iacc instituto profesional

EVALUACIÓN

ESTRUCTURA DE DATOS
Semana 8

Nombre del estudiante:
Fabiola Arrué Ravanal
Fecha de entrega: 03-03-2025
Carrera:
TÉCNICO DE NIVEL SUPERIOR EN INFORMÁTICA

Árboles Binarios.

1.- ¿Cuál es la estructura básica de un árbol binario y cómo se organizarían las rutas de entrega en la implementación del sistema de gestión utilizando un árbol binario?

R: Un árbol binario es una estructura de datos jerárquica en la que cada nodo puede tener un máximo de dos nodos hijos: uno a la izquierda y otro a la derecha. Este tipo de organización facilita la manipulación eficiente de los datos, optimizando tareas como la búsqueda, inserción y eliminación de elementos.

En un árbol binario, la información se distribuye de manera estructurada:

- La raíz es el nodo principal del árbol, desde donde se derivan los demás nodos.
- Cada nodo puede tener hasta dos nodos hijos, que amplían la estructura en diferentes direcciones.
- La altura del árbol corresponde al número máximo de niveles que lo componen.
- La profundidad de un nodo indica la distancia entre ese nodo y la raíz.

Existen distintas formas de estructurar un árbol binario según sus propiedades:

- Un árbol binario completo tiene todos los niveles llenos, salvo el último, que puede no estar completamente ocupado.
- Un árbol binario perfecto mantiene todos sus niveles completamente llenos y balanceados.
- Un árbol binario balanceado tiene una diferencia de altura de, como máximo, uno entre sus subárboles, lo que permite mejorar la eficiencia en la búsqueda y recuperación de información.
- Un árbol binario de búsqueda (BST) organiza los nodos de manera ordenada: los valores menores se ubican en el subárbol izquierdo y los mayores en el subárbol derecho, lo que optimiza las operaciones de búsqueda.

En la gestión de rutas de una empresa de logística, el árbol binario de búsqueda permite administrar los datos de manera eficiente. En este caso, cada nodo representa una ruta, y su organización dentro del árbol facilita la optimización y consulta de información. Para estructurar este sistema:

- La raíz se asigna a la ruta más utilizada o prioritaria, asegurando acceso rápido a la información más relevante.
- En el subárbol izquierdo se almacenan rutas con menor tráfico o importancia.
- En el subárbol derecho se organizan las rutas con mayor demanda.

Por ejemplo, si la ruta más utilizada conecta la bodega central con un centro comercial, esta se ubicará en la raíz. Otras rutas menos transitadas, como la que lleva a una zona residencial, estarán en la izquierda, mientras que aquellas con alta demanda, como la que abastece supermercados, estarán en la derecha.

Además, un Árbol Binario de Búsqueda (BST) permite que las operaciones de búsqueda, inserción y eliminación sean más eficientes en comparación con una estructura lineal como una lista. En promedio, estas operaciones tienen una complejidad de O(log n), lo que significa que el tiempo de ejecución crece mucho más lento que el número de rutas almacenadas. Gracias a esta organización, la consulta y administración de las rutas se optimizan, facilitando una gestión rápida y estructurada del sistema.

Dentro del sistema de gestión, esta organización permite que las rutas puedan consultarse, modificarse y eliminarse de manera eficiente, optimizando el acceso a la información y mejorando la planificación logística.

Esta estructura no solo optimiza la búsqueda de rutas, sino que también facilita su actualización y gestión, lo que mejora la eficiencia operativa de la empresa.

2.- ¿Cuáles son las principales operaciones y aplicaciones que se pueden realizar en un árbol binario y cómo se aplican en el contexto de la gestión de rutas de entrega?

R: Los árboles binarios permiten organizar información de manera eficiente a través de operaciones que optimizan su estructura y manipulación. Estas operaciones son esenciales para gestionar datos en sistemas donde la organización jerárquica es clave, como en la administración de rutas de entrega.

Entre las principales operaciones de un árbol binario se encuentran:

- **Inserción**: Cada nodo se posiciona dentro del árbol en función de su relación con los demás. En estructuras organizadas, como los árboles binarios de búsqueda, las rutas menos relevantes se ubican en la parte izquierda, mientras que las más utilizadas se posicionan a la derecha. Este método garantiza que la inserción de una nueva ruta no altere la eficiencia del sistema. En promedio, esta operación tiene una complejidad de O(log n), lo que permite agregar nuevas rutas sin afectar significativamente el rendimiento.
- **Búsqueda:** Gracias a la disposición jerárquica de los nodos, encontrar una ruta en un árbol binario no requiere recorrer todos los elementos. En cada paso, se toma una decisión que reduce el espacio de búsqueda, permitiendo identificar la mejor opción de manera rápida y eficiente. El tiempo de búsqueda también es O(log n) en promedio, lo que optimiza el acceso a la información sobre rutas.
- **Eliminación:** Existen tres escenarios al eliminar un nodo del árbol:
    - Si el nodo no tiene descendientes, se elimina sin afectar la estructura general.
    - Si tiene un único nodo hijo, este lo reemplaza directamente en la estructura, conectándose con el padre del nodo eliminado. Esto mantiene la continuidad del árbol sin afectar su organización.
    - Si el nodo tiene dos hijos, se reemplaza por el nodo con el menor valor dentro de su subárbol derecho (sucesor inorden) o el mayor valor dentro de su subárbol izquierdo (predecesor inorden). Esto garantiza que el árbol conserve su orden y estructura.

- **Recorridos:** Los diferentes métodos de recorrido permiten obtener información del árbol según las necesidades del sistema:
    - **Inorden (izquierda-raíz-derecha):** Organiza la información en un orden secuencial útil para reportes. Este recorrido se eligió porque permite listar las rutas de forma ordenada en función de un criterio numérico, como la distancia o el identificador de la ruta. Si el objetivo hubiera sido priorizar las rutas más utilizadas en primer lugar, el preorden habría sido una mejor opción. Sin embargo, para reportes estructurados y organizados, el inorden es ideal.
    - **Preorden (raíz-izquierda-derecha):** Se usa para replicar estructuras del árbol.
    - **Postorden (izquierda-derecha-raíz):** Facilita el análisis de rutas menos utilizadas para su optimización.

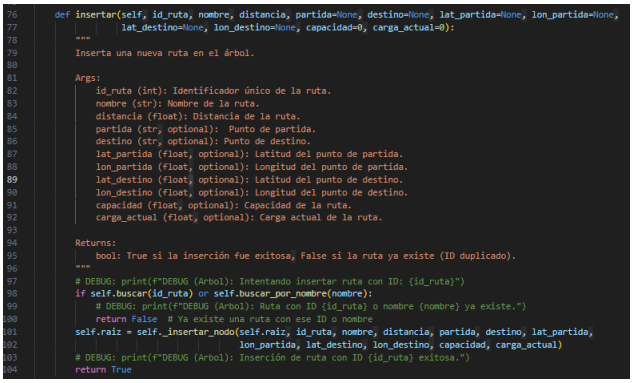
En la implementación dentro del sistema de gestión de rutas, estas operaciones permiten que las nuevas entregas se integren sin alterar el rendimiento del sistema, asegurando que las rutas más utilizadas tengan prioridad y que la eliminación de rutas innecesarias no afecte la integridad de la estructura. Dado que las principales operaciones del BST tienen una complejidad de O(log n) en promedio, el sistema mantiene su eficiencia incluso al manejar grandes volúmenes de datos.
Esto optimiza el acceso a la información, mejora la toma de decisiones y permite una administración más eficiente de la logística empresarial.

3.- Implementa un árbol binario en Python para resolver de manera eficiente y efectiva los problemas relacionados con la gestión de rutas de entrega. Considera la necesidad de agregar, buscar y eliminar rutas, así como generar informes sobre su eficiencia y capacidad de carga.

R: La gestión eficiente de rutas de entrega requiere una estructura de datos que optimice la organización y consulta de la información. Para ello, se ha implementado un Árbol Binario de Búsqueda (BST) en Python, permitiendo realizar las operaciones fundamentales necesarias para la administración de rutas.

Los principales procedimientos que permite este sistema son:

- **Agregar rutas:** Inserta nuevas rutas asegurando que el árbol conserve su orden lógico, lo que facilita búsquedas eficientes.



```
    def insertar(self, id_ruta, nombre, distancia, partida=None, destino=None, lat_partida=None, lon_partida=None,
                 lat_destino=None, lon_destino=None, capacidad=0, carga_actual=0):
        """
        Inserta una nueva ruta en el árbol.

        Args:
            id_ruta (int): Identificador único de la ruta.
            nombre (str): Nombre de la ruta.
            distancia (float): Distancia de la ruta.
            partida (str, optional):  Punto de partida.
            destino (str, optional): Punto de destino.
            lat_partida (float, optional): Latitud del punto de partida.
            lon_partida (float, optional): Longitud del punto de partida.
            lat_destino (float, optional): Latitud del punto de destino.
            lon_destino (float, optional): Longitud del punto de destino.
            capacidad (float, optional): Capacidad de la ruta.
            carga_actual (float, optional): Carga actual de la ruta.

        Returns:
            bool: True si la inserción fue exitosa, False si la ruta ya existe (ID duplicado).
        """
        # DEBUG: print(f"DEBUG (Arbol): Intentando insertar ruta con ID: {id_ruta}")
        if self.buscar(id_ruta) or self.buscar_por_nombre(nombre):
            # DEBUG: print(f"DEBUG (Arbol): Ruta con ID {id_ruta} o nombre {nombre} ya existe.")
            return False  # Ya existe una ruta con ese ID o nombre
        self.raiz = self._insertar_nodo(self.raiz, id_ruta, nombre, distancia, partida, destino, lat_partida,
                                         lon_partida, lat_destino, lon_destino, capacidad, carga_actual)
        # DEBUG: print(f"DEBUG (Arbol): Inserción de ruta con ID {id_ruta} exitosa.")
        return True
```

- **Buscar rutas:** Encuentra una ruta específica sin necesidad de recorrer toda la estructura, optimizando el acceso a la información.

```python
    def buscar(self, id_ruta):
        """
        Busca una ruta en el árbol por su ID.

        Args:
            id_ruta (int): El ID de la ruta a buscar.

        Returns:
            Nodo: El nodo que contiene la ruta, o None si no se encuentra.
        """
        return self._buscar_nodo(self.raiz, id_ruta)

    def _buscar_nodo(self, nodo, id_ruta):
        """
        Función auxiliar recursiva para buscar un nodo por ID.
        """
        if nodo is None:
            return None
        if nodo.id_ruta == id_ruta:
            return nodo
        if id_ruta < nodo.id_ruta:
            return self._buscar_nodo(nodo.izquierda, id_ruta)
        return self._buscar_nodo(nodo.derecha, id_ruta)
```

- **Eliminar rutas:** Permite eliminar rutas innecesarias sin comprometer la estructura del árbol, aplicando correctamente los tres casos de eliminación.

```python
    def eliminar(self, id_ruta):
        """
        Elimina una ruta del árbol por su ID.

        Args:
            id_ruta (int): El ID de la ruta a eliminar.
        Returns:
            bool: True si se elimino correctamente, False si no se encontro.
        """
        self.raiz, eliminado = self._eliminar_nodo(self.raiz, id_ruta)
        return eliminado

    def _eliminar_nodo(self, nodo, id_ruta):
        """
        Funcion auxiliar recursiva para eliminar un nodo por ID
        """
        if nodo is None:
            return nodo, False #Nodo no encontrado

        if id_ruta < nodo.id_ruta:
            nodo.izquierda, eliminado = self._eliminar_nodo(nodo.izquierda, id_ruta)
        elif id_ruta > nodo.id_ruta:
            nodo.derecha, eliminado = self._eliminar_nodo(nodo.derecha, id_ruta)
        else: #Encontramos el nodo a eliminar
            if nodo.izquierda is None:
                return nodo.derecha, True #Caso 1 y 2: 0 o 1 hijo
            elif nodo.derecha is None:
                return nodo.izquierda, True #Caso 2: 1 hijo

            #Caso 3: 2 hijos
            sucesor = self._minimo_valor(nodo.derecha) #Encontrar el sucesor inorden
            #Copiar los datos del sucesor al nodo actual
            nodo.id_ruta, nodo.nombre, nodo.distancia, nodo.partida, nodo.destino, nodo.lati
            #Eliminar el sucesor
            nodo.derecha, _ = self._eliminar_nodo(nodo.derecha, sucesor.id_ruta)
            return nodo, True
        return nodo, eliminado #Retorna el nodo modificado y si se elimino
```

- **Generar informes:** Utiliza un recorrido inorden, que permite listar las rutas en orden basado en su prioridad numérica o su distancia, facilitando reportes organizados.

```python
def generar_informe(self):
    rutas = self.arbol.obtener_rutas()
    if not rutas:
        messagebox.showinfo("Informe", "No hay rutas registradas.")
        return

    # --- Usar filedialog.asksaveasfilename ---
    try:
        ruta_archivo = filedialog.asksaveasfilename(
            initialdir=os.getcwd(),  # Directorio inicial (opcional)
            title="Guardar informe como",
            filetypes=(("Archivos CSV", "*.csv"), ("Todos los archivos", "*.*")),
            defaultextension=".csv"  # Importante: Agregar extensión por defecto
        )

        if not ruta_archivo:  # El usuario canceló el diálogo
            return

        with open(ruta_archivo, "w", newline="", encoding="utf-8") as f:
            escritor_csv = csv.writer(f)
            escritor_csv.writerow(["ID", "Ruta", "Distancia (km)", "Partida", "Destino","Latitud Partida", "Longitud Partida", "Latitud Destino", "Lo
            for id_ruta, nombre, distancia, partida, destino, lat_partida, lon_partida, lat_destino, lon_destino, capacidad, carga_actual in rutas:
                #Calculemos la eficiencia
                eficiencia = "N/A"
                if capacidad > 0:
                    porcentaje = (carga_actual / capacidad) * 100
                    if porcentaje < 50:
                        eficiencia = "Baja"
                    elif 50 <= porcentaje < 80:
                        eficiencia = "Media"
                    else:
                        eficiencia = "Alta"
                escritor_csv.writerow([id_ruta, nombre, distancia, partida, destino, lat_partida, lon_partida, lat_destino, lon_destino, capacidad, c

        messagebox.showinfo("Informe", f"Informe guardado en: {ruta_archivo}")

        # --- Abrir el archivo ---
        if os.name == 'nt':
            os.startfile(ruta_archivo)
        else:
            try:
                subprocess.run(["xdg-open", ruta_archivo], check=True)
            except FileNotFoundError:
                try:
                    subprocess.run(["open", ruta_archivo], check=True)
                except FileNotFoundError:
                    messagebox.showwarning("Advertencia","No se pudo abrir el archivo automáticamente.")

    except Exception as e:
        messagebox.showerror("Error", f"Error al guardar/abrir el informe: {e}")
```

Según Fritelli, Guzmán y Tymoschuk (2020), "La implementación de un árbol binario puede hacerse en forma dinámica, de manera similar a lo realizado con las listas. Cada nodo del árbol se define como un registro con al menos tres campos." Esto se adapta perfectamente al contexto de la gestión de rutas, permitiendo estructurar la información de manera eficiente y escalable.

Uno de los aspectos clave de esta implementación es la posibilidad de ingresar direcciones de dos maneras:

- **De forma manual**, escribiendo la dirección.
- **A través de un mapa interactivo**, permitiendo seleccionar visualmente los puntos de partida y destino.

Independientemente del método de ingreso, el sistema calcula automáticamente la distancia entre las ubicaciones seleccionadas, reduciendo la posibilidad de errores en la estimación de distancias.

El código incorpora diversas bibliotecas para mejorar su funcionalidad y precisión:

- **geopy**, utilizada para obtener coordenadas y calcular distancias geográficas.
- **tkintermapview**, que permite la integración de un mapa interactivo en la interfaz gráfica.
- **csv**, que facilita el almacenamiento y recuperación automática de rutas.
- **webbrowser**, que permite abrir la ruta seleccionada en OpenStreetMap para su visualización.

Además, se ha desarrollado una interfaz gráfica en Tkinter, que permite a los usuarios:

- Ingresar nuevas rutas sin necesidad de manipular el código directamente.
- Consultar rutas almacenadas mediante búsquedas eficientes.
- Eliminar rutas innecesarias de forma sencilla.
- Generar informes visuales sobre la eficiencia y la capacidad de carga de cada ruta.

Se presentan a continuación imágenes referenciales del programa y de la interfaz gráfica en ejecución. **Su código fuente se subirá junto con esta tarea a la plataforma de entrega**.

Fragmentos de código

```python
class Nodo:
    """
    Representa un nodo en el árbol binario.  Cada nodo contiene la información
    de una ruta de entrega.
    """
    def __init__(self, id_ruta, nombre, distancia, partida=None, destino=None,
                 lat_partida=None, lon_partida=None, lat_destino=None,
                 lon_destino=None, capacidad=0, carga_actual=0):
        """
        Inicializa un nuevo nodo.

        Args:
            id_ruta (int): Identificador único de la ruta.
            nombre (str): Nombre de la ruta.
            distancia (float): Distancia de la ruta en kilómetros.
            partida (str, optional): Lugar de partida.
            destino (str, optional): Lugar de destino.
            lat_partida (float, optional): Latitud del punto de partida.
            lon_partida (float, optional): Longitud del punto de partida.
            lat_destino (float, optional): Latitud del punto de destino.
            lon_destino (float, optional): Longitud del punto de destino.
            capacidad (float, optional): Capacidad máxima de carga de la ruta.
            carga_actual (float, optional): Carga actual de la ruta.
        """
        self.id_ruta = id_ruta
        self.nombre = nombre
        self.distancia = distancia
        self.partida = partida
        self.destino = destino
        self.latitud_partida = lat_partida
        self.longitud_partida = lon_partida
        self.latitud_destino = lat_destino
        self.longitud_destino = lon_destino
        self.izquierda = None  # Nodo hijo izquierdo
        self.derecha = None    # Nodo hijo derecho
        self.capacidad = capacidad
        self.carga_actual = carga_actual
```

```python
import tkinter as tk
from tkinter import messagebox, filedialog
from tkinter import ttk
import os
import csv
from geopy.geocoders import Nominatim
from geopy.exc import GeocoderTimedOut, GeocoderUnavailable
from geopy.distance import geodesic
import webbrowser
import tkintermapview
import subprocess

# Asegurar el directorio de trabajo correcto
script_dir = os.path.dirname(os.path.abspath(__file__))
os.chdir(script_dir)

class Nodo:
    """
    Representa un nodo en el árbol binario.  Cada nodo contiene la información
    de una ruta de entrega.
    """
```

```python
if __name__ == "__main__":
    root = tk.Tk()
    app = Aplicacion(root)
    root.mainloop()
```

Interfaz gráfica:

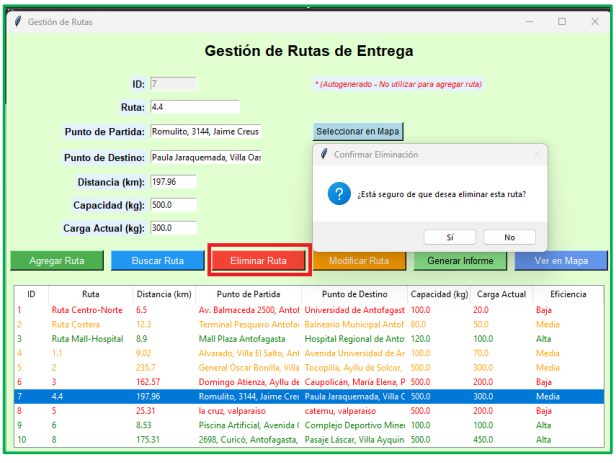
Al inicializar

Buscar ruta (en este caso inexistente)

Ingresar Ruta

Generar Informe

Eliminar Ruta



La implementación del BST en Python permite gestionar rutas de entrega de manera eficiente, asegurando acceso rápido a la información y facilitando la toma de decisiones estratégicas. Además, la incorporación de tecnologías de geolocalización y mapas interactivos garantiza precisión en la administración de rutas. El código está documentado, lo que facilita su mantenimiento y futuras mejoras.

REFERENCIAS BIBLIOGRÁFICAS

IACC. (2025). *Estructura de datos.* Árboles Binarios. Semana 8

Fritelli, V., Guzmán, A., & Tymoschuk, J. (2020). *Algoritmos y estructuras de datos* (2ª ed.). Córdoba, Argentina: Jorge Sarmiento Editor - Universitas. Recuperado de https://elibro.net/es/ereader/iacc/175249?page=314.

IACC. (2024). *S8 podcast inicio comencemos con los Árboles Binarios.* https://soundcloud.com/user-571686720/s8-podcast-inicio-etgdt1303/s-MDRaAWINKOa?si=4311e7b56a8649bbae7c5bd31624a8c1&utm_source=clipboard&utm_medium=text&utm_campaign=social_sharing

IACC. (2024). *Concepto de Árbol Binario.* https://rise.articulate.com/share/Z2YjanKhYAj8zPQpaik-zwSn8HBZO3rl#/

IACC. (2024). *Operaciones con árboles binarios de búsqueda.* https://view.genially.com/64777c0e119e9d0017da8467

IACC. (2024). *Sinteticemos sobre los Árboles Binarios.* https://publicaciones.iacc.cl/publicacion/2849-estructuras-de-datos