Alonso Rubén Maya Martínez

10 de Septiembre del 2019

1 Practica 1

1.1 Enunciado

Generar todas las cadenas binarias de tamaño n, contar cuantos 1's tiene cada cadena y graficar esta cantidad.

1.2 Código

```
import matplotlib.pyplot as plt
import numpy as np
n = int(input("Ingrese una n: "))

file = open("datos_1.txt", "w")
file.write('{')

cadena = ''

unos = []

def guardarCadenas(cadena, tam,n):
  if tam == n:
    file.write(cadena + ',')
    unos.append(cadena.count("1"))
    return
  cadena_0 = cadena+'0'
  cadena_1 = cadena+'1'
  guardarCadenas(cadena_0,tam+1,n)
  guardarCadenas(cadena_1,tam+1,n)

guardarCadenas(cadena,0,n)
file.write("}")
print("Cadena generada")
plt.scatter(np.array(range(len(unos))), unos, s=2)
plt.xlabel("Numero de Cadena")
plt.ylabel("Numero de 1's")
```

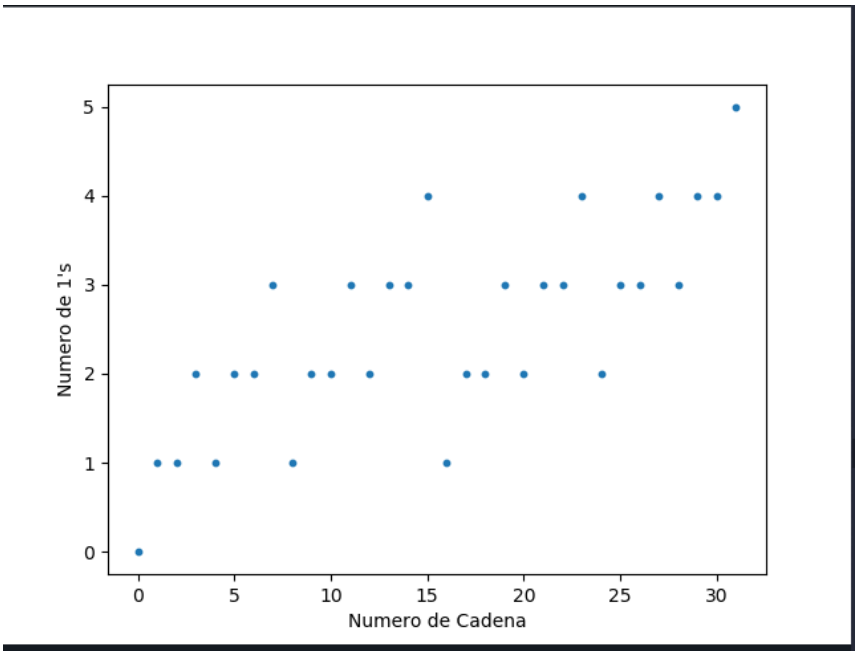
```
unos.clear()
plt.show()
```

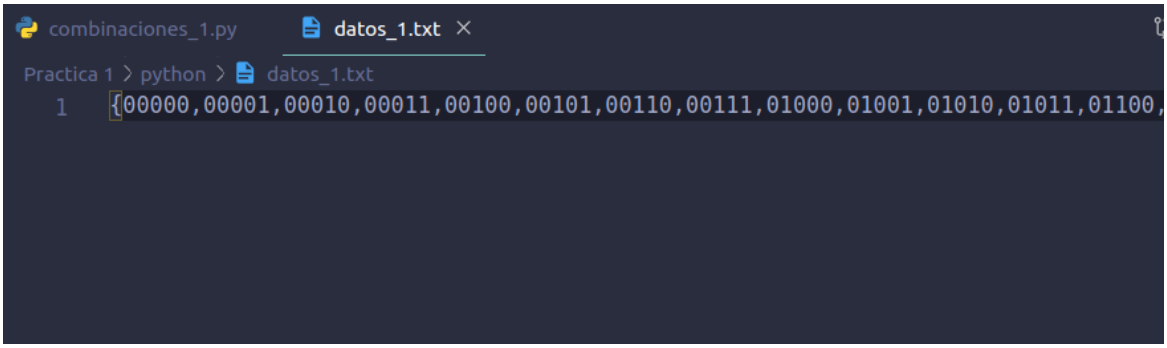
1.3 Ejemplo y capturas

Probamos con n = 5. El archivo de salida que nos genera contiene lo siguiente

{00000,00001,00010,00011,00100,00101,00110,00111,01000,
01001,01010,01011,01100,01101,01110,01111,10000,10001,
10010,10011,10100,10101,10110,10111,11000,11001,11010,
11011,11100,11101,11110,11111}

1.3.1 Capturas





2 Practica 2

2.1 Enunciado

Generar todas las potencias sobre el alfabeto 0,1 hasta la potencia n. Contar los 1's por cada cadena y graficarlo.

2.2 Código

```
import matplotlib.pyplot as plt
import numpy as np
n = int(input("Ingrese una n: "))

file = open("datos_2.txt", "w")
file.write('{E')

cadena = ''

unos = []

def guardarCadenas(cadena, tam,n):
  if tam == n:
    file.write(cadena + ',')
    unos.append(cadena.count("1"))
    return
  cadena_0 = cadena+'0'
  cadena_1 = cadena+'1'
  guardarCadenas(cadena_0,tam+1,n)
  guardarCadenas(cadena_1,tam+1,n)
for i in range(n+1):
  guardarCadenas(cadena,0,i)
file.write("}")
file.close()
print("Cadena generada")
plt.scatter(np.array(range(len(unos))), unos,  s=10)
plt.xlabel("Numero de Cadena")
plt.ylabel("Numero de 1's")
unos.clear()
plt.show()
```

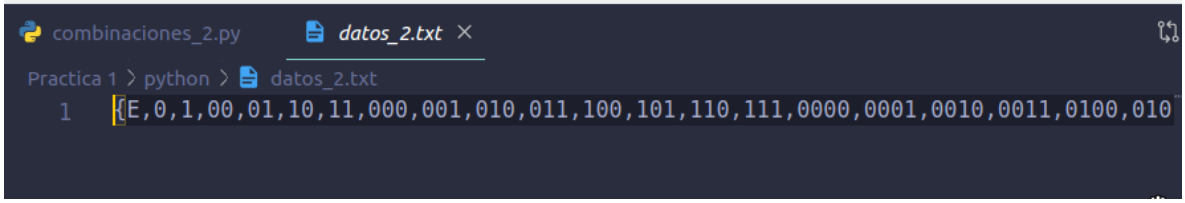
3 Ejemplo y capturas

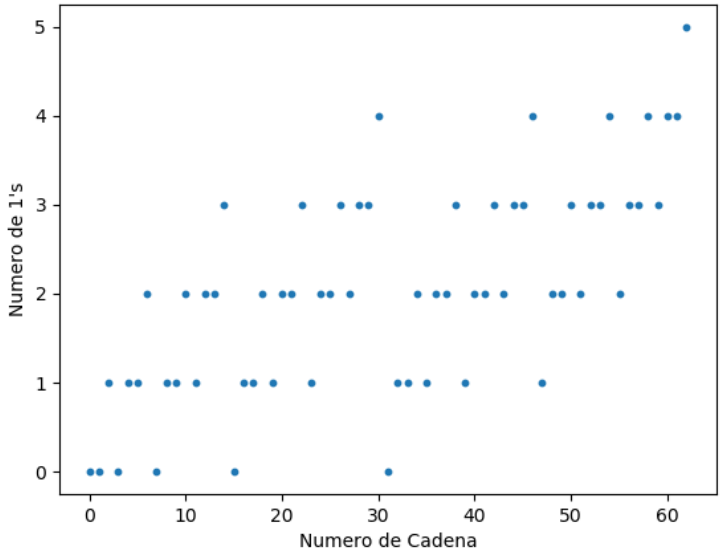
Probamos con n = 5. El archivo de salida que genera contiene lo siguiente:

{E,0,1,00,01,10,11,000,001,010,011,100,101,
110,111,0000,0001,0010,0011,0100,0101,0110,
0111,1000,1001,1010,1011,1100,1101,1110,1111,

00000,00001,00010,00011,00100,00101,00110,00111,
01000,01001,01010,01011,01100,01101,01110,01111,
10000,10001,10010,10011,10100,10101,10110,10111,
11000,11001,11010,11011,11100,11101,11110,11111}

3.0.1 Capturas





4 Practica 3

4.1 Enunciado

Generar todas las potencias sobre el alfabeto 0,1 hasta la potencia n y ponerlas todas juntas en un archivo. Dividir la cadena resultante en fragmentos de 8bits, contar los 1's por cada cadena y graficar.

4.2 Código

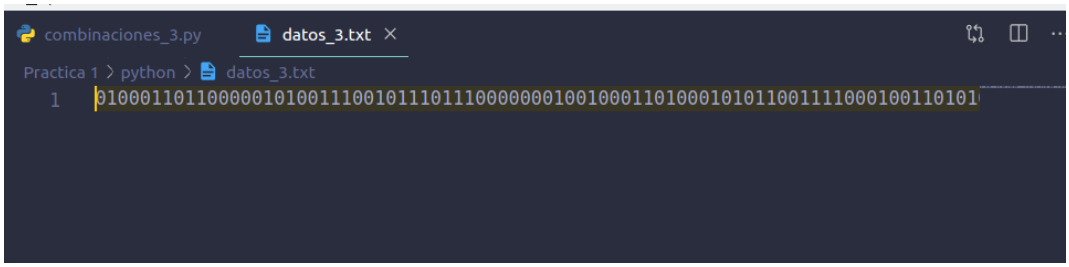
```
import matplotlib.pyplot as plt
import numpy as np
n = int(input("Ingrese una n: "))

file = open("datos_3.txt", "w")

cadena = ''

unos = []

def guardarCadenas(cadena, tam,n):
  if tam == n:
    file.write(cadena )
    return
  cadena_0 = cadena+'0'
  cadena_1 = cadena+'1'
  guardarCadenas(cadena_0 ,tam+1,n)
  guardarCadenas(cadena_1 ,tam+1,n)
for i in range(n+1):
  guardarCadenas(cadena ,0 , i)
file.close()
cadTemp = ""
bits = 0

print("Cadena generada")
with open("datos_3.txt") as f:
  while True:
    c = f.read(1)
    if not c:
      break
    bits += 1
    cadTemp += c
    if bits == 8:
      unos.append(cadTemp.count("1"))
      cadTemp = ""
      bits = 0
plt.scatter(np.array(range(len(unos))), unos)
unos.clear()
plt.show()
```

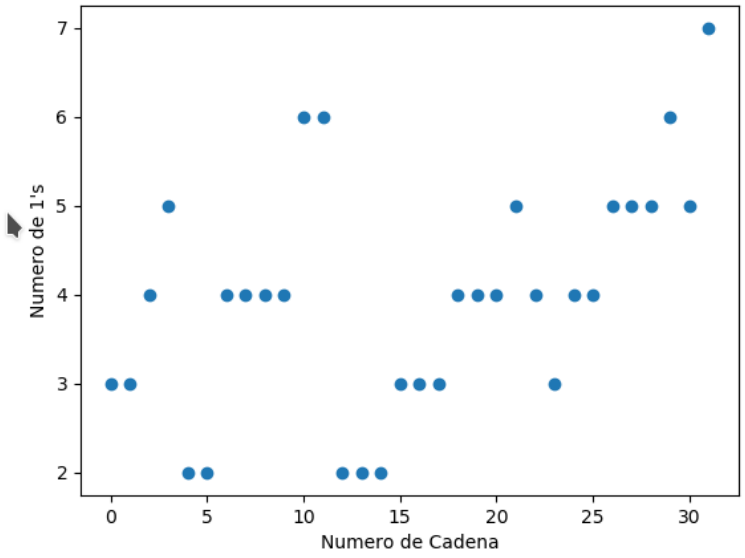
4.3 Ejemplo y capturas

Probamos con n = 5. El archivo de salida que genera contiene lo siguiente (Sin saltos de linea):

0100011011000001010011100101110111000000010010001101000
1010110011110001001101010111100110111101111000000000100
0100001100100001010011000111010000100101010010110110001
1010111001111100001000110010100111010010101101101011111
0001100111010110111110011101111101111

4.3.1 Capturas





5 Practica 4

5.1 Enunciado

Generar todas las potencias sobre el alfabeto 0,1 hasta la potencia n y ponerlas todas juntas en un archivo. Dividir la cadena resultante en fragmentos de 16bits, contar los 1's por cada cadena y graficar.

5.2 Código

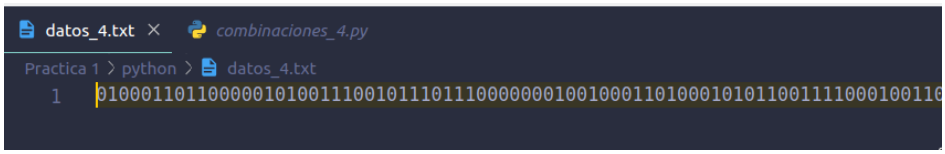
```
import matplotlib.pyplot as plt
import numpy as np
n = int(input("Ingrese una n: "))

file = open("datos_3.txt", "w")

cadena = ''

unos = []

def guardarCadenas(cadena, tam,n):
  if tam == n:
    file.write(cadena )
    return
  cadena_0 = cadena+'0'
  cadena_1 = cadena+'1'
  guardarCadenas(cadena_0 ,tam+1,n)
  guardarCadenas(cadena_1 ,tam+1,n)
for i in range(n+1):
  guardarCadenas(cadena ,0 , i)
file.close()
cadTemp = ""
bits = 0

print("Cadena generada")
with open("datos_3.txt") as f:
  while True:
    c = f.read(1)
    if not c:
      break
    bits += 1
    cadTemp += c
    if bits == 8:
      unos.append(cadTemp.count("1"))
      cadTemp = ""
      bits = 0
plt.scatter(np.array(range(len(unos))), unos)
unos.clear()
plt.show()
```

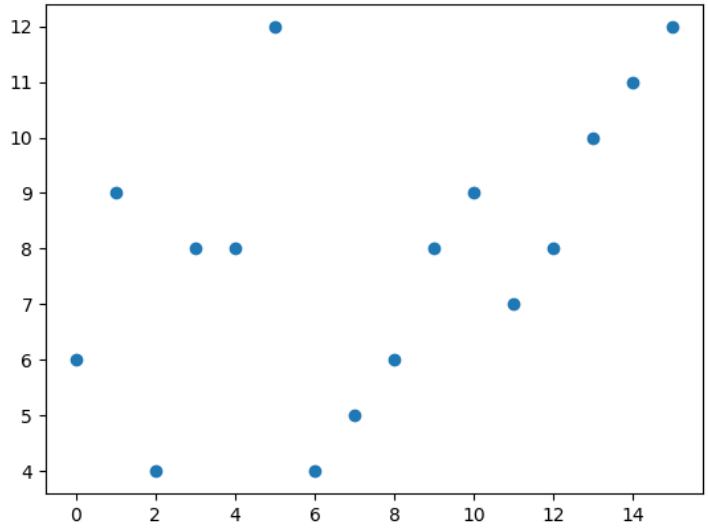
5.3 Ejemplo y capturas

Probamos con n = 5. El archivo de salida que genera contiene lo siguiente (Sin saltos de linea):

0100011011000001010011100101110111000000010010001101000
1010110011110001001101010111100110111101111000000000100
0100001100100001010011000111010000100101010010110110001
1010111001111100001000110010100111010010101101101011111
0001100111010110111110011101111101111

5.3.1 Capturas





6 Practica 4

6.1 Enunciado

Generar todos los numeros primos del 2 hasta n, convertir ese numero a su forma binaria, contar los 1's en esa cadena y graficar.

7 Código

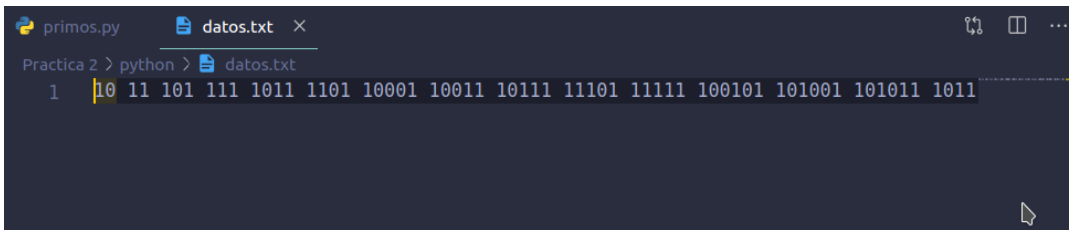
```
import matplotlib.pyplot as plt
import numpy as np
n = int(input("Ingrese maximo: "))

x = 0
prims = [0] * (n+1)
nums = [0] * (n+1)
file = open("datos.txt", "w")
unos = []
for i in range(2,n+1):
  if nums[i] != 1 or i == 2:
    prims[0] = i
    cad = str(bin(i)[2:])
    file.write(cad+"\n")
    unos.append(cad.count("1"))
    p = 2
    while p*i <= n:
      nums[p*i] = 1
      p+=1
    x+=1

file.close()
plt.scatter(np.array(range(len(unos))), unos)
plt.xlabel("Numero de Cadena")
plt.ylabel("Numero de 1's")
unos.clear()
plt.show()
```

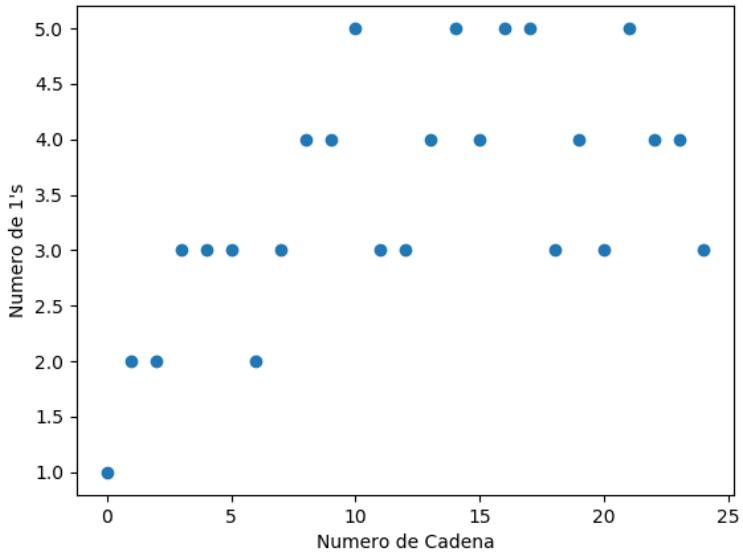
7.1 Ejemplo y capturas

Probamos con n = 100. El archivo de salida que genera contiene lo siguiente:

```
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
```

7.1.1 Capturas





8 Practica 6

8.1 Enunciado

8.2 Código

8.3 Clase DFA

```
# Automata mejorado

from graphviz import Digraph


class FiniteAutomata(object):

    "Clase que representa al automata finito"

    def __init__(self, num_Q, sigma, delt,
                 q_0, F, guardarPasos=False):

        Q_list = []
        self.delta = {}
        for i in range(num_Q):
            Q_list.append("q"+str(i+1))
```

```
        self.Q = Q_list
        self.sigma = sigma
        self.delta = delt
        self.completar_diccionario()

        self.q_0 = q_0
        self.F = F
        self.estado_actual = q_0
        self.guardarPasos = guardarPasos
        if self.guardarPasos:
            self.archivo = open("pasos.txt", "w")

    def get_estado_actual(self):
        return self.estado_actual

    def get_estado_inicial(self):
        return self.q_0

    def completar_diccionario(self):
        for i in self.delta.keys():
            for j in self.sigma:
                if not j in self.delta[i].keys():
                    self.delta[i][j] = None

    def prueba(self, cadena):

        if self.guardarPasos:
            self.archivo.write
                        ("Cadena: " + cadena + "\n")

        for i in cadena:
            siguiente_estado = self.delta[self.estado_actual][i]

            if self.guardarPasos:
                self.archivo.write
                            (i + " " +
                            self.estado_actual + " -> " +
                            siguiente_estado + '\n')

            self.estado_actual = siguiente_estado

            if self.estado_actual == None:
                self.estado_actual =
                        self.get_estado_inicial()
                return False
        if self.estado_actual in self.F:
```

```
            self.estado_actual =
                    self.get_estado_inicial()
            self.archivo.write("El automata SI acepta la
 cadena\n")
            return True
        else:
            self.estado_actual =
                    self.get_estado_inicial()
            self.archivo.write
                    ("El automata NO acepta la cadena\n")
            return False

    def drawn(self):

        # inicializa el diagrama
        f = Digraph('finite_state_machine',
                filename='fsm.gv', format='png')
        f.attr(rankdir='LR', size='8,5')

        # Dibuja los nodos finales con doblecirculo
        f.attr('node', shape='doublecircle')

        for i in self.F:
            f.node(i)

        f.attr('node', shape='circle')

        # Agrega todos los nodos de todos los estados
        for key in self.delta.keys():
            f.node(key)
        #Agrega todas las conexiones
        #Cuando una misma conexion tiene varios
        #inputs los une
        en una sola flecha
        for estado, conexiones in self.delta.items():
            contados = []
            for nombre, conex in conexiones.items():
                count = 0
                repetidos = []
                if conex in contados or conex == None:
                    continue
                for nombreTemp, conexTemp in
                conexiones.items():
                    if nombreTemp in repetidos:
                        continue
                    if conex == conexTemp:
```

```
                    count += 1
                    repetidos.append(nombreTemp)
            contados.append(conex)
            if len(repetidos) == 1:
                f.edge(estado, conex, nombre)
            else:
                cadena = ""
                for i in repetidos:
                    cadena += (i+",")
                cadena = cadena[:-1]
                f.edge(estado, conex, cadena)

        #Se crea un nodo transparente para poner la
        #flecha del estado inicial
        f.attr('node', style='filled')
        f.attr('node', color='white')
        f.edge('', "q1")

        f.view()
```

8.3.1 Protocolos

```
from DFA import FiniteAutomata
import string
import random
import progressbar
import time

#Automata de paridad
q = 4
sigma = ["0", "1"]
q_0 = "q1"
F = ["q1"]
delta={
        "q1":{"0":"q2", "1":"q4"},
        "q2":{"0":"q1", "1":"q3"},
        "q3":{"0":"q4", "1":"q2"},
        "q4":{"0":"q3", "1":"q1"}
        }
F_A = FiniteAutomata(q,sigma,delta,q_0,F,True)

numCadenas = int(input(
    "Ingrese el numero de cadenas que desea generar: "))
tamCadenas = int(input("Ingrese el tama o de las cadenas: "))
```

```
while random.randint(0,1):
  file = open("datos.txt", "w")
  print("El protocolo inicio!")
  print("Generando cadenas!")
  for i in progressbar.progressbar(range(numCadenas)):
    cadTemp = ""
    for j in range(tamCadenas):
      cadTemp += str(random.randint(0,1))
    file.write(cadTemp+ "\n")
  file.close()
  print("Se generaron las cadenas!")
  file = open("datos.txt", "r")
  print("Probando las cadenas!")
  for i in progressbar.progressbar(range(numCadenas)):
      line = file.readline()
      F_A.prueba(line[:-1])
  print("Cadenas probadas!")
  file.close()
  print("2s Timeout")
  time.sleep(2)

print("Fin de los protocolos")
```

8.4 Ejemplo y capturas

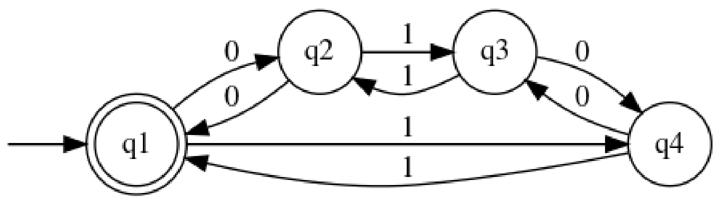
Si probamos generando 10 cadenas de tamaño 10 entonces puede que al esperar el aleatorio de 0 y termine el programa. De lo contrario empieza a generar las cadenas y determina con el automata si son pares o no. Las cadenas que genera son estas: 1000110100 0101011001 1010011110 0011001000 0010110100 0011100011 1111111110 0111011101 0001111100 0001110010 0110111001 0000001010 0111111010 0110000110 0110001000 0100101001 1110101010 1011101011 1011100100 0110111110 0010001111 1011001010 0001010100 1000101100 1001111111 1101011111 0011001111 0111111110 0111110011 0011010000 0110000110 0110101010 1010011111 0111110000 0001000001 0010100111 1001100011 0111001011 1100001101 1011111110 Y el una parte del archivo en donde se guarda el recorrido del automata queda asi:

```
Cadena: 1000110100
1 q1 -> q4
0 q4 -> q3
0 q3 -> q4
0 q4 -> q3
1 q3 -> q2
1 q2 -> q3
0 q3 -> q4
1 q4 -> q1
0 q1 -> q2
0 q2 -> q1
```

```
El automata SI acepta la cadena
Cadena: 0101011001
0 q1 -> q2
1 q2 -> q3
0 q3 -> q4
.
.
.
```

Despues te este proceso el programa tiene un tiempo de espera de 2 segundos para generar el siguiente numero aleatorio. Este proceso de generación y prueba de cadenas se vuelva a repetir hasta que en random de 0.

8.4.1 Capturas




```
Ingrese el numero de cadenas que desea generar: 10
Ingrese el tamaño de las cadenas: 10
El protocolo inicio!
Generando cadenas!
100% (10 of 10) |####################################| Elapsed Time: 0:00:00 Time:  0:00:00
Se generaron las cadenas!
Probando las cadenas!
100% (10 of 10) |####################################| Elapsed Time: 0:00:00 Time:  0:00:00
Cadenas probadas!
2s Timeout
El protocolo inicio!
Generando cadenas!
100% (10 of 10) |####################################| Elapsed Time: 0:00:00 Time:  0:00:00
Se generaron las cadenas!
Probando las cadenas!
100% (10 of 10) |####################################| Elapsed Time: 0:00:00 Time:  0:00:00
Cadenas probadas!
2s Timeout
El protocolo inicio!
Generando cadenas!
100% (10 of 10) |####################################| Elapsed Time: 0:00:00 Time:  0:00:00
Se generaron las cadenas!
Probando las cadenas!
100% (10 of 10) |####################################| Elapsed Time: 0:00:00 Time:  0:00:00
Cadenas probadas!
2s Timeout
Fin de los protocolos
```

paridad.py datos.txt

Practica 3 > datos.txt

```
1000110100
0101011001
1010011110
0011001000
0010110100
0011100011
1111111110
0111011101
0001111100
0001110010
0110111001
0000001010
0111111010
0110000110
0110001000
0100101001
1110101010
1011101011
1011100100
0110111110
0010001111
1011001010
0001010100
1000101100
```

paridad.py pasos.txt

Practica 3 > pasos.txt

```
Cadena: 1000110100
1 q1 -> q4
0 q4 -> q3
0 q3 -> q4
0 q4 -> q3
1 q3 -> q2
1 q2 -> q3
0 q3 -> q4
1 q4 -> q1
0 q1 -> q2
0 q2 -> q1
El automata SI acepta la cadena
Cadena: 0101011001
0 q1 -> q2
1 q2 -> q3
0 q3 -> q4
1 q4 -> q1
0 q1 -> q2
1 q2 -> q3
1 q3 -> q2
0 q2 -> q1
0 q1 -> q2
1 q2 -> q3
El automata NO acepta la cadena
Cadena: 1010011110
```

9 Practica 7