LAB-PRAC-10_MAT-FUN

Stack (p1v1d1)

Stack [20 marks]

Problem Statement
The stack is a data structure that is incredibly useful in computer science. Mr C uses a stack to manage your programs. Your Windows and Linux operating systems also use stacks to perform various tasks. However, the basic premise of a stack is extremely simple and all of you encounter stacks all the time.

In the hall mess, when several plates are placed on top of each other, it is called a *stack* of plates. Note that when you want to take a plate from this stack. it is not very easy to take a plate from the bottom of the stack. Instead, what you do is take the plate at the top of the stack. Similarly, when the mess workers want to add more plates to the existing stack, they dont add new plates to the bottom of the stack since it is very difficult to do that. Instead, they simply add new plates to the top of the existing stack. Stacks are often called LIFO (last-in-first-out) structures since the last element (e.g. plate, or) that was added to the stack is the first to be remove.

The computer science stacks work exactly the same way. The stacks we will implement in this question will have three operations. The stack will have a limit size of MAX

1. Push x: if there are already MAX elements on the stack, print the error message "FULL" (without quotes), otherwise put the element x on the top of the stack.
2. Pop: If the stack is empty, print the error message "EMPTY" (without quotes), otherwise remove the element at the top of the stack and print that element.
3. Check: If the stack is empty, print "EMPTY" (without quotes), else if it is full, print "FULL" (without quotes), else print "NOT EMPTY" (without quotes)

In the first line of the input, you will be give the value of MAX as a strictly positive integer. In the next several lines, you will see instructions on what to do

1. E x: push the integer x onto the stack or print the error message. There will be a single space between the character 'E' and the integer x.
2. D: pop the element at the top of the stack and print it or print the error message
3. C: perform the check operation and print the appropriate message
4. X: no more operations to perform

Thus, the list of operations will be terminated by an X. All elements pushed onto the stack will be integers.

Caution

1. Print all error messages as well as popped elements on separate lines.
2. If you are using getchar() to process input character by character, be careful to read in the newlines at the end of each line as well.
3. We will not penalize you for extra newlines at the end of your output. However, do not have extra newlines in the middle of your output or else have trailing spaces in any line of the output.

HINTS:

1. Use an array with MAX elements to manipulate the stack. Use an integer, say idx, to store the index of the top of the stack. Initially idx can be -1 to indicate that the stack is empty. If we then push 2 then push 3, the array will look like {2, 3, ...} and idx = 1. If we now pop an element, idx will become 0 to indicate that the top of the stack is the element 2 now since 3 got popped.
2. Write functions to perform various stack operations to ensure your code looks neat and clean.

EXAMPLE:
INPUT
2
C
E 5
C
E 4
C
E 3
D
D
D
X

OUTPUT:
EMPTY
NOT EMPTY
FULL
FULL
4
5
EMPTY

Explanation

1. The stack only has a capacity of 2 elements.
2. Command 1: C - Stack is empty at the moment so print EMPTY
3. Command 2: E 5 - Push 5 into the stack. Now stack looks like [5]
4. Command 3: C - Stack is neither full nor empty so print NOT EMPTY
5. Command 4: E 4 - - Push 4 into the stack. Now stack looks like [5 4]
6. Command 5: C - Stack is full so print FULL
7. Command 6: E 3 - Stack is already full - cannot push another element so print FULL - stack still looks like [5 4]
8. Command 7: D - Pop top element 4 from stack and print 4. Now stack looks like [5]
9. Command 8: D - Pop top element 5 from stack and print 5. Now stack is empty
10. Command 9: D - Stack is already empty - cannot pop another element so print EMPTY - stack is still empty
11. Command 10: X - No more processing to do.

Grading Scheme:

Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
2 C E 5 C E 4 C E 3 D D D X	 EMPTY NOT EMPTY FULL FULL 4 5 EMPTY
1 E 10 D E 4 D D E 4 E 5 C D D X	 10 4 EMPTY FULL FULL 4 EMPTY
10 E 1 E 2 E 3 E 4 E 5 E 6 D D C E 7 E 8 E 9 E 0 E 5 C E 6 C D D	6 5 NOT EMPTY NOT EMPTY FULL 6 5 0 9 8 7 4 3 5 4 2 1 EMPTY

D D D D D D E 4 E 5 D D D D C X	
5 E 10 E 20 E 30 E 40 E 50 E 60 E 70 E 80 E 90 E 100 D D D C D D D D C D D D C X	FULL FULL FULL FULL FULL 50 40 30 NOT EMPTY 20 10 EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY
12 E 1 E 2 E 3 E 4 D D D D X	4 3 2 1
3 E 1 C E 2 C E 3 C E 4	NOT EMPTY NOT EMPTY FULL FULL FULL 3 2

C	1
D	EMPTY
D	
D	
D	
X	

The Prutor Editor (p1v2d1)

The Prutor Editor [20 marks]

Problem Statement
We have been using Prutor for a long time now. The Prutor has an excellent editor where we enter our code and edit it. We can enter any visible/whitespace characters, and also move left and right using arrow keys within and across different lines of our code. In this problem, we will implement a very very simple text editor.

The first line of your input will give you a strictly positive integer n denoting the number of operations to be performed. The next n lines will contain those operations. Each operation will be denoted by a single character and will be an instruction to either move the cursor one space to the left or one space to the right or else insert a character. The only characters we will give you are English alphabet characters A-Z, a-z and digits 0-9. Below are given rules on how to implement the editor

1. Initially the string will be empty and the cursor will be at index 0
2. If the instruction is the character '<' (without quotes), move the cursor one index to the left. If the cursor is already at index 0, it stays there.
3. If the instruction is the character '>' (without quotes), move the cursor one index to the right. If the cursor is already at the end of the current input, it does not move.
4. If the instruction is an alphabet or digit character, it gets inserted at the cursor position and the cursor moves one index to the right. However, be careful. If the cursor is at the end of the string, simply append the new character to the end of the string and move the cursor one position right. However, if the cursor is in the middle of the string, first shift all characters to the right of the cursor by one index to the right to create an empty location, then insert the new character at the cursor position, and then move the cursor one index to the right.

After every character insertion operation i.e. when the character is not < or >, print the current state of the string on a separate line after inserting the new character.

Caution

1. We assure you that the total length of the string will never exceed 99 characters.
2. Be careful about extra/missing lines and extra/missing spaces in your output.

EXAMPLE:
INPUT

7
a
<
<
b
c
>
d

OUTPUT:
a
ba
bca
bcad

Explanation: The following explains the location of the cursor and the reasoning after every instruction. The location of the cursor is denoted using the pipe symbol.

1. Initially the string is empty
2. Instruction 'a': a|
3. Instruction '<': |a
4. Instruction '<': |a (cursor cannot move any more left)
5. Instruction 'b': b|a
6. Instruction 'c': bc|a
7. Instruction '>': bca|
8. Instruction 'd': bcad|

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
7 a < b c > d	a ba bca bcad
8 a <	a ab

> > b c < d	abc abdc
4 a b c d	 a ab abc abcd
7 p < q < r < s	 p qp rqp srqp
8 a < t < c < > d	 a ta cta cdta
10 r s t < > < < > < 5	 r rs rst r5st

Finding your identity (p1v3d1)

Finding your identity [20 marks]

Problem Statement
In the first line of your input, you will be given two strictly positive integers n and m. In the next m lines, you will be given the n rows of an n x m matrix A, with each row on a separate line and two elements in a row separated by a single space. The matrix A will contain entries that are either 0 or 1.

In the first line of your output, print the size of the largest identity submatrix. In the next line, print the row index and column index of the top-left element of the largest identity submatrix in the format (rowIdx,colIdx). Note that there are no spaces in the output. If there is no identity submatrix, print "0"

(without quotes) in the first line of your output and print "(-1,-1)" (without quotes) in the second line of your output.

About Submatrices
Given a string, any contiguous set of characters occurring in that string is considered a substring of that string. Similarly we can extend that notion to submatrices as well. Given any matrix, every contiguous rectangle/square of elements inside that matrix is a submatrix of that matrix. E.g. consider the matrix
1 2 3 4
5 6 7 8
0 2 4 6

Then the following are submatrices of the above matrix. Note that submatrices may be square or rectangular.
Example 1
1

Example 2
6 7 8
2 4 6

Example 3
2 3
6 7

Example 4
1 2 3 4
5 6 7 8
0 2 4 6

Caution

1. Your code may be cleaner and easier to debug if you write a function to check for identity submatrices. This is not necessary though.
2. We assure you that there will always be a unique largest identity submatrix in the input given to you
3. You have to print the row index and column index of the top left element of the matrix - not the human readable notation of row number and column number. Remember, indices start from 0.
4. Be careful about extra/missing lines and extra/missing spaces in your output.

EXAMPLE:
INPUT
3 4
1 1 1 1
1 1 0 1
1 0 1 0

OUTPUT:
2
(1,1)

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are two lines in your output. Printing each line correctly, in the correct order, carries 50% weightage. Each visible test case is worth 2 points and each hidden test case

is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
3 4 1 1 1 1 1 1 0 1 1 0 1 0	2 (1,1)
4 5 1 1 1 1 1 1 0 0 0 1 0 1 0 0 0 0 0 1 1 0	3 (1,0)
6 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0	1 (3,4)
8 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 (-1,-1)
8 5 0 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1	4 (2,1)
6 6 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1	6 (0,0)

Queue (p2v1d1)

Queue [20 marks]

Problem Statement
The queue is a data structure that is incredibly useful in computer science. The Prutor server uses a queue to handle things when several students request compilation or evaluation at the same time. All these requests are put in a queue and the requests that came first are handled first and so on. Queues are also essential to the working of the internet as well as operating systems. However, the basic premise of a queue is extremely simple and all of you encounter queues all the time.

When we are in line to get a movie ticket, the line is also called a *queue*. Note that the person who came first to the ticket counter first is at the front of the queue and when the ticket master issues the next ticket, it is to the person at the front of the queue. People who come later are at the back of the queue and are serviced later. Queues are often called FIFO (first-in-first-out) structures since the first element (e.g. person in a line) that enters the queue is also the first to leave the queue.

The computer science queues work exactly the same way. The queues we will implement in this question will have three operations. The queues will have a limit size of MAX

1. Enqueue x: if there are already MAX elements in the queue, print the error message "FULL" (without quotes), otherwise put the element x at the back of the queue.
2. Dequeue: If the queue is empty, print the error message "EMPTY" (without quotes), otherwise remove the element at the front of the queue and print that element.
3. Check: If the queue is empty, print "EMPTY" (without quotes), else if it is full, print "FULL" (without quotes), else print "NOT EMPTY" (without quotes)

In the first line of the input, you will be give the value of MAX as a strictly positive integer. In the next several lines, you will see instructions on what to do

1. E x: enqueue the integer x into the queue or print the error message. There will be a single space between the character 'E' and the integer x.
2. D: dequeue the element at the front of the queue and print it or else print the error message
3. C: perform the check operation and print the appropriate message
4. X: no more operations to perform

Thus, the list of operations will be terminated by an X. All elements enqueued will be integers.

Caution

1. Print all error messages as well as dequeued elements on separate lines.
2. If you are using getchar() to process input character by character, be careful to read in the newlines at the end of each line as well.
3. We will not penalize you for extra newlines at the end of your output. However, do not have extra newlines in the middle of your output or else have trailing spaces in any line of the output.

HINTS:

1. Use an array with MAX elements to manipulate the queue. You will have to be a bit careful about how you manipulate the array since dequeue operations will leave empty slots at one end of the array. One

solution is to shift the array elements one place to the left after every dequeue operation and use a variable to keep track of how many elements are left in the queue.
2. Write functions to perform various queue operations to ensure your code looks neat and clean.

EXAMPLE:
INPUT
2
C
E 5
C
E 4
C
E 3
D
D
D
X

OUTPUT:
EMPTY
NOT EMPTY
FULL
FULL
5
4
EMPTY

Explanation

1. The queue only has a capacity of 2 elements.
2. Command 1: C - queue is empty at the moment so print EMPTY
3. Command 2: E 5 - enqueue 5. Now queue looks like [5]
4. Command 3: C - queue is neither full nor empty so print NOT EMPTY
5. Command 4: E 4 - - enqueue 4. Now stack looks like [5 4]
6. Command 5: C - queue is full so print FULL
7. Command 6: E 3 - queue is already full - cannot enqueue another element so print FULL - queue still looks like [5 4]
8. Command 7: D - dequeue front element 5 and print 5. Now queue looks like [4]
9. Command 8: D - dequeue front element 4 and print 4. Now queue is empty
10. Command 9: D - queue is already empty - cannot dequeue another element so print EMPTY - queue is still empty
11. Command 10: X - No more processing to do.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of

your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
2 C E 5 C E 4 C E 3 D D D X	 EMPTY NOT EMPTY FULL FULL 5 4 EMPTY
1 E 10 D E 4 D D E 4 E 5 C D D X	 10 4 EMPTY FULL FULL 4 EMPTY
10 E 1 E 2 E 3 E 4 E 5 E 6 D D C E 7 E 8 E 9 E 0 E 5 C E 6 C D D D D D D D D E 4	1 2 NOT EMPTY NOT EMPTY FULL 3 4 5 6 7 8 9 0 5 6 4 5 EMPTY

E 5 D D D D C X	
5 E 10 E 20 E 30 E 40 E 50 E 60 E 70 E 80 E 90 E 100 D D D C D D D D C D D D C X	FULL FULL FULL FULL FULL 10 20 30 NOT EMPTY 40 50 EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY EMPTY
12 E 1 E 2 E 3 E 4 D D D D X	1 2 3 4
3 E 1 C E 2 C E 3 C E 4 C D D D D X	NOT EMPTY NOT EMPTY FULL FULL FULL 1 2 3 EMPTY

The Prutor Editor Part II (p2v2d1)

The Prutor Editor Part II [20 marks]

Problem Statement
We have been using Prutor for a long time now. The Prutor has an excellent editor where we enter our code and edit it. We can enter visible or whitespace characters, move left and right using arrow keys within and across different lines of our code, as well as use delete and backspace to delete characters. In this problem, we will implement a very very simple text editor.

The first line of your input will give you a string with at most 99 characters. The string will be presented in a single line and may contain alphabet characters (upper and lower case), digits 0-9 and spaces. The next lines will each contain an instruction to either move the cursor left or right or delete a character in one of two ways described below.

Try to use the gets function to read in the string.

1. Initially the string will be what is give in the input and the cursor will be at the very end of the string i.e. if the string has n characters (n < 100), then the cursor will be at index n.
2. If the instruction is the character '<' (without quotes), move the cursor one index to the left. If the cursor is already at index 0, it stays there.
3. If the instruction is the character '>' (without quotes), move the cursor one index to the right. If the cursor is already at the end of the current input, it does not move.
4. If the instruction is the character ')' (without quotes), perform the "delete" operation i.e. delete the character at the cursor position. The cursor position stays the same. The length of the string decreases by one after this operation. If the string is already empty or if the cursor is at the end of the input, the delete operation does nothing.
5. If the instruction is the character '(' (without quotes), perform the "backspace" operation i.e. delete the character just before the cursor position. The cursor index decreases by one. The length of the string decreases by one after this operation. If the string is already empty or if the cursor is at index 0, the backspace operation does nothing.
6. If the instruction is the character '#', no more operations to perform. Do not print the string and simply exit the program.

After every delete or backspace operation i.e. when the character is not < or >, print the current state of the string on a separate line after performing the said operation.

Caution

1. We assure you that the total length of the string will never exceed 99 characters.
2. We will not penalize you for extra newlines at the end of your output. However, do not have extra newlines in the middle of your output or else have trailing spaces in any line of the output.
3. Be careful about extra/missing lines and extra/missing spaces in your output.

EXAMPLE:
INPUT

ABCD
)
(
<
(
#

OUTPUT:
ABCD
ABC
AC

Explanation: The following explains the location of the cursor and the reasoning after every instruction. The location of the cursor is denoted using the pipe symbol.

1. Initially the string is ABCD|
2. Instruction ')': No effect since cursor at the end already. String is still ABCD|
3. Instruction '(': Backspace. String is ABC|
4. Instruction '<': Move cursor left. String is AB|C
5. Instruction '(': Backspace. String is A|C
6. Instruction '#': Terminate

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
ABCD) (< (#	ABCD ABC AC
ABC < < < < ( )) #	ABC BC C

Hello ESC101 < <) > > > (((#	Hello ESC11 Hello ESC1 Hello ESC Hello ES
Hi Mr C) ((((< < <) #	Hi Mr C Hi Mr Hi Mr Hi M Hi i
123456890 < < < > > <))))) < < < (( ) #	12345680 1234568 1234568 1234568 1234568 123568 12568 1268
12345 < < < < <))) #	2345 345 45

Only Ones (p2v3d1)

Only Ones [20 marks]

Problem Statement
In the first line of your input, you will be given two strictly positive integers n and m. In the next n lines, you will be given the n rows of an n x m matrix A, with each row on a separate line and two elements in a row separated by a single space. The matrix A will contain entries that are either 0 or 1.

You are allowed to change the matrix given to you by swapping the values at any two positions in the matrix. You can also do any number of swaps in succession. For example, if the given matrix mat is
0 1 1 0
1 0 0 1
0 1 1 1
Then you can swap mat[0][1] with mat[1][1] to get
0 0 1 0
1 1 0 1
0 1 1 1
and then swap mat[0][2] with mat[1][2] to get
0 0 0 0
1 1 1 1
0 1 1 1

You have to find the size (in terms of number of elements) of the largest submatrix of 1s you can create using such swaps. Details about submatrices are given below. Remember, the submatrix should only contain 1s and no zeros.

In the first line of your output, print the number of 1s in the matrix. In the second line of your output, if this number is prime, print 1 else print 0. If the number of 1s is 0 or 1, print the number itself. In the third line of your output, print the size of the largest submatrix of 1s that can be formed using any number of swaps.

About Submatrices
Given a string, any contiguous set of characters occurring in that string is considered a substring of that string. Similarly we can extend that notion to submatrices as well. Given any matrix, every contiguous rectangle/square of elements inside that matrix is a submatrix of that matrix. E.g. consider the matrix
1 2 3 4
5 6 7 8
0 2 4 6

Then the following are submatrices of the above matrix. Note that submatrices may be square or rectangular.
Example 1
1

Example 2
6 7 8
2 4 6

Example 3
2 3
6 7

Example 4
1 2 3 4
5 6 7 8
0 2 4 6

Caution

1. You may need to do zero or more swaps to find the largest submatrix of 1s.
2. Be careful about extra/missing lines and extra/missing spaces in your output.

HINTS:

1. The first two lines of the output may help you with the final solution.

EXAMPLE:
INPUT
3 4
1 1 1 1
1 1 0 1
1 0 1 0

OUTPUT:
9
0
9

Explanation: Using two successive swaps, we can convert the matrix to
1 1 1 0
1 1 1 0
1 1 1 0
which presents to us the largest submatrix containing only 1s.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are three lines in your output. The first two lines are worth 6.25% each. The last line is worth 87.5% marks. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
3 4 1 1 1 1 1 1 0 1 1 0 1 0	9 0 9
4 5 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0	10 0 10

5 6 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0	5 1 5
10 2 1 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1	15 0 14
6 3 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 1	8 0 8
6 3 1 0 1 0 1 0 1 0 1 0 0 0 1 0 0 0 0 1	7 1 6

Graphs (p3v1d1)

Graphs [20 marks]

Problem Statement
Graphs are an integral component of computer science. The entire internet is basically one giant graph. Social networking websites like Facebook and Instagram are also, at their core, enormous graphs. A graph has nodes (also called vertices) and connections between nodes. In you have a Facebook account, you are a node in the FB graph and have connections to all your "friends" on FB. Nodes on a graph can communicate with each other.

In the first line of the input, you will be given the number of nodes in the graph as a strictly positive integer n. Nodes in the graph will be index from 0, 1, ... n-1. In the next lines, we will give you a list of nodes which sent messages (imagine an FB chat message) to other nodes. Thus, in each subsequent line, you will be given two non-negative integers i and j. This will imply that node i sent node j a message. The list will end when we give you i and j as -1.

In your output, you have to print four quantities

1. The number of nodes that did not send any messages at all
2. The number of nodes that did not receive any messages at all
3. The node that sent the most messages. If there are many such nodes, use the index of the node with smallest index.
4. The node that received the most messages. If there are many such nodes, use the index of the node with smallest index.

Caution

1. Nodes may send messages to themselves
2. One or more nodes may send no messages at all
3. See below on how to format your output. Be careful about spelling, case, and spaces.
4. Be careful about extra/missing lines and extra/missing spaces in your output.

HINTS:

1. You can use a 2D array to maintain the count of which node sent messages to which node and how many messages were sent.
2. If there are n nodes, then you can maintain an n x n array graph where graph[i][j] can store how many messages has node i sent to node j and graph[j][i] can store how many messages has node j sent to node i.

EXAMPLE:
INPUT
5
1 0
2 0
3 0
4 0
1 2
1 3
1 4
2 4
-1 -1

OUTPUT:
1 NODES SENT NO MESSAGES
1 NODES RECEIVED NO MESSAGES
MAX 4 MESSAGES SENT BY NODE 1
MAX 4 MESSAGES RECEIVED BY NODE 0

Explanation: Node 0 did not send any messages whereas node 1 did not receive any messages. Node 1 sent the maximum messages (4 messages) whereas node 0 received maximum messages (4 messages).

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are four lines in your output. Printing each line correctly, in the correct order, carries 25% weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
5 1 0 2 0 3 0 4 0 1 2 1 3 1 4 2 4 -1 -1	1 NODES SENT NO MESSAGES 1 NODES RECEIVED NO MESSAGES MAX 4 MESSAGES SENT BY NODE 1 MAX 4 MESSAGES RECEIVED BY NODE 0
4 1 3 1 0 1 2 -1 -1	3 NODES SENT NO MESSAGES 1 NODES RECEIVED NO MESSAGES MAX 3 MESSAGES SENT BY NODE 1 MAX 1 MESSAGES RECEIVED BY NODE 0
10 2 8 5 1 2 9 1 7 1 4 3 8 1 7 2 8 1 4 2 6 -1 -1	6 NODES SENT NO MESSAGES 4 NODES RECEIVED NO MESSAGES MAX 4 MESSAGES SENT BY NODE 1 MAX 3 MESSAGES RECEIVED BY NODE 8
10 -1 -1	10 NODES SENT NO MESSAGES 10 NODES RECEIVED NO MESSAGES MAX 0 MESSAGES SENT BY NODE 0 MAX 0 MESSAGES RECEIVED BY NODE 0
10 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 -1 -1	9 NODES SENT NO MESSAGES 9 NODES RECEIVED NO MESSAGES MAX 8 MESSAGES SENT BY NODE 4 MAX 8 MESSAGES RECEIVED BY NODE 4
15 14 13 13 12	0 NODES SENT NO MESSAGES 0 NODES RECEIVED NO MESSAGES

12 11 11 10 10 9 9 8 8 7 7 6 6 5 5 4 4 3 3 2 2 1 1 0 0 14 -1 -1	MAX 1 MESSAGES SENT BY NODE 0 MAX 1 MESSAGES RECEIVED BY NODE 0

How Mr C actually does Math (p3v2d1)

How Mr C actually does Math [20 marks]

Problem Statement
When you ask Mr C to evaluate a certain math expression with lots and lots of brackets and operators, he actually does not evaluate that expression as is. He first converts that expression into a format known as the *postfix* format and then evaluates the expression. An example of a postfix expression is

2 3 + 7 * #

The above expression is actually equivalent to the usual expression (2 + 3) * 7 and both expressions evaluate to 35. If you are wondering how the expression 2 + (3 * 7) (which is just 2 + 3 * 7 due to BODMAS rules) is represented in postfix, it is represented as 2 3 7 * +. The rules on how bracketed expressions are converted to postfix expressions will be taught to you in a more advanced course like ESO207 or CS345. However, below are given the rules of how postfix expressions are evaluated. Note that postfix expressions we looked at do not have any brackets. This is true of all postfix expressions - none of them need brackets.

Your post fix expression will be given in a single line in the input and will only consist of single digit non-negative integers, the characters + (addition), * (multiplication), and the termination symbol #. There will be a single space between every two characters in the input. You have to maintain an integer array of length 100 and follow the rules given below

1. Initially the array will be empty
2. If you encounter a digit symbol, simply insert that integer in the next available position in the array. Number of elements stored in the array goes up by one). Print the entire array on a single line of your output with a single space between two elements of the array.
3. If you encounter a math operator symbol i.e. + or *, take the integers in the last two positions in the array, erase them from the array (number of elements stored in the array down by two), perform the addition/multiplication operation with those two numbers, and then insert the result back into the next available position in the array (number of elements in the array goes up by one). Print the entire array on a single line of your output with a single space between two elements of the array. However, if there are less than 2 elements in the array to begin with, print "ERROR" and do not change the array at all.
4. If you encounter the # character, if there is exactly one element in the array, print that element. If there are no elements in the array or else if there are more than one elements left in the array, print "ERROR" (without quotes).

The postfix format may seem a bit funny to you but it comes with a huge advantage that no brackets are necessary in math expression which are in postfix format. You may also find it interesting that some languages like Sanskrit also operate in postfix format at times. The sentence "Ram and Sita went to the garden" translates to "Ram Sita cha udyaanam gachchatah". Note that the "and" operator appears in postfix notation in Sanskrit!

Caution

1. All your error messages or array element prints should be on separate lines. There should be no trailing spaces at the end of any line.
2. There will be no spaces after the # character. However, there will be a single space after every other character in the expression.
3. The results of the computations you do (addition/multiplication) may be numbers with more than one digit. However, the input will only contain single digit non-negative numbers.
4. Be careful about extra/missing lines and extra/missing spaces in your output.

HINTS:

1. Use a variable to keep track of how many elements are present in the array at any given point of time. Although your array is of size 100, most of the time you will have less than 100 elements in the array.

EXAMPLE 1:
INPUT
2 3 + 7 * #

OUTPUT:
2
2 3
5
5 7
35
35

Explanation Let us look at the characters and the state of the array after each operation

1. Initially array is empty []
2. Character 2 : Put 2 at index 0 since that is the first empty location. Array is now [2]
3. Character 3 : Put 3 at index 1 since that is the next empty location. Array is now [2 3]
4. Character + : Take out the last two elements in the array - array is now empty []. Put the sum of those two elements into the next available position in the array which is index 0 since array is now empty - array is now [5]
5. Character 7 : Put 7 at index 1 since that is the next empty location. Array is now [5 7]
6. Character * : Take out the last two elements in the array - array is now empty []. Put the product of those two elements into the next available position in the array which is index 0 since array is now empty - array is now [35]
7. Character # : There is only one element in the array so things are nice. Print that element 35 and exit

EXAMPLE 2:
INPUT
#

OUTPUT:
ERROR

EXAMPLE 3:
INPUT
1 5 + 3 #

OUTPUT:
1
1 5
6
6 3
ERROR

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
2 3 + 7 * #	2 2 3 5 5 7 35 35
1 5 + 3 #	1 1 5 6 6 3 ERROR
1 2 3 4 5 + + + + #	1 1 2 1 2 3 1 2 3 4 1 2 3 4 5 1 2 3 9 1 2 12 1 14 15 15
6 5 4 3 2 1 * * * * * #	6 6 5

	6 5 4 6 5 4 3 6 5 4 3 2 6 5 4 3 2 1 6 5 4 3 2 6 5 4 6 6 5 24 6 120 720 720
2 3 7 * + #	2 2 3 2 3 7 2 21 23 23
5 5 + 1 2 3 + + 2 3 2 5 2 * + * * * * * 1 + 0 * #	5 5 5 10 10 1 10 1 2 10 1 2 3 10 1 5 10 6 10 6 2 10 6 2 3 10 6 2 3 2 10 6 2 3 2 5 10 6 2 3 2 5 2 10 6 2 3 2 10 10 6 2 3 12 10 6 2 36 10 6 72 10 432 4320 4320 1 4321 4321 0 0 0

The Hidden Positives and Negatives (p3v3d1)

The Hidden Positives and Negatives [20 marks]

Problem Statement
The first line of the input will contain two strictly positive numbers m and n. The second line of the input will contain two strictly positive numbers p and q. The next m lines will contain the m rows of an m x n matrix A whose entries are only 0 or 1, with a single space between two entries of a row. The next p lines will contain the p rows of a p x q matrix B whose entries are only 0 or 1, with a single space between two entries of a row.

We suspect that the matrix B is hidden inside the matrix A as a submatrix, or else the complemented matrix B is hidden inside the matrix A as a submatrix (a helpful description of submatrices is given below). It may be the case that B and complemented B are both hiding inside A, possibly multiple times. For a matrix B, its complement is obtained by replacing every 0 by a 1 and every 1 by a 0. For example, the complement of the matrix
1 0 1
0 1 1
is the matrix
0 1 0
1 0 0

Report all instances where matrix B occurs either as itself, or in complemented form, as a submatrix of A, in the manner described below. To report an instance, report the row and column indices of the top left corner of the submatrix in the manner described below. Instances should be reported in increasing order of row and within a row, in increasing order of column of the top left corner of the submatrix, i.e. report all occurrences in the first row (if any) from left to right, then all occurrences in the second row (if any) from left to right, and so on.

About Submatrices
Given a string, any contiguous set of characters occurring in that string is considered a substring of that string. Similarly we can extend that notion to submatrices as well. Given any matrix, every contiguous rectangle/square of elements inside that matrix is a submatrix of that matrix. E.g. consider the matrix
1 2 3 4
5 6 7 8
0 2 4 6

Then the following are submatrices of the above matrix. Note that submatrices may be square or rectangular.
Example 1
1

Example 2
6 7 8
2 4 6

Example 3
2 3
6 7

Example 4
1 2 3 4
5 6 7 8
0 2 4 6

Caution

1. We will not penalize you for trailing newlines at the end of your output. However, make sure that there are no trailing spaces at the end of any lines in your output.
2. Take care of the capitalization and spacing in your output lines.
3. Be careful to report all occurrences in the first row (if any) from left to right, then all occurrences in the second row (if any) from left to right, and so on.

EXAMPLE 1:
INPUT
2 3

1 1
1 0 1
0 1 0
1

OUTPUT:
POSITIVE MATCH AT (0, 0)
NEGATIVE MATCH AT (0, 1)
POSITIVE MATCH AT (0, 2)
NEGATIVE MATCH AT (1, 0)
POSITIVE MATCH AT (1, 1)
NEGATIVE MATCH AT (1, 2)

Explanation: The matrix A is
1 0 1
0 1 0
and the matrix B is
1
and the complemented matrix B is
0

EXAMPLE 1:
INPUT
2 3
1 2
1 0 1
0 1 0
1 1

OUTPUT:
NO MATCHES

Explanation: The matrix B is
1 1
and it never occurs as a submatrix of A.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
2 4 1 2 1 0 1 1 1 0 1 0 1 0	POSITIVE MATCH AT (0, 0) NEGATIVE MATCH AT (0, 1) POSITIVE MATCH AT (1, 0) NEGATIVE MATCH AT (1, 1) POSITIVE MATCH AT (1, 2)

5 5 2 3 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0 1 1 0 0 0 0 0 1 0 1 1 1 1	POSITIVE MATCH AT (1, 1) NEGATIVE MATCH AT (3, 0)
5 6 2 4 1 0 1 0 1 1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 1 1 1 0 0 0 0 1 1	POSITIVE MATCH AT (1, 0) NEGATIVE MATCH AT (1, 2) NEGATIVE MATCH AT (2, 0) POSITIVE MATCH AT (2, 2) POSITIVE MATCH AT (3, 0)
5 7 2 2 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 0 0	POSITIVE MATCH AT (0, 5) NEGATIVE MATCH AT (1, 0) POSITIVE MATCH AT (2, 0) NEGATIVE MATCH AT (2, 5) NEGATIVE MATCH AT (3, 0) POSITIVE MATCH AT (3, 5)
5 7 2 2 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 0 0	POSITIVE MATCH AT (0, 5) NEGATIVE MATCH AT (2, 5) NEGATIVE MATCH AT (3, 0) POSITIVE MATCH AT (3, 5)
7 7 3 3 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1	POSITIVE MATCH AT (0, 0) POSITIVE MATCH AT (1, 1) POSITIVE MATCH AT (2, 2) POSITIVE MATCH AT (3, 3) POSITIVE MATCH AT (4, 4)

How Prutor Manages Memory (p4v1d1)

How Prutor Manages Memory [20 marks]

Problem Statement

Prutor gets tons of requests for memory allocation from you and your friends. These can be due to your programs declaring variables, static arrays or dynamic arrays. Prutor also has to free all this memory when either your programs end or else when you release dynamically allocated memory using the free() function.

Prutor handles such requests on a first-come-first-serve basis. This is very much like a queue (or line) for food in the mess - if you arrive first in the mess, you will be the first in the queue and when food is served, you will be the first one to receive food. In the first line of the input, you will be given a strictly positive integer MAX denoting the total amount of memory available to Prutor in bytes.

In the next several lines, you will see four different kinds of instructions which are outlined below. We assure you that we will not give you more than 100 instructions. However, there may be zero or more instructions. Each instruction will be given on a separate line.

1. Memory Allocation request: If the instruction line contains the character 'M' (without quotes) followed by a space followed by a strictly positive integer n, it is a request to allocate n bytes of memory. However, do not execute this instruction just yet - just store it somewhere.
2. Memory Release request: If the instruction line contains the character 'F' (without quotes) followed by a space followed by a strictly positive integer n, it is a request to release n bytes of allocated memory. However, do not execute this instruction just yet - just store it somewhere.
3. Request execution request: If the instruction line contains the character 'X' (without quotes), this is a request to execute the earliest execution stored with you that has not been executed yet. If there are no non-executed instructions stored with you, print "NO MORE INSTRUCTIONS" (without quotes), else execute the earliest instruction.
4. Termination: If the instruction line contains the character '#' (without quotes), this means that there are no more instructions to execute and you should end your program now.

Keep in mind the following rules while processing instructions.

1. If it is a memory allocation request and there are less bytes remaining than what is requested, print the error message "NOT ENOUGH MEMORY". Otherwise, reduce the number of available bytes by the amount of memory requested, and print the message "Y BYTES LEFT" (without quotes) where Y is the number of bytes left after performing the allocation operation.
2. If it is a memory release request and the request tries to release more memory than what has so far been allocated in total, print the error message "SEGFAULT". Otherwise, increase the number of available bytes by the amount of memory released, and print the message "Y BYTES LEFT" (without quotes) where Y is the number of bytes left after performing the free operation.

Caution

1. Do not execute allocation/release instructions the moment you receive them. Execute them only when you get the execute instruction.
2. Print all error messages as well as status messages on separate lines.
3. If you are using getchar() to process input character by character, be careful to read in the newlines at the end of each line as well.
4. We will not penalize you for extra newlines at the end of your output. However, do not have extra newlines in the middle of your output or else have trailing spaces in any line of the output.

5. The malloc/calloc/realloc/free process is much more careful than what we described above. However, to keep the question simple, we have not bothered you with all those details. With malloc/calloc/realloc/free, we cannot release arbitrary amounts of memory at arbitrary locations. We can only use free to release the entire chunk of memory allocated by a single malloc/calloc/realloc operation. Whereas you do not have to worry about these details in the question, do remember this when actually using malloc/calloc/realloc/free in your programs.

HINTS:

1. Use arrays with 100 elements to store instructions that have not been executed yet (there wont be more than 100 instructions). You can use a variable to store the location of the earliest instruction that has not been executed yet.
2. In order to differentiate between allocation and release instructions, you can maintain two arrays, one that stores the type of instruction 'M' or 'F' and the other that stores the amount of bytes in that instruction. Another neat way is to use a single array but store allocation instructions as positive integers and release instructions as negative integers.
3. Write functions to perform various queue operations to ensure your code looks neat and clean.

EXAMPLE:
INPUT
5
X
M 3
M 2
X
M 1
F 4
X
X
X
X
X
F 2
F 1
X
#

OUTPUT:
NO MORE INSTRUCTIONS
2 BYTES LEFT
0 BYTES LEFT
NOT ENOUGH MEMORY
4 BYTES LEFT
NO MORE INSTRUCTIONS
NO MORE INSTRUCTIONS
SEGFAULT

Explanation

1. There are a total of 5 bytes available. Instruction list is empty []
2. Command 1: X - no instructions to execute

3. Command 2: M 3 - instruction list is [(M,3)]
4. Command 3: M 2 - instruction list is [(M,3) (M,2)]
5. Command 4: X - execute instruction (M,3) - 2 bytes left - instruction list is [(M,2)]
6. Command 5: M 1 - instruction list is [(M,2) (M,1)]
7. Command 6: F 4 - instruction list is [(M,2) (M,1) (F,4)]
8. Command 7: X - execute instruction (M,2) - 0 bytes left - instruction list is [(M,1) (F,4)]
9. Command 8: X - execute instruction (M,1) - not enough bytes - instruction list is [(F,4)]
10. Command 9: X - execute instruction (F,4) - 4 bytes left - - instruction list is empty []
11. Command 10: X - no instructions to execute
12. Command 11: X - no instructions to execute
13. Command 12: F 2 - instruction list is [(F,2)]
14. Command 13: F 1 - instruction list is [(F,2) (F,1)]
15. Command 14: X - execute instruction (F,2) - but right now only 1 byte is allocated - segfault - instruction list is [(F,1)]
16. Command 15: #

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
5 X M 3 M 2 X M 1 F 4 X X X X X F 2 F 1 X #	NO MORE INSTRUCTIONS 2 BYTES LEFT 0 BYTES LEFT NOT ENOUGH MEMORY 4 BYTES LEFT NO MORE INSTRUCTIONS NO MORE INSTRUCTIONS SEGFAULT
5 M 3 X M 1 X M 2 X	2 BYTES LEFT 1 BYTES LEFT NOT ENOUGH MEMORY 0 BYTES LEFT 5 BYTES LEFT

M1 X F 5 X #	
5 M 3 X M 1 X M 2 X F 5 X F 1 X X #	2 BYTES LEFT 1 BYTES LEFT NOT ENOUGH MEMORY SEGFAULT 2 BYTES LEFT NO MORE INSTRUCTIONS
14 M 10 M 2 F 4 F 10 X X X F 8 M 3 X X #	4 BYTES LEFT 2 BYTES LEFT 6 BYTES LEFT SEGFAULT 14 BYTES LEFT 11 BYTES LEFT
8 M 9 M 8 F 8 M 5 M 7 M 1 X X X X X X #	NOT ENOUGH MEMORY 0 BYTES LEFT 8 BYTES LEFT 3 BYTES LEFT NOT ENOUGH MEMORY 2 BYTES LEFT
3 M 1 M 1 M 1 F 2 F 1 X X X X	2 BYTES LEFT 1 BYTES LEFT 0 BYTES LEFT 2 BYTES LEFT 3 BYTES LEFT

X #	

Message in the Matrix (p4v2d1)

Message in the Matrix [20 marks]

Problem Statement
In the first line you will be given two strictly positive integers m and n. In the next line you will given a string of length at most 100. The string will contain only lowercase English alphabets. In the next m lines, you will be given an m x n matrix of characters, with all elements being lowercase English alphabets, one row on each line. There will be no spaces between individual characters, nor any space at the end of any line.

You have to find whether the given string occurs in the matrix or not. The string can occur in the matrix in 3 ways - either left to right, or diagonally upwards, or diagonally downwards, as the example below suggests. Report all occurrences of the matrix by giving the row and column index of the first character of the occurrence. If there are multiple occurrences in the matrix, report them following the rules given below.

1. First, report all occurrences where the first character is in the first row, then all occurrences where the second row, and so on.
2. If there are multiple occurrences in a row, report them in increasing order of column index of the first character in the occurrence.
3. If there are multiple occurrences at a given index e.g. flat as well as diagonally up, then report the diagonally upward occurrence first (if present), then the flat occurrence (if present), then the diagonally downward occurrence first (if present).

Caution

1. Use scanf("%d %d\n", &m, &n); to read the size of the matrix so that you read in the newline at the end of the first line as well.
2. We will not penalize you for extra newlines at the end of your output. However, do not have extra newlines in the middle of your output or else have trailing spaces in any line of the output.
3. Use gets() to read the string. However, be careful if you are using getchar to read the matrix element by element since there will be newline characters at the end of each row which will need to be excluded.

EXAMPLE:
INPUT
3 3
ab
vbr
abq
cbd

OUTPUT:
FOUND AT (1, 0) DIAG UP

FOUND AT (1, 0) FLAT
FOUND AT (1, 0) DIAG DOWN

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
3 3 ab vbr abq cbd	FOUND AT (1, 0) DIAG UP FOUND AT (1, 0) FLAT FOUND AT (1, 0) DIAG DOWN
3 3 abc vbr abq cbd	NOT FOUND
3 3 aa aaa aaa aaa	FOUND AT (0, 0) FLAT FOUND AT (0, 0) DIAG DOWN FOUND AT (0, 1) FLAT FOUND AT (0, 1) DIAG DOWN FOUND AT (1, 0) DIAG UP FOUND AT (1, 0) FLAT FOUND AT (1, 0) DIAG DOWN FOUND AT (1, 1) DIAG UP FOUND AT (1, 1) FLAT FOUND AT (1, 1) DIAG DOWN FOUND AT (2, 0) DIAG UP FOUND AT (2, 0) FLAT FOUND AT (2, 1) DIAG UP FOUND AT (2, 1) FLAT
5 5 hiesc hdefe diefe deefe defse defec	FOUND AT (0, 0) DIAG DOWN
5 4 flat raft frat slit	FOUND AT (1, 0) DIAG DOWN FOUND AT (3, 0) DIAG UP

frat east	
6 6 raft master arootf bsafea craftd drestt breade	FOUND AT (1, 1) DIAG DOWN FOUND AT (3, 1) FLAT FOUND AT (4, 1) DIAG UP

The Hidden Key (p4v3d1)

The Hidden Key [20 marks]

Problem Statement
In the first line of the input you will be given a strictly positive integer n denoting the length of the messages. In the next two lines you will be given two messages as list of n integers separated by a single space. The first message will be the original (plain) message and the next will be the encrypted message.

The encrypted message was obtained by taking a secret key which is itself a message of length k (we do not know k) and adding it to the message. However, since the message length may be longer than the length of the key, the key is repeated as many times as required.

For example, if the plain message is [1 2 3 4 5 6 7 8] (i.e. n = 8) and the key is [1 2 3] (i.e. k = 3) then we first repeat the key enough to obtain a list of 8 elements as [1 2 3 1 2 3 1 2] (notice that we omitted the last 3 since we do not need it - we already have 8 integers). The encrypted message is now obtained by adding the repeated key message to the original message to obtain the encrypted message as [2 4 6 5 7 9 8 10]

In your output, you have to first give the length k of the secret key and then output the secret key. If more than one secret key is possible, output the secret key of the smallest length possible.

Caution

1. Secret keys may be of any non-negative length
2. As the example below indicates, the integers in the messages as well as in the keys, may be negative or even zero.
3. Be careful about extra/missing lines and extra/missing spaces in your output.

EXAMPLE:
INPUT
2
31 43
31 43

OUTPUT:
1
0

Explanation The plain and encrypted message are the same. The shortest key that makes this possible is the unit length key 0.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are two lines in your output. Printing each line correctly, in the correct order, carries 50% weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
2 31 43 31 43	1 0
10 223 15 22 175 179 92 5 30 185 249 323 215 322 275 379 392 105 230 485 349	3 100 200 300
5 1 2 3 4 5 5 4 3 2 1	5 4 2 0 -2 -4
15 181 65 122 124 7 231 67 130 128 201 214 90 119 135 128 114 153 215 39 19 32 155 47 219 221 17 130 218 220 148	15 -67 88 93 -85 12 -199 88 -83 91 20 -197 40 99 85 20
10 223 15 22 175 179 92 5 30 185 249 323 215 322 275 379 392 105 130 485 349	9 100 200 300 100 200 300 100 100 300
12 223 15 22 175 179 92 5 30 185 249 645 770 523 215 322 275 379 392 305 230 485 349 845 1070	6 300 200 300 100 200 300