Project 1 Morally Due Feb 26

This project is worth 10% of your grade. But ALSO, if you score higher on this project then you did on hw01 then we will REPLACE your hw01 grade with this project grade.

In Java:

hw01grade = max(hw01grade,proj01grade)

In Pascal:

hw01grade := max(hw01grade,proj01grade)

In Pseudocode:

hw01grade $\leftarrow$ max(hw01grade,proj01grade)

For this project, you will write a program that will demonstrate the behavior of a DFA classifier.

1 Input

Your program will receive a sequence of command-line arguments that dictate the output that your program should provide. The parameters corresponding to these arguments are specified below.

1. b - a base-10 number given by the 1st argument. This represents the base of the numerical input that your DFA classifier will receive.

2. m - a base-10 number given by the 2nd argument. This represents the modulo value corresponding to your DFA classifier.

3. i - a base-10 number given by the 3rd argument. See part 3 of the project for details.

4. x - a sequence of base-10 numbers given by the remaining supplied arguments. Each number k in this sequence represents a base-b digit (so $0 \leq k < b$). x will represent the sequence of base-b digits that will be supplied *in right to left order* to your DFA classifier. NOTE: There may be multiple arguments corresponding to this parameter.

2 Your task

Your program will print a sequence of information to standard output demonstrating that your program works properly. There will be three parts to this project, which are given below.

1. Recall that the weights for the given input are given by the sequence of values

   $$b^0 \pmod m, b^1 \pmod m, \ldots$$

   This sequence has associated with it both an initial segment and a repeated segment. So in the repeating sequence $1, 0, 1, 2, 3, 7, 1, 2, 3, 7, \ldots$, the initial segment would be $[1, 0]$ and the repeated segment would be $[1, 2, 3, 7]$.

   For this part of the project, output on one line the initial segment of the sequence of weights, followed by the repeated segment on the next line. Each segment should be formatted as a sequence of base-10 numbers in list form. That is, your sequence of values should be bounded by square brackets, and consecutive values should be separated by commas. (See examples in next section.)

2. For this part of the project, output a table representing the DFA classifier that, given a number y in base b, read right to left, is in a state k where $y \equiv k \pmod m$.

   Let w be the combined lengths of the initial and repeated segments that you found in the previous step. Your DFA classifier, following the standard construction, should have *exactly* $w \cdot m$ states. (This means do not give a "clever" solution with less states.) Each state of your DFA classifier should be represented by a pair (j, k), where j is the weight index of the state ($0 \leq j < w$) and k is the classifier value at the state ($0 \leq k < m$). Additionally, for a given state, the transition function at that state should be represented as a list of length b, where the α-th element of the list ($0 \leq \alpha < b$) should identify the new state that is reached upon transitioning on digit α.

   Your table should be formatted as a sequence of lines, with a one-to-one correspondence between lines and states. To properly output your table, begin by enumerating each state (j, k) in lexicographical order (so state $(1, 2)$ will come before state $(2, 0)$). Upon reaching state (j, k),

output that state, and then on the same line, output the transition function at that state. Each state you output should be formatted as a pair of base-10 numbers separated by commas, surrounded in parentheses. Each transition function represented as a list of states should be displayed in list form (see part 1). You do not need to specify the start state since $(0, 0)$ will always be the start state.

After outputting the table, output on a new line how many states the DFA classifier has.

(Note that this part does not require i or x.)

3. For the final part of the project, output the sequence of states that would be encountered, from first to last, upon inputting the sequence of base-b digits x into the DFA classifier above. This sequence of states should be displayed on a single line in list form, following the formatting rules used in the previous two parts. By convention, the initial state $(0, 0)$ should be included in your list as well.

   On a new line, you should also output "accept" if and only if $x \equiv i \pmod{m}$, otherwise you should output "reject". (Here, x denotes the number given by the sequence of base-b digits supplied as arguments to x. Keep in mind that the digits of x will be given in *right to left* order.)

3 Examples

The following examples demonstrate how your program should work and how output should be formatted. (The first line in each example is not part of the output, but rather the prompt showing the command-line arguments given to the program.)

3.1 Example 1

```
$>program 6 8 0   4 5 2
[1, 6, 4]
[0]
(0, 0) [(1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5)]
(0, 1) [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6)]
(0, 2) [(1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7)]
(0, 3) [(1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 0)]
(0, 4) [(1, 4), (1, 5), (1, 6), (1, 7), (1, 0), (1, 1)]
(0, 5) [(1, 5), (1, 6), (1, 7), (1, 0), (1, 1), (1, 2)]
(0, 6) [(1, 6), (1, 7), (1, 0), (1, 1), (1, 2), (1, 3)]
(0, 7) [(1, 7), (1, 0), (1, 1), (1, 2), (1, 3), (1, 4)]
(1, 0) [(2, 0), (2, 6), (2, 4), (2, 2), (2, 0), (2, 6)]
(1, 1) [(2, 1), (2, 7), (2, 5), (2, 3), (2, 1), (2, 7)]
(1, 2) [(2, 2), (2, 0), (2, 6), (2, 4), (2, 2), (2, 0)]
(1, 3) [(2, 3), (2, 1), (2, 7), (2, 5), (2, 3), (2, 1)]
(1, 4) [(2, 4), (2, 2), (2, 0), (2, 6), (2, 4), (2, 2)]
(1, 5) [(2, 5), (2, 3), (2, 1), (2, 7), (2, 5), (2, 3)]
(1, 6) [(2, 6), (2, 4), (2, 2), (2, 0), (2, 6), (2, 4)]
(1, 7) [(2, 7), (2, 5), (2, 3), (2, 1), (2, 7), (2, 5)]
(2, 0) [(3, 0), (3, 4), (3, 0), (3, 4), (3, 0), (3, 4)]
(2, 1) [(3, 1), (3, 5), (3, 1), (3, 5), (3, 1), (3, 5)]
(2, 2) [(3, 2), (3, 6), (3, 2), (3, 6), (3, 2), (3, 6)]
(2, 3) [(3, 3), (3, 7), (3, 3), (3, 7), (3, 3), (3, 7)]
(2, 4) [(3, 4), (3, 0), (3, 4), (3, 0), (3, 4), (3, 0)]
(2, 5) [(3, 5), (3, 1), (3, 5), (3, 1), (3, 5), (3, 1)]
(2, 6) [(3, 6), (3, 2), (3, 6), (3, 2), (3, 6), (3, 2)]
(2, 7) [(3, 7), (3, 3), (3, 7), (3, 3), (3, 7), (3, 3)]
(3, 0) [(3, 0), (3, 0), (3, 0), (3, 0), (3, 0), (3, 0)]
(3, 1) [(3, 1), (3, 1), (3, 1), (3, 1), (3, 1), (3, 1)]
(3, 2) [(3, 2), (3, 2), (3, 2), (3, 2), (3, 2), (3, 2)]
(3, 3) [(3, 3), (3, 3), (3, 3), (3, 3), (3, 3), (3, 3)]
(3, 4) [(3, 4), (3, 4), (3, 4), (3, 4), (3, 4), (3, 4)]
(3, 5) [(3, 5), (3, 5), (3, 5), (3, 5), (3, 5), (3, 5)]
(3, 6) [(3, 6), (3, 6), (3, 6), (3, 6), (3, 6), (3, 6)]
(3, 7) [(3, 7), (3, 7), (3, 7), (3, 7), (3, 7), (3, 7)]
32
```

```
[(0, 0), (1, 4), (2, 2), (3, 2)]
reject
```

3.2 Example 2

```
$>program 2 5 2   0 1 0 1 0 1
[]
[1, 2, 4, 3]
(0, 0) [(1, 0), (1, 1)]
(0, 1) [(1, 1), (1, 2)]
(0, 2) [(1, 2), (1, 3)]
(0, 3) [(1, 3), (1, 4)]
(0, 4) [(1, 4), (1, 0)]
(1, 0) [(2, 0), (2, 2)]
(1, 1) [(2, 1), (2, 3)]
(1, 2) [(2, 2), (2, 4)]
(1, 3) [(2, 3), (2, 0)]
(1, 4) [(2, 4), (2, 1)]
(2, 0) [(3, 0), (3, 4)]
(2, 1) [(3, 1), (3, 0)]
(2, 2) [(3, 2), (3, 1)]
(2, 3) [(3, 3), (3, 2)]
(2, 4) [(3, 4), (3, 3)]
(3, 0) [(0, 0), (0, 3)]
(3, 1) [(0, 1), (0, 4)]
(3, 2) [(0, 2), (0, 0)]
(3, 3) [(0, 3), (0, 1)]
(3, 4) [(0, 4), (0, 2)]
20
[(0, 0), (1, 0), (2, 2), (3, 2), (0, 0), (1, 0), (2, 2)]
accept
```

4 Writing and uploading your code

This project is (somewhat) language-agnostic, meaning that we do not require you to write your code in any one specific language. With that said, we intend to use the standard output produced by your code as the only

measure of determining whether your code works correctly. Since we want to be able to run your code directly from the command line, we want your code to be written in a *scripting language*. (We don't want to be burdened with trying to manually compile code from several different languages.) In order for us to determine how to run your code, you will place a shebang on the *very first* line of your code indicating what interpreter to use. Your shebang should take the form

`#!/usr/bin/env` *interpreter*

where *interpreter* is the name of the interpreter you plan to use. For this project, you are allowed to use any of the following interpreters: `python2`, `python3`, and `ruby`. If you there is an interpreter not previously listed that you would like to use instead, ask about it on Piazza. On the next line of code (below the shebang), put a comment containing your full name and your UID.

Your code will be run against numerous different inputs. Since the output provided by your program alone will be used to verify whether your code works correctly, it is VERY IMPORTANT that your output is properly formatted and that you outputted everything required. Use the examples provided above as a reference. However, to make things slightly easier on your end, space characters and trailing newline characters will be erased from your program output, so you do not need to worry about those.

Once you have completed the project, name your code file `proj01_`*lastname*. That is, name it "proj01" underscore {your last name in lowercase}. Submit your code to project `proj01` on the Submit Server. To help boost your grades, if your grade on this project is greater than your grade on hw01, then your hw01 grade will be changed to the grade for this project.