CMPUT 331, Fall 2023, Assignment 3

Before beginning work on this assignment, carefully read the Assignment Submission Specifications posted on eClass.

You will submit four files for this assignment: "a3.pdf" or "a3.txt" for problems 1 and 3, "a3p2.py" for problem 2, "a3p4.py" for problem 4, and a README file.

Some problems in this assignment require you to solve for key variables (e.g. a and b), or to write code which does so. Consult your textbook and the lecture videos on the details of modular arithmetic and computing a modular inverse. **No credit will be given for "brute force" solutions, i.e. solutions which simply try various possible values until a solution is found. To receive credit, you, and your code, must find the key variables in an efficient, mathematical way. Please do not use print in your code!**

Problem 1

Short answer: Consider a text made up of symbols from a symbol set containing 71 elements, each corresponding to a unique integer from 0 to 70, encrypted with the affine cipher, with keys a and b encrypting each plaintext character p according to the formula $p \cdot a + b \pmod{71}$. Suppose we know that '52' is enciphered as '6', '20' is enciphered as '51', and '4' is enciphered as '38'. Find the keys a and b mod 71. Include your solution, including all relevant work and explanation, in your a3.pdf.

Problem 2

Random numbers are an integral component of cryptography. Alice knows random numbers are difficult to generate efficiently. In order to securely communicate with Bob, she is trying to use pseudorandom number generators (PRNGs), algorithms which produce sequences of *pseudorandom* numbers, which appear random, but which, with some extra (typically hidden) information, can be predicted. Alice found an algorithm which uses a recurrence relation similar to the affine cipher's encryption function:

$$R_{i+2} = (aR_{i+1} + bR_i + c) \pmod{m} \qquad i \geq 0$$

where a, b, c, m, R_1, and R_0 are chosen in advance. For reference, a, b, and c are called the "keys", m is called the "modulus", and R_0 and R_1 are called "seeds". For example, if we run this algorithm with $a = 3$, $b = 5$, $c = 9$, $m = 16$, $R_0 = 11$ and $R_1 = 6$, then $R_2 = 2$ since $(3 \cdot 6 + 5 \cdot 11 + 9) = 82 \equiv 2 \pmod{16}$, and $R_3 = 13$ since $(3 \cdot 2 + 5 \cdot 6 + 9) = 45 \equiv 13 \pmod{16}$.

Create a module "a3p2.py" containing a function "random_generator(a, b, c, m, $r0$, $r1$, n)", where a, b, c, and m are as above, and $r0$ and $r1$ are the seeds values of R_0 and R_1, which **returns a list of integers** containing the next n elements of generated numbers (R_2, R_3, ..., R_{n+1}). The following demonstrates invocation sequences that should run without error and produce identical output:

```
 >> random_generator(3, 5, 9, 16, 11, 6, 3)
[2, 13, 10]

>> random_generator(22695477, 77557187, 259336153, 9672485827, 42, 51, 8)
[4674207334, 3722211255, 3589660660, 1628254817,
8758883504, 7165043537, 4950370481, 2261710858]
```

```
>> random_generator(2**31-5, 743, 549, 2**64 - 1, 97, 101, 7)
[216895920563, 4611839799731136226, 4610466323469147181,
6003130118321022275, 14149176448272843437, 14229211546334517160,
10244020959373064815]

>> random_generator(1128889, 1023, 511, 222334565193649, 65535, 329, 8)
[438447297, 50289200612813, 17962583104439, 47361932650166,
159841610077391, 19587857129781, 111993173627854, 7567964632208]
```

Problem 3

Short answer: According to the given algorithm in problem 2, Alice started to generate some random numbers with $m = 467$, generates the numbers $R_2 = 28$, $R_3 = 137$, $R_4 = 41$, $R_5 = 118$, and $R_6 = 105$. Help Eve to predict next random numbers by determining the values of a, b, c, R_0, R_1 and R_7. Include the values of these **six** variables, with all relevant work and explanation for how you found them, in your a3.pdf or a3.txt.

Problem 4

In problem 3, Eve was able to "hack" Alice's algorithm by obtaining a, b, and c and thus predict all of its future output (you predicted R_7, but of course, since a, b, and c are fixed, you could predict any quantity of future values). In this problem, you will automate this process.

Write a module "a3p4.py" containing a function "crack_rng(m, *sequence*)", where m is a **prime number**, and *sequqnce* is list of five consecutive numbers which are generated by the generator (i.e. $[R_2, R_3, R_4, R_5, R_6]$) and they're all between 0 and $m-1$, inclusive. This function **must return a list of integers** $[a, b, c]$, where a, b, and c are the keys for an the provided random generator at problem 2, with modulus m, which outputs five consecutive numbers such as input sequence. It is guaranteed that a and b are non-zero.

Hint: While not advisable in practice, it is perfectly valid for our algorithm to have $c = 0$, and still have $a, b \neq 0$. Your crack_rng function *must be able to crack RNGs with* $c = 0$, $a, b \neq 0$, still returning the three-element list $[a, b, c]$. We guaranteed that there is a unique solution for a, b, and c.

```
 >> crack_rng(17, [14, 13, 16, 3, 13])
[3, 5, 9]

>> crack_rng(9672485827,[4674207334,3722211255,3589660660,1628254817,8758883504])
[22695477, 77557187, 259336153]

>> crack_rng(101, [0, 91, 84, 16, 7])
[29, 37, 71]

>> crack_rng(222334565193649,
        [438447297,50289200612813,17962583104439,47361932650166,159841610077391])
[1128889, 1023, 511]
```