Guided Decision System for Data Structures

- Alejandro Luis Cisneros Bernal
- ID: 182246

- Materia : Estructura de Datos
- 24 de Noviembre de 2025

Objetivo del Proyecto

- - Analizar requisitos del usuario mediante preguntas
- - Aplicar reglas basadas en estructuras de datos
- - Recomendar la estructura adecuada
- - Generar reporte completo (razón, visual, pseudocódigo, ejemplo)

Requisitos del Enunciado

- - 14 preguntas simples
- - Recomendación determinista
- - Output con:
- 1. Estructura sugerida
- 2. Razonamiento
- 3. Visual
- 4. Pseudocódigo
- 5. Ejemplo basado en tarea
- - Ejecutable con un solo comando

Estructuras Consideradas

- Array / List
- Linked List
- Stack (LIFO)
- Queue (FIFO)
- Binary Tree
- Graph

Arquitectura del Sistema

- /src → main.py, questions.py, decision.py, output.py
- /tests → test_rules.py
- /docs → preguntas, reglas, diagramas, ejemplo

Flujo General

1. USUARIO RESPONDE CUESTIONARIO

2. SE CREA OBJETO ANSWERS

3. REGLAS PROCESAN INFORMACIÓN

4. SE ASIGNA ESTRUCTURA

5. SE GENERA REPORTE

Cuestionario (Preguntas)

- - Acceso por índice
- - Inserciones / eliminaciones
- - LIFO / FIFO
- - Jerarquía
- - Relaciones múltiples
- - Tamaño dinámico

Reglas de Decisión

- - Índice rápido → Array
- - Muchas inserciones → Linked List
- - LIFO → Stack
- - FIFO → Queue
- - Jerarquía → Binary Tree
- - Relaciones múltiples → Graph

Ejemplo de uso

- **Ejemplo Real del Sistema – Hash Table**
- **Respuestas del usuario (ejecución real):**
- Índice por posición → **sí**
- Búsqueda rápida crítica → **sí**
- Clave–valor → **sí**
- Buscar por rango → **no**
- Prefijos → **sí**
- Muchas inserciones → **sí**
- Muchas eliminaciones → **sí**
- Inserciones en medio → **no**
- Tamaño variable → **no**
- Orden automático → **no**
- Prioridad → **sí**
- FIFO → **sí**
- LIFO → **sí**
- Relaciones múltiples → **no**

Resultado del sistema: Hash Table

```
Se recomienda: Hash Table

1. Búsqueda rápida crítica → Hash Table ofrece O(1) promedio
2. Asociación clave-valor requerida → Hash Table especializado
3. Inserción y eliminación: O(1) promedio
4. Ideal para diccionarios, cachés, bases de datos

=== Diagrama Visual ===

        Buckets:
        0: [ ]
        1: [("key1", value1)]
        2: [("key2", value2)]
        3: [ ]

        Hash function: h(key) → bucket_index
        Lookup promedio: O(1)


=== Pseudocódigo (con Complejidad Temporal) ===

// HASH TABLE OPERATIONS
PUT(key, value):            // O(1) promedio
    index = hash(key) % table_size
    bucket = table[index]
    append((key, value) to bucket)

GET(key):                   // O(1) promedio
    index = hash(key) % table_size
    bucket = table[index]
    for each (k, v) in bucket:
        if k == key:
            return v
    return NULL
```

Conclusiones

- - Integra teoría + práctica
- - Modular, explicable y probado
- - Útil como herramienta educativa
- - Cumple requisitos del proyecto
- - Escalable a GUI o nuevas estructuras