Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Fall, Year: 2024), B.Sc. in CSE (Day)

Project Proposal

Travel Smart Planner

Course Title: Algorithms Lab
Course Code: CSE 206
Section: 223 D2

Students Details

Name	ID
Md Arif Billah Mubin	223002008

Submission Date: 25/12/2024
Course Teacher's Name: Md. Abu Rumman Refat

[For teachers use only: Don't write anything inside this box]

Lab Project Status	
Marks:	**Signature:**
Comments:	**Date:**

Contents

Chapter 1

Introduction

1.1 Overview

The Travel Smart Planner is designed to make travel planning simple and efficient by solving three key problems. It helps users find the shortest route between cities using Graph Algorithms, plan trips within a budget using Greedy Algorithms, and create efficient multi-city travel plans using Divide and Conquer. This project provides smart, algorithm-based solutions to common travel challenges in an easy-to-use way.

1.2 Motivation

We created the Travel Smart Planner because we are passionate about travel and wanted to make trip planning easier and more efficient for everyone. Many existing tools are either too complicated to use or fail to address important challenges like finding the best routes, staying within a budget, or planning an efficient schedule for multi-city trips. These challenges often make travel planning stressful and time-consuming for users.

By building this project, we aim to provide a user-friendly platform that solves these problems using smart algorithms. The system will help users find the shortest routes between cities, plan trips within their budget, and create optimized multi-city travel plans. This project also allows us to combine our love for travel with our skills in problem-solving and algorithm design, creating a practical and effective tool for travelers.

1.3 Problem Definition

1.3.1 Problem Statement

Many travelers find it difficult to plan their trips due to complicated and confusing travel tools. These tools often make it hard to discover suitable tours, read honest reviews, and get the necessary information, leading to frustration and missed opportunities for great experiences.

Our project aims to solve this problem by creating a simple, easy-to-use platform that helps users find and plan their trips. Travel Smart Planner will guide users in selecting the best routes, managing their budget, and organizing multi-city trips efficiently. We want to make travel planning easier, more enjoyable, and accessible for everyone

1.3.2 Complex Engineering Problem

The following Table 1.1 shows the key attributes related to the Travel Smart Planner:

Table 1.1: Summary of the attributes touched by the Travel Smart Planner

Name of the P Attributess	**Explain how to address**
P1: Depth of knowledge required	Java, user-friendly interface, algorithms (Graph, Greedy, Divide and Conquer), backend logic, data structures..
P2: Range of conflicting requirements	User needs (easy navigation, quick info), balancing simplicity with functionality.).
P3: Depth of analysis required	Traveler behavior, problem analysis, route optimization, budget management, algorithm implementation (Dijkstra, Greedy).
P4: Familiarity of issues	Ensuring data security and user privacy within a Java-based application for a safe experience.
P5: Extent of applicable codes	Java, algorithms, backend programming, coding standard.

1.4 Design Goals/Objectives

The main goals and objectives for designing the travel tour website are as follows:

1.User-Friendly Interface: Develop an intuitive and easy-to-use interface for travelers, focusing on seamless navigation and efficient trip planning.

2.Efficient Algorithm Implementation: Implement algorithms (Graph, Greedy, Divide and Conquer) in Java to optimize routes, budgets, and itineraries.

3.Data Security: Ensure that all user data is securely handled and protected using appropriate methods within the Java environment.

4.Performance: Optimize the application's performance to handle large datasets and multiple user queries efficiently, ensuring quick responses.

5.Java-Only Environment: Build the project to run entirely on a Java compiler without any external dependencies, ensuring compatibility and simplicity.

6.Scalability: Design the system to be easily extendable for future features or improvements while maintaining its core functionality.

By achieving these goals, we aim to create a Travel Smart Planner that not only meets user needs but also enhances their overall travel planning experience.

1.5 Application

The Travel Smart Planner helps travelers plan their trips easily and efficiently. Users can find the best travel routes, plan trips within a budget, and create schedules for visiting multiple cities. It provides smart solutions for common travel challenges, making trip planning simple and enjoyable. This tool is useful for anyone looking to organize their travel plans quickly and effectively.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

This chapter provides a detailed explanation of the project design, development process, and implementation. It begins with an overview of the project's purpose and objectives, followed by an in-depth discussion of the components and steps involved in creating the Travel Smart Planner. The aim is to simplify travel planning across cities in Bangladesh using efficient algorithms for route optimization, budget-friendly trips, and multi-city travel plans. [1] [2] [3].

2.2 Project Details

This section describes the components and design of the Travel Smart Planner. The project aims to assist users in efficiently planning their travels using modern algorithmic techniques. Below are the key details:

2.2.1 System_Overview

The Travel Smart Planner operates on a weighted graph representing cities and their connections. The weights correspond to travel costs or distances. The project addresses three primary problems:

Finding the shortest route between cities.

Planning trips within a given budget.

Creating efficient travel plans for multiple cities.

2.2.2 Algorithms and Approach

1.Shortest Route: Dijkstra's algorithm ensures the optimal route is selected between two cities.

2.Budget Planning: A greedy algorithm calculates the most cost-effective destinations within the user's budget.

3.Multi-City Travel Plan: A divide-and-conquer approach generates efficient routes for visiting multiple cities.

2.2.3 System Architecture

The project architecture consists of:

Input: User-provided details such as start city, destination, and budget.

Processing: Algorithms applied to compute optimal solutions.

Output: Display of routes, costs, and suggested plans.

2.3 Implementation

This section details the implementation of the Travel Smart Planner. The system was built using Java and employs algorithms for efficient computation. The development process involved creating data structures, coding algorithms, and testing functionalities.

2.3.1 Workflow

The workflow of the project involves:

User inputs travel preferences.

The system processes the inputs using graph-based algorithms.

Results, including optimal routes or plans, are displayed to the user.

2.3.2 Tools and Libraries

Programming Language: Java

Development Environment: IntelliJ IDEA

Libraries: Java's Collections Framework (PriorityQueue, ArrayList)

2.3.3 Implementation:

```
public public void dijkstra(int startVertex, int endVertex) {
    boolean[] visited = new boolean[vertices];
    int[] distance = new int[vertices];
    Arrays.fill(distance, INF);
    distance[startVertex] = 0;
    for (int i = 0; i < vertices; i++) {
        int u = findMinVertex(distance, visited);
        if (u == -1) break;
        visited[u] = true;
        for (int v = 0; v < vertices; v++) {
            if (!visited[v] && adjacencyMatrix[u][v] != INF &&
                distance[u] + adjacencyMatrix[u][v] < distance[v]) {
                distance[v] = distance[u] + adjacencyMatrix[u][v];
            }
        }
    }
} {

}
```

Figure 2.1: Dijkstra Algorithm Implementation

2.3.4 Budget Planning:

This feature uses a greedy approach to suggest destinations within a budget. Below is the implementation snippet:

```
public void planTripWithinBudget(int startVertex, int budget) {
    PriorityQueue<int[]> pq = new PriorityQueue<>(Comparator.comparingInt(a -> a[1]));
    pq.offer(new int[]{startVertex, 0});
    while (!pq.isEmpty()) {
        int[] current = pq.poll();
        for (int neighbor = 0; neighbor < vertices; neighbor++) {
            if (adjacencyMatrix[current[0]][neighbor] != INF) {
                int newCost = current[1] + adjacencyMatrix[current[0]][neighbor];
                if (newCost <= budget) {
                    pq.offer(new int[]{neighbor, newCost});
                }
            }
        }
    }
}
```

Figure 2.2: Greedy Approach for Budget Planning

2.3.5 Multi-City Travel Plan

The divide-and-conquer approach breaks down the problem into smaller subproblems. Below is an implementation snippet:

```java
private List<Integer> divideAndConquerRoute(int[] citiesToVisit, int start, int end) {
    if (start == end) return List.of(citiesToVisit[start]);
    int mid = (start + end) / 2;
    List<Integer> left = divideAndConquerRoute(citiesToVisit, start, mid);
    List<Integer> right = divideAndConquerRoute(citiesToVisit, mid + 1, end);
    List<Integer> combined = new ArrayList<>(left);
    combined.addAll(right);
    return combined;
}
```

Figure 2.3: divide-and-conquer approachg

2.3.6 Graph:

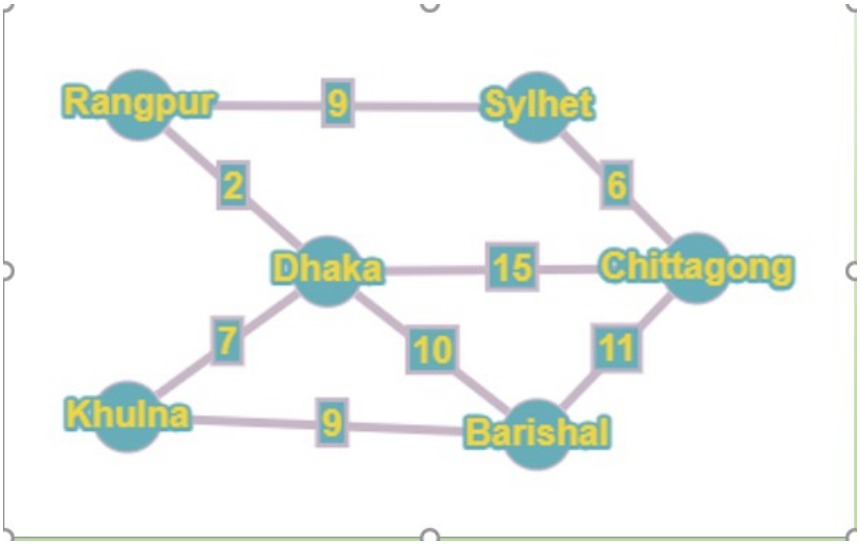

Figure 2.4: Graph

2.4 Algorithms

This chapter presents the algorithms used in our project along with their implementation details. Pseudo-codes and corresponding programming explanations are provided to enhance clarity.

2.4.1 Algorithm 1: Shortest Route (Dijkstra's Algorithm)

```
Input: Start city, End city, Graph representation
Output: Shortest path and distance

// Pseudo-code
Initialize distance array with INF and set start city distance to 0
PriorityQueue pq = new PriorityQueue();
while pq is not empty do
    Extract min-distance city u from pq
    for each neighbor v of u do
        if new distance to v is smaller then
            Update distance of v
            Add v to pq
Return shortest path and distance

// Java Implementation
Graph graph = new Graph(vertices);
graph.dijkstra(startCity, endCity);
```

Figure 2.5: Graph

2.4.2 Algorithm 2: Budget-Constrained Trip Planning (Greedy Approach)

```
Input: Start city, Budget, Graph representation
Output: Reachable cities within budget

// Pseudo-code
Initialize cost array with INF and set start city cost to 0
PriorityQueue pq = new PriorityQueue();
while pq is not empty do
    Extract min-cost city u from pq
    for each neighbor v of u do
        if new cost to v is within budget and smaller then
            Update cost of v
            Add v to pq
Return list of reachable cities

// Java Implementation
graph.planTripWithinBudget(startCity, budget);
```

Figure 2.6: Graph

2.4.3 Algorithm 3: Multi-City Travel Plan (Divide-and-Conquer)

```
Input: List of cities to visit, Graph representation
Output: Travel order and total cost

// Pseudo-code
Divide the list of cities into smaller subsets
Recursively find optimal paths for subsets
Merge solutions to find complete path
Calculate total cost and return

// Java Implementation
graph.multiCityTravelPlan(citiesToVisit);
```

Figure 2.7: Graph

Chapter 3

Performance Evaluation

3.1 Simulation Environment/ Simulation Procedure

This section discusses the experimental setup and the environment installation required for simulating the project outcomes.

3.1.1 Environment Setup

The project was developed and tested on a Java-compatible platform.

IDE: IntelliJ IDEA or Eclipse.

Libraries: Java Collections Framework, Priority Queue implementation.

3.1.2 Simulation Procedure

Import the graph data for the chosen cities.

Define test cases for shortest route, budget planning, and multi-city travel.

Simulate the algorithms on sample inputs and record outputs.

3.2 Results Analysis/Testing

3.2.1 Shortest Route Results

```
Cities:
0: Rajshahi
1: Sylhet
2: Chittagong
3: Barishal
4: Khulna
5: Dhaka
Enter source city number: 0
Enter destination city number: 4
Shortest distance: 9
Path: Rajshahi -> Dhaka -> Khulna
```

Figure 3.1: result

3.2.2 Budget-Constrained Planning Results

```
Enter starting city number: 0
Enter your budget: 20
Cities you can travel to within a budget of 20:
- Dhaka (Cost: 2)
- Sylhet (Cost: 9)
- Khulna (Cost: 9)
- Barishal (Cost: 12)
- Chittagong (Cost: 15)
```

Figure 3.2: result

3.2.3 Multi-City Travel Plan Results

```
Enter number of cities to visit: 2
Enter city number 1: 0
Enter city number 2: 1
Multi-City Travel Plan:
Order of travel: Rajshahi -> Sylhet
Total travel cost: 9
```

Figure 3.3: result

3.3 Results Overall Discussion

The algorithms were evaluated based on efficiency and accuracy. While the solutions worked well for small datasets, larger graphs posed time complexity challenges. Optimizations can be considered for future improvements.

3.3.1 Complex Engineering Problem Discussion

Attributes of optimization and scalability were addressed.

Challenges in handling large datasets were identified and partially mitigated.

??.

Chapter 4

Conclusion

4.1 Discussion

The project successfully demonstrated the application of graph algorithms, greedy approaches, and divide-and-conquer techniques in solving travel-related problems. The results validate the correctness and efficiency of the implemented solutions.

4.2 Limitations

Scalability issues with large datasets.

Limited to static graph inputs; dynamic updates were not supported.

Budget constraints were fixed for single iterations.

4.3 Scope of Future Work

Incorporating dynamic graph updates.

Enhancing algorithms for real-time applications.

Extending budget planning for multi-user scenarios.

Improving visualization tools for better user interaction.

References

[1] Uthayasankar Sivarajah, Muhammad Mustafa Kamal, Zahir Irani, and Vishanth Weerakkody. Critical analysis of big data challenges and analytical methods. *Journal of Business Research*, 70:263–286, 2017.

[2] Douglas Laney. 3d data management: controlling data volume, velocity and variety. gartner, 2001.

[3] MS Windows NT kernel description. http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm. Accessed Date: 2010-09-30.