**Instituto Nacional de Educación Diversificada**

**INED**

**Santa Cruz Naranjo, Santa Rosa**

**Nombre**: Helen Daniela Arrecis Puaque

**Catedrático**: Gustavo Blanco

**Grado:** 5to Baco

**Catedra**: Computación (Producción)

**Clave:** 2

  Fecha: 15/03/2025

SIstema

GNet

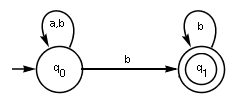
2025

**1. Teoría de autómatas**

* **Definición y usos de los autómatas finitos deterministas (AFD)**.

Es un sistema determinista; es decir, para cada estado en que se encuentre el autómata, y con cualquier símbolo del alfabeto leído, existe siempre no más de una transición posible desde ese estado y con ese símbolo.

* **Ejemplos y aplicaciones** en informática y ciencias de la computación.



Son conjunto de señales recibidas por el procesador de un ordenador, el que está compuesto por un número **finito** de compuertas (estados), cada una de las cuales pueden utilizar dos condiciones posibles, por lo general denotadas por 0 y 1(transiciones).

**2. Gramáticas formales**

* **Gramáticas regulares**: concepto y ejemplos de su uso en lenguajes de programación.

Generan los lenguajes regulares (aquellos reconocidos por un autómata finito). Son las gramáticas

Más restrictivas. El lado derecho de una producción debe contener un símbolo terminal y, como

Máximo, un símbolo no terminal. Estas gramáticas pueden ser:

- Lineales a derecha, si todas las producciones son de la forma

A ∈ N ∪ {S}

A → aB ó A → a B ∈ N

a ∈ T

(en el lado derecho de las producciones el símbolo no terminal aparece a la derecha del símbolo

terminal)

* **Gramáticas recursivas**: explicación y diferencias con las gramáticas regulares.

Una gramática recursiva es aquella que contiene reglas de producción que se repiten, lo que significa que al expandir un no terminal se puede obtener una cadena que incluya nuevamente ese no terminal

**3. Análisis léxico**

* **Tokenización**: qué es y cómo se usa en el proceso de compilación.

La tokenización en autómatas finitos deterministas (AFD) es el proceso de analizar cadenas de caracteres para aceptar o rechazarlas

* **Expresiones regulares**: su función en el análisis léxico.

Las expresiones regulares son un equivalente algebraico de los autómatas finitos deterministas (AFD). Ambas describen los mismos lenguajes, conocidos como lenguajes regulares

* **Reconocimiento de palabras clave**: cómo se identifican en los lenguajes de programación.
* es un proceso que identifica palabras clave en un programa de entrada. Se utiliza en el análisis léxico para generar tokens o secuencias de palabras clave.
* **Identificación de tokens**: clasificación y ejemplos prácticos.
* es un proceso que se realiza en el análisis léxico para obtener secuencias de palabras clave, identificadoras y constantes.

**4. Análisis sintáctico**

* **Tipos de análisis sintáctico**:
  + Análisis sintáctico descendente.

es un proceso que se realiza en el análisis léxico para obtener secuencias de palabras clave, identificadores y constantes.

* + Análisis sintáctico ascendente.

Es el analizador más costoso de implementar, aunque es el que reconoce más gramáticas.

* **Árboles de análisis sintáctico**: qué son y cómo ayudan en la estructura de los programas.

son representaciones gráficas que muestran la estructura de una gramática. Son útiles para analizar texto y capturar la jerarquía de la entrada.

**5. Algoritmos de análisis de cadenas**

* **Algoritmo de Boyer-Moore**: concepto, funcionamiento y su aplicación en la búsqueda de cadenas dentro de textos o programas.

son representaciones gráficas que muestran la estructura de una gramática. Son útiles para analizar texto y capturar la jerarquía de la entrada.

**6. Compiladores e intérpretes**

* **Estructura de un compilador**: definición y componentes.
* **Fases del compilador**:

son modelos matemáticos que se usan en compiladores para reconocer frases de programas

* + Análisis léxico.

es un proceso que permite reconocer palabras en un lenguaje regular. Se basa en expresiones regulares para identificar las unidades del lenguaje, o tokens.

* + Análisis sintáctico.

es el proceso de reconocer si una palabra de entrada es aceptada o rechazada por el autómata.

* + Generación de código.

es el proceso de reconocer si una palabra de entrada es aceptada o rechazada por el autómata.

* + Optimización de código.

es un proceso que mejora el rendimiento de un programa durante su ejecución. Se utiliza en compiladores para generar código más eficiente.

.

* **Lenguajes de programación utilizados en la creación de compiladores**: como **Lex y Yacc**.

como C, C++, Java, C#, BASIC, Pascal, COBOL, entre otro

**7. Aplicaciones prácticas de autómatas**

* **Reconocimiento de patrones** en diferentes áreas.

El reconocimiento de patrones es la ciencia que se ocupa de los procesos sobre ingeniería, computación y matemáticas relacionados con objetos físicos

* **Procesamiento de lenguaje natural** y su uso en inteligencia artificial.

Como rama de la inteligencia artificial, **utiliza el aprendizaje automático para procesar e interpretar textos y datos**. El reconocimiento y la generación de lenguaje natural son tipos de PLN.

* **Análisis de texto y minería de datos**: su importancia en el análisis de información.

La minería de textos es la práctica de analizar vastas colecciones de materiales textuales para capturar conceptos clave, tendencias y relaciones ocultas

**8. Lenguaje de programación C y C++**

* **Librerías fundamentales**: **STDIO y CONIO,** un resumen de las funciones de cada una.

Las funciones de STDIO son un conjunto de funciones que permiten realizar operaciones de entrada y salida de datos, manipular cadenas y caracteres, y realizar cálculos matemáticos

Las funciones de CONIO son funciones de la librería conio.h que permiten realizar entradas y salidas por consola.

Funciones de lectura

* **getch**: Lee un carácter directamente de la consola sin mostrar salida
* **getche**: Lee un carácter directamente de la consola mostrando salida

 **cgets**: Lee una cadena de caracteres y la guarda en un parametro

* **Lista de 10 librerías más comunes en C/C++**, además de las ya mencionadas.
* **assert.h**: Contiene la macro assert, que se utiliza para detectar errores lógicos y otros fallos en la depuración de un programa
* **string.h**: Se utiliza para manipular cadenas de caracteres
* **math.h**: Contiene funciones matemáticas
* **time.h**: Se utiliza para tratar y convertir entre formatos de fecha y hora
* **wchar.h**: Se utiliza para manipulas flujos de datos anchos y varias clases de cadenas de caracteres anchos
* **Iostream**: Para entrada y salida
* **Cmath**: Para operaciones matemáticas y transformaciones
* **Cstring**: Para manipulación de cadenas
* **Ctime**: Para manejo de fecha y hora
* **Algorithm**: Librería estándar de C++
* **Chrono**: Librería estándar de C++
* **Complex**: Librería estándar de C++
* **Exception**: Librería estándar de C++
* **Fstream**: Para manejo de archivos
* **List**: Para contenedores como listas
* **Vector**: Para contenedores como vectores
* **Math**: Para matemáticas
* **Memory**: Para memoria
* **String**: Para cadenas
* **Stdlib.h**: Para funciones generales
* **Manejadores de formatos para diferentes tipos de datos**, con ejemplos de uso en código.