

Git Status (git)

Today William was looking for a simple solution to backup his entire hard drive. Basically, he wants to make a full copy of the entire file system structure, starting from the root / and copying every subdirectory, recursively. To do this, normally you would use something like *rsync*, however he decided that *git* was the best software for the job¹.

The way *git* works is that it monitors which files are changed, and at any point in time you can choose to *commit* the latest changes and finally *push* them to a remote repository: backup done!

To make his life easier, William wants to write a program that will show which files were changed since the last commit.

Yes, there is a *git* command for it (*git status*) but it's not really optimal: if a folder **only** contains files that were modified, the command will still list each individual file, like this:

```
/some-directory/some-file
/some-directory/some-other-file
```

In such a case, William prefers to only list the directory which contains only-modified files:

```
/some-directory
```

Formally, you will be given a list of N files, each with its full path (starting with the / character) and an indication of whether that file was modified or not. William wants a program that, from this list, will produce a new “compressed” list containing only the paths that changed, and, if a folder contains **only** files that changed, then the list should only mention the folder, and not the files inside of it.


Note: when we say that a folder “contains” files we are referring also to the files that exist in subdirectories. For example, if you have these files:

```
/dir-a/this-was-changed
/dir-a/this-was-not-changed
/dir-b/sub/this-was-changed
/dir-b/sub/this-was-changed-too
```

Then the “compressed” list of changed files would simply be:

```
/dir-a/this-was-changed
/dir-b
```

Since “/dir-b” only contains modified files (even if they are not direct children) we can safely omit the “/sub” subdirectory. Notice, also, that if the “/dir-a/this-was-not-changed” file were to be modified too, then we would simply output “/” as the compressed list: neat!

 Among the attachments of this task you may find a template file `git.*` with a sample incomplete implementation.



¹It's worth mentioning that he has lots of heavy files, like music and movies, but Edoardo reassured him that *git* is absolutely the right tool for big non-text files.

Input

The first line contains the only integer N . Each of the following N lines is formed by a number M_i (equal to 1 if the file was modified, 0 if it was not) and a string P_i (the full path of the file).

Output






You need to write one line for each entry of the “compressed” list of modified files, as shown in the examples. The lines can be printed in any order.

Constraints

- $1 \leq N \leq 100\,000$, and the length of each P_i is between 2 and 1000 characters.
- M_i is either 0 or 1.
- P_i only contains lowercase letters, dashes (-), and slashes (/). There are no spaces.
- P_i is a valid path: it starts with a slash, and it does not contain two consecutive slashes.
- P_i always identifies a *file*, i.e. there are no empty folders.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- Subtask 1 (0 points) Examples.

- Subtask 2 (10 points) Each path has **only one** slash (there are no subdirectories).

- Subtask 3 (30 points) Each path has **up to two** slashes (max nesting level is 1).

- Subtask 4 (30 points) $N \leq 100$.

- Subtask 5 (30 points) No additional limitations.


Examples

input	output
4 1 /dir-a/this-was-changed 0 /dir-a/this-was-not-changed 1 /dir-b/sub/this-was-changed 1 /dir-b/sub/this-was-changed-too	/dir-a/this-was-changed /dir-b
4 1 /dir-a/this-was-changed 1 /dir-a/this-was-now-changed 1 /dir-b/sub/this-was-changed 1 /dir-b/sub/this-was-changed-too	/

Picarats Collection (picarats)

Professor Layton¹ is a series of puzzle games where the Professor and his small assistant Luke are challenged with a mystery to solve.

During the investigation the duo walks around the city and encounters various puzzles of different levels of difficulty: some of them are very easy, others are very challenging. Each of the N puzzles has a known positive level of difficulty P_i , expressed in *Picarats*, that denotes how hard it is to solve without any help.

Fortunately, during their journey they can also find *hint coins*. A coin is a magic aid that, when used, reduces the difficulty of a puzzle.

After the Professor and his assistant solve a puzzle, they can proceed to the next one, collecting all the *hint coins* they find in the journey. The decision on where to go next is entirely up to them, with the only requirement that after solving a puzzle one cannot come back to it again! The Professor is also very methodic: when he encounters a puzzle he has to solve it, he is unable to just ignore it and proceed.

Luke is trying to imitate the Professor, but he is scared not to be smart enough. He wants to solve the mystery, going from the puzzle 0 to the puzzle $N - 1$ using *the least amount of smartness needed*. Solving a puzzle requires you to be as smart as its difficulty.

You can lower the difficulty of a puzzle by spending a *hint coin*, halving and rounding down its difficulty. For example a puzzle worth 15 *Picarats* with a *hint coin* is worth 7 *Picarats*, with 2 coins 3 *Picarats* and with 3 just 1 *Picarats*. Spending any more coin reduces the difficulty to zero, essentially revealing the solution!

The *smartness* one needs to solve the mystery is therefore the difficulty of the hardest puzzle solved. What is the value of this difficulty, assuming you start with C_0 coins?



Calm down, Luke.

Among the attachments of this task you may find a template file `picarats.*` with a sample incomplete implementation.

Input

The first line of the input contains 3 integers: N , M and C_0 , respectively the number of puzzles in the game, the number of connections between puzzles and the number of coins Luke has before the start of the game.

The next line contains N integers P_i , the difficulties of the puzzles in *Picarats*.

The next M lines contain 3 integers each: a , b , c . They indicate that after solving the puzzle a one can go to puzzle b collecting c hint coins in the journey. The order matters: it does not mean that one can also go from b to a .

Output

You need to write a single line with an integer: the minimum smartness Luke needs to solve the mystery.








¹https://en.wikipedia.org/wiki/Professor_Layton

Constraints

- $2 \leq N \leq 10\,000$.
- $1 \leq M \leq 50\,000$.
- $0 \leq C \leq 100$ where C is the total number of coins you can ever find in the game, including C_0 .
- $0 \leq P_i \leq 10^9$ for each $i = 0 \dots N - 1$.
- Luke cannot go arbitrarily from one puzzle to another one unless an explicit connection between them is present.

Scoring

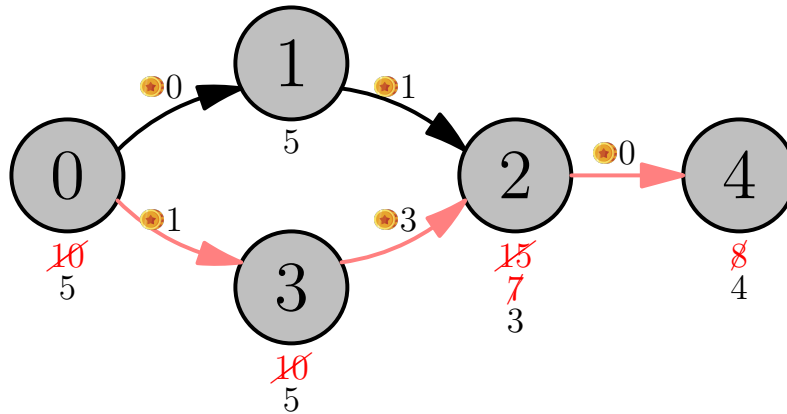
Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** (0 points) Examples.

- **Subtask 2** (10 points) At each puzzle there is at most one next puzzle and $C = 0$.

- **Subtask 3** (13 points) $N \leq 10$ and $C \leq 10$.

- **Subtask 4** (12 points) $C = 0$.

- **Subtask 5** (21 points) $C = 1$.

- **Subtask 6** (19 points) $P_i = P_j$ for all i, j .

- **Subtask 7** (25 points) No additional limitations.


Examples

input	output
5 5 2 10 5 15 10 8 0 1 0 0 3 1 1 2 1 3 2 3 2 4 0	5
5 4 1 100 51 123 40 6 0 1 0 1 2 0 2 3 0 3 4 0	100

Explanation



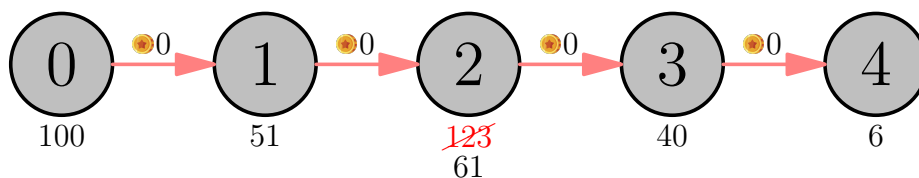
Representation of the first sample case.

In the **first sample case** the optimal solution (highlighted in red) is:

- Start at puzzle 0 with $C_0 = 2$ coins.
- Spend a coin, halving its difficulty.
- Proceed to puzzle 3 collecting 1 coin. Now Luke has 2 coins in his pocket.
- Spend a coin halving its difficulty.
- Proceed to puzzle 2 collecting 3 coins. Now Luke has 4 coins in his pocket.
- Spend 2 coins halving its difficulty twice.
- Proceed to puzzle 4 collecting 0 coins. Now Luke has 1 coin in his pocket.
- Spend 1 coin halving its difficulty.

Now Luke has solved the final puzzle, effectively solving the mystery! The hardest puzzles he solved were 0 and 3, both with the highest difficulty: 5.

Note that by going through puzzle 1 the solution is suboptimal: he can spend a coin on 0, then 2 coins on 2 but he has no coins left to spend on 4, increasing the solution to 8. If instead he spends only a coin on 2, the solution would be 7 and still suboptimal.



Representation of the second sample case.

In the **second sample case** Luke has a single path to the solution. He can spend his only coin on the puzzle 2, lowering the difficulty from 123 to 61. The hardest puzzle he has to solve is the first.

Prizes Assignment (prizes)

Marco is the coach of a team with N members, which just won N prizes in a prestigious international competition. Now starts the hardest part: assigning one of the prizes to each one of the team members!



Figure 1: The prizes that need to be assigned.

By being the coach, Marco has the duty of deciding an assignment of the prizes. For each team member $i = 0 \dots N - 1$ and prize $j = 0 \dots N - 1$, Marco knows S_{ij} , which represents how much member i would be satisfied by prize j . His goal is to assign a different prize P_i to each team member i , so that the total satisfaction is *minimal*: no easy win for his fellow comrades! However, if he assign prizes in such a way that team member i prefers prize P_j over P_i , and conversely team member j prefers P_i over P_j , the two comrades will immediately swap prizes, ruining Marco's effort to minimise his team's satisfaction. Help Marco find an assignment minimising the total satisfaction and such that no swap is possible!

📎 Among the attachments of this task you may find a template file `prizes.*` with a sample incomplete implementation.

Input

The first line contains the only integer N . The following N lines, for $i = 0 \dots N - 1$, contain N integers S_{ij} each, the satisfaction of member i by prize j .

Output







You need to write a single line with N integer P_i : the prizes assigned to each member.

Constraints

- $1 \leq N \leq 100$.
- $1 \leq S_{ij} \leq 1\,000\,000$ for each $i, j = 0 \dots N - 1$.
- S_{ij} are all distinct.

Scoring

Your program will be tested against several test cases grouped in subtasks. Your score on a subtask will be equal to the minimum score for one of its testcases multiplied by the value of the subtask. Your score on a test case will be zero if the output does not represent a stable assignment of prizes. If the output is valid assignment (not allowing swaps), let S_{out} be the sum of the satisfactions according to the assignment reported, and S_{ref} be the total satisfaction of a reference solution. Your score on a test case will be 1 if $S_{\text{out}} \leq S_{\text{ref}}$, or $(S_{\text{ref}}/S_{\text{out}})^3$ otherwise.

- **Subtask 1** (0 points) Examples.

- **Subtask 2** (10 points) $N \leq 2$.

- **Subtask 3** (20 points) $N \leq 5$.

- **Subtask 4** (30 points) $N \leq 10$.

- **Subtask 5** (25 points) $N \leq 15$.

- **Subtask 6** (15 points) No additional limitations.


Examples

input	output
2 1 5 4 3	1 0
2 1 5 3 4	0 1

Explanation

In the **first sample case**, there is only one stable assignment: the one in which every member gets its preferred prize. The opposite assignment is not valid as the two members would then swap their prizes.

In the **second sample case**, both assignments are stable. The assignment $P_0 = 0$ and $P_1 = 1$ produces a total satisfaction of $1 + 4 = 5$, and would get full score. The assignment $P_0 = 1$ and $P_1 = 0$ produces a total satisfaction of $5 + 3 = 8$, and would get a score of $5/8 = 0.625$.

Perfect Pizza (recipe)

During hard lockdown times, everyone's darkest side comes out from the mist! In Giorgio's case, that means compulsive and competitive cooking: he now wants to prepare the *perfect pizza*, and there's nothing that can stop him!



Figure 1: Giorgio working on the perfect pizza.

However, achieving such a goal is not a simple matter, and he is still stuck on deciding the perfect balance of the N ingredients (numbered with $i = 0 \dots N - 1$). He found M trusted websites, numbered with $j = 0 \dots M - 1$, each of them proposing different weights P_{ij} (in milligrams) for the various ingredients. The situation is made even more confusing by the fact that the doses on each website refer to a different total weight, and thus cannot be easily combined! To get out of the impasse, there is but one solution: convert each recipe to a total of 1 kg (10^6 mg), by multiplying each weight for an appropriate factor, and then average out all the resulting weights for each ingredient across the various recipes. Help Giorgio compute the perfect pizza recipe!

Among the attachments of this task you may find a template file `recipe.*` with a sample incomplete implementation.

Input

The first line contains the two integers N and M . The following M lines each contain the N integers P_{ij} .

Output

You need to write a single line with N integer: the averaged weights of the ingredients across recipes converted to 1 kg total.

The averaged weights might **not** be integer values. Use variables of `double` type (C/C++/Pascal) to compute intermediate results. Then, truncate the final result `r` to an integer number of milligrams through `(int)r` (C/C++) or `trunc(r)` (Pascal).

Constraints

- $1 \leq N, M \leq 1000$.
- $0 \leq P_{ij} \leq 10^6$ for each $i = 0 \dots N - 1$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** (0 points) Examples.
- **Subtask 2** (10 points) $N = 1$.
- **Subtask 3** (20 points) $M = 1$.
- **Subtask 4** (30 points) $M = 2$.
- **Subtask 5** (40 points) No additional limitations.

Examples

input	output
4 1 400000 50000 10000 40000	800000 100000 20000 80000
4 2 400000 50000 10000 40000 600000 200000 60000 140000	700000 150000 40000 110000

Explanation

In the **first sample case**, there is a single website, whose recipe adds up to 500 grams. In order to refer it to 1 kilogram, every amount needs to be doubled.

In the **second sample case** there are two websites. The first website is the one in the first sample case. The second has a different recipe, which however is already referred to 1 kilogram. By averaging the recipes referred to 1 kilogram, we obtain for the four ingredients' weights:

$$\begin{aligned} 700\,000 &= (800\,000 + 600\,000)/2 \\ 150\,000 &= (100\,000 + 200\,000)/2 \\ 40\,000 &= (20\,000 + 60\,000)/2 \\ 110\,000 &= (80\,000 + 140\,000)/2 \end{aligned}$$

Second Dose (seconddose)

The Italian government has successfully started its vaccine campaign against COVID-19 but is having some troubles after the injections of the second dose have begun.



Figure 1: For some vaccines, doses need to be firstly extracted from vials.

To protect patients' privacy, the only piece of information collected after the vaccination is an anonymous identifier stored in a database that is supposed to uniquely identify a person through the whole process.

This means that at the current point we have a list with N_1 identifiers assigned to people who have been vaccinated with the first dose, and another shorter list with N_2 identifiers assigned to those who have also been vaccinated with the second shot.

Being in a rush, some centers made some mistakes in assigning the proper identifiers when administering the second dose. These errors make it very hard to produce reliable statistics about the number of people fully vaccinated with both doses, but Luca offered himself as a consultant for the government to help. Can you tell how many people have been fully vaccinated?

Among the attachments of this task you may find a template file `seconddose.*` with a sample incomplete implementation.

Input

The first line contains two integers, N_1 and N_2 . The second line contains N_1 integers, the identifiers of people who received the first dose. The third line contains N_2 integers, the identifiers of people who received the second dose.

Output





You need to write a single line with an integer: the number of people who have been fully vaccinated, which means that their identifier appears in both lists.

Constraints

- $1 \leq N_2 \leq N_1 \leq 100\,000$.
- All identifiers are integers between 1 and 10^9 .
- In each of the two lists, considered separately, identifiers are unique (i.e., in a list an identifier is never repeated).

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** (0 points) Examples.

- **Subtask 2** (30 points) $N_1 \leq 10\,000$.

- **Subtask 3** (20 points) $N_1 \leq 30\,000$.

- **Subtask 4** (50 points) No additional limitations.


Examples

input	output
4 3 89 34 13 21 13 34 21	3
4 3 89 34 13 21 13 34 50	2

Explanation

In the **first sample case**, four people received the first dose and three people the second one. All the three identifiers (13, 34, and 21) that appear in the second list also appear in the first one, which means that we are certain that three people have been fully vaccinated.

In the **second sample case**, four people received the first dose and three people the second one. Only two identifiers (13 and 34) that appear in the second list also appear in the first one, which means that we are certain that two people have been fully vaccinated. The extra identifier in the second list, 50, is one of the errors made by the vaccination centers and will need further investigation.

Smart Thermostat (sensor)

Edoardo is getting interested in making his house *smarter*. For this reason, he has bought a smart thermostat, which measures the temperature in his room. Edoardo has gathered so far N measures V_i produced by the sensor and would like to calculate some statistics.



Figure 1: Edoardo's new smart thermostat.

However, the thermostat is not always able to produce a stable measurement: in the case of an unreliable reading, $V_i = -1$ is reported as a special value instead of the correct measure of the room's temperature.

The N measurements were all obtained at regular intervals in a single morning, starting just after switching on the heating system. For this reason, we are certain that during the whole morning the temperature of the room was non-decreasing.

Edoardo wants to fix the errors by changing all the unreliable measurements so that they respect the non-decreasing property (formally, $V_i \geq V_{i-1}$ for each $i = 1 \dots N - 1$). What is the *minimum* sum of the measurements that he can get?

📎 Among the attachments of this task you may find a template file `sensor.*` with a sample incomplete implementation.

Input

The first line contains the only integer N , the number of measurements. The second line contains N integers V_i , the room temperature's measurements.

Output







You need to write a single line with an integer: the minimum sum of V Edoardo can get, after he changes all the wrong measurements (that is when $V_i = -1$).

Constraints

- $1 \leq N \leq 10^5$.
- $0 \leq V_i \leq 10^4$ or $V_i = -1$ for each $i = 0 \dots N - 1$.
- $V_0 \neq -1$.
- It is guaranteed that Edoardo can change the unreliable measurements in such a way that $V_{i-1} \leq V_i$ for each $i = 1 \dots N - 1$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** (0 points) Examples.

- **Subtask 2** (15 points) $N \leq 6, V_i \leq 10$.

- **Subtask 3** (15 points) $V_i \neq -1$ for each $i = 0 \dots N - 1$.

- **Subtask 4** (15 points) $V_i = -1$ for each $i = 1 \dots N - 1$.

- **Subtask 5** (25 points) $N \leq 1000$.

- **Subtask 6** (30 points) No additional limitations.


Examples

input	output
5 1 -1 2 3 -1	10

Explanation

In the **first sample case**, the minimum sum of V is obtained if Edoardo fixes the errors by setting $V_1 = 1$ and $V_4 = 3$; in this case $V = [1, 1, 2, 3, 3]$.

Team Representatives (team)

William, after playing a lot of soccer games on his smartphone, wanted to get involved in an actual soccer team; right now he is helping to manage his local one. There are N players the team would like to buy. For this reason, for the i -th player a meeting has been scheduled on day D_i .



Figure 1: A representative of the soccer team.

Of course, at every meeting there must be at least one representative of the team to negotiate the terms of the contracts. However, the team manager would like to minimize the number of representatives needed. Since the players might live in different cities, it takes C_{ij} days to go from the city of the i -th player to the city of the j -th one. This means that a team representative can attend the j -th player's meeting after attending the meeting for the i -th player only if $D_i + C_{ij} \leq D_j$. What is the minimum number of representatives needed in order to be able to attend all the meetings?

Among the attachments of this task you may find a template file `team.*` with a sample incomplete implementation.

Input

The first line contains the only integer N , the number of players. The second line contains N integers D_i , the day the meeting for the i -th player has been scheduled on. Then N lines follow; the i -th line contains N integers C_{ij} , the time it takes to go from the city of the i -th player to the one of the j -th player.

Output

You need to write a single line with an integer: the minimum number of representatives needed.







Constraints

- $1 \leq N \leq 500$.

- $0 \leq D_i \leq 10^6$ for each $i = 0 \dots N - 1$.
- $1 \leq C_{ij} \leq 10^6$ for each $i \neq j, i = 0 \dots N - 1, j = 0 \dots N - 1$.
- $C_{ii} = 0$ for each $i = 0 \dots N - 1$.
- C_{ij} is not guaranteed to be equal to C_{ji} for any $i \neq j$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- **Subtask 1** (0 points) Examples.

- **Subtask 2** (15 points) $N \leq 10$.

- **Subtask 3** (20 points) $N \leq 17$.

- **Subtask 4** (25 points) $C_{ij} = 1$ for each $i \neq j$.

- **Subtask 5** (20 points) $N \leq 100$.

- **Subtask 6** (20 points) No additional limitations.


Examples

input	output
3 1 2 3 0 1 3 2 0 3 3 3 0	2
3 2 1 3 0 1 1 1 0 2 2 2 0	1

Explanation

In the **first sample case**, the minimum number of representatives needed is two. In particular, the first representative can attend the meeting of the first player on day 1, then travel for one day to the meeting of the second player on day 2. The second representative, instead, will attend the meeting of the third player on day 3.

In the **second sample case**, only one representative is needed. He can attend the meeting of the second player on day 1, then travel for one day to the meeting of the first player on day 2, finally travel for one day and then attend the meeting of the third player on day 3.

Trending Topics (trending)

Luca has been hired as an intern at Twitter to improve tools for understanding virality on social media. The company is known for having popularized the now widespread *hashtags*, constituted by the hash symbol # followed by a keyword, used to tag contents (especially tweets, in the case of Twitter).




Figure 1: The tweet that made history proposing the use of hashtags in 2007!

Each second, internal Twitter functions report the most used hashtag worldwide in that second (the most used hashtags signal the so-called *trending topics*).

However, this level of granularity can be difficult to comprehend: for this reason, Luca wants to create a tool that aggregates data and reports the most popular hashtag for each given period of time lasting T seconds. For example, when $T = 60$ seconds, the tool should output the most used hashtag in the first minute (seconds 0 to 59, inclusive), then the most used one in the minute that lasts from second 1 to second 60 (inclusive), and so on.

Given the list of the most popular hashtags in each of the last N seconds, are you able to produce such a report?

 Among the attachments of this task you may find a template file `trending.*` with a sample incomplete implementation.

Input

The first line contains two integers: N and T . The i -th of the following N lines contains the most popular hashtag in the i -th second.

Output







You need to write a line containing the most popular hashtag *for each period of time that lasts T seconds* (i.e., the first line will contain the most popular hashtag from second 0 to $T - 1$, the second line the most popular hashtag from second 1 to T , and so on).

Constraints

- $1 \leq T \leq N \leq 50\,000$.
- Each hashtag consists of up to 20 lowercase letters of the English alphabet.
- In case of ties, the lexicographically smallest hashtag is chosen.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

- Subtask 1 (0 points) Examples.

- Subtask 2 (6 points) $N \leq 1000, T = 1$.

- Subtask 3 (24 points) $N \leq 1000, T \leq 10$.

- Subtask 4 (29 points) $N \leq 5000$.

- Subtask 5 (27 points) $N \leq 20\,000$.

- Subtask 6 (14 points) No additional limitations.


Examples

input	output
5 3 ois final ranking ois ois	final final ois
6 4 potus biden trump trump biden capitolhill	trump biden trump

Explanation

In the **first sample case**, we have to consider periods of time of $T = 3$ seconds. In the first of such periods, three different hashtags are observed and therefore the most popular is **final** (due to lexicographical order). In the second period of time, hashtags **final**, **ranking** and **ois** are considered: it is again a tie and **final** is chosen. In the third period of time, **ois** wins because it occurs twice.

In the **second sample case**, we have to consider periods of time of $T = 4$ seconds. In the first of such periods, **trump** occurs twice and is the most popular hashtag. In the second one, **biden** and **trump** both occur twice, but **biden** is preferred due to lexicographical order. In the third one, **trump** occurs twice and is again the most popular hashtag.