Programa Universo

Serrano Gayosso José Eduardo
Grupo: 4CV2

2 de octubre de 2024

1 Introducción

En teoría de lenguajes formales y autómatas, un **alfabeto** se define como un conjunto finito y no vacío de símbolos, comúnmente representado como Σ. Por ejemplo, el alfabeto binario se denota como $\Sigma = \{0, 1\}$. El **universo** de un alfabeto está compuesto por todas las posibles cadenas (incluyendo la cadena vacía) que se pueden formar con los símbolos del alfabeto. Las cadenas son secuencias finitas de símbolos pertenecientes al alfabeto, y su longitud se denota con el número de símbolos en la cadena.

Formalmente, el conjunto de todas las cadenas posibles que se pueden construir a partir de un alfabeto Σ se representa como Σ^*. Este conjunto incluye todas las combinaciones posibles de los símbolos en Σ, desde la cadena vacía hasta cadenas de longitud infinita en teoría, aunque en la práctica se suelen considerar cadenas hasta una longitud máxima N.

En esta práctica, se calculará el universo para el alfabeto $\{0, 1\}$ con cadenas de longitud máxima $N = 29$. Esto implica generar todas las cadenas binarias posibles de longitud hasta 29, contando la cantidad de ceros y unos en cada una, y posteriormente realizar un análisis gráfico de estos resultados.

2 Descripción del Programa

El universo Σ^* de un alfabeto $\{0, 1\}$ está compuesto por todas las posibles cadenas binarias de longitud finita. Para ilustrar el cálculo de Σ^*, este se hará hasta una longitud máxima $N = 3$, por lo que se presentan los siguientes subconjuntos para cada valor de i desde 0 hasta 3:

- Para $i = 0$, el único elemento es la cadena vacía: $\Sigma^0 = \{\epsilon\}$.

- Para $i = 1$, se tienen las cadenas binarias formadas por 1 dígito: $\Sigma^1 = \{0, 1\}$.
- Para $i = 2$, las cadenas posibles de 2 dígitos son: $\Sigma^2 = \{00, 01, 10, 11\}$. Esto corresponde a los números decimales del 0 al $2^2 - 1 = 3$ en binario, representados con 2 dígitos.
- Para $i = 3$, las cadenas posibles de 3 dígitos son: $\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$. Esto equivale a los números decimales del 0 al $2^3 - 1 = 7$ en binario, con representaciones de 3 dígitos.

Como se puede observar, para cada i, el conjunto de cadenas binarias es equivalente a los números del 0 al $2^i - 1$ representados en binario con i dígitos. Σ^* está dado por la unión de todos los Σ^i, por lo tanto, una manera eficiente de calcular el universo Σ^* para un valor N dado es generar todos los números decimales desde 0 hasta $2^i - 1$ y convertirlos a su representación binaria de i dígitos para cada conjunto de la unión, desde $i = 0$ hasta $i = N$.

2.1 Métodos del Programa

El programa contiene varios métodos que desempeñan funciones específicas para generar, contar y graficar las cadenas binarias. A continuación se presenta una descripción detallada de cada uno de ellos.

2.1.1 Método `Binario`

El método `Binario` es responsable de convertir un número decimal en su representación binaria con una longitud específica. Este método recibe los siguientes parámetros:

- `int decimal`: El número decimal que se desea convertir a binario.
- `int corrida`: El número de dígitos que debe tener la representación binaria resultante.

El funcionamiento de este método se describe a continuación:

1. Se crea un arreglo de caracteres llamado `binario` de longitud igual a `corrida + 1`, y se inicializa con '0's usando el método `Arrays.fill`. Esto garantiza que la cadena resultante tendrá la longitud adecuada, llenando con ceros a la izquierda si es necesario.

2. Se verifica si el número decimal es 0. Si es así, se convierte el arreglo en una cadena y se devuelve inmediatamente, ya que la representación binaria de 0 es simplemente una cadena de ceros.

3. Si el número es distinto de 0, el método entra en un ciclo `while` que continúa dividiendo el número decimal por 2 hasta que el número se vuelva 0:

    - En cada iteración, se calcula el residuo de la división por 2, el cual determina si el bit correspondiente es un '1' o un '0'.
    - Si el residuo es 1, se asigna '1' en la posición correspondiente del arreglo de caracteres.
    - El número decimal se divide por 2 y el contador se incrementa para llenar la siguiente posición en la representación binaria.

4. Finalmente, el arreglo de caracteres se convierte en una cadena y se devuelve como resultado. Este proceso garantiza que la cadena binaria resultante tenga una longitud igual a `corrida + 1`, incluso si el número original tiene menos dígitos.

Este método es esencial para generar las cadenas binarias en el programa, ya que convierte números decimales a su representación binaria de una longitud específica, lo cual se requiere para construir el universo de cadenas binarias.

2.1.2 Método `contarCeros`

El método `contarCeros` cuenta la cantidad de ceros en una cadena binaria dada. Este método recibe el siguiente parámetro:

- `String binario`: La cadena binaria cuya cantidad de ceros se va a contar.

Su funcionamiento es el siguiente:

1. Se inicializa un contador llamado `cuenta` en 0.

2. El método recorre cada carácter de la cadena binaria utilizando un ciclo `for`.

3. Para cada caracter:

    - Si el caracter es 0, el contador se incrementa en 1.

4. Después de recorrer toda la cadena, el valor del contador, que representa la cantidad de ceros en la cadena, se devuelve.

2.1.3 Método `generarGrafica`

El método `generarGrafica` se encarga de crear y mostrar las gráficas basadas en las listas de cantidad de ceros y unos para las cadenas generadas. Este método recibe los siguientes parámetros:

- `List<Integer> ceros`: Una lista que contiene la cantidad de ceros en cada cadena binaria generada.
- `List<Integer> unos`: Una lista que contiene la cantidad de unos en cada cadena binaria generada.

El método realiza las siguientes acciones:

1. **Creación de Series de Datos:**
   (a) Se crean dos series (`XYSeries`) llamadas `serieCeros` y `serieUnos` para almacenar los datos originales de la cantidad de ceros y unos.
   (b) Se crean dos series adicionales (`serieCerosLog` y `serieUnosLog`) para almacenar los valores logarítmicos de la cantidad de ceros y unos.
2. **Rellenar las Series:**
   (a) Se recorre cada elemento de las listas `ceros` y `unos`, y se agregan a las series originales con su correspondiente índice.
   (b) Para las series logarítmicas, se calcula el logaritmo base 10 de cada valor (agregando 1 para evitar problemas con log(0)) y se agregan a las series `serieCerosLog` y `serieUnosLog`.
3. **Creación de los Conjuntos de Datos:**
   (a) Se crean cuatro conjuntos de datos (`XYSeriesCollection`), uno para cada serie, que se utilizarán para generar las gráficas.
4. **Creación de las Gráficas:**
   (a) Se crean cuatro gráficas de dispersión (`scatter plots`) usando el método `ChartFactory.createScatterPlot`:
      i. Una gráfica para la cantidad de ceros.
      ii. Una gráfica para la cantidad de unos.
      iii. Una gráfica para el logaritmo base 10 de la cantidad de ceros.
      iv. Una gráfica para el logaritmo base 10 de la cantidad de unos.

5. **Mostrar las Gráficas:**

   (a) Cada gráfica se asigna a un `JFrame` que se configura para mostrarse en una ventana independiente. Se utiliza `JFrame.DISPOSE_ON_CLOSE` para permitir que las ventanas se cierren sin terminar el programa.

   (b) Las gráficas se añaden a los marcos (`frames`) y se hacen visibles para el usuario.

Este método es esencial para la visualización de los datos generados por el programa, ya que permite analizar de manera gráfica la distribución de ceros y unos en las cadenas binarias.

2.2 Cálculo del universo

El método `main` es el punto de entrada del programa y controla todo el flujo de ejecución. A continuación, se detalla su funcionamiento, los parámetros que maneja, y las acciones que realiza:

1. **Declaración de Variables:**

   (a) Se declaran dos variables: `int opc` para almacenar la opción elegida por el usuario y `int n` para representar la longitud máxima de las cadenas binarias a generar.

   (b) Se crea una instancia de `Scanner` llamada `scanner` para capturar la entrada del usuario desde la consola.

2. **Presentación del Menú:**

   (a) El programa entra en un ciclo `while (true)` para presentar el menú de opciones al usuario.

   (b) Dentro del ciclo, se usa un `do-while` para asegurar que el usuario elija una opción válida:

      i. Se imprimen tres opciones:
         - 1) Elegir N.
         - 2) Dejar que la máquina decida N de forma aleatoria.
         - 3) Salir.

      ii. El programa lee la entrada del usuario y almacena la opción seleccionada en la variable `opc`.

   (c) Si el usuario selecciona la opción 3, el programa llama a `System.exit(0)` para terminar la ejecución.

3. **Definición del Valor de N:**

   (a) Si el usuario elige la opción 1:

      i. Se inicia un ciclo `do-while` que solicita al usuario ingresar un valor para N.

      ii. El valor ingresado se almacena en la variable `n` y se imprime en la consola.

      iii. El ciclo continúa hasta que el valor de N sea un número válido (entre 0 y 1000).

   (b) Si el usuario elige la opción 2:

      i. Se crea un objeto `Random` para generar un número aleatorio.

      ii. El valor aleatorio para N se genera en el rango de 1 a 1000 y se almacena en la variable `n`.

      iii. Se imprime el valor de N seleccionado automáticamente.

4. **Inicialización de Listas para Almacenar Resultados:**

   (a) Se crean dos listas: `List<Integer> cerosList` y `List<Integer> unosList` para almacenar la cantidad de ceros y unos de cada cadena binaria generada.

5. **Generación del Universo de Cadenas Binarias:**

   (a) El programa intenta abrir un archivo llamado `salida.txt` para escribir los resultados, utilizando un bloque `try-with-resources` para manejar automáticamente el cierre del archivo al finalizar.

   (b) Se escribe el inicio del conjunto de cadenas binarias en el archivo con la cadena "`Sigma* = {epsilon`", que representa el comienzo de la declaración del universo junto con su primer elemento: la cadena vacía.

   (c) El programa entra en un ciclo doble anidado para generar cadenas binarias:

      i. El ciclo externo recorre valores de 0 hasta $n-1$, representando la longitud de las cadenas.

      ii. El ciclo interno genera números binarios desde 0 hasta $2^{i+1}-1$ usando el método `Binario`, donde i es la corrida, o dicho de otra manera, la longitud actual.

      iii. Para cada número generado:

         - Se convierte a binario y se escribe en el archivo.

- Se llama al método `contarCeros` para contar los ceros en la cadena binaria.
- Se calcula la cantidad de unos restando la cantidad de ceros al total de dígitos en la cadena.
- Los valores de ceros y unos se almacenan en las listas `cerosList` y `unosList`.

(d) El ciclo final genera las cadenas binarias de longitud n utilizando el mismo proceso, con la diferencia de que, si se trata de la última cadena de todas, lo siguiente que se imprimirá es "}" para terminar con la declaración del universo.

6. **Llamada al Método `generarGrafica`:**

   (a) Una vez completada la generación y conteo de las cadenas binarias, se llama al método `generarGrafica` pasando las listas `cerosList` y `unosList` como argumentos.

   (b) El método `generarGrafica` crea y muestra las gráficas correspondientes.

7. **Manejo de Excepciones:**

   (a) Si ocurre un error durante la escritura en el archivo, se captura una excepción `IOException` y se imprime un mensaje de error en la consola.

2.3 Código fuente

```
1  package com.mycompany.universo;
2
3  import java.io.FileWriter;
4  import java.io.IOException;
5  import java.io.PrintWriter;
6  import java.util.Arrays;
7  import java.util.ArrayList;
8  import java.util.List;
9  import java.util.Random;
10 import java.util.Scanner;
11 import org.jfree.chart.ChartFactory;
12 import org.jfree.chart.ChartPanel;
13 import org.jfree.chart.JFreeChart;
14 import org.jfree.chart.plot.PlotOrientation;
15 import org.jfree.data.xy.XYSeries;
16 import org.jfree.data.xy.XYSeriesCollection;
17 import javax.swing.JFrame;
```

```
18
19 public class Universo {
20
21     public static String Binario(int decimal, int corrida) {
22         char[] binario = new char[corrida + 1];
23         Arrays.fill(binario, '0');
24
25         if (decimal == 0) {
26             return new String(binario);
27         }
28
29         int residuo;
30         int contador = 0;
31         while (decimal > 0) {
32             residuo = decimal % 2;
33             if (residuo == 1) {
34                 binario[(binario.length - 1) - contador] = '1
    ';
35             }
36             decimal = decimal / 2;
37             contador++;
38         }
39         return new String(binario);
40     }
41
42     public static int contarCeros(String binario) {
43         int cuenta = 0;
44         for (char c : binario.toCharArray()) {
45             if (c == '0') {
46                 cuenta++;
47             }
48         }
49         return cuenta;
50     }
51
52     public static void generarGrafica(List<Integer> ceros,
    List<Integer> unos) {
53         XYSeries serieCeros = new XYSeries("Ceros");
54         XYSeries serieUnos = new XYSeries("Unos");
55
56         XYSeries serieCerosLog = new XYSeries("Log10(Ceros)")
    ;
57         XYSeries serieUnosLog = new XYSeries("Log10(Unos)");
58
59         for (int i = 0; i < ceros.size(); i++) {
60             int cantidadCeros = ceros.get(i);
61             int cantidadUnos = unos.get(i);
62
63             serieCeros.add(i + 1, cantidadCeros);
```

```
64              serieUnos.add(i + 1, cantidadUnos);
65
66              double logCeros = Math.log10(cantidadCeros + 1);
67              double logUnos = Math.log10(cantidadUnos + 1);
68
69              serieCerosLog.add(i + 1, logCeros);
70              serieUnosLog.add(i + 1, logUnos);
71          }
72
73          XYSeriesCollection datasetCeros = new
    XYSeriesCollection();
74          datasetCeros.addSeries(serieCeros);
75
76          XYSeriesCollection datasetUnos = new
    XYSeriesCollection();
77          datasetUnos.addSeries(serieUnos);
78
79          XYSeriesCollection datasetCerosLog = new
    XYSeriesCollection();
80          datasetCerosLog.addSeries(serieCerosLog);
81
82          XYSeriesCollection datasetUnosLog = new
    XYSeriesCollection();
83          datasetUnosLog.addSeries(serieUnosLog);
84
85          JFreeChart chartCeros = ChartFactory.
    createScatterPlot(
86                  "Numero de Ceros por Cadena", "Cadena", "
    Cantidad de Ceros",
87                  datasetCeros, PlotOrientation.VERTICAL, true,
     true, false);
88
89          JFreeChart chartUnos = ChartFactory.createScatterPlot
    (
90                  "Numero de Unos por Cadena", "Cadena", "
    Cantidad de Unos",
91                  datasetUnos, PlotOrientation.VERTICAL, true,
    true, false);
92
93          JFreeChart chartCerosLog = ChartFactory.
    createScatterPlot(
94                  "Log10(Numero de Ceros) por Cadena", "Cadena"
    , "Log10(Cantidad de Ceros)",
95                  datasetCerosLog, PlotOrientation.VERTICAL,
    true, true, false);
96
97          JFreeChart chartUnosLog = ChartFactory.
    createScatterPlot(
98                  "Log10(Numero de Unos) por Cadena", "Cadena",
```

```
        "Log10(Cantidad de Unos)",
99                  datasetUnosLog, PlotOrientation.VERTICAL,
    true, true, false);
100
101         JFrame frameCeros = new JFrame("Grafica de Ceros");
102         frameCeros.setDefaultCloseOperation(JFrame.
    DISPOSE_ON_CLOSE);
103         frameCeros.getContentPane().add(new ChartPanel(
    chartCeros));
104         frameCeros.pack();
105         frameCeros.setVisible(true);
106
107         JFrame frameUnos = new JFrame("Grafica de Unos");
108         frameUnos.setDefaultCloseOperation(JFrame.
    DISPOSE_ON_CLOSE);
109         frameUnos.getContentPane().add(new ChartPanel(
    chartUnos));
110         frameUnos.pack();
111         frameUnos.setVisible(true);
112
113         JFrame frameCerosLog = new JFrame("Grafica de Log10(
    Ceros)");
114         frameCerosLog.setDefaultCloseOperation(JFrame.
    DISPOSE_ON_CLOSE);
115         frameCerosLog.getContentPane().add(new ChartPanel(
    chartCerosLog));
116         frameCerosLog.pack();
117         frameCerosLog.setVisible(true);
118
119         JFrame frameUnosLog = new JFrame("Grafica de Log10(
    Unos)");
120         frameUnosLog.setDefaultCloseOperation(JFrame.
    DISPOSE_ON_CLOSE);
121         frameUnosLog.getContentPane().add(new ChartPanel(
    chartUnosLog));
122         frameUnosLog.pack();
123         frameUnosLog.setVisible(true);
124     }
125
126
127     public static void main(String[] args) {
128         int opc;
129         int n;
130         Scanner scanner = new Scanner(System.in);
131
132         while (true) {
133             do {
134                 System.out.println("1) Elegir N\n2) Dejar que
     la maquina decida \n3) Salir");
```

```
135                 opc = scanner.nextInt();
136             } while (opc != 1 && opc != 2 && opc != 3);
137 
138             if (opc == 3) {
139                 System.exit(0);
140             }
141 
142             if (opc == 1) {
143                 do {
144                     System.out.println("\nIngresa la N:");
145                     n = scanner.nextInt();
146                     System.out.println("\nN = " + n + "\n");
147                 } while (n < 0 || n > 1000);
148             } else {
149                 Random rand = new Random();
150                 n = rand.nextInt(1000) + 1;
151                 System.out.println("\nN = " + n + "\n");
152             }
153 
154             List<Integer> cerosList = new ArrayList<>();
155             List<Integer> unosList = new ArrayList<>();
156 
157             try (PrintWriter writer = new PrintWriter(new
    FileWriter("C:/Users/JESG/OneDrive - Instituto Politecnico
     Nacional/Documents/salida.txt"))) {
158                 writer.print("Sigma* = {epsilon ");
159                 for (int i = 0; i < n - 1; i++) {
160                     for (int j = 0; j < Math.pow(2, (i + 1));
     j++) {
161                         String binario = Binario(j, i);
162                         writer.print(binario + ", ");
163 
164                         int ceros = contarCeros(binario);
165                         int unos = binario.length() - ceros;
166 
167                         cerosList.add(ceros);
168                         unosList.add(unos);
169                     }
170                     System.out.println("N = " + (i + 1));
171                 }
172 
173                 for (int i = 0; i < Math.pow(2, n); i++) {
174                     String binario = Binario(i, n - 1);
175                     if (i == Math.pow(2, n) - 1)
176                         writer.print(binario + "}");
177                     else
178                         writer.print(binario + ", ");
179 
180                     int ceros = contarCeros(binario);
```

```
                    int unos = binario.length() - ceros;

                    cerosList.add(ceros);
                    unosList.add(unos);
                }
                System.out.println("N = " + n);

                generarGrafica(cerosList, unosList);

            } catch (IOException e) {
                System.err.println("Error al escribir en el
    archivo: " + e.getMessage());
            }
        }
    }
}
```

Listing 1: Código fuente

3 Ejecución para $N = 29$

Al ejecutar el programa con $N = 29$, se presentaron los siguientes resultados:

3.1 Ejecución del Código

```
1) Elegir N
2) Dejar que la maquina decida
3) Salir
1

Ingresa la N:
29

N = 29

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
N = 21
N = 22
N = 23
N = 24
N = 25
N = 26
N = 27
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at java.base/java.util.Arrays.copyOf(Arrays.java:3512)
    ...
    at com.mycompany.universo.Universo.main(Universo.java:172)
```

3.2 Error

Durante la ejecución, se presentó un error de falta de memoria (*OutOfMemoryError*), indicando que el programa intentó utilizar más memoria de la que está disponible en la Máquina Virtual de Java (JVM). Esto sucedió debido al crecimiento exponencial de cadenas binarias que se intentaron generar y almacenar. La traza del error es la siguiente:

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at java.base/java.util.Arrays.copyOf(Arrays.java:3512)
    at java.base/java.util.ArrayList.grow(ArrayList.java:238)
    ...
```

3.3 Archivo Generado

A pesar del error, el programa logró generar parcialmente un archivo de salida antes de alcanzar el límite de memoria. El archivo contiene cadenas

binarias generadas hasta el valor de $N = 27$. El contenido parcial del archivo generado es:

```
Sigma* = {epsilon 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, 000000, 000001, 000010, 000011, 000100, 000101, 000110, 000111, 001000, 001001, 001010,
001011, 001100, 001101, 001110, 001111, 010000, 010001, 010010, 010011, 010100, 010101, 010110,
010111, 011000, 011001, 011010, 011011, 011100, 011101, 011110, 011111, 100000, 100001, 100010,
100011, 100100, 100101, 100110, 100111, 101000, 101001, 101010, 101011, 101100, 101101, 101110,
101111, 110000, 110001, 110010, 110011, 110100, 110101, 110110, 110111, 111000, 111001, 111010,
111011, 111100, 111101, 111110, 111111, 0000000, 0000001, 0000010, 0000011, 0000100, 0000101,
0000110, 0000111, 0001000, 0001001, 0001010, 0001011, 0001100, 0001101, 0001110, 0001111, 0010000,
0010001, 0010010, 0010011, 0010100, 0010101, 0010110, 0010111, 0011000, 0011001, 0011010, 0011011,
0011100, 0011101, 0011110, 0011111, 0100000, 0100001, 0100010, 0100011, 0100100, 0100101, 0100110,
0100111, 0101000, 0101001, 0101010, 0101011, 0101100, 0101101, 0101110, 0101111, 0110000, 0110001,
0110010, 0110011, 0110100, 0110101, 0110110, 0110111, 0111000, 0111001, 0111010, 0111011, 0111100,
0111101, 0111110, 0111111, 1000000, 1000001, 1000010, 1000011, 1000100, 1000101, 1000110, 1000111,
1001000, 1001001, 1001010, 1001011, 1001100, 1001101, 1001110, 1001111, 1010000, 1010001, 1010010,
1010011, 1010100, 1010101, 1010110, 1010111, 1011000, 1011001, 1011010, 1011011, 1011100, 1011101,
1011110, 1011111, 1100000, 1100001, 1100010, 1100011, 1100100, 1100101, 1100110, 1100111, 1101000,
1101001, 1101010, 1101011, 1101100, 1101101, 1101110, 1101111, 1110000, 1110001, 1110010, 1110011,
1110100, 1110101, 1110110, 1110111, 1111000, 1111001, 1111010, 1111011, 1111100, 1111101, 1111110,
1111111, 00000000, 00000001, 00000010, 00000011, 00000100, 00000101, 00000110, 00000111, 00001000,
00001001, 00001010, 00001011, 00001100, 00001101, 00001110, 00001111, 00010000, 00010001, 00010010,
00010011, 00010100, 00010101, 00010110, 00010111, 00011000, 00011001, 00011010, 00011011, 00011100,
```

Figure 1: Archivo de salida generado por el programa

Las propiedades son las siguientes:

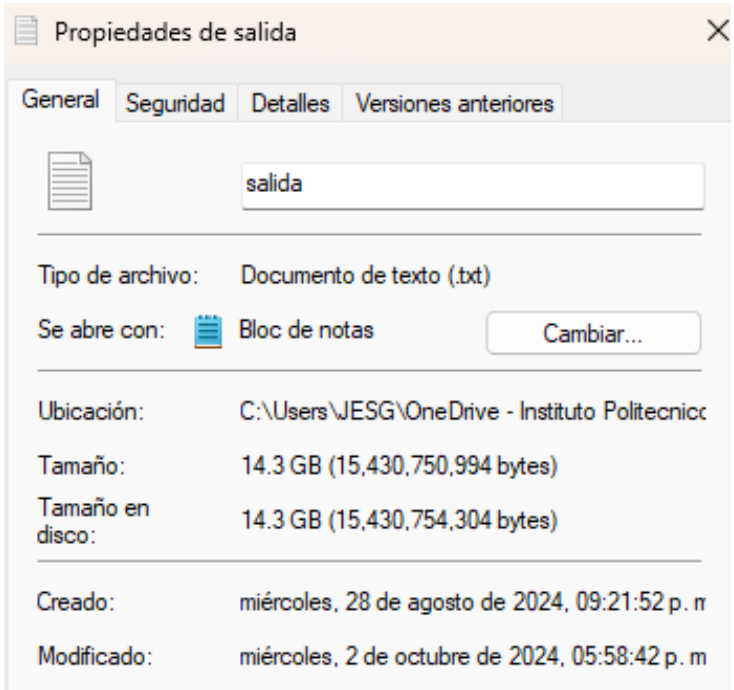

Figure 2: Archivo de salida generado por el programa

3.4 Explicación

El error *OutOfMemoryError* se produjo porque el programa intentó almacenar todas las cadenas binarias generadas para un N grande en memoria. Este almacenamiento tiene implicaciones tanto para la memoria RAM (memoria volátil utilizada durante la ejecución del programa) como para la ROM (memoria no volátil donde se almacena el archivo generado).

Para entender el problema de memoria, debemos considerar el crecimiento exponencial del número de cadenas binarias con respecto a N. Para cada i, el número de cadenas generadas es 2^i y la longitud de cada cadena es i. La cantidad total de memoria necesaria para almacenar estas cadenas se puede calcular usando la siguiente expresión matemática:

$$\text{Memoria total (bytes)} = \sum_{i=0}^{N} i \cdot 2^i$$

Esta expresión representa la suma de los productos entre la longitud i de las cadenas y el número de cadenas 2^i para cada i desde 0 hasta N.

Ejemplo de cálculo para $N = 29$:

$$\sum_{i=0}^{29} i \cdot 2^i$$

Al evaluar esta suma, se obtiene un número extremadamente grande, que representa la cantidad de caracteres que se deben almacenar. Dado que en Java cada carácter se almacena utilizando 2 bytes (debido al uso de codificación UTF-16), la memoria requerida sería el doble de la cantidad calculada por la fórmula. Esta demanda de memoria rápidamente supera la capacidad de la memoria RAM disponible, causando el error.

Además, antes de alcanzar este límite, el programa intenta escribir todas estas cadenas en un archivo, utilizando espacio en la memoria ROM. Aunque la memoria ROM puede almacenar grandes cantidades de datos, el problema sigue siendo la cantidad de datos generados en la RAM antes de ser escritos en el archivo. Esto conduce al fallo cuando se excede la capacidad de la memoria RAM asignada a la Máquina Virtual de Java (JVM).

4 Ejecución para $N = 5$

Dado el error ocurrido para $N = 29$, se muestra una prueba con un valor para N más pequeño, a fin de que se puedan mostrar las gráficas.

4.1 Ejecución del Código

```
1) Elegir N
2) Dejar que la maquina decida
3) Salir
1

Ingresa la N:
5

N = 5

N = 1
N = 2
N = 3
N = 4
N = 5
1) Elegir N
2) Dejar que la maquina decida
3) Salir
```

4.2 Archivo Generado

El archivo generado por el programa para $N = 5$ contiene las cadenas binarias hasta la longitud 5. A continuación se muestra una captura del archivo generado:

```
Sigma* = {epsilon 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}
```

Figure 3: Captura del archivo generado para $N = 5$

4.3 Gráficas

Las gráficas que muestran la cantidad de ceros y unos en las cadenas binarias generadas para $N = 5$ son las siguientes:

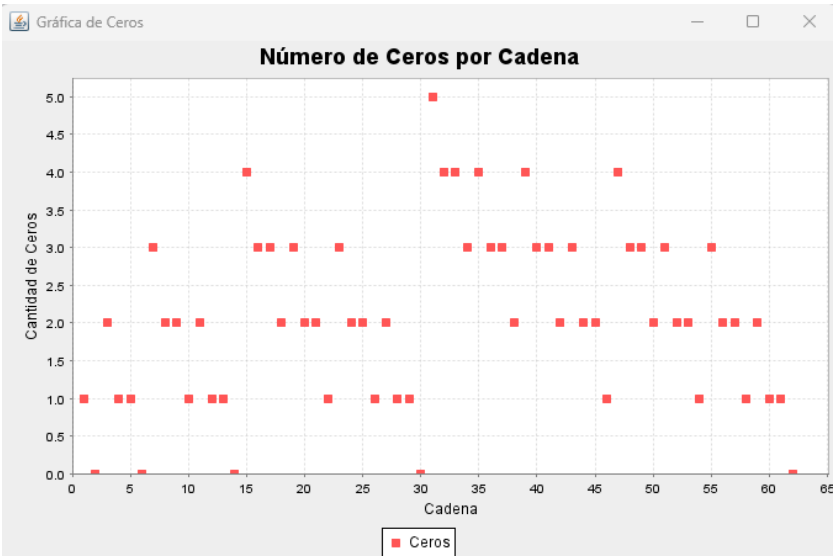

Figure 4: Gráfica de la cantidad de ceros para $N = 5$

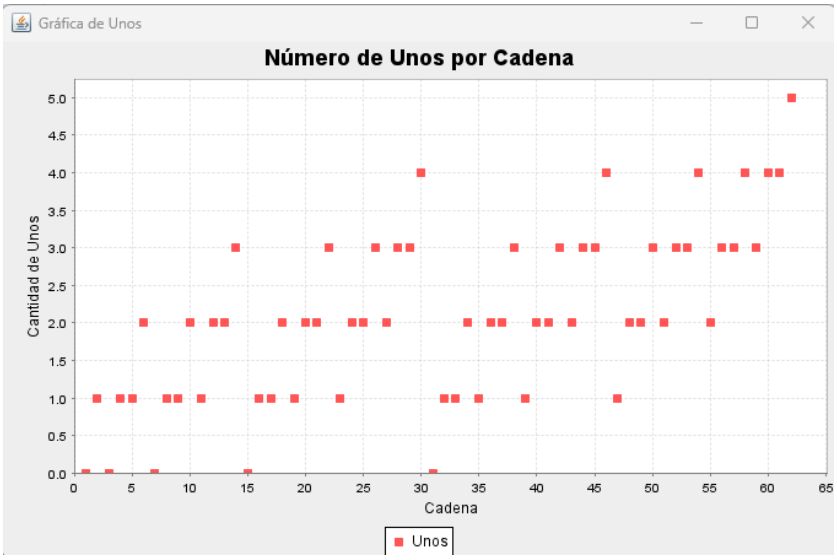

Figure 5: Gráfica de la cantidad de unos para $N = 5$

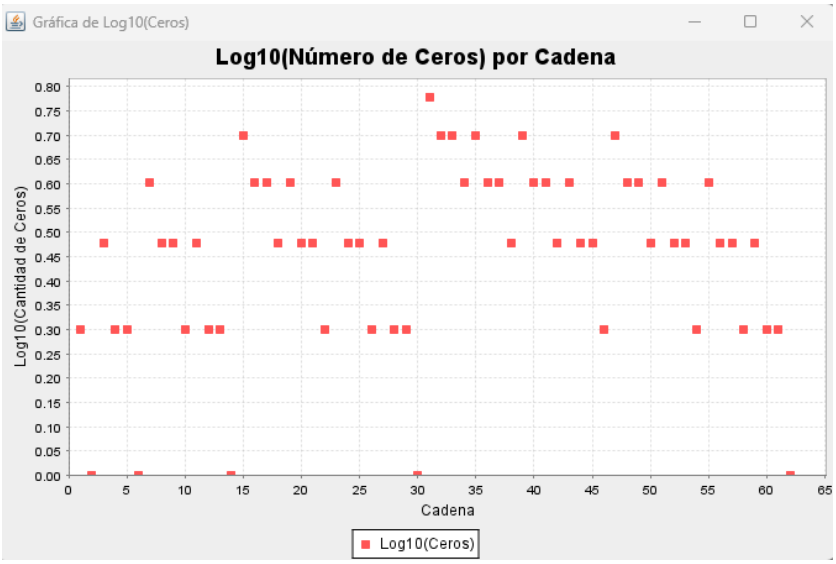

Figure 6: Gráfica logarítmica de la cantidad de ceros$N = 5$

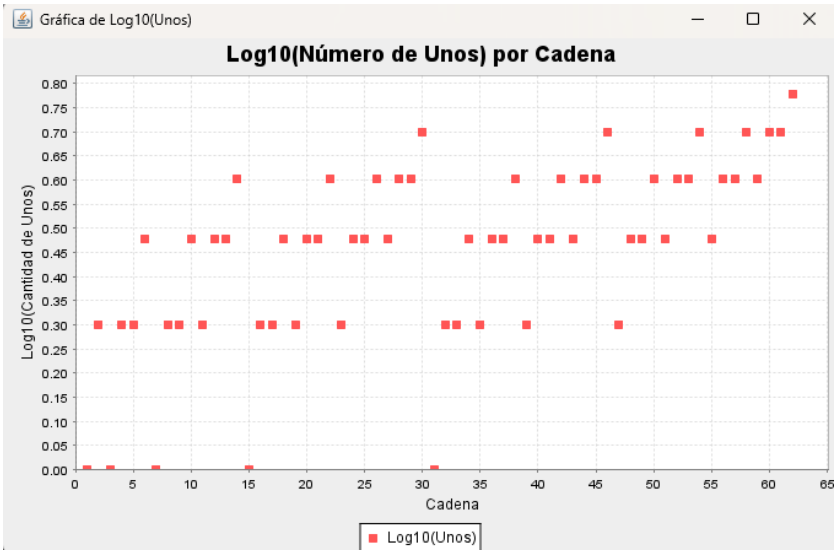

Figure 7: Gráfica logarítmica de la cantidad de unos para $N = 5$

5 Conclusión

En esta práctica, se abordó la generación y análisis de cadenas binarias de longitud N para estudiar los desafíos relacionados con el manejo de grandes cantidades de datos en memoria. La implementación del programa mostró cómo el crecimiento exponencial del número de cadenas binarias afecta significativamente el uso de memoria, lo que llevó a un error de *OutOfMemoryError* al intentar ejecutar el programa para $N = 29$. Esto evidenció la importancia de considerar las limitaciones de la memoria RAM y ROM al desarrollar aplicaciones que manejan grandes volúmenes de datos.

Para $N = 5$, el programa logró generar exitosamente las cadenas binarias y producir las gráficas correspondientes, lo que permitió analizar la cantidad de ceros y unos presentes en las cadenas. Las gráficas obtenidas mostraron cómo los ceros y unos se distribuyen conforme se incrementa la longitud de las cadenas, y se observó una distribución simétrica, algo esperado debido a la naturaleza de las combinaciones binarias.

Además, la expresión para calcular el tamaño en bytes de las cadenas binarias generadas destaca el impacto que tiene el crecimiento exponencial en el uso de memoria. Esta práctica demuestra la importancia de optimizar los algoritmos para el manejo eficiente de datos, especialmente al trabajar con problemas de complejidad exponencial. También resalta la necesidad de limitar el valor de N o utilizar técnicas de procesamiento directo para evitar el consumo excesivo de memoria.

6 Referencias

1. Hopcroft, J. E., Motwani, R., Ullman, J. D. (2006). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

2. Sipser, M. (2006). *Introduction to the Theory of Computation*. Cengage Learning.

3. Cohen, D. I. A. (1997). *Introduction to Computer Theory*. John Wiley & Sons.

4. Linz, P. (2011). *An Introduction to Formal Languages and Automata*. Jones & Bartlett Learning.

5. Jantzen, M. (2016). *Formal Languages and Automata Theory*. CreateSpace Independent Publishing Platform.

6. Oracle. (s.f.). *Java SE Documentation.* [Online] Disponible en: `https://docs.oracle.com/javase/8/docs/technotes/guides/vm/`.