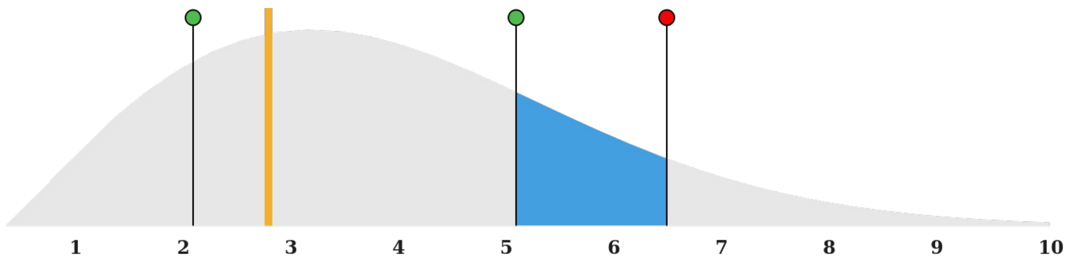
Mustafa S. Hamada

E-MAIL	TEST
mustafa.s.hamada@gmail.com	**Programming Aptitude Test**
DATE OF LAST ACTIVITY	LANGUAGES USED
2018-10-03	**C++, Python 3**

Technical Skill



○	Strongly recommended regarding technical potential.
●	Recommended regarding technical potential.
○	More information needed.
○	Not recommended if technical ability is crucial.

TASKS IN TEST	CORRECT	SUBMISSIONS	PLAGIARISM	LANGUAGE
EASY **Toilet Seat**	✓	3	✓	C++
MEDIUM **Virus Replication**	✓	9	✓	C++
HARD **Proving Equivalences**	✗	6	✓	Python 3

Kattis

EASY

Toilet Seat

SHOWN AS	MEMORY LIMIT	CORRECTNESS	PLAGIARISM
Problem A	**1024 MB**	✓ **Accepted**	✓ **Not detected**
CPU TIME LIMIT	LANGUAGE USED	ATTEMPTS	
1 second	**C++**	**3**	

Many potential conflicts lurk in the workplace and one of the most sensitive issues involves toilet seats. Should you leave the seat "up" or "down"? This also affects productivity, particularly at large companies. Hours each week are lost when employees need to adjust toilet seats. Your task is to analyze the impact different bathroom policies will have on the number of seat adjustments required.

Photo by Henry Stern (http://www.flickr.com/photos/artisticrichmond/2716147621/)

The classical assumption is that a male usually uses a toilet with the seat "up" whereas a female usually uses it with the seat "down". However, we will divide the population into those who prefer the seat up and those who prefer it down, regardless of gender.

Now, there are several possible policies that one could use, here are a few:

1. When you leave, always leave the seat up
2. When you leave, always leave the seat down
3. When you leave, always leave the seat as you would like to find it

So, a person may have to adjust the seat prior to using the toilet and, depending on policy, may need to adjust it before leaving.

Task

Your task is to evaluate these different policies. For a given sequence of people's preferences, you are supposed to calculate how many seat adjustments are made for each policy.

Input

The first and only line of input contains a string of characters 'U' and 'D', indicating that a person in the sequence wants the seat *up* or *down*. The string has length at least 2 and at most 1000.

The first character indicates the initial position of the toilet seat, and the following $n-1$ characters indicate how a sequence of $n-1$ people prefer the seat. You should compute the total number of seat adjustments needed for each of the three policies described above.

Output

Output three numbers, each on a separate line, the total number of seat adjustments for each policy.

Sample Input 1

```
UUUDDUDU
```

Sample Output 1

```
6
7
4
```

problemA.cpp

```cpp
/*
Author: Mustafa S. Hamada
Date: 2018-09-18
File name: problemA.ccp
*/

#include <iostream>
#include <string>

int main(int argc, char const *argv[])
{
    /* main running program */
    std::string input;        // defining 'input' as a string sequence of characters
    std::cin >> input;        // reading string sequence input from user console
    char initStat = input[0]; // index of the initial character in the string stream indicating the
'toilet seat status'
    int upState = 0;          // defining & initialising policy (1) to 0
    int downState = 0;        // defining & initialising policy (2) to 0
    int leaveState = 0;       // defining & initialising policy (3) to 0

    if (initStat != input[1]){
        upState += 1;
        downState += 1;
        leaveState += 1;
    }
    if (input[1] == 'D'){
        upState += 1;
    }
    if (input[1] == 'U'){
        downState += 1;
    }

    int precPer = input[1];
    for(int i = 2; i <= input.substr(2).length() + 1; i++){
        if (input[i] == 'D'){
            upState += 2;
        }
        if (input[i] == 'U'){
            downState += 2;
        }
        if (precPer != input[i]){
            leaveState += 1;
        }
        precPer = input[i];
    }


    std::cout << upState << std::endl;
    std::cout << downState << std::endl;
    std::cout << leaveState << std::endl;

    return 0;
}

```

MEDIUM

Virus Replication

SHOWN AS	MEMORY LIMIT	CORRECTNESS	PLAGIARISM
Problem B	**1024 MB**	✓ **Accepted**	✓ **Not detected**
CPU TIME LIMIT	LANGUAGE USED	ATTEMPTS	
1 second	**C++**	**9**	

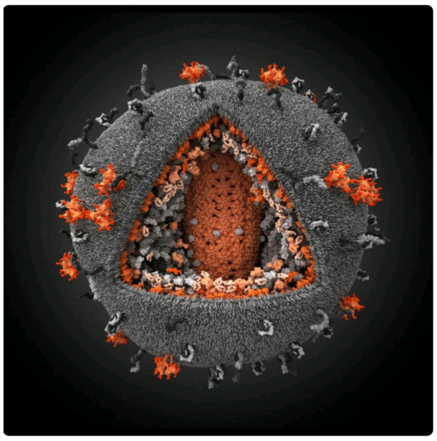


Image from Microbe World (http://www.flickr.com/photos/microbeworld/6217704321/)

Some viruses replicate by replacing a piece of DNA in a living cell with a piece of DNA that the virus carries with it. This makes the cell start to produce viruses identical to the original one that infected the cell. A group of biologists is interested in knowing how much DNA a certain virus inserts into the host genome. To find this out they have sequenced the full genome of a healthy cell as well as that of an identical cell infected by a virus.

The genome turned out to be pretty big, so now they need your help in the data processing step. Given the DNA sequence before and after the virus infection, determine the length of the smallest single, consecutive piece of DNA that can have been inserted into the first sequence to turn it into the second one. A single, consecutive piece of DNA might also have been removed from the same position in the sequence as DNA was inserted. Small changes in the DNA can have large effects, so the virus might insert only a few bases, or even nothing at all.

Input

The input consists of two lines containing the DNA sequence before and after virus infection respectively. A DNA sequence is given as a string containing between 1 and 10^5 upper-case letters from the alphabet {A, G, C, T}.

Output

Output one integer, the minimum length of DNA inserted by the virus.

Sample Input 1

```
AAAAA
AGCGAA
```

Sample Output 1

```
3
```

Sample Input 2

```
GTTTGACACACATT
GTTTGACCACAT
```

Sample Output 2

```
4
```

problem_B.cpp

```
/*
Author: Mustafa S. Hamada
Date: 2018-09-18
File name: problem_B.cpp
*/

#include <iostream>
#include <string>

int main(int argc, char const *argv[])
{
    /* main code */

    // define string DNA sequences
    std::string dnaSeq_1;
    std::string dnaSeq_2;
    // readin in DNA sequence from console
    std::cin >> dnaSeq_1;
    std::cin >> dnaSeq_2;
    // check if first DNA strand is a substring of the second DNA strand e.g. dnaSeq_1 = "AAA" &
dnaSeq_2 = "AAAAA"
    if (dnaSeq_2.find(dnaSeq_1) != std::string::npos) {
        std::cout << dnaSeq_2.length() - dnaSeq_1.length() << std::endl;
        return 0;
    }
    // firstBase counting for the first unmatched DNA nucleobase from the left (beginning of DNA
sequence)
    int firstBase = 0;
    while (firstBase < dnaSeq_1.length() && firstBase < dnaSeq_2.length() && dnaSeq_1[firstBase] ==
dnaSeq_2[firstBase]) {
        firstBase += 1;
    }
    // endBase counting for the first mismatched DNA nucleobase from the right (end of DNA
sequence)
    int endBase_1 = dnaSeq_1.length() - 1; // set the index values for dnaSeq_1
    int endBase_2 = dnaSeq_2.length() - 1; // // set the index values for dnaSeq_2
    while (endBase_1 >= 0 && endBase_2 >= 0 && dnaSeq_1[endBase_1] == dnaSeq_2[endBase_2]) {
        endBase_1 -= 1;
        endBase_2 -= 1;
    }
    if (endBase_2 < firstBase) {
        std::cout << "0" << std::endl;
    } else {
        // print out index difference
        std::cout << endBase_2 - firstBase + 1 << std::endl;
    }

    return 0;
}
```

HARD

Proving Equivalences

SHOWN AS	MEMORY LIMIT	CORRECTNESS	PLAGIARISM
Problem C	**1024 MB**	✗ **Wrong Answer**	✓ **Not detected**
CPU TIME LIMIT	LANGUAGE USED	ATTEMPTS	
5 seconds	**Python 3**	**6**	

Consider the following exercise, found in a generic linear algebra textbook.

> *Let A be an $n \times n$ matrix. Prove that the following statements are equivalent:*
>
> 1. *A is invertible.*
> 2. *$Ax = b$ has exactly one solution for every $n \times 1$ matrix b.*
> 3. *$Ax = b$ is consistent for every $n \times 1$ matrix b.*
> 4. *$Ax = 0$ has only the trivial solution $x = 0$.*

The typical way to solve such an exercise is to show a series of implications. For instance, one can proceed by showing that (a) implies (b), that (b) implies (c), that (c) implies (d), and finally that (d) implies (a). These four implications show that the four statements are equivalent.

Another way would be to show that (a) is equivalent to (b) (by proving that (a) implies (b) and that (b) implies (a)), that (b) is equivalent to (c), and that (c) is equivalent to (d). However, this way requires proving six implications, which is clearly a lot more work than just proving four implications!

I have been given some similar tasks, and have already started proving some implications. Now I wonder, how many more implications do I have to prove? Can you help me determine this?

Input

On the first line one positive number: the number of testcases, at most 100. After that per testcase:

- One line containing two integers n $(1 \le n \le 20\,000)$ and m $(0 \le m \le 50\,000)$: the number of statements and the number of implications that have already been proved.
- m lines with two integers s_1 and s_2 $(1 \le s_1, s_2 \le n$ and $s_1 \ne s_2)$ each, indicating that it has been proved that statement s_1 implies statement s_2.

Output

Per testcase:

- One line with the minimum number of additional implications that need to be proved in order to prove that all statements are equivalent.

Sample Input 1

```
2
4 0
3 2
1 2
1 3
```

Sample Output 1

```
4
2
```

problem_C__final.py

```python
import sys
visited = []
order = []

sys.setrecursionlimit(50000)


def dfs(node):
    global visited, order

    visited[node] = True
    if node in graph:
        for neighbour in graph[node]:
            if visited[neighbour]:
                continue
            dfs(neighbour)

    order.append(node)


def load_graph():

    input_line = input()
    inputs = [int(s) for s in input_line.split()]

    vertices_count = inputs[0]
    edges_count = inputs[1]
    graph = {}

    for vertex in range(vertices_count):
        graph[vertex + 1] = []

    for _ in range(edges_count):
        input_line = input()
        inputs = [int(s) for s in input_line.split()]
        neighbours = graph[inputs[0]]
        neighbours.append(inputs[1])
        graph[inputs[0]] = neighbours

    return graph


def reverse_graph(graph):

    graph_rev = {}
    for x in range(len(graph)):
        graph_rev[x+1] = []

    for first in graph.keys():
        for neighbour in graph[first]:
            current = graph_rev[neighbour]
            current.append(first)
            graph_rev[neighbour] = current

    return graph_rev

if __name__ == "__main__":

    test_cases = int(input())
    for _ in range(test_cases):


        graph = load_graph()
        visited = [False] * (len(graph) + 1)
        order = []

        for node in graph.keys():
            if visited[node]:
                continue

            dfs(node)

        graph_rev = reverse_graph(graph)

        visited = [False] * (len(graph_rev) + 1)
        result = [0] * (len(graph_rev)+1)

        for index in range(len(order), 0, -1):

            start_node = order[index-1]
            if visited[start_node]:
                continue

            stack = [start_node]
            while len(stack) > 0:
                current = stack.pop()
                if visited[current]:
                    continue

                result[current] = start_node
                visited[current] = True

                for neighbour in graph_rev[current]:
                    stack.append(neighbour)

        dag = {}

        sinks = set()
        sources = set()

        for x in range(len(graph)):

            vertex = result[x+1]
            sources.add(vertex)
            sinks.add(vertex)
            neighbours = dag.get(vertex, set())
            for neighbour in graph[vertex]:
                if result[neighbour] != vertex:
                    neighbours.add(neighbour)

            dag[vertex] = neighbours

        if len(dag) == 1:
            print(0)
            continue

        for key, value in dag.items():
            if len(value) > 0:
                sinks.discard(key)
                for source in value:
                    sources.discard(source)

        print (max(len(sinks), len(sources)))
```