Generador Pseudoaleatorio con Máquina de Turing

Institución académica: Universidad Nacional de la Patagonia San Juan Bosco (UNPSJB)

Materia: Fundamentos teóricos de la informática

Docente/s: Leonardo Moreno y Leonardo Morales

Alumno: Brandon Adrian Amarillo Silva

Fecha de entrega: 20/11/2025

ÍNDICE

INTRODUCCIÓN

Máquina de Turing

Una Máquina de Turing es un modelo teórico de computación creado por Alan M. Turing en 1936. Manipula símbolos sobre una cinta infinita según un conjunto finito de reglas predefinidas.

Una máquina de Turing es una quintupla $T= (S, \Sigma, \delta, s_0, F)$ donde:
S es el conjunto de estados, $S = \varnothing$.
Σ es el alfabeto de trabajo.
δ es una función parcial, $\delta : S \times \Sigma \rightarrow S \times \Sigma \times \{I, D, N\}$ donde:
- I denota “movimiento a izquierda”
- D denota “movimiento a derecha”
- N denota “sin movimiento”

s0 es el estado inicial, $s0 \in S$.
F es el conjunto de estados finales, $F \subseteq S$

Objetivo

Implementar una Máquina de Turing que simule el comportamiento de un generador de números pseudoaleatorios del tipo XOR-Shift. El propósito del trabajo es comprender cómo una máquina determinista puede generar secuencias que parecieran ser aleatorias, y estudiar cómo el período (repetición, hasta volver a encontrarse el número disparador) depende de los parámetros y del tamaño de palabra binaria.

DISEÑO DE LA MÁQUINA DE TURING

Descripción del algoritmo XOR-Shift

Un generador **XOR-Shif**t manipula una secuencia de bits aplicando operaciones de desplazamiento y XOR (suma módulo 2). A partir de un número binario inicial (que denominamos semilla binaria inicial) cada iteración genera un nuevo número, que se usa como entrada para la siguiente.
La fórmula general es:

x = x XOR (x << a)
x = x XOR (x >> b)

x = x XOR (x << c)

donde a, b y c son desplazamientos fijos enteros positivos.

Estas transformaciones mezclan los bits del estado de forma que la secuencia resultante aparenta aleatoriedad, aunque es completamente determinista.

Para un estado de **n bits**, el período máximo posible es **2^n − 1**. En este trabajo, como se utiliza un estado de 6 bits

Transformación del algoritmo en una MT

Estados de la Máquina de Turing XOR-Shift

Grupo: INICIALIZACIÓN

INIT

- **Descripción**: Estado inicial de cada iteración. Determina la posición de la última cadena binaria generada y prepara las variables para comenzar una nueva copia.
- **Transición**: → PRE_SHIFT_COPY_MARK

Grupo: COPIA CON MARCADORES

PRE_SHIFT_COPY_MARK

- **Descripción**: Lee un bit de la posición actual, lo guarda temporalmente y lo reemplaza con el símbolo '**#**' (marcador). Este marcador permite rastrear la posición original durante el proceso de copia.
- **Color**: Rojo para el marcador '**#**'
- **Transición**: → PRE_SHIFT_COPY_FIND_TARGET

PRE_SHIFT_COPY_FIND_TARGET

- **Descripción**: Mueve el cabezal paso a paso hacia la derecha hasta llegar a la posición de destino calculada (copyTargetPos + counter), donde se escribirá el bit copiado.
- **Color**: Cyan durante el recorrido
- **Transición**: → PRE_SHIFT_COPY_WRITE_BIT

PRE_SHIFT_COPY_WRITE_BIT

- **Descripción**: Escribe el bit guardado en la posición de destino (debe ser un '▲'). Esta operación efectúa la copia del bit.

- **Color**: Verde para el bit escrito
- **Transición**: → PRE_SHIFT_COPY_RETURN

PRE_SHIFT_COPY_RETURN

- **Descripción**: Retrocede paso a paso hacia la izquierda hasta encontrar el marcador '#' que indica la posición original del bit copiado.
- **Color**: Rosa durante el retroceso
- **Transición**: → PRE_SHIFT_COPY_MARK (siguiente bit) o PRE_SHIFT_COPY_RESTORE (si terminó)

PRE_SHIFT_COPY_RESTORE

- **Descripción**: Verifica que todos los bits hayan sido copiados correctamente. Dependiendo de la fase (phaseStep), determina la siguiente operación: otra copia, un desplazamiento, o continuar con el flujo del algoritmo.
- **Transición**: → PRE_SHIFT_COPY_MARK (si phaseStep=0) o SHIFT_POSITION (si phaseStep=1,2,3)

Grupo: DESPLAZAMIENTO (SHIFT)

SHIFT_POSITION

- **Descripción**: Posiciona el cabezal al inicio (para shift left) o al final (para shift right) de la cadena que será desplazada.
- **Transición**: → SHIFT_START

SHIFT_START

- **Descripción**: Verifica si aún quedan desplazamientos individuales por realizar (shiftCounter < shiftAmount). Si es así, inicia un nuevo desplazamiento; si no, finaliza y prepara el XOR.
- **Transición**: → SHIFT_READ (si quedan shifts) o XOR_MARK (si terminó)

SHIFT_READ

- **Descripción**: Lee el bit adyacente que se va a mover a la posición actual. Para shift left lee el bit de la derecha (counter+1); para shift right lee el bit de la izquierda (counter-1).
- **Color**: Amarillo para el bit leído
- **Transición**: → SHIFT_WRITE (si hay bit) o SHIFT_FILL_ZERO (si llegó al límite)

SHIFT_WRITE

- **Descripción**: Escribe (pisa) el bit leído en la posición actual, efectuando el desplazamiento de un bit.
- **Color**: Naranja para el bit pisado
- **Transición**: → SHIFT_READ (siguiente bit)

SHIFT_FILL_ZERO

- **Descripción**: Rellena con '0' la posición que quedó vacía después del desplazamiento (al final para shift left, al inicio para shift right).
- **Color**: Cyan para el cero agregado
- **Transición**: → SHIFT_START (para repetir si shiftAmount > 1)

Grupo: XOR (OPERACIÓN LÓGICA)

XOR_MARK

- **Descripción**: Lee el bit actual de la cadena desplazada, lo guarda y lo reemplaza con '#'. Este marcador permite compararlo con el bit correspondiente de la cadena original.
- **Color**: Rojo para el marcador '#'
- **Transición**: → XOR_FIND_BLANK (si hay más bits) o finishXOR() (si terminó)

XOR_FIND_BLANK

- **Descripción**: Mueve el cabezal hacia la derecha hasta encontrar el símbolo '▲' que separa la cadena desplazada de la cadena original.
- **Color**: Cyan para los '▲' encontrados
- **Transición**: → XOR_FIND_COMPARE

XOR_FIND_COMPARE

- **Descripción**: Continúa moviendo el cabezal hacia la derecha hasta encontrar el bit correspondiente de la cadena original con la cual se va a comparar.
- **Color**: Amarillo para el bit encontrado
- **Transición**: → XOR_COMPARE_BIT

XOR_COMPARE_BIT

- **Descripción**: Reemplaza el bit de comparación con '▲', marcando que ya fue procesado. Guarda el bit para la operación XOR.
- **Color**: Rosa para el '▲' escrito
- **Transición**: → XOR_RETURN

XOR_RETURN

- **Descripción**: Retrocede paso a paso hacia la izquierda hasta encontrar el marcador '#'. Al encontrarlo, calcula el resultado XOR (0 si los bits son iguales, 1 si son diferentes) y escribe el resultado en esa posición.
- **Color**: Rosa durante retroceso, Verde para el resultado
- **Transición**: → XOR_MARK (siguiente bit)

Grupo: COMPARACIÓN

COMPARE_INIT

- **Descripción**: Compara el valor generado en la iteración actual con todos los valores previamente generados (almacenados en generatedValues). Si encuentra una coincidencia exacta, detecta un ciclo; si no, agrega el nuevo valor y prepara la siguiente iteración.
- **Color**: Magenta para resaltar comparación
- **Transición**: → CYCLE_DETECTED (si hay repetición) o PRE_SHIFT_COPY_MARK (nueva iteración)

Grupo: FINALIZACIÓN

CYCLE_DETECTED

- **Descripción**: Estado alcanzado cuando se detecta que un valor generado ya existía previamente. Registra el período del ciclo y prepara la máquina para detenerse.
- **Transición**: → HALT

HALT

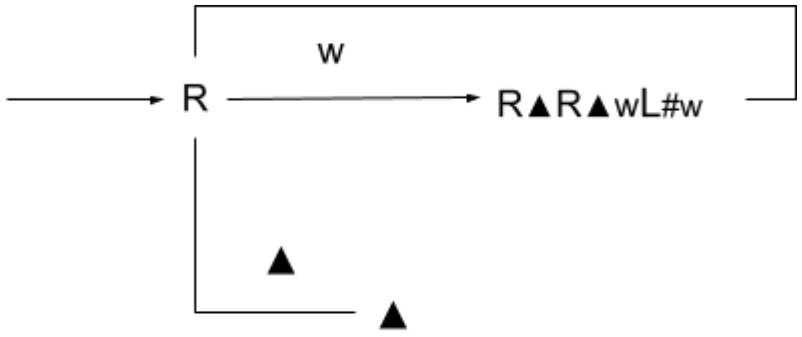
- **Descripción**: Estado final de la máquina. No realiza ninguna operación y el método step() retorna false, indicando que la ejecución ha terminado.
- **Transición**: Ninguna (estado terminal)

Resumen por Grupo

Grupo	Estados	Función Principal
Inicialización	INIT	Preparar nueva iteración
Copia	RE_SHIFT_COPY_MARK, PRE_SHIFT_COPY_FIND_TARGET, PRE_SHIFT_COPY_WRITE_BIT, PRE_SHIFT_COPY_RETURN, PRE_SHIFT_COPY_RESTORE	Copiar cadenas binarias bit a bit usando marcadores
Des. Izquierda	SHIFT_POSITION, SHIFT_START, SHIFT_READ, SHIFT_WRITE, SHIFT_FILL_ZERO (con isShiftLeft=true)	Implementar operación x << a o x << c
Des. Derecha	SHIFT_POSITION, SHIFT_START, SHIFT_READ, SHIFT_WRITE, SHIFT_FILL_ZERO (con isShiftLeft=false)	Implementar operación x >> b
XOR	XOR_MARK, XOR_FIND_BLANK, XOR_FIND_COMPARE, XOR_COMPARE_BIT, XOR_RETURN	Realizar operación XOR bit a bit
Comparación	COMPARE_INIT	Detectar ciclos comparando valores
Finalización	CYCLE_DETECTED, HALT	Terminar ejecución

Diagrama de estados

MT de copiar



▲110010▲[1]10010▲▲▲▲▲▲▲▲

▲110010▲#[1]0010▲▲▲▲▲▲▲▲

▲110010▲#1[0]010▲▲▲▲▲▲▲▲

▲110010▲#10[0]10▲▲▲▲▲▲▲▲

▲110010▲#100[1]0▲▲▲▲▲▲▲▲

▲110010▲#1001[0]▲▲▲▲▲▲▲▲

▲110010▲#10010[▲]▲▲▲▲▲▲▲

▲110010▲#10010▲[1]▲▲▲▲▲▲

▲110010▲#10010[▲]1▲▲▲▲▲▲

▲110010▲#1001[0]▲1▲▲▲▲▲▲

▲110010▲#100[1]0▲1▲▲▲▲▲▲

▲110010▲#10[0]10▲1▲▲▲▲▲▲

▲110010▲#1[0]010▲1▲▲▲▲▲▲

▲110010▲#[1]0010▲1▲▲▲▲▲▲

▲110010▲[1]10010▲1▲▲▲▲▲▲

▲110010▲1[#]0010▲1▲▲▲▲▲▲

▲110010▲1#[0]010▲1▲▲▲▲▲▲

▲110010▲1#0[0]10▲1▲▲▲▲▲▲

▲110010▲1#00[1]0▲1▲▲▲▲▲▲

▲110010▲1#001[0]▲1▲▲▲▲▲▲

▲110010▲1#0010[▲]1▲▲▲▲▲▲

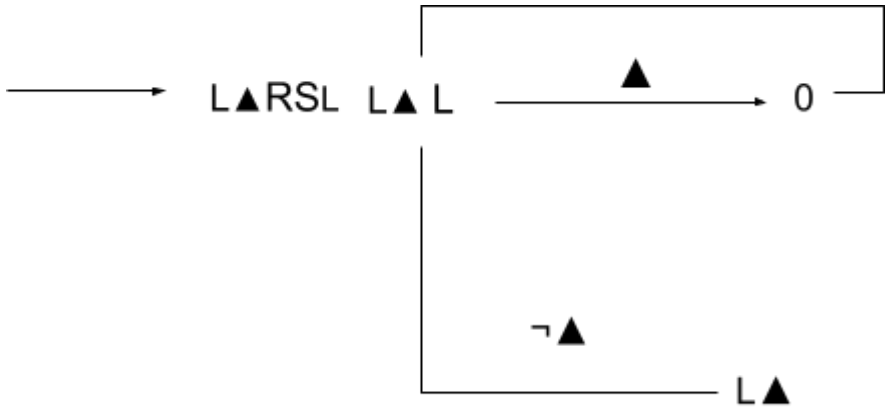
▲110010▲1#0010▲[1]▲▲▲▲▲▲

▲110010▲1#0010▲1[1]▲▲▲▲▲ y así hasta que termine de hacer la copia

▲110010▲110010▲110010▲

La tercera copia se realiza ya que con ese realizare el XOR y el binario del medio es el que guardará las operaciones Shfit y XOR

MT para Shift-Left



▲110010[▲]110010▲110010▲

▲110010▲[1]10010▲110010▲

▲110010▲1[▲]0010▲110010▲

▲110010▲10[▲]010▲110010▲

▲110010▲100[▲]10▲110010▲

▲110010▲1001[▲]0▲110010▲

▲110010▲10010[▲]▲110010▲

▲110010▲10010▲[▲]110010▲

▲110010▲10010[▲]▲110010▲

▲110010▲10010[0]▲110010▲

▲110010▲1001[0]0▲110010▲

▲110010▲100[1]00▲110010▲

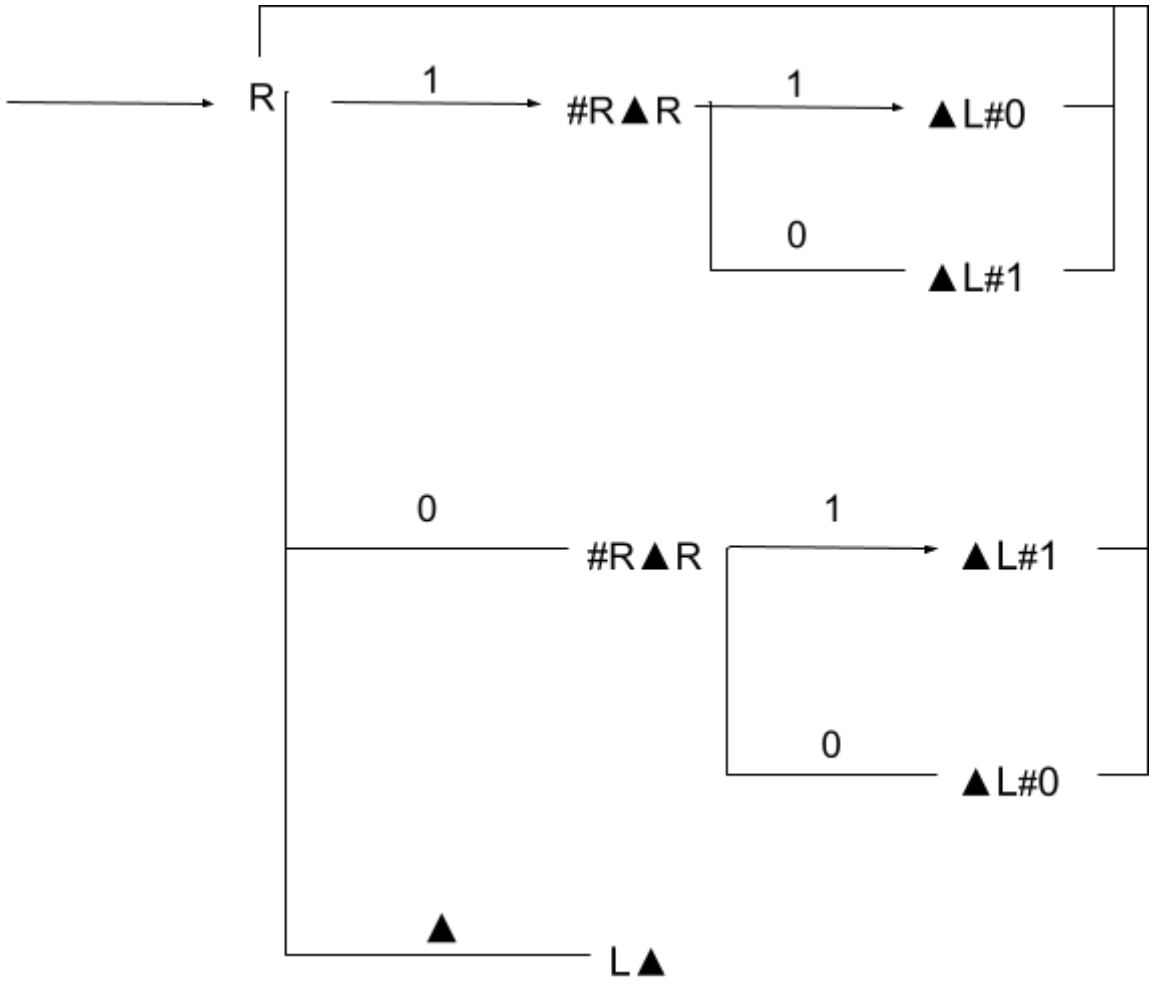
▲110010▲10[0]100▲110010▲

▲110010▲1[0]0100▲110010▲

▲110010▲[1]00100▲110010▲

▲110010[▲]100100▲110010▲

MT para XOR



▲110010▲10010[0]▲110100▲

▲110010▲10010[0]▲110100▲

▲110010▲10010[#]▲110100▲

▲110010▲10010#[▲]110100▲

▲110010▲10010#▲[1]10100▲

▲110010▲10010#▲1[1]0100▲

▲110010▲10010#▲110100[▲]

▲110010▲10010#▲11010[0]▲

▲110010▲10010#▲11010[▲]▲

▲110010▲10010#▲1101[0]▲▲

▲110010▲10010#▲110[1]0▲▲

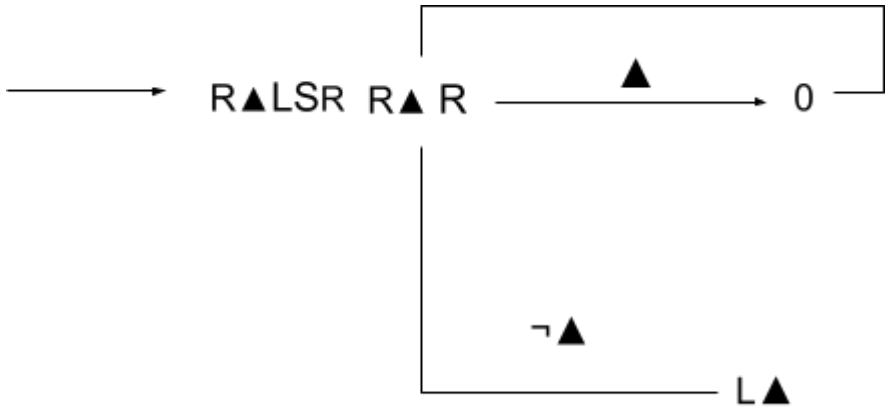
▲110010▲10010[0]▲11010▲▲

▲110010▲1001[0]0▲11010▲▲

▲110010▲1001[#]0▲11010▲▲

▲110010▲010110▲▲▲▲▲▲▲▲▲

MT para Shift-Right



▲110010[▲]010110▲010110▲

▲110010▲[0]10110▲010110▲

▲110010▲0[1]0110▲010110▲

▲110010▲01[0]110▲010110▲

▲110010▲010[1]10▲010110▲

▲110010▲0101[1]0▲010110▲

▲110010▲01011[0]▲010110▲

▲110010▲010110[▲]010110▲

▲110010▲01011[0]▲010110▲

▲110010▲0101▲[1]▲010110▲

▲110010▲0101[▲]1▲010110▲

▲110010▲010[▲]11▲010110▲

▲110010▲01[▲]011▲010110▲

▲110010▲0[▲]1011▲010110▲

▲110010▲[▲]01011▲010110▲

▲110010[▲]▲01011▲010110▲

▲110010▲[▲]01011▲010110▲

▲110010▲[0]01011▲010110▲

▲110010[▲]001011▲010110▲

MT para Comparar

Flujo general de operación

Inicialización
↓
Copia (inicial)
↓
Copia (pre-shift A)
↓
Desplazamiento Izquierda (x << a)
↓
XOR (x XOR resultado shift A)
↓
Copia (pre-shift B)
↓
Desplazamiento Derecha (x >> b)
↓
XOR (x XOR resultado shift B)
↓
Copia (pre-shift C)
↓
Desplazamiento Izquierda (x << c)
↓
XOR (x XOR resultado shift C)
↓
Comparación
↓
¿Repetición? → Sí: Finalización
→ No: Nueva iteración (volver a Copia inicial)

Estados del sistema y justificación del diseño

El diseño final de la máquina se compone de 22 estados internos que representan las micro-operaciones necesarias para simular paso a paso el comportamiento de los operadores XOR, desplazamiento, copia y comparación.

Debido a que una máquina de Turing sólo puede operar leyendo y escribiendo un símbolo por vez, cada operación de mayor nivel (como **`x = x XOR (x << a)`**) debe descomponerse en múltiples sub-estados.

Cada transición, cada movimiento del cabezal y cada operación sobre la cinta se modela explícitamente.

Para claridad expositiva, los 22 estados se agrupan en 5 diagramas principales: copia, desplazamientos, XOR, comparación y término. Cada grupo representa un módulo lógico del algoritmo XOR-Shift implementado como una MT clásica.

METODOLOGÍA

Proceso de Desarrollo de la Máquina de Turing

La implementación de la Máquina de Turing para el generador XOR-Shift se desarrolló siguiendo una metodología incremental basada en la comprensión profunda del algoritmo antes de su codificación:

1. Análisis Manual del Algoritmo

Como primer paso, se realizó una ejecución manual completa del algoritmo XOR-Shift utilizando el ejemplo proporcionado (semilla **`110010`** con parámetros **`a=1, b=2, c=1`**). Este proceso permitió:

- Comprender el funcionamiento de cada operación individual (desplazamientos y XOR)
- Identificar los estados intermedios de la cinta
- Detectar las transiciones necesarias entre estados
- Visualizar cómo el cabezal debe moverse celda por celda

análisis manual

Semilla 110010 con parámetros a = 1, b=2, c=1
▲110010▲110010▲copia **110010(x0)**
▲110010▲100100▲ a << 1 y se guarda **110010**
▲110010▲010110▲ 110010 XOR (100100)
▲110010▲000101▲b >> 2 y se guarda 010110
▲110010▲010011▲ 010110 XOR (000101)
▲110010▲100110▲c << 1 y se guarda 010011
▲110010▲110101▲ 010011 XOR (100110)
▲**110010**▲**110101**▲ Compara, no son iguales

▲110010▲110101▲110101▲copia **110101(x1)**
▲110010▲110101▲101010▲ a << 1 y se guarda **110101**
▲110010▲110101▲011111▲ 110101 XOR (101010)

▲110010▲110101▲000111▲ b >>2 y se guarda 011111
▲110010▲110101▲011000▲ 011111 XOR (000111)
▲110010▲110101▲110000▲ c << 1 y se guarda 011000
▲110010▲110101▲101000▲ 011000 XOR (110000)
▲110010▲**110101**▲**101000**▲ Compara, no son iguales
▲**110010**▲110101▲**101000**▲ Compara, no son iguales

▲110010▲110101▲101000▲101000▲ copia **101000(x2)**
▲110010▲110101▲101000▲010000▲ a << 1 y se guarda **101000**
▲110010▲110101▲101000▲111000▲ 101000 XOR (010000)
▲110010▲110101▲101000▲001110▲ b >> 2 y se guarda 111000
▲110010▲110101▲101000▲110110▲ 111000 XOR (001110)
▲110010▲110101▲101000▲101100▲ c<< 1 y se guarda 110110
▲110010▲110101▲101000▲010110▲ 110110 XOR (101100)
▲110010▲110101▲**101000**▲**010110**▲ Comparar, no son iguales
▲110010▲**110101**▲101000▲**010110**▲ Comparar, no son iguales
▲**110010**▲110101▲101000▲**010110**▲ Comparar, no son iguales

2. Diseño de Diagramas de Estados

A partir del análisis manual, se diseñaron diagramas de transición de estados para cada operación fundamental:

- **Diagrama de Copia con Marcadores**: Estados para leer, marcar, avanzar, escribir y retroceder
- **Diagrama de Desplazamiento**: Estados para pisar bits secuencialmente y rellenar con ceros
- **Diagrama de XOR**: Estados para marcar, buscar, comparar y escribir resultados
- **Diagrama de Comparación**: Estados para verificar repetición de valores

Estos diagramas permiten identificar el nivel de granularidad necesario para que la implementación se comportara como una Máquina de Turing real, donde cada movimiento del cabezal y cada lectura/escritura constituye un paso discreto.

3. Refinamiento del Nivel de Granularidad

Durante el diseño, se identificó la necesidad de descomponer operaciones aparentemente simples en múltiples estados para lograr un comportamiento fiel a una MT:

- **Operación de Copia**: Inicialmente se consideró como un único estado, pero se descompuso en 5 estados (MARK, FIND_TARGET, WRITE_BIT, RETURN, RESTORE) para mostrar el movimiento paso a paso del cabezal.
- **Operación de Shift**: Se diseñó para pisar bit por bit la posición actual en lugar de crear una nueva sección desplazada, logrando así un comportamiento más realista.
- **Operación de XOR**: Se implementó con marcadores temporales (# y ▲) para permitir la comparación bit a bit sin requerir memoria externa a la cinta.

4. Implementación y Optimización

La implementación se realizó en Java, estructurando el código en:

- **Clase `TuringTape`**: Gestión de la cinta y movimiento del cabezal
- **Clase `TapeCell`**: Representa la celda de la cinta
- **Enum `State`**: Representa los estados de la máquina
- **Clase `MachineTuring`**: Lógica de estados y transiciones
- **Clase `TuringGUI`**: Interfaz gráfica para visualización

Durante la implementación, se aplicaron las siguientes optimizaciones:

- **Reutilización de estados genéricos**: Los estados de copia (PRE_SHIFT_COPY_*) se reutilizan para todas las operaciones de copia, reduciendo redundancia
- **Control mediante variables de fase** (**`phaseStep`**): Permite que los mismos estados se comporten de manera diferente según el contexto
- **Unificación de operaciones de desplazamiento**: Un único conjunto de estados (SHIFT_*) maneja tanto desplazamientos a izquierda como a derecha mediante el flag **`isShiftLeft`**

5. Validación y Pruebas

La validación se realizó mediante:

- Ejecución paso a paso con visualización gráfica
- Verificación de ciclos para los parámetros recomendados
- Comparación de resultados con implementaciones de referencia
- Análisis del contador de pasos para verificar la complejidad del algoritmo

---

Decisiones de Diseño Clave

1. **Uso de marcadores temporales** (#, ▲): Permite realizar operaciones complejas sin memoria adicional, manteniendo toda la información en la cinta como requiere el modelo de MT.
2. **Separación en tres secciones de cinta**: Durante cada iteración se mantienen tres valores (original, intermedio, comparación) que permiten realizar las operaciones XOR correctamente.
3. **Visualización con colores**: Cada tipo de operación tiene un color asociado (rojo para marcadores, amarillo para lectura, verde para escritura, etc.) facilitando la comprensión del proceso.
4. **Control de velocidad variable**: Permite tanto demostraciones didácticas lentas como ejecuciones rápidas para alcanzar ciclos largos.

CASOS DE PRUEBA

Tabla de parámetros usados para la semilla 110010

Caso	Semilla	a	b	c	Periodo esperado	Periodo obtenido
1	110010	1	2	1	7	7
2	110010	1	3	2	63	63

Casos

Caso 1: Semilla 110010 con parámetros (1, 2, 1)

1. Copy:
    a. copy: ▲110010▲**110010**▲
2. Shift-Left x << 1:
    a. copy: ▲110010▲110010▲**110010**▲
    b. shift: ▲110010▲**100100**▲110010▲
    c. xor: ▲110010▲**010110**▲▲▲▲▲▲▲▲
3. Shift-Right x >> 2:
    a. copy: ▲110010▲010110▲**010110**▲
    b. shift: ▲110010▲**000101**▲010110▲
    c. xor: ▲110010▲**010011**▲▲▲▲▲▲▲▲
4. Shift-Left x << 1:
    a. copy: ▲110010▲010011▲**010011**▲
    b. shift: ▲110010▲**100110**▲010011▲
    c. xor: ▲110010▲**110101**▲▲▲▲▲▲▲▲
5. Compare:
    a. compare: ▲**110010**▲**110101**▲▲▲▲▲▲▲▲ ¿110101 es igual a un valor anterior? No
    b. Se generará un nuevo ciclo

Tabla de valores generados

Paso	x_n generado	¿Se repite?
0	110010	—
1	110101	No
2	101000	No

3	011010	No
4	101111	No
5	000111	No
6	011101	No
7	110010	Si, con x_0

Caso 2: Semilla 110010 con parámetros (1, 3, 2)

1. copy:
    a. copy: ▲110010▲**110010**▲
2. Shift-Left x << 1:
    a. copy: ▲110010▲110010▲**110010**▲
    b. shift: ▲110010▲**100100**▲110010▲
    c. xor: ▲110010▲**010110**▲▲▲▲▲▲▲▲
6. Shift-Right x >> 3:
    a. copy: ▲110010▲010110▲**010110**▲
    b. shift: ▲110010▲**000010**▲010110▲
    c. xor: ▲110010▲**010101**▲▲▲▲▲▲▲▲
7. Shift-Left x << 2:
    a. copy: ▲110010▲010101▲**010101**▲
    b. shift: ▲110010▲**010100**▲010011▲
    c. xor: ▲110010▲**000111**▲▲▲▲▲▲▲▲
8. Compare:
    a. compare: ▲**110010**▲**000111**▲▲▲▲▲▲▲▲ ¿000111 es igual a un valor anterior? No
    b. Se generará un nuevo ciclo

Esto genera 63 ciclos en donde termina con **110010** que se repite.

Variación del período según los parámetros (a, b, c)

El algoritmo XOR-Shift utiliza una combinación de desplazamientos y operaciones XOR para generar el siguiente valor pseudoaleatorio a partir de una semilla inicial. Para una semilla de **n bits**, el período máximo posible es:

$$2^n - 1$$

En este trabajo, n = 6, por lo que el período máximo teórico es:

$$2^6 - 1 = 63$$

Sin embargo, **no todas las combinaciones de parámetros (a, b, c)** logran alcanzar este período.

El comportamiento del período depende directamente de **cómo interactúan los desplazamientos con la estructura binaria del número**.

Cada parámetro controla un desplazamiento particular:

- **a** → shift-left
- **b** → shift-right
- **c** → shift-left

Al variar estos valores:

- cambia qué bits se mezclan entre sí,
- cambia qué patrones son destruidos o preservados,
- y por lo tanto cambia el **grado de difusión** de la operación XOR.

Cuando los parámetros logran mezclar correctamente todos los bits, la secuencia recorre **todos los 63 valores posibles** antes de volver a la semilla: esto se conoce como **período máximo**.

Máquina determinista y pseudoaleatoriedad

A pesar de que la Máquina de Turing utilizada en este trabajo es completamente **determinista**, es capaz de generar secuencias que **aparentan ser aleatorias**.

Sensibilidad a la semilla

Un solo bit distinto en la semilla inicial produce una secuencia completamente distinta.

Difusión de bits mediante XOR y desplazamientos

Cada operación del XOR-Shift:

- mezcla bits de posiciones distintas,
- destruye patrones internos de la semilla,
- redistribuye información dentro de los 6 bits

Naturaleza determinista del proceso

Es importante destacar que la secuencia generada **no es realmente aleatoria**:

- si se repite la misma semilla,
- con los mismos parámetros (a, b, c),
- se obtendrá **exactamente la misma secuencia**.

El ciclo revela la naturaleza pseudoaleatoria

Aunque los valores parecen aleatorios, el algoritmo está limitado a un espacio finito de $2^6=64$ combinaciones posibles.
Por lo tanto, el sistema debe:

- recorrer una parte de ese espacio,
- volver a un valor ya generado,
- caer en un **ciclo**,
- y detenerse.

Conclusiones

En este trabajo se logró implementar una Máquina de Turing capaz de simular el comportamiento del generador pseudoaleatorio XOR-Shift en su forma más clásica con un nivel de granularidad para cada operación.

La simulación mostró claramente:

- las operaciones de copia,
- los desplazamientos lógicos,
- las comparaciones,
- la mezcla de bits con XOR,
- y la detección del ciclo.

Se observa que algunos ciclos producen periodos correo y otras un máximo de 63.

Futuras implementaciones

- Corrección en el comportamiento Shift para que tenga un comportamiento adecuado a una MT.
- Corrección en el comportamiento del comparador. Hay que darle un nivel de granularidad para que tenga el comportamiento esperado de una MT.
- Ajustar el llenado de celdas, ya que hay problema si la velocidad es muy alta, el puntero espera para agregar un valor cuando se llega al final, se pierden valores que rompen la lógica.

Referencias

GitHub de RickyTB
https://github.com/RickyTB/maquina-turing/tree/master
PROGRAMA EN JAVA QUE SIMULA UNA MAQUINA DE TURING COMPUTABLE
https://josemariavalenciaramirezmt.blogspot.com/2011/09/programa-en-java-que-simula-una-maquina.html

Libro - Fundamentos de ciencia de la computación - Juan Carlos Augusto
https://campusvirtual.unp.edu.ar/pluginfile.php/39901/mod_resource/content/0/Fundamentos_Juan_Augusto.pdf

Libro - Teoria de la Computación - J.Glenn Brookshear
https://campusvirtual.unp.edu.ar/pluginfile.php/634905/mod_resource/content/1/Teoria_de_la_Computacion_J._Glenn_Brookshear_compressed.pdf

Wikipedia- Xorshift
https://en.wikipedia.org/wiki/Xorshift

Pseudo random number using xorshift algorithm
https://www.educative.io/answers/pseudo-random-number-using-xorshift-algorithm