Documentación Sintetizada del Sistema de Gestión de Proveedores

Yax Puác Kevin Miguel – 1529422 – Estructura De Datos II

Este sistema es una aplicación web completa para la gestión de proveedores de servicios que utiliza un **Árbol B** como estructura de datos principal. El sistema permite almacenar, buscar y organizar información de proveedores de manera eficiente, ofreciendo múltiples criterios de búsqueda y ordenamiento.

Documentación de Clases

Clase: Proveedor

Propósito:
Modelo de datos que representa un proveedor de servicios.

Atributos:

- id: int — Identificador único (clave primaria)
- nombre: str — Nombre del proveedor
- servicio: str — Tipo de servicio ofrecido
- calificacion: float — Calificación del 1 al 5
- ubicacion: str — Ubicación geográfica

Métodos:

- to_dict() → dict
  Serializa el objeto a diccionario para almacenamiento y respuesta JSON.
- from_dict(datos: dict) → Proveedor
  Deserializa un diccionario a una instancia de Proveedor.
- __str__() → str
  Representación legible para humanos.
- __repr__() → str
  Representación técnica para debugging.

Clase: NodoB

Propósito:
Representa un nodo individual dentro del Árbol B.

Atributos:

- claves: List[int] — Lista ordenada de claves (IDs de proveedores)
- datos: List[dict] — Datos asociados a cada clave
- hijos: List[NodoB] — Referencias a nodos hijos
- es_hoja: bool — Indica si es un nodo hoja

Métodos:

- esta_lleno(grado: int) → bool
  Determina si el nodo ha alcanzado su capacidad máxima.

Invariantes:

- len(claves) == len(datos)
- Si es_hoja == False, entonces len(hijos) == len(claves) + 1
- Las claves están siempre ordenadas ascendentemente

Clase: ArbolB

Propósito:
Implementa la estructura de datos Árbol B para almacenar proveedores.

Atributos:

- raiz: NodoB — Nodo raíz del árbol
- grado: int — Grado mínimo del árbol (controla el balanceamiento)

Métodos Públicos:

- insertar(clave: int, datos: dict) → None
  Inserta un nuevo proveedor en el árbol.
- buscar(clave: int) → dict | None
  Busca un proveedor por ID.
- buscar_por_servicio(tipo_servicio: str) → List[dict]
  Encuentra proveedores por tipo de servicio.
- buscar_por_ubicacion(ubicacion: str) → List[dict]
  Encuentra proveedores por ubicación.
- obtener_todos_ordenados(orden_por: str) → List[dict]
  Obtiene todos los proveedores ordenados por nombre, calificación o ubicación.
- obtener_estadisticas() → dict
  Genera métricas estructurales del árbol.

Métodos Privados:

- _insertar_no_lleno()
- _dividir_hijo()
- _buscar_en_nodo()
- _buscar_servicio_en_nodo()
- _buscar_ubicacion_en_nodo()
- _recorrer_inorden()
- _contar_nodos()

- obtener_altura()

Flujo de Datos

- Inserción de Proveedor:
    a. Cliente envía POST con datos JSON
    b. Flask valida formato y campos requeridos
    c. Validaciones verifican integridad (ID único, calificación válida)
    d. Proveedor se instancia con datos validados
    e. ArbolB inserta usando ID como clave
    f. Árbol se rebalancea automáticamente si es necesario.
    g. Respuesta JSON confirma éxito o error

- Búsqueda por ID:
    a. Cliente solicita GET con ID específico
    b. Flask extrae ID de la URL
    c. ArbolB ejecuta búsqueda logarítmica
    d. Medición de tiempo de ejecución
    e. Respuesta con datos encontrados o mensaje de error

- Búsqueda por Servicio/Ubicación:
    a. Cliente solicita GET con criterio de búsqueda
    b. Flask extrae parámetro de la URL
    c. ArbolB recorre todo el árbol buscando coincidencias
    d. Comparación insensible a mayúsculas/minúsculas
    e. Acumulación de resultados coincidentes
    f. Respuesta con lista de resultados y métricas

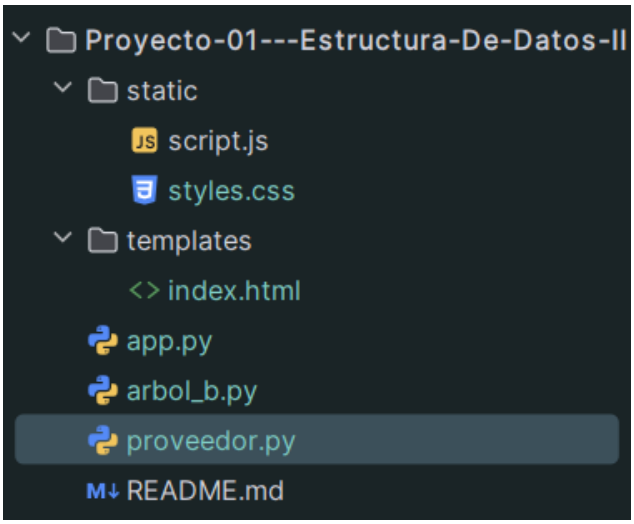
Requisitos del Sistema:

- Python 3.7+
- Flask 2.0+
- Sistema operativo: Windows/Linux/macOS

Instalación:

- Clonar o descargar los archivos
- Instalar Flask
- pip install flask

Estructura



Ejecución

- python app.py

Conclusión

Este sistema implementa una solución robusta y eficiente para la gestión de proveedores utilizando conceptos avanzados de estructuras de datos. El Árbol B proporciona excelente rendimiento para operaciones de búsqueda por ID, mientras que las búsquedas por atributos secundarios mantienen funcionalidad completa, aunque con mayor complejidad temporal.

Fortalezas del Sistema:

- Escalabilidad: Rendimiento logarítmico para operaciones principales
- Flexibilidad: Múltiples criterios de búsqueda y ordenamiento
- Robustez: Validaciones completas y manejo de errores
- Monitoreo: Métricas integradas para análisis de rendimiento
- Extensibilidad: Arquitectura modular fácil de expandir

Limitaciones Conocidas:

- Persistencia: Datos en memoria solamente
- Búsquedas Secundarias: O(n) para servicio y ubicación
- Concurrencia: No optimizado para múltiples usuarios simultáneos

El sistema es ideal para aplicaciones de tamaño medio donde se requiere búsqueda eficiente por ID y funcionalidad completa de gestión de proveedores de servicios.

Enlace A Repositorio:

https://github.com/KevinYax11/Proyecto-01---Estructura-De-Datos-II