Explicación del Código de la Aplicación de Recomendación

Justin Daniel Rivera López – 9490-23-10643
Brayan Kenet Rivera Quinilla – 9490-23-2835
Marco Tulio Pineda Recinos – 9490-23-2906

29 de mayo de 2025

1. Introducción

Este documento ofrece una explicación detallada del código Python utilizado para construir una aplicación web en Flask que incluye las siguientes funcionalidades:

- Estructura de datos de un *Árbol B* para almacenamiento eficiente.
- Modelado de entidades turísticas y de hospedaje.
- Construcción de un grafo ponderado para cálculo de rutas.
- Lógica de recomendación de rutas.
- Endpoints REST para interacción con el frontend.

2. Módulo de Árbol B

Se define la clase `BTreeNode` que representa cada nodo del árbol y la clase `BTree` que implementa las operaciones principales:

- `search`: búsqueda de claves.
- `insert` y `_insert_nonfull`: inserción en nodos no llenos.
- `split_child`: división de nodos llenos.
- `traverse`: recorrido inorden para obtener todos los valores.

```python
class BTreeNode:
    def __init__(self, t, leaf=False):
        self.t = t  # grado m nimo del  rbol
        self.leaf = leaf
        ...

class BTree:
    def __init__(self, t=3):
        self.root = BTreeNode(t, leaf=True)
        self.t = t

    def search(self, k, x=None):
        ...

    def insert(self, k, v):
        ...
```

Listing 1: Definición de BTreeNode y BTree

3. Modelado de Entidades

Se crean las clases base y derivadas para representar los lugares de interés:

- `Entity`: clase genérica con atributos comunes.
- `TouristSpot` y `Hotel`: extienden `Entity` con detalles específicos.
- Método `add_rating` para actualizar calificaciones.

```python
class Entity:
    def __init__(self, identifier, name, entity_type, lat, lon,
       price, avg_rating):
        ...

class TouristSpot(Entity):
    def __init__(self, identifier, name, lat, lon, price,
       avg_rating, est_stay):
        super().__init__(...)

class Hotel(Entity):
    def __init__(self, identifier, name, lat, lon, price,
       avg_rating):
        super().__init__(...)
```

Listing 2: Definición de entidades

4. Grafo Ponderado y Dijkstra

Se implementa un grafo mediante la clase `WeightedGraph`:

- `add_vertex` y `add_edge` para construir la red.
- `dijkstra` para calcular distancias mínimas desde un nodo origen.

```
class WeightedGraph:
    def __init__(self):
        self.adj = {}
    def add_vertex(self, v): ...
    def add_edge(self, u, v, w): ...
    def dijkstra(self, start): ...
```

Listing 3: Definición de `WeightedGraph`

5. Construcción del Grafo y Funciones Auxiliares

Antes de recomendar rutas, se:

1. Se parsean valores numéricos con `parse_float`.
2. Se construye el grafo calculando distancias geodésicas entre entidades.
3. Se define `score_place` para evaluar lugares según presupuesto y tiempo.

```
def parse_float(v, d=0.0): ...

def build_graph(): ...

def score_place(entity, budget, time_left, travel_time): ...
```

Listing 4: Parseo, construcción de grafo y scoring

6. Lógica de Recomendación y Endpoints

Se define `recommend_route` que utiliza Dijkstra y `score_place` para elegir hasta 5 lugares. Asimismo, se exponen varios endpoints en Flask:

- `/upload_entities` y `/upload_ratings` para carga de datos CSV.
- `/recommend` para obtener recomendaciones JSON.
- `/add_manual`, `/get_entities` y `/download_btree`.

```
@app.route('/recommend', methods=['POST'])
def recommend():
    data = request.json
    ...
    return {"recommendations": out}
```

Listing 5: Ejemplo de endpoint de recomendación

7. Capturas de Pantalla de Funcionamiento

En esta sección puedes insertar ejemplos visuales del funcionamiento de la aplicación, como resultados de peticiones `/recommend`, visualización del Árbol B, o rutas generadas:

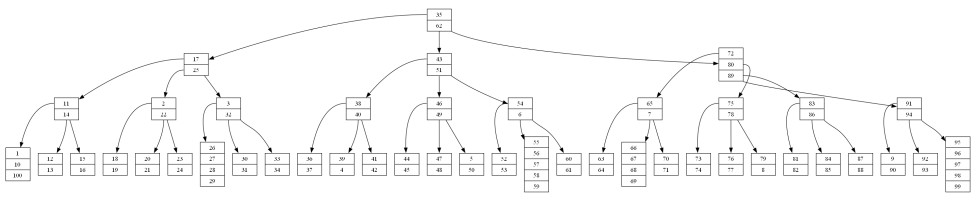

Figura 1: Visualización del Árbol B tras inserciones

Figura 2: Ejemplo de respuesta JSON de /recommend

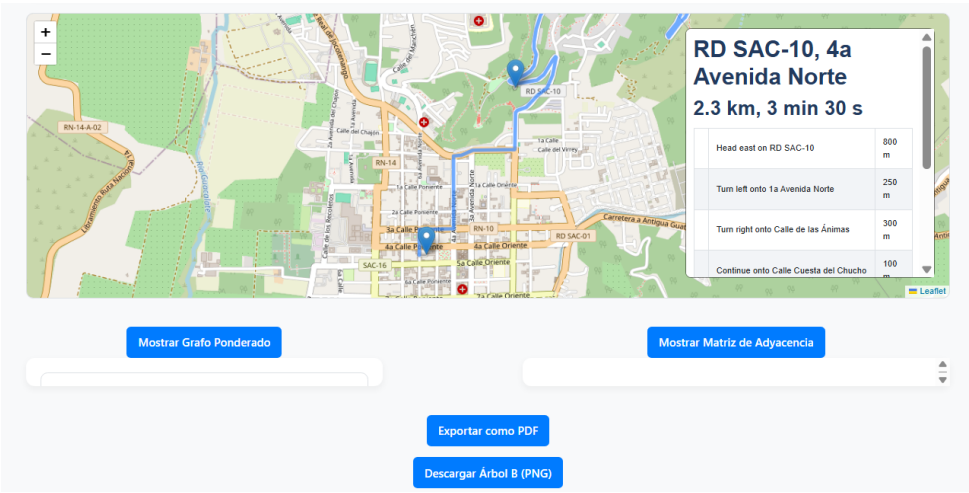

Figura 3: Visualización del recomendaciones en mapa

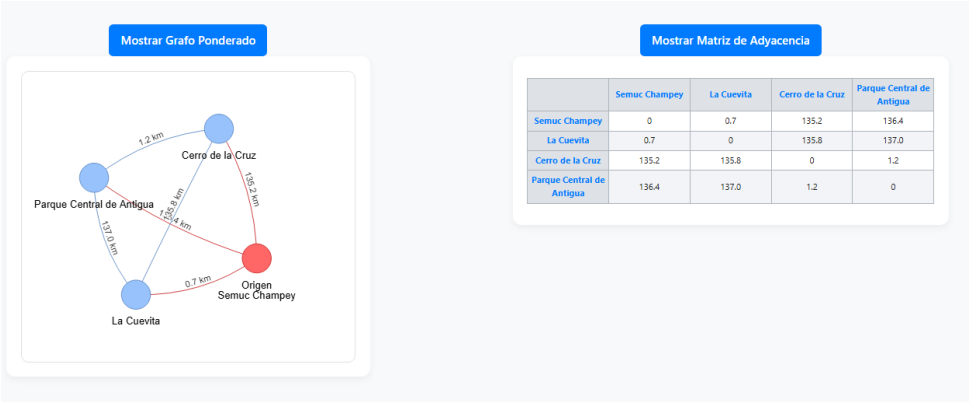

Figura 4: Visualización del grafo ponderado y matriz de adyacencia