2.- Creación del diccionario de numeros primos(Combinaciones Binarias).

Felipe Sánchez Martínez
Escuela Superior de Cómputo, IPN

Septiembre de 2023

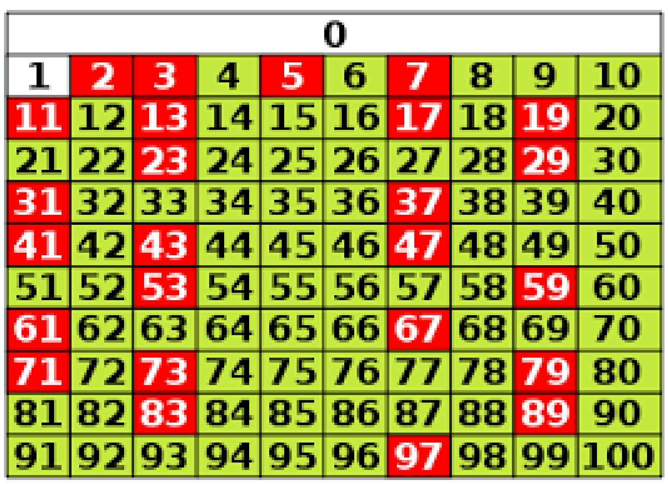
Índice general

Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

CAPÍTULO 1

Planteamiento del problema.

Un lenguaje es un conjunto de cadenas que pertenece al universo. Dentro del código binario, junto con todas sus combinaciones, y en el mundo de los números reales, existe un conjunto que ocasiona cierta dificultad al momento de encontrar algún patrón que siga y ser calculada su secuencia mediante una función. Ese problema se encuentra dentro del conjunto de los números primos.



0									
1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Figura 1.1: Números primos.

El cual es complicado de encontrar una secuencia que siga para calcular, debido a su naturaleza de que un número primo es primo si:

- Es divisible por 1 y por sí mismo -

Y para poder encontrar una función que nos diga la secuencia de los números primos, podemos buscar dentro del código binario. Si podemos representar los 1's de los números binarios en una gráfica, posiblemente encontraremos una solución a los números primos.

Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

CAPÍTULO 2

Marco Teórico.

2.1. Los Conceptos Centrales de la Teoría de Autómatas

2.1.1. Lenguajes

Un conjunto de cadenas, todas elegidas de algún Σ^*, donde Σ es un alfabeto particular, se llama un lenguaje. Si Σ es un alfabeto y $L \subseteq \Sigma^*$, entonces L es un lenguaje sobre Σ. Nótese que un lenguaje sobre Σ no necesita incluir cadenas con todos los símbolos de Σ, por lo que una vez que hemos establecido que L es un lenguaje sobre Σ, también sabemos que es un lenguaje sobre cualquier alfabeto que sea un superconjunto de Σ.

La elección del término "lenguaje"puede parecer extraña. Sin embargo, los lenguajes comunes pueden verse como conjuntos de cadenas. Un ejemplo es el inglés, donde la colección de palabras en inglés legales es un conjunto de cadenas sobre el alfabeto que consiste en todas las letras. Otro ejemplo es C, o cualquier otro lenguaje de programación, donde los programas legales son un subconjunto de las posibles cadenas que se pueden formar a partir del alfabeto del lenguaje. Este alfabeto es un subconjunto de los caracteres ASCII. El alfabeto exacto puede diferir ligeramente entre diferentes lenguajes de programación, pero generalmente incluye letras mayúsculas y minúsculas, dígitos, signos de puntuación y símbolos matemáticos.

Sin embargo, también existen muchos otros lenguajes que aparecen cuando estudiamos autómatas. Algunos son ejemplos abstractos, como:

- El lenguaje de todas las cadenas que consisten en n ceros seguidos de n unos, para algún $n \geq 0$: {ϵ, 01, 0011, 000111,...}.
- El conjunto de cadenas de 0's y 1's con igual número de cada uno: {ϵ, 01, 10, 0011, 0101, 1001,...}.
- El conjunto de números binarios cuyo valor es primo: {10, 11, 101, 111, 1011,...}.
- E es un lenguaje para cualquier alfabeto Σ.
- $\emptyset$, el lenguaje vacío, es un lenguaje sobre cualquier alfabeto.
- $\{\epsilon\}$, el lenguaje que consiste solo en la cadena vacía, también es un lenguaje sobre cualquier alfabeto. Observa que $\{\epsilon\}$ no tiene cadenas y $\emptyset$ no tiene cadenas.

La única restricción importante en lo que puede ser un lenguaje es que todos los alfabetos son finitos. Por lo tanto, los lenguajes, aunque pueden tener un número infinito de cadenas, están restringidos a consistir en cadenas extraídas de un alfabeto fijo y finito.

Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

CAPÍTULO 3

Desarollo del problema.

3.0.1. Approach.

Se decidió implementar el algoritmo utilizando dos ciclos 'for'. Por ejemplo, para un valor de $k = 3$, donde k representa la longitud de la cadena, la salida deseada debería tener el siguiente formato:

$$\{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111\}$$

En este caso, el primer ciclo abarcaría el número máximo que se puede formar con k bits (2^k), mientras que el segundo ciclo completaría el conjunto hasta ese punto.

```python
def isPrime(n):
    if(n > 1):
        for i in range(2, int(sqrt(n)) + 1):
            if (n % i == 0):
                return False

        return True
    else:
        return False
```

Una vez que se haya completado el código fuente, se desea graficar la cantidad de unos por cadena. Para esto, se utilizará la función de peso de Hamming para reducir el tiempo de ejecución en el cálculo del número de unos por cadena. Los resultados se almacenarán en un archivo '.csv' para su posterior manipulación con la biblioteca Pandas.

```python
def hammingWeight(n):
    """
    :type n: int
    :rtype: int
    """
```

```
    mask = 1
    count = 0
    while n != 0:
        if (mask & n == 1):
            count = count + 1
        n = n>>1

    return count
```

3.0.2. Complejidad

Complejidad Temporal

La complejidad del algoritmo se divide en dos componentes principales: la generación de las cadenas y la función de Hamming.

Para la generación de las cadenas, la complejidad temporal es:

$$T : O(2^k + 2^{k-1} + 2^{k-2} + \ldots + 2^1 + 2^0) = O(2^k)$$

Por otro lado, al considerar la función de Hamming, la complejidad temporal es:

$$O : O(k \cdot 2^k)$$

En conjunto, la complejidad temporal total del algoritmo es:

$$O(2^k) + O(k \cdot 2^k) = O(2^k)$$

Complejidad Espacial

Dentro del algoritmo, la generación del output no se considera en la complejidad espacial, lo que resulta en una complejidad espacial de:

$$O : O(1)$$

Sin embargo, al momento de graficar, se utiliza memoria RAM, lo que eleva la complejidad espacial a:

$$O : O(2^k)$$

El uso de un archivo .csv en combinación con las bibliotecas Pandas y Matplotlib permite una gestión más eficiente de la memoria en RAM al evitar cargar grandes conjuntos de datos completamente en memoria. Sin embargo, es importante tener en cuenta que la lectura y escritura de archivos .csv desde y hacia el disco pueden tener un impacto en el rendimiento en términos de velocidad de acceso al disco."

3.0.3. Codigo fuente.

```
import random
import time
from math import sqrt
```


Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

```python
# graph libs
import pandas as pd
import matplotlib.pyplot as plt

import argparse

import math

def main(args):
    while True:
        n = input("Pls give me the max number to check prime numbers or just press 'enter': ")
        if n == '':
            n = random.randint(0, 10000000)
        else:
            n = int(n)

        print(f"Numbers checked:: {n}")



        # OUTPUT FILES
        primes = open('primes.txt', 'w')
        df = open('primes_data.txt', 'w')
        df_log = open('primes_log10.txt','w')

        # BASIC INFO
        primes.write("{")
        df.write('chain,number_of_1s\n')
        df_log.write('chain,number_of_1s\n')
        # LOGIC
        i = 0
        for number in range(n + 1):
            if (isPrime(number)):
                primes.write(bin(number)[2::] + ', ')

                df.write(str(i) + ',' + str(hammingWeight(number)) + '\n')
                df_log.write(str(i) + ',' + str(round(math.log10(hammingWeight(number)), 4)) + '\n')
                i += 1

        primes.write('}')


        # CLOSE FILES
        primes.close()
        df.close()
```

```python
        df_log.close()


        # PLOT
        df = pd.read_csv('primes_data.txt')
        df_log = pd.read_csv('primes_log10.txt')
        # Plotting
        # only dots and integers
        plt.plot(df['chain'], df['number_of_1s'], 'bo')
        plt.title('Primes Binary Strings')
        plt.xlabel('Chain')
        plt.ylabel('Number of 1s')
        plt.show()

        plt.plot(df_log['chain'], df_log['number_of_1s'], 'bo')
        plt.title('Primes Binary Strings in log10')
        plt.xlabel('Chain')
        plt.ylabel('Number of 1s')
        plt.show()

        if input("Do you want to continue? (y/n): ") == 'n':
            break
        print("-"*60)


def isPrime(n):
    if(n > 1):
        for i in range(2, int(sqrt(n)) + 1):
            if (n % i == 0):
                return False

        return True
    else:
        return False


def hammingWeight(n):
    """
    :type n: int
    :rtype: int
    """
    mask = 1
    count = 0
    while n != 0:
        if (mask & n == 1):
            count = count + 1
        n = n>>1
```

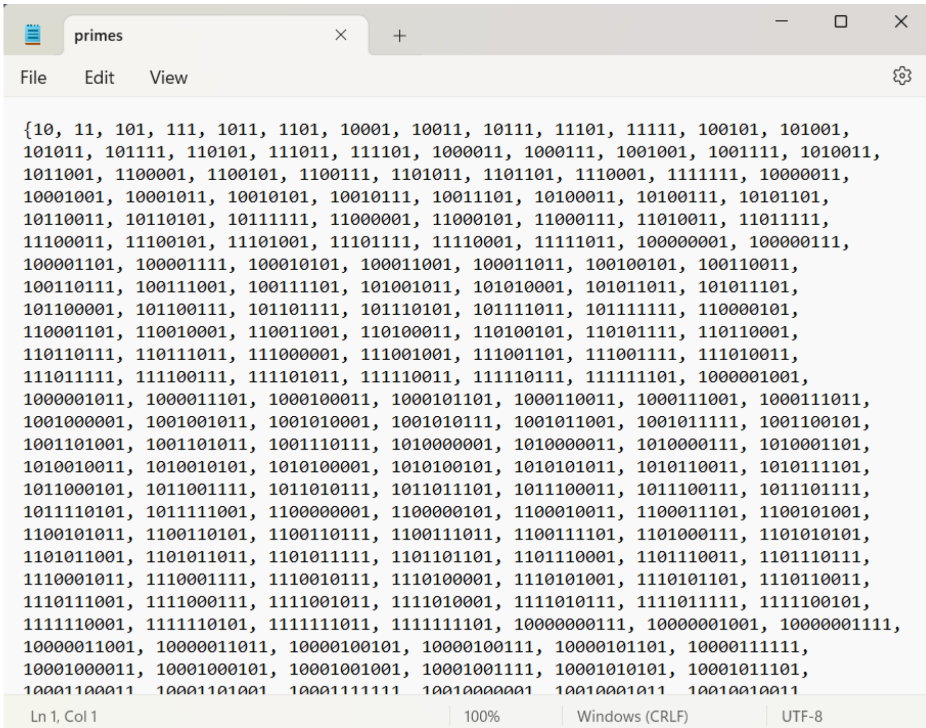
```
    return count


def parse_args():
    # setup arg parser
    parser = argparse.ArgumentParser()

    # random default value
    default_n = random.randint(0, 10000000)


    # add arguments
    parser.add_argument("n",
                        type=int, help="number of primes numbers to check",
                        default=default_n, nargs='?')
    # parse args
    args = parser.parse_args()

    # return args
    return args

# run script
if __name__ == "__main__":
    # add space in logs
    print("\n\n")
    print("*" * 60)
    start = time.time()

    # parse args
    args = parse_args()

    # run main function
    main(args)

    end = time.time()
    print("Total time taken: {}s (Wall time)".format(end - start))
    # add space in logs
    print("*" * 60)
    print("\n\n")
```

3.0.4. Casos de prueba.

Sin nigun Input.

- **Output en archivo .txt:**

$$\{10, 11, 101, 111, ..., 1011110111010010001101, 1011110111010010001111, 1011110111010010010101001,\}$$



primes

File Edit View

```
{10, 11, 101, 111, 1011, 1101, 10001, 10011, 10111, 11101, 11111, 100101, 101001,
101011, 101111, 110101, 111011, 111101, 1000011, 1000111, 1001001, 1001111, 1010011,
1011001, 1100001, 1100101, 1100111, 1101011, 1101101, 1110001, 1111111, 10000011,
10001001, 10001011, 10010101, 10010111, 10011101, 10100011, 10100111, 10101101,
10110011, 10110101, 10111111, 11000001, 11000101, 11000111, 11010011, 11011111,
11100011, 11100101, 11101001, 11101111, 11110001, 11111011, 100000001, 100000111,
100001101, 100001111, 100010101, 100011001, 100011011, 100100101, 100110011,
100110111, 100111001, 100111101, 101001011, 101010001, 101011011, 101011101,
101100001, 101100111, 101101111, 101110101, 101111011, 101111111, 110000101,
110001101, 110010001, 110011001, 110100011, 110100101, 110101111, 110110001,
110110111, 110111011, 111000001, 111001001, 111001101, 111001111, 111010011,
111011111, 111100111, 111101011, 111110011, 111110111, 111111101, 1000001001,
1000001011, 1000011101, 1000100011, 1000101101, 1000110011, 1000111001, 1000111011,
1001000001, 1001001011, 1001010001, 1001010111, 1001011001, 1001011111, 1001100101,
1001101001, 1001101011, 1001110111, 1010000001, 1010000011, 1010000111, 1010001101,
1010010011, 1010010101, 1010100001, 1010100101, 1010101011, 1010110011, 1010111101,
1011000101, 1011001111, 1011010111, 1011011101, 1011100011, 1011100111, 1011101111,
1011110101, 1011111001, 1100000001, 1100000101, 1100010011, 1100011101, 1100101001,
1100101011, 1100110101, 1100110111, 1100111011, 1100111101, 1101000111, 1101010101,
1101011001, 1101011011, 1101011111, 1101101101, 1101110001, 1101110011, 1101110111,
1110001011, 1110001111, 1110010111, 1110100001, 1110101001, 1110101101, 1110110011,
1110111001, 1111000111, 1111001011, 1111010001, 1111010111, 1111011111, 1111100101,
1111110001, 1111110101, 1111111011, 1111111101, 10000000111, 10000001001, 10000001111,
10000011001, 10000011011, 10000100101, 10000100111, 10000101101, 10000111111,
10001000011, 10001000101, 10001001001, 10001001111, 10001010101, 10001011101,
10001100011, 10001101001, 10001111111, 10010000001, 10010001011, 10010010011
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

Figura 3.1: Archivo de Texto

- **Output en archivo .txt:**

```
chain,number_of_1s
0,1
1,2
2,2
3,3
4,3
5,3
...
426893,15
426894,13
426895,14
426896,13
426897,14
```


Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

```
426898,15
426899,13
```

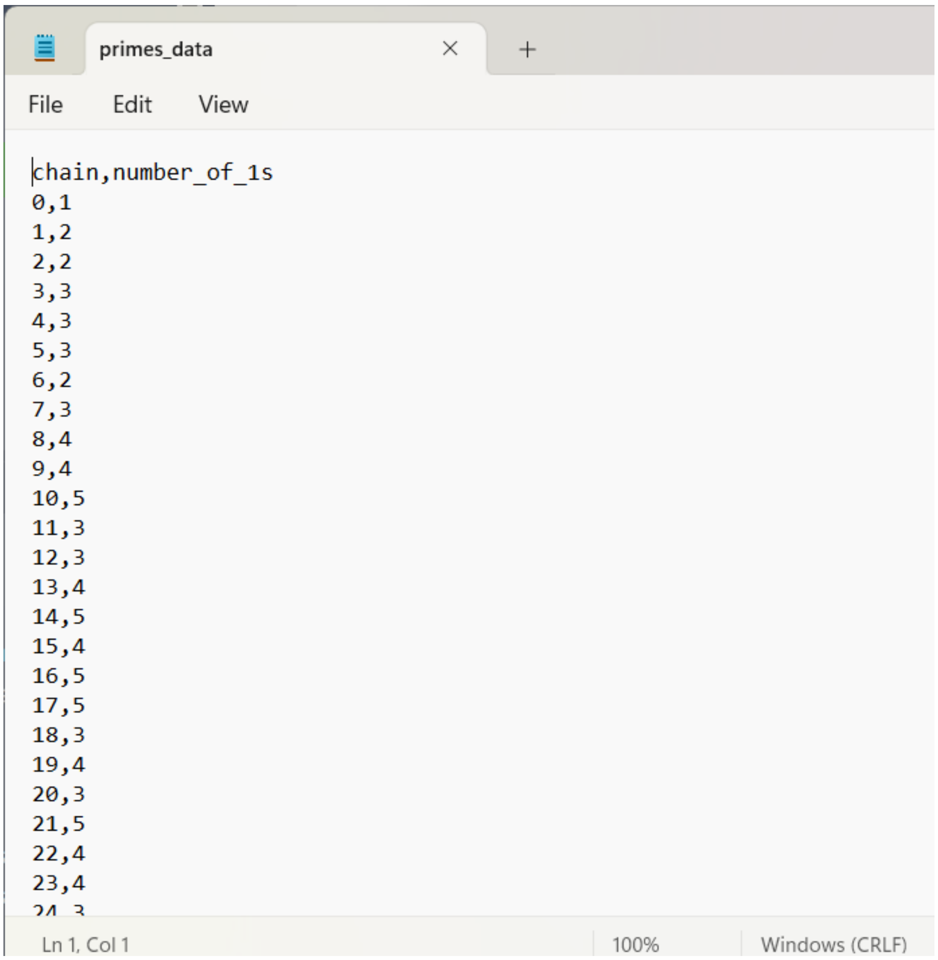


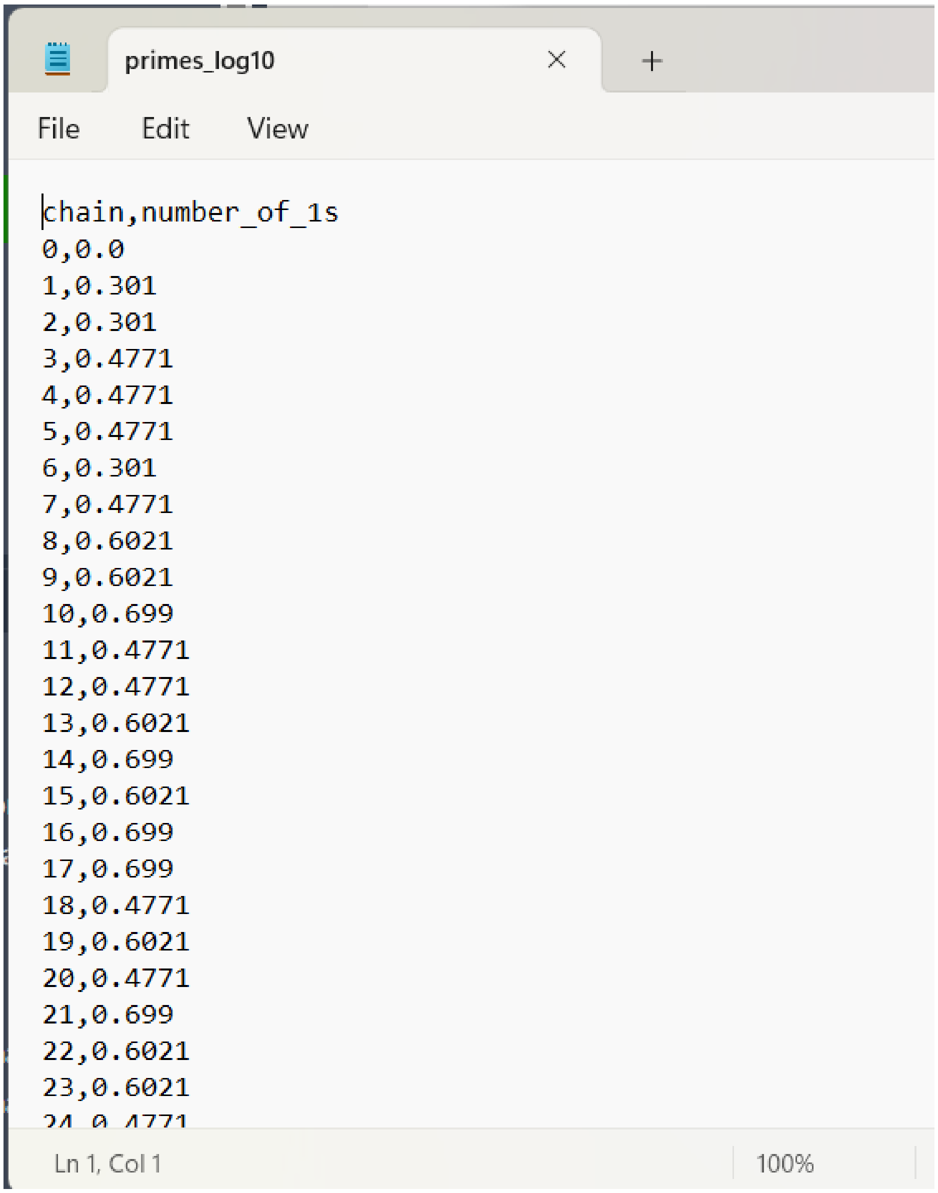
Figura 3.2: Archivo de Texto

- **Output en archivo .txt de numeros en logaritmo base 10:**

```
 chain,number_of_1s
0,0.0
1,0.301
2,0.301
3,0.4771
4,0.4771
5,0.4771
...
```

```
426895,1.1461
426896,1.1139
426897,1.1461
426898,1.1761
426899,1.1139
```


Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com


primes_log10

File Edit View

chain,number_of_1s
0,0.0
1,0.301
2,0.301
3,0.4771
4,0.4771
5,0.4771
6,0.301
7,0.4771
8,0.6021
9,0.6021
10,0.699
11,0.4771
12,0.4771
13,0.6021
14,0.699
15,0.6021
16,0.699
17,0.699
18,0.4771
19,0.6021
20,0.4771
21,0.699
22,0.6021
23,0.6021
24,0.4771

Ln 1, Col 1 100%

Figura 3.3: Archivo de Texto

- **Mensaje en la terminal:**

```
C:\Users\CristoRey\Escom\SEM_5>py prime_numbers.py
*************************************************************
Total time taken: 84.34974026679993s (Wall time)
Number of primes numbers checked: 6220079
*************************************************************
```

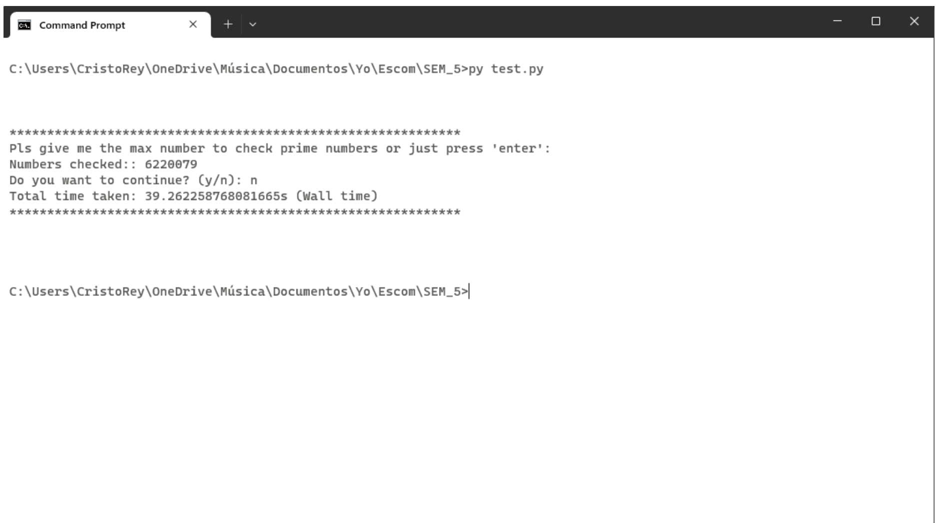


Figura 3.4: Terminal

Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

- **Gráfica:**

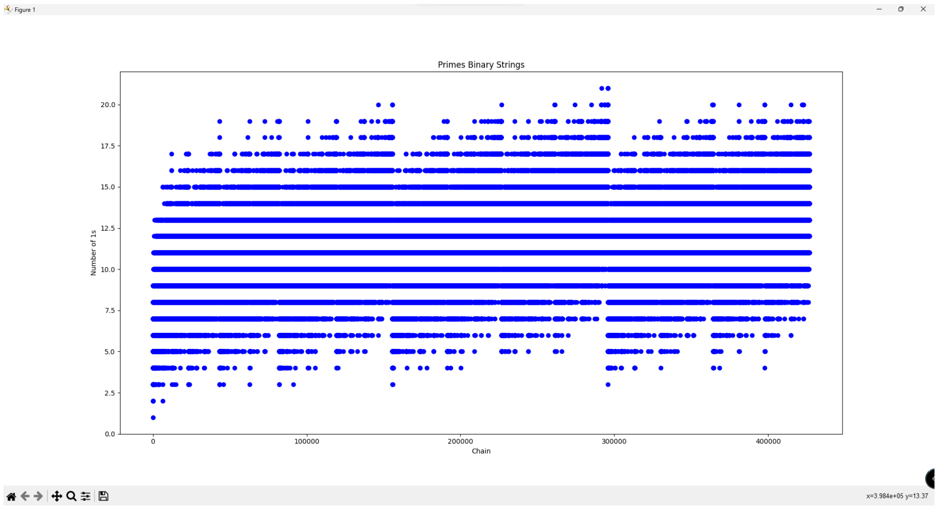


Figura 3.5: Gráfica de puntos con n = 6220079 (random).

- **Gráfica de primos con logaritmo base 10:**

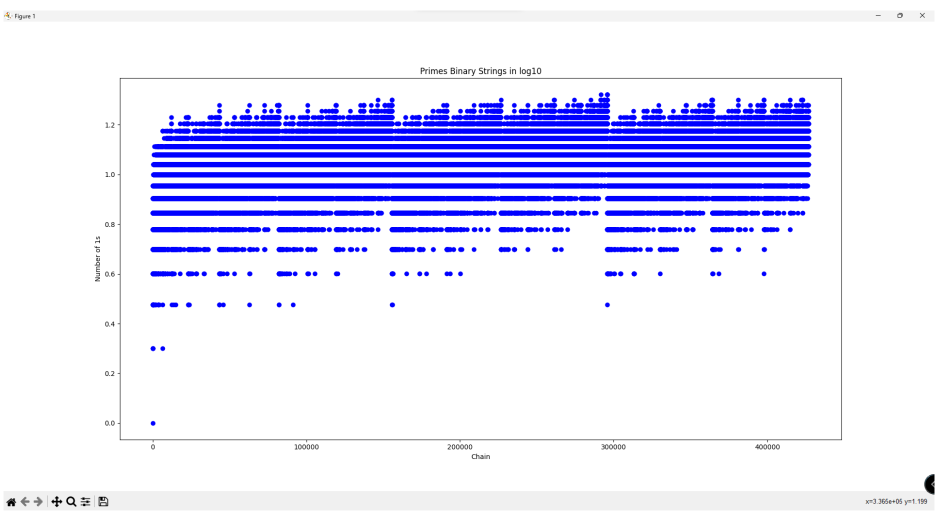


Figura 3.6: Gráfica de puntos con n = 6220079 (random) y logaritmo base 10.

Tamaño de la cadena igual a 8000000

Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

- **Output en archivo .txt:**

$$\{10, 11, 101, 111, ..., 1110100001000111011011, 11110100001000111111001, \}$$

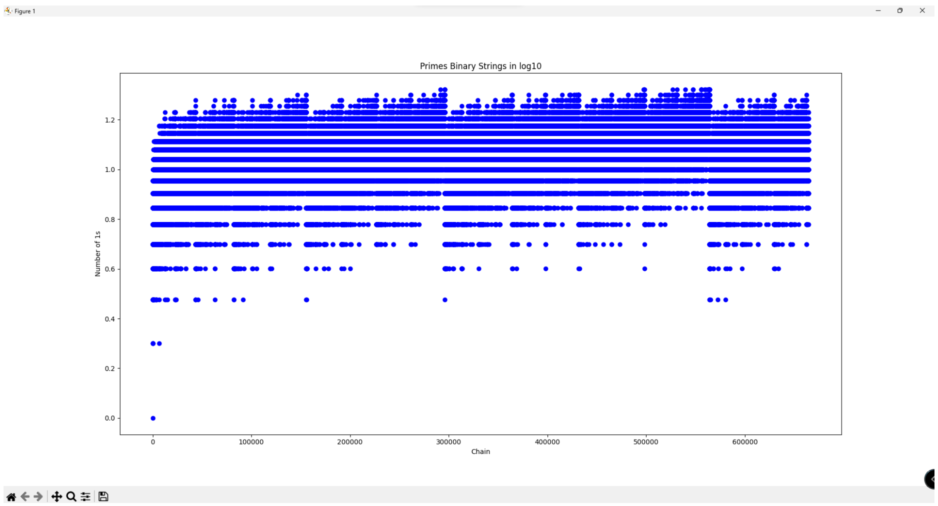


Figura 3.7: Terminal.

- **Output en archivo .txt:**

```
chain,number_of_1s
0,1
1,2
2,2
3,3
4,3
5,3
6,2
7,3
8,4
539767,10
539768,10
539769,10
539770,12
539771,11
539772,11
539773,13
539774,11
539775,13
539776,13
```

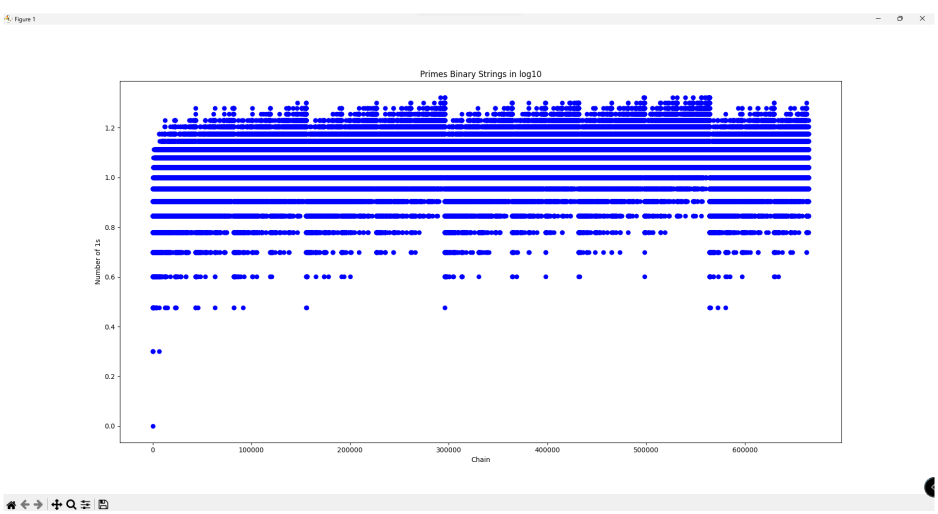


Figura 3.8: Terminal.

- **Output en archivo .csv de numeros en logaritmo base 10:**

```
chain,number_of_1s
0,0.0
1,0.301
```


Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

```
2,0.301
3,0.4771
4,0.4771
5,0.4771
6,0.301
7,0.4771
8,0.6021
9,0.6021
10,0.699

539772,1.0414
539773,1.1139
539774,1.0414
539775,1.1139
539776,1.1139
```

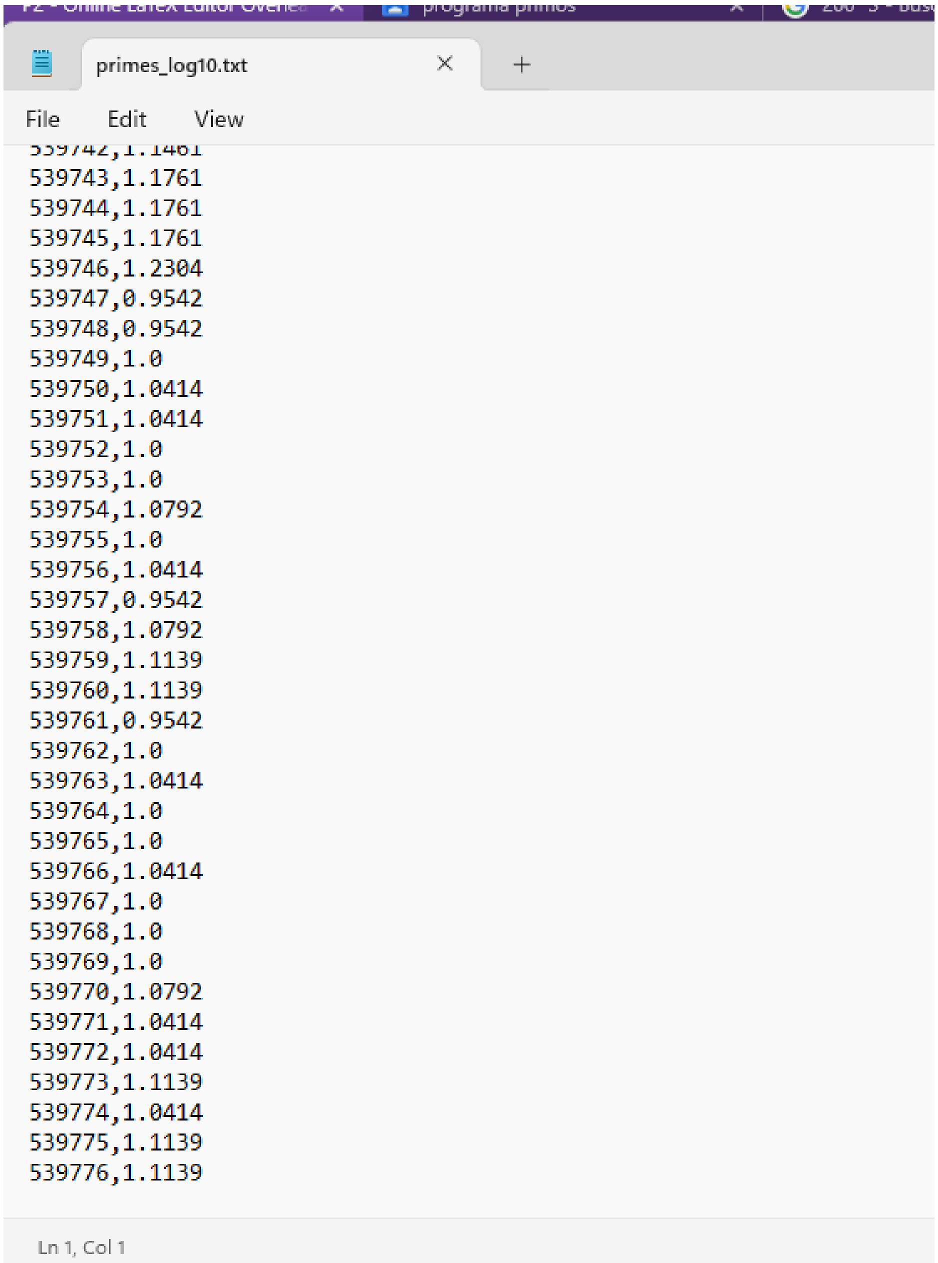


Figura 3.9: Terminal.

Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

- **Mensaje en la terminal:**

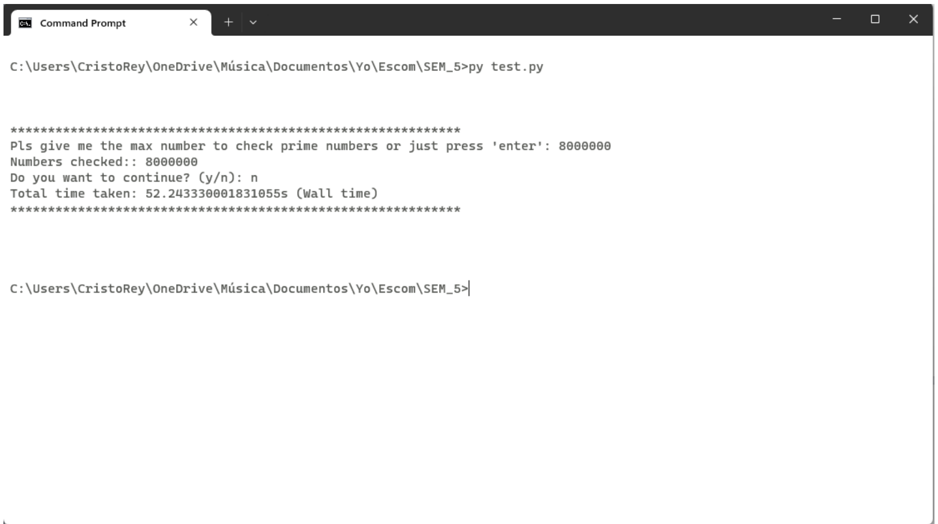


Figura 3.10: Terminal.

- **Gráfica:**

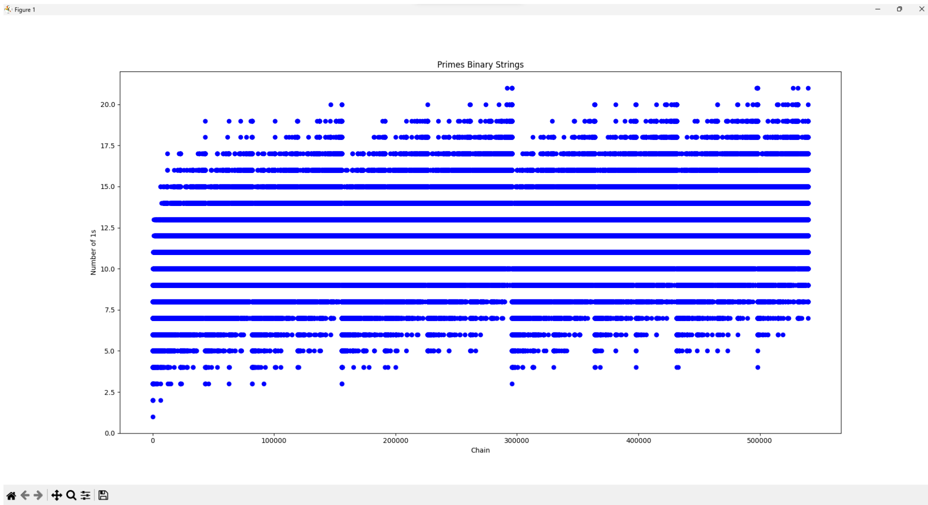


Figura 3.11: Gráfica de puntos con n = 8000000.

Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

- **Gráfica de primos con logaritmo base 10:**

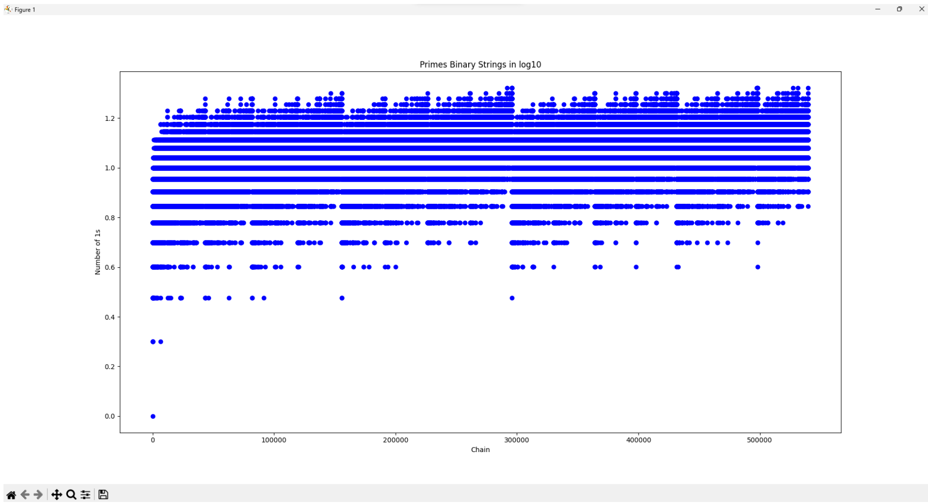


Figura 3.12: Gráfica de puntos con n = 8000000 y logaritmo base 10.

Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

CAPÍTULO 4

Conclusión.

La busqueda de una funcion que represente la secuencia de los numeros primos, es uno de los grandes problemas del siglo, y leyendo la grafica representada por el numero de 1's en las cadenas de 10 millones de numeros primos, no nos da mucha ayuda para poder lograr ver la funcion representada en el grafico, aun representandola en su funcion logaritmica, por lo que su solucion posdxria llegar a estar en su representacion en otros diccionarios, los cuales pueden ser cambiados con las funciones acrtuiales.

Si se lograra conocer una funcion que represente la serie de las suceciones de los numeros primos, esto tendria gran impacto para la solucion de varios algoritmos cripto

CAPÍTULO 5

Referencias

- Hopcroft, J. E., Motwani, R., y Ullman, J. D. (2006). *Introduction to Automata Theory, Languages, and Computation* (3ra ed.). Pearson.
- Eduardo. (2019, junio 21). La Hipótesis de Riemann y los números primos. *La Hipótesis de Riemann Parte 3* [Archivo de video]. Derivando. https://www.youtube.com/watch?v=T4FgqV0F_bY