Kelly Sovacool

CS315-001
12 Oct. 2017

Homework 4

Design

I used the same object-oriented design from homework 2 in the Python 3 programming language to implement the algorithms. I created a class called BinaryNumber which inherits from Python's built-in list type. It stores each binary number as a list of integers, with the leftmost digit being the least significant digit. I overloaded the division, modulo, and exponentiation operators to solve Problems 1 and 2. I also modified the subtraction operator (which I overloaded for homework 2) to use the two's complement method, which seems to be faster than the school algorithm.

The division operator uses the long division method. If the dividend is less than the divisor, a quotient of zero and a remainder equal to the dividend is returned. Otherwise, a sub-dividend is found, which is the smallest section of the dividend from the significant end that is greater than the divisor. Then, digits of the dividend are iterated over from the most significant end. A sub-quotient and remainder is found, which is calculated by dividing the divisor from the sub-dividend using the subtraction method. The sub-quotient is inserted into the least significant end of the quotient. The next digit of the dividend is inserted into the least significant end of the remainder; this number becomes the next sub-dividend. This continues until all digits of the dividend are exhausted. The quotient and remainder are returned as a tuple.

The exponentiation operator uses the algorithm covered in class, but takes an optional modulo parameter N (this matches Python's power operator specifications). The result z is initialized as the binary number one, and the intermediate power w is initialized as x. Digits of the power are then iterated over from the least significant end. If the digit is equal to one, a new value for z is calculated as z times w mod N. The variable w is always updated as w times w mod N. If no modulo parameter is specified, the modulo operation is omitted from the calculations for z and w. The modulo operator simply calls the division algorithm and returns only the remainder. After all digits of the power have been iterated over, the result z is returned.

Problem 3 tests the runtime of modular exponentiation for increasing input sizes. The modular exponentiation operation is called for the first case of n=1. While the runtime of the previous run is less than the 600-second time limit, n is doubled and the next test case is generated and run with the modular exponentiation operation. The value of n will continue to double until a completed runtime exceeds 600 seconds. The algorithm is expected to run in cubic time ($O(n^3)$); therefore, a doubling of input size should elicit no more than an eight-fold increase in runtime ($2^3 = 8$). The test cases for Problem 3 met this expectation: each runtime was less than an eight-fold increase from the previous runtime (see the table for Problem 3 in the results section). The runtime exceeded the 600-second time limit at n=512.

I chose to hard-code the test cases given in the assignment instructions. Although I would never do this for production-quality code, this method is desirable for school assignments because it saves time for the tester and eliminates the possibility for confusion regarding input format. By default, binary numbers are printed as lists with the leftmost digit being the *least* significant. If the first and only optional positional argument is specified as "human-readable", then the binary numbers will be printed as strings with the leftmost digit being the *most* significant.

Results

Below are the results of the test cases for each problem. Note that in these tables, the binary numbers are represented with the leftmost digit being the *most* significant for improved readability.

Problem 1			
X	Y	Quotient	Remainder
111000110	101101111	1	1010111
11100011001	10110	1010010	1101
111000111000111000111	101010101010101	1010101	1110010001110

Problem 2			
X	Y	N	Result
10	1010	10100	100
1101	1000110000	1000110001	1
11001100110011001100110011001100110011	1100011	11110001110001110001110000111000111	11000111111100001101101110001001 0

Problem 3		
n size	Runtime (secs)	Current Runtime / Previous Runtime
1	0.002609	N/A
2	0.006022	2.31
4	0.018868	3.13
8	0.075909	4.02
16	0.309297	4.07
32	1.429554	4.62
64	9.809213	6.86
128	71.144159	7.25
256	497.006892	6.99
512	3639.998598	7.32

Instructions

This program is written in Python 3. Either ensure that the hash-bang (#!) on the first line contains the proper path to Python 3 on your machine and give the user permission to execute the file, or explicitly call the program using Python 3 on the command line. Usage:

```
python3 ./homework2.py
python3 ./homework2.py human-readable
```