TEORIAS DE INVESTIGACION

TEORIA DE AUTOMATAS:

La **teoría de autómatas** es una rama de la [teoría de la computación](https://es.wikipedia.org/wiki/Teor%C3%ADa_de_la_computaci%C3%B3n) que estudia las máquinas abstractas y los problemas que éstas son capaces de resolver. La teoría de autómatas está estrechamente relacionada con la teoría del [lenguaje formal](https://es.wikipedia.org/wiki/Lenguaje_formal) ya que los autómatas son clasificados a menudo por la clase de lenguajes formales que son capaces de reconocer. También son de gran utilidad en la [teoría de la complejidad computacional](https://es.wikipedia.org/wiki/Teor%C3%ADa_de_la_complejidad_computacional).

Ejemplo y explicaciones



Un autómata es un modelo matemático para una [máquina de estado finito](https://es.wikipedia.org/wiki/Aut%C3%B3mata_finito) (FSM sus siglas en inglés). Una FSM es una máquina que, dada una entrada de símbolos, "salta" a través de una serie de estados de acuerdo a una función de transición (que puede ser expresada como una tabla). En la variedad común "Mealy" de FSMs, esta función de transición dice al autómata a qué estado cambiar dados unos determinados estado y símbolo

**Gramáticas formales**

**Gramáticas regulares**: concepto y ejemplos de su uso en lenguajes de programación. En [informática](https://es.wikipedia.org/wiki/Inform%C3%A1tica) una **gramática regular** es una [gramática formal](https://es.wikipedia.org/wiki/Gram%C3%A1tica_formal) (V, Σ, R, S) que puede ser clasificada como regular izquierda o regular derecha. Las gramáticas regulares sólo pueden generar a los [lenguajes regulares](https://es.wikipedia.org/wiki/Lenguaje_regular) de manera similar a los autómatas finitos y las expresiones regulares.

Dos gramáticas regulares que generan el mismo lenguaje regular se denominan equivalentes. Toda gramática regular es una [gramática libre de contexto](https://es.wikipedia.org/wiki/Gram%C3%A1tica_libre_de_contexto).

Una **gramática regular derecha** es aquella cuyas reglas de producción P son de la siguiente forma:

1. *A* → *a*, donde *A* es un símbolo no-terminal en *N* y *a* uno terminal en Σ
2. *A* → *aB*, donde *A* y *B* pertenecen a *N* y *a* pertenece a Σ
3. *A* → ε, donde *A* pertenece a *N*.
* **Análisis léxico**
* **Tokenización**: La tokenización es un concepto que cada vez cobra más relevancia en el mundo de la tecnología, la economía y la sociedad. Se trata de un **proceso que permite transformar cualquier tipo de activo o dato en una unidad digital llamada token, que puede ser almacenada, transferida y gestionada de forma segura y eficiente**..
* **Expresiones regulares**: Las expresiones regulares **son un equivalente algebraico para un autómata**. Utilizado en muchos lugares como un lenguaje para describir patrones en texto que son sencillos pero muy útiles. Pueden definir exactamente los mismos lenguajes que los autómatas pueden describir: Lenguajes regulares.
* **Reconocimiento de palabras clave** Las keywords de lenguaje, también conocidas como palabras clave o palabras reservadas del lenguaje, son **términos predefinidos que tienen un significado especial y una funcionalidad específica en un lenguaje de programación**.
* **Identificación de tokens**: Se pueden utilizar para diversos fines, como representar una criptomoneda, un valor o una materia prima, y ​​se pueden clasificar en diferentes tipos, incluidos **tokens de utilidad, tokens de valor, tokens de gobernanza y tokens no fungibles**.

**Ejemplo:** Se llama token (en inglés, 'ficha', como por ejemplo las que se utilizan en las máquinas recreativas o los coches de choque) a una **unidad de valor basada en criptografía y emitida por una entidad privada en una 'blockchain', como Bitcoin o Ethereum**

**Análisis sintáctico**

* **Tipos de análisis sintáctico**:
	+ Análisis sintáctico descendente: Es la **fase del analizador que se encarga de chequear el texto de entrada en base a una gramática dada**. Y en caso de que el programa de entrada sea válido, suministra el árbol sintáctico que lo reconoce.
	+ Análisis sintáctico ascendente. Es la **fase del analizador que se encarga de chequear el texto de entrada en base a una gramática dada**. Y en caso de que el programa de entrada sea válido, suministra el árbol sintáctico que lo reconoce.
* **Árboles de análisis sintáctico**: Los árboles de análisis son útiles en varias aplicaciones, incluido el procesamiento del lenguaje, el diseño del compilador y el análisis de sintaxis. **Proporcionan una representación estructural de la cadena de entrada, lo que permite el análisis y la manipulación de sus propiedades sintácticas**.

**Algoritmos de análisis de cadenas**

* **Algoritmo de Boyer-Moore**: El **algoritmo de búsqueda de cadenas Boyer-Moore** es un particularmente eficiente [algoritmo de búsqueda de cadenas](https://es.wikipedia.org/wiki/Algoritmos_de_b%C3%BAsqueda_de_subcadenas), y ha sido el punto de referencia estándar para la literatura de búsqueda de cadenas práctica.
* El algoritmo Boyer-Moore **utiliza información recopilada durante el paso de preprocesamiento para omitir secciones del texto** , lo que da como resultado un factor constante más bajo que muchos otros algoritmos de búsqueda de cadenas. En general, el algoritmo se ejecuta más rápido a medida que aumenta la longitud del patrón

**Compiladores e intérpretes**

* **Estructura de un compilador**: El funcionamiento de un compilador está dividido fundamentalmente en dos partes: **Una de análisis del código fuente y otra de síntesis del código objeto**, para lo cual, al igual que el ensamblador, debe construir y analizar varias tablas (ver Figura 1).

. **Fases del compilador**: Análisis léxico.

* + Análisis sintáctico. El análisis sintáctico es una parte importante en el contexto de la perspectiva de diseño de autómatas y compiladores. El análisis sintáctico se refiere al **proceso de analizar una cadena de símbolos, ya sea en lenguaje natural o en lenguaje de programación, de acuerdo con las reglas de una gramática formal**.
	+ Generación de código. La **generación de código** AI implica el uso de herramientas de software avanzadas impulsadas por inteligencia artificial (IA), procesamiento de lenguaje natural (PNL) y machine learning (ML) para escribir **código** de computadora.
	+ Optimización de código. La optimización de código es el **conjunto de fases de un compilador que transforman un fragmento de código en otro fragmento con un comportamiento equivalente y que se ejecuta de forma más eficiente**, es decir, usando menos recursos de cálculo como memoria o tiempo de ejecución
	+ **Lenguajes de programación utilizados en la creación de compiladores**: como **Lex y Yacc**. Instituto Internacional de Tecnología de la Información (I²IT) **Lex es un analizador léxico y constituye la primera fase del diseño de un compilador.** **YACC es un generador de analizadores sintácticos que toma un archivo de entrada con una especificación de gramática BNF enriquecida con atributos** .

**Aplicaciones prácticas de autómatas**

* **Reconocimiento de patrones**: El **reconocimiento de patrones** es una de las técnicas de inteligencia artificial que permite a las máquinas identificar y clasificar **patrones** en los datos, imitando la misma capacidad humana de reconocer y procesar información.
* **Procesamiento de lenguaje natural**: l procesamiento del lenguaje natural **es una disciplina dentro de la inteligencia artificial que se centra en la interacción entre las computadoras y el lenguaje humano**. Su objetivo principal es permitir que las máquinas comprendan, interpreten y generen texto o voz de manera similar a como lo haría una persona
* **Análisis de texto y minería de datos**: La minería de datos es una **técnica asistida por computadora que se utiliza en los análisis para procesar y explorar grandes conjuntos de datos**. Gracias a las herramientas y métodos de minería de datos, las organizaciones pueden descubrir patrones y relaciones ocultas en sus datos.
* **Ejemplo:** La minería de datos es el proceso de hallar anomalías, patrones y correlaciones en grandes conjuntos de datos para predecir resultados. Empleando una amplia variedad de técnicas, puede utilizar esta información para **incrementar sus ingresos, recortar costos, mejorar sus relaciones con clientes, reducir riesgos y más**

**Lenguaje de programación C y C++**

* **Librerías fundamentales**:
* **STDIO :** es el [archivo de cabecera](https://es.wikipedia.org/wiki/Archivo_de_cabecera) que contiene las definiciones de las [macros](https://es.wikipedia.org/wiki/Macro), las [constantes](https://es.wikipedia.org/wiki/Constante_%28programaci%C3%B3n%29), las declaraciones de funciones de la [biblioteca estándar](https://es.wikipedia.org/wiki/Biblioteca_est%C3%A1ndar_de_C) del [lenguaje de programación](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) [C](https://es.wikipedia.org/wiki/C_%28lenguaje_de_programaci%C3%B3n%29) para hacer operaciones, estándar, de entrada y salida, así como la definición de [tipos](https://es.wikipedia.org/wiki/Tipo_de_dato) necesarias para dichas operaciones
* **CONIO:** es un [archivo de cabecera](https://es.wikipedia.org/wiki/Archivo_de_cabecera) escrito en [C](https://es.wikipedia.org/wiki/C_%28lenguaje_de_programaci%C3%B3n%29) usado mayormente por los compiladores de MS-DOS para proveer un sistema de [E/S](https://es.wikipedia.org/wiki/Perif%C3%A9rico_de_entrada/salida) por consola.[1](https://es.wikipedia.org/wiki/Conio.h#cite_note-1)​ Éste no es parte de la [biblioteca estándar de C](https://es.wikipedia.org/wiki/Biblioteca_est%C3%A1ndar_de_C) o [ISO C](https://es.wikipedia.org/wiki/ISO_C), ni está definida por [POSIX](https://es.wikipedia.org/wiki/POSIX).
* **Lista de 10 librerías más comunes en C/C++**, además de las ya mencionadas.
* **Manejadores de formatos para diferentes tipos de datos**, con ejemplos de uso en código.
* NOMBRE: SELVIN GIOVANNI PANIAGUA MARTINEZ