Composable Columns (`columns`)

By selling his marvellous mega-mansion, Edoardo was able to cover the OIS deficit and even save some money to build a more humble replacement. However, the only affordable way left is to use some of the N discounted prefabricated columns of length L_i available from exhausting stocks. These columns may be used for their exact length L_i, or may be combined in **pairs** (i, j) to produce a single column of length $L_i + L_j$. *Combining more than two columns together is never allowed for safety reasons.*

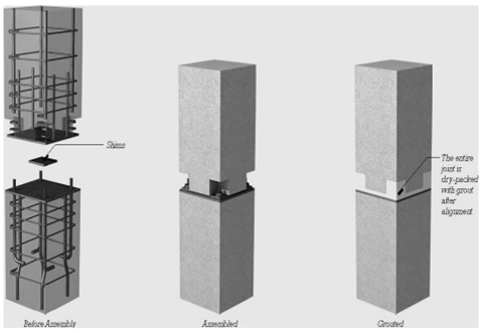


Figure 1: Instructions for joining two (and no more) prefabricated columns.

The new OIS building will require M columns each of the same height H (produced from one or two of the available prefabricated columns). This height has to to be at least T in order to fit all the workers and equipments, and of course, the smaller the building, the cheaper the final price. Help Edoardo determine the minimal height H for which a working building can be made!

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

Input

The first line contains integers N, M, T. The second line contains N integers L_i.

Output

You need to write a single line with an integer: the minimal $H \geq T$ such that M columns of length H are constructible from the N available. If there is no such H, you should output the string '`IMPOSSIBLE`'.

Constraints

- $1 \leq M \leq N \leq 1\,000\,000$.
- $1 \leq T \leq 10\,000$.
- $1 \leq L_i \leq 10\,000$ for each $i = 0 \ldots N - 1$.

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) $L_i = L_j$ for each i, j.

– **Subtask 3** (10 points) $M = 1$.

- **Subtask 4** (30 points) $N, T \leq 500$.
- **Subtask 5** (25 points) $N \leq 10\,000$.
- **Subtask 6** (20 points) No additional limitations.

Examples

input	output
`8 3 100`	`IMPOSSIBLE`
`100 30 50 40 50 80 40 30`	
`8 3 50`	`80`
`100 30 50 40 50 80 40 30`	

Explanation

In the **first sample case**, only two columns of height 100 can be obtained (a single 100 and $50 + 50$, as triplets like $40 + 30 + 30$ are not allowed). Only one column of height 110 is constructible ($80 + 30$), one of height 120 ($80 + 40$), two of height 130 ($100 + 30$ and $80 + 50$), and one each of the heights 140, 150, 180 (in all three cases involving the 100 column). Thus, it is impossible to obtain three columns of the same height above 100.

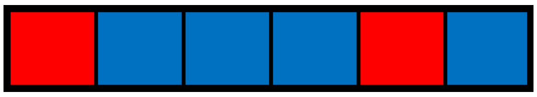
In the **second sample case**, only two columns of height 50 are readily available, and combining does not help. Only one column of height 60 is constructible ($30 + 30$), two of height 70 (both $30 + 40$), and four of height 80 (one 80, two $50 + 30$ and a $40 + 40$). Thus, the required height is 80.

Exatlon Battle (exatlon)

Like every year, Luca is organizing the famous Exatlon Battle: a test of strength, skill and logic. As usual, the competitors will be split in two teams, *The Famous* (the red team) and *The Warriors* (the blue team).

Luca's favorite test in the competition is as follows: each team receives a square formed by N boards of equal shape, but of different color. Each board has length N and width 1. The boards are formed by square cells of sizes 1×1, each of them being painted red or blue.

For example, the following figure shows a board of length 6, where the first and the fourth zones are painted red, while the zones 2, 3, 4 and 6 are blue:



The players can reorder the boards, but they cannot change the colors in each individual board. The goal of each player is to produce a rectangular zone of the maximum possible area, painted with the color of their team. The team which forms the largest rectangular area of their own color will win the match.

Before starting the live stream of the match, Luca wants to determine the winning team and the area of the largest rectangular zone, painted in the color of the winning team. Help him!

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

Input

The first line of the input contains a positive integer N, the number of painted boards in the game. Each of the following N lines contains N characters (either `A` or `R`, meaning respectively a *blue* or a *red* cell), representing the way in which the boards are painted.

Output

You should write two integer numbers separated by a space. The first number represents the winning team (1 for *The Famous*, 2 for *The Warriors*, 0 if it's a draw) and the second number represents the area of the maximum rectangular zone that can be produced.

Constraints

- $2 \leq N \leq 1000$.

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 board is *monochromatic*: either all-blue or all-red.

– **Subtask 3** (30 points) On each board all the blue squares precede the red ones.

– **Subtask 4** (40 points) $N \leq 100$.

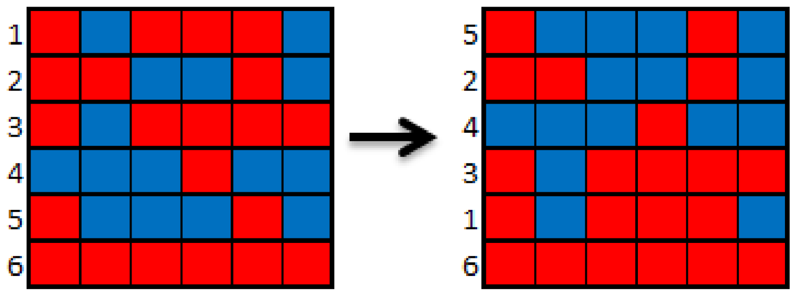
– **Subtask 5** (20 points) No additional limitations.

Examples

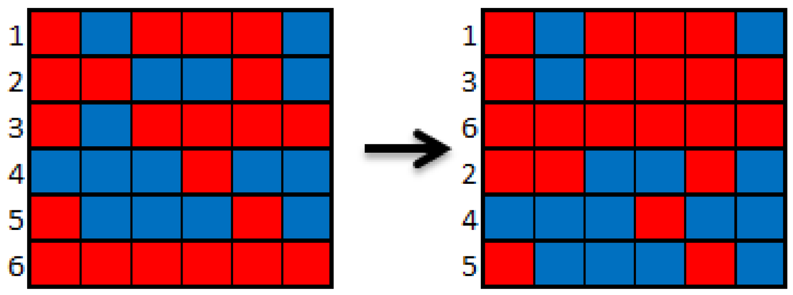
input	output
6 RARRRA RRAARA RARRRR AAARAA RAAARA RRRRRR	1 9
2 RA AR	0 1

Explanation

In the **first sample case**, The Famous (the red team) can arrange the board in the order 5, 2, 4, 3, 1, 6 and obtain a rectangular zone of area 9, as shown here:



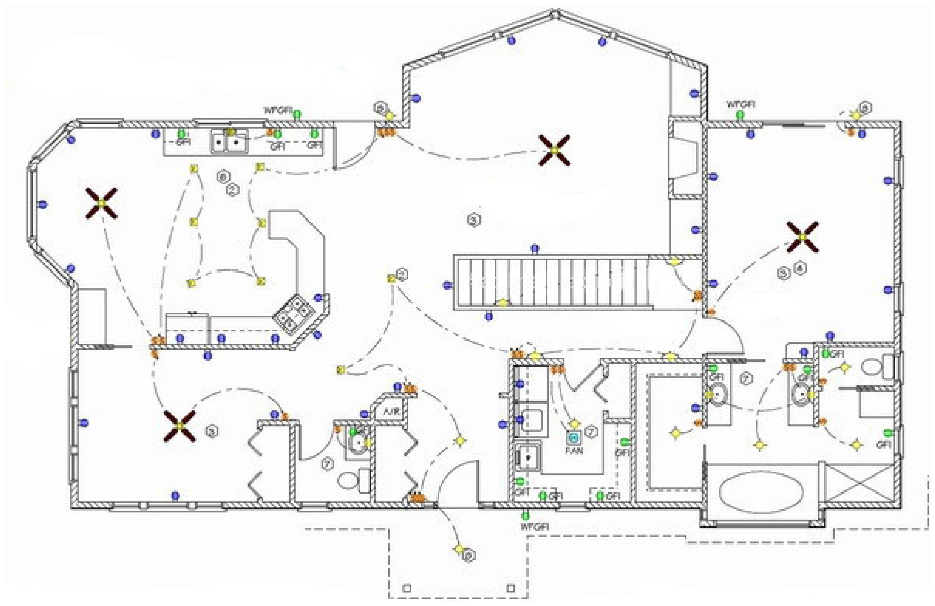
The Warriors (the blue team) can arrange the boards in the order 1, 3, 6, 2, 4, 5 and obtain a rectangular zone of area 4, thus losing the match:



In the **second sample case** each team can at most produce a rectangle of area 1. It's a draw.

Crazy Lights Hotel (joc)

Giorgio loves reading before going to sleep, but today he's not at home but in a cheap hotel abroad for a conference. In the *Crazy Lights* hotel, each room has N lights and the corresponding N buttons; if you press the i-th button and the i-th light was off, it will immediately turn on!



Now Giorgio is in the bed and he wants to start reading: he needs to switch off all the lights but the K-th. There is an odd behaviour though: he noticed that by using the i-th button not only the i-th light turns on but also some other lights turn off! He spent a few minutes to map all the switches to the lights and now he knows exactly what each button does. The management told Giorgio to **never use the i-th button if the i-th light is already on**, otherwise there will be a peak of power and all the hotel will go black!

Help Giorgio, who wants to start reading as soon as possible: which is the *minimum* number of buttons to press in order to switch off all the lights but the K-th one?

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

Input

The first line contains two integers: N and K, respectively the number of lights and the index of the light that needs to be on (1-based).

The next line contains N values, either 0 or 1. The i-th represents the initial state of the i-th light (0 means off and 1 means on).

The following N lines describe what each button does. The i-th line, relative to the i-th button, is composed as follows: an integer t followed by t integers, the indexes (1-based) of the lights that will turn off by pressing this button.

Output

You need to write a single line with an integer: the minimum number of buttons to press in order to have all the lights turned off but the K-th one.

Constraints

- $2 < N < 20$.
- $1 \leq K \leq N$.
- It's always possible to reach the final configuration.
- Pressing the i-th button will turn off all the corresponding lights, no matter if they were on or off.
- Only turned off lights can be turned on.
- The i-th button will not turn off the i-th light.

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) Every button turns off every other light.

– **Subtask 3** (10 points) Every light is turned off by exactly one button.

– **Subtask 4** (35 points) There is a single light on.

– **Subtask 5** (25 points) $N = 3$.

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

Examples

input	output

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

`6 3`
`0 0 1 1 0 1`
`2 2 3`
`0`
`3 2 5 6`
`1 1`
`3 1 4 6`
`0`
	`3`

Explanation

In the **first sample case** there are 3 lights and only the last must be on in the end. To do so the best strategy is to:

- Press the first button, this will turn off the second and the third lights, resulting in `1 0 0`.
- Press the last button, this will turn off the first light (and the second one, but it's already off), resulting in `0 0 1`.

In the **second sample case** there are 6 lights, the best strategy is to press:

- 1-st button, leading to `1 0 0 1 0 1`.
- 5-th button, leading to `0 0 0 0 1 0`.
- 3-rd button, leading to the final configuration `0 0 1 0 0 0`.

Gift Delivery (pasi)

Christmas is coming and, as you may easily imagine, this is the most stressful period of the year for Santa Claus. One of his resolutions for this year is to organize better the delivery of the gifts.

Figure 1: Santa driving a van all around the city: look at his load!

He has instructed his assistants to take a map of the city and divide it in $N \times M$ rectangular neighborhoods. For each, they have also reported the total number of gifts which should be delivered. Santa's headquarter, which is the starting point, is located in the neighborhood (x, y) in the map. From there, the van is able to reach an adjacent neighborhood (in the four directions) in a minute.

Santa's benevolence and generosity are not in his favour: when he crosses with his van a neighborhood, even if he has already delivered the gifts, children demand again the same gifts. Santa Claus, who is pure-hearted, is unable to refuse and gives them another load of gifts.

Help Santa with preparation: how many gifts *at maximum* will he need, knowing that after K minutes he wants to be back in the headquarter to enjoy Christmas?

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

Input

The first line contains five integers N, M, x, y, K which indicate that:

- the map has been divided in $N \times M$ neighborhoods (N rows and M columns);
- the neighborhood (x, y) is reserved for Santa's headquarter;
- Santa has K minutes to deliver gifts and come back to his headquarter.

The following N lines contain the description of the map. Each one contains M integers: the number of gifts g requested by children in that neighborhood.

Output

You need to write a single line with an integer: the maximum number of gifts which Santa has to deliver.

Constraints

- $2 \leq N, M \leq 100$.
- $2 \leq K \leq 10^9$ and K is an even number.
- $1 \leq x \leq N$, $1 \leq y \leq M$.
- $0 \leq g \leq 10^9$.
- The top-left neighborhood in the map is at position $(1, 1)$, the bottom-right one is at position (N, M).
- The neighborhood (x, y) hosts the headquarter and thus there are no childen (i.e., there will be zero gifts to deliver there).

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)	$N, M \leq 10$ and $K/2 < N \cdot M$.
– **Subtask 3** (30 points)	$K \leq 1000$ and $K/2 < N \cdot M$.
– **Subtask 4** (25 points)	$K/2 < N \cdot M$.
– **Subtask 5** (25 points)	No additional limitations.

Examples

input	output
`2 2 1 1 4`	`13`
`0 5`	
`3 3`	
`2 3 1 2 6`	`20`
`6 0 2`	
`1 3 0`	

Explanation

In the **first sample case** the headquarter is located in the neighborhood $(1, 1)$. In $K = 4$ minutes Santa can reach $(1, 2)$ (five gifts), $(2, 2)$ (three gifts), $(1, 2)$ (five gifts again) and come back in time.

In the **second sample case** the headquarter is located in the neighborhood $(1, 2)$. One of the most expensive paths consists in visiting these neighborhoods in sequence: $(1, 1)$, $(2, 1)$, $(1, 1)$, $(2, 1)$, $(1, 1)$ and then return to the headquarter. Santa delivers $6 + 1 + 6 + 1 + 6 = 20$ gifts.

Railway Schedule (paths)

The railway network in the *Pordenone* county consists of N train stations connected by $N-1$ tracks (X_i, Y_i) so that from every station is possible to reach any other station: in other words, the tracks form a *tree*.

This choice makes the transportation system extremely inefficient: trains going in opposite directions cannot cross each other on a single track, so they need to perform lengthy and complex manoeuvres to pass each other. The new administration founded its campaign trail on changing this situation once and for all... and now it's time to keep promises!

Figure 1: The Pordenone railway network.

Edoardo, the local leading expert in logistics, already has a mind-blowing idea for fixing the situation: making each track one-way, so that no crossings will ever occur! Of course, the tricky part is choosing the orientations so that the service remains acceptable for the majority of the population. After inspecting the traffic patterns, Edoardo discovered that most people travel between one of M pairs (A_i, B_i) of stations. Thus, an orientation of the tracks will be considered *acceptable* by the population only if for each such pair, either a path from A_i to B_i or a path from B_i to A_i should exist.

However, many acceptable orientations exist and Edoardo cannot choose among them, otherwise his system would be deemed as unfair: the only solution is to use *all* of them in a periodic schedule of daily track orientations. Help Edoardo design such a schedule by counting how many acceptable orientations exist! Since this number may be large, report it **modulo 1 000 000 007**.

✍ The *modulo* operation ($a \bmod m$) can be written in C/C++ as `(a % m)` and in Pascal as `(a mod m)`. To avoid the *integer overflow* error, remember to reduce all partial results through the modulus, and not just the final result!

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

Input

The first line contains the only integer N. The following $N-1$ lines contain integers X_i, Y_i. The next line contains the only integer M. The last M lines contain integers A_i, B_i.

Output

You need to write a single line with an integer: the number of different acceptable orientations.

Constraints

- $1 \leq N, M \leq 300\,000$.
- $1 \leq X_i, Y_i, A_i, B_i \leq N$ for each $i = 0 \ldots N-1$.
- $X_i \neq Y_i$ and $A_i \neq B_i$ for each $i = 0 \ldots N-1$.

- The tracks connect all the stations together.
- There exists at least one acceptable orientation.

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.

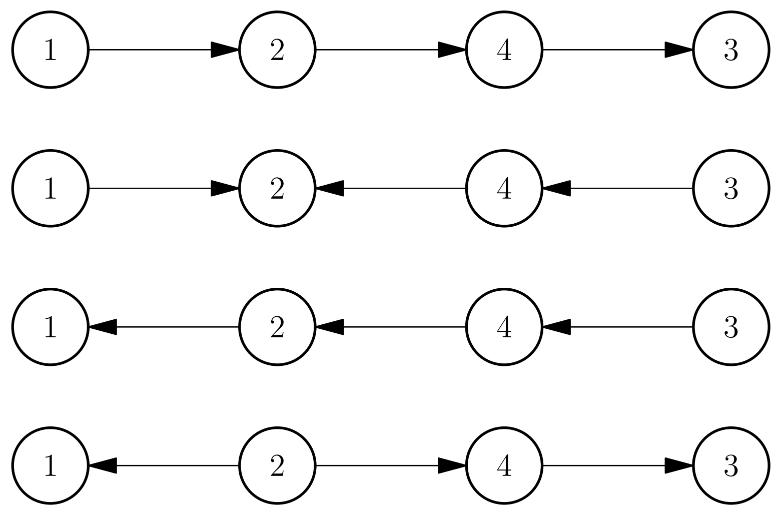
– **Subtask 1** (0 points)	Examples.
– **Subtask 2** (15 points)	$N, M \leq 100$ and $Y_i = X_i + 1$ (the tracks form a path).
– **Subtask 3** (20 points)	$Y_i = X_i + 1$ (the tracks form a path).
– **Subtask 4** (10 points)	$N, M \leq 10$.
– **Subtask 5** (25 points)	$N, M \leq 1000$.
– **Subtask 6** (30 points)	No additional limitations.

Examples

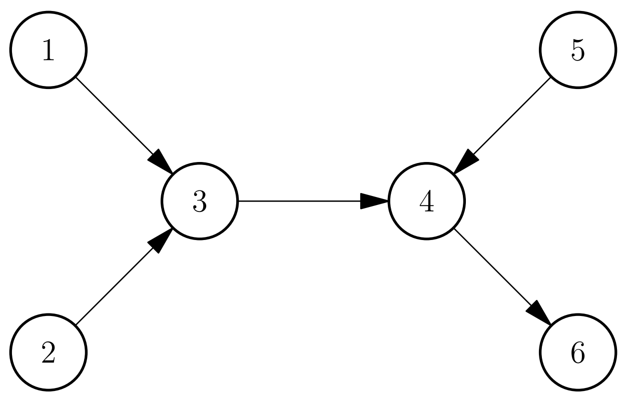
input	output
4	4
4 2	
3 4	
1 2	
1	
2 3	
6	2
1 3	
2 3	
3 4	
4 5	
4 6	
4	
1 6	
5 6	
3 4	
4 2	

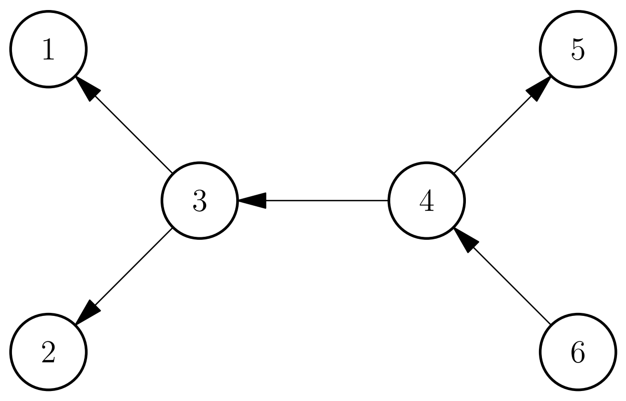
Explanation

In the **first sample case**, the acceptable orientations are the following:



In the **second sample case**, the acceptable orientations are the following:





Forgotten Pencil Case (pens)

Forgetting things is one of the most played "sport" worldwide, and university students make no exception. To be more accurate, students keep forgetting their pencil cases, often because they leave it on the desk at their home.

When the first lecture starts in the morning, they suddenly realize that they are unable to take class notes, as they have no pen at their disposal. What usually happens next is that these students start asking pens to other students, seated increasingly away, until someone happened to brought many pens with them and is thus able to lend one.

Luca, who is notoriously a calm person, finds that this process is rather messy and produces too much noise. Wouldn't it be simpler if, considered a row of seatings, a student could only ask a pen to the student seated at their immediate left or right?

Figure 1: A modern university classroom.

Luca wants to experiment with the rule above, worried that maybe too many students will be left without a pen. Help him: given a row of N students, each with P_i pens in their pencil case, what is the minimum number of students who will not be able to get a pen from their immediate neighbors (using the best strategy while lending pens)?

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

Input

The first line contains the number N of students in the row. The second line contains N integers P_i, the number of pens that the i-th student has in their pencil case (possibly zero, if the pencil case was forgotten).

Output

You need to write a single line with an integer: the minimum number of students that will be left without a pen, respecting the rule described above.

Constraints

- $1 \le N \le 1\,000\,000$.
- $0 \le P_i \le 100$ for each $i = 0 \ldots N-1$.
- The i-th student can ask for a pen only to the $(i-1)$-th and to the $(i+1)$-th student, unless the i position is an extreme of the row: the student seated in position 0 can only ask to the student in position 1, and the student in position $N-1$ can only ask to the student in position $N-2$.
- When a student has more than one pen, they will always lend one when asked. Conversely, if a student has just one pen, they will never lend their only pen.
- Students who have not forgot their pencil case (i.e., $P_i \ge 1$) will never ask others for more pens.

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** (5 points) $N = 1$.

– **Subtask 3** (10 points) $0 \le P_i \le 1$ for each $i = 0 \ldots N-1$.

– **Subtask 4** (35 points) $N \le 15$.

– **Subtask 5** (50 points) No additional limitations.

Examples

input	output
2	0
0 3	
6	2
0 2 0 1 1 0	

Explanation

In the **first sample case** there are just two students in the row: the first one has forgotten his pencil case and has no pens, while the second one has three pens. The first student can ask a pen to the student sat on his right, and they will have one and two pens, respectively. Thus, no student will be left without a pen.

In the **second sample case** there are six students in the row and the first, the third and the sixth one have forgotten their pencil case. The last student is unable to get a pen: she has only one neighbor, who has just a pen for himself. The first student can ask a pen to the student sat on his right, but doing so prevents the third student from getting a pen. Simmetrically, if the pen is asked by the third student to the second one, the first one will not be able to get a pen.

Summing it up, there is no way of lending pens that leaves less than two students without a pen.

Italian Scopa (scopa)

Scopa is an Italian card game, and one of the two major national card games in Italy. The name is an Italian noun meaning "broom", since taking a scopa means "to sweep" all the cards from the table. Watching a game of scopa can be a highly entertaining activity, since games traditionally involve lively, colorful, and somewhat strong-worded banter in between hands.

Figure 1: Italian people playing cards – © Michal Dzierza https://www.dzierza.com/

Each hand is formed by 3 cards, and there are 4 more cards which are placed on the table. Cards have a number (from 1 to 10) and a suit (Golds, Swords, Cups, Batons). When it's their turn, a player will use one of his cards to take some more cards from the table, he can do so only if the values of those cards sum up to the value of the card he's using (regardless of the suit). If he manage to obtain such a sum he will keep all those cards.

For example if in his hand he has a *seven* and on the table there is a 3, a 4, a 5 and a 10 he can choose to take the 3 and the 4.

There are many strategy to play this game, Giorgio's one is the following:

- If he can get a **sette bello** (the 7 of Golds), he will;
- If he can make a **scopa** (take all the cards from the table), he will;
- He'll try to take as many 7 as he can;
- Otherwise he will try to take as many cards as he can from the table.

As you can imagine, Giorgio is an avid scopa player. He likes this game a lot but he keeps on losing against the more experienced elders of his village. That's why he's working on a complicated Machine

Learning software that is able to beat, utterly destroy and humiliate any opponent that comes its way. This will surely earn Giorgio the respect he deserves from his fellow villagers.

After warming up his Neural Network, now he's missing the core piece of the program: the function that, given a hand of cards and the set of cards on the table, chooses *which is the best move* to make. Help Giorgio by implementing this logic.

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

Input

Each card is described by a number followed by a suit. The first line contains 3 cards separated by whitespace: those in a possible hand. The second line contains 4 cards separated by whitespace: those on the table.

Output

You need to write a single line with all the cards taken in the best move, separated by whitespace, including the one from the hand. The order of the cards does not matter.

Constraints

- Each card's number is an integer from 1 to 10.
- Each card's suit is one of four characters: `G`, `S`, `C` or `B`, respectively: golds, swords, cups, batons.
- There are no duplicate cards.
- It will always be possible to take at least one card from the table.
- The solution is unique.

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) The solution is to use a 7 to take a 7 from the table.

– **Subtask 3** (50 points) The best move never requires to take more than a card from the table.

– **Subtask 4** (30 points) No additional limitations.

Examples

input	output

`5G 7G 8S`
`2G 3C 7S 2C`
	`7G 7S`

`5G 7S 8S`
`2G 3C 1S 2C`
	`8S 2G 3C 1S 2C`

Explanation

In the **first sample case** the best solution is to take the *sette bello* from the hand and the 7 from the table. According to the rules it's better to take a single seven than more non-7 cards.

In the **second sample case**, the best move is to use the 8 of Swords to sweep all the cards on the table. Note that because the 7 is not a *sette bello* it's better to score a *scopa*.

Meal Tickets (`tickets`)

Luca is starving, his fridge is empty and in his small town there aren't fast food resturants nor food delivery. His only option is to go to the small supermarket near his house.

The plan is simple: he will rush between the aisles, pick just the dinner, pay and finally eat on his way home. There are just a few lines where to pay, and of course he went to the less crowded one: what a bad move! The only person in front of him is Evil William, and he has *loads* of meal tickets.

A *meal ticket* is like a banknote of a certain value, you can use it to pay instead of money, but some rules apply in this supermarket: you cannot use 2 tickets of the same value and the total value of used tickes must not exceed the total of your shopping.

Evil William has to pay T euros and he has the ticket of *all* the possible values (from 1 euro up to at least T euros). William could use the single ticket of T euros, but he is evil and is going to use as many tickets as he can in order to waste as much time as possible!

Help Luca, who is hungry, counting how many tickets Evil William can use at most.

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

Input

The first and only line contains the only integer T.

Output

You need to write a single line with an integer: the maximum number of tickets Evil William can use.

Constraints

- $1 \leq T \leq 10^{18}$.

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) $T \leq 10$.
- **Subtask 3** (20 points) T is the sum of the first k natural numbers.
- **Subtask 4** (20 points) $T \leq 100$.
- **Subtask 5** (30 points) $T \leq 10^6$.
- **Subtask 6** (20 points) No additional limitations.

Examples

input	output
`9`	`3`
`10`	`4`

Explanation

In the **first sample case** Evil William can use the tickes of value $5 + 3 + 1 = 9$. He can't use 4 tickets without exceeding 9 euros, so 3 is the greatest amount.

In the **second sample case** he can use the tickets of value $1 + 2 + 3 + 4 = 10$. This is the only valid combination of four tickets.