Quiz 5

111550057 資工15 莊婷馨

Problem 1

a) How to run the code:

Run “python3 RNG.py” in terminal. Running the code will create a “random.bin” file consists of 1M bytes of random numbers.

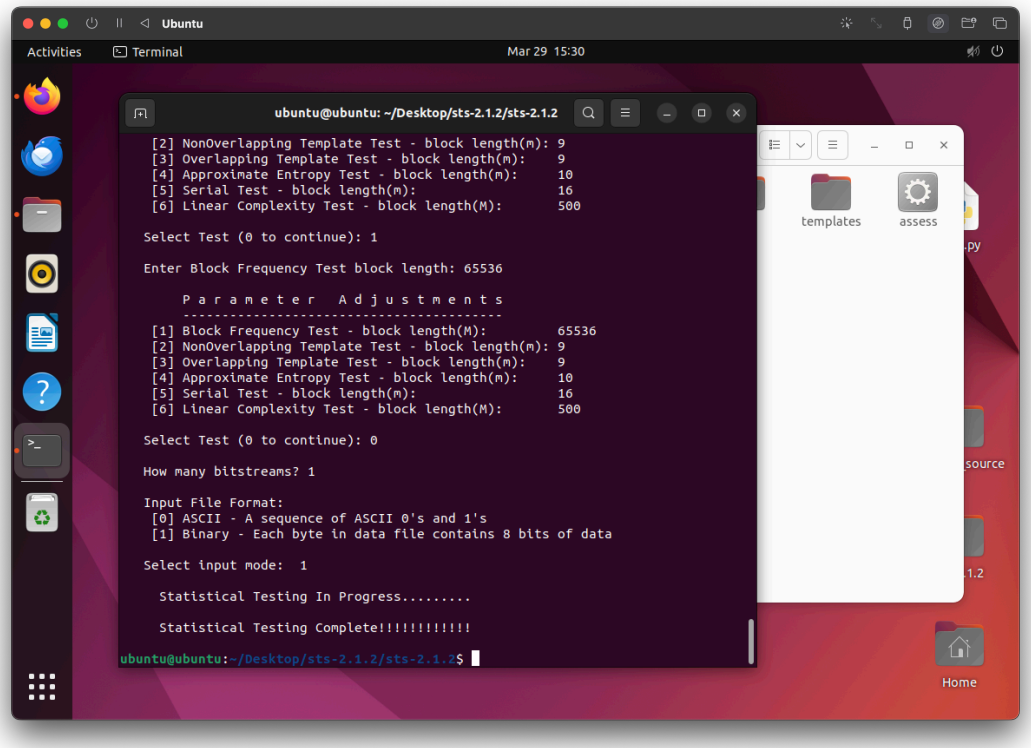
Implementation:

Use “SystemRandom” in the random library to generate $2^{20} \times 8$ random 0s and 1s.
Then split them into groups of 8.
Lastly, transform them into byte type and store the data in “random.bin" file.

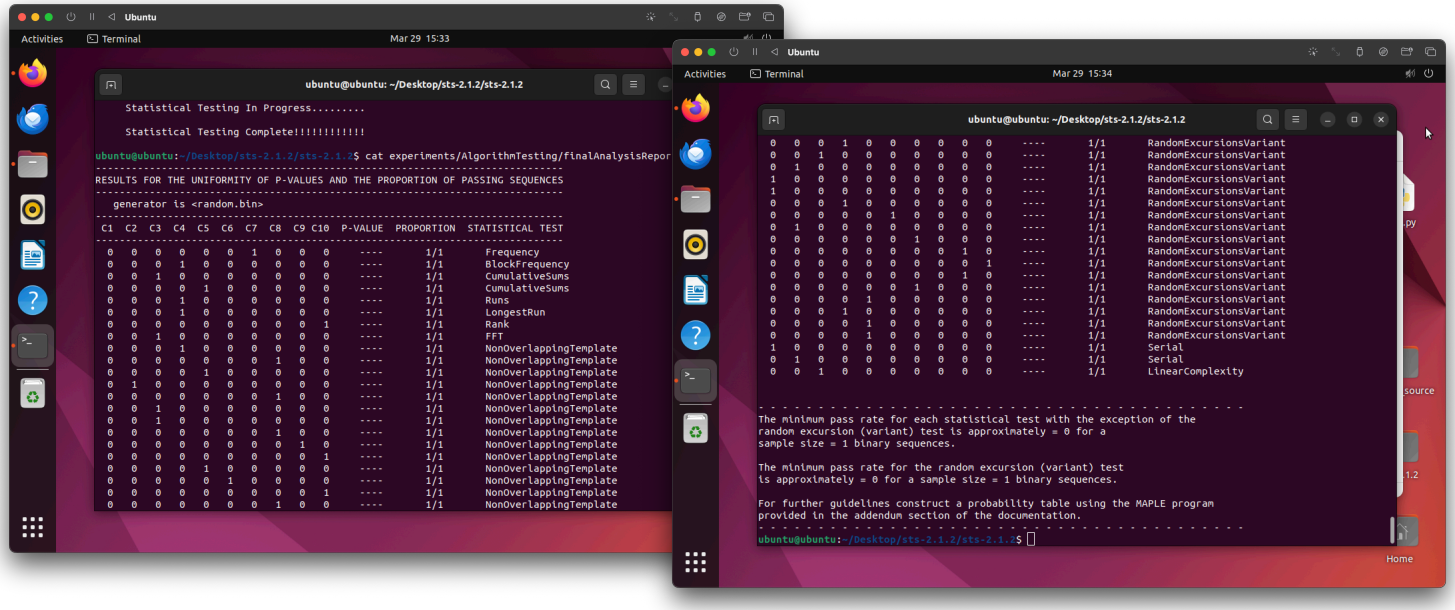
```python
from random import SystemRandom
rng = SystemRandom()

random_lst = [rng.randrange(2) for i in range(1048576*8)]
byte_lst = [random_lst[i:i+8] for i in range(0,len(random_lst),8)]

with open("random.bin","ab") as file:
    for byte in byte_lst:
        byte_data = bytes([int(''.join(map(str, byte)), 2)])
        file.write(byte_data)
```

b)

1. Run the NIST SP 800-22 statistical test. I ran the test on virtual machine of Ubuntu system.



The results:



2. Brief explanation of the process of the tests
   (1) Frequency
   Determine whether the number of 0s and 1s are approximately the same.
   (2) Block Frequency
   Determine whether the number of 0s and 1s in a M-bit block are approximately M/2.
   (3) Cumulative Sums
   Determine whether the cumulative sum of partial sequence is too large or too small relative to the behavior of cumulative sum for random sequences.
   (4) Runs
   A run of length k is a sequence of k identical bits bounded before and after with an opposite bit. Runs test determine whether the number of runs of 0s and 1s of various lengths is as expected for a random sequence.
   (5) Longest Run of Ones
   Determine whether the length of the longest run of 1s in a sequence is the same as the length expected in a random sequence.
   (6) Rank
   Assess the ranks of fixed length substrings of the original string to detect non-random patterns.
   (7) Discrete Fourier Transform
   Detect the periodic feature of the tested sequence that would indicate a deviation from the assumption of randomness.
   (8) Non-overlapping Template Matchings
   Detect the number of occurrences of pre-specified target strings to determine if the generator produce too many occurrences of a non-periodic pattern. If the pattern is not found, the searching window slides one bit position. If the pattern is found, the window is reset to the bit after the found pattern, and the search resumes.

(9) Overlapping Template Matchings

Same as the “Non-overlapping Template Matchings”, this test search for pre-specified target strings in the tested sequence. The difference between this test and the previous test is that when the pattern is found, the searching window slides only one bit before resuming the search.

(10)Universal Statistical

Determine whether the sequence can be significantly compressed without loss of information or not. A significantly compressible sequence is considered to be non-random. It focuses on the number of bits between matching patterns.

(11)Approximate Entropy

Compare the frequency of overlapping blocks of two consecutive/adjacent lengths (m and m+1) against the expected result for a random sequence.

(12)Random Excursions

Transfer the (0,1) sequence into (-1,+1) sequence, then calculate the partial sums. A cycle of a random walk consists of a sequence of steps of unit length taken at random that begin at and return to the origin. Determine if the number of visits to a particular state (-4, -3, -2, -1 and +1, +2, +3, +4) within a cycle deviates from what one would expect from a random sequence.

(13)Random Excursions Variant

Determine if the number of visits to a particular state (-9,-8,..., -1 and +1, ...,+8, +9) within a cycle deviates from what one would expect from a random sequence.

(14)Serial Test

Determine whether the number of occurrences of the 2^m m-bit overlapping patterns is approximately the same as would be expected for a random sequence. Random sequences have uniformity. If m=1, the Serial Test is equivalent to the Frequency Test.

(15)Linear Complexity

Determine the length of LFSR. If the sequence is characterized by a LFSR that is too short, it implies non-randomness.

c) How to run the code:

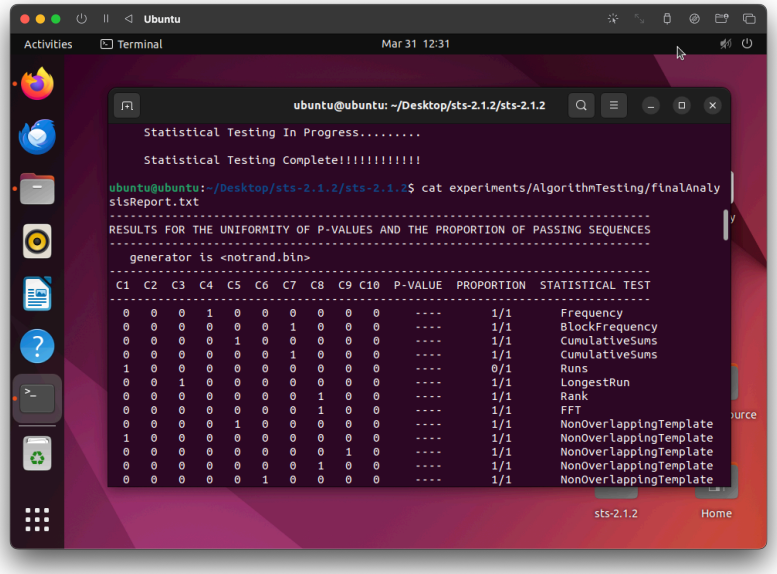
Run “python c.py” in the terminal. Running the code will create a “notrand.bin” file consists of 1M bytes of data.

Implementation:

I used “randint” in random library to generate random numbers.

```
import random

random_lst = [random.randint(0,1) for _ in range(1024*1024*8)]
byte_lst = [random_lst[i:i+8] for i in range(0,len(random_lst),8)]

with open("notrand.bin","ab") as file:
    for byte in byte_lst:
        byte_data = bytes([int(''.join(map(str, byte)), 2)])
        file.write(byte_data)
```

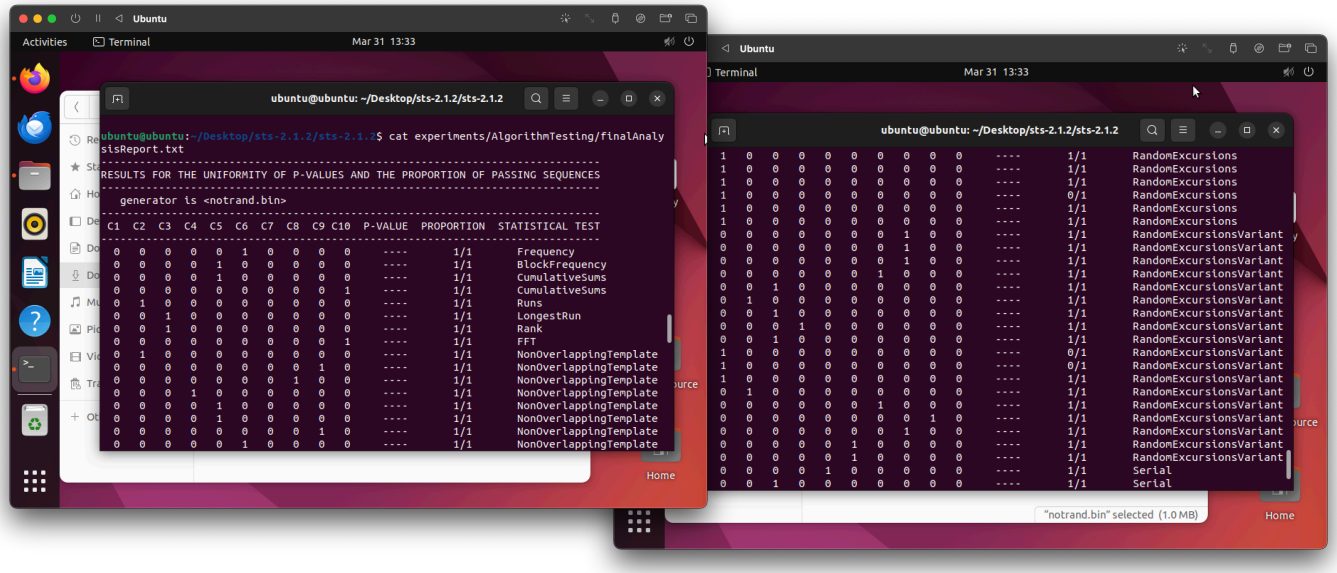
The result shows that the sequence doesn't pass the Runs Test.



To fix the result of Runs Test, I xor all the bits in the sequence with its next bit.

```
def xor(a,b):
    if a==b:
        return 0
    return 1

random_lst = [random.randint(0,1) for _ in range(1024*1024*8)]
for i in range(len(random_lst)-1):
    random_lst[i]=xor(random_lst[i],random_lst[i+1])
byte_lst = [random_lst[i:i+8] for i in range(0,len(random_lst),8)]
```

This approach did fix the result of Runs Test. However, it makes other test results 0.



After running the same code several times, I found out that the results seem to vary each time. Sometimes the generated sequence are more secure (like the file attached in the bonus folder).

The only approach I could think of is to generate a cryptographically sequence and xor it with the original sequence, but this approach is not reasonable and inefficient.