Problem 2: RandoHMMM Numbers

Problem 2: PseudoRandoHMMM Number Generation

In this problem, you'll create a new HMMM program that generates pseudorandom numbers.

Here is some starter code:

```
00 read r1    # input a
01 read r2    # input c
02 read r3    # input m
03 read r4    # input
X_0
04 read r5    # input N
```

The next sections first explain these inputs, and then explain how to generate pseudorandom numbers with them.

The Math Behind (Pseudo-)Random Number Generation

A linear congruential generator (LCG) is a "pseudorandom-number generator" algorithm that generates a sequence of numbers that "look and feel" like random numbers. The numbers generated are not truly random because they are generated by a mathematical formula, but they have statistical properties that make them behave like true random numbers.

The LCG algorithm is defined by the recurrence relation:

$X_{n+1} = (a X_n + c) \% m$

where:

- m is a divisor (whose remainder is preserved)
- a is a multiplier
- c is an increment
- X_0 is a "seed value," which is between 0 and m (excluding m)

Typically, the user is asked to enter the seed value, X_0.

An LCG random-number generator then uses the above formula to compute X_1, which is the first "pseudorandom" number.

After that, X_2 is computed from X_1, and so on forever (or until we have enough pseudorandom numbers for our needs!).

Notice that the pseudorandom numbers generated this way are always between `0` and `m-1` because we are "modding" our numbers by `m`. Mod already exists in HMMM (you don't need to write it!).

Since the sequence of numbers produced depends only on X_0 and the generator's parameters, the maximum period (how many numbers are generated before the sequence repeats) of the LCG is at most `m` (why?). If the LCG actually *has* period `m`, then it is said to have **full period**. This is a desirable property for a random number generator since it means that it generates many different numbers before repeating!

Part 1: Writing the RandoHMMM Number Generator

Your first job is to implement the LCG algorithm in HMMM! Your program should work as follows: the user will input **five** values in the following order (please use this order, since your program will be graded by providing inputs in the same order):

- First, the user enters the number `a`, the multiplier in the LCG algorithm.
- Second, the user enters the number `c`, the increment in the LCG algorithm.
- Third, the user enters the number `m`, the modulus divisor in the LCG algorithm.
- Fourth, the user enters the seed, X_0, in the LCG algorithm.
- Fifth, the user enters a number `N`, indicating the number of pseudorandom numbers that should be printed.

The HMMM program should then print the `N` pseudorandom numbers, beginning with X_1 (X_0 is not considered one of the pseudorandom numbers and is not printed.)

Extend the `random` starter code provided above to the full random-number generator as just described.

Note that **`mod`** is built-in to HMMM! **Don't write mod yourself!**

You will find you need to copy one register into another. The
```
copy r4
r8
```
command copies the contents of register `r8` **into** register `r4`.

Caution! the `copy` command is right-to-left!

Checking your generator

To check that your random-number generator is working, try running it with the following inputs:

- First, enter the number `a = 10`, the multiplier in the LCG algorithm.
- Second, enter the number `c = 7`, the increment in the LCG algorithm.
- Third, enter the number `m = 11`, the modulus in the LCG algorithm.
- Fourth, enter the seed, X_0 `= 3`, the starting value in the LCG algorithm.
- Fifth, enter the number `N = 10`, indicating that 10 pseudorandom numbers should be printed.

The output should then be ten alternating `4`s and `3`s:

```
4
3
4
3
4
3
4
3
4
3
```

Clearly, these are not good values for our random-number generator—they are not very "random"! The next section will ask you to choose much better values.

Part 2: Picking Parameter Values for the LCG

In this part you will choose "good" values for the parameters `a`, `c`, and `m` in the LCG algorithm. You should do this *by hand*, guided by the constraints below:

It turns out that the LCG algorithm has its best-possible performance—that is, it generates `m` different values before repeating—if the following three conditions are met:

- Condition 1: `c` and `m` are relatively prime (that is, `c` and `m` have no common divisors other than 1)
- Condition 2: (`a`-1) is divisible by all *prime factors* of `m` (not *all* factors of `m`, all *prime factors* of `m`)
- Condition 3: (`a`-1) must be a multiple of 4 if `m` is a multiple of 4

Your boss at SPRANG Corp. (Spam-Processed RAndom Number Generation) has asked you to construct a random number generator with `m` equal to 100. Find the smallest values of `a` and `c` that can be used with this value of `m` and satisfy these three conditions.

Place a comment at this point of your code indicating the values that you found for `a` and `c`.

Part 3: Writing `unique`

Your boss has also asked you to *check* to be sure all of the values from your LCG program are unique.

This check, unlike the random number generator, can use Python!

So, write a Python function `unique(L)` in a new trinket.

Your function `unique(L)` will take a list as its only input and should return `True` if that list has only unique elements (no elements repeated) and `False` otherwise.

Your function only needs to use recursion, list indexing, and slicing. You may find Python's `in` operator useful to check if `L[0]` is `in` the rest of the list `L`!

To test `unique`, simply run the file and then try a few inputs, e.g.,

```
>>> unique([1, 2, 3])
True

>>> unique([2, 42, 3,
42])
False
```

Part 4: Generating and Testing Your 100 Values

Now, we'll use your `unique` function to help determine whether your LCG program with the parameter values that you computed for `a` and `c` really give you full period when you run with `m` set to 100. To that end, run your HMMM LCG program with those parameter values. Use any seed you want from 0 to 99. Also, make the fifth input (`N`) equal to 100, in order to get 100 numbers of output.

Then, check if those those 100 numbers are unique. Fortunately, we have the `unique` program that does that. Unfortunately, `unique` takes as its argument, **but** the output that we have on our screen is not a list (no commas and no brackets at the beginning and end).

No worries! First, copy-and-paste the following bit of Python code into your trinket, underneath `unique`:

```
NUMBERS = """
103
102
104
101
105
"""

def test( S ):
    """ test accepts a (triple-quoted) string, S,
        containing one number per line. Then, test
        returns True if those numbers are all unique
        (or if S is empty); otherwise it returns
False
    """
    ListOfStrings = S.strip().split()
    # print "ListOfStrings is", ListOfStrings
    ListOfIntegers = [int(s) for s in ListOfStrings]
    # print "ListOfIntegers is", ListOfIntegers
    return unique(ListOfIntegers)
```

There are two things in the code above: first, there is a triple-quoted string named `NUMBERS` that contains a few integers.

Second, there is a small function named `test` that accepts a string (`S`), which certainly can be a triple-quoted string, and then uses `unique` to determine whether all of the integers in that triple-quoted string are unique.

The commented-out print statements are there in case you'd like to get a bit more intuition about how `test` works.

Copy-and-paste the 100 numbers that were output by your HMMM program into the triple-quoted string named `NUMBERS` (replacing the current contents of that string).

Then, run

```
test(NUMBERS)
```

at the bottom of your file.

If all goes well, Python will output `True`.

Once you've checked your program to verify that it is correctly giving you 100 different values, this problem is complete!

Submit Your Values of a and c

5.0/5.0 points (graded)

Enter the values you obtained for `a` and `c` below. Recall that you were asked to provide the **smallest** values of `a` and `c` that satisfy the given conditions.

`a`
`=`

correct

21

`c`
`=`

correct

1

You have used 2 of 3 attempts Some problems have options such as save, reset, hints, or show answer. These options follow the Submit button.

Submit Homework 7, Problem 2 HMMM Code

10.0/10.0 points (graded)

To submit your Homework 7, Problem 2 HMMM code, you'll need to copy it from the trinket above and paste it into the box below. After you've pasted your code below, click the "Check" button.

Remember, this is the **HMMM** code!

IMPORTANT: Make sure that there aren't spaces at the beginning of your code, and that you copied all of the characters. If there are extra spaces or you are missing spaces, our server won't be able to run your code and we won't be able to give you any of the points you deserve for your hard work.

1

```
# PSEUDORANDOHMMM NUMBER
GENERATION
```

2

```
00 read r1          # input
a
```

3

```
01 read r2          # input
c
```

4

```
02 read r3          # input
m
```

5

```
03 read r4          # input
X_0
```

6

```
04 read r5          # input
N
```

7

```
05 mul r4 r1 r4     # r4 =
r1*r4
```

8

```
06 add r4 r4 r2     # r4 = r4 +
r2
```

9

```
07 mod r4 r4 r3     # r4 = r4 %
r3
```

10

```
08 write r4         # print what's in
r4
```

11

```
09 addn r5 -1        # r5 = r5 -
1
```

12

```
10 jgtzn r5 05       # if r5 > 0 jump to line
03
```

13

```
11 halt              #
stop
```

14

15

```
# m = 100, a = 21, c =
1
```

Press ESC then TAB or click outside of the code editor to exit
correct

correct

Test results
CORRECT See full outputSee full output

You have used 2 of 3 attempts Some problems have options such as save, reset, hints, or show answer. These options follow the Submit button.

Submit Homework 7, Problem 2 Python Code

5.0/5.0 points (graded)

To submit your Homework 7, Problem 2 Python code, you'll need to copy it from the trinket above and paste it into the box below. After you've pasted your code below, click the "Check" button.

Remember, this is the **Python** code!

IMPORTANT: Make sure that there aren't spaces at the beginning of your code, and that you copied all of the

characters. If there are extra spaces or you are missing spaces, our server won't be able to run your code and we won't be able to give you any of the points you deserve for your hard work.

1

```
def unique(L):
```

2

```
  """return True if that list has only unique elements
```

3

```
  (no elements repeated) and False
otherwise."""
```

4

```
  if L ==
[]:
```

5

```
    return
True
```

6

```
else:
```

7

```
    return (L[0] not in L[1:]) and
unique(L[1:])
```

8

9

Press ESC then TAB or click outside of the code editor to exit
correct

correct

Test results
CORRECT See full outputSee full output

You have used 1 of 3 attempts Some problems have options such as save, reset, hints, or show answer. These options follow the Submit button.