TCS CodeVita Season-9 Set-1

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.

Example 2

Input

12

.#.***#.*.

.#..*.#***

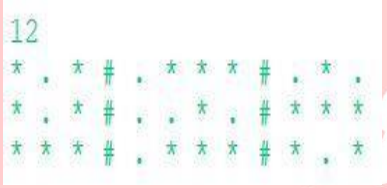
#.#*.*

Output

U#I#A

Explanation

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



Codekart

Problem Description

Codu wants to create a shopping application. The application would sell only SHIRT and SHOE and have a cost that can be modified based on market needs. This application should allow users in two roles, viz. store manager(SM) and shopper(S).

Codu wants to test the app. He wants the application to execute a few commands and print the output.

Following is the list of allowed **commands**:
CMD SM ADD [ITEM NAME] [ITEM QTY] - adds the given quantity of the item to the inventory and prints(returns) the quantity added, otherwise prints -1 when there is any error or invalid input. Item qty can only be whole numbers > 0.
CMD SM REMOVE [ITEM NAME] - removes the item from the inventory, returns and prints -1 when there is an error, otherwise prints(returns) 1.

CMD SM GET_QTY [ITEM NAME] - returns and prints the currently available quantity for the item in the inventory, otherwise prints(returns) 0 in case the item is not found.
CMD SM INCR [ITEM NAME] [ITEM QTY] - adds the given quantity of the item to the inventory and prints(returns) the quantity added, otherwise prints(returns) -1 when there is any error or invalid input. item qty can only be whole numbers > 0.
CMD SM DCR [ITEM NAME] [ITEM QTY] - removes the given quantity of the item from the inventory and prints(returns) the quantity added, otherwise prints(returns) -1 when there is any error or invalid input. item qty can only be whole numbers > 0.
CMD SM SET_COST [ITEM NAME] [COST] - sets the cost of the item, returns the value, otherwise prints -1 in case of any errors or invalid input. Cost must be decimal.

CMD S ADD [ITEM NAME] [ITEM QTY] - adds the given quantity of the item to the shopping cart and prints(returns) the quantity added, otherwise prints -1 when there is any error or invalid input. Item qty can only be whole numbers > 0.
CMD S REMOVE [ITEM NAME] - removes the item from the shopping cart, returns and prints -1 when there is an error, otherwise prints(returns) 1.
CMD S INCR [ITEM NAME] [ITEM QTY] - adds the given quantity of the item to the shopping cart and prints(returns) the quantity added, otherwise prints(returns) -1 when there is any error or invalid input. item qty can only be whole numbers > 0.
CMD S DCR [ITEM NAME] [ITEM QTY] - removes the given quantity of the item from the shopping cart and prints(returns) the quantity added, otherwise prints(returns) -1 when there is any error or invalid input. item qty can only be whole numbers > 0.
CMD S GET_ORDER_AMOUNT - gets the total price of the items in the cart, returns the value, otherwise prints -1 in case of any error or invalid input. The total amount should be rounded and printed up to two decimal places.
NOTE- Increment and Decrement operations are only possible when the item is already in the inventory or cart. If increment or decrement is attempted on items that do not exist in the cart, then the command should return and print -1.

If an attempt is made to add an item that is already in the inventory or cart, such operations should result in an error and must return and print -1.

If an item which is present in the cart of inventory is removed using *remove* command and an increment or decrement operation is performed on it, such operations should result in an error and must return and print -1.

If any item quantity after decrement becomes zero, the same is removed from the corresponding inventory or cart. Performing increment or decrement operation after such a previous decrement operation, should result in an error and return -1.

You need to think of other similar error conditions while implementing the solution.

Please note at the beginning of a test case or command set, both the inventory as well as the cart is empty.

Here,

SM= STORE MANAGER
S= SHOPPER

You are required to create the application for Codu to manage the shopping kiosk.

The first line of input **T**, gives the number of test cases.

Each test set is a set of commands, which ends with "END" string.

Each command in a test case is on a new line

Constraints

1<=T<=10

Input

First line contains an integer T, which denotes the number of test cases

Second line onwards, there will be commands until we receive END command. Any command after the END command belongs to next test case.

For command format refer to Example section.

Output

Print output of every command. (Print double value for command SET_COST(rounding to one decimal places) and GET_ORDER_AMOUNT(rounding to two decimal places))

Time Limit

1

Examples

Example 1

Input

1

CMD SM SET_COST SHOE 5

CMD SM SET_COST SHIRT 10

CMD SM ADD SHOE 5

CMD SM ADD SHIRT 10

CMD SM DCR SHIRT 5

CMD SM INCR SHOE 5

CMD SM GET_QTY SHIRT

CMD SM GET_QTY SHOE

CMD SM REMOVE SHIRT

CMD SM GET_QTY SHIRT

CMD S ADD SHOE 2

CMD S INCR SHOE 2

CMD S DCR SHOE 1

CMD S GET_ORDER_AMOUNT

END

Output

5.0

10.0

5

10

5

5

5

10

1

0

2

2

1

15.00

Explanation :

From commands "**CMD SM SET_COST SHOE 5**" and "**CMD SM SET_COST SHIRT 10**"

We are successfully setting the cost as 5.0 and 10.0 respectively.

From next commands "**CMD SM ADD SHOE 5**" and "**CMD SM ADD SHIRT 10"**

Quantity of 5 shoes and 10 shirts has been successfully added to the inventory.

From next commands "**CMD SM DCR SHIRT 5**"" and "**CMD SM INCR SHOE 5"**

Shirt quantity is decremented by 5 and shoe quantity is incremented by 5. This leaves us with 5 shirts and 10 shoes in the inventory.

From next commands "**CMD SM GET_QTY SHIRT**" and "**CMD SM GET_QTY SHOE"**

We are getting the quantity of shirt and shoe, which is 5 and 10 respectively.

From next command "**CMD SM REMOVE SHIRT"**

Shirt is removed from the inventory and hence 1 is printed.

From next command "**CMD SM GET_QTY SHIRT"**

We are querying the quantity of shirt, which is 0 as it was removed in the previous command.

From next command "**CMD S ADD SHOE 2"**

Shopper adds two shoes to the cart hence 2 is printed.

From next commands "**CMD S INCR SHOE 2**" and "**CMD S DCR SHOE 1**"

The user increments these shoes by 2 and then decrements by 1 hence 2 and 1 are printed. SO current shoes in cart= 2+2-1=3

From next commands "**CMD S GET_ORDER_AMOUNT**"

The next command asks to print order amount or cart value, which is the cost of shoes * the number of shoes=5*3=15 hence 15.00 is printed by rounding to two decimal places.

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.

Engagement Ring

Problem Description

Sejal was on a month-long vacation to Europe and has a return trip to India from Lisbon, a city in south west part of Europe. However, a day before the return flight she realizes that she lost her engagement ring. After much contemplation, she decides to go to all the cities she visited to find her ring.

She maps all the cities she visited on a graph with Lisbon being at point (0,0). She then makes a route plan to visit all the cities and return to Lisbon by taking the shortest possible distance. She does not remember having her ring even in the first city she visited so there are high chances that she may have lost her ring in the initial part of her trip also. In case there are more than one routes which have the shortest possible distance, she picks that route in which the first city she visited, comes first. For example, if she visited cities 2,5,7,1,8,3 in that order and routes 0,1,8,3,2,5,7,0 and 0,8,3,1,5,2,7,0 (0 being Lisbon) have the same shortest possible distance then she will choose route 0,1,8,3,2,5,7,0 because she visited city 1 before city 8.

Her travel guide, Harry also offers to help Sejal. Sejal asks him to travel separately on the same route, but in reverse direction such that each city is visited only once. They plan to travel 20 Kms in each city on taxi to search the ring. Inter-city travel is done on trains only.

A secret service officer knows the coordinates of the city that Sejal visited during her the trip in that order. He also knows the city in which the ring is lost but will inform Sejal or Harry only when one of the two is in that city.

He knows the path that Sejal has drawn to visit all the cities and return to Lisbon. With Sejal and Harry following that path and either one of them reaching the city where the ring is lost, the secret service officer will inform the person in that city, that the ring will be 10 km away from their current location. They will travel back 10 km in the same city, to catch the train back to Lisbon. Calculate the total distance traveled by Sejal in her search and her return to Lisbonfrom that city.

If the ring is found by Sejal, she goes back to Lisbon from that city. If the ring is found by Harry, he informs Sejal on call at that point. If Sejal is in a city (searching in taxi) she returns to Lisbon via train from that city (without searching any further in that city). If Sejal is on train, she will need to complete the journey and then return from that other city. If the call comes at the exact point she is taking a train, she can return from that city itself.

Each unit in the graph is equal to 1 Km. Assume the speed of all trains and taxis is same. Do not consider the decimal values while calculating the distance between two cities, ie. distance will be the floor of the calculated distance.

Constraints

Floor of the value is to be used while calculating distance between cities

Each city is connected to every other city via trains

Total number of cities <10

Input

First Line will provide the coordinates (x|y) of cities separated by semicolon (;)

Second Line will provide the number of city where the ring is found

Output

One integer representing the number total distance traveled by Sejal

Time Limit

1

Examples

Example 1

Input

0|90;90|90;90|0

2

Output

347

Explanation

Since routes 1,2,3 and 3,2,1 will give the shortest distance, she will select route 1,2,3 as she visited city 1 before city 3. However, Harry will follow the opposite path - 3,2,1. They both will be in city 2 when they will find it. Total distance Sejal covers will be Lisbon to city 1 + 20Kms, City 1 to City 2 + 10 km +10 km and then City 2 to Lisbon. i.e. 90 + 20 + 90 + 20 +127 = 347 km

Example 2

Input

10|70;30|30;80|20;120|75;90|120

3

Output

334

Explanation

Sejal will choose the route 1, 5, 4, 3, 2 with minimum distance of 378 km. However, Harry will follow the opposite path - and will find the ring in City 3. The moment Harry finds the ring, Sejal will be travelling from City 1 to City 5. So she will complete the journey till City 5 and return from there to Lisbon. So the total distance covered by Sejal will be 70+20+94+150 = 334 km

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.

Paste Reduction

Problem Description

Some keys in Codu's computer keyboard are not working. Fortunately for Codu, these characters are present in a previously existing text file.

He wants to write a paragraph which involves typing those characters whose keys are defunct in Codu's keyboard. Only option left for Codu is to copy-paste those characters from the previously existing text files. However, copy-pasting keys is a laborious operation since one has to switch windows and also previously copied items are lost once a new set of characters are copied. Hence, Codu wants to minimize the number of times he needs to copy-paste from that text file. Fortunately there can be situations where previously copied characters are readily available for pasting. Help Codu devise a method to minimize the number of times a paste operation is needed, given - the text he intends to type and the faulty keys

Constraints

0 < Length of paragraph <= 1000 characters

0 < number of faulty keys in keyboard <= 36

Only a to z and 0 to 9 keys can be faulty.

Input paragraph will not contain any upper-case letter.

Input

First line contains a paragraph that is be to be written.

Second line contains a string. This string has to be interpreted in the following fashion

All characters in that string correspond to faulty keys

That string is available for copy as-is from the other text file that Codu is referring

Also, individual characters can always be copied from the same text file

Output

Single integer denoting minimum number of copy operations required

Time Limit

1

Examples

Example 1

Input

supreme court is the highest judicial court

su

Output

4

Explanation

Codu will first paste su from the file when typing characters su in the word supreme.

In the second instance, Codu will need to copy character u and paste it when typing character u in the word court.

In the third instance, Codu will need to copy character su and paste su when typing character s in the word is. Codu will back track using the left arrow key and type the characters "the highe".

In the fourth instance, Codu will again paste su. The overall string typed until this moment is "supreme court is the highesuu". Codu will then back track again using left arrow key and type characters "t j". At this point the typed string is "supreme court is the highest juu". Cursor is after character j. Codu will use right arrow key and now the cursor will be after ju. Codu will now type "udicial co". String typed till this point is "supreme court is the highest judicial cou". Cursor is after character o. Codu will use right arrow and the cursor will be after character u. Finally Codu will type "rt".

Final string - "supreme court is the highest judicial court" - is thus fully typed. Here, the number of paste operations are 4.