Problem author: **Peter Fulla**, problem preparation: **Peter Fulla, Daniel Bundala**

The MP3 Player

Georg's new MP3 player has many interesting features, one of them being the key lock. All the keys are locked after more than T seconds of inactivity. After the key lock is engaged, no key performs its original function, but if any key is pressed, the key lock is disengaged.

For example, assume that $T = 5$ and the player is currently locked. Georg presses the key A, waits for 3 seconds, presses the key B, waits for 5 seconds, presses C, waits for 6 seconds, and presses D. In this case only the keys B and C perform their regular functions. Note that the keys became locked between C and D was pressed.

Sound level of the MP3 player is controlled by the `+` and `-` keys, increasing and decreasing volume by 1 unit respectively. The sound level is an integer between 0 and V_{max}. Pressing the `+` key at volume V_{max} or pressing the `-` key at volume 0 leaves the volume unchanged.

Task specification

Georg does not know the value of T. He wanted to find it by an experiment. Starting with a locked keyboard, he pressed a sequence of N `+` and `-` keys. At the end of the experiment Georg read the final volume from the player's display. Unfortunately, he forgot to note the volume before his first keypress. For the purpose of this task, the unknown initial volume will be denoted V_1 and the known final volume will be denoted V_2.

You are given the value V_2 and a list of keystrokes in the order in which Georg made them. For each key, you are given the type of the key (`+` or `-`) and the number of seconds from the beginning of the experiment to the moment when the key was pressed. The task is to find the largest possible **integer** value of T which is consistent with the outcome of the experiment.

Input specification

The first line of the input contains three space-separated integers N, V_{max} and V_2 ($0 \leq V_2 \leq V_{max}$). Each of the next N lines contains a description of one key in the sequence: a character `+` or `-`, a space and an integer C_i ($0 \leq C_i \leq 2 \cdot 10^9$), the number of seconds from the beginning of the experiment. You may assume that the keypresses are in sorted order and that all times are distinct (i.e., $C_i < C_{i+1}$ for all $1 \leq i < N$).

Constraints

You may assume that $2 \leq N \leq 100\,000$ and $2 \leq V_{max} \leq 5\,000$.

In test cases worth 40 points $N \leq 4\,000$.

In test cases worth 70 points $N \cdot V_{max} \leq 400\,000$.

Output specification

If T can be arbitrarily large, output a single line containing the word "`infinity`" (quotes for clarity).

Otherwise, output a single line containing two integers T and V_1 separated by a single space.

The values must be such that carrying out the experiment with locking time T starting at volume V_1 gives the final volume V_2. If there are multiple possible answers, output the one with the largest T; if there are still multiple possible answers, output the one with the largest V_1.

(Note that at least one solution always exists: for $T = 0$ none of the keys performs its action, so it suffices to take $V_1 = V_2$.)

Examples

input:

```
6 4 3
- 0
+ 8
+ 9
+ 13
- 19
- 24
```

output:

```
5 4
```

For T = 5 the keys perform the following actions: unlock, unlock, `+`*,* `+`*, unlock,* `-`*.*

For any $V_1 \in \{2, 3, 4\}$ *we would get* $V_2 = 3$*. Note that the output contains the largest possible* V_1*.*

For $T \geq 6$ *the last two keystrokes will both be active, hence it will be impossible to* have $V_2 = 3$.

input:

```
3 10 10
+ 1
+ 2
+ 47
```

output:

```
infinity
```

If $V_1 = 10$ *then for any T we'll have* $V_2 = 10$*.*

Problem author: **Lukáš Poláček**, problem preparation: **Lukáš Poláček, Daniel Bundala, Michal Forišek, Ján Katrenič**

PIN

Martin has just been hired as a computer administrator in a big company. The company did not change its authorization system since 1980s. Every person has a four-digit personal identification number (PIN). Nobody uses usernames or passwords, you can login just by typing your PIN. As the company grew, they added the possibility to use letters as well, but the length of the PIN remained the same.

Martin is not happy with the situation. Suppose there are people whose PINs differ only at a single place, for example `61ab` and `62ab`. If the first person accidentally presses 2 instead of 1, the system would still let him in. Martin would like to make the statistics about the PINs currently in use, in particular, compute the number of pairs of PINs that differ at 1, 2, 3 or 4 positions. He hopes that these numbers will be alarming enough to convince his boss to invest in a better system.

Task specification

Given the list of PINs and an integer *D*, find the number of pairs of PINs that differ at exactly *D* positions.

Input specification

The first line of the input contains two space-separated positive integers *N* and *D*, where *N* is the number of PINs and *D* is the chosen number of differences. Each of the following *N* lines contains a single PIN.

Constraints

You may assume that in all test cases $2 \le N \le 50\,000$ and $1 \le D \le 4$.

Each PIN is of length 4 and each character is either a digit or a lowercase letter between 'a' and 'z', inclusive. You may assume that all PINs in the input are different.

In test cases worth 15 points, $N \le 2000$.

In test cases worth 60 points, $D \le 2$. Out of those, in test cases worth 30 points, *D* = 1.

In test cases worth 75 points, every PIN will only consist of digits or lowercase letters between 'a' and 'f', inclusive. Thus it can be viewed as a hexadecimal number.

Output specification

Output a single line with a single number: the number of pairs of PINs that differ at **exactly** *D* positions.

Examples

input:

```
4 1
0000
a010
0202
a0e2
```

output:

```
0
```

For these PINs each pair of PINs differs at more than one position.

input:

```
4 2
0000
a010
0202
a0e2
```

output:

```
3
```

There are three pairs that differ at exactly 2 positions: (0000,a010), (0000,0202), and (a010,a0e2).

Problem author: **Michal Forišek**, problem preparation: **Monika Steinová, Michal Forišek, Michal Nánási**

A Huge Tower

The ancient Babylonians decided to build a huge tower. The tower consists of N cubic building blocks that are stacked one onto another. The Babylonians gathered many building blocks of various sizes from all over the country. From their last unsuccessful attempt they have learned that if they put a large block directly onto a much smaller block, the tower will fall.

Task specification

Each two building blocks are different, even if they have the same size. For each building block you are given its side length. You are also given an integer D with the following meaning: you are not allowed to put block A directly onto block B if the side length of A is strictly larger than D plus the side length of B.

Compute the number of different ways in which it is possible to build the tower using **all** the building blocks. Since this number can be very large, output the result modulo $10^9 + 9$.

Input specification

The first line of the input contains two positive integers N and D: the number of building blocks and the tolerance respectively.

The second line contains N space-separated integers; each represents the size of one building block.

Constraints

All numbers in the input files are positive integers not exceeding 10^9.

N is always at least 2.

In test cases worth 70 points N will be at most 70.

Out of those, in test cases worth 45 points, N will be at most 20.

Out of those, in test cases worth 10 points, N will be at most 10.

For some of the test cases the total number of valid towers will not exceed 1 000 000. These test cases are worth 30 points in total.

For the last six test cases (worth 30 points) the value of N is larger than 70. No upper bound on N is given for these test cases.

Output specification

Output a single line containing a single integer: the number of towers that can be built, modulo 1 000 000 009.

Examples

input:

```
4 1
1 2 3 100
```

output:

```
4
```

We can arrange the first three blocks in any order, except for 2,1,3 or 1,3,2. The last block has to be at the bottom.

input:

```
6 9
10 20 20 10 10 20
```

output:

```
36
```

We are not allowed to put a cube of size 20 onto a cube of size 10. There are six ways to order the cubes of size 10, and six ways to order the cubes of size 20.
