Programa 1: Universo

Dueñas Jiménez Cristian Alexis

November 6, 2022

1 Introducción

Como bien hemos visto en clase, un autómata es un modelo computacional que consiste en un conjunto de estados bien definidos, un estado inicial, un alfabeto de entrada y una función de transición. El problema consta de encontrar el Universo(cantidad de combinaciones posibles de cadenas dado un alfabeto) dada una N. Como se nos plantea, el alfabeto es binario, con el uno y el cero como únicos caracteres. La N representa la longitud en bits máxima que se pueden encontrar en el Universo. Para encontrar esto, se hace uso de la suma binaria. Se suma de uno en uno, para encontrar todas las combinaciones posibles. Sin embargo, hay algunas combinaciones que no se reflejan únicamente sumando de uno en uno.

2 Marco Teórico

Un autómata es un modelo matemático para una máquina de estado finito, en el que dada una entrada de símbolos, salta mediante una serie de estados de acuerdo a una función de transición (que puede ser expresada como una tabla). Esta función de transición indica a qué estado cambiar dados el estado actual y el símbolo leído. Su universo está dado por todas las posibilidades que puede encontrar un autómata dado un lenguaje.

3 Desarrollo

Para encontrara todas las combinaciones posibles, implementamos una función llamada *sigma* que tiene como parámetro la N que ingresa el usuario.

Primero, se hace uso de listas para guardar datos que nos serán de utilidad para posteriormente graficar nuestros datos.

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 import os
4 import sys
5
6 def graficar(largo, unos, ejeX, ejeY):
```

```
7
8       plt.plot(largo, unos, marker="")
9       plt.xlabel(ejeX)
10      plt.ylabel(ejeY)
11      plt.title("GRAFICA")
12      plt.show()
13
14  def sigma(n):
15
16      unos = [] #Arreglo para almacenar la cantidad de unos en cada
        cadena binaria.
17      largo =[] #Arreglo para determinar cuantas cadenas se generan
18      cont = -1 #Contador el cual incrementa con cada iteraci n de
        los fors.
19      concat = ""
20      countElements = 0
21      newString = ""
22      unos64 = []
23      cadenas64 = 0
24      largo64 = []
25      f1 = open("../PARCIAL 1/PROGRAMAS/Programa1/outputs/universo.
        txt", "w", encoding="utf-8")
26      f2 = open("../PARCIAL 1/PROGRAMAS/Programa1/outputs/
        Concatenadas.txt", "w", encoding="utf-8")
27      f3 = open("../PARCIAL 1/PROGRAMAS/Programa1/outputs/Segmentadas
        .txt", "w", encoding="utf-8")
28      f1.write("{")
29
30      for a in range(n+1): ##Generamos nuestro bucle, este s lo es
        para imprimir la cantidad de iteraciones que se llevan a cabo.
        Ponemos n+1, ya que
31          print("n = ",a) #Imprimimos las iteraciones realizadas,
        conforme se va realizando cada una de ellas.
32          for x in range(2 ** a): #En este segundo for, determinamos
        la cantidad de cadenas que habr  seg n el valor de a, es
        decir, seg n el n mero de iteraci n. Esto se determina con
        2^a.
33
34              cont += 1 #Incrementamos el contador que inicializamos
        anteriormente, esto para llevar el conteo de cadenas.
35              binario = bin(x)[2:]
36
37              if len(binario) < a: #Condici n para agregar ceros.
38                  binario = ('0' * (a-len(binario))) + binario
39
40              if a == 0:
41                  binario = '  ' #Cuando nuestro valor ingresado es
        "0", el universo se determina con su respectivo s mbolo.
42
43              if a > 0:
44                  concat = concat+binario
45
46              f1.write(binario + ',') #Mandamos a escribir en nuestro
         archivo cada una de las cadenas generadas.
47              unos.append(binario.count('1')) #Realizamos el conteo
        de 1's en cada cadena y lo guardamos en el arreglo de unos.
48              largo.append(cont) #Almacenamos la cantidad de cadenas
```

```
    que van en cada iteraci n
49
50      f2.write(concat)
51      f2.close()
52      f1.write("}") #Una vez finalizados los ciclos, cerramos nuestro
         conjunto con una llave.
53      f1.close() #Cerramos el archivo
54      f3.write("Segmnentadas" + os.linesep)
55
56      for char in concat:
57          countElements+=1
58          newString = newString+char
59
60          if countElements == 64:
61              cadenas64 += 1
62              unos64.append(newString.count('1'))
63              largo64.append(cadenas64)
64              countElements = 0
65              f3.write(newString + os.linesep)
66              newString = ""
67
68      graficar(largo,unos, "Cantidad de Cadenas", "Cantidad de 1's")
        #Mandamos llamar la funci n graficar, la cual nos muestra la
        cantidad de 1's en cada cadena, por lo cual le mandamos la
        variable largo y unos.
69      graficar(largo,np.log10(unos), "Cantidad de Cadenas", "Cantidad
         de 1's (Log10)")
70      graficar(largo64,unos64, "Cantidad de Cadenas", "Cantidad de 1'
        s")
71      graficar(largo64,np.log10(unos64), "Cantidad de Cadenas", "
        Cantidad de 1's (Log10)")
72
73  #Generamos nuestro men  de opciones
74  opc = "1"
75  print("MENU DE OPCIONES")
76  while opc == "1":
77
78      opc = input("Seleccione una opci n: \n1.- Ingresar n \n2.-
        Finalizar\n")
79
80      if opc == "1":
81          n = int(input("Ingrese el valor de n: "))
82          sigma(n)
83      else:
84          if opc == "2":
85              print(f"FIN DE LA EJECUCION")
86              sys.exit(0)
87          else:
88              if opc != "1":
89                  print("Seleccione una opcion valida")
```

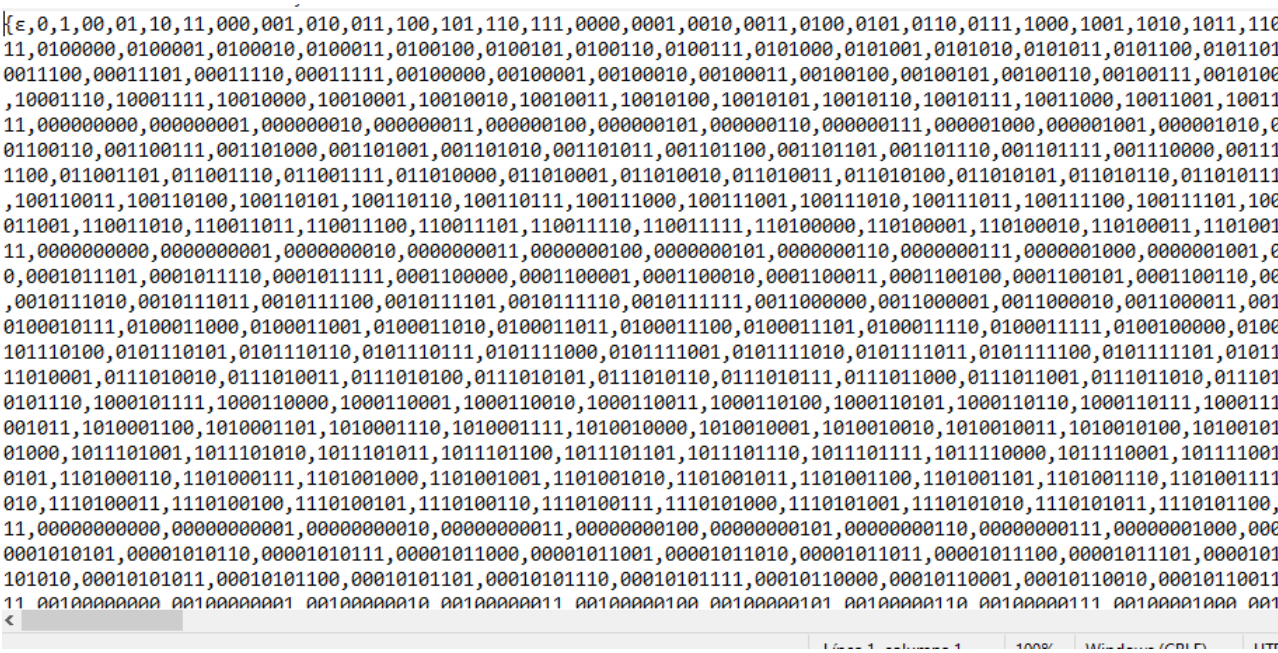
Algoritmo 1: Implementación del Universo

El primer ciclo, nos ayuda a saber en que n nos encontramos. En otras palabras, nos dice la cantidad de bits que vamos a tener en la cadena generada. El segundo for hace uso de la suma binaria. Aumenta de uno en uno los elementos.

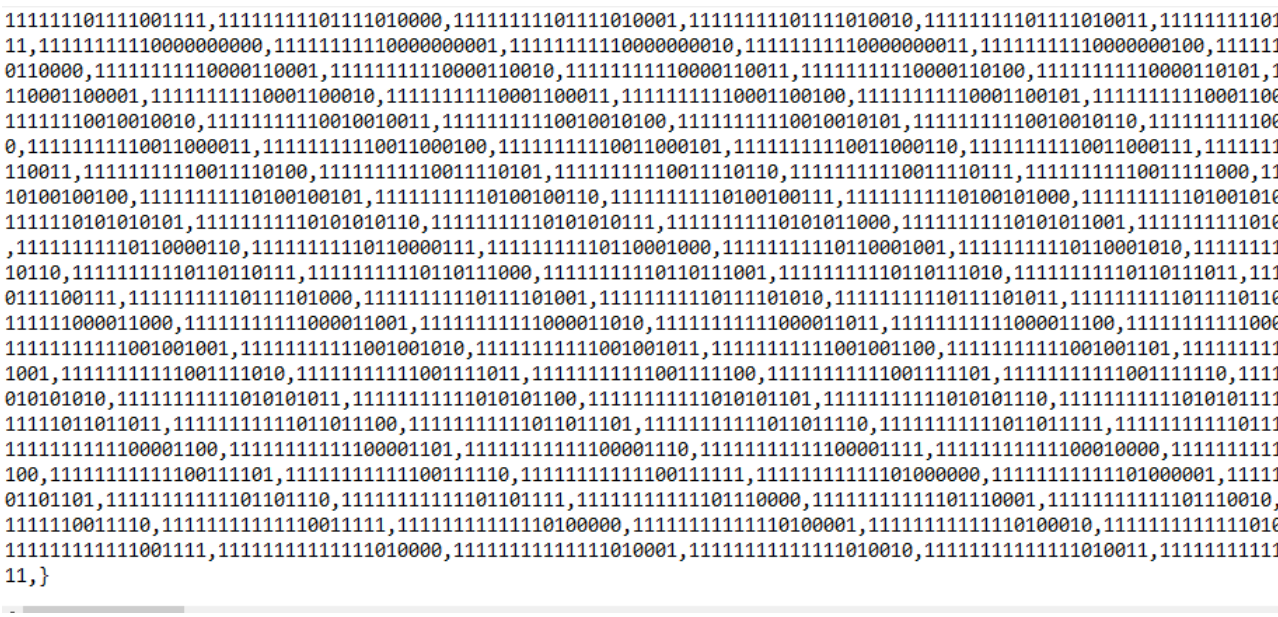
La sentencia $iflen(binario) < a$: nos permite agregar ceros a la izquierda para tener una cadena que sea existente dentro de la cantidad de bits. Para entenderlo mejor, supongamos que el usuario ingresa $n = 3$. El primer ciclo for irá de 0 a 3. En la primera iteración, la longitud en bits máxima que puede tener la cadena es 0. En la segunda iteración, la longitud ahora es de uno. Entra en el segundo for, donde despliega todos las combinaciones posibles cuando $n = 1$. Al terminar el ciclo, $n = 2$. Entra de nuevo en el segundo for donde mostrará de nuevo las combinaciones posibles. Sin embargo, al convertir los decimales en binario, no toma en cuenta la condición de la longitud. Para que esto se cumpla, resta la cantidad esperada de bits, menos la longitud de la cadena la arrojada al convertir de decimal a binario. El resultado de eso es la cantidad de ceros a la izquierda que se la agrega a la cadena. De esta forma, se garantiza que el programa no tendrá cadenas repetidas. Una vez obtenida la cadena, la escribe en el archivo. Del mismo modo, guarda en arreglos la cantidad de unos en la cadena, el total de caracteres en la cadena, y la cantidad de cadenas encontradas al momento. Además, cada que se genera una cadena, se va concatenando para que al final forme una sola, establecemos una condición para que no se agregue el símbolo de Epsilon en la cadena. Una vez que tenemos la cadena unida, la recorremos para formar cadenas de 64 bits.

Para graficar, hacemos uso de la librería matplotlib. El programa tiene una función la cual se reciben los arreglos llenados en la función sigma, que usa para representarlos en la gráfica. Dado que el archivo generado para n=27 es muy grande y ocupa mucho espacio, se tomó una prueba con n=20, donde el archivo sí es manejable.

3.1 Implementación

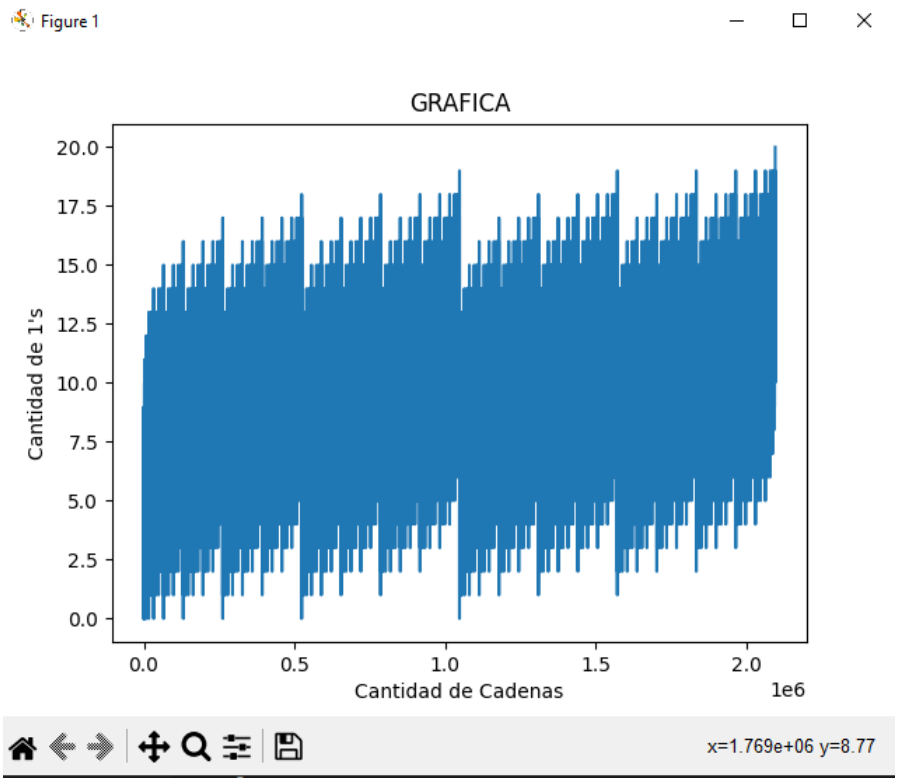


Archivo Generado



```
1.- Ingresar n
2.- Finalizar
1
Ingrese el valor de n: 20
n =  0
n =  1
n =  2
n =  3
n =  4
n =  5
n =  6
n =  7
n =  8
n =  9
n =  10
n =  11
n =  12
n =  13
n =  14
n =  15
n =  16
n =  17
n =  18
n =  19
n =  20
```

Consola



Grafica 1)

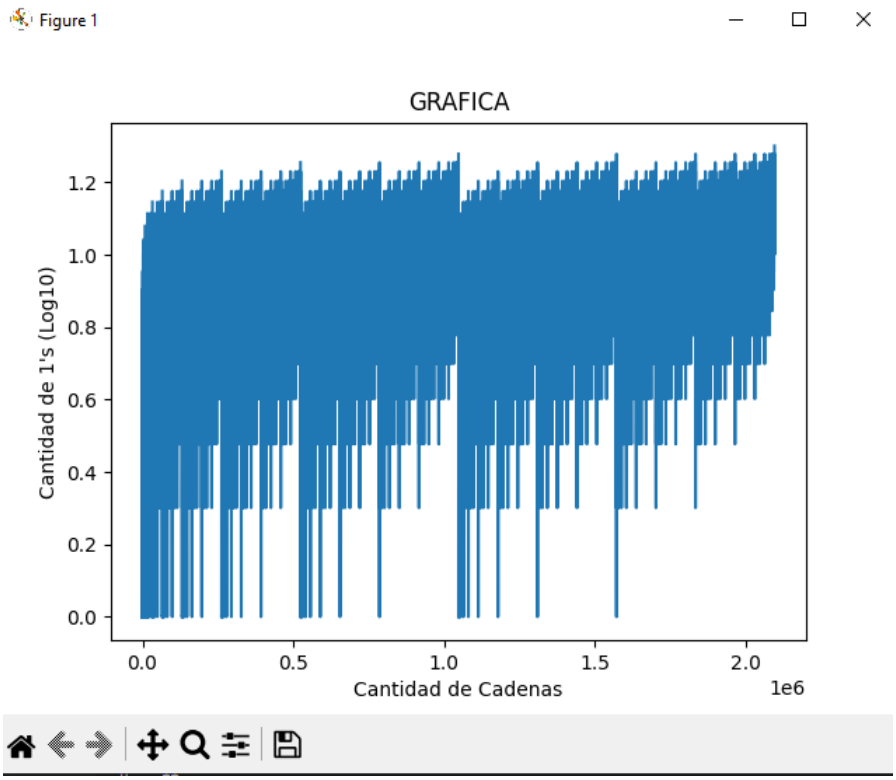
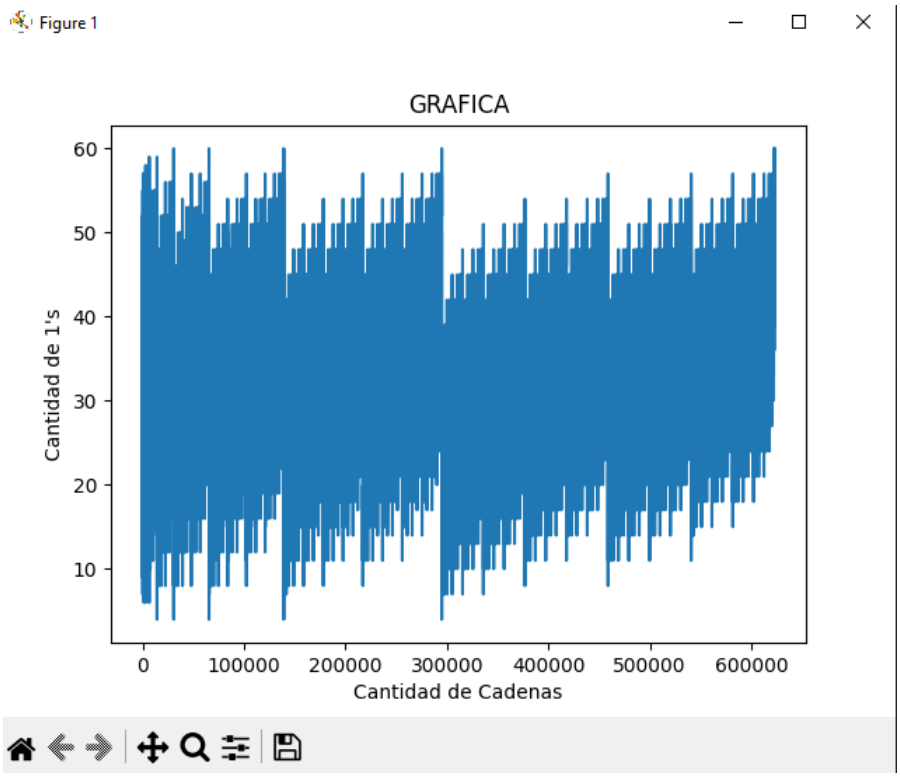
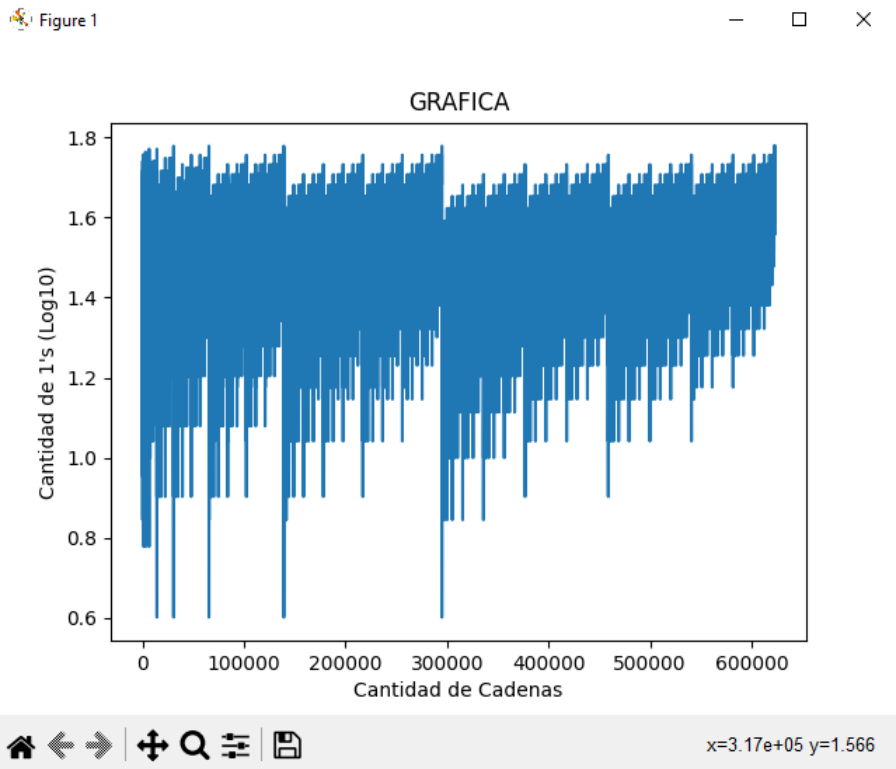


Figure 1
GRAFICA
Cantidad de 1's (Log10)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0.0
0.5
1.0
1.5
2.0
Cantidad de Cadenas
1e6

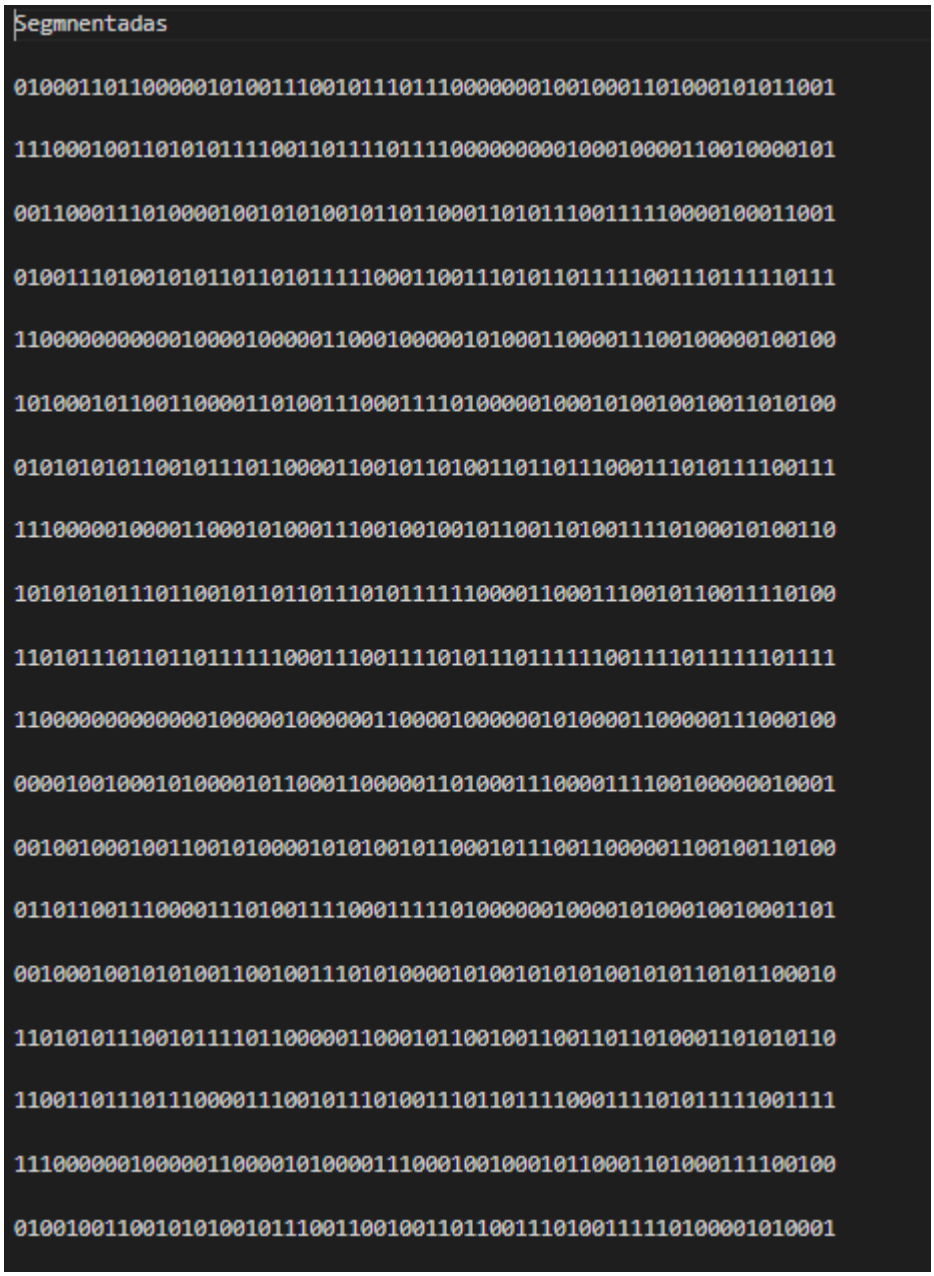
Grafica 2



Grafica 3 (Cadenas de 64 bits)



Grafica 4 (Cadenas de 64 bits)



Cadenas de 64 bits

References

[1] Sánchez, E. H. (2019). Lic. en Informática. Unidad de Apoyo para el Aprendizaje. https://programas.cuaed.unam.mx/repositorio/moodle/pluginfile.php/1163/modresource/content/1/contenido/index.html

[2] GeeksforGeeks. (2019, 16 octubre). N Queen Problem — Backtracking-3. https://www.geeksforgeeks.org/n-queen-problem-backtracking-3/?ref=rp

[3] Ocho reinas reina atacarr.JPG. (2006, 20 mayo). [Imagen]. Wikipedia. https://upload.wikimedia.org/wikipedia/commons/d/d6/Ocho_reinas_reina_atacarr.JPG

[4] Ocho reinas reina atacar fila. (2006, 10 mayo). [Imagen]. Wikipedia. https://upload.wikimedia.org/wikipedia/commons/a/ad/Ocho_reinas_reina_atacar_fila.JPG

[5] GeeksforGeeks. (2021, 9 febrero). Backtracking — Introduction. https://www.geeksforgeeks.org/backtracking-introduction/

[6] Backtracking. (s. f.). [Imagen]. InterviewBit. https://ibpublicimages.s3-us-west-2.amazonaws.com/tutorial/backtracking1.pngFigura 2

[7] [Árbol de búsqueda del problema de las 4 reinas]. (2015, 5 febrero). Ivan Voroshilin's Blog. https://ivoroshilin.files.wordpress.com/2015/02/backtrack.png