INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO

Fundamentos Matemáticos de la Computación

Proyecto Final

Complejidad de Kolmogorov

“Manual de usuario”

Equipo 5

Integrantes

Susana Muñoz Acosta 165889

Manuel Fernandez Verda 166496

Sebastián Valderrábano Cabrera 163791

Rubén Romero Ortega 174178

13 de mayo de 2020

Introducción

Este proyecto contiene un programa que calcula un aproximado de la complejidad de Kolmogorov de una máquina de Turing. Esto se realiza obteniendo la mejor máquina de Turing que minimice el número de estados necesarios para generar una cinta dada.

Para poder ejecutar dicho programa, es necesario tener un IDE que pueda correr código JAVA, se recomienda NetBeans. El usuario deberá de contar con los siguientes archivos:

1. Códigos elaborados por Angel Fernando Kuri
    - AGF.class
    - AGH.java
    - EGA20.class
    - EGA.java
    - N2bS.class
    - N2bS.java

2. Códigos elaborados por Angel Fernando Kuri y modificados por el equipo
    - RMF.class
    - RMF.java
    - RMH20a.class
    - RMH20a.java

3. Códigos elaborados por el equipo
    - UTM.java
    - UTM.class

Instrucciones:

Se deberá crear un nuevo proyecto de java en el que se encuentren todos los archivos. Una vez hecho lo anterior, el usuario deberá ubicarse en el archivo RMH20a.java y correrlo.

Primero le aparecerá la siguiente pantalla:

```
1) Funcion a optimizar:     33
2) Bits para enteros:       1023
3) Bits para decimales:     0
4) Numero de variables:     1
** Long. del genoma:        1024
5) Numero de iteraciones:   500
6) Minimiza[0]/Maximiza[1]: 0

Modificar (S/N)?
```

Este menú le permitirá al usuario escoger qué problema de RMF se quiere ejecutar. La opción que deseamos y los parámetros que necesitamos son los que se muestran en la imagen superior. Si alguno de los valores es diferente a los que se muestran, estos deberán de actualizarse indicándose que ***SI*** se desea modificar un dato. Se deberá escribir en la terminal cual es el dato a modificar y dar el nuevo valor. Una vez que todos los datos son correctos, se indica que ya ***NO*** se quiere modificar.

```
Modificar (S/N)? N
```

La siguiente ventana le pedirá al usuario ubicar un archivo *.txt* que contenga la cinta a replicar, la cual se quiera leer. Este archivo también deberá estar ubicado en la carpeta del proyecto para poder ser leído.

```
Deme el nombre del archivo de datos que quiere leer:
```

Se deberá indicar el nombre del archivo y continuar.

```
Deme el nombre del archivo de datos que quiere leer:
RESMT1.txt
```

Este archivo deberá contener puros números, preferiblemente en formato binario. Si se encuentra en cualquier otro formato, los datos serán convertidos a su equivalente binario de ASCII. La única limitación será que no contenga caracteres especiales no identificables por ASCII, como los acentos.

Hemos adjuntado varias posibles cintas cuyo formato es el siguiente: RESMT#.txt, donde # va de 1 a 5. Esto le permitirá al usuario hacer varias pruebas. Es importante notar que el contenido de estos archivos no siempre está en binario.

Si el archivo cumple con las especificaciones anteriores, el algoritmo comenzará a correr. Este algoritmo correrá la cantidad de iteraciones o generaciones que el usuario indicó anteriormente. No obstante, nuestro código mostrará solo resultados cuando se obtenga una solución mejor a la que se tenía.

```
BEST new error: 0.7451274362818591     Tamaño: 667.0     Matches: 170.0     Pos Inicial: 4333.0
BEST new error: 0.5787106446776611     Tamaño: 667.0     Matches: 281.0     Pos Inicial: 1.0
BEST new error: 0.25337331334332835     Tamaño: 667.0     Matches: 498.0     Pos Inicial: 0.0
```

BEST new error muestra el porcentaje de desaciertos entre el resultado de nuestra cinta y la cinta deseada. *Tamaño* indica la longitud de bits de la cinta objetivo. *Matches* indica la cantidad de coincidencias y *Pos Inicial* indica la posición en nuestra cinta en la cual se inició la comparación (debido a que los tamaños de las cintas son distintos).

Es importante notar que el algoritmo no es determinístico. Cada vez que el usuario corra el código podrá obtener resultados distintos. Por ejemplo si volvemos a correr la simulación encontramos lo siguiente:

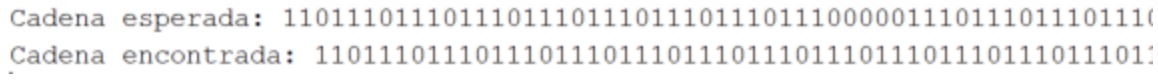
```
BEST new error: 0.25337331334332835     Tamaño: 667.0     Matches: 498.0     Pos Inicial: 0.0
BEST new error: 0.2518740629685158     Tamaño: 667.0     Matches: 499.0     Pos Inicial: 3.0
BEST new error: 0.004497751124437732     Tamaño: 667.0     Matches: 664.0     Pos Inicial: 4323.0
```

Ahora, una vez finalizado el proceso anterior, en la terminal se muestra el error óptimo final (es decir porcentaje de diferencias), la Máquina de Turing, la cantidad de estados en la máquina y la complejidad de Kolmogorov de dicha Máquina de Turing.

```
Óptimo:    .0044977
1000000010000110111100000100001101110100101000011010110001000100
Hay 24 estados en la Maquina de Turing
**************************************
Complejidad de Kolmogorov: 384
**************************************
```

EA	O	M	SE	O	M	SE
0	0	R	34	0	R	56
8	1	L	42	1	R	22
13	1	R	14	0	L	47
14	1	R	17	0	L	29
16	0	R	34	0	L	43
17	1	L	37	1	R	33
18	0	R	57	1	R	13
19	1	L	42	0	R	31
25	0	L	8	1	R	19
30	0	L	48	0	L	H
31	1	L	42	1	L	61
33	1	R	40	1	R	60
34	1	R	18	1	L	2
35	0	L	25	0	L	13
37	0	L	10	0	L	30
40	1	R	35	1	R	43
42	1	L	47	0	L	13
47	0	R	53	0	L	19
48	1	L	6	1	R	34
50	1	L	7	0	R	16
52	1	L	28	0	L	31
53	0	R	35	1	R	52
57	1	R	17	0	L	47
61	0	R	50	0	L	50

Además, podemos observar que el programa imprime dos cintas. La primera contiene la cadena esperada, mientras que la segunda contiene la cadena encontrada.



Como la imagen no se puede ver completa, adjuntamos los resultados reales.

Cadena esperada:
1101110111011101110111011101110111000001110111011101110111011101110111011101110111011101110111011
1011101110111011101110111011101110111011101110111011101110111011101110111011101110111011101110111
0111011101110111011101110111011101110111011101110111011101110111011101110111011101110111011101110
1110111011101110111011101110111011101110111011101110111011101110111011101110111011101110111011101
1101110111011101110111011101110111011101110111011101110111011101110111011101110111011101110111011
1011101110111011101110111011101110111011101110111011101110111011101110111011101110111011101110111
0111011101110111011101110111011101110111011101110111011101110111011101110111011101110111011101110

1110111011101110111011101110111011101110111011101110111011101110111011101110111011101110111011101110
1101110111011

Cadena encontrada:
1101110111011101110111011101110111011101110111011101110111011101110111011101110111011101110111011101110
1011101110111011101110111011101110111011101110111011101110111011101110111011101110111011101110111011101
0111011101110111011101110111011101110111011101110111011101110111011101110111011101110111011101110111011
1110111011101110111011101110111011101110111011101110111011101110111011101110111011101110111011101110111
1101110111011101110111011101110111011101110111011101110111011101110111011101110111011101110111011101110
1011101110111011101110111011101110111011101110111011101110111011101110111011101110111011101110111011101
0111011101110111011101110111011101110111011101110111011101110111011101110111011101110111011101110111011
1110111011101110111011101110111011101110111011101110111011101110111011101110111011101110111011101110111
1101110111011

Estas cadenas nos permiten hacer comparaciones visuales de los resultados e identificar dónde están los bits diferentes.

Podemos notar que, una vez finalizado el proceso, el programa nos pregunta si queremos correr otra función. Si la respuesta es afirmativa, repite el proceso anterior, de lo contrario, cierra el programa.

```
Otra funcion (S/N)?
N

*** FIN DE ESCALADOR ***

BUILD SUCCESSFUL (total time: 4 minutes 55 seconds)
```