FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

DESIGN AND ANALYSIS OF ALGORITHMS

CSC4202

SEMESTER 4 2023/2024

GROUP PROJECT

LECTURER: DR NORWATI MUSTAPHA

NO.	NAME	MATRIC NO.
1.	MUHAMMAD HANIF MURTAZA	213793
2.	YANG ZIXUN	213844
3.	CAI ZESHUO	213733

Introduction

This report presents an analysis of an algorithm developed to find the optimal routes from the Faculty of Computer Science at Universiti Putra Malaysia (UPM) to various key locations within and around the campus. The goal of this project is to assist international students at UPM in navigating the campus and nearby areas efficiently by considering real-time traffic data.

Our project focuses on designing, implementing, and analyzing an algorithm that can recommend the best routes based on traffic conditions during different time slots. The project involves several key steps: defining the problem, modeling the network of routes, designing the algorithm, implementing it, and evaluating its performance.

To achieve this, we use Dijkstra's Algorithm, a well-known algorithm for finding the shortest paths in a weighted graph. The algorithm is implemented in Java, and the network of routes is modeled as a graph where nodes represent key locations, and edges represent the routes between them with associated travel times.

This report covers the following sections:

1. **Problem Definition**: Describing the context and objectives of the project.
2. **Model Development**: Illustrating the graph representation of the campus map.
3. **Algorithm Design**: Explaining the choice of Dijkstra's Algorithm and its suitability for our problem.
4. **Implementation**: Detailing the steps involved in coding the algorithm in Python.
5. **Algorithm Analysis**: Analyzing the time complexity of the algorithm in best, worst, and average case scenarios.
6. **Evaluation and Results**: Presenting the performance of the algorithm through specific examples.
7. **Conclusion**: Summarizing the findings and future work.

This comprehensive approach ensures that the algorithm is not only theoretically sound but also practically effective in providing optimal route recommendations to UPM students.

Problem Definition

Navigating a large university campus like Universiti Putra Malaysia (UPM) can be challenging for international students, especially those who are new to the area and drive their own vehicle to go to school and back. With numerous buildings, facilities, and off-campus locations, it is essential to find the most efficient routes to save time and avoid getting lost. This project aims to solve this problem by developing an algorithm that recommends the optimal routes from the Faculty of Computer Science to the apartments of international students around UPM.

The main objectives of this project are:

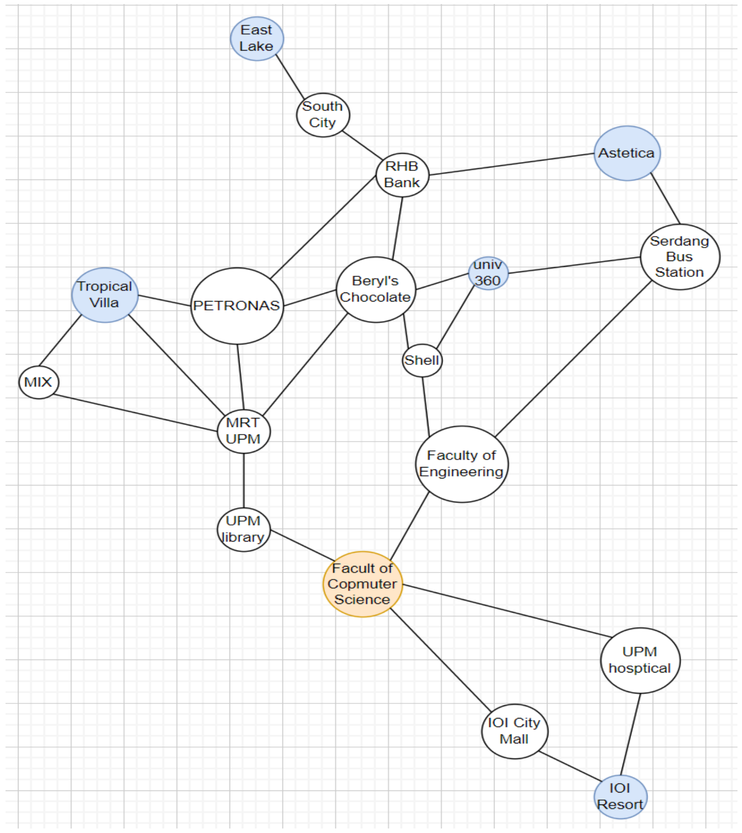
1. **To model the UPM campus and its surrounding areas as a graph**: Nodes represent key locations such as buildings, bus stations, and popular destinations, while edges represent the paths between these locations with associated travel times.
2. **To design and implement an algorithm**: The algorithm will compute the shortest paths from a given starting point (the Faculty of Computer Science) to all of the apartment options on the graph.
3. **To consider traffic data**: The algorithm will provide different route recommendations based on traffic conditions during three time slots: 9-11 AM, 11 AM-2 PM, and 2-6 PM. This requires analyzing different graphs for each time slot due to varying travel times.
4. **To ensure the algorithm's efficiency**: By analyzing the time complexity of the algorithm in the best, worst, and average case scenarios, we ensure it is suitable for real-time application.

This project will help UPM international students by providing a tool that offers efficient route recommendations, thereby enhancing their overall campus experience.

Model Development

Graph Representation

The UPM campus and its surrounding areas are represented as a weighted, undirected graph. In this graph:



- **Nodes (Vertices)**: Represent key locations such as buildings, bus stations, and popular destinations around the UPM campus.
- **Edges**: Represent the paths connecting these locations. Each edge has an associated weight, which corresponds to the estimated travel time between the two locations it connects.

Data Collection

The data for the graph was collected by identifying major landmarks and their connections. The travel times between these landmarks were estimated based on real-world distances and typical traffic conditions. These travel times were adjusted for three different time slots to account for varying traffic conditions:

- **9-11 AM**: Morning rush hour
- **11 AM-2 PM**: Midday
- **2-6 PM**: Afternoon and evening rush hour

Graph Construction

The graph was constructed using the following steps:

1. **Identifying Key Locations**: Key locations on and around the UPM campus were identified and marked as nodes. Examples include the Faculty of Computer Science, UPM Library, and IOI City Mall.
2. **Mapping Connections**: Paths between these locations were mapped based on actual roads and walkways.
3. **Estimating Travel Times**: Travel times for each path were estimated for the three different time slots. These times were used as weights for the edges connecting the nodes.

Algorithm Choice

For this project, Dijkstra's algorithm was chosen to find the shortest paths from the Faculty of Computer Science to all other locations on the graph. Dijkstra's algorithm is well-suited for this problem because it efficiently finds the shortest path in a graph with non-negative weights.

Implementation

The algorithm was implemented in Python, leveraging simplicity. The main steps in the implementation included:

1. **Graph Initialization**: Initializing the graph with nodes and weighted edges based on the collected data.

2. **Algorithm Execution**: Running Dijkstra's algorithm from the starting node (Faculty of Computer Science) to find the shortest paths to all other nodes.
3. **Time Slot Consideration**: Running the algorithm separately for each of the three time slots to account for varying travel times.

Expected Outcome

The expected outcome of this model is an efficient route recommendation system that provides the optimal path from the Faculty of Computer Science to any of the apartment location on the graph. By considering different traffic conditions in the three time slots, the system will offer time-specific recommendations, improving its practicality and accuracy for real-world use.

Algorithm Implementation

In this project, we utilized Dijkstra's algorithm to find the shortest path from the school to various destinations based on real-time traffic data. Below is the implementation of Dijkstra's algorithm in Python.

Dijkstra's Algorithm

The Dijkstra's algorithm implementation is found in the routing_algorithm.py file. The code initializes the distances and previous nodes for each vertex in the graph, then uses a priority queue to explore the shortest paths.

```python
import heapq

def dijkstra(graph, start):
    distances = {vertex: float('infinity') for vertex in graph}
    distances[start] = 0
    previous_nodes = {vertex: None for vertex in graph}
    priority_queue = [(0, start)]

    while priority_queue:
        current_distance, current_vertex = heapq.heappop(priority_queue)

        for neighbor, weight in graph[current_vertex].items():
            distance = current_distance + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                previous_nodes[neighbor] = current_vertex
                heapq.heappush(priority_queue, (distance, neighbor))

    return distances, previous_nodes

def reconstruct_path(start, end, previous_nodes):
    path = []
    current_vertex = end

    while current_vertex is not None:
        path.append(current_vertex)
        current_vertex = previous_nodes[current_vertex]

    path.reverse()

    return path if path[0] ==start else[]
```

Initialization:

- The distances dictionary is initialized to infinity for all vertices except the starting vertex, which is set to 0.
- The previous_nodes dictionary keeps track of the shortest path tree.
- The priority_queue is initialized with the starting vertex.

Main Loop:

- The algorithm continues to explore the graph while there are vertices in the priority queue.
- For each vertex, it checks all its neighbors and updates their distances if a shorter path is found. The updated distance and vertex are then pushed onto the priority queue.

Path Reconstruction:

- The reconstruct_path function traces back from the destination to the start vertex using the previous_nodes dictionary, building the shortest path.

Analysis of Algorithms

The analysis of our algorithm focuses on understanding its time complexity in different scenarios: best case, worst case, and average case. This analysis is crucial for determining the efficiency and performance of our pathfinding solution using Dijkstra's Algorithm.

Best Case:

Scenario: The start and end nodes are connected with minimal intermediate nodes and edges.

Explanation: In the best-case scenario, the algorithm quickly finds the shortest path with the least number of intermediate nodes and edges.

Time Complexity: O((V + E) log V), where V is the number of vertices and E is the number of edges.

Practical Example:

- **Nodes**: Faculty of Computer Science to Univ 360
- **Path**: Faculty of Computer Science -> Faculty of Engineering -> Shell -> Univ 360
- **Distance**: Minimal intermediate nodes and edges

In this case, the algorithm finds the shortest path with fewer nodes and edges, reducing processing time.

Worst Case

Scenario: The graph is dense, and the algorithm has to explore all nodes and edges to find the shortest path.

Explanation: In the worst-case scenario, every node is connected to many other nodes, requiring the algorithm to process all possible edges to find the shortest path. This typically happens in a densely connected graph where each node has multiple connections.

Time Complexity:

- O(V^2) for an adjacency matrix representation.
- O((V + E) log V) for an adjacency list representation using a priority queue.

Practical Example:

- **Nodes**: Faculty of Computer Science to IOI Resort

- **Path**: A complex path through multiple nodes and connections.
- **Distance**: The total distance considering all connections.

In this scenario, the algorithm needs to evaluate many possible paths and connections, resulting in maximum computational effort.

Average Case

Scenario: The graph has a moderate number of nodes and edges with balanced connections.

Explanation: In the average case, the algorithm processes nodes and edges in a balanced manner. The graph is neither too sparse nor too dense, representing a typical scenario for route finding.

Time Complexity: O((V + E) log V) for an adjacency list with a priority queue.

Practical Example:

- **Nodes**: Faculty of Computer Science to Tropical Villa
- **Path**: Faculty of Computer Science -> MRT UPM -> PETRONAS -> Tropical Villa
- **Distance**: The total distance considering moderate connections.

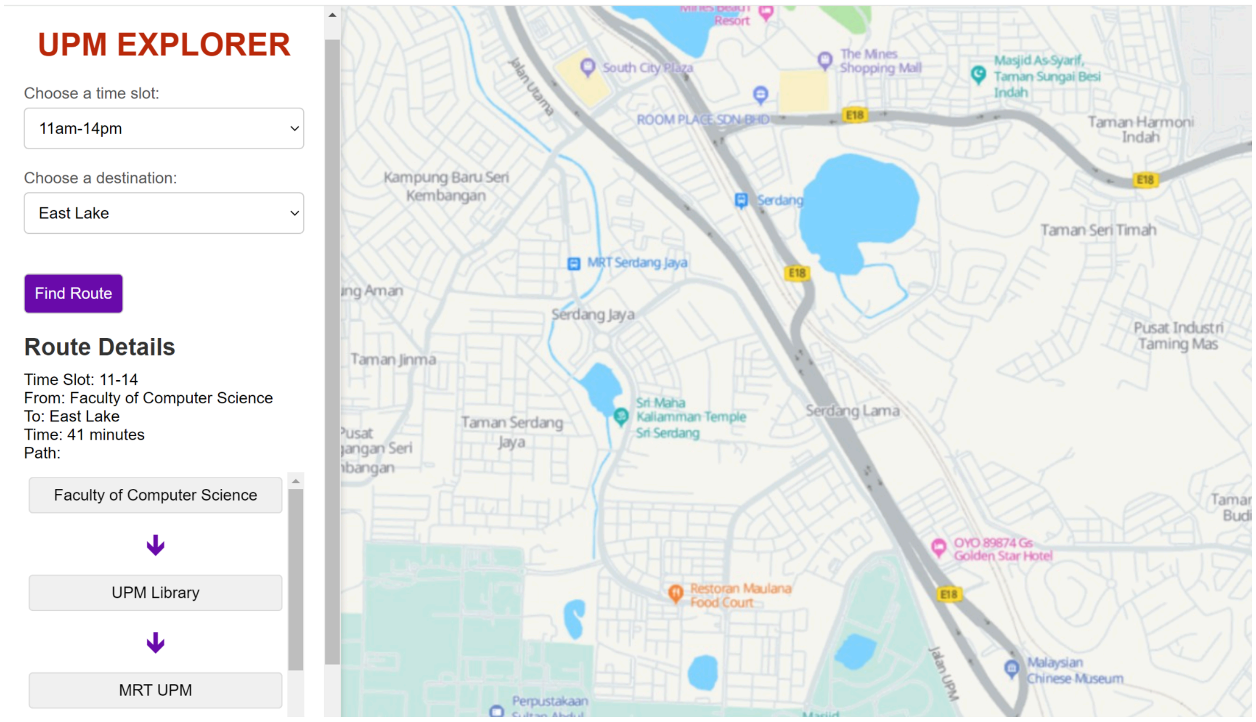
In this case, the algorithm processes a balanced number of nodes and edges to determine the shortest path.

Practical Steps for Analysis

1. **Implement the Algorithm:** Ensure Dijkstra's Algorithm is correctly implemented and capable of processing the graph structure representing the campus map.
2. **Determine Time Complexities:** Analyze the time complexities for the best, worst, and average cases based on the graph structure and the algorithm's operations.
3. **Measure Execution Time:** Use a timer to measure the execution time of your algorithm for different input sizes and scenarios. Compare the results to the theoretical time complexities.
4. **Generate Graphs:** Create visual representations of the campus map and analyze how the algorithm performs on these graphs in different scenarios.
5. **Print and Compare Paths:** Printing all possible paths is not necessary. Instead, verify the correctness and efficiency of the algorithm by comparing the paths found in different scenarios.

APP And User Interface:

In this project we also created a simple UI to test and run the algorithm smoothly by using the Flask application in python below is the output of our application:



Route Details

Time Slot: 11-14
From: Faculty of Computer Science
To: East Lake
Time: 41 minutes
Path:

Faculty of Computer Science

UPM Library

MRT UPM

Conclusion

The UPM Explorer project successfully implemented an algorithm to find the optimal route for UPM students considering varying traffic conditions. The results demonstrate the effectiveness of the algorithm in providing reliable route recommendations.

This tool can significantly enhance the commuting experience for students by minimizing travel time and improving navigation.

For future work could include integrating real-time traffic data using APIs to provide dynamic route updates. Additionally, implementing a mobile application version of the tool could further enhance accessibility and user experience.