Universidad Mariano Gálvez de Guatemala
Ingeniería en Sistemas de la Información y Ciencias de la Computación
Programación III
Ing. José Miguel Villatoro Hidalgo

PROYECTO FINAL (MANUAL TÉCNICO)

Erwin Fernando Blanco Melendres
Carnet: 9490-23-7748

Sara Abigail Solis Ixquiactap

9490-23-12295
Sección: “B”

Guatemala 30 de mayo de 2025

Introducción

Sistema de Búsqueda turísticas de viajes, el siguiente manual es para la comprensión exacta de código fuente que se utilizo y diagrama de clases que se utilizaron para la creación inicial, media y final del proyecto, el objetivo principal es poner en practica las estructuras de datos B para su búsqueda y conceptos vistos en clase, para ello el diagrama de clases es primordial para verificar lo que se busca en cada una de las entidades y clases en el código que se explicará a continuación.


Entidad
<<abstract>>
- id: str
- nombre: str
- tipo: str
// "Hospedaje" o "Turístico"
- latitud: float
- longitud: float
- precio: float
- calificacion: float
+mostrar_info(): str
LugarTuristico
- tiempo_estadia: float
+recomendar(): bool
Hospedaje
Calificacion
- entidad_id: str
- puntuacion: float
- comentario: str
Mapa
- grafo: Grafo
+mostrar_mapa()
+calcular_ruta()
Recomendador
- mapa: Mapa
- presupuesto: float
+recomendar_lugares()
Grafo
- nodos: dict
- aristas: dict
+agregar_nodo()
+agregar_arista()
+camino_optimo()
ArbolB
- raiz: NodoB
+insertar()
+buscar()
+exportar_graphviz()

Proyecto final

- Classes
  - B_tree.py
    - La clase funciona para crear un primer nodo con grado mínimo y verificar si hay hoja o raíz inicial, listar por claves los nodos raíz, o listar nodos hijos o punteros
    - Se inicializa con 0 (valor actual de la raíz)

```python
class BTreeNode:
    def __init__(self, t, is_leaf):
        self.t = t  # Grado mínimo (orden del árbol)
        self.is_leaf = is_leaf  # Booleano: ¿Es un nodo hoja?
        self.keys = []  # Lista de claves en este nodo
        self.children = []  # Lista de nodos hijos (punteros a otros nodos)
        self.n = 0  # Número actual de claves en el nodo

    def __repr__(self):
        return f"Node(keys={self.keys}, leaf={self.is_leaf})"
```

    - La clase BTree se inicializa con la raíz.

```python
class BTree:
    def __init__(self, order):
        self.order = order  # Corresponde a 't' en la teoría de B-trees
        self.root = BTreeNode(order, True)  # La raíz es inicialmente una hoja
        self.next_id_counter = 1 # Para generar IDs únicos si se necesita
```

    - Se crean clases como buscar

```python
def search(self, key):
    return self._search(self.root, key)

def _search(self, x, key):
    i = 0
    while i < x.n and key > x.keys[i].id:
        i += 1
    if i < x.n and key == x.keys[i].id:
        return x.keys[i]  # Se encontró la clave (el objeto Place)
    if x.is_leaf:
        return None  # No se encontró y es una hoja
    return self._search(x.children[i], key) # Busca en el hijo apropiado
```

    - La clase insertar

```python
def insert(self, key_id, value):
    root_node = self.root
    # Si la raíz está llena, el árbol crece en altura
    if root_node.n == (2 * self.order) - 1:
        s = BTreeNode(self.order, False) # Nueva raíz no hoja
        s.children.insert(0, root_node) # La vieja raíz es ahora el primer hijo de la nueva raíz
        self._split_child(s, 0) # Divide la vieja raíz
        self.root = s # La nueva raíz es ahora la raíz del árbol
    self._insert_non_full(self.root, key_id, value)

def _insert_non_full(self, x, key_id, value):
    i = x.n - 1
    if x.is_leaf:
        # Encuentra la posición correcta para insertar la clave
        while i >= 0 and key_id < x.keys[i].id:
            i -= 1
        x.keys.insert(i + 1, value) # Inserta el objeto Place
        x.n += 1
    else:
        # Encuentra el hijo donde se debería insertar
        while i >= 0 and key_id < x.keys[i].id:
            i -= 1
        i += 1 # i es el índice del hijo donde se insertará

        # Si el hijo está lleno, divídelo
        if x.children[i].n == (2 * self.order) - 1:
            self._split_child(x, i)
            # Decide en qué hijo continuar la inserción después de la división
            if key_id > x.keys[i].id:
                i += 1
        self._insert_non_full(x.children[i], key_id, value)
```

- Clases necesarias para verificar si el puntero está lleno según la clave dada

```python
def _split_child(self, x, i):
    t = self.order
    y = x.children[i] # y es el hijo que está lleno
    z = BTreeNode(t, y.is_leaf) # z es el nuevo nodo que se creará

    x.keys.insert(i, y.keys[t - 1]) # Sube la clave mediana de y a x
    x.n += 1

    z.n = t - 1 # El nuevo nodo z toma t-1 claves
    y.n = t - 1 # El nodo y se queda con t-1 claves

    # Copia las claves de la mitad derecha de y a z
    z.keys = y.keys[t : (2 * t) - 1]
    y.keys = y.keys[0 : t - 1]

    if not y.is_leaf:
        # Si y no es una hoja, z también toma la mitad derecha de los hijos de y
        z.children = y.children[t : (2 * t)]
        y.children = y.children[0 : t]

    x.children.insert(i + 1, z) # Inserta z como un nuevo hijo de x
```

- La ultima parte de código del b_tree contiene el generar árbol B por graphiz

```python
def get_all_elements(self):
    elements = []
    self._traverse_and_collect(self.root, elements)
    return elements

def _traverse_and_collect(self, node, elements):
    if not node:
        return

    # Recorre los hijos y las claves
    for i in range(node.n):
        if not node.is_leaf:
            self._traverse_and_collect(node.children[i], elements)
        elements.append(node.keys[i]) # Las claves son los objetos Place

    # Asegúrate de recorrer el último hijo si no es una hoja
    if not node.is_leaf:
        self._traverse_and_collect(node.children[node.n], elements)

def to_dot(self):
    """Genera una representación DOT del árbol para Graphviz."""
    dot = graphviz.Digraph(comment='B-Tree Structure', graph_attr={'rankdir': 'TB', 'splines': 'true'})
    node_counter = [0] # Usar una lista para mutabilidad

    def add_nodes_edges(node, parent_name=None, child_idx=None):
        current_node_name = f'node{node_counter[0]}'
        node_counter[0] += 1

        # Crear la etiqueta del nodo
        label = '|'.join([f'<p{j}>' for j in range(node.n)] +
                         [str(key.id) for key in node.keys] +
                         [f'<p{j}>' for j in range(node.n, 2 * self.order - 1)])

        # Simplificar la etiqueta para mostrar solo las claves
        label_keys = '|'.join([str(key.id) for key in node.keys])
        dot.node(current_node_name, label=f"{{ {label_keys} }}", shape='record')

        if parent_name:
            # Conectar el nodo padre con este nodo hijo
            # Si queremos conectar a un puerto específico del padre:
            dot.edge(f'{parent_name}:p{child_idx}', current_node_name)

        if not node.is_leaf:
            for i, child in enumerate(node.children):
                if child: # Asegurarse de que el hijo no sea None
                    add_nodes_edges(child, current_node_name, i)
```

- Place.py
  - Se comienza con una clase place que almacenar las calificaciones, hospedajes, y turismos por sus id’s según los csv, el cual se puede actualizar según lo que se almacene

```python
class Place:
    def __init__(self, id, name, latitude, longitude, price, rating):
        self.id = id
        self.name = name
        self.latitude = latitude
        self.longitude = longitude
        self.price = price
        self.average_rating = rating
        self.ratings = [] # Para almacenar objetos UserRating

    def update_rating(self, new_rating, comment=""):
        # Esto es una simplificación; en un sistema real, querrías una lista de UserRating
        # y recalcular el promedio. Por ahora, solo actualiza el promedio.
        # Asumiendo que 'ratings' es una lista de UserRating objetos
        self.ratings.append(UserRating(self.id, "some_user", new_rating, comment)) # 'some_user' es un placeholder
        total_rating = sum(r.rating for r in self.ratings)
        self.average_rating = total_rating / len(self.ratings) if self.ratings else 0

    def to_dict(self):
        return {
            'id': self.id,
            'name': self.name,
            'latitude': self.latitude,
            'longitude': self.longitude,
            'price': self.price,
            'average_rating': self.average_rating,
        }
```

- Estas clases siguientes únicamente funcionan para el almacenaje y carga de datos de archivos con contenido masivo

```python
class Hospedaje(Place):
    def __init__(self, id, name, latitude, longitude, price, rating):
        super().__init__(id, name, latitude, longitude, price, rating)

    def to_dict(self):
        return super().to_dict()

class LugarTuristico(Place):
    def __init__(self, id, name, latitude, longitude, price, rating, estimated_stay_hours=0):
        super().__init__(id, name, latitude, longitude, price, rating)
        self.estimated_stay_hours = estimated_stay_hours
        # Cargar calificaciones existentes si las hay, o inicializar lista vacía
        self.ratings = []

    def to_dict(self):
        data = super().to_dict()
        data['estimated_stay_hours'] = self.estimated_stay_hours
        return data

class UserRating:
    def __init__(self, place_id, user_id, rating, comment=""):
        self.place_id = place_id
        self.user_id = user_id
        self.rating = rating
        self.comment = comment

    def to_dict(self):
        return {
            'place_id': self.place_id,
            'user_id': self.user_id,
            'rating': self.rating,
            'comment': self.comment
        }
```

- Data
  - Hospedajes.csv (archivo que se crea al ejecutar sistema)
  - Turísticos.csv (archivo que se crea al ejecutar sistema)
  - Calificaciones.csv (archivo que se crea al ejecutar sistema)
- Static
  - Css (contiene el estilo de la pagina index principal)
  - Img (aquí se almacena la imagen generada en graphiz)
  - Js (aquí se almacena la funcionalidad del mapa)
    - Se almacena la funcionalidad del mapa basados en Google Maps, comenzando con las coordenadas de Guatemala central.

```
function initMap() {
    const guatemala = { lat: 14.6349, lng: -90.5069 };
    map = new google.maps.Map(document.getElementById("map"), {
        zoom: 7,
        center: guatemala,
    });
    console.log("Google Map initialized.");
}
```

- Esta clase funciona para buscar en los archivos csv los lugares turísticos

```
function toggleEstimatedStay() {
    const placeType = document.getElementById('add_place_type').value;
    const estimatedStayDiv = document.getElementById('estimated_stay_div');
    if (placeType === 'Turistico') {
        estimatedStayDiv.style.display = 'block';
        document.getElementById('estimated_stay_hours').setAttribute('required', 'required');
    } else {
        estimatedStayDiv.style.display = 'none';
        document.getElementById('estimated_stay_hours').removeAttribute('required');
    }
}
```

- Esta función busca en el mapa las ubicaciones y las muestra.

```
function toggleSearchFields() {
    const searchType = document.getElementById('search_type').value;
    const nameFields = document.getElementById('search_by_name_fields');
    const coordsFields = document.getElementById('search_by_coords_fields');

    if (searchType === 'name') {
        nameFields.style.display = 'block';
        coordsFields.style.display = 'none';
        document.getElementById('search_name').setAttribute('required', 'required');
        document.getElementById('search_latitude').removeAttribute('required');
        document.getElementById('search_longitude').removeAttribute('required');
        document.getElementById('search_tolerance').removeAttribute('required'); // No requerido para nombre
    } else { // coords
        nameFields.style.display = 'none';
        coordsFields.style.display = 'block';
        document.getElementById('search_name').removeAttribute('required');
        document.getElementById('search_latitude').setAttribute('required', 'required');
        document.getElementById('search_longitude').setAttribute('required', 'required');
        document.getElementById('search_tolerance').setAttribute('required', 'required');
    }
}
```

- Existe la clase donde se verifica datos existentes con los csv cargados o almacenados con anterioridad

```
async function searchPlace(event) {
    event.preventDefault();

    const searchType = document.getElementById('search_type').value;
    const searchResultsDiv = document.getElementById('search_results');
    searchResultsDiv.innerHTML = ''; // Clear previous results
    clearMapElements(); // Clear existing markers/polylines from map

    let formData = new FormData();
    formData.append('search_type', searchType);

    if (searchType === 'name') {
        const nameQuery = document.getElementById('search_name').value;
        if (!nameQuery) {
            alert("Por favor, ingresa un nombre para buscar.");
            return;
        }
        formData.append('query', nameQuery);
    } else { // coords
        const latQuery = document.getElementById('search_latitude').value;
        const lonQuery = document.getElementById('search_longitude').value;
        const tolerance = document.getElementById('search_tolerance').value;
        if (!latQuery || !lonQuery || !tolerance) {
            alert("Por favor, ingresa latitud, longitud y tolerancia para buscar por coordenadas.");
            return;
        }
        formData.append('lat_query', latQuery);
        formData.append('lon_query', lonQuery);
        formData.append('tolerance', tolerance);
    }
```

- Esta clase verifica si hay o no datos cargados o algunas referencias por mostrar

```
try {
    const response = await fetch('/search_place', {
        method: 'POST',
        body: formData
    });
    const results = await response.json();

    if (results.length === 0) {
        searchResultsDiv.innerHTML = '<p>No se encontraron lugares turísticos que coincidan.</p>';
    } else {
        const ul = document.createElement('ul');
        results.forEach(place => {
            const li = document.createElement('li');
            li.textContent = `${place.name} (ID: ${place.id}, Lat: ${place.latitude}, Lon: ${place.longitude}, Calificación: ${place.average_rating})`;
            ul.appendChild(li);

            // Add marker for found place
            addMarker({ lat: place.latitude, lng: place.longitude }, place.name, "🔍");
        });
        searchResultsDiv.appendChild(ul);
        if (results.length > 0) {
            // Centrar el mapa en el primer resultado encontrado
            map.setCenter({ lat: results[0].latitude, lng: results[0].longitude });
            map.setZoom(10); // Zoom in on the found place(s)
        }
    }
} catch (error) {
    console.error('Error searching place:', error);
    searchResultsDiv.innerHTML = '<p>Hubo un error al buscar el lugar.</p>';
}
```

- B_tree_structure.png (esta clase almacena el árbol B)
- Templates
    - Index.html (muestra todo la pagina inicial de Flask al iniciar)
    - Map.html
        - Este apartado únicamente contiene el API que se utilizo para Google Maps
        - ```

          <script>
              // Initialize map (Google Maps or Leaflet.js)
              function initMap() {
                  // Google Maps example
                  const guatemala = { lat: 15.7835, lng: -90.2308 }; // Central Guatemala
                  const map = new google.maps.Map(document.getElementById("map"), {
                      zoom: 7,
                      center: guatemala,
                  });

                  // Add markers for places from your data (fetched via API or preloaded)
                  // Draw routes for recommendations
              }
          </script>
          <script async defer src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&callback=initMap"></script>
          ```
- Utils
    - Data_loader.py
        - Contiene principalmente la carga de datos masiva, aquí ya se verifica lo antes cargado para ver si existen o no datos o asegurar una carga limpia
        - ```
          def load_data_from_csv(filepath, entity_type, b_tree_instance):
              """
              Carga datos de un archivo CSV en un B-tree.
              """
              # Reinicia el B-tree si ya tiene elementos para asegurar una carga limpia
              # Pequeña corrección aquí, la condición estaba incorrecta.
              if b_tree_instance.root is not None:
                  b_tree_instance.__init__(order=b_tree_instance.order)

              print(f"--- Attempting to load {entity_type} data from {filepath} ---")
          ```
        - Esta clase analiza y verifica los datos cargados por calificaciones.csv

```python
def load_user_ratings_from_csv(filepath, turisticos_b_tree_instance):
    """
    Carga las calificaciones de los usuarios y las asocia a los lugares turísticos.
    """
    print(f"--- Attempting to load User Ratings from {filepath} ---")
    try:
        with open(filepath, mode='r', newline='', encoding='utf-8') as file:
            reader = csv.DictReader(file)
            if not reader.fieldnames:
                print(f"Warning: {filepath} is empty or has no headers.")
                return

            for row in reader:
                try:
                    place_id = row['place_id']
                    user_id = row['user_id']
                    rating = float(row['rating'])
                    comment = row.get('comment', '')

                    place = turisticos_b_tree_instance.search(place_id)
                    if place:
                        # Asegúrate de que place.ratings sea una lista de objetos UserRating
                        place.ratings.append(UserRating(place_id, user_id, rating, comment))
                        # Recalcular el average_rating
                        total_rating = sum(r.rating for r in place.ratings)
                        place.average_rating = total_rating / len(place.ratings)
                        # print(f"Loaded rating for {place.name}: {rating}")
                    else:
                        print(f"Warning: Place ID {place_id} not found in B-tree for rating: {row}")
                except KeyError as ke:
                    print(f"Skipping rating row due to missing column '{ke}': {row}")
                except ValueError as ve:
                    print(f"Skipping rating row due to data type error: {ve} in row {row}")
                except Exception as e:
                    print(f"Skipping rating row due to unexpected error: {e} in row {row}")

        print(f"--- Finished loading User Ratings from {filepath} ---")

    except FileNotFoundError:
        print(f"Error: User ratings file not found at {filepath}")
    except Exception as e:
        print(f"An unexpected error occurred while loading user ratings from {filepath}: {e}")
```

- Esta ultima clase realiza si no existen los csv los crea con los encabezados necesarios.

```python
def export_b_tree_to_csv(b_tree_instance, filepath, entity_type):
    """
    Exporta los datos de un B-tree a un archivo CSV.
    """
    print(f"--- Attempting to export {entity_type} data to {filepath} ---")
    elements = b_tree_instance.get_all_elements()
    if not elements:
        print(f"No {entity_type} elements to export.")
        # Asegurarse de crear el archivo con encabezado si no hay datos
        if entity_type == "Hospedaje":
            fieldnames = ['id', 'name', 'latitude', 'longitude', 'price', 'average_rating']
        elif entity_type == "Turistico":
            fieldnames = ['id', 'name', 'latitude', 'longitude', 'price', 'average_rating', 'estimated_stay_hours']
        else:
            fieldnames = [] # No debería pasar
        if fieldnames:
            with open(filepath, mode='w', newline='', encoding='utf-8') as file:
                writer = csv.DictWriter(file, fieldnames=fieldnames)
                writer.writeheader()
        return
```

- El demás código que aparece son las explicaciones de verificación si se cargo algo o no.

- Map_utils.py
  - Esta clase almacena la longitud en kilómetros de las recomendaciones

```python
# utils/map_utils.py
import math

def haversine_distance(lat1, lon1, lat2, lon2):
    R = 6371  # Radius of Earth in kilometers

    lat1_rad = math.radians(lat1)
    lon1_rad = math.radians(lon1)
    lat2_rad = math.radians(lat2)
    lon2_rad = math.radians(lon2)

    dlon = lon2_rad - lon1_rad
    dlat = lat2_rad - lat1_rad

    a = math.sin(dlat / 2)**2 + math.cos(lat1_rad) * math.cos(lat2_rad) * math.sin(dlon / 2)**2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))

    distance = R * c
    return distance
```

- Route_calculator.py
  - Esta clase calcula las dimensiones solicitadas, precio, estancia por hora, costo de estacionamiento por hora.

```python
def find_best_routes(origin_lat, origin_lon, daily_budget, turistic_places, gmaps_client):
    """
    Encuentra las mejores rutas de lugares turísticos basadas en el presupuesto, la ubicación,
    el tiempo de viaje y priorizando calificación.
    """
    print(f"DEBUG(route_calculator): find_best_routes called with budget {daily_budget} and {len(turistic_places)} places.")

    origin_coords = f"{origin_lat},{origin_lon}"
    recommended_places_with_details = []

    # Constantes para el cálculo de costos (ejemplo, ajusta según necesidad real)
    COSTO_POR_KM_GASOLINA = 1.21 # Precio de gasolina por km en Q (referencia de Guatemala)
    COSTO_ESTACIONAMIENTO_POR_HORA = 15.0 # Costo de estacionamiento en Q/hora en Antigua
    TIEMPO_ESTIMADO_VISITA_MIN = 1 # Consideramos al menos 1 hora de estacionamiento si hay visita
```

- El código restante únicamente explica las direcciones por Google Maps API, y prácticamente calcula lo necesario para la estancia, distancia, duración.

- App.py
  - Específicamente hablaremos de librerías utilizadas como Flask, Graphiz, exportación de las clases anteriores y llamados.

```python
# app.py
from flask import Flask, render_template, request, redirect, url_for, jsonify, send_file
from utils.data_loader import load_data_from_csv, load_user_ratings_from_csv, export_b_tree_to_csv, export_user_ratings_to_csv
from classes.b_tree import BTree
from classes.place import Hospedaje, LugarTuristico
from utils.route_calculator import find_best_routes
from utils.map_utils import haversine_distance # Aunque haversine ya no se usa para rutas, es buena práctica tenerla
from config import Maps_API_KEY
import os
import graphviz
import googlemaps # Asegúrate de tener 'google-maps-services-python' instalado (pip install google-maps-services-python)
```

  - Se inicia el árbol

```python
app = Flask(__name__)

# Initialize your B-trees globally
hospedajes_b_tree = BTree(order=3)
turisticos_b_tree = BTree(order=3)
```

  - Esta clase define y crea los archivos csv con llamados anteriores los crea únicamente si no existen, y también inicializa googlemaps.client que funciona para las API

```python
# Define data paths
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
DATA_DIR = os.path.join(BASE_DIR, 'data')
HOSPEDAJE_CSV = os.path.join(DATA_DIR, 'hospedajes.csv')
TURISTICOS_CSV = os.path.join(DATA_DIR, 'turisticos.csv')
CALIFICACIONES_CSV = os.path.join(DATA_DIR, 'calificaciones.csv')

# Crea la carpeta 'data' si no existe
os.makedirs(DATA_DIR, exist_ok=True)

# Crea archivos CSV vacíos con encabezados si no existen.
for filename in [HOSPEDAJE_CSV, TURISTICOS_CSV, CALIFICACIONES_CSV]:
    if not os.path.exists(filename):
        print(f"Creating empty CSV file with headers: {filename}")
        with open(filename, 'w', newline='', encoding='utf-8') as f:
            if 'hospedajes' in filename:
                f.write('id,name,latitude,longitude,price,average_rating\n')
            elif 'turisticos' in filename:
                f.write('id,name,latitude,longitude,price,average_rating,estimated_stay_hours\n')
            elif 'calificaciones' in filename:
                f.write('place_id,user_id,rating,comment\n')


# Inicializa el cliente de Google Maps API
gmaps = googlemaps.Client(key=Maps_API_KEY)
```

- El resto de código únicamente son llamados de las clases ya mencionadas con anterioridad.
- Este código analiza y exporta y crea los archivos csv y los almacena en la data del sistema y crea la imagen png del arbol

```python
@app.route('/export_entities')
def export_entities():
    export_b_tree_to_csv(hospedajes_b_tree, HOSPEDAJE_CSV, "Hospedaje")
    export_b_tree_to_csv(turisticos_b_tree, TURISTICOS_CSV, "Turistico")
    export_user_ratings_to_csv(turisticos_b_tree, CALIFICACIONES_CSV)
    return "All entity data exported to CSV files in the 'data/' directory. Check your server's 'data/' folder."

@app.route('/export_structure')
def export_structure():
    dot_source = turisticos_b_tree.to_dot()
    graph = graphviz.Source(dot_source)
    try:
        output_path = os.path.join(app.root_path, 'static', 'b_tree_structure')
        graph.render(output_path, view=False, format='png')
        print(f"B-tree structure exported to {output_path}.png")
        return send_file(output_path + '.png', mimetype='image/png', as_attachment=True, download_name='b_tree_structure.png')
    except Exception as e:
        print(f"Error exporting Graphviz: {e}")
        return f"Error generating B-tree structure: {e}. Make sure Graphviz is installed (e.g., 'brew install graphviz' on macOS, 'sudo apt-get install graphviz' on Debian/Ubuntu
```

- Config.py
  - Se utilizo una API con cuenta personal para la clave, donde se utilizan APIS especiales de Google

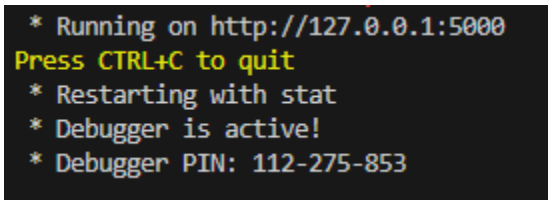
```python
# config.py
# Aquí puedes poner configuraciones globales
# Por ahora, solo una clave de API de Google Maps (si la usas)
Maps_API_KEY = "AIzaSyBvGwZtAS2CT840Pqc73hCuDZAm9RsM1Qg" # Reemplaza con tu clave
```

- Mis_calificaciones.csv (archivo masivo de carga de datos)
- Mis_hospedajes.csv (archivo masivo de carga de datos)
- Mis_turisticos.csv (archivo masivo de carga de datos)
- Readme.md (archivo de requerimientos)

El sistema fue desarrollado en entorno web con FLASK y PYTHON, los cuales fueron recomendados, con la herramienta de visual code. Por la base del proyecto con arboles de tipo B, no se procedió a utilizar base de datos.

Pruebas:

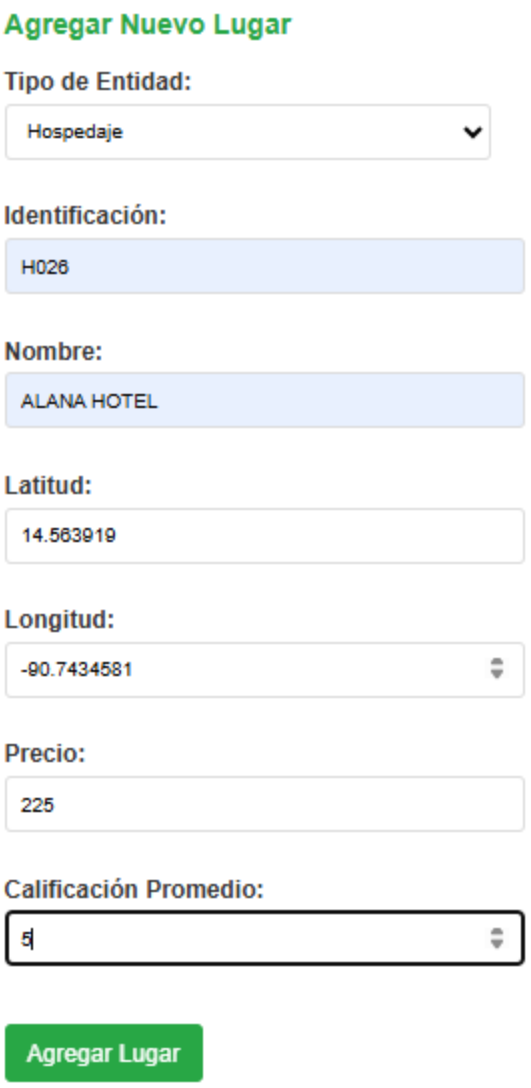
Se corre en el siguiente puerto



Carga masiva de datos

Se utilizara calificar lugar existente para verificar la carga masiva

Se agrega nuevo lugar



La carga masiva de datos son 25 lugares, dando a notar que los datos que se colocaron anteriormente se añadieron correctamente

```
id,name,latitude,longitude,price,average_rating
H001,Hotel Real InterContinental Guatemala,14.6062,-90.5284,150.0,4.6
H002,The Westin Camino Real,14.6015,-90.5255,180.0,4.7
H003,Barceló Guatemala City,14.6,-90.535,120.0,4.5
H004,Hyatt Centric Guatemala City,14.6065,-90.505,165.0,4.7
H005,Hotel Tikal Futura,14.617,-90.565,100.0,4.4
H006,Hostal Tequila Sunrise,14.63,-90.515,25.0,4.5
H007,Central Hostel Reforma,14.603,-90.512,20.0,4.2
H008,Hotel Casa Santo Domingo (Antigua),14.5601,-90.7302,250.0,4.8
H009,Hotel Porta Antigua (Antigua),14.556,-90.734,190.0,4.6
H010,Hotel Palacio de Doña Beatriz (Antigua),14.5565,-90.7305,170.0,4.7
H011,Hotel Atitlán (Lago Atitlán),14.73,-91.2,140.0,4.5
H012,Hotel Panajachel (Lago Atitlán),14.742,-91.16,80.0,4.0
H013,Hotel Sac Chich (Cobán),15.485,-90.37,75.0,4.3
H014,Ramada by Wyndham Guatemala City,14.609,-90.508,95.0,4.1
H015,Comfort Inn Guatemala City,14.6085,-90.502,70.0,3.9
H016,Hostal Los Cuatros (Flores),16.929,-89.892,30.0,4.1
H017,Hotel La Posada de Don Rodrigo (Antigua),14.558,-90.733,160.0,4.6
H018,Hotel Soleil Antigua (Antigua),14.554,-90.736,110.0,4.3
H019,Hotel Regis (Quetzaltenango),14.83,-91.52,60.0,3.8
H020,Hotel Casa del Parque (Antigua),14.5605,-90.7315,130.0,4.5
H021,La Casa de Don David (Flores),16.9295,-89.891,50.0,4.2
H022,Hotel Maya Palace (Petén),17.228,-89.622,90.0,4.0
H023,Hotel Casona de la Calle Real (Antigua),14.559,-90.7325,145.0,4.7
H024,Hotel Museo Casa de Botrán (Guatemala City),14.605,-90.52,115.0,4.3
H025,Hilton Garden Inn Guatemala City,14.612,-90.51,105.0,4.2
H026,ALANA HOTEL,14.563919,-90.7434581,225.0,5.0
```

Nos refleja al momento de buscar recomendaciones las mejores 3 (top 3) ya que basados por las claves de árbol b y promedio de calificación la preferencia fue de top 3. Los muestra en el mapa y la distancia que hay entre ellos según los requerimientos solicitados.

Generar Recomendaciones

Latitud de Origen (Ej: 14.6349):

14.6349

Longitud de Origen (Ej: -90.5069):

-90.5069

Presupuesto Diario (Q):

1000

Obtener Recomendaciones

Nuestras Mejores Recomendaciones:

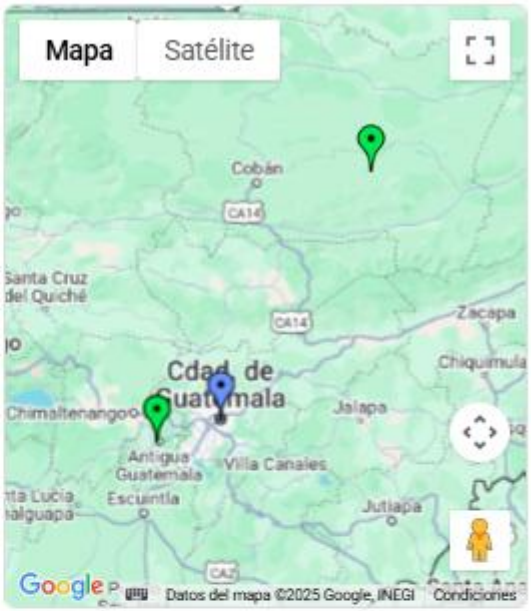
Ingresa tu destino y presupuesto para obtener recomendaciones.

Arco de Santa Catalina (Antigua) (ID: T012)
Calificación: 5 / 5
Precio de Entrada: Q0.00
Estancia Estimada: 1 horas
Tiempo de Viaje desde Origen: 1 hour 5 mins
Distancia desde Origen: 39.5 km
Costo Estimado de Viaje: Q62.80
Costo Total Estimado (Entrada + Viaje): Q62.80

Antigua Guatemala (Centro Histórico) (ID: T002)
Calificación: 4.8999999999999995 / 5
Precio de Entrada: Q0.00
Estancia Estimada: 6 horas
Tiempo de Viaje desde Origen: 1 hour 4 mins
Distancia desde Origen: 39.4 km
Costo Estimado de Viaje: Q137.66
Costo Total Estimado (Entrada + Viaje): Q137.66

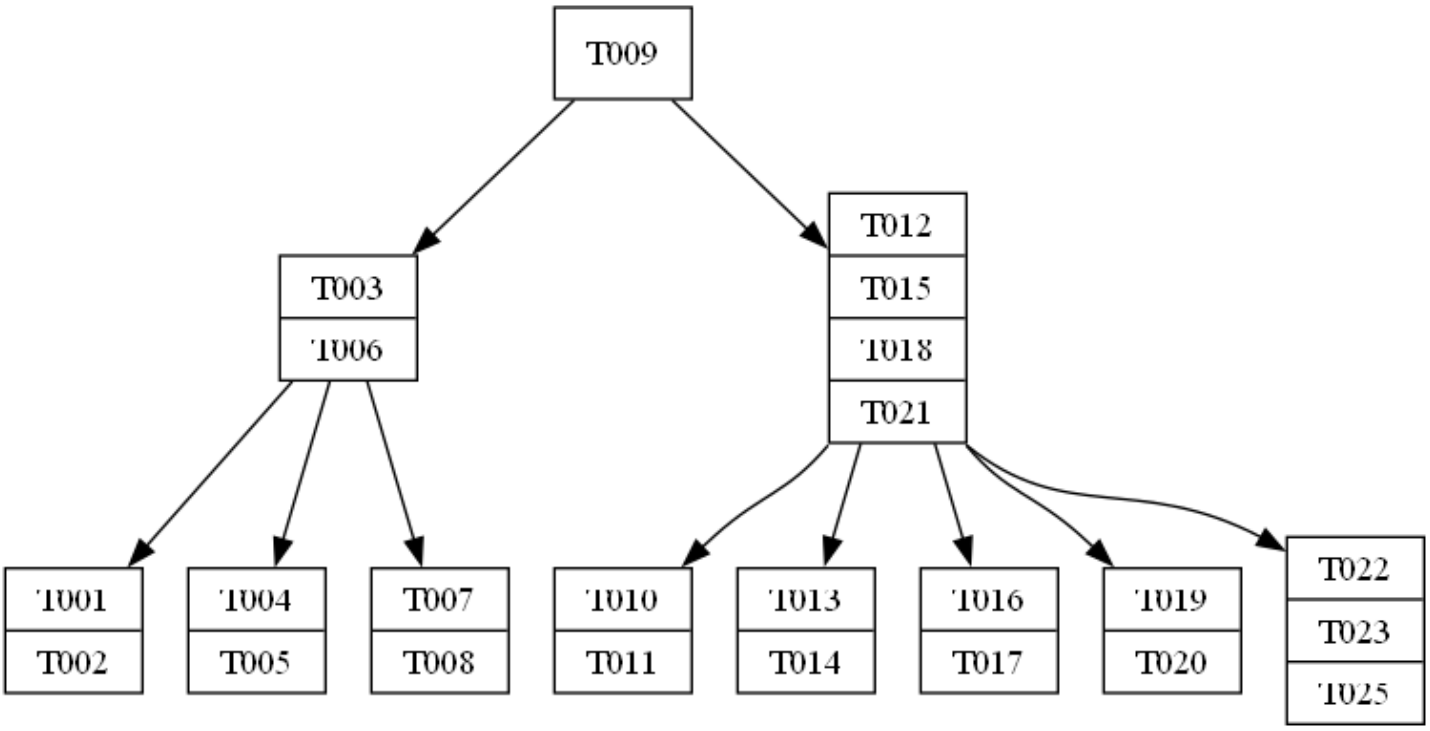
Semuc Champey (ID: T005)
Calificación: 4.8999999999999995 / 5
Precio de Entrada: Q15.00
Estancia Estimada: 6 horas
Tiempo de Viaje desde Origen: 6 hours 24 mins
Distancia desde Origen: 289 km
Costo Estimado de Viaje: Q439.88
Costo Total Estimado (Entrada + Viaje): Q454.88

Visualización del Mapa



Se finalizaría con las exportaciones de datos, tantas entidades (se almacenan ya en la data del proyecto) y exporta árbol b en formato png.

Dando por finalizada el árbol B con lo requerido según los lugares turísticos cargados



El link del proyecto puede encontrarlo en

https://github.com/ErwinBlanco-UMG/ProyectoFinal.git