Problem A. Lucky 7

Source file name:	Lucky.c, Lucky.cpp, Lucky.java, Lucky.py
Input:	`Standard`
Output:	`Standard`

Fact or Fiction, some people consider 7 to be a lucky digit/number.

Given a number, determine how lucky the number is by printing one of four values:

- Print 0 if the number does not contain 7 and is not divisible by 7.
- Print 1 if the number does not contain 7 but is divisible by 7.
- Print 2 if the number does contain 7 but is not divisible by 7.
- Print 3 if the number does contain 7 and is divisible by 7.

Input

There is only one input line; it contains an integer between 1 and 10^9, inclusive.

Output

Print one of the four messages as described above.

Example

Input	Output
25	0
42	1
170	2
777	3
1	0
70	3

Problem B. We Want You Happy!

Source file name:	Happy.c, Happy.cpp, Happy.java, Happy.py
Input:	`Standard`
Output:	`Standard`

The United Credit Finance (UCF) is running a simple scenario to see how many customers are happy with the company. UCF has one person (teller) serving the customers. Customers are numbered $1 \ldots n$, and they arrive for service in sequential order, i.e., Customer 1 arrives first, then Customer 2, then Customer 3, etc. Also, no two customers arrive at the same time, i.e., Customer 2 will arrive later than Customer 1, Customer 3 will arrive later than Customer 2, etc. Customers are also processed in the order of arrival (i.e., not out of order).

As you might have noticed while waiting in a line, some customers get impatient and leave. Given the information about the UCF customers, you are to determine which customers are happy, i.e., they don't leave before being processed.

Input

The first input line contains an integer, n ($1 \le n \le 10^3$), indicating how many different customers are arriving. Each of the next n input lines contains the information about a customer as follows:

- customer number (1, 2, 3, etc., in sequential order),
- arrival time (an integer between 1 and 10^4, inclusive),
- service time (an integer between 1 and 10^4, inclusive) indicating how long it will take the teller to process this customer (once the customer is at the teller),
- patience time (an integer between 1 and 10^4, inclusive) indicating how long the customer will wait in line before giving up and leaving, i.e., if the customer is not at the teller, the customer will leave. Note that if a customer leaves, the teller will not process them, i.e., the customer will not take the teller's time.

Output

Print the happy customer numbers in sequential order (happy means they were processed, i.e., did not leave).

Example

Input	Output

```
7
1 50 10 5
2 52 5 4
3 58 20 5
4 85 7 10
5 86 10 1
6 88 20 3
7 89 30 3
```
	```
	1
	3
	4
	7
	```

Explanation

Explanation of Example Input/Output:

- The teller is done with Customer 1 at time 60 ($50 + 10$). This means the teller can process the next customer at time 60.

- Customer 2 arrives at time 52. The patience time for this customer is 4 so, since they arrive at time 52, if processing them does not start by time 56 $(52 + 4)$, they leave. The teller is done with Customer 1 at time 60 so Customer 2 leaves. Note that if a customer leaves, they will not take the teller's time.
- Since Customer 2 leaves, the teller can start Customer 3 at time 60 (this customer was willing to wait until time 63 to start being processed). The teller is done with Customer 3 at time 80 $(60+20)$.
- The teller is done with Customer 4 at time 92 $(85 + 7)$. Note that the teller doesn't have customers from time 80 (when the teller is done with Customer 3) to time 85 (when Customer 4 arrives).
- Customer 5 arrives at time 86 and expects to be processed starting at time 87, so they leave (the teller is done with Customer 4 at time 92).
- Customer 6 arrives at time 88 and expects to be processed starting at time 91, so they leave (the teller is done with Customer 4 at time 92).
- Customer 7 arrives at time 89 and expects to be processed starting at time 92. The teller is available at that time (teller is done with Customer 4 at time 92) so this customer is processed.

Problem C. Snailography

Source file name:	Snailography.c, Snailography.cpp, Snailography.java, Snailography.py
Input:	`Standard`
Output:	`Standard`

Snails have many enemies including snakes, turtles, and birds. So, snails need to communicate their travel paths using cryptography to avoid their routes being detected.

The encryption technique: The message will contain only letters. Let's assume the message length is L. We use the smallest odd integer N such that $N \times N \geq L$. Then, an $N \times N$ table is used to encrypt the message as follows: Put the first letter of the message in the cell at the center of the table and then put the remaining letters in the table by moving in a circular way (snail like) around the center cell. For example, the table to the right shows the order for placing the letters from the message in a 7×7 table. So, from the center cell, we move up, then move right, then move down, then move left, then move up, etc.

49	26	27	28	29	30	31
48	25	10	11	12	13	32
47	24	9	2	3	14	33
46	23	8	**1**	4	15	34
45	22	7	6	5	16	35
44	21	20	19	18	17	36
43	42	41	40	39	38	37

Below are some sample encryptions. To help with the illustrations, when encrypting the message, if there are more cells in the table than there are letters in the message, we put the character '#' in the extra cells.

```
Message: ABCDEFGH
Encryption:
#BC
HAD
GFE

Message: ABCDEFGHIJKLMNOPQRSTUVW
Encryption:
#JKLM
#IBCN
WHADO
VGFEP
UTSRQ

Message: ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuv
Encryption:
#Zabcde
vYJKLMf
uXIBCNg
tWHADOh
sVGFEPi
rUTSRQj
qponmlk
```

Given the size of the two-dimensional table to use and the original message, you are to encrypt the message (to help snails live longer lives).

Input

The first input line contains an odd integer, n ($1 \leq n \leq 19$), indicating the table size to use. The second input line will provide the message to encrypt, a string of $1 - 361$ (19×19) letters (lowercase and uppercase). Assume that the message will fit in the table.

Output

Print the encrypted message on one output line using the row-major order, i.e., print Row 1 followed by Row 2, followed by Row 3, etc.

Remember to print a newline character after printing the last row.

The output should not include '#' characters.

Example

Input	Output
3 ABCDEFGH	BCHADGFE
5 ABCDEFGHIJKLMNOPQRSTUVW	JKLMIBCNWHADOVGFEPUTSRQ

Problem D. Good Goalie

Source file name:	Goalie.c, Goalie.cpp, Goalie.java, Goalie.py
Input:	`Standard`
Output:	`Standard`

The 2023 FIFA Women's World Cup was held during the summer. Several games ended in ties and penalty kicks had to be taken to determine the winner.

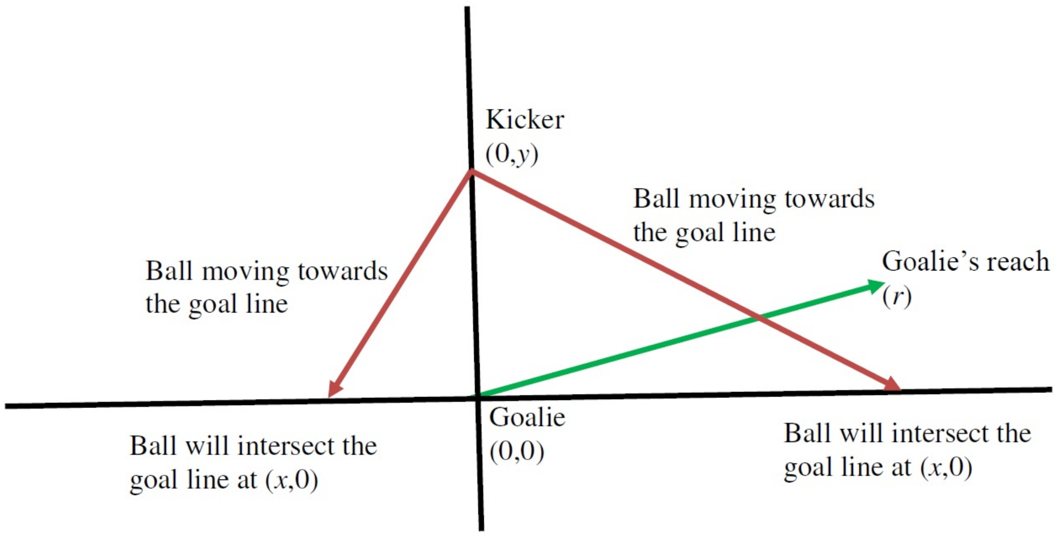
The professional soccer goal size (length) is 7.32m. The success rate of penalty kicks at the professional level is above 80%. If the goal length was much longer, can this success rate reach 100%?

To simplify the problem, we will assume that there is no concept of time or ball speed, so if the goalie's reach intersects the ball's travel path, the goalie will block the ball.

As illustrated in the picture below, we'll assume the goalie is at $(0,\ 0)$ and the penalty taker (kicker) is at $(0,\ y)$; y will be a positive integer.

The kicker will kick the ball towards the goal line (to the left or right of the goalie). We'll assume the ball will intersect the goal line at $(x,\ 0)$; x will be a non-zero integer.

The goalie will dive to the left or right (based on the direction of the ball). The goalie has a limited reach r; r will be a positive integer.


Kicker
(0,y)
Ball moving towards the goal line
Ball moving towards the goal line
Goalie's reach (r)
Goalie (0,0)
Ball will intersect the goal line at (x,0)
Ball will intersect the goal line at (x,0)

The goalie will save the penalty if the point representing the ball (while travelling) is on the line representing the goalie's reach. The goalie can save the penalty (i.e., block the ball) on the X-axis as well.

The primary objective for the goalie is to save the penalty and the secondary objective is to dive as close to the X-axis as possible, i.e., the angle between the goalie's reach and the X-axis is as small as possible.

Given the kicker's position (y), intersection point of the ball with the goal line (x) and the goalie's reach (r), determine whether the goalie will save (block) the penalty kick.

Input

There is only one input line; it provides three integers y, x and r, as described above. Assume $1 \leq y \leq 10^3$, $-10^3 \leq x \leq 10^3$ (excluding zero), and $1 \leq r \leq 10^3$.

Output

Print -1 if the goalie cannot save the penalty; otherwise print the minimum angle that the goalie needs to dive to block the ball. Answers within an absolute error of 10^{-6} will be accepted.

Example

Input	Output
10 100 2	-1
10 3 3	0
3 10 6	11.915197445846822
20 -200 5	-1
20 -5 5	0
5 -20 8	23.289018023569476

Problem E. Most Valuable Pez

Source file name:	MVP.c, MVP.cpp, MVP.java, MVP.py
Input:	`Standard`
Output:	`Standard`

Some UCF students know that Arup loves collecting Pez dispensers. Pez dispensers dispense 12 candies in a fixed sequence. Arup is so discerning, that he can actually tell the difference between the taste of different Pez candies and values them differently.

Imagine a situation where he wants to eat exactly 7 Pez candies and his four dispensers have candies with the following values, each listed from top to bottom of the respective dispenser:

3	4	5	1
6	1	3	4
8	1	2	3
2	1	6	8
2	1	6	4
5	70	2	9
6	4	3	2
1	1	1	1
2	5	6	6
3	3	2	5
1	2	3	4
4	3	2	1

In this situation, if he wanted to maximize the total value of the 7 candies he eats, he should take 6 candies from the second Pez dispenser and 1 candy from the third dispenser, for a total value of 83. He is required to eat the candies in a specific order from each dispenser (from the top). In this particular situation, it's worth it to eat all of the candies with value 1 in the second stack in order to get to the candy worth 70. From there, the maximum remaining candy at the top of any of the dispensers makes the most sense to add.

Given the values of each of the candies in Arup's Pez dispensers as well as the number of candies he wants to eat, determine the maximum total value he will be able to obtain if he chooses his candies optimally.

Input

The first input line contains two integers: n ($1 \leq n \leq 1000$), indicating the number of Pez dispensers Arup has, and k ($1 \leq k \leq 12 \cdot n$), representing the number of candies Arup is willing to eat.

Each of the next n input lines contains 12 integers (each integer between 1 and 300, inclusive), indicating the value of the candies in one of Arup's Pez dispensers. These values are listed from top to bottom of each Pez dispenser.

Output

Print the maximum total value of the candies Arup could choose, if he chooses optimally.

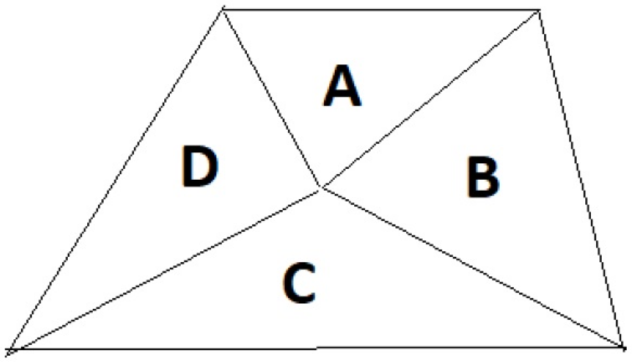
Example

Input	Output
4 7	83
3 6 8 2 2 5 6 1 2 3 1 4	
4 1 1 1 1 70 4 1 5 3 2 3	
5 3 2 6 6 2 3 1 6 2 3 2	
1 4 3 8 4 9 2 1 6 5 4 1	
1 6	120
20 30 40 15 1 14 200 2 300 2 2 2	

Problem F. Land Division

Source file name:	Land.c, Land.cpp, Land.java, Land.py
Input:	`Standard`
Output:	`Standard`

Uncle Trapezoid has a piece of land in the shape of a trapezoid that he would like to split into four pieces to bequeath to his four nieces. He is fairly lazy, so all he will do is pick a particular point inside of the trapezoid, and then the four pieces will be the four triangles formed by this point and each of the four sides of the trapezoid as illustrated below:



Naturally, Uncle Trapezoid wants to be as fair as possible and minimize the difference in area between the smallest triangle and the largest triangle.

Given the length of both bases and the height of a trapezoid, determine the smallest possible difference in area between the smallest and largest triangles formed by choosing any point inside of the trapezoid to form the four triangles (one with each side of the trapezoid).

Note: The bases must be parallel with a distance equal to the height between the two lines the bases define. However, where the bases are located relative to each other does not affect the answer to this problem.

Input

There is only one input line; it contains three integers: b_1 $(1 \leq b_1 \leq 10^4)$, b_2 $(1 \leq b_2 < b_1)$, and h $(1 \leq h \leq 10^4)$, representing the lengths of the long base, short base and height of the trapezoid, respectively.

Output

Print the smallest possible difference in area between the smallest and largest triangles that the trapezoid could be partitioned into. Any answer within an absolute tolerance of 10^{-6} will be accepted.

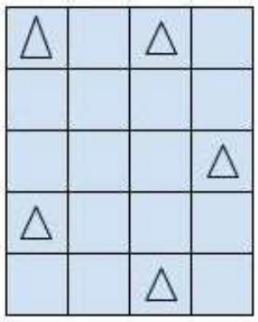
Example

Input	Output
`20 12 16`	`8.000000000`
`20 18 10`	`0.263157895`

Problem G. Not So Close

Source file name:	Notsoclose.c, Notsoclose.cpp, Notsoclose.java, Notsoclose.py
Input:	`Standard`
Output:	`Standard`

Orlando is quickly growing and new houses have to be built. However, people don't like being too close to each other these days. Universal Condos Forever (UCF) is building some housing units. The land they have is parceled out in grid squares as shown below:



In the example above, there are 5 rows and 4 columns. The owners of the condos, denoted by triangles, do NOT want other condos in ANY of the potentially 8 adjacent (up, down, left, right, diagonal) grid squares. For example, the above layout is a valid arrangement of five condos.

Given the number of rows and columns in the lot that UCF is building condo units, determine the number of different sets of placements of condos they could choose. Two sets are different if in one set a condo is built on a specific square but in the other set no condo is on that same exact square, or vice versa. Since the number of different arrangements could be very large, find the value modulo $10^9 + 7$.

Note: trivially, building no condos is always a valid arrangement.

Input

There is only one input line; it contains two integers: r ($1 \leq r \leq 10$) and c ($1 \leq c \leq 10^3$), representing the number of rows and columns (respectively) for UCF's lot.

Output

Print the number of valid arrangements of condos, modulo $10^9 + 7$.

Example

Input	Output
`2 3`	`11`
`5 4`	`1213`

Explanation

Of the first Example Input/Output:

There is 1 way of placing zero condos.
There are 6 ways of placing one condo.
There are 4 ways of placing two condos.
We cannot place three or more condos.
So, the total number of valid arrangements is $1 + 6 + 4 = 11$.

Problem H. The Duel of Smokin' Joe

Source file name:	Duel.c, Duel.cpp, Duel.java, Duel.py
Input:	`Standard`
Output:	`Standard`

In the wild west, they take their Computer Science very seriously. Smokin' Joe is widely known for his skills, and outlaws come from across the land to challenge him in a good ol' fashioned sortin' duel. The way that outlaws duel in the wild west, of course, is trying to be the one who makes the last move to sort an array of integers.

One fateful day, Smokin' Joe's nemesis, an infamous character known as The Outlaw, came to town and challenged Joe to a duel. The rules of the duel are as follows: the outlaw will spin the chambers, and a random permutation of length n will fall out (a permutation of length n is an array of the integers from 1 to n in any order). Smokin' Joe and The Outlaw will take turns swapping two elements in this permutation, until the permutation is sorted in increasing order. The winner is the player who makes the last swap. Once an element is in its final position (i.e., value k is at position k), that element cannot be moved for the rest of the game; otherwise, there are no restrictions on which two elements can be swapped.

Both Smokin' Joe and The Outlaw will always play perfectly, i.e., they play to win and not necessarily to sort the list as quickly as possible. Can you predict who will win?

Given a permutation, predict the winner of the game, where players take turns swapping elements, and an element cannot move once it's in its final sorted position. The winner is the person who makes the final move to sort the array. Note that Smokin' Joe goes first, i.e., makes the first move.

Input

The input will consist of two lines. The first input line contains an integer, n ($1 \leq n \leq 10^6$), the length of the permutation. The second input line contains n distinct integers, the permutation. It is guaranteed that it will be a permutation of the integers from 1 to n, and the permutation is not already sorted.

Output

If Smokin' Joe wins, print "Smokin Joe!" (note that the apostrophe of Smokin' is omitted in the output since apostrophe may come out as different ASCII characters on different machines and may cause "format errors" in the output). If The Outlaw wins, print "Oh No!".

Example

Input	Output

`3`
`1 3 2`
	`Smokin Joe!`

`4`
`4 3 2 1`
	`Oh No!`

Problem I. Drake Robbing

Source file name:	Drake.c, Drake.cpp, Drake.java, Drake.py
Input:	`Standard`
Output:	`Standard`

A greedy drake in a long-forgotten castle stores their treasures. The drake is obsessed with certain types of treasures. It has been rumored that although the types are few, the volume is plenty. You know that the treasures would be worth a lot in the black market, which is why you waited with your carrying pack for the time when the drake leaves to pillage a nearby town.

When you enter the drake's abode, you notice near countless treasures and you are afraid that your pack may not be big enough. You are not planning on a return trip so your goal will be to get the most value out of this one and only opportunity.

Given the number of treasure types (each with some multiplicity, worth, and volume), and some maximum volume that can be carried, determine the most worth you can carry within the given volume limit.

Input

The first input line contains two integers: n ($1 \leq n \leq 20$), indicating the number of treasure types and V ($1 \leq V \leq 10^{15}$), indicating the maximum volume you can carry. Each of the next n input lines contains three integers: m ($1 \leq m \leq 10^{15}$), indicating the multiplicity of the item, w ($1 \leq w \leq 10^3$), indicating the worth of each of this item, and v ($1 \leq v \leq 10^3$), representing the volume of each of this item.

Output

Print the maximum worth you can carry.

Example

Input	Output

`3 100`
`20 1 1`
`12 5 4`
`10 11 10`
	`117`

`1 1000000000000`
`1000000000000 1000`
	`1000000000000000`

Problem J. Grow Measure Cut Repeat

Source file name:	GMCR.c, GMCR.cpp, GMCR.java, GMCR.py
Input:	`Standard`
Output:	`Standard`

The UCF Programming Team Coaches, besides being the most experienced coaching group in the world, are also conscious of the forests in our planet disappearing. The coaches have taken the initiative of developing a `1-D` forest. They have planted a tree seed at each integer location in the range of 1 and 10^5. These 10^5 trees are currently of height 0 (zero) but the coaches have recruited three forest rangers (Alice, Bob, and Cutz) to maintain this `1-D` forest. The forest rangers are really like fairies once you see their power!

Alice loves to watch the trees grow. She was gifted a magical watering pot that can make a tree grow K units, where K is the amount of water she pours on the tree. Additionally, each tree that is D ($D < K$) distance away from the spot will grow K-D units.

Bob loves to climb the trees in the forest. Bob has a special tape that can measure the height of any tree.

Cutz likes to cut all the trees in the forest to a specific height, i.e., taller trees are cut to all have a specific height (note that shorter trees are not affected/cut). More specifically, Cutz decides on a *threshold* (maximum height) and then all the trees taller than this threshold will be cut to have this new height. Cutz likes decreasing sequence (!) so they always cut at a height lower than the previous one they used (with the only exception being the first cut).

For a given list of commands that Alice, Bob, and Cutz performed, determine the height Bob measured in each of his commands. Remember that all the trees start at height 0.

Input

Input will begin with a single integer, t ($1 \le t \le 2 \cdot 10^5$), representing the number of transactions.

Each of the following t input lines will represent a single transaction in one of the following three forms:

- $A\ L\ K$
- $B\ L$
- $C\ H$

For the transaction beginning with A, Alice will water the tree at position L ($1 \le L \le 10^5$) with K ($1 \le K \le 10^6$) units of water, i.e., certain trees grow in height as described above.

For the transaction beginning with B, Bob will measure the tree at position L ($1 \le L \le 10^5$).

For the transaction beginning with C, Cutz will cut all the trees in the forest to height H ($1 \le H \le 10^8$).

Output

There is no output required for transactions beginning with A or C. For each transaction beginning with B, print the height of the tree that Bob measured.

Example

Input	Output
6	4
A 7 5	7
B 6	6
C 4	
A 5 4	
B 6	
B 7	
7	8
A 10 8	2
B 10	4
C 6	
A 10 8	
C 4	
B 3	
B 5	

Problem K. Bad Bunny

Source file name:	Bunny.c, Bunny.cpp, Bunny.java, Bunny.py
Input:	`Standard`
Output:	`Standard`

In an unenchanted forest there is a mundane rabbit that steals from your garden and you end up chasing after it. The rabbit constantly moves its burrow, so knowing your options for catching the bugger is not easy. The forest is a collection of clearings joined by trails, each trail connecting exactly two clearings. You know, when you chase the rabbit, where it starts and ends its journey. You are solely interested in knowing clearing(s) you can set up a snare to ensure that you trap the rabbit. Note that setting up a snare in rabbit's starting or ending clearings will definitely catch the rabbit. (We trust that you don't plan on injuring the rabbit, only taking back your vegetables.)

It will always be the case that the rabbit can reach any clearing from any other clearing, i.e., the rabbit can travel from any clearing to any other clearing.

Given the description of the forest, determine the number of possible clearings where you are guaranteed to be able to trap the rabbit.

Input

The first input line contains two integers: C ($1 \leq C \leq 10^5$), representing the number of clearings, and T ($C - 1 \leq T \leq 2 \cdot 10^5$), representing the number of trails.

Each of the following T input lines contains two integers a and b ($1 \leq a, b \leq C$; $a \neq b$), representing that clearings a and b are connected by a trail. No pair of clearings are connected directly by more than one trail.

After the initial information for all the clearings, the input will have a set of transactions (operations) to be processed. This section of the input starts with an integer, t ($1 \leq t \leq 10^5$), indicating the number of transactions. Each of the next t input lines contains a transaction to be processed with the following format:

- Trap Transaction: This input line provides two valid clearings s and d. The bunny is moving from s to d and you need to determine the number of clearings a trap could be placed such that the bunny can be caught. Note that only a single trap must be placed in a single clearing to catch the rabbit, and we are interested in knowing how many such clearings exist. Note also that s and d are two of the answers, i.e., the result will be at least 2.

Output

For each Trap transaction, print the number of locations that you can place a trap to guarantee the bunny will be caught as it travels from its starting clearing to the ending clearing.

Example

Input	Output
9 10	2
1 2	5
1 3	4
2 4	3
3 4	2
3 5	3
3 6	3
5 6	2
4 7	3
7 8	2
7 9	
10	
1 2	
8 6	
1 9	
9 8	
4 2	
6 4	
1 6	
9 7	
4 5	
3 4	
4 4	2
1 2	3
2 3	2
3 4	
4 2	
3	
1 2	
1 4	
2 3	

Problem L. Cram

Source file name:	Cram.c, Cram.cpp, Cram.java, Cram.py
Input:	`Standard`
Output:	`Standard`

You want to compress a given text passage using backreferences. A *backreference* is a pair of numbers $[a,\ b]$ indicating that the next b characters of the string are the same as the b characters starting a characters back from the current position. The two strings may overlap, i.e., a may be smaller than b.

Each backreference costs three bytes to encode, regardless of the number of characters represented by the backreference. String characters cost one byte each to encode.

For instance, the string

```
abcabcabcabc
```

has 12 characters. But the last nine can be represented as a backreference to the first nine, as follows:

```
abc[3,9]
```

The total cost of this encoded string is 6: 3 bytes for the string `abc`, and 3 bytes for the backreference.

Output the minimum cost to encode the text passage.

Input

The single line of input contains a string s, with $1 \le |s| \le 10^5$. This line of text consists of uppercase letters (‘`A`’-‘`Z`’), lowercase letters (‘`a`’-‘`z`’), and spaces. There will not be any spaces at the beginning or end of the line, and no space character will be adjacent to another space character.

Output

Output a single integer, which is the minimum cost to represent the input string using backreferences.

Example

Input	Output
`abcabcabcabc`	`6`
`aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`	`4`
`A man a plan a canal Panama`	`25`