

## Arena Freaks (arena)

William is a long-time fanatic of *OpenArena*, an open source FPS (First-Person Shooter) game. For this reason he decided to start a brand-new web portal gathering OpenArena addicts from all over the world: [arenafreaks.net](http://arenafreaks.net). This website addresses multiple different needs of a professional OA player through forums, dedicated social networks, tutorials, rankings, and much more. The feature which William is most proud of, however, is the game database: a huge collection of the most spectacular games ever played in OpenArena, transcribed in *arenaic notation* (an alphanumeric notation subtly inspired by the algebraic notation of chess games).



Figure 1: A typical OpenArena “capture the flag” match.

In this notation, a game between  $N$  players (numbered from 0 to  $N - 1$ ) each starting with  $L$  lives is transcribed as a series of  $E$  events which can be (among many others) of the following types:

- *frags* (denoted as ‘ $P \text{ f } Q$ ’), where a player  $P$  “kills” a player  $Q$  thus reducing the number of  $Q$ ’s lives by 1;
- *explosions* (denoted as ‘ $P \text{ e}$ ’), where a player  $P$  frags simultaneously all the other players, thus reducing the number of their lives by 1.

Help William count the number of players still alive (with some lives left) at the end of the game!

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

## Input

The input file contains the full transcription of a game in arenaic notation. The first line contains the three integers  $N$ ,  $E$ ,  $L$ . Other  $E$  line follow, each containing the transcription of an event (as described in the task description).

## Output







You need to write a single line with an integer: the number of players with some lives left at the end of the game.

## Constraints

- $1 \leq N, E, L \leq 100\,000$ .
- $0 \leq P, Q \leq N - 1$  in each event.
- All events are valid, that is, players  $P$  and  $Q$  in both frags and explosions cannot have already lost all their lives.

## 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 (20 points)       $N \leq 5$  and there are no explosions.  

- Subtask 3 (20 points)      There are no explosions.  

- Subtask 4 (20 points)      There are no frags.  

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

- Subtask 6 (20 points)      No additional limitations.  


## Examples

input.txt	output.txt
4 4 2 0 f 3 1 e 2 f 1 0 f 2	2
6 7 4 5 e 5 f 1 1 f 2 0 f 2 3 f 2 5 e 3 e	4

## Explanation

In the **first sample case**, the number of player's lives changes as follows:

Player	0	1	2	3
	2	2	2	2
0 f 3	2	2	2	1
1 e	1	2	1	×
2 f 1	1	1	1	
0 f 2	1	1	×	

In the **second sample case**, the number of player's lives changes as follows:

Player	0	1	2	3	4	5
	4	4	4	4	4	4
5 e	3	3	3	3	3	4
5 f 1	3	2	3	3	3	4
1 f 2	3	2	2	3	3	4
0 f 2	3	2	1	3	3	4
3 f 2	3	2	×	3	3	4
5 e	2	1		2	2	4
3 e	1	×		2	1	3

## Irreversible fight (breakup)

After years of idyllic cooperation, Giorgio and William had a big fight on how to organize the final round of the OIS and eventually broke up, leaving the whole Italian Informatics Olympiads (OII) team shaken to the core.

The OII team consists of  $N$  people, where 0 is William and  $N - 1$  is Giorgio. Among them, there are  $M$  directed friendship bonds, so that person  $A_i$  values friendship with  $B_i$  as  $V_i$  worth. As usual with breakups, everyone now needs to choose on which side they want to be: should they blame William or Giorgio?




Of course, this choice is never done based on the actual facts happened. Instead, everyone chooses the side to which he is

*more bonded*. More precisely, if  $S$  is a set of people and  $x$  is a person, the *total bond*  $\text{bond}_S(x)$  of  $x$  to  $S$  is equal to the sum of values  $V_i$  of bonds between  $x$  and his friends in  $S$ . Every person chooses to support Giorgio or William trying to maximise the total bond with other people supporting the same person. After some transitory initial period, this will inevitably lead to a *stable splitting*  $\langle G, W \rangle$ . In such splitting, we will have that  $0 \in W$ ,  $N - 1 \in G$  and:

- For each  $x \in W$  apart from 0,  $\text{bond}_W(x) \geq \text{bond}_G(x)$  (people in  $W$  do not want so switch sides);
- For each  $x \in G$  apart from  $N - 1$ ,  $\text{bond}_G(x) \geq \text{bond}_W(x)$  (people in  $G$  do not want so switch sides).

William knows these mechanics very well, and wants to exploit them to keep as many friends as possible, while leaving Giorgio to a grim isolation. Help William find the stable splitting  $\langle G, W \rangle$  with the largest value for  $\text{bond}_W(0) - \text{bond}_G(N - 1)$ !

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

### Input

The first line contains integers  $N$ ,  $M$ . The following  $M$  lines contain integers  $A_i$ ,  $B_i$ ,  $V_i$ .

### Output

You need to write a string of  $N$  characters 'W' or 'G' depending on who the  $i$ -th person supports.






### Constraints

- $2 \leq N \leq 100\,000$ .
- $2N - 4 \leq M \leq 200\,000$ .
- $0 \leq A_i, B_i \leq N - 1$  and  $A_i \neq B_i$  for each  $i = 0 \dots M - 1$ .
- $1 \leq V_i \leq 1000$  for each  $i = 0 \dots M - 1$ .

- There are no repeated friendship bonds, but the bond of  $x$  to  $y$  is possibly different from the bond of  $y$  to  $x$ . (love is not always paid!)
- Both Giorgio and William are bonded to all other people  $1 \dots N - 2$ .
- If there are multiple solutions, you can output any of them.

## 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)      Each person  $1 \dots N - 2$  is bonded to a single other one.  

- **Subtask 3** (35 points)       $N \leq 10$ .  

- **Subtask 4** (25 points)       $N \leq 1000$ .  

- **Subtask 5** (10 points)      No additional limitations.  


## Examples

input.txt	output.txt
4 8 0 1 13 0 2 42 3 2 25 3 1 30 1 0 10 2 0 14 1 2 20 2 3 15	WGGG
4 8 0 1 13 0 2 42 3 2 25 3 1 30 1 0 10 2 0 14 1 2 20 2 3 14	WWWG

## Explanation

In the **first sample case**, the splitting is stable:  $_G(2) = 15 \geq 14 =_W(2)$ ,  $_G(1) = 20 \geq 10 =_W(1)$ . Even though  $_W(0) -_G(N - 1) = -55$ , there is no better splitting for William.

In the **second sample case**, the splitting is stable since  $_W(2) = 14 \geq 14 =_G(2)$  and  $_W(1) = 20 \geq 0 =_G(1)$ , scoring an optimal  $_W(0) -_G(N - 1) = 55$ .

## Olympic cake (cake)

The preparations for the big final competition of this year's Olympiads have already begun!


A big *rectangular* cake will be served at the end of the meal, to celebrate the end of the 2018 edition. Many guests are expected to take part in the festivities and, obviously, everyone wants to taste at least a bit of the special cake.



Figure 1: A rectangular cake baked for the 2014 final of the Italian Olympiad.

The baker wants to please as many people as possible, but time is running out: he can only do *at most*  $N$  cuts. Each cut is made either vertically or horizontally, for the whole length of the cake.

How many pieces of cake can the baker obtain at most?

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

### Input

The first and only line of the input contains a single integer:  $N$ .

### Output

You need to write a single line with one integer: the number of pieces that can be obtained at most, making no more than  $N$  cuts.

### Constraints

- $0 \leq N \leq 1\,000\,000\,000$ .

### 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)       $N \leq 1\,000$ .



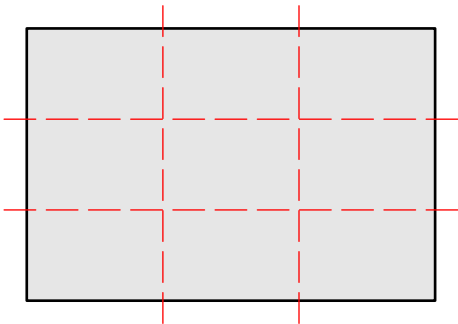
- Subtask 3 (20 points)  $N \leq 50\,000$ .  
🍰🍰🍰🍰
- Subtask 4 (20 points)  $N \leq 1\,000\,000$ .  
🍰🍰🍰🍰
- Subtask 5 (30 points) No additional limitations.  
🍰🍰🍰🍰

Examples

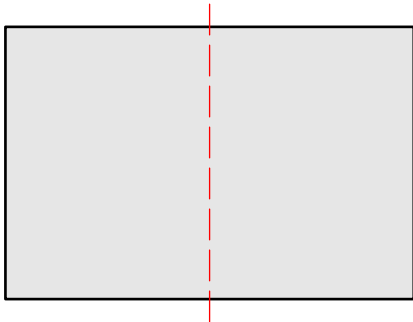
stdin/input.txt	stdout/output.txt
4	9
1	2

Explanation

In the **first example**, an optimal solution is to make two vertical cuts and two horizontal cuts. This splits the cake in nine pieces, as shown in the following picture.



In the **second example**, we can obtain two pieces making a single vertical cut. No other allowed cutting strategy leads to more pieces.





## Algorithmic excursion (excursion)

Giorgio is a passionate hiker, and every weekend goes for an excursion in his beloved mountains: *the Alps*.

Today, he is going to explore the *Val Troncea*, a nice rectangular valley consisting of  $H \times W$  square meters of land, each with a different altitude of  $A[i, j]$  millimeters above sea level (for  $i = 0 \dots H - 1, j = 0 \dots W - 1$ ).

He will start from one of its corners, labelled  $(0, 0)$ , then he will wander around without following maps nor trails: he will follow an *algorithm*!

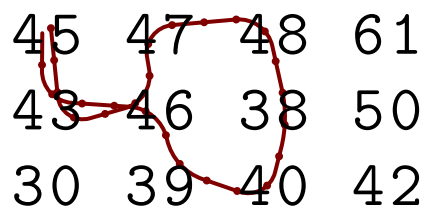
More specifically, Giorgio follows six simple rules:



Figure 1: A typical Giorgio's hike.

1. **No jumps:** step only between adjacent squares (in directions North, South, East or West).
2. **No flight:** never get out of the borders of the valley.
3. **No return:** never come back to the square from which you just arrived.
4. **No cliffs:** always choose the square with an altitude closer to the current one (least absolute difference of altitudes).
5. **No effort:** if you have to choose between two squares with the same altitude difference, then choose the one which goes downwards (with lower altitude).
6. **No repeats:** if you end up in a square where you have already been, then come back home (using the same path you used to get there on the first place).

For example, when Giorgio practised the hike in his  $3m \times 4m$  back yard, he followed this path:



Starting from the 45, Giorgio could move to squares 43 or 47 by rules 1 and 2. Since rule 4 does not decide among them, rule 5 comes into play selecting the 43. Then, Giorgio could move either to squares 30 or 46 (by rules 1, 2, 3) and rule 4 selects the 46. The walk proceeds similarly with squares 47, 48, 38, 40 and 39; until it reaches again square 46. Following rule 6, Giorgio has to come back through the same path used before (squares 43 and 45), instead of looping through the cycle he just closed.

The next round of the IOIT is approaching, and Luca, Edoardo and William are worried that Giorgio might not come back in time! Help them calculate how long the hike will take, given the *Val Troncea* altitudes map and knowing that Giorgio takes exactly one step per second.



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

## Input

The first line contains the two integers  $H, W$ . The following  $H$  lines for  $i = 0 \dots H - 1$  contain  $W$  integers each: altitudes  $A[i, j]$  for  $j = 0 \dots W - 1$ .

## Output






You need to write a single line with an integer: the number of steps (or, equivalently, seconds) that Giorgio's hike will take.

## Constraints

- $2 \leq H, W \leq 2000$ .
- $0 \leq A[i, j] \leq 4\,800\,000$  for each  $i = 0 \dots H - 1, j = 0 \dots W - 1$ .
- The altitudes are all distinct:  $A[i, j] \neq A[k, h]$  if  $(i, j) \neq (k, h)$ .

## 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** (20 points)      Giorgio's hike encompasses altitudes  $0, 1, \dots, k$  in order.  

- **Subtask 3** (30 points)       $H, W \leq 10$ .  

- **Subtask 4** (30 points)       $H, W \leq 100$ .  

- **Subtask 5** (20 points)      No additional limitations.  


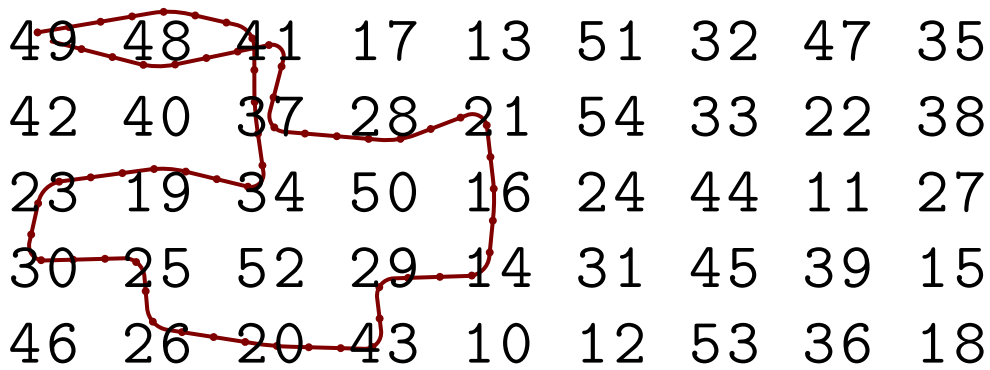
## Examples

input.txt	output.txt
3 4 45 47 48 61 43 46 38 50 30 39 40 42	10
5 9 49 48 41 17 13 51 32 47 35 42 40 37 28 21 54 33 22 38 23 19 34 50 16 24 44 11 27 30 25 52 29 14 31 45 39 15 46 26 20 43 10 12 53 36 18	20

Explanation

The first sample case is discussed in the task statement.

In the second sample case, the path followed by Giorgio is the following:



## Enrichment Center (glados)

After playing *Portal*, Giorgio decided to open his own *Enrichment Center* and be like GLaDOS in real life. In this kind of research facility, test subjects are locked into chambers and faced with several dangerous puzzles, which need to be solved to gain access to the exit (and to another even riskier chamber). Obviously, there are annoying laws about human rights that would interfere with a faithful implementation of this concept; but Giorgio already planned to get around that by building a very small (and cheap!) Enrichment Center designed for lab rats.

Of course, lab rats are not exceptionally smart so the puzzles need to be fairly straightforward. In the first test chamber, Giorgio is planning to arrange into a grid  $H \times W$  some blocks of the following types:

- plain floor, represented by ‘.’;
- walls, represented by ‘#’;
- exits, represented by ‘0’;
- transparent glue, represented by ‘@’.




Giorgio put the first lab rat in row-column coordinates  $(R; C)$  into the grid, facing north. Quite surprisingly, he noticed that the rat is using one of the most ancient techniques for getting out of a maze: *stick to your right hand*.

Figure 1: A prototypical chamber in the *Aperture Science™* Enrichment Center.

In other words, the rat is following the wall that was on his right at the beginning, as if always sticking one hand on it.

Giorgio can't wait to know whether the rat will end up stuck in the glue, or if it will keep cycling forever or eventually reach an exit. Satisfy Giorgio's curiosity by calculating the rat's fate!

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

## Input

The first line contains the four integers  $H$ ,  $W$ ,  $R$ ,  $C$ . Other  $H$  lines follow, each containing a string consisting of  $W$  characters among ‘.’, ‘#’, ‘0’, ‘@’.

## Output

You need to write a single line with a single word:




- **stuck** if the rat will first reach a block with the glue,
- **free** if the rat will first reach a block with an exit,
- **cycling** if the rat will keep cycling forever.

## Constraints

- $3 \leq H, W \leq 100$ .
- $(R; C)$  vary from  $(0; 0)$  — top left to  $(H - 1, W - 1)$  — bottom right.
- All blocks on the perimeter are walls (so the rat cannot start on a border block).
- There is a wall on the right of the rat at the beginning (i.e., in position  $(R, C + 1)$ ).

## 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.

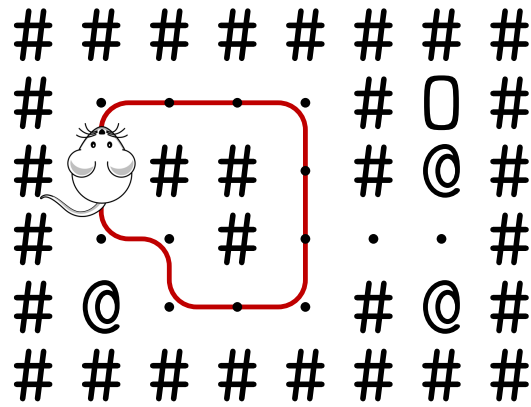
- **Subtask 1** ( 0 points)      Examples.  

- **Subtask 2** (50 points)      The answer is not **cycling**.  

- **Subtask 3** (50 points)      No additional limitations.  


## Examples

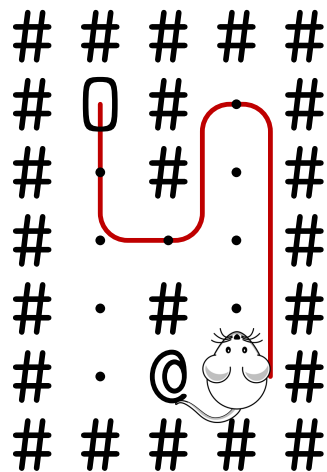
input.txt	output.txt
6 8 2 1 ##### #...#0# #...#@# #...#...# #@...#@# #####	cycling
7 5 5 3 ##### #0#.# #...# #...# #...# #...# #...# #####	free
6 8 2 1 ##### #...@.#0# #...#@# #...#...# #@...#@# #####	stuck

## Explanation

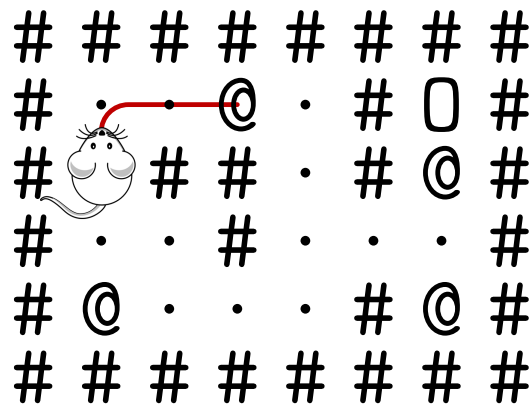
In the **first sample case**, the path followed by the rat is the following:



In the **second sample case**, the path followed by the rat is the following:



In the **third sample case**, the path followed by the rat is the following:



## Magnetic alignment (magneti)


Despite his age, Gabriele loves playing with magnets. One of his favourite constructions is the so-called “snake of magnets”, which is a long line of magnets. Obviously, magnets can stay one next to the other only when the positive terminal of one magnet matches with the negative terminal of the next magnet, and vice versa. For instance, the following sequences of magnets are valid snakes of magnets:

$$(+-)(+-)(+-)(+-)(+-) \quad \text{and} \quad (-+)(-+)(-+),$$

while the following ones are not:

$$(+-)(-+)(-+)(-+) \quad \text{and} \quad (-+)(+-)(-+).$$

Given a generic (not necessarily stable) sequence of magnets, Gabriele wonders which is the minimum number of magnets that must be rotated in order to get a snake of magnets.

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

### Input

The first line contains the only integer  $N$ . The second line contains the string `description`, which consists of  $N$  characters.

### Output





You need to write a single integer, the minimum number of magnets that you need to rotate in order to get a snake of magnets.

### Constraints

- $4 \leq N \leq 100\,000$ .
- The string `description` consists only of characters `(,+,-,)` and is well formed. This, among other things, implies that  $N$  is a multiple of 4.
- If the initial sequence of magnets is already stable and there is no need to rotate any magnet, the answer is 0.

## 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)       $N \leq 20$ .  
    
- **Subtask 3** (40 points)       $N \leq 1000$ .  
    
- **Subtask 4** (30 points)      No additional limitations.  
    

## Examples

input.txt	output.txt
8 (+-) (-+)	1
12 (+-) (++) (-+)	1

## Explanation

In the **first sample case**, it is enough to rotate the first magnet.  
In the **second sample case**, it is enough to rotate the last magnet.



## Friendly Note (neighborly)

Italian people have a very peculiar way to permanently settle a dispute: they prepare a *friendly note*, politely describing their personal point of view about the matter, and deliver it secretly to each other. In order to add a dash of fun, they usually prepare the note by cutting and pasting sequences of characters from newspapers.

Giorgio makes no exception, and his long-standing dispute with his neighbour (about noise complaints) requires him to finally take action and defend his honour. He thus collected *a lot* of copies of his favourite newspaper, *La Stampa*, and is now ready to prepare his friendly note *N*.

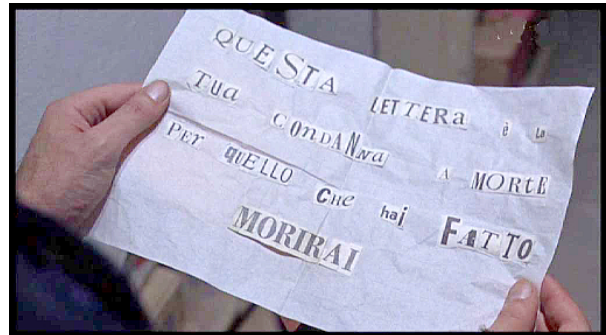


Figure 1: A typical Italian friendly note.

Since Giorgio feels uncomfortable with scissors, he would prefer to cut and paste the smallest number of character sequences from the newspaper content *S*. Help Giorgio produce text *N* by juxtaposing **as few** contiguous subsequences of text *S* as possible, assuming that these subsequences can freely overlap (remember: he collected *a lot* of copies of that newspaper).

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

### Input

The first line contains string *N*. The second line contains string *S*.

### Output

You need to write a single line with an integer: the least number of subsequences Giorgio needs to cut and paste.

### Constraints







- Strings *N* and *S* consist of ASCII printable characters only.
- $1 \leq \text{len}(N) \leq 1\,000\,000$ .
- $1 \leq \text{len}(S) \leq 10\,000$ .
- It is always possible to obtain the string *N* from substrings of *S*.

# Scoring

Your program will be tested against several test cases grouped in subtasks. The score in each subtask will be calculated as the **minimum** score obtained in any of its test cases, multiplied by the value of the subtask. The score in a test case will be **0** if you output a number that is lower than the optimal solution or higher than the length of  $N$ . Otherwise, it will be calculated as:

$$\frac{\Delta_{\max} - \Delta_{\text{out}}}{\Delta_{\max} + 10\Delta_{\text{out}}}$$

where  $\Delta_{\text{out}}$  is the difference between your solution and the optimal solution, and  $\Delta_{\max}$  is the difference between the length of  $N$  and the optimal solution.

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

- **Subtask 2** (15 points)       $N$  consists of a single character repeated multiple times.  

- **Subtask 3** (20 points)       $\text{len}(N), \text{len}(S) \leq 20$ .  

- **Subtask 4** (25 points)       $\text{len}(N) \leq 1000$ .  

- **Subtask 5** (25 points)       $\text{len}(S) \leq 1000$ .  

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


# Examples

input.txt	output.txt
@dd10 v1cin* 0 v@d1ci*n	6
caro vicino vuoi smetterla di fare casino Il caro vita sale ancora: se sei mancino vuoi smettere di comprare oggetti appositi, per esempio. Fare cassa e' sempre piu' difficile; parola di fiscalista.	5

# Explanation

In the **first sample case**, the friendly note can be composed by the following fragments: "@d", "d1", "0 v", "1ci", "n", "\*".

In the **second sample case**, the friendly note can be composed by the following fragments: "caro vi", "cino vuoi smetter", "la di f", "are cas", "ino".

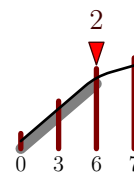
## Rollercoaster Typhoon (rollercoaster)

William is a huge fan of the *Rollercoaster Typhoon* videogame, and spends a great deal of his time designing rollercoaster tracks. However, William's wild imagination often clashes with the laws of physics that such a track has to comply with.


A rollercoaster track is defined by a sequence  $H_i$  of  $N$  integer values, representing the height of the pillars upon which the track is laid, in the order in which they are connected by the tracks. If a contiguous subsequence of at least three of these values is in arithmetic progression (that is, the difference between consecutive values is positive and constant), the corresponding section of the track is automatically *motorized*. Whenever William tests his design, a test car drives through it from  $i = 0$  on, following these rules:

1. Whenever the car reaches the start of a motorized section, it is hooked by the lifter which drops it at the last pillar of the section, from which the car starts a *free fall* as in rule 2.
2. If the car starts a free fall, it continues falling as long as the heights passed are *strictly decreasing*: at the last strictly decreasing pillar reached, the car starts an *inertial ascent* as in rule 3.
3. If the car starts an inertial ascent, it continues rising as long as the height passed are *weakly increasing* (increasing or constant) and *lower than the starting height of the preceding free fall*. If an inertial ascent ends because of a decreasing height, the car starts a free fall as in rule 2. Otherwise, the car loses control and crashes.

The test car starts in free fall from the beginning of the track, and rule 1 has the priority so that both free falls and inertial ascents are interrupted by motorized sections. Notice that free falls and inertial ascents may consist in only one pillar: in the example on the right, at the end of the motorized section, the car starts a 1-pillar free fall, which becomes a 1-pillar inertial ascent, resulting in a crash.



Help William design his rollercoaster track, by computing the last pillar that the test car is able to reach without a crash!

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

### Input

The first line contains integer  $N$ . The second line contains integers  $H_i$ .

### Output





You need to write a single integer: the last pillar reachable by the car without a crash.

### Constraints

- $1 \leq N \leq 100\,000$ .
- $1 \leq H_i \leq 100\,000$  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.

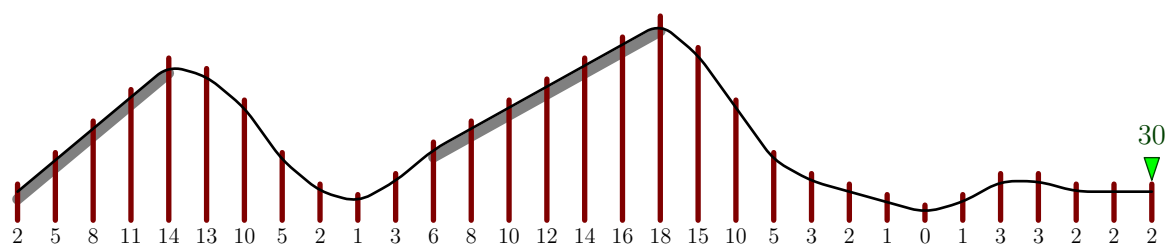
- Subtask 1 ( 0 points)      Examples.  
    
- Subtask 2 (25 points)       $N \leq 100$ .  
    
- Subtask 3 (40 points)      There are no motorized sections.  
    
- Subtask 4 (35 points)      No additional limitations.  
    

Examples

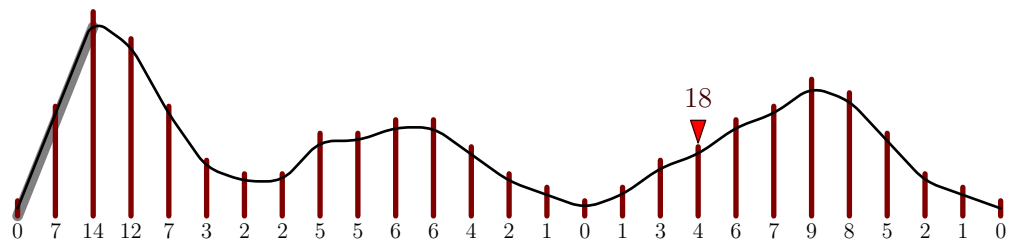
input.txt	output.txt
4 0 3 6 7	2
31 2 5 8 11 14 13 10 5 2 1 3 6 8 10 12 14 16 18 15 10 5 3 2 1 0 1 3 3 2 2 2	30
27 0 7 14 12 7 3 2 2 5 5 6 6 4 2 1 0 1 3 4 6 7 9 8 5 2 1 0	18
29 32 25 15 12 10 10 8 6 3 2 1 0 2 3 3 3 4 6 7 7 9 10 5 1 0 3 6 9 8	20

Explanation

The first sample case is described in the task statement.  
In the second sample case, the car is able to proceed until the end of the track without crashes.



In the **third sample case**, the car is first lifted through a motorized section, then performs a free fall with inertial ascent, followed by a second free fall and inertial ascent ending in a crash.



In the **fourth sample case**, the car performs a free fall followed by an inertial ascent (consisting in 2 pillars and flat), then a second free fall and inertial ascent ending in a crash.

