Kelly Sovacool
CS315-001
16 Nov. 2017

Homework 6

Design

I used the same object-oriented design from homework 2 & 4 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 defined a module-level function to generate a BinaryNumber of a given bit size with the least and most significant digits as 1 and randomly pick 0 or 1 for all the digits in between.

The first task was to generate 100 pseudo-random odd binary numbers and compare the results of Fermat's test versus a brute-force approach for determining if the number is prime. I defined two methods in the BinaryNumber class to carry out these tests. The function is_prime_fermat returns the Boolean result of a single Fermat's test on the binary number N, as specified in the assignment instructions: $3^{N-1} \equiv 1 (\text{mod } N)$. The function is_prime_brute operates on the decimal representation of the binary number N. It checks all numbers from 2 up to the square root of N; if any number divides N (N mod number = 0), then N is not prime. If no number between 2 and the square root of N divides N, then N is prime. The purpose of this task is to empirically test how often Fermat's test falsely identifies a composite number as prime.

The second task was to generate binary numbers with 16, 32, and 64 bits. For each bit size, binary numbers were randomly generated until a prime was found and this number is reported below. Lagrange's prime number theorem states that for a bit size N, the probability of randomly picking a prime number is approximately 1.44/N. The purpose of this task is to empirically test the chance of finding a random prime number and compare this to Lagrange's theorem.

Results

Task 1: 100 random numbers				
i	**binary**	**decimal**	**Is prime**	**Fermat**
0	1111110101010000	64849	TRUE	TRUE
1	1010000001110110	41079	FALSE	FALSE
2	1011110101110110	48503	FALSE	FALSE
3	1101110010110010	56499	FALSE	FALSE
4	1001010110010000	38289	FALSE	FALSE
5	1110000000010110	57367	TRUE	TRUE
6	1000010100010010	34067	FALSE	FALSE
7	1110110111010010	60883	FALSE	FALSE
8	1010111111011000	45017	FALSE	FALSE
9	1110101110001000	60297	FALSE	FALSE
10	1110001100010100	58133	FALSE	FALSE

11	1110000000010110	57367	TRUE	TRUE
12	1000100011111000	35065	FALSE	FALSE
13	1100111001110100	52853	FALSE	FALSE
14	1000001000111100	33341	FALSE	FALSE
15	1001011001000100	38469	FALSE	FALSE
16	1011000000011000	45081	FALSE	FALSE
17	1100011010010000	50833	TRUE	TRUE
18	1001000100110010	37171	TRUE	TRUE
19	1100111001100000	52833	FALSE	FALSE
20	1101010011001100	54477	FALSE	FALSE
21	1100000010010010	49299	FALSE	FALSE
22	1100111110110100	53173	TRUE	TRUE
23	1000100000101100	34861	FALSE	FALSE
24	1000110110111010	36283	FALSE	FALSE
25	1000001100010000	33553	FALSE	FALSE
26	1011100111101000	47593	FALSE	FALSE
27	1001110010011110	40095	FALSE	FALSE
28	1000111111000110	36807	FALSE	FALSE
29	1010000000101110	41007	FALSE	FALSE
30	1000110001010000	35921	FALSE	FALSE
31	1110011011011110	59103	FALSE	FALSE
32	1100100101000110	51527	FALSE	FALSE
33	1011000111011010	45531	FALSE	FALSE
34	1011100111001010	47563	TRUE	TRUE
35	1011101101000110	47943	FALSE	FALSE
36	1110001100011010	58139	FALSE	FALSE
37	1000101001111000	35449	TRUE	TRUE
38	1000101111101110	35823	FALSE	FALSE
39	1110110011001010	60619	FALSE	FALSE
40	1010100111100010	43491	FALSE	FALSE
41	1101101010010100	55957	FALSE	FALSE
42	1110010001110110	58487	FALSE	FALSE
43	1111111110001100	65421	FALSE	FALSE
44	1100111010001100	52877	FALSE	FALSE
45	1100011001010010	50771	FALSE	FALSE
46	1010101010011110	43679	FALSE	FALSE
47	1001101100001100	39693	FALSE	FALSE
48	1110001100000110	58119	FALSE	FALSE

49	1111100001110110	63607	TRUE	TRUE
50	1000110111111010	36347	FALSE	FALSE
51	1111000100010010	61715	FALSE	FALSE
52	1001111010011100	40605	FALSE	FALSE
53	1101001100010100	54037	TRUE	TRUE
54	1011111101000100	48965	FALSE	FALSE
55	1001111011110010	40691	FALSE	FALSE
56	1110100001011000	59481	FALSE	FALSE
57	1010101011000100	43717	TRUE	TRUE
58	1001111110100010	40867	TRUE	TRUE
59	1010000001011100	41053	FALSE	FALSE
60	1111011111110100	63477	FALSE	FALSE
61	1011101000101010	47659	TRUE	TRUE
62	1001110011011010	40155	FALSE	FALSE
63	1000000001110010	32883	FALSE	FALSE
64	1110110010001010	60555	FALSE	FALSE
65	1101100011010100	55509	FALSE	FALSE
66	1010100001000010	43075	FALSE	FALSE
67	1100111110010010	53139	FALSE	FALSE
68	1100101000000000	51713	TRUE	TRUE
69	1101011010101010	54955	FALSE	FALSE
70	1110110100000110	60679	TRUE	TRUE
71	1110000100111110	57663	FALSE	FALSE
72	1101111010000000	56961	FALSE	FALSE
73	1101111001010010	56915	FALSE	FALSE
74	1110110010000010	60547	FALSE	FALSE
75	1010010101110110	42359	TRUE	TRUE
76	1011111110100100	49061	FALSE	FALSE
77	1110110110011010	60827	FALSE	FALSE
78	1011011100100110	46887	FALSE	FALSE
79	1110011001100000	58977	FALSE	FALSE
80	1100111010100000	52897	FALSE	FALSE
81	1011100101001000	47433	FALSE	FALSE
82	1100011100011110	50975	FALSE	FALSE
83	1110111001100100	61029	FALSE	FALSE
84	1100010111111010	50683	TRUE	TRUE
85	1110011110100100	59301	FALSE	FALSE
86	1000011100001010	34571	FALSE	FALSE

87	1100100100100000	51489	FALSE	FALSE
88	1010001010110110	41655	FALSE	FALSE
89	1101110111110100	56821	TRUE	TRUE
90	1011000100111110	45375	FALSE	FALSE
91	1100100000101110	51247	FALSE	FALSE
92	1000011000001010	34315	FALSE	FALSE
93	1100100100010000	51473	TRUE	TRUE
94	1000000110111010	33211	TRUE	TRUE
95	1110000100101000	57641	TRUE	TRUE
96	1110001111100110	58343	FALSE	FALSE
97	1110111100110110	61239	FALSE	FALSE
98	1111110010001000	64649	FALSE	FALSE
99	1110111101100110	61287	FALSE	FALSE
		Prime numbers:	21	
		False Fermat tests:	0	

Task 2: chance of random prime			
digits	**Randoms before prime**	**Empirical probability**	**Lagrange probability**
16	13	0.08	0.09
32	18	0.06	0.045
64	47	0.02	0.0225

Discussion

For task 1, out of 100 16-digit binary numbers generated, 21 were prime. Not a single composite number triggered a false-positive Fermat's test. I find this very surprising. When a number is composite, we expect Fermat's test to return the incorrect result about half the time. I suspect that either using 3 as the base for Fermat's test every time and/or that having every random number be odd may have skewed the results to make Fermat's test fail less often. Perhaps trying multiple bases for Fermat's test for each number may have elicited more false positives.

The probabilities calculated for each bit size were all rather close to the probabilities predicted by Lagrange's theorem. Each empirical probability was within 0.015 of its respective Lagrange probability. For bit sizes of 16 and 64, the prime numbers were slightly less abundant than the Lagrange theorem predicted. For the bit size of 32, the empirical probability was slightly greater than the Lagrange theorem predicted. I actually expected all of the empirical probabilities to be slightly higher than the Lagrange probability, since all pseudo-random numbers generated were odd. Executing these tests for many more trials and allowing even numbers would likely result in empirical probabilities even closer to the Lagrange theorem.

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
```