Institut für
Pervasive Computing

Algorithms and Data Structures 2
Winter term 2021

Assignment 01

Deadline: **Wed 3.11.2021, 23:59**
Submission via: **Moodle**

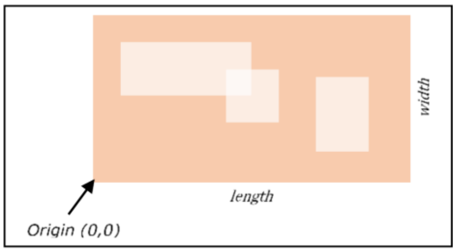
Time log

Remember the time you needed to implement your solution of this assignment and log it in the exercise specific survey in Moodle! This information is fully anonymous.

Random Numbers

1. Monte Carlo Integration 10+2 points

Given an **enclosing** rectangle (orange) of size `length x width` (with the origin in the bottom-left corner at location `(0,0)`) and a variable number of **embedded** rectangles (white) of different sizes in it, which can also **intersect** with each other (see the figure below as an example with 3 embedded rectangles).
Compute the **area** of the **enclosing** rectangle that is **not covered** by any of the **embedded** rectangles. For this make us of the **Monte Carlo Integration** method presented in the exercise, given a specified **sampling** density (number of random points).
Remember: The Monte Carlo method tests if random points are located inside a particular area and uses the **ratio of hits versus the number of total points** to **estimate** the extent of an enclosed area. In this case, a "*hit*" is a point in the enclosing rectangle that is not inside any of the embedded rectangles.



To generate a random sequence of points use the method `random()` of the class `random`, which returns a random floating point number in the range `[0.0, 1.0)`, and scale that to the required range (of the enclosing rectangle).
Hint: A simple multiplication should do the trick.

a) **Implement** the class `MonteCarlo` using the provided skeleton and interface, as well as the class `Rectangle` to create rectangle objects.
   - The class `MonteCarlo` defines the size of the enclosing rectangle, as well as the embedded rectangles in its constructor.
   - The shape and location of the embedded rectangle is defined in the dedicated class `Rectangle`.

b) Write a **main method** for testing the following scenario: The outer rectangle has a size of **100x30**, and there is one embedded rectangle which covers **exactly half of the area**. Do the test with 10, 100, 1000 and 100000 random points to evaluate quality of the estimated result.

```
class MonteCarlo:

    def __init__(self, length, width, rectangles):
        """constructor

        Keyword arguments:
        :param length -- length of the enclosing rectangle
        :param width -- width of the enclosing rectangle
        :param rectangles -- array that contains the embedded rectangles
        """

    def area(self, num_of_shots):
        """Method to estimate the area of the enclosing rectangle that is not covered by the embedded rectangles

        Keyword arguments:
        :param num_of_shots -- Number of generated random points whose location (inside/outside) is analyzed
        :return float -- the area of the enclosing rectangle not covered.
        :raises ValueError if any of the parameters is None
        """

    def inside(self, x, y, rect):
        """Method to determine if a given point (x,y) is inside a given rectangle

        Keyword arguments:
        :param x,y -- coordinates of the point to check
        :param rect -- given rectangle
        :return bool
        :raises ValueError if any of the parameters is None
        """
```

2. Linear Feedback Shift Register 12 points

Linear Feedback Shift Registers (LFSR) are often used in cryptography. The *A5 algorithm* for instance was used as an encryption algorithm for the first GSM standards for mobile phones. For this exercise we use a slight variation of that algorithm based on three LFSRs.
In fig. 1 the structure of the random number generator is illustrated, after the registers have been initialized, except the placeholders $D_1..D_8$. These have to be filled with digits of your personal student ID (according to fig. 2).

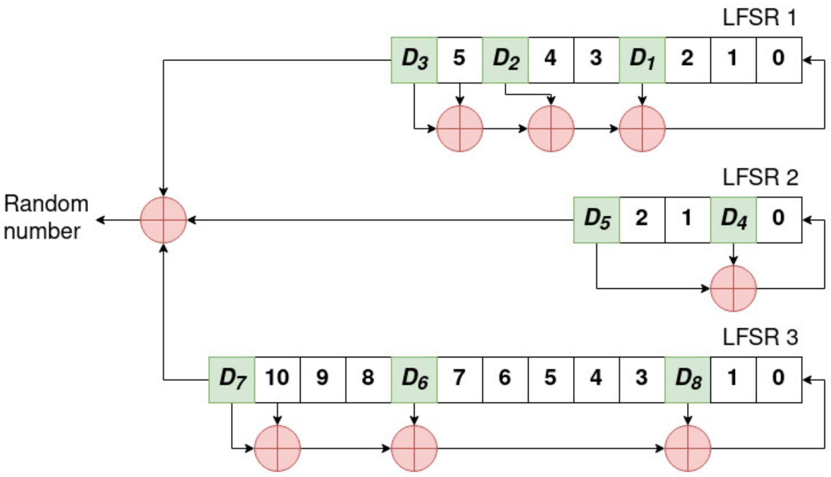

LFSR 1
D_3 5 D_2 4 3 D_1 2 1 0
LFSR 2
D_5 2 1 D_4 0
LFSR 3
D_7 10 9 8 D_6 7 6 5 4 3 D_8 1 0
Random number

Fig. 1: Structure of the LFSR random number generator with register contents after initialization.

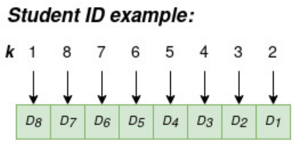


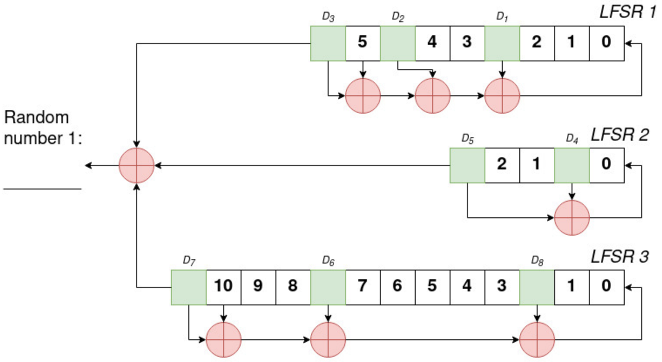
Fig. 2: Mapping of a student ID to placeholders $D_1..D_8$

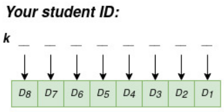
Algorithm procedure:

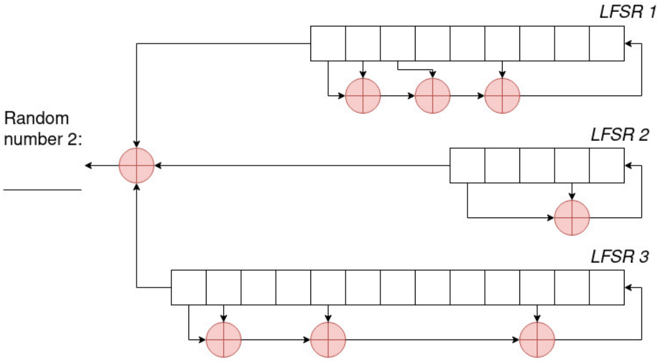
1. Link the **leftmost value of each register** with **XOR** ⊕ (eXclusive OR) to calculate the new random number.
2. Calculate the new register values to be inserted on the right end, by linking the appropriate elements again with **XOR**.
3. Finally, shift register contents of all LFSRs to the left and insert the calculated register values from step 2.

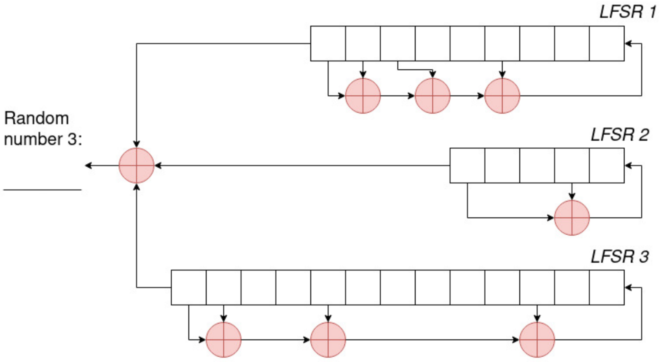
For this exercise you have to execute the generator using **pen & paper**. To do so you will find the empty LFSRs on the next page, where you have to **fill in** the **register values** and the **generated random number**. Do this for the first 5 iterations!

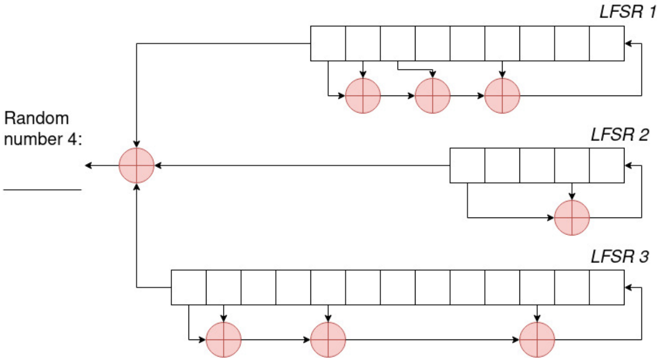
Submission:
A PDF containing the first 5 iterations of your personal random number generator. You can print the LFSR skeleton on the next page and scan your final solution, or you can also fill in the skeleton digitally.

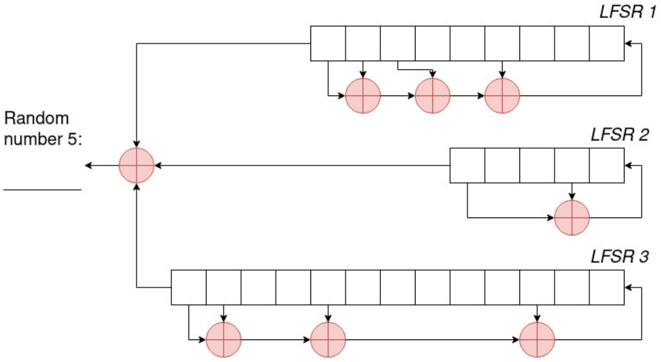

LFSR 1
D3
D2
D1
5
4
3
2
1
0
Random number 1:
LFSR 2
D5
D4
2
1
0
LFSR 3
D7
D6
D8
10
9
8
7
6
5
4
3
1
0


Your student ID:
k
D8
D7
D6
D5
D4
D3
D2
D1


LFSR 1
Random number 2:
LFSR 2
LFSR 3


LFSR 1
Random number 3:
LFSR 2
LFSR 3


LFSR 1
Random number 4:
LFSR 2
LFSR 3


LFSR 1
Random number 5:
LFSR 2
LFSR 3