Problem A. Aperiodic Appointments

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

Nick has always struggled with maintaining habits. The problem is that he just can't stop maintaining them. If Nick does something K times in a row, he has to keep doing it forever.

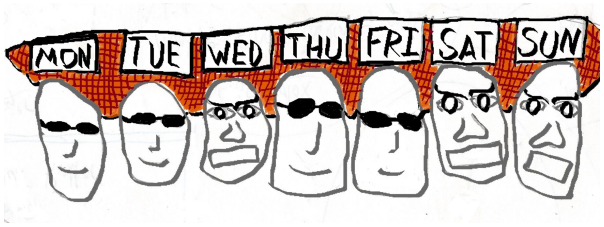
Luckily, he has started visiting Dr Patternson, an expert in PBT (Pattern Breaking Therapy). The principle of PBT is simple: Nick will visit Dr Patternson every day, and if he has done the same thing K times in a row on a specific visit, the doctor will charge him money. This will motivate Nick to not continue this habit.

PBT has worked out great for Nick, as he has now successfully quit all his habits. Except for one, the habit of visiting Dr Patternson. The frequent visits are starting to take a toll on Nick's economy, so your task is to calculate how many times he has to pay the doctor for the next N days.

Formally, let $s = s_1 s_2 s_3 \dots s_N$ be a string consisting of zeroes and ones. A one means that Nick has to pay the doctor on the ith day. This string is generated one character at a time, in the following way:

1. $s_i = 0$ if $i \leq K$.
2. If $i > K$, then $s_i = 1$ if the previous characters contains a pattern that repeats K times. More specifically, let $s' = s_1 s_2 \dots s_{i-1}$. If there is a nonempty string t such that the last $|t| \cdot K$ characters of s' can be written as $t + t + \dots + t$, then $s_i = 1$. Otherwise $s_i = 0$.

You are given the numbers N and K, and your task is to calculate the number of ones in the string s.



The picture represents Example 1. An angry face means that Nick had to pay on the corresponding day.

Input

The input consists of one line with the integers N and K ($1 \leq N \leq 10^9$, $2 \leq K \leq 10^9$).

Output

Print one integer, the number of ones in the string s.

Example

Input	Output
7 2	3
99 5	19

Problem B. Baseball Court

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


Image taken from commons.wikimedia.org

After lengthy meetings on the subject, the NCPC jury has decided that getting the contestants' blood pumping would result in better contests. Thus they figured it would be a good idea to compete for bragging rights in some kind of sport prior to the programming contest. For this they need some place to compete and to pick a sport to compete in. One of those is solved easily by drawing at random from a hat, which results in the chosen sport being baseball. That simply leaves the matter of making a suitable court to play on. The NCPC jury has access to a rectangular plot of land a by b meters in size. Furthermore, they have N square tiles of grass 1 by 1 meter in size they can place on this plot to create the court. All grass tiles must be placed with their sides parallel to the edges of the plot.

The south-west corner of the land is chosen to be the batting point. For the placement of the grass tiles to constitute a valid court, two conditions must be met. Firstly, for any given tile the south and west sides must either lay directly against the edge of the plot or directly against another tile. This is required to make sure the ball doesn't exit and reenter the court while following a straight trajectory. Secondly, all tiles which have no adjacent tile to the north nor to the east must have the same manhattan distance from the batting point. This is to prevent batters from preferring some directions over others.

Your task is to find the number of ways to place the tiles so that it creates a valid baseball court.

Input

The first line of the input contains a single positive integer N ($1 \leq N \leq 10^4$), the number of grass tiles available. The second line of the input contains two positive integers a and b ($1 \leq a, b \leq 10^4$), the size of the land the court can be made within.

Output

Print the number of valid ways to place the grass tiles to make up a baseball court. All n grass tiles must be used. Since this number might be very large, print the answer modulo $10^9 + 7$.

Example

Input	Output
15 3 8	3
15 3 5	1
15 3 4	0

Problem C. Converting Romans

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

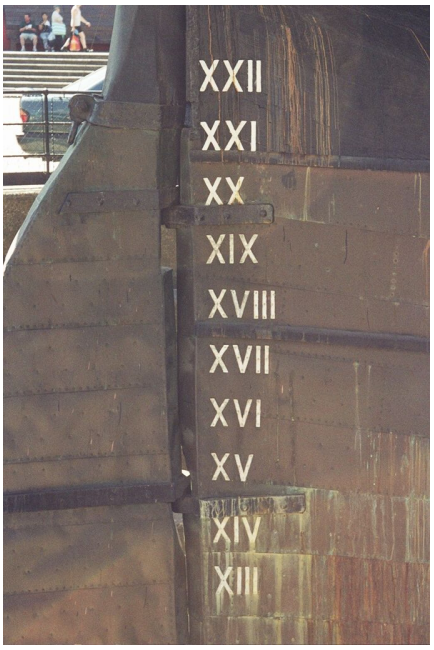
The Roman numerals are a numeral system that originated in ancient Rome and was widely used throughout Europe well into the Late Middle Ages. It differs from the Arabic system that we mostly use today in that numbers are written with combinations of letters from the Latin alphabet, where each letter is assigned a fixed integer value. Over the years many different letters were used, which was a cause of frequent confusion. But "modern" style uses only these seven:

I	**V**	**X**	**L**	**C**	**D**	**M**
1	5	10	50	100	500	1000

A very common misunderstanding of the Roman numerals is that they use a universal subtractive syntax. In a subtractive syntax, a lower digit written before a larger digit will be subtracted from the larger digit. 4 can be written as IIII and IV $(5 - 1)$. Romans themselves only really used the subtractive syntax for smaller numbers. For larger numbers, it was largely avoided, to give more clarity. For example, 499 can be written LDVLIV, XDIX, VDIV or ID, while older sources most likely would use CDXCIX. Ain't Roman numbers fun?

You are given n numbers written in the Roman numeral system, and your task is to convert them to the regular Arabic number system.

For the purposes of this problem we will subtract any digit written to the left of a larger digit, even if they are not directly adjacent.



Roman numerals on stern of the ship Cutty Sark showing draught in feet. The numbers range from 13 to 22, from bottom to top. (CC BY-SA 3.0)

Input

The inputs starts with an integer $0 < n \leq 10^3$. Then follow n lines, each containing up to $3 \cdot 10^5$ Roman digits. The total number of Roman digits in the input is at most $3 \cdot 10^5$.

Output

Output the corresponding Arabic numbers, one number per line.

Example

Input	Output
5 CDXCIX ID IV IIIIV IIIIIIV	499 499 4 1 -1

Problem D. Double Deck

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

You are playing a new card game. In the game you have two decks of cards each consisting of $N \cdot K$ cards labeled with an integer from 1 to N, inclusive. Also, each type of card appears precisely K times in each deck.

Image taken from wikimedia.org

The rules of the game are simple. You shuffle both decks and place them face up in front of you, so at each point in time you see the top card in each deck. If the top cards are the same you can take them both and get one point. Otherwise you must discard either card. Your goal is to get as many points as possible.

You have just finished playing a round of this game and you want to know what the maximum score was, knowing the layout of both decks.

Input

The first line of the input contains two integers N and K ($1 \leq N \leq 10^4, 1 \leq K \leq 15$). The second and third line of the input each contain $N \cdot K$ integers x_i ($1 \leq x_i \leq N$), describing the layout of the decks. The first number x_1 is the topmost card in the deck, x_2 is the second, and so on.

No integer in the second line and third line is repeated more than K times per line.

Output

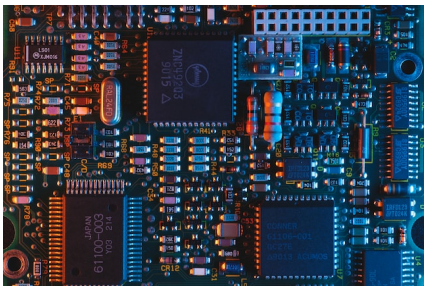
Print a single integer, the maximum possible score.

Example

Input	Output
3 2 3 1 2 3 1 2 2 1 3 1 3 2	4
5 3 2 3 4 5 3 5 2 2 4 3 5 1 1 1 4 5 2 3 2 3 1 4 5 1 4 5 1 4 3 2	8

Problem E. Electronic Components

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



Picture by Umberto, Unsplash Licence

Sara is doing her summer internship at NCPC (Never Crashing Personal Computers). One day, a rare creature appeared in the office: an algorithmic problem!

The company has a machine that places electronic components on circuit boards. Normally, it would do this one component at a time. But recently the machine has received an update which allows it to place two different components simultaneously. The bottleneck then becomes the component with greater placement time. Now it is far from obvious what strategy the machine should use in order to minimize the total placement time. Sara decides to write an algorithm to determine this strategy.

You have N different types of electronic components. There are f_i copies of the ith type, and the components of this type have a placement time of t_i nanoseconds. The goal is to place all of the components using a sequence of moves. In one move, the machine can take two components of type i and j, where $i \neq j$, and place both of them simultaneously. This takes $\max(t_i, t_j)$ nanoseconds. The machine can also place a single component of type i in one move, which takes t_i nanoseconds.

Calculate the minimum possible time to place all components.

Input

The first line of input contains the integer N ($1 \leq N \leq 1000$).

The following N lines each contain two integers f_i and t_i ($1 \leq f_i \leq 10^4$, $1 \leq t_i \leq 10^9$).

Output

Print one integer, the minimum time to place all components.

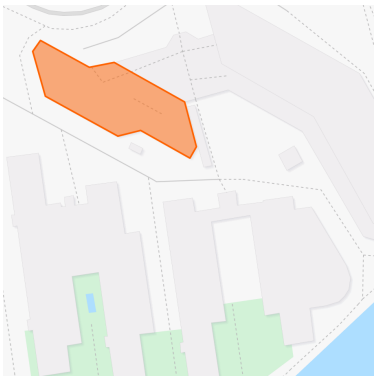
Example

Input	Output
3 2 7 2 1 3 10	31
3 2 10 2 11 2 12	35
4 2 11 7 10 3 5 1 1	72

Problem F. Fence Fee

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

The National Crop Protection Commission (NCPC) is dedicated to supporting local farmers by offering subsidies that are proportional to the area of their crop fields. Each farmer can have several crop fields, each uniquely shaped but geometrically defined as a polygon, bounded by fences that meet only at the corners.



In a classic display of political bureaucracy, and to incentivize well-shaped fields, the NCPC will subsidize each crop field based on the square of its area. This, to reward well-encapsulated fields. They now require a tool that calculates the sum of the squared areas of these polygons to ensure that the subsidies are distributed fairly.

Input

The first line contains an integer F ($3 \leq F \leq 1000$), the number of fence line segments. The next F lines contain four integers each, x_1, y_1, x_2, y_2 ($0 \leq x_1, y_1, x_2, y_2 \leq 1000$), representing a straight-line fence section.

No two fence line segments intersect. Since fencing is both expensive and tedious, you may assume that every fence line segment is necessary and serves to bound fields. All fences farmers have are connected.

In other words, the graph that consists of endpoints and fences is planar, connected, and has no bridges.

Output

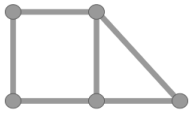
Output a single line representing the sum of the squared areas of all the fields formed by the given fence sections. Your answer will be correct if it has an absolute or relative error of at most 10^{-6}.

Example

Input	Output
5 0 0 0 1 0 1 1 1 0 0 1 0 1 0 2 0 1 1 2 0	2.25
6 0 0 0 1 0 1 1 1 0 0 1 0 1 0 2 0 1 1 2 0 1 0 1 1	1.25

Explanation

The picture represents example 2. The areas of the two fields are 1 and 0.5, so the sum of their squares is $1 + 0.25 = 1.25$.



Problem G. Game: Heroes of Velmar

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

Welcome to the world of *Heroes of Velmar*, the critically acclaimed trading card game developed by *Sidney Games*! After the tremendous success of the physical card game, Sidney Games has decided to take it to the next level and transform it into an immersive video game experience.

As Sidney Games embarks on this ambitious video game project, they seek the expertise of talented developers like you to bring this digital version to life. The challenge lies in coding the algorithm that determines the winner in the virtual battles that unfold between players. The video game will need to retain the same core mechanics as the original card game, where players compete on three distinct locations over six turns, with abilities and power levels shaping the outcomes.

The full rules of the game are listed below. You are given the state of the locations after *Setup* and *Gameplay* have finished and the *End of the Game* has been reached. Sidney Games has tasked you with implementing the *Location Resolution* part of the game rules, including the application of *Special Abilities*, to determine the winner.

The game designers have provided you with images of the cards as well as a JSON file with their specifications.

Heroes of Velmar - Game Rules

Objective

The objective of the two-player card game *Heroes of Velmar* is to win more locations than the opponent over six turns. If there is a tie, the total power level across all locations acts as a tiebreaker.

Setup

1. Each player selects a deck of cards containing heroes with different abilities and power levels.
2. The players shuffle their decks and draw a starting hand of 7 cards each.
3. Designate three distinct locations for the battle. Each location will have its own separate battlefield.

Gameplay

1. Each player draws 1 card from the deck, to their hand, if possible. Maximum cards per hand is 7.
2. Players take turns playing cards from their hand on any of the three locations.
3. On each turn, players have energy equal to the turn number, for example, on Turn 3, they have 3 energy to spend.
4. Each card has a cost that must be paid using energy to play it on a location.
5. Players can play as many cards as they want as long as the following conditions are met:
   (a) They have sufficient energy to spend on the card.
   (b) The location has an empty spot. Each player has 4 available spots per location.

6. The power level of each card represents its strength in the game.
7. When the turn ends, the energy not used is lost.

End of the Game

1. The game ends after six turns, at which point locations are resolved, with each location having a winner or a tie.
2. The player who wins the most locations is the overall winner.
3. If the players win an equal number of locations, the total power level across all locations is used as a tiebreaker to determine the winner.
4. If the total power level across all locations is also tied, then the game results in a tie.

Location Resolution

1. Locations are resolved separately at the end of the game.
2. Compare the total power levels of all cards played by each player at the location after applying special abilities.
3. The player with the higher total power level wins the location.
4. If there is a tie, no one wins the location.

Special Abilities

1. Some cards have special abilities that can affect the game. These abilities trigger at the end of the game, before location resolution.
2. Abilities can buff the card's power level, interact with other cards, or manipulate the game state.
3. Two cards that portray the same character are considered to portray distinct characters for the sake of abilities.

The game *Heroes of Velmar* offers a mix of strategic card play, resource management, and tactical decision-making. Players must choose their heroes wisely, coordinate their plays, and employ their unique abilities to outwit their opponents and emerge victorious in the epic battle for control over Velmar!

Card design

Card Explanation:

- **Top Left Corner:** The energy cost required to play the card on the battlefield.
- **Top Right Corner:** The power level of the card, representing its strength.
- **Text on the Bottom:** The name of the card, identifying its character.
- **Text Underneath:** The card's ability, describing its unique effect during gameplay.

The cards are shown on the next page.

Input

Input consists of six lines representing the state of the three locations after six turns of play. This means that there will be at most 24 cards total listed in the input.

First the left location is described, then the center location is described, and finally the right location is described. Each location is described by two lines, the first of which represents player 1's cards and the second of which represents player 2's cards. Each line lists the number of cards in the line and then the names of the cards played by the player, separated by spaces. There will be at most four cards in each line.

Note that a player may leave a location empty. Each input is guaranteed to be a valid reachable final game state according to the game rules.

Output

Output `"Player 1"` if player 1 won, `"Player 2"` if player 2 won, or `"Tie"` if there was no victor.

Example

Input	Output
3 Shadow Seraphina Ironwood 2 Voidclaw Voidclaw 1 Vexia 0 1 Ranger 0	Player 1
1 Guardian 1 Anvil 2 Seraphina Seraphina 1 Ranger 2 Ironwood Ranger 1 Guardian	Tie
1 Guardian 1 Anvil 2 Seraphina Seraphina 1 Ranger 1 Ironwood 1 Guardian	Player 2


4
6
SHADOW


3
5
GALE


2
4
RANGER


5
7
ANVIL


2
3
VEXIA


6
8
GUARDIAN


5
6
THUNDERHEART
Phalanx - If the location this card is played at has 4 friendly cards, including this one, then its power is doubled. Note that other buffs the card receives are not doubled.


1
2
FROSTWHISPER


1
3
VOIDCLAW


1
3
IRONWOOD


6
4
ZENITH
Centered focus - If this card is played at the center location, +5 power.


1
1
SERAPHINA
Celestial Healing - Seraphina grants +1 power to each friendly card at this location, other than itself. This includes other Seraphina cards.

Problem H. Hotfix

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

Image taken from commons.wikimedia.org

In an earlier contest, contestants had to solve a simple problem. They were given a string and had to print each unique substring of it, along with the number of occurrences it had in the original string. For example `AB` would print `A 1 B 1 AB 1` and `AAA` would print `A 3 AA 2 AAA 1`.

When copying over this problem for reuse in this contest, several mistakes were made. The input constraints were changed significantly, making the problem absolutely impossible! Luckily this was partially cancelled out by the output validator being mangled as well. Now instead of checking for absolute correctness it only requires the number of occurrences of each character in the output to be correct. With a quick hotfix of applying run length encoding to the output, the problem was finally solvable again and the contest could continue on as planned. Right?

Input

The input contains a single string of length at least 1 and at most 10^6. It contains only ASCII upper and lower case characters. This string is then followed by a single newline character.

Output

For each non-whitespace character that appears a non-zero number of times in the output of the problem described above, print it along with its number of occurrences on a single line, separated by a space. Print the lines ordered by ascending values of the ASCII characters.

Example

Input	Output
ABC	1 6 A 3 B 4 C 3
aaaab	1 6 2 1 3 1 4 1 a 20 b 5

Problem I. Infinite Cash

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

Image taken from flickr.com

Svalur Handsome has finally graduated with a degree in computer science, and it couldn't have happened sooner. He has some rather unwise spending habits which he hopes will be more sustainable now that he can get a high paying job as a programmer. He has applied to a few places, and now has a contract in his hands that he could sign and start working almost immediately. But before he takes the offer he wants to figure out how long it could support his spending habits.

At the start of every day Svalur spends half of his remaining money, rounded up. The new job would pay s ISK at the end of every d-th day, starting with the d-th day. He currently has m ISK to spend as well.

Input

The input has three lines, each containing the positive integers s, d, m respectively. They satisfy $1 \leq s, d, m \leq 2^{1000}$. As these payment details are for a computer science job the numbers are all given in binary, naturally.

Output

Print the number of the day that Svalur wants to spend money, but has none. This should naturally also be printed in binary. If he can support his spending habits indefinitely instead print `Infinite money!`.

Example

Input	Output
101110101 1010 10001110101010101	10011
101110101 1000 100011101	Infinite money!
101110101 1010 100011101	1001

Problem J. Jamboree

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

Picture Image by brgfx at Freepik

A group of scouts are preparing to go to a large meeting with other scouts. Their leader Hildeborg, in spirit of the scout motto "be prepared", wants to distribute some useful items among the scouts that they most probably will need on their adventure. The items come in different sizes, so to make this as fair as possible, she wants to make sure that the total size of items carried by any scout is as small as possible. Furthermore, Hildeborg does not want to give more than two items to any scout as she is afraid that it otherwise will be too hard for them to remember to bring everything. Given the sizes of the items, what is the least maximum total size, computed as the sum of items, any scout will have to carry?

Input

The first line of input contains two positive integers N and M ($1 \leq N \leq 2M$, $1 \leq M \leq 100$). N is the number of useful items, and M is the number of scouts. The second line contains N positive integers a_i ($1 \leq a_i \leq 10^7$) giving the sizes of the items.

Output

Print one integer, the smallest total size that any scout has to carry.

Example

Input	Output
3 4 10 10 10	10
5 4 9 12 3 9 10	12

Problem K. Knitting Pattern

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

JÃ¶rmunrekur had found himself with some extra time on his hands, so he decided to try to find a new hobby. After discussing this with some of his relatives, his grandparents lent him a book with knitting guides and knitting patterns.

He wants to start with something big, so he decides to make a sweater. He has also picked out a pattern from the book that he will repeat around the circumference of the sweater. He wants the pattern to be centered and then repeat out towards the back in either direction, but never wants to have less than the full pattern on the sweater. He will not place any patterns that leaves the placements of patterns asymmetric. Now he has to know how much empty space he should leave at the back of the sweater to achieve this.

The empty space that is not covered by the patterns must be a contiguous (possibly empty) section at the back of the sweater.

Image by Linn Bryhn Jacobsen, from commons.wikimedia.org

Input

The input contains two positive integers N, the length of the sweater, and P, the length of the pattern. They satisfy $1 \leq P \leq N \leq 10^{18}$ and they have the same parity, as otherwise the pattern could never be perfectly centered.

Output

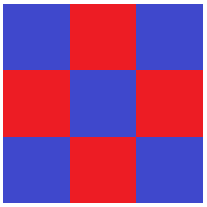
Print a single integer, the amount of empty space left on the back of the sweater.

Example

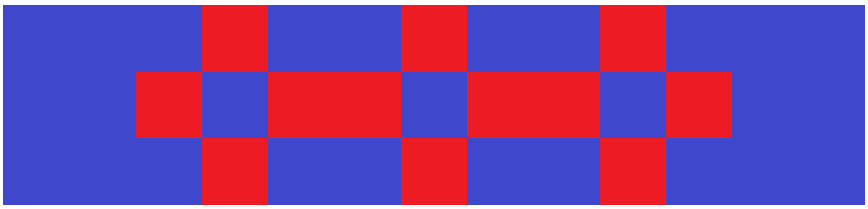
Input	Output
13 3	4
16 4	0

Explanation

In the first example the sweater is 13 loops in circumference. Thus the centered pattern is placed at loops 6, 7 and 8. There's space for another pattern in either direction at loops 3, 4, 5 and 9, 10, 11. There's not enough space to place two more, and a single pattern would make things asymmetric. Thus loops 1, 2, 12 and 13 are empty, so the answer is 4.

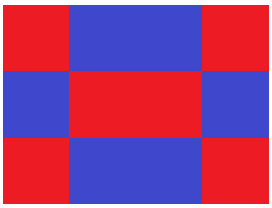


A possible pattern for example 1

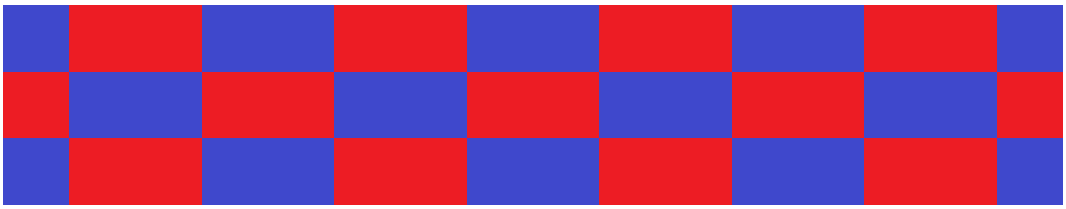


Applying the pattern to the full width for example 1

In the second example the sweater is 16 loops in circumference. The first pattern is placed at loops 7, 8, 9 and 10. Two more are placed at 3, 4, 5, 6 and 11, 12, 13, 14. This leaves 1, 2, 15 and 16, which exactly fits one more pattern that will be perfectly centered at the back of the sweater, creating no asymmetry. Thus that pattern is placed, leaving no empty space.



A possible pattern for example 2



Applying the pattern to the full width for example 2