# Algorithms and Data Structures 2021: Second Practical Assignment

November 26, 2021

## 1 Instructions

The practical assignment consists of two parts, a program and a report.

You must write your program in Java, C++, C or Python 3. You can only use libraries that come with a minimal installation of the language. You must write all code yourself; copying code from fellow students or the internet is considered plagiarism.[1].

In your report, you must explain your algorithm, analyse the correctness and analyse the complexity. The report must have **at least two and at most ten** pages. Include your names on the first page of the report.

The deadline for the first practical assignment is **Friday, January 7th, 16:00**. You should submit your code and report via Brightspace. You are allowed to work on the assignment in pairs. Then you must be enrolled in the same practical group on Brightspace. Only one team member has to submit the assignment.

## 2 Grading

Grades will be determined as follows. You may earn up to 100 points for your solution:

- 20 points for the explanation of your algorithm.
- 10 points for the correctness analysis.
- 10 points for the complexity analysis.
- 50 points for the test results. See section 4.1 for more (important) information.
- 10 points for the quality of the code.
- 10 points bonus for further analysis. See section 3.4 for some suggestions.

The grade is the total number of points divided by 10. If you have questions, do not hesitate to contact Jelmer Firet, `jelmer.firet@ru.nl`.

[1]We will check this using MOSS. Note that MOSS results are available without logging in. Therefore we will not require you to put your name or student number in your code.

## 3 Folding

Frank has bought a long strip of stamps for all the Christmas cards he wants to send. To make the strip more compact, he folded them in half until it formed a single pile of stamps. Frank unfolded the pile at home. He noticed a surprising pattern when he looked at the strip from the side: the dragon curve. With his curiosity piqued, Frank asked his friend to design other foldable creatures.

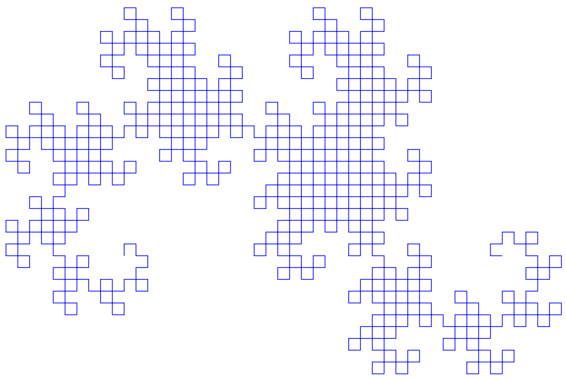

Figure 1: The order 10 dragon curve. Author: António Miguel de Campos

His friend produces a list of crease patterns for the creatures he designed. In a crease pattern, each crease between stamps is either a mountain fold or a valley fold. A mountain fold indicates that the strip folds down; the bottom of the strip will face itself after the fold. A valley fold means that the strip folds up; the top of the paper will face itself.

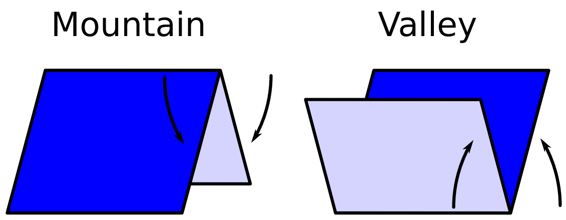

Figure 2: Visualisation of a mountain and a valley fold. Author: Jelmer Firet

We will assume that a strip of $n$ stamps is $n$ cm long. When Frank wants to fold the strip, he first chooses a place to fold: a crease between two stamps. These creases are at $1, 2, \ldots, n-1$ cm from the left side of the strip. Then he will fold the left side of the crease to the right side. There are two ways to make this fold: "left over right" or "left under right". After a fold at crease $i$ the remaining strip will be $\max(i, n-i)$ cm long.

To fold the order $k$ dragon curve, you start with a strip of $2^k$ stamps. Then you repeatedly fold the crease in the middle (left over right) until you have a single pile of stamps. The first fold is $2^{k-1}$ cm from the left, the next fold $2^{k-2}$ cm from the new left, etcetera. Due to the halving nature of this procedure, you can fold the $2^k - 1$ creases of the order $k$ dragon curve in $k$ folds. Frank is curious about how many folds he needs to make the patterns of his friend.

We will assume the stamps are infinitely thin. We can fold at a crease regardless of how many layers of stamps we have at that crease. Furthermore, we will always fold all layers at a crease. Suppose we have the pattern $\mathtt{M}_1\mathtt{V}_2\mathtt{M}_3\mathtt{M}_4\mathtt{V}_5$. Then we are not allowed to fold in the order $\mathtt{V}_2, \mathtt{M}_4, \mathtt{V}_5$ since $\mathtt{M}_1$ would be folded with $\mathtt{V}_5$ but in the wrong direction. Finally, we don't unfold before every crease has a fold. So we can't fold $\mathtt{V}_2$, unfold it again and then fold $\mathtt{M}_4, \mathtt{V}_5, \mathtt{M}_1$.

## 3.1 Input

Your program should read a single string $S$ from standard input. Each character in $S$ is either an 'M' or a 'V'. You may assume that $S$ has at most 1000 characters.

## 3.2 Output

Print a single integer to standard output, the minimum number of folds Frank needs to produce the given crease pattern.

## 3.3 Examples

Sample input 1 (the dragon curve):

```
VVMVVMMVVVMMVMM
```

Sample output 1:

```
4
```

Sample input 2 (the humble square):

```
VVMMMVMVVVMVMMMVV
```

Sample output 2:

```
9
```

Sample input 3 (the special spiral):

```
MMMMMMMMMMMMMMM
```

Sample output 3:

```
15
```

Sample input 4 (the acyclic accordion):

```
MVMVMVMVMVMVMVMV
```

Sample output 4:

```
16
```

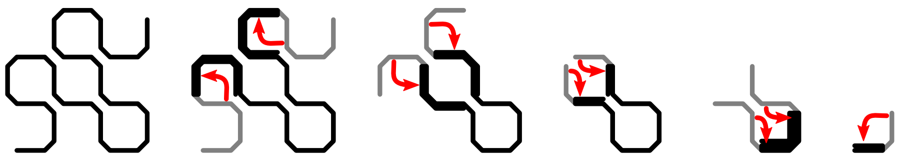

Figure 3: Solution for sample input 2. Author: Jelmer Firet 🅭🄍
Note: Instead of folding a flat strip of paper, the visualisation folds up the final pattern.

### 3.4 Challenge

Solving the problem above is enough for a good grade, but we encourage you to analyse the task further. Below are a few suggestions (which may not have easy answers).

Can your algorithm solve test cases with more than 1000 creases? How large?
What if the pattern has creases that should not have a fold?
What if Frank may fold "right over left" but not "left under right"?
What if Frank may choose to fold some layers? (instead of always folding all layers)
What if Frank may unfold before all creases have a fold?

## 4 Testing your code

You can test your program using DOMJudge (`https://domjudge.science.ru.nl`). You should have received credentials from Jelmer via mail (the same as the first practical). You should ensure your program works in DOMJudge, as we will use it to judge your solution. During the last days before the deadline, the test server generally receives more submissions. Therefore the server will be slower in judging your program. To prevent this, you should start testing early.

You can also test your code locally. You can download the test cases from DOMJudge and feed them to your program. Redirect the `.in` file to the standard input of your program and compare the output with the `.ans` file.

```
./program <1.in >1.out && diff -bq 1.out 1.ans
```

### 4.1 Test results

We will run your program on a fresh set of test cases. You will get points for every test case where your program produces the correct answer within the time limit. If your code does not compile or does not read and write via `stdin` and `stdout`, you will get zero points on the test cases. So please test your code on the test server! If you pass all test cases on the test server, you can assume you will get most of the points for the test results. If your program also finishes all test cases within a fraction of the time limit, you will likely get all points. Ask for help early if you encounter any issues!