Constellation

Problem Description

Three characters { #, *, . } represents a constellation of stars and galaxies in space. Each galaxy is demarcated by # characters. There can be one or many stars in a given galaxy. Stars can only be in shape of vowels { A, E, I, O, U } . A collection of * in the shape of the vowels is a star. A star is contained in a 3x3 block. Stars cannot be overlapping. The dot(.) character denotes empty space.

Given 3xN matrix comprising of { #, *, . } character, find the galaxy and stars within them.

Note: Please pay attention to how vowel ***A*** is denoted in a 3x3 block in the examples section below.

Constraints

3 <= N <= 10^5

Input

Input consists of single integer N denoting number of columns.

Output

Output contains vowels (stars) in order of their occurrence within the given galaxy. Galaxy itself is represented by # character.

Time Limit

1

Examples

Example 1

Input

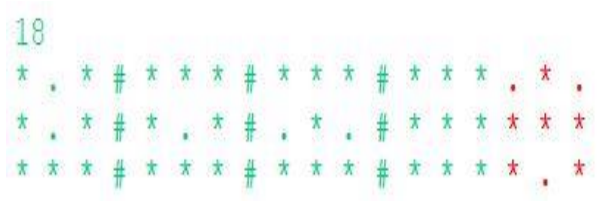
```
18

* . * # * * * # * * * # * * * . * .

* . * # * . * # . * . # * * * * * *

* * * # * * * # * * * # * * * * . *
```

Output

U#O#I#EA

Explanation

As it can be seen that the stars make the image of the alphabets U, O, I, E and A respectively.



```
18
* . * # * * * # * * * # * * * . * .
* . * # * . * # . * . # * * * * * *
* * * # * * * # * * * # * * * * . *
```

Example 2

Input

12

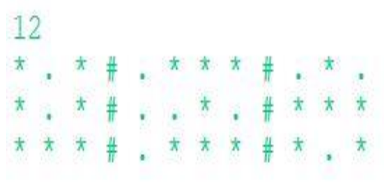
```
* . * # . * * * # . * .
* . * # . . * . # * * *
* * * # . * * * # * . *
```

Output

U#I#A

Explanation

As it can be seen that the stars make the image of the alphabet U, I and A.



```
12
* . * # . * * * # . * .
* . * # . . * . # * * *
* * * # . * * * # * . *
```

Diet Plan

Problem Description

Arnold is planning to follow a diet suggested by his Nutritionist. The Nutritionist prescribed him the total protein, carbohydrates and fats, he should take daily. Arnold searches on an online food store and makes a list of protein, carbohydrates and fats contained in a single unit of various food items.

His target is to have the maximum protein, carbohydrates and fats in his diet without exceeding the prescribed limit. He also wants to have as much diverse food items as much as possible. That is, he does not want to have many units of one food item and 0 of others. Multiple combinations of 'units of food items' are possible to achieve the target. Mathematically speaking, diversity is more if the difference between the number of units of food item chosen the most and the number of units of another food item chosen the least, is as small as possible.

To solve this problem, he uses maximum possible number of units of all the items so that total amount of protein, carbohydrates and fats is not more than prescribed limit. For example - if total nutrition required is 100P, 130C and 130F (where P is Protein, C is Carbohydrates and F is Fats) and 2 different food items, viz. Item A and Item B, have following amount of nutrition:

Item A - 10P, 20C, 30F

Item B - 20P, 30C, 20F

then, he can have (maximum possible) 2 units of all the items as having 3 units will exceed the prescribed amount of Carbohydrates and fats.

Next, he chooses food items to fulfill the remaining nutrients. He chooses one more units of maximum number of food items. He continues this process till he cannot add a unit of any food item to his diet without exceeding the prescribed limit. In this example, he can choose one more unit of item B or one more unit of item A. In case he has two sets of food items then the priority is given to fulfill the requirements of Protein, Carbohydrates, and Fats in that order. So he chooses item B.

You will be provided the maximum nutrients required and the nutrients available in various food items. You need to find the amount of nutrients for which there is a shortfall as compared to the prescription, after making his selection using the process described above. In the example he still needs 20P, 0C, 10F to achieve his target.

Constraints

Number of Food Items <= 10

Maximum amount of Nutrients is less than 1500 i.e. x +y + z <= 1500

Amount of P, C, F in two food items will not be same

P, C, F values in input can be in any order

Output should be in order - P, C, F.

Input

First line contains the maximum limit of nutrients in the following format.

xP yC zF, where x, y and z are integers

Second line contains nutrient composition of different food items separated by pipe (|).

Output

Print the shortfall for each nutrient type, fulfilled separated by space.

E.g. If the output is 10P, 20 C, 30 F, then print "10 20 30" (without quotes).

Time Limit

1

Examples

Example 1

Input

100P 130C 130F

10P 20C 30F|20P 30C 20F

Output

20 0 10

Explanation

Having 2 units of item A and 3 units of item B provides - 2 * [10P 20C 30F] + 3 * [20P 30C 20F] = 100P, 130C, 120F. This is the best combination that can reduce the shortfall [20P, 0C, 10F] without exceeding his prescription. In contrast, if 3 units of A and 2 units of B are chosen then 3 * [10P 20C 30F] + 2 * [20P 30C 20F] = 70P, 120C, 130F produces a shortfall of [30P, 10C, 0F]. However, since protein shortfall in this case is more than the previous combination, this is not the right combination to follow.

Example 2

Input

130P 120C 110F

4P 9C 2F|4P 3C 2F|7P 1C 3F

Output

2 4 50

Explanation

Having 9 units of item A, 9 units of item B and 8 units of Item C provides - 9 * [4P 9C 2F] + 9 * [4P 3C 2F] + 8 * [7P 1C 3F] = 108P, 116C, 60F. This is the best combination that can reduce the shortfall [2P, 4C, 50F] without exceeding his prescription.

Number Distancing

Problem Description

Consider 9 natural numbers arranged in a 3x3 matrix:

n11 n12 n13

n21 n22 n23

n31 n32 n33

Define numbers "in contact" with a given number to be those that appear closest to it on the same row, column or diagonally across:

Contacts of number n11: n12, n22 and n21

Contacts of number n12: n11, n21, n22, n23, n13

Contacts of number n13: n12, n22, n23

Contacts of number n21: n11, n12, n22, n32, n31

Contacts of number n22: n11, n12, n13, n23, n33, n32, n31, n21

Contacts of number n23: n13, n12, n22, n32, n33

Contacts of number n31: n21, n22, n32

Contacts of number n32: n31, n21, n22, n23, n33

Contacts of number n33: n32, n22, n23

The problem now is that numbers having a common factor (other than 1) should not be "in contact". In other words, a pair of numbers can remain neighbours only if their highest common factor is 1.

The following rules apply to enforce this "distancing":

1. The central number (n22) stays put.

2. The corner numbers (n11, n13, n33, n31) can move in the same row or column or diagonally away from the centre.

3. The numbers "on the walls" (n12, n23, n32, n21) can only move from the walls i.e. n21 can only move "left", n12 can only move "up", n23 can only move "right" and n32 can only move "down".

4. Each number should stay put as far as possible and the "distancing" operation should result in the least number of numbers ending up without any contacts.

5. After satisfying rule 4, if there are multiple options for the final matrix, then the "distancing" operation should result in the smallest (m x n matrix, including the intervening blank space elements, with the least possible value of m*n).

6. If, after satisfying all the rules above, there are multiple distancing options for a set of numbers, the largest number keeps to its original cell.

Constraints

1 <= Element of grid <= 100

Input

First line consists of 9 space separated integers denoting n11, n12, n13, ... n23, n33 respectively.

Output

Print the "contact" less numbers in ascending order of their value separated by space. Output "None" if there are no such numbers.

Time Limit

1

Examples

Example 1

Input

23 33 12 1 2 5 25 6 10

Output

10

Explanation

Initial configuration

23 33 12

1 2 5

25 6 10

The optimal distancing options result in the following possibility (space denoted by *):

23 33 * 12

1 2 5 *

25 * * *

* 6 * 10

10 ends up as the number without contacts.

Example 2

Input

1 2 3 4 5 6 7 8 9

Output

None

Explanation

Initial configuration:

1 2 3

4 5 6

7 8 9

The optimal distancing options result in the following 5x3 matrix (space denoted by *):

* 2 3

1 * *

4 5 6

7 * *

* 8 9

There is finally no number without a contact.

Example 3

Input

2 6 2 10 19 12 2 20 2

Output

2 2 2 2 10 12

Explanation

Initial Configuration:

Approach 1) Moving 6 and 20

Final Matrix

2 * 6 * 2

* * * * *

* 10 19 12 *

* * * * *

2 * 20 * 2

Approach 2) Moving 10 and 12

2 * * * 2

* * 6 * *

10 * 19 * 12

* * 20 * *

2 * * * 2

We prefer approach 2) and not 1) since the largest of all the elements (20) needs to be retained in it's original cell.

Critical Planets

Problem Description

War between Republic and Separatist is escalating. The Separatist are on a new offensive. They have started blocking the path between the republic planets (represented by integers), so that these planets surrender due to the shortage of food and supplies. The Jedi council has taken a note of the situation and they have assigned Jedi Knight Skywalker and his Padawan Ahsoka to save the critical planets from blockade (Those planets or system of planets which can be accessed by only one path and may be lost if that path is blocked by separatist).

Skywalker is preparing with the clone army to defend the critical paths. He has assigned Ahsoka to find the critical planets. Help Ahsoka to find the critical planets(C) in ascending order. You only need to specify those planets which have only one path between them and they cannot be accessed by any other alternative path if the only path is compromised.

Constraints

M <= 10000

N <= 7000

Input

First line contains two space separated integers M and N, where M denotes the number of paths between planets and N denotes the number of planets.

Next M lines, each contains two space separated integers, representing the planet numbers that have a path between them.

Output

C lines containing one integer representing the critical planet that they need to save in ascending order of the planet number if no planet is critical then print -1

Time Limit

1

Examples

Example 1

Input

3 4

0 1

1 2

2 3

Output

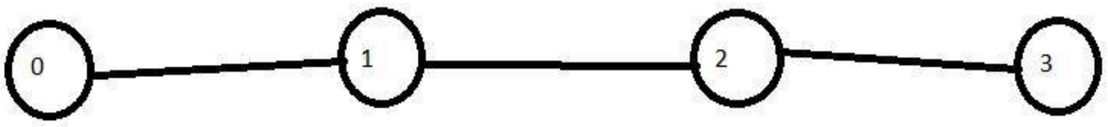
0

1

2

3

Explanation



Since all the planets are connected with one path and cannot be accessed by any alternative paths hence all the planets are critical.

Example 2

Input

7 6

0 2

0 1

1 2

2 3

4 5

3 4

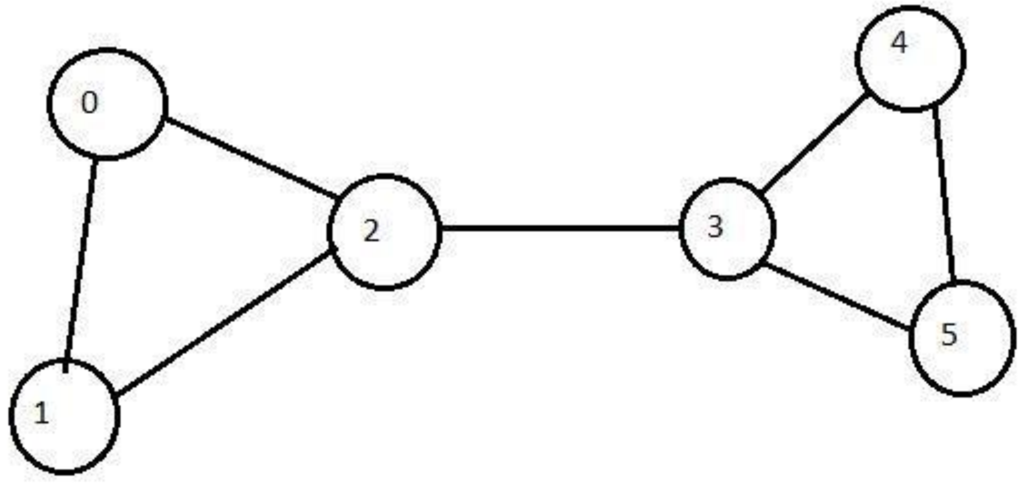
3 5

Output

2

3

Explanation



If the republic loose the path between 2 and 3 then the two system of planets will not be able to communicate with each other. Hence 2 and 3 are critical planets.

Lift

Problem Description

In a building there are some lifts. Optimize the allocation of lifts.

Say there are N requests and M lifts. Minimize the maximum request waiting time.

Rules of lift allocation

1) One needs to assign indexes to lifts. i.e. decide lift #1, lift #2, lift #3, etc apriori.

2) Since all N requests are known apriori decide the initial (at time t = 0) location of lift such that it will help in minimizing waiting time.

3) Even if all the N requests time are known apriori, other than initial time, i.e. t = 0, the waiting lifts cannot be moved towards target floor until the request is actually issued.

4) After a request is issued for optimality calculation, even a lift is in transit it can be considered for serving that request.

5) If at any moment, more than one lift can serve the request with same waiting time, give preference to the lift with lower index. i.e. If Lift #2 and Lift #4 can serve a particular request in 2 seconds, then prefer Lift #2 over Lift #4.

6) Neglect the time required to get in and get out of a lift.

7) Once a lift starts serving a request it would not stop in between. If would finally stop at destination of that request.

8) The speed of lift is 1 floor/second.

Constraints

0 <= T <= 86400.

0 <= S, D <= 100.

1 <= N, M <= 1000.

Input

First line contains 2 integers, N and M, representing the number of requests and number of lifts in the building respectively.

Next N lines contain 3 integers, T, S, and D, representing the timestamp of request, source floor, destination floor respectively.

Output

Print single integer representing the waiting time

Time Limit

1

Examples

Example 1

Input

3 2

0 2 3

4 2 5

6 7 3

Output

3

Explanation

There are 3 requests and 2 lifts.

Initially at time t = 0, both the lifts, lift #1 and lift #2 will be at floor 2 (Initial allocation).

Request #1 and Request #2 can be responded instantly, i.e. waiting time = 0.

Lift #1 will get unallocated at floor 3 at time t = 1.

Lift #1 can move only after the next request is actually received at time t = 4, if required. Relate this with Rule #3 mentioned in problem description.

Lift #2 will get unallocated at floor 5 at time t = (4 + 3) = 7.

Now, if Lift #1 is used to respond request #3, waiting time would be 4 seconds since Lift#1 is at floor #3 and will need 4 seconds to reach floor #7.

In this case, waiting time of all requests - {0,0,4} and the maximum waiting time is 4.

Instead, if Lift #2 is used to respond request #3, waiting time would be calculated as follows

Lift #2 will get unallocated at time t = 7, so we will have to wait 7-6 = 1 seconds for lift #2 to complete it's previous request.

Time needed for Lift #2 to travel from floor #5 to floor #7 = 7-5 = 2 seconds. Therefore, total waiting time = 1 + 2 = 3 seconds.

In this case, waiting time of all requests - {0, 0, 3} and the maximum waiting time is 3.

As we have to minimize the maximum waiting time, since 2nd option is yielding lesser waiting time than 1st option, 2nd option is the answer.

Therefore, the output is 3.

Example 2

Input

5 3

0 2 3

4 2 5

6 7 3

3 5 6

2 5 7

Output

1

Explanation

There are 5 requests and 3 lifts.

Initially, at t = 0, lift #1 will be at floor #2 whereas lift #2 and #3 will be at floor#5 (Initial allocation).

Request #1, #4, #5 can be responded instantly, i.e. waiting time = 0.

Lift #1 will get unallocated at floor 3 at time t = 1.

Lift #2 will get unallocated at floor 7 at time t = 2 + 2 = 4.

Lift #3 will get unallocated at floor 6 at time t = 3 + 1 = 4.

Request #2 can be served by lift #1. Waiting time would be 2 - 1 = 1.

Now, lift #1 will get unallocated at floor #5 at time t = 4 + 1 + 3 = 8.

Request #3 can be served by lift #3 as it will be floor #6 at t = 4. Waiting time = 0.

Waiting time of all requests - {0, 1, 0, 0, 0}.

Therefore, the output is 1.

Lift

Problem Description

In a building there are some lifts. Optimize the allocation of lifts.

Say there are N requests and M lifts. Minimize the maximum request waiting time.

Rules of lift allocation

1) One needs to assign indexes to lifts. i.e. decide lift #1, lift #2, lift #3, etc apriori.

2) Since all N requests are known apriori decide the initial (at time t = 0) location of lift such that it will help in minimizing waiting time.

3) Even if all the N requests time are known apriori, other than initial time, i.e. t = 0, the waiting lifts cannot be moved towards target floor until the request is actually issued.

4) After a request is issued for optimality calculation, even a lift is in transit it can be considered for serving that request.

5) If at any moment, more than one lift can serve the request with same waiting time, give preference to the lift with lower index. i.e. If Lift #2 and Lift #4 can serve a particular request in 2 seconds, then prefer Lift #2 over Lift #4.

6) Neglect the time required to get in and get out of a lift.

7) Once a lift starts serving a request it would not stop in between. If would finally stop at destination of that request.

8) The speed of lift is 1 floor/second.

Constraints

0 <= T <= 86400.

0 <= S, D <= 100.

1 <= N, M <= 1000.

Input

First line contains 2 integers, N and M, representing the number of requests and number of lifts in the building respectively.

Next N lines contain 3 integers, T, S, and D, representing the timestamp of request, source floor, destination floor respectively.

Output

Print single integer representing the waiting time

Time Limit

1

Examples

Example 1

Input

3 2

0 2 3

4 2 5

6 7 3

Output

3

Explanation

There are 3 requests and 2 lifts.

Initially at time t = 0, both the lifts, lift #1 and lift #2 will be at floor 2 (Initial allocation).

Request #1 and Request #2 can be responded instantly, i.e. waiting time = 0.

Lift #1 will get unallocated at floor 3 at time t = 1.

Lift #1 can move only after the next request is actually received at time t = 4, if required. Relate this with Rule #3 mentioned in problem description.

Lift #2 will get unallocated at floor 5 at time t = (4 + 3) = 7.

Now, if Lift #1 is used to respond request #3, waiting time would be 4 seconds since Lift#1 is at floor #3 and will need 4 seconds to reach floor #7.

In this case, waiting time of all requests - {0,0,4} and the maximum waiting time is 4.

Instead, if Lift #2 is used to respond request #3, waiting time would be calculated as follows

Lift #2 will get unallocated at time t = 7, so we will have to wait 7-6 = 1 seconds for lift #2 to complete it's previous request.

Time needed for Lift #2 to travel from floor #5 to floor #7 = 7-5 = 2 seconds. Therefore, total waiting time = 1 + 2 = 3 seconds.

In this case, waiting time of all requests - {0, 0, 3} and the maximum waiting time is 3.

As we have to minimize the maximum waiting time, since 2nd option is yielding lesser waiting time than 1st option, 2nd option is the answer.

Therefore, the output is 3.

Example 2

Input

5 3

0 2 3

4 2 5

6 7 3

3 5 6

2 5 7

Output

1

Explanation

There are 5 requests and 3 lifts.

Initially, at t = 0, lift #1 will be at floor #2 whereas lift #2 and #3 will be at floor#5 (Initial allocation).

Request #1, #4, #5 can be responded instantly, i.e. waiting time = 0.

Lift #1 will get unallocated at floor 3 at time t = 1.

Lift #2 will get unallocated at floor 7 at time t = 2 + 2 = 4.

Lift #3 will get unallocated at floor 6 at time t = 3 + 1 = 4.

Request #2 can be served by lift #1. Waiting time would be 2 - 1 = 1.

Now, lift #1 will get unallocated at floor #5 at time t = 4 + 1 + 3 = 8.

Request #3 can be served by lift #3 as it will be floor #6 at t = 4. Waiting time = 0.

Waiting time of all requests - {0, 1, 0, 0, 0}.

Therefore, the output is 1.

SudoKube

Problem Description

John, a research scholar / Professor / Puzzle solver wants your help in publishing his work on SudoKube on his online blog for his followers and students.

A SudoKube is a mixture of Rubics cube and Sudoku. A SudoKube has exactly 6 appearances of every digit from 1 to 9 across the cube, whereas Rubics cube has 6 different colours.

As John wants to publish his work in text /document form (no video) he's concerned how he would depict the step by step work of rotation in 2D form. Following are the notions and concepts John follows:

1. The six faces of the cube are named FRONT, BACK, UP, DOWN, LEFT and RIGHT respectively.

2. Just like a Rubics cube which move in 90 and 180 degrees in both clockwise and anti clockwise directions, so can the SudoKube

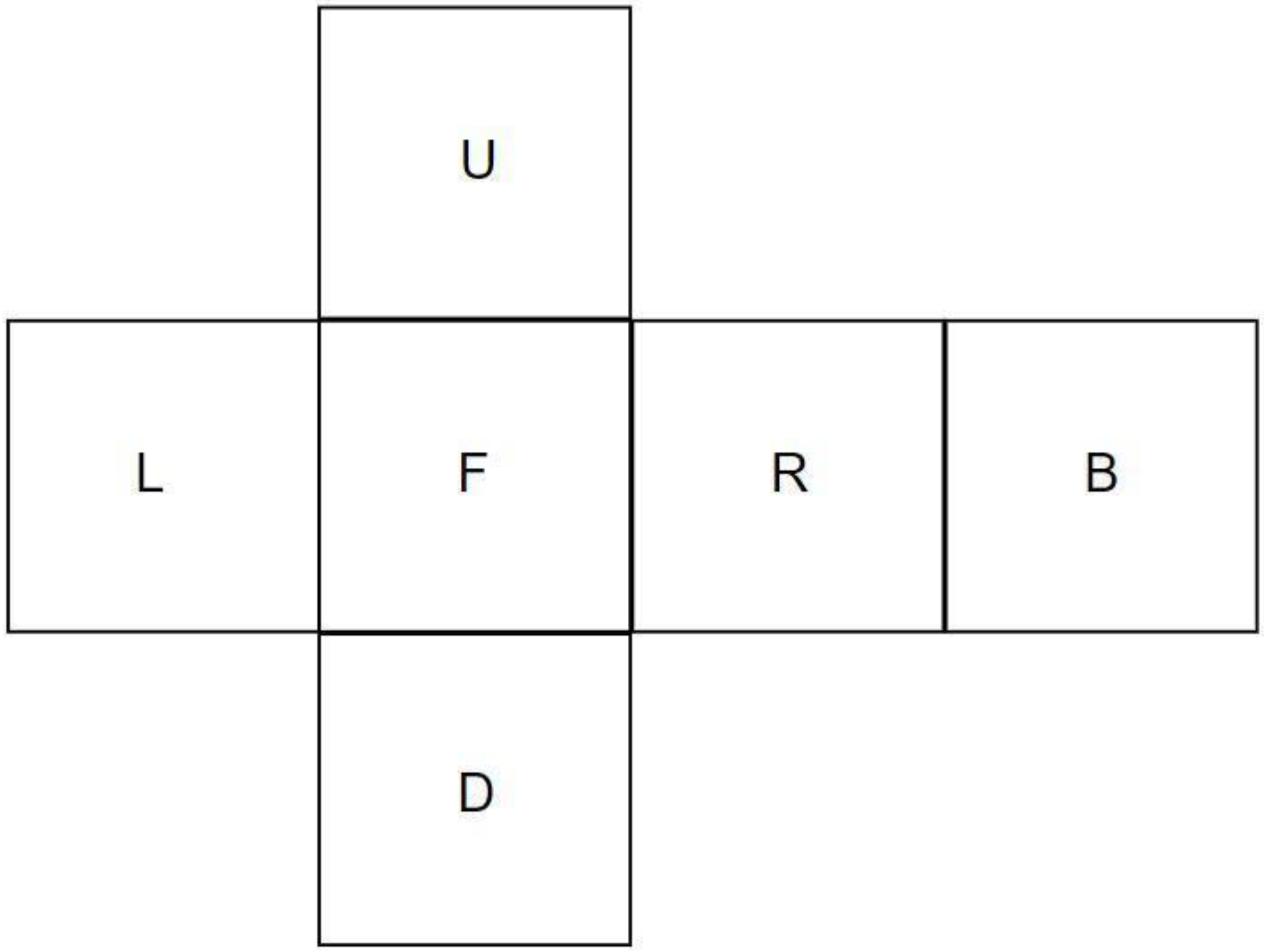
3. Any given face of the cube is a 3x3 square matrix whose indices are denoted by (0,0) to (2,2). Diagram below illustrates the same.

4. An elementary move is denoted in the following fashion.

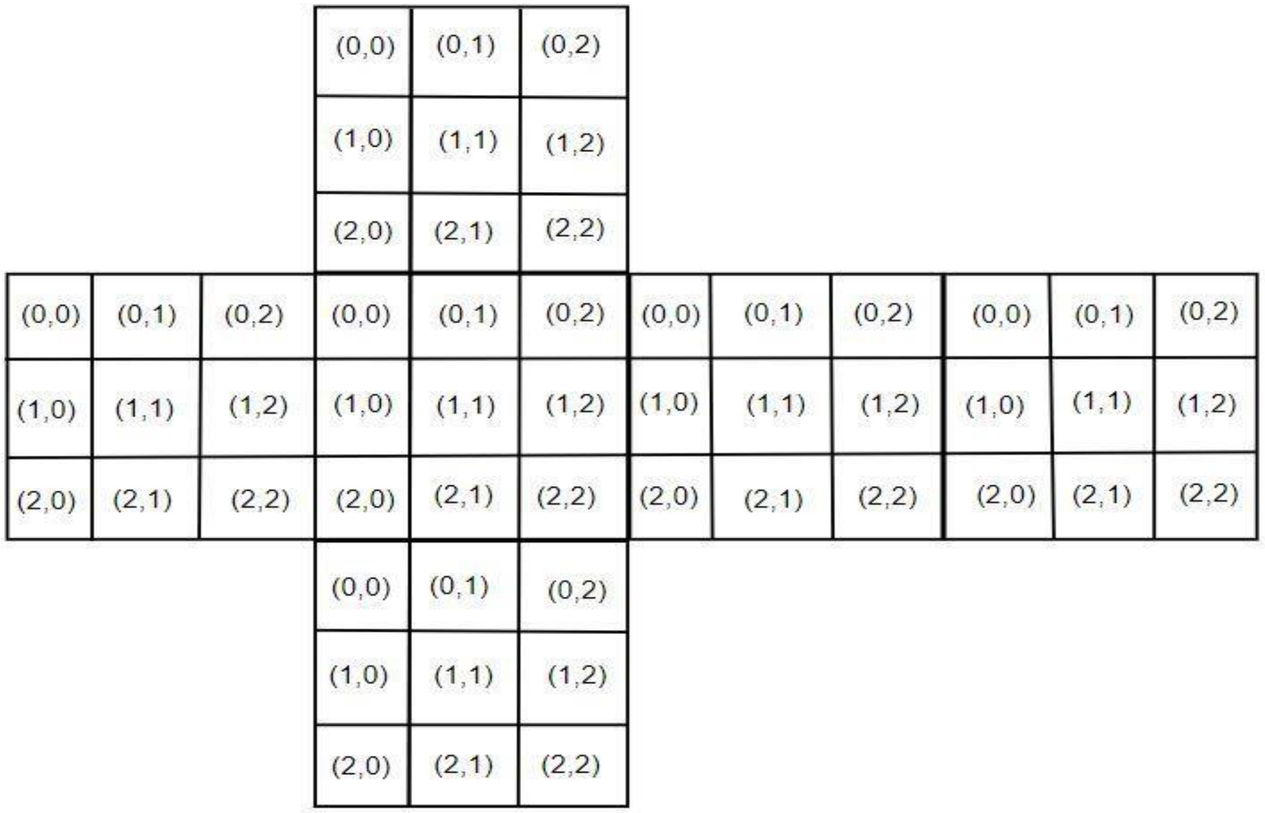
    i. If a given face is rotated by 90 degrees clockwise about the axis passing from the centre of the face to the centre of the cube, the move is denoted by the first letter of the name of the face.

    ii. If the rotation is anticlockwise by 90 degrees, the letter is followed by an apostrophe (').

    iii. If the rotation is by 180 degrees, the letter is followed by a 2.



Above image display the position of the faces



Above diagram displays the indices of the matrix on the faces

John wants to test his notations on you. He has given you the initial position of the SudoKube and he has given you a set of operations to be performed on the SudoKube basis his notation. After applying all the operations, the final SudoKube state should be the same as what John expects. Your task is to apply the operations and print the final SudoKube state.

Constraints

Values in SudoKube will be between 1 and 9

No of moves < 15

Input

• First eighteen lines contain the values of the faces on SudoKube in the order given below

D D D

D D D

D D D

U U U

U U U

U U U

L L L

L L L

L L L

F F F

F F F

F F F

R R R

R R R

R R R

B B B

B B B

B B B

where

- D for Down face
- U for upper face
- L for Left face
- F for Front face
- R for Right face
- B for Back face.

Input contains digits from 1 to 9 instead of letters; letters are displayed for better understanding of the faces and the expected input format

• Nineteenth line contains a sequence of space delimited moves that need to be performed on the SudoKube
Example 1: D F2 R' U - to understand this please refer second example from the *Examples* section below

Example 2: L2 U B F' D2 R - lets understand how to interpret this set of operations

- L2 means rotate the Left side by 180 degrees
- U means rotate the Up side by 90 degrees clockwise
- B means rotate the Back side by 90 degrees clockwise
- F' means rotate the Front side by 90 degrees anticlockwise
- D2 means rotate the Down side by 180 degrees
- R means rotate the Right side by 90 degrees clockwise

In summary, first eighteen lines denotes the state of the SudoKube, 19th line denotes the operation to be performed on that state and output should be the resulting state.

Output

Print 3x3 matrix corresponding to the order (D, U, L, F, R, B). Between every 3x3 matrix there should be a new line.

Time Limit

1

Examples

Example 1

Input

4 7 1

2 8 7

6 3 5

5 8 3

3 1 6

9 4 2

5 2 4

3 7 8

5 1 9

6 1 4

9 4 8

2 5 7

```
7 9 1
1 9 6
6 2 8
8 6 3
7 2 5
3 9 4
F
```

Output

```
6 1 7
2 8 7
6 3 5

5 8 3
3 1 6
9 8 4

5 2 4
3 7 7
5 1 1

2 9 6
5 4 1
7 8 4

9 9 1
4 9 6
2 2 8
```

8 6 3

7 2 5

3 9 4

Explanation:

The output shows the state of SudoKube when the front side is rotated clockwise

Example 2

Input

4 7 1

2 8 7

6 3 5

5 8 3

3 1 6

9 4 2

5 2 4

3 7 8

5 1 9

6 1 4

9 4 8

2 5 7

7 9 1

1 9 6

6 2 8

8 6 3

7 2 5

3 9 4

D F2 R' U

Output

2 4 5

```
3 8 9
5 7 6

4 3 5
2 1 8
8 7 6

9 1 3
3 7 1
3 9 7

1 6 7
8 4 6
4 1 6

1 6 3
9 9 5
4 8 4

5 2 2
7 2 5
9 2 8
```

Explanation

The above output prints the state of cube after D F2 R' U operation are performed. Here

· D means rotate the Down side by 90 degrees clockwise

· F2 means rotate the Front side by 180 degrees

· R' means rotate the Right side by 90 degrees anti clockwise

· U means rotate the Up side by 90 degrees clockwise