KH4061CEM Programming and Algorithms 1
Coursework Report

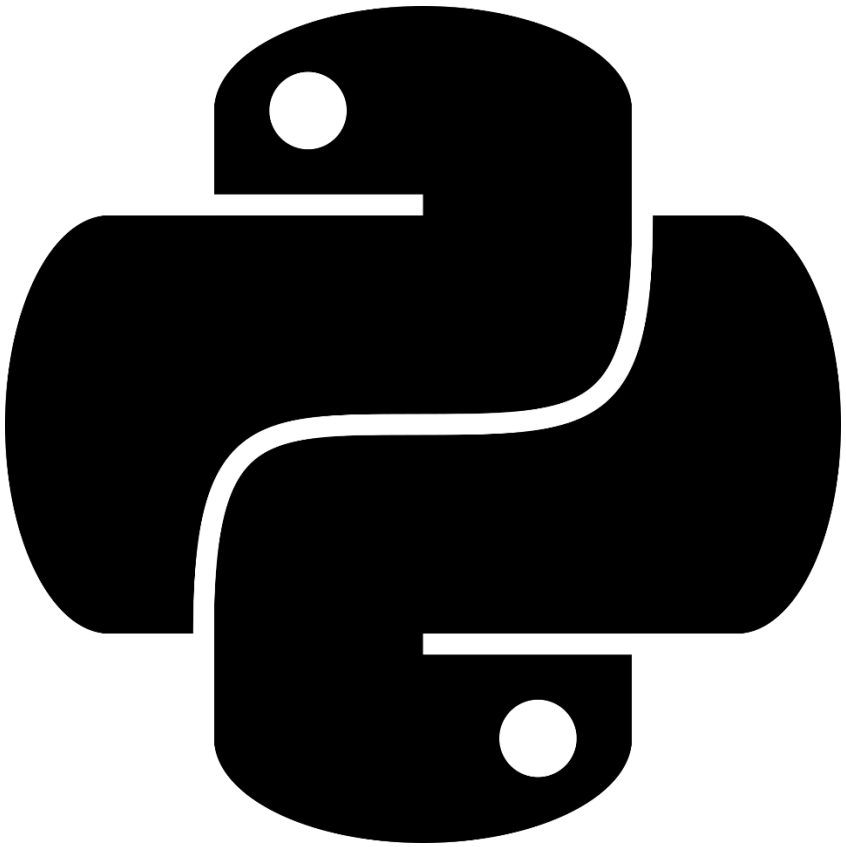
KH4061CEM Programming and Algorithms 1
Coursework Report

COURSEWORK 1: PRNG ALGORITHMS

Ali Mohamed Abdelhamid 2100274

Linear Congruential Generator

4/11/2021



KH4061CEM Programming and Algorithms 1
Coursework Report

Contents

KH4061CEM Programming and Algorithms 1
Coursework Report

Introduction

Linear congruential generator (LCG) consists of an algorithmic system that contains an array of pseudo-random numbers determines using an incoherent piecewise linear equation, the algorithm serves as one of if not the oldest, most efficient, and easiest pseudo-random number generator equation; This resulted due to the effortless and swift implementation they offer, specifically on computerized devices which offer 'modular arithmetic by storage-bit' truncation. I chose this code on the basis of both trying to implement a fairly easy code to start with and to learn about one of the most efficient and known random number generators. My code correctly utilizes the LCG by using this equation `rand = (a*rand + c) % m` all of the mentioned values (rand, a, c, m) are outputted by the code as shown here

```
a = 1140671485
c = 128201163
m = 2**24
```

except the rand which is a result of changing the main function defined earlier in the code into the system time, the system line is then printed to show the user the seed used while calculating this equation. To use this program the user first needs to install Spyder (a programming language compiler) they then need to open the code file called "LCG", the only thing required after that is to press run and answer whether they require a TRN (true random number generator) or a pseudo number generator by entering TRN, pseudo, or file. Depending on their answer they are prompted to enter a seed number then the number of lines of random numbers, enter the file name desired for the program to scan for the seed then the number of lines desired, or directly enter the number of lines of random numbers, the resulting output is a list of random numbers based on either the system time or the provided seed from either a file or a user input.

Instructions for Running the Program

There aren't many requirements to run or edit this program, the only essential requirement is installing a programming compiler, the recommended one is Spyder, which is actually the one used to write this program. You should then open the code file called "LCG" and pressing run/ debug on your compiler, you will be then prompted to answer a simple question asking whether you require a LCG with TRN or an LCG that depends on your seed input, the input should be strictly 'TRN' or 'pseudo' and no other, any other answer will result in a "Please enter a valid string" message and the termination of the code. If you type in TRN you will be presented with a line displaying the current seed depending on the system time which I will explain further in the description. Next, you'll be asked to enter the number of random number lines that you'd like to have, this is purely for freedom of choice and does not affect the equation what so ever, but the input should be a whole number integer, for example, 194; any other input will result in failure of the code, you will then be presented with a number of random numbers organized in separate lines, the number of lines of course will depend on the number you entered when proposed. On the other hand, if you enter "pseudo" you will be directly prompted to enter the number of random lines you require, the input should also be restricted to only whole numbers as failing to do that will also cause total code failure and you'll have to relaunch the code. Whether you choose to run the TRN or the pseudo code, both outputs will be similar, a designated number of lines of random numbers; the only difference is that in the TRN the code has more complexity since it has an extra variable which is the system time, the result will be more unpredictable, therefore, called a true random number generator (TRN). The last code (Open file code) functions similarly to the input code, as the only difference is that it takes the seed input from a designated file instead of a user input; if you enter 'file' you will be presented with a message asking you to choose from three file names, 'seed.txt', 'seed2.txt', 'seed3.txt', 'seed4.py', and 'seed5.py'. It is crucial that while entering the name of the file that you

specify the file extension, in this instance the only two extensions available are '.txt' and '.py'. Failing to meet the requirements to correctly run the program will result in the program not running as intended or not running at all.

Unit Tests

	LCG with TRN comparison between 15 different value iterations						
Conditions	Seed (Rand)	Increment (C)	Multiplier (A)	Modulus (M)	variance	Mean	Range of outputs
Even values	1635964645.6696918	128201162	1140671484	2**24	0.11313782	0.47323667208354	0.0377655029296875-0.19762253761291504
Odd values	1635965212.7908945	128201163	1140671485	2**24	0.089736455	0.42466484	0.004246947673904822-0.9288676350097274
Even and Odd	1635965393.5884278	128201162	1140671485	2**24	0.066396115	0.41709848	0.03934621810913086-0.9535390138626099
Prime values	1635965825.8403263	1140671509	1140671489	2**24	0.084197153	0.50198321	0.06255908667798477-0.9165520422614922
Prime, even, and odd	1635965967.3839734	1140671509	1140671485	2**24	0.092861156	0.54043435	0.019973039627075195-0.9199174642562866

KH4061CEM Programming and Algorithms 1
Coursework Report

	LCG comparison between 15 different value iterations						
Conditions	Seed (Rand)	Increment (C)	Multiplier (A)	Modulus (M)	variance	Mean	Range of outputs
Even values	15	128201162	1140671484	2**24	0.056065506	0.42430928	0.05012786388397217-0.9018086194992065
Odd values	15	128201163	1140671485	3**24	0.065004792	0.31021983	0.013023437250329678-0.8736289672365728
Even and Odd	15	128201162	1140671485	2**24	0.11499483	0.50548649	0.013992846012115479-0.9775537848472595
Prime values	15	1140671509	1140671489	2**24	0.11929944	0.50879997	0.027472972869873047-0.9457581639289856
Prime, even, and odd	15	1140671509	1140671485	2**24	0.083006563	0.62505249	0.08244484663009644-0.9881559014320374

Description of work

To begin with, the code starts with implementing a built-function called ‘sys’, this function doesn’t server much purpose other than to end the code after an incorrect user input; I then proceed to define the ‘initial’ term by asking the user to choose a certain keyword in order to determine the code that is going to be used, the choices are ‘TRN’, ‘pseudo’, and ‘file’. Furthermore, after entering the keyword one of the codes starts to initialize, for example, let’s start with the first one ‘TRN’, the fundamental element of this code is that it uses the system time as a seed to generate the random numbers, so the

first line imports the 'time' function, this acts as a time reader and it converts it into an integer that the compiler can read; the second line sets the term 'seconds' as the time value, I then decided to add the third print line since I was faced with an issue during the unit test table, that I wouldn't know the current seed since time is constantly changing. I then define the 'initial Val' as a function, proceeded with the globalization of the identifier 'rand' which is actually the number that is then changed into 'initial Val', the initial Val is later converted into the system time, so basically it becomes a full loop. I then define a second function called 'LCG', this function contains the main equation that I've placed in the introduction, but basically, the function states that the multiplier term is equal to '1140671485', the increment term is equal to '128201163', and the modulus term is equal to '2**24' which means 2 to the power of 24, and the role of each of these number in the equation, rand(system time, initial value) = (A(multiplier term)*rand + C(increment term)) % M(modulus term); the next line of code returns the final rand value over the modulus term again, this shows the complexity that in return causes this generator to be called a random number generator, as it's so unpredictable. The user is then asked to enter the number of lines he requires to be printed, and the 'for I in range(xi)' just defines that repetition, the last line just prints the LCG function which is the end code. The next code isn't that much different from the previous one as it has the same principal components, the only difference is that there is no 'import time' at the start of the code and that in line 63 the user is asked to enter a seed number, this input is defined as (seedinput), and then returned in the (seedLCG) function instead of the system time. The third code only has one difference from the previous one, it used a user input to determine which file is going to be read and use the seed inside to perform the LCG equation.

Source Code

GitHub repository: https://github.com/AliElFekki/Course-Work.git

KH4061CEM Programming and Algorithms 1
Coursework Report

```
#imports the 'sys' built in function in order to use sys.exit at the end if neither inputs were correct
import sys
#asks the user to input either trn or pseudo to decide the code going to be used
initial = str(input("Please choose whether you want a TRN or a pseudo-generator by entering 'TRN' and 'pseudo': "))
#initalizes the first if condition that determines which code to use depending on the users input
if (initial=='TRN'):
    #Imports the time function
    import time
    #This returns the number of seconds passed since the start of the program as an int
    seconds = time.time()
   #Outputs the number of the seed which in this case is the time
    print("This is the time value", seconds)
    #Defines the Seed LCG
    def seedLCG(initVal):
        #Sets the "rand" as a global function
        global rand
        #sets rand as the initial val, as initial val is later set as the seconds value
        rand = initVal
    #defines the numbers of the random equation
    def lcg():
        a = 1140671485
        c = 128201163
        m = 2**24
        #sets rand as a global function
        global rand
        #Sets rand = the equation of the random number
        rand = (a*rand + c) % m
        #returns rand over the modulus parameter
        return rand/m
    #Sets the rand as the system time
    seedLCG(seconds)
    #Sets the iterations of random numbers resulted
    xi=int(input("How many lines of random numbers do you want?: "))
    for i in range(xi):
        #Prints the initially defined function above
        print(lcg())
    '''
    M = Modulus parameter
    A= Multiplier term
    C= Increment term
    Rand= Seed
    '''
#sets an elif function where the user inputs the pseudo code
elif (initial=='pseudo'):
    #same as the code above defines the initial function to be called upon later
    def seedLCG(initVal):
        #sets the rand as a global function
        global rand
        #sets the rand as initial value which is later changed to the seed inputed by the user
        rand = initVal
        #defines the second function which is the equation itself
    def lcg():
        a = 1140671485
        c = 128201163
        m = 2**24
        #sets rand as a global function in the second lcg
        global rand
        #sets rand = the lcg equation
        rand = (a*rand + c) % m
        #returns rand over the modulus
        return rand/m
    #asks the user to input a number to be used as the seed
    seedinput=int(input("Please enter a seed number: "))
    #sets the seedLCG which is also the initval which has been set to rand as the seed
    seedLCG(seedinput)
    #asks the user for the number of lines he/she wants
    xii=int(input("How many lines of random numbers do you want?: "))
    for i in range(xii):
        #prints the lines of random numbers
        print(lcg())
    '''
    M = Modulus parameter
    A= Multiplier term
    C= Increment term
    Rand= Seed
    '''
else:
    #uses the else command to tell the user to enter one of specified strings and terminates the code
    print("Please enter a valid string")
    #terminates the code/ exits
    sys.exit()
```

KH4061CEM Programming and Algorithms 1
Coursework Report

References

1- Holton, A. G. (2016, July 4). *Linear congruential generators - Monte Carlo method*. Value. Retrieved November 4, 2021, from https://www.value-at-risk.net/linear-congruential-generators/.

2- *Linear Congruential Generator*. Linear congruential generator - Rosetta Code. (n.d.). Retrieved November 4, 2021, from https://rosettacode.org/wiki/Linear_congruential_generator.

3- Linear Congruential random number generator. (n.d.). Retrieved November 4, 2021, from https://asecuritysite.com/encryption/linear?val=2175143%2C3553%2C10653%2C1000000.

4- *Use of algorithm linear congruential generator (LCG) on ...* (n.d.). Retrieved November 4, 2021, from https://robots.iopscience.iop.org/article/10.1088/1757-899X/536/1/012151/meta.

5- 262588213843476. (n.d.). *Python implementation of the LCG (linear congruential generator) for generating pseudo-random numbers.* Gist. Retrieved November 4, 2021, from https://gist.github.com/oss6/8085178.

6- Anthony. (2019, November 3). *Random number generator and linear congruential generator(lcg) using python*. Welcome...enjoy coding? Electronics? C is beautiful, Python is brilliant. Retrieved November 4, 2021, from https://coding-engineer.com/2019/10/28/random-number-generator-python/.

7- *Linear congruence method for generating pseudo random numbers*. GeeksforGeeks. (2021, July 17). Retrieved November 4, 2021, from https://www.geeksforgeeks.org/linear-congruence-method-for-generating-pseudo-random-numbers/.

8- Published by adpoe View all posts by adpoe, adpoe, P. by, Adpoe, & adpoe, V. all posts by. (2016, August 25). *Experimenting with linear congruential generators in Python*. tonypoer.io. Retrieved November 4, 2021, from https://tonypoer.io/2016/03/23/experimenting-with-linear-congruential-generators-in-python/.