ICPC SOUTH PACIFIC PRELIMINARIES LEVEL A

SEPTEMBER 3, 2023

Contest Problems

A: A Game of Sets
B: Bacterial Genome
C: Convex Hole
D: Dangerous Monoliths
E: Equitable Dice
F: Floor of Chocolate
G: Genetic Reconstruction
H: Home Invasion
I : Infernal Internal Integer Introspection
J : Jay, Jay's son ja$on, and their Socks
K: Killer Code Coverage
L: Long Distance Calling

This contest contains twelve problems. Good luck.

For problems that state "*Your answer should have an absolute or relative error of less than* 10^{-9}", your answer, x, will be compared to the correct answer, y. If $|x-y| < 10^{-9}$ or $\frac{|x-y|}{|y|} < 10^{-9}$, then your answer will be considered correct.

Definition 1

For problems that ask for a result modulo m:
If the correct answer to the problem is the integer b, then you should display the unique value a such that:

- $0 \leq a < m$
  and
- $(a-b)$ is a multiple of m.

Definition 2

A string $s_1 s_2 \cdots s_n$ is lexicographically smaller than $t_1 t_2 \cdots t_\ell$ if

- there exists $k \leq \min(n, \ell)$ such that $s_i = t_i$ for all $1 \leq i < k$ and $s_k < t_k$
  or
- $s_i = t_i$ for all $1 \leq i \leq \min(n, \ell)$ and $n < \ell$.

Definition 3

- Uppercase letters are the uppercase English letters ($A, B, \ldots, Z$).
- Lowercase letters are the lowercase English letters ($a, b, \ldots, z$).

Definition 4

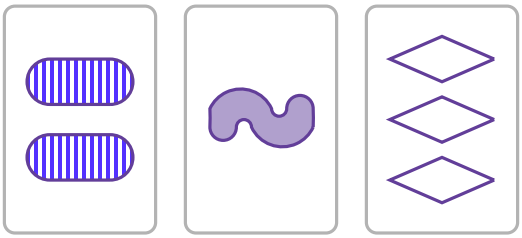
Unless otherwise specified, the distance between two points (x_0, y_0) and (x_1, y_1) is defined as its Euclidean distance:

$$\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}.$$

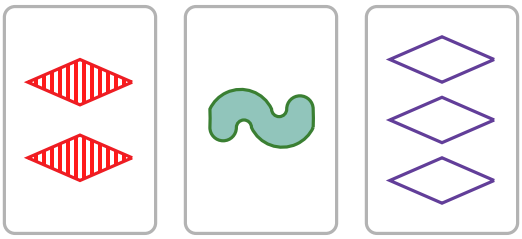
Problem A
A Game of Sets
Time limit: 1 second

Set is a very popular card game amongst the math community. The game of Set is played with a deck of specialized cards. Each card is described by four properties: number, shape, colour, and shading. Each property has three options: the number is one of (1, 2, or 3), the shape is one of (diamond, oval, or squiggle), the colour is one of (green, purple, or red), and the shading is one of (empty, solid, or striped). This gives a total of 81 distinct cards (3^4).

Three different cards form a *set* if each property is either all the same or all different on the three cards. For example, the following is a set since the numbers are different (2, 1, 3), the shapes are different (oval, squiggle, diamond), the colours are the same (all purple), and the shadings are different (striped, solid, empty):



However, this is not a set since the shapes do not satisfy the requirement (two cards with diamonds and one with squiggle):



Your hand contains several cards. Determine if there are three cards in your hand that form a set.

Input

The first line of input consists of a single integer n ($3 \leq n \leq 81$), which is the number of cards in your hand.

The next n lines describe the cards in your hand. Each card has four strings describing its properties: its number (`one`, `two`, or `three`), its shape (`diamond`, `oval`, or `squiggle`), its colour (`green`, `purple`, or `red`), and its shading (`empty`, `solid`, or `striped`).

No two cards are identical.

Output

If there are no sets in your hand, display 0. Otherwise, display the three indices of the three distinct cards that form a set (the first card in the input is card 1, the second card in the input is card 2, and so on). You may display the indices in any order. If there are multiple solutions, any will be accepted.

Sample Input 1	Sample Output 1
3 two oval purple striped one squiggle purple solid three diamond purple empty	1 2 3

Sample Input 2	Sample Output 2
3 two diamond red striped one squiggle green solid three diamond purple empty	0

Sample Input 3	Sample Output 3
5 one squiggle purple solid two diamond red striped one squiggle green empty two oval purple empty two squiggle green solid	5 2 4

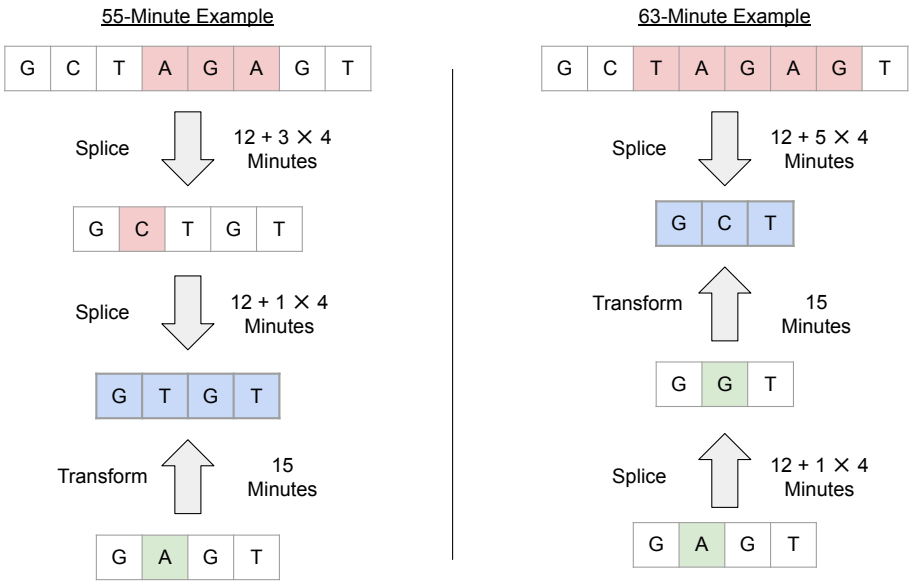
Problem B
Bacterial Genome
Time limit: 7 seconds

A genetic sequence is a string from an alphabet of four letters: 'A', 'C', 'G', and 'T', representing the four types of base in the sequence. We are considering genetic sequences from a new type of organism called *Eyseepeecee* which can be mutated in different ways to other organisms. An Eyseepeecee genetic sequence can be mutated in two ways: splicing and transformation.

A splice of a sequence is the sequence remaining after removing exactly one non-empty contiguous substring of letters. For example, AGCT, ACT, AC, and T are all splices of ACGCT, while ACAT, ACGGT, CG, and ACGCT are not. The time taken to splice a sequence that removes k letters is $S + kC$ minutes.

A transformation of a sequence is the sequence after changing exactly one letter to some other letter. For example, TCGCT, ATGCT, ACACT, and ACGCA are all transformations of ACGCT, while AAGAT, CAGCT, TACGCT, and ACGCT are not. The time taken to transform one letter is T minutes.

Given two genomes, you wish to mutate them zero or more times so that they become the same. For example, say $S = 12$, $C = 4$, and $T = 15$. The two sequences GCTAGAGT and GAGT can be mutated into GTGT in 55 minutes or into GCT in 63 minutes using the mutations in the image.



The *similarity score* between two genomes is the minimum amount of time needed to mutate the genomes so that they become the same. It does not matter what the genomes end up as, so long as they are the same. Ending up as empty strings is valid. In the example above, the minimum amount of time is to remove the first four letters of GCTAGAGT: a splice that takes 28 minutes. This gives a similarity score of 28.

Given two genetic sequences, what is their similarity score?

Input

The first line of input contains three integers, S ($1 \leq S \leq 10^9$), C ($1 \leq C \leq 10^9$), and T ($1 \leq T \leq 10^9$). The time taken for a splice that removes k letters is $S + kC$ minutes and the time taken for a transformation is T minutes.

The next line of input contains a string representing the first genetic sequence. The next line of input contains a string representing the second genetic sequence. Both genetic sequences will comprise only the characters A, C, G, or T. The lengths of both sequences will be in the inclusive range between 1 and 3000.

Output

Display the similarity score between the two genetic sequences.

Sample Input 1	Sample Output 1
12 4 15 GCTAGAGT GAGT	28

Sample Input 2	Sample Output 2
12 2 4 GCCCCCGT ACCCA	26

Sample Input 3	Sample Output 3
3 1 1 GCCCCCAT ACCCA	8

Sample Input 4	Sample Output 4
2 1 19 CC AAAAA	11

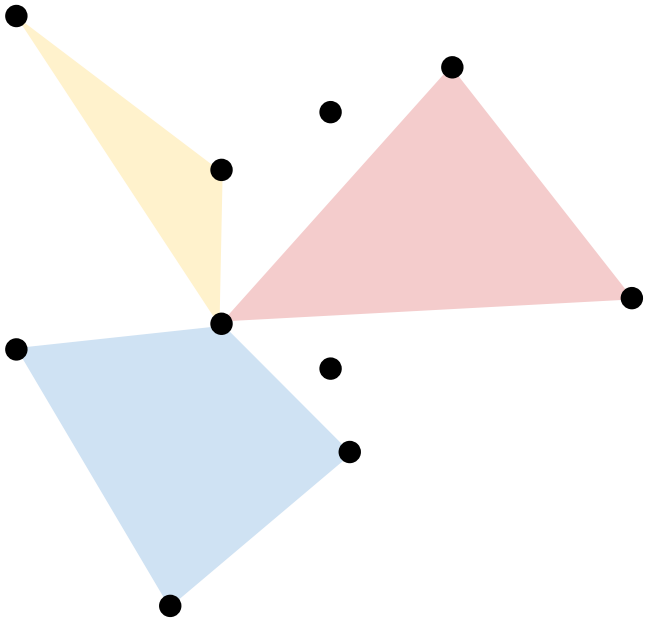
Sample Input 5	Sample Output 5
3 5 1 ACCCA GCCCACGT	19

Sample Input 6	Sample Output 6
4 1 5 AAT AAT	0

Problem C
Convex Hole
Time limit: 10 seconds

The *convex hull* of a set of points is the smallest convex polygon that contains all of the points. A polygon is convex if all of its internal angles are less than or equal to 180 degrees and its area is non-zero.

A *convex hole* of a set of points is a convex polygon, using points from the set as vertices, that does not contain any of the points from the set, except on its boundary.



Given a set of N (not all collinear) points, find the largest convex hole by area.

Input

The first line of the input contains a single integer N ($3 \leq N \leq 75$), which is the number of points in the set.

The following N lines each contain two integers X ($-10^6 \leq X \leq 10^6$) and Y ($-10^6 \leq Y \leq 10^6$), which are the coordinates of each point in the set.

No two points are the same, and not all points are collinear.

Output

Display a single integer showing the area of the largest convex hole, multiplied by two. It can be shown that this value is always an integer.

Sample Input 1	Sample Output 1
3 -1 -1 1 -1 1 1	4

Sample Input 2	Sample Output 2
9 0 0 0 1 0 2 1 0 1 1 1 2 2 0 2 1 2 2	4

Sample Input 3	Sample Output 3
4 1 3 2 3 3 3 3 2	2

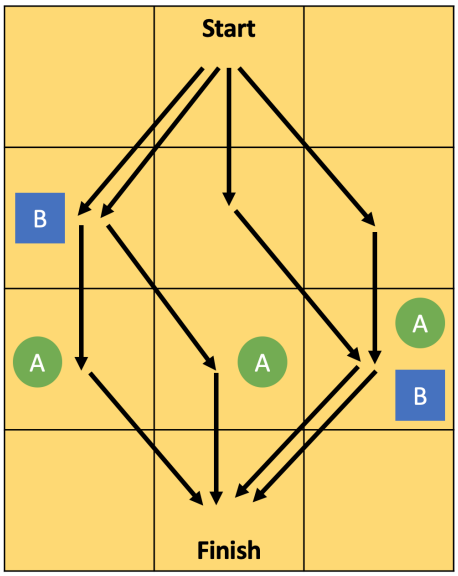
Problem D
Dangerous Monoliths
Time limit: 7 seconds

You are visiting a vast desert filled with strange monoliths. The desert is a grid with N rows (numbered 1 to N) and M columns (numbered 1 to M), where each grid cell has 0 or more monoliths in it. There are K different types of monoliths (represented by the first K letters of the alphabet), and each cell has at most one monolith of each type. However, the same type of monolith could appear in several cells.

You are planning a path through this grid. Your path must obey the following conditions:

- You start in the top row at cell $(1, S)$.
- You finish in the bottom row at cell (N, F).
- If you are at cell (i, j), then the next cell you visit must be one of the following:
  - $(i + 1, j - 1)$,
  - $(i + 1, j)$,
  - $(i + 1, j + 1)$.
- You cannot exit the desert during the path, and so every cell (i, j) you visit must satisfy $1 \leq i \leq N$ and $1 \leq j \leq M$.

For each cell you visit on your path, you will see all of the monoliths in that cell. You want to see every type of monolith, however, you have heard that seeing more than one monolith of the same type could lead to dangerous consequences. Therefore, you want to see every type of monolith **exactly once** along your path.



Sample Input 1. The four different paths are shown.

How many different paths are there that see every type of monolith exactly once? Since this number could be very large, compute it modulo $1\,000\,000\,007$.

Input

The first line of input contains the five integers N ($2 \leq N \leq 26$), which is the number of rows, M ($2 \leq M \leq 26$), which is the number of columns, K ($1 \leq K \leq 26$), which is the number of different types of monoliths, S ($1 \leq S \leq M$), which is the starting column, and F ($1 \leq F \leq M$), which is the finishing column.

The next N lines describe the desert. Each of these lines contains M strings. The j-th string on the i-th line represents cell (i, j). If the string is '$-$' then the cell has no monoliths. Otherwise, the string contains uppercase letters representing the monoliths found in this cell, in alphabetical order.

Output

Display the number of paths that see every monolith exactly once, modulo $1\,000\,000\,007$.

Sample Input 1

```
4 3 2 2 2
- - -
B - -
A A AB
- - -
```

Sample Output 1

```
4
```

Sample Input 2

```
6 4 6 2 3
A A A A
B B B B
C C C C
D D D D
E E E E
F F F F
```

Sample Output 2

```
43
```

Sample Input 3

```
5 3 3 1 2
A B C
A - BC
B AC B
C - B
A C AB
```

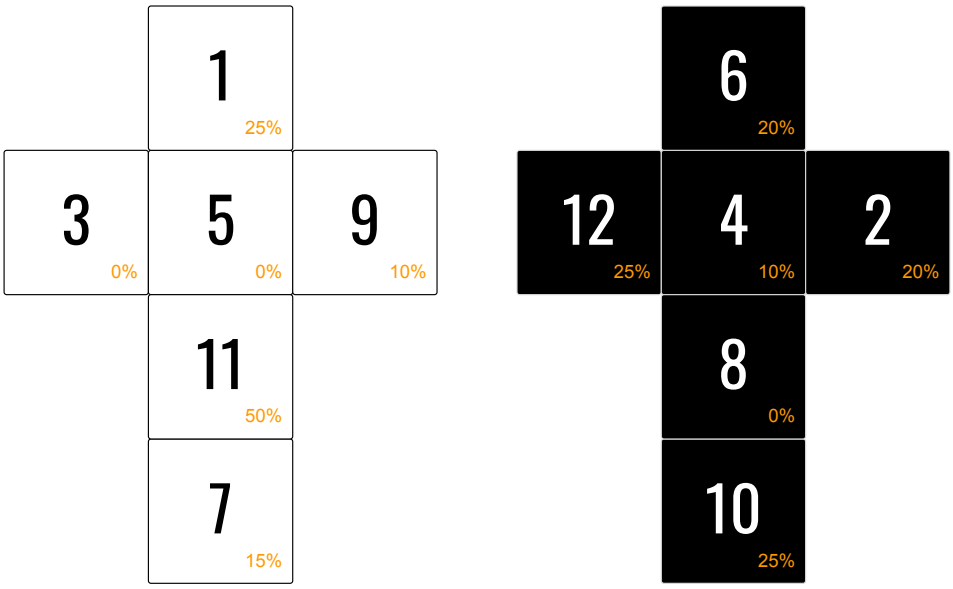
Sample Output 3

```
2
```

Problem E
Equitable Dice
Time limit: 1 second

In my favourite board game, making the first move of the game is extremely important. Each player in the game has a 6-sided die. The numbers on the faces of the dice are all different (there are n dice and $6n$ distinct numbers on the faces). All of the n players will roll their die, and the player with the highest rolled number goes first.

A set of dice are *equitable* if when every die in the set is rolled, every die has the same probability of being the die that rolls the highest value. These dice are special—the probability of rolling each side does not need to be the same. For example, the following dice are equitable:



Note that some die faces may have a 0% probability of being rolled. The players have shown up for game night with their dice, and you must now adjust the probabilities of each die so that all of the dice are equitable. You may not change the values of the faces, just the probability of each face being rolled. The probabilities on a particular die are *valid* if they sum to 100%.

Given the values on the faces of the dice, find probabilities so that each of the dice are valid and the set of dice is equitable.

Input

The first line of input contains a single integer n ($2 \leq n \leq 100$), which is the number of players.

The next n lines describe the dice. Each line contains 6 integers $f_1, f_2, \ldots, f_6$ ($1 \leq f_i \leq 6n$), which are the values of the die's faces.

Each possible die face value (between 1 and $6n$, inclusive) appears on exactly one die face.

Output

If there is no weighting possible so that the dice are valid and equitable, display `IMPOSSIBLE`.

Otherwise, display `POSSIBLE`, then $6n$ probabilities. The $6n$ probabilities must be between 0 and 1, inclusive, and represent the probabilities of the faces, in the order they appeared in the input.

If there are multiple answers, any will be accepted. Your answer will be considered correct if (1) each die is valid and (2) the dice are equitable.

A die will be considered valid if the sum of the 6 probabilities is within 10^{-6} of 1. The set of dice will be considered equitable if the probability of each die winning is within 10^{-6} of $\frac{1}{n}$.

Sample Input 1

```
2
1 3 5 7 9 11
12 10 8 6 4 2
```

Sample Output 1

```
POSSIBLE
0.25 0 0 0.15 0.1 0.5
0.25 0.25 0 0.2 0.1 0.2
```

Sample Input 2

```
3
1 2 3 4 5 6
7 8 9 10 11 12
13 14 15 16 17 18
```

Sample Output 2

```
IMPOSSIBLE
```

Sample Input 3

```
3
8 6 18 9 17 7
15 16 1 3 14 2
4 11 10 5 13 12
```

Sample Output 3

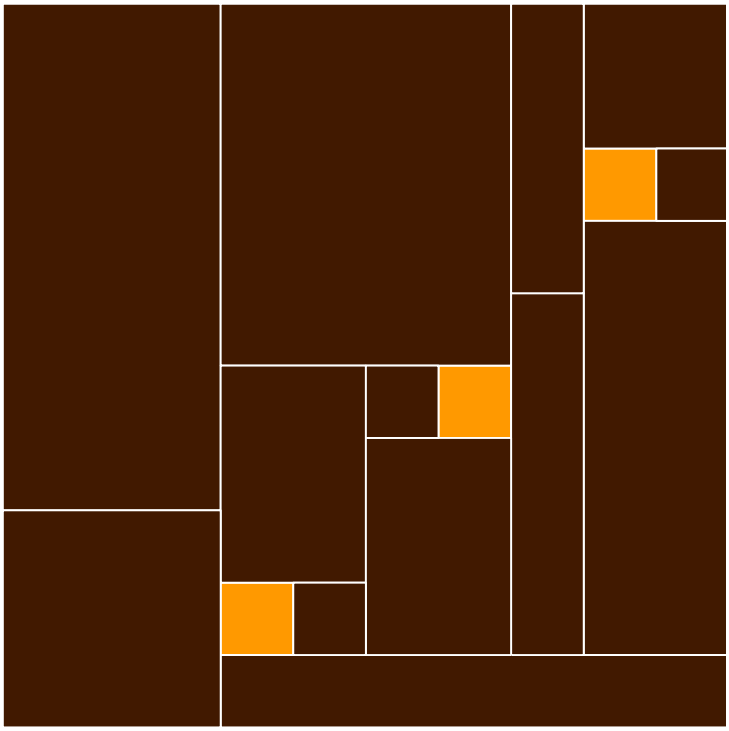
```
POSSIBLE
0.5 0.1 0.05 0.1 0.2 0.05
0.2222222222 0.05555555556 0 0.11111111111 0.1666666667 0.4444444444
0.1 0.55 0.05 0.1 0.15 0.05
```

Problem F
Floor of Chocolate
Time limit: 2 seconds

The owner of the world's biggest chocolate factory is remodeling and they have tasked you with handling the flooring for one of the rooms! The room is $N \times N$ metres in size and has at most 1234 support columns, each 1×1 metre in size, spread out around the room. The flooring will be made of real chocolate.

Luckily, Chocolate Flooring Inc. has everything you need to install a chocolate floor by yourself. They sell rectangular chocolate floor tiles in all integer-metre sizes. Each chocolate tile costs only \$1 (no matter the dimensions)! For example, you could get a 1×1 metre tile, a 2×5 metre tile, and six 4×4 metre tiles for only \$8! The tiles are easy to place on the floor but have been specially treated to not melt and thus cannot be cut.

Your goal is to cover the entire floor with tiles, except where the support columns are. The placed tiles may not overlap with each other and must be placed completely inside the room. The tiles may not be placed on top of the support columns. The tiles, support columns, and edges of the room may touch, but not overlap. A covering is *valid* if it satisfies these constraints.



You have been given a budget of \$4321 so you can buy at most 4321 tiles for the floor. Find a valid covering. Your covering need not use the minimum number of tiles.

Input

The first line of input contains two integers N ($50 \leq N \leq 10^9$), which is the side length in metres of the room, and K ($1 \leq K \leq 1234$), which is the number of support columns in the room. The room's four corners are located at $\{(0,0),(0,N),(N,N),(N,0)\}$.

The next K lines describe the locations of the support columns. Each line contains two integers X ($0 \leq X < N$) and Y ($0 \leq Y < N$), representing that a support column is located at $\{(X,Y),(X,Y+1),(X+1,Y+1),(X+1,Y)\}$.

No two support columns are at the same location.

Output

Display T ($1 \leq T \leq 4321$), which is the number of floor tiles used in the valid covering of the floor.

Then, display T lines describing the floor tiles' placements in the valid covering. Each line should contain four integers x ($0 \leq x < N$), y ($0 \leq y < N$), w ($w \geq 1$ and $x + w \leq N$), and h ($h \geq 1$ and $y + h \leq N$), representing this floor tile's four corners are located at $\{(x, y), (x, y + h), (x + w, y + h), (x + w, y)\}$.

If there are multiple solutions, any of them will be accepted.

Sample Input 1

```
50 1
10 15
```

Sample Output 1

```
4
0 0 10 50
10 0 40 15
10 16 1 34
11 15 39 35
```

Sample Input 2

```
101 5
0 0
100 100
0 100
100 0
50 50
```

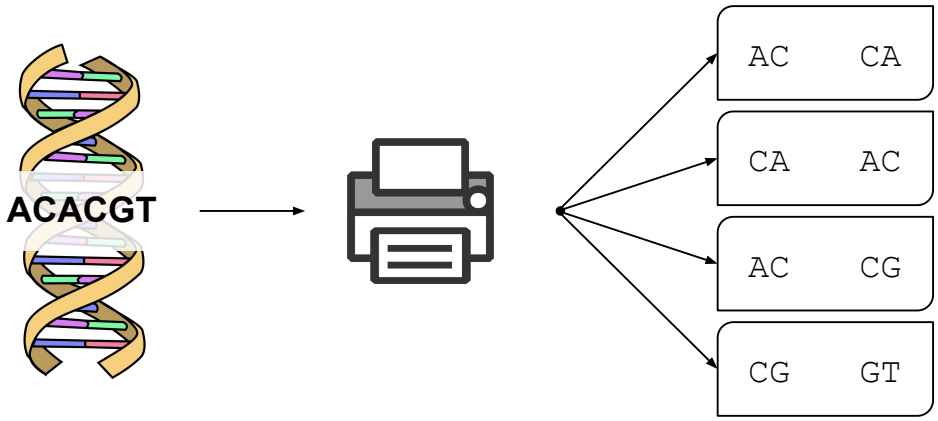
Sample Output 2

```
6
1 0 99 1
1 100 99 1
0 1 50 99
51 1 50 99
50 1 1 49
50 51 1 49
```

Problem G
Genetic Reconstruction
Time limit: 1 second

A team of scientists has been doing deep sequencing of DNA and they have asked you for help. The sequencing machine works on a piece of DNA that can be represented as a string of length N consisting of only the letters from `A` to `Z`.

The machine works by creating every substring of length K from the original string, resulting in $N - K + 1$ substrings. Then each pair of consecutive substrings is printed onto a piece of paper. For example, if the original DNA string was `ACACGT` and $K = 2$, then 4 papers would be printed out containing (`AC`, `CA`), (`CA`, `AC`), (`AC`, `CG`), and (`CG`, `GT`).



Unfortunately, one of the scientists dropped the stack of papers that the DNA sequencing machine outputted! Given pairs of strings (each on a separate piece of paper in no particular order), can you reconstruct a possible DNA sequence?

Input

The first line of input contains two integers N ($3 \leq N \leq 1000$), which is the original length of the DNA sequence being sequenced, and K ($2 \leq K \leq \min(N - 1, 10)$), which is the length of each substring.

The next $N - K$ lines describe the papers the machine outputted (in no particular order). Each line contains two strings, A_i and B_i, of length K consisting of only the letters from `A` to `Z`. This means that A_i and B_i were consecutive substrings in the original DNA sequence. The last $K - 1$ characters of A_i is a prefix of B_i.

It is guaranteed that there is at least one DNA sequence that is consistent with all the papers provided.

Output

Display a DNA sequence of length N that is consistent with the machine's output.

If there are multiple solutions, any will be accepted.

Sample Input 1	Sample Output 1
6 2 CA AC CG GT AC CA AC CG	ACACGT

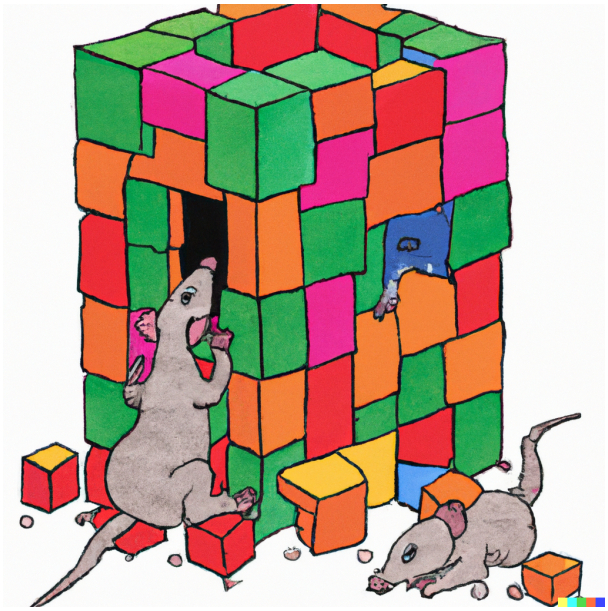
Sample Input 2	Sample Output 2
7 3 GAT ATA AGG GGA ATA TAG GGA GAT	AGGATAG

Problem H
Home Invasion
Time limit: 3 seconds

Alice lives in a house built entirely of cubic bricks, arranged in an $N \times N \times N$ grid. Each brick occupies a unit cube of space. However, rats have managed to infiltrate her house and are constantly raiding her snacks.

To safeguard her snacks, Alice needs to determine the minimum number of new bricks required to block all the paths from the rats to any of the snacks within the house. Each brick can be placed in any empty space within the grid.

Each of the N^3 grid cells contains one of the following: (1) an existing brick that cannot be moved, (2) a rat, (3) a snack, or (4) an empty space where a new brick can be placed.



The rats can move around the house in the following way:

1. Rats can move to an adjacent cell in any of the six directions: up, down, left, right, forward, or backward, as long as the target cell is not occupied by a brick.
2. If a rat is adjacent to a snack, it can move to the snack's cell and devour it.
3. If a rat is adjacent to an empty cell, it can move to that cell.
4. If a rat is located on any of the edges of the house, it can exit the house and enter from any empty space on any edge of the house. If there is a snack on some edge of the house, it may be devoured from the outside.

Before the rats wake up, Alice has the opportunity to place as many bricks as she wants in the empty spaces within the house. Once Alice is done placing bricks, the rats start moving, and Alice cannot place any more bricks.

Help Alice determine the minimum number of new bricks she needs to add to ensure no rat currently in the house can devour any of her snacks using any sequence of valid moves.

Input

The first line of input consists of a single integer, N ($2 \leq N \leq 10$), which is the size of the grid.

This is followed by N blocks. Each block is an $N \times N$ grid of characters. The characters represent the elements in the grid. The k-th character in the j-th line of the i-th block in the input describes cell (i, j, k) of Alice's house. Each character is one of `#` (existing brick), `R` (rat), `S` (snack), or `.` (empty space).

There is at least one rat and at least one snack in the house.

Output

Display the minimum number of new bricks required to block all the paths from the rats to any of the snacks. If it is not possible to block all the paths, display -1.

Sample Input 1	Sample Output 1
4 #### #.## #### #### #### #R.# #### #### #### ##.# .S.# #### #### #### #.## ####	2

Sample Input 2	Sample Output 2
2 #R ## ## S#	-1

Problem I

Infernal Internal Integer Introspection

Time limit: 8 seconds

Jay is playing a game of Introspection involving binary numbers. A binary number consists of only 0s and 1s, and the first digit is always 1. Jay is introspecting the *internal numbers* of a binary number. An internal number is a substring of a number's digits that is also a binary number. For example, 1011 has 5 internal numbers: 1, 10, 101, 1011, and 11.

Jay has a single *blessed* binary number, X. Every internal number of X is also blessed. All other numbers are *cursed*. Jay wants to find a binary number that is cursed so he can curse people. Because he is lazy, he doesn't want it to be too long to type out, so he wants the shortest (by number of binary digits) positive cursed binary number. If there are multiple cursed binary numbers of the same length, Jay wants the one that represents the smallest value. Help him find it, please!

Input

The input consists of a single line containing a single binary number X ($1 \leq X < 2^{1\,000\,000}$). That is, the number of binary digits in X is at most $1\,000\,000$.

Output

Display the smallest positive cursed binary number.

Sample Input 1	Sample Output 1
10	11

Sample Input 2	Sample Output 2
1	10

Sample Input 3	Sample Output 3
1011	100

This page is intentionally left (almost) blank.

Problem J

Jay, Jay's son ja$on, and their Socks

Time limit: 2 seconds

Jay and Jay's son ja$on have a big problem! It's their socks, which they need to keep clean. So, they have decided to keep track of the *wear score* for each of their socks. The wear score is an integer that is initially zero. Whenever someone wears a sock, the wear score is increased by one. Whenever a sock is washed, its wear score is decreased by one.

There's only one problem: Jay and Jay's son ja$on are very forgetful and their socks are worn by everyone in their family. They often forget to keep track of the wear score of their socks. Jay and Jay's son ja$on want to know whenever the wear score of a sock changes, unless they are sleeping, in which case they want the updates when they wake up.

Jay and Jay's son ja$on have compiled a list of N events that they want to get updates on. Each event is either a *wear event*, *wash event*, *sleep event*, or *wake event*. The events take the following form:

- `WEARSOCK` s: A sock with integer ID s wear score has increased by 1.
- `WASHSOCK` s: A sock with integer ID s wear score has decreased by 1.
- `SLEEP`: Jay and Jay's son ja$on are going to sleep. Don't wake them with updates!
- `WAKE`: Jay and Jay's son ja$on are waking up. They want wear score updates!

Whenever a sock's wear score changes and Jay and Jay's son ja$on are awake, print out the new wear score of that sock in the form: `SOCK` s k, where s is the ID of the sock and k is the new wear score of the sock. If Jay and Jay's son ja$on are sleeping, don't print anything. When they wake up, print out all the IDs and wear scores of the socks that have wear scores that are different from when they went to sleep, in integer order of the sock IDs.

For example, if we have this series of events:

```
WEARSOCK 1
WEARSOCK 5
SLEEP
WASHSOCK 1
WASHSOCK 5
WEARSOCK 5
WAKE
```

Then we first tell Jay and Jay's son ja$on:

```
SOCK 1 1
SOCK 5 1
```

Then Jay and Jay's son ja$on go to sleep. When they wake up, we tell them:

```
SOCK 1 0
```

Note that sock 5 didn't change its wear score overall, so we don't tell Jay and Jay's son ja$on about it.

Input

The first line of input contains an integer N ($1 \leq N \leq 1000$), which is the number of queries to follow.

The following N lines each contain a query of the form described above. Each sock ID s satisfies $1 \leq s \leq 10^9$. It is guaranteed that Jay and Jay's son ja$on will never go to sleep while they are sleeping or wake up while they are awake.

Output

Display each update to the wear score of a sock in the format described above, using one line for each update.

Sample Input 1

```
6
WASHSOCK 1
WASHSOCK 9
SLEEP
WASHSOCK 1
WASHSOCK 1
WASHSOCK 6
```

Sample Output 1

```
SOCK 1 -1
SOCK 9 -1
```

Sample Input 2

```
6
SLEEP
WEARSOCK 5
WEARSOCK 1
WEARSOCK 9
WASHSOCK 1
WAKE
```

Sample Output 2

```
SOCK 5 1
SOCK 9 1
```

Sample Input 3

```
4
SLEEP
WEARSOCK 1
WASHSOCK 1
WAKE
```

Sample Output 3

Sample Input 4

```
3
SLEEP
WEARSOCK 1
WAKE
```

Sample Output 4

```
SOCK 1 1
```

Sample Input 5

```
1
WEARSOCK 1
```

Sample Output 5

```
SOCK 1 1
```

Problem K
Killer Code Coverage
Time limit: 1 second

Any good software engineer knows that writing unit tests for your code is important. When running a unit test, a line coverage tool tracks which lines of code are exercised during the test. Consider the following functions:

```
def f(x: int):
  if x >= 3:
    print("ICPC")
  else:
    print("Qualifiers")
```

```
def g(x: int):
  if x >= 3:
    if x % 2 == 0:
      print("South")
    else:
      print("Pacific")

    if x > 10:
      print("Programming")
    else:
      print("Contest")
  else:
    print("2023")
```

When running `f(0)`, the `Qualifiers` line is exercised and when running `f(10)`, the `ICPC` line is exercised. Note that the line coverage tool does not track the lines containing the if or else statements. With just two test cases (`f(0)` and `f(10)`), all lines of code have been exercised by some test. It is impossible to exercise all lines of code in `g` with two test cases. However, three cases is enough (`g(0)`, `g(7)`, `g(50)`). A function is *fully tested* if every line of code has been exercised.

The stripped signature of a function is a list of `if`, `else`, and `endif` statements with the conditions removed. For example, the stripped signature of `f` is [`if`, `else`, `endif`] and the stripped version of `g` is [`if`, `if`, `else`, `endif`, `if`, `else`, `endif`, `else`, `endif`]. In this language, every if-statement has an else-statement, and there is at least one line of code in every block (there is at least one line of code inside every if-block and inside every else-block).

Two functions may have the same stripped signature. For example, `g1` has the same signature as `g` above. Note that it is impossible to exercise the `A` line in `g1` (since $x \geq 10$ and $x < 10$ is impossible), so `g1` cannot be fully tested no matter how many test cases are used.

```
def g1(x: int):
  if x >= 10:
    if x < 10:
      print("A")
    else:
      print("B")

    if x > 20:
      print("C")
    else:
      print("D")
  else:
    print("E")
```

⟶

```
IF
IF
ELSE
ENDIF
IF
ELSE
ENDIF
ELSE
ENDIF
```

The *complexity* of a stripped signature is c if there exists some function with this stripped signature that can be fully tested using c test cases and there is no function with this stripped signature that can be fully tested using fewer than c test cases.

For example, even though `g1` cannot be fully tested, `g` has the same stripped signature and can be fully tested using 3 test cases. There is no function with the same stripped signature as `g1` that can be tested using fewer than 3 test cases, so the complexity of the stripped signature of `g1` is 3.

Given the stripped signature of a function, what is its complexity?

Input

The first line of input contains a single integer n ($3 \leq n \leq 99$), which is the number of statements in the stripped signature.

The next n lines describe the stripped signature. Each line contains a single string s ($s \in \{\texttt{if}, \texttt{else}, \texttt{endif}\}$), describing a statement.

The signature is guaranteed to be valid: every if-statement has a corresponding else and endif, nested at the appropriate level. For example, [`if`, `if`, `else`, `else`] is invalid and will not appear.

Output

Display the complexity of the stripped signature.

Sample Input 1

```
3
if
else
endif
```

Sample Output 1

```
2
```

Sample Input 2

```
9
if
if
else
endif
if
else
endif
else
endif
```

Sample Output 2

```
3
```

Sample Input 3

```
9
if
else
if
else
if
else
endif
endif
endif
```

Sample Output 3

```
4
```

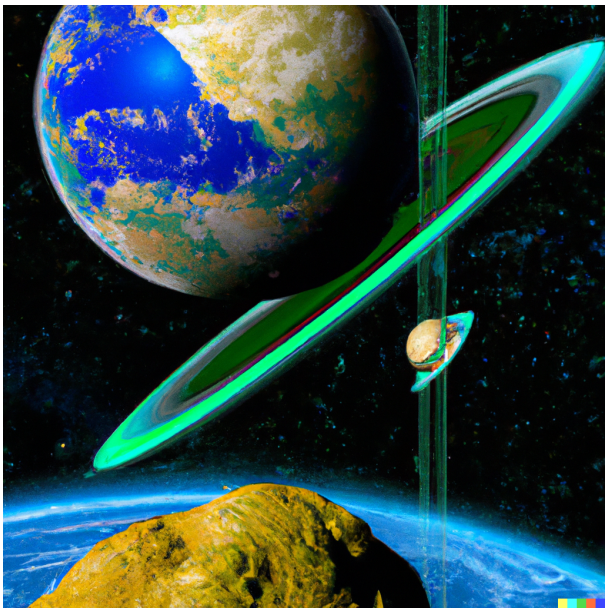
Problem L
Long Distance Calling
Time limit: 1 second

The people of the planets Earth and Tatooine have made first contact via interplanetary video communication. The people of the two planets are quickly becoming friends, but it's a real hassle to schedule video calls between the two planets.

This is because Earth and Tatooine have different day/night cycles, and their days have different durations. A day on Earth might equal 2.5 days, 0.9 days, or 1.8 days on Tatooine.

Due to science fiction magic, real-time video calls are possible between Earth and Tatooine. There is no latency or time dilation. Also, planets Earth and Tatooine both use minutes in their systems for measuring time, and minutes are exactly the same duration on both planets. This coincidence makes the people of Earth and Tatooine very happy. Their friendship is meant to be.

A day on Earth can be perfectly divided into minutes. A day on Tatooine can also be perfectly divided into minutes. However, a day on Earth and a day on Tatooine may have different durations. Similarly, an hour on Earth and an hour on Tatooine can both be perfectly divided into minutes but may have different durations.



Days on Earth are 24 (Earth-)hours long, and hours on Earth are 60 minutes long. Days on Tatooine are X (Tatooine-)hours long, and hours on Tatooine are Y minutes long.

To commemorate the occasion of Earth and Tatooine making first contact, both planets decided to get rid of their old calendars. They now label days by counting the number of days passed since first contact. There are no months or years. Day 0 is the day of first contact.

While first contact occurred on day 0 on both planets, it may have occurred at different times. Of course, it's at the same moment in time; but, the clocks show different times on different planets.

On Earth, first contact happened at H_E (Earth-)hours and M_E minutes after midnight, on day 0. On Tatooine, first contact happened at H_T (Tatooine-)hours and M_T minutes after midnight, on day 0.

People on Earth and Tatooine want to schedule a video call. Given a day, hour, and minute on Earth, what is the corresponding day, hour, and minute on Tatooine?

Input

The first line of input contains two integers, X ($1 \leq X \leq 100$) and Y ($1 \leq Y \leq 100$), which represent the number of (Tatooine-)hours in a day on Tatooine and the number of minutes in a (Tatooine-)hour.

The second line describes the time of first contact on Earth on day 0. This line contains two integers, H_E ($0 \leq H_E < 24$) and M_E ($0 \leq M_E < 60$), which are the number of (Earth-)hours and minutes after midnight.

The third line describes the time of first contact on Tatooine on day 0. This line contains two integers, H_T ($0 \leq H_T < X$) and M_T ($0 \leq M_T < Y$), which are the number of (Tatooine-)hours and minutes after midnight.

The final line of input contains three integers, d ($0 \leq d \leq 1\,000$), h ($0 \leq h < 24$), and m ($0 \leq m < 60$), which are the day, hour, and minute of the video call on Earth. If the video call is on day 0, then it happens strictly after first contact.

Output

Display the day, hour, and minute of the video call on Tatooine.

Sample Input 1

```
19 67
10 45
12 30
23 11 30
```

Sample Output 1

```
26 13 30
```

Sample Input 2

```
23 61
12 30
15 56
21 15 0
```

Sample Output 2

```
22 8 7
```

Sample Input 3

```
5 10
13 10
0 9
2 1 0
```

Sample Output 3

```
43 0 9
```