic behind the squares. At last, our kingdom has legs!

On positive even-numbered seconds, we can either remain in place or shift exactly one unit in either a horizontal or vertical direction, though never both at once. While the kingdom can now move, it cannot be placed on top of water nor the great wall – unlike the magic squares, which can.

Given this information, you, the shell-script shaman, have been tasked with determining how many magic squares can be avoided. But the king isn't just interested in the number of magic squares the kingdom can avoid – he also wants to know the number of distinct movement sequences that achieve this. According to him, reporting this to the Hopping Haven will severely damage their morale.

Input

The first line of the input contains the integers W, H and N, the width and height of the map, as well as the number of magic squares.

Then follow H lines, each with exactly W characters each, representing the map. The map will only contain the following characters:

- '`.`', representing land
- '`~`', representing water
- '`#`', representing a piece of the great rectangular wall
- '`x`', representing a piece of our rectangular kingdom

Finally, N lines follow, each representing a single magic square. Each line contains 5 integers, t_i, x_i, y_i, Δx_i, Δy_i, denoting the time t_i the magic square appears at $\langle x_i, y_i \rangle$, teleporting $\langle \Delta x_i, \Delta y_i \rangle$ units every odd-numbered second.

Rows are indexed from the top, so the first row is $y = 0$, the second is $y = 1$, and so on.

Output

Output the number of magic squares the kingdom can evade and the number of distinct move sequences that achieve this, assuming time starts at second 0. As the number of move sequences may be very large, output both numbers modulo 1 000 000 007.

The move sequence ends at the latest time at which any magic square vanishes.

Limits

- $3 \leq W, H \leq 50$
- $0 < N < 100$
- There is at most 6 active magic squares at any given time

- The border of the map will only contain '#' characters, and '#' characters will only appear there
- All 'x' characters on the map form a rectangle
- $0 \leq t_i < 200$, and t_i is always an even number
- $\langle x_i, y_i \rangle$ is always on the border of the map
- $|\Delta x_i| < W$ and $|\Delta y_i| < H$
- $0 < |\Delta x_i| + |\Delta y_i|$

Sample Input 1

```
5 5 1
#####
#..~#
#xx.#
#xx.#
#####
0 1 0 0 1
```

Sample Output 1

```
1 2
```

Sample Input 2

```
5 5 2
#####
#..~#
#xx.#
#xx.#
#####
2 1 0 0 1
10 3 4 0 -1
```

Sample Output 2

```
2 48
```

Sample Input 3

```
5 5 3
#####
#...#
#...#
#xx.#
#####
10 0 0 1 0
14 2 0 0 1
14 2 4 0 -1
```

Sample Output 3

```
1 627232
```

Incinerating Incantations
Problem ID: incineratingincantations

Ivar is an experienced mage and want to help his community celebrating Saint John's Eve. They want a big bonfire, and with his spells, he can help them make the bonfire even bigger. Ivar has in total N spells he can cast, but can only cast each spell at most once. Additionally, he only has M mana, so he may not be able to cast all the spells he knows.

And the spells are complicated and tedious to work with. They affect many different things – including future spells you may want to cast. More specifically, when casting a spell i, the following happens in this exact order:

1. It consumes m_i mana (Ivar can't cast it if he does not have enough mana)
2. It increases the fire power by f_i
3. It increases the mana cost of the **next** spell cast by m_i^{next}
4. It increases the mana cost of **all** following spells cast by m_i^{all}
5. It increases the fire power of the **next** spell cast by f_i^{next}
6. It increases the fire power of **all** following spells cast by f_i^{all}

Of course, the cost of a spell cannot go negative: If a spell costs 1 mana to cast and is reduced by 2 (i.e. increased by -2), it will cost 0 mana to cast, not give you back 1 mana when casting. Similarly, the fire power of a spell cannot go negative either.

Ivar isn't entirely sure which order he should cast his spells however, and he's come to you to get some help on the matter. Can you help Ivar find out how much additional fire power he can add to the bonfire, if he optimally chooses which spells to cast, and in what order?

Input

The first line contains two integers N and M, specifying the total number of spells Ivar knows, and his current mana. Then follow N lines, specifying a spell. Each spell line contain 6 integers m_i, f_i, m_i^{next}, m_i^{all}, f_i^{next}, and f_i^{all}, as described above.

Output

Output the most fire power Ivar can emit.

Limits

- $0 < N \leq 9$
- $0 \leq M \leq 100$
- $0 \leq m_i, f_i \leq 200$
- $-200 \leq m_i^{next}, m_i^{all}, f_i^{next}, f_i^{all} \leq 200$

Sample Input 1	Sample Output 1

3 3
10 10 0 0 0 0
1 0 -3 -3 -10 -2
7 4 -8 0 0 -2
	6

Sample Input 2	**Sample Output 2**

```
5 3
0 0 0 0 0 0
10 0 0 1 1 3
0 0 0 0 0 0
0 0 -8 -1 0 -100
0 0 0 0 0 0
```
	`7`

Timely Treatments
Problem ID: timelytreatments

Dr. Kassandra is a busy doctor in a hospital in Barcelona. She has N patients waiting for treatment, where each patient i requires t_i time units for their treatment and must be fully treated by deadline d_i.
Kassandra can only treat one patient at a time. She starts working at time 0 and can treat patients in any order, continuing without breaks if she wishes. Once she starts treating a patient, she must complete their treatment without interruption.
For example, if she has three patients:

- Patient 1: $t_1 = 3$, $d_1 = 5$
- Patient 2: $t_2 = 2$, $d_2 = 2$
- Patient 3: $t_3 = 1$, $d_3 = 3$

She could treat Patient 2 first (finishing at time 2), then Patient 3 (finishing at time 3). Note that Patient 3 must be treated continuously for 1 time unit, so starting at time 2 means they would finish at time 3. While Patient 1 could theoretically be treated within their deadline if scheduled first, treating Patients 2 and 3 instead leads to the maximum possible number of treated patients.
Your task is to help Kassandra determine the maximum number of patients she can successfully treat while meeting their deadlines.

Input

The first line contains an integer N, the number of patients.
Each of the following N lines contains two integers, t_i and d_i, where t_i is the time required to treat patient i, and d_i is the deadline by which their treatment must be completed.

Output

Output a single integer: the maximum number of patients whose treatments can be completed by their respective deadlines when scheduled optimally.

Limits

- $1 \leq N \leq 10^6$
- $1 \leq t_i \leq 10^6$
- $1 \leq d_i \leq 10^9$

Sample Input 1

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

Sample Output 1

```
2
```

Sample Input 2

```
5
2 3
2 1
5 10
3 10
2 2
```

Sample Output 2

```
3
```