Algorithms and Data structures 2016: First assignment

1 Instructions

You may work in groups of two. For this assignment you have to hand in two things:

- Source code. We will run the code for grading.
- A report. In the report you have to explain the algorithm and analyse it.

Source code You are allowed to submit a solution in either C, C++ or Java. If you prefer another language, please contact us. You are only allowed to use the Standard Library corresponding to your selected language.

You will find a set of examples to test your solution on the course wiki. We will set up a website, on which you can upload your code. It will be automatically be tested against test cases.

Report Besides handing in code, we would like to receive a report in which you explain your algorithm and analyse its correctness and runtime complexity.

Submission The deadline for sending in your solution is on November 11. You must submit your solutions via Blackboard. Only one team member has to submit a solution; the names and student numbers of both team members must be mentioned in the report.

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

- 15 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. We will be running several tests and you will get points for every 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 well!
- 15 points for the quality of the code.

If you have any questions, do not hesitate to send a email to Joshua (or someone else related to the course).

2 Red-Black Mobiles

Helene participates in the Algorithms and Data Structures course at Radboud University. She is intrigued by the cover picture of the course textbook, which shows a mobile designed by the American sculptor Alexander Calder. She shows the picture to her friend Vincent, a student at the ArtEz University of the Arts in Arnhem. They find out that Alexander Calder has become famous for his mobiles, and that he designed his first mobile following a visit to the studio of the Dutch artist Piet Mondriaan. The mobile displayed on the cover of Introduction to Algorithms is known under the name Big Red and was created in 1959. Another version of Calder's signature work Big Red was constructed ten years later and offered for 2.000.000-3.000.000 dollar at Christie's in 2014. [1]

Helene and Vincent believe they can make a lot of money by starting an on-line shop in which they sell similar mobiles of varying sizes. In their business meeting they decided to use two materials, one heavy (metal, red) and one light-weight (plastic, black). Shortly after, Vincent took off to make a couple of these mobiles. Helene was very impressed by his results but saw that they were not quite balanced nicely. She believes that the best way of balancing these mobiles is by counting the heavy leaves in both branches and compare the counts (note that a rod in a mobile always has exactly two things attached to it: on both ends either a leaf or another mobile). She states that the difference in those counts may be at most one. Of course this property should hold in all the sub mobiles found in the sculpture as well. Since the black leaves and the rods are light-weight, their counts do not matter. Vincent agrees with her approach.

Now the problem is that Vincent already made a couple of these mobiles and he does not want to discard all his work, as it is very time consuming. So he wants fix the mobiles he made, by swapping red and black leaves inside one mobile, leaving the overal structure of the rods fixed. Can you help them to compute the minimal number of swaps Vincent has to perform to make the mobile balanced, if possible?

2.1 Input

Your program will be run for each mobile separately. The mobiles will be given via `stdin` (your program should not open any files) in the following recursive format:

- `B` denotes a black, light-weight leave,
- `R` denotes a red, heavy leave, and
- `(`$m_1 m_2$`)` denotes a rod with two sub mobiles m_1 and m_2 (again, represented in the same format).

The number of rods is bounded by 1500.

The string will contain no whitespace, except for a new line at the end (as well as an `EOF` token).

[1] The sculpture can be viewed in motion here: `https://youtu.be/Z82Z7kvio40`.

2.2 Output

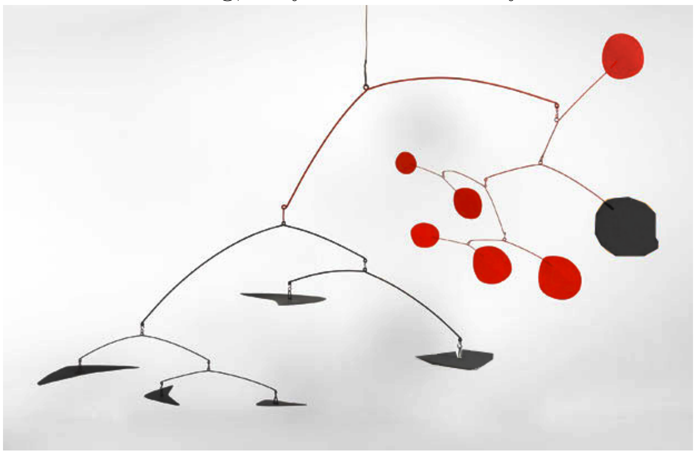
The output (via `stdout`) should contain a single integer: the minimal number of swaps needed to make the mobile balanced. If it is not possible to balance the mobile, output "`discard`" (without the quotes).

We will check exactly with this specification. Any other output (besides whitespace) will be regarded as a wrong output.

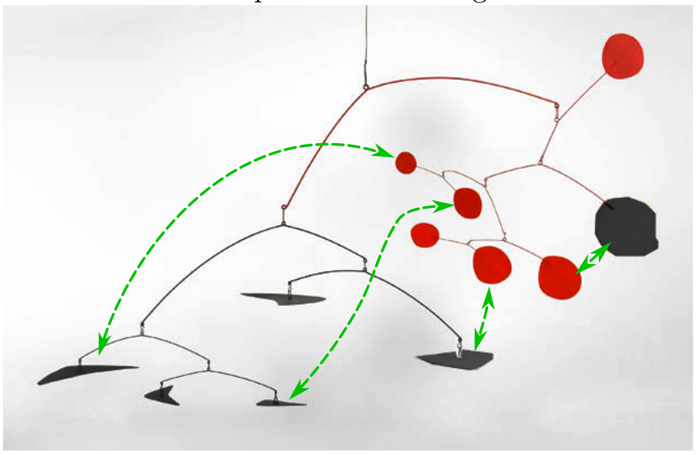
2.3 Examples

- `(BR)` $\mapsto$ `0`
- `(R((RB)(RR)))` $\mapsto$ `discard`
- `(((B(BB))(BB))((((RR)((RR)R))B)R))` $\mapsto$ `4`

The last example is also depicted in the following sculpture (by Alexander Calder, so it doesn't actually need re-balancing, only the imitation by Vincent needs re-balancing):



One can see that four swaps is indeed enough to make it balanced:



Try to convince yourself that four is also the minimal number of swaps.