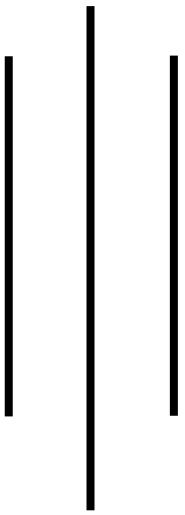
National College of Computer Studies

Paknajol, Kathmandu



Report on Tour Planner

Submitted by:

Ayush Tuladhar

BSc.CSIT 3rd Sem

Section: A

Submitted to:

Er. Khushbu K. Sarraf

Submission Date: 18th April 2025

Table of Contents

Abstract i

Chapter 1: Introduction 1

1.1 Background and Objectives 1

1.2 Problem Statement 1

Chapter 2: Project Description 2

2.1 Overview 2

2.2 Features 2

Chapter 3: Algorithm 4

Algorithm for Tour Planner program: 4

Chapter 4: Methodology 6

4.1 Development Approach 6

4.2 Requirements Analysis 6

4.3 System Design 6

4.4 Testing 7

4.5 Deployment 8

Chapter 5: Results and Discussion 9

5.1 Results 9

5.2 Discussion 9

Chapter 6: Conclusion 10

References 11

Source Code 12

Output 27

Abstract

This project presents a comprehensive C++ application titled Tour Planner, designed to assist travelers in organizing their trips with enhanced budget management, route optimization, and destination planning. The program integrates a modular structure encompassing key entities such as TravellerProfile, Destination, Attraction, Transportation, and a graph-based RouteGraph system. Users can input and manage travel profiles, add destinations with associated attractions and costs, and define transportation routes between locations. Advanced features include the ability to save and load data from files, visualize all possible paths between destinations, and determine the most cost-effective or time-efficient travel routes using Dijkstra's algorithm. Through a menu-driven interface with Windows console manipulation (e.g., gotoxy, SetConsoleCursorPosition), the program delivers an interactive user experience. The Tour Planner provides a useful way to create travel itineraries by finding a balance between algorithmic accuracy and convenience for the best possible trip planning.

Chapter 1: Introduction

1.1 Background and Objectives

In today's fast-paced world, efficient travel planning is essential for individuals aiming to make the most of their time and budget. Whether for leisure or business, travelers often face challenges in organizing destinations, estimating costs, optimizing routes, and balancing their plans with financial constraints. Manual trip planning can be both time-consuming and error-prone, especially when trying to account for multiple variables like transportation modes, attraction schedules, and budget limits.

To address these challenges, this project introduces an interactive Tour Planner application that simulates a real-world travel planning assistant. Developed in C++, the program leverages object-oriented principles and graph-based data structures to provide users with a structured, intuitive, and automated approach to organizing their journeys. It integrates destination management, budget analysis, and route optimization into a cohesive system.

The main objective of this project is to implement a comprehensive tour planning system that allows users to create traveler profiles, manage destinations and attractions, and analyze travel routes for cost and time efficiency. Through this application, users can explore the practical applications of graph algorithms like Dijkstra’s for finding optimal paths, and understand the significance of modular design, file handling, and user interaction in building real-world systems. This project also serves to deepen knowledge of C++ concepts such as class inheritance, dynamic memory management, and persistent storage.

1.2 Problem Statement

Planning a multi-destination trip manually can be time-consuming and error-prone, especially when balancing factors like budget, travel time, and route efficiency. Travelers often lack tools that integrate all aspects of trip planning—such as destination management, transportation options, and budget analysis—into a single, interactive platform. This project addresses the need for a streamlined solution by developing a C++ based Tour Planner application that enables users to organize travel profiles, manage destinations and attractions, analyze routes, and optimize travel plans using algorithmic techniques like Dijkstra’s algorithm for cost and time efficiency.

Chapter 2: Project Description

2.1 Overview

Using C++, this project implements a comprehensive Tour Planner system that assists users in organizing multi-destination travel plans while managing time and budget constraints effectively. The application integrates advanced data structures such as arrays, classes, and graph representations to provide functionalities like profile management, destination tracking, attraction management, transportation routing, and budget analysis. The system allows users to create traveler profiles, add and view destinations with detailed attractions, define transportation routes, and analyze travel plans for cost or time optimization. Dijkstra's algorithm is utilized within a graph-based structure to determine the shortest or most cost-effective travel paths, offering a real-world application of graph theory. Additionally, persistent file storage is supported, enabling users to save and load travel data for reuse. This project demonstrates how modular design, and algorithmic logic can work together to provide an intelligent and user-friendly travel planning experience.

2.2 Features

The Tour Planner project includes the following key features:

1) **Traveller Profile Management**
    a) Users can create a personal profile including name, age, budget, and planned travel days.
    b) Profiles are saved to a file and can be viewed or loaded in future sessions.
2) **Destination and Attraction Handling**
    a) Users can add destinations with details such as stay duration and associated attractions.
    b) Each attraction includes a name, cost, and time required, stored dynamically using object-oriented techniques.
    c) Destinations can be displayed with a total estimated cost for better budget tracking.
3) **Route Management (Graph-Based Implementation)**
    a) Routes between destinations are defined with transport type, cost, and duration.
    b) A directed graph is used to model the travel network, storing transportation data in adjacency matrices.
    c) Routes can be displayed for review and stored in a file for persistence.

4) **Route Optimization and Pathfinding (Dijkstra's Algorithm)**
    a) Users can find the most cost-efficient or time-saving path between two locations.
    b) The system implements Dijkstra's algorithm to compute the shortest paths within the travel graph.
    c) Path details, including costs, durations, and transport types, are displayed for user reference.
5) **Budget Analysis and Cost Planning**
    a) The application compares the estimated travel cost with the user's available budget.
    b) Users are notified of any surplus or deficit, helping them make informed travel decisions.
    c) All possible paths between two destinations can also be explored to assess alternatives.
6) **Data Persistence (File Handling)**
    a) Traveler profiles, destinations, and route data can be saved to and loaded from external files.
    b) This enables long-term planning and easy access to previous trip data.

This Tour Planner system is designed to simplify and enhance the travel planning process by integrating algorithmic optimization with user-friendly features. It provides a practical application of C++ programming concepts and demonstrates the use of object-oriented design and graph algorithms in real-world scenarios.

Chapter 3: Algorithm

Algorithm for Tour Planner program:

Step 1: Start the program and initialize the TripPlanner object.

Step 2: Display the main menu with the following options:

i) Traveller Profile
ii) Destination
iii) Route
iv) Budget Analysis
v) Exit

Step 3: Prompt the user to enter their menu choice.

Step 4: If the user selects Traveller Profile:

i) Show sub-options to either set up or view the profile.
ii) If setting up, ask for name, age, budget and planned travel days then save to a file.
iii) If viewing, load and display saved profile details.

Step 5: If the user selects Destination:

i) Show sub-options: Add, View, Save or Load Destinations.
ii) To add a destination, prompt for the name, stay duration and one or more attractions(name,cost,time).
iii) Save destinations and attractions in dynamic memory.
iv) Allow saving or loading all destinations data from a file.

Step 6: If the user selects Route:

i) Show sub-options: Add, View, Save or Load Routes.
ii) For adding a route, ask for source, destination, transport type, cost and duration.
iii) Store route information in a graph structure using adjacency matrices.
iv) Allow saving and loading of all route data from a file.

Step 7: If the user selects Budget Analysis:

i) Show sub-options: Best Path or All Paths.

ii) For Best Path, prompt for source and destination and optimizations preferences(cost or time)
iii) Apply Dijkstra's algorithm to find and display the optimal route.
iv) For All Paths, use DFS to display every possible path along with cost and time breakdown.

Step 8: After completing an operation, ask if the user wants to return it to the main menu.

Step 9: Repeat Steps 2 to 8 until the user selects Exit.

Step 10: On exit, display a closing message and terminate the program.

Step 11: Stop.

Chapter 4: Methodology

4.1 Development Approach

The Tour Planner project is developed using a modular and incremental approach. Each component—traveler profile, destination handling, transportation routing, budget analysis, and pathfinding—is designed and implemented as an independent module before being integrated into the complete system. The application is built entirely in C++, employing object-oriented programming and algorithmic principles. Key structures like classes and adjacency matrices are used to represent travel data, while Dijkstra's and DFS algorithms provide optimization and analysis capabilities. This methodology ensures clarity, maintainability, and scalability throughout development.

4.2 Requirements Analysis

Functional Requirements:

- Create and manage traveler profiles (name, age, budget, travel days).
- Add, view, save, and load destinations with attractions.
- Define and manage transportation routes between destinations.
- Analyze routes to determine optimal travel paths based on cost or time.
- Provide visual feedback on budget surplus or deficit.
- Display all possible paths and their details between selected locations.
- Save and retrieve travel data using file operations.

4.3 System Design

- Traveller Profile Module:

Stores traveler details and allows persistence through file I/O operations. Used to personalized and guide travel planning decisions.

- Destination and Attraction Module:

  Each destination is represented by a class containing attraction objects. Attractions include cost and time estimates. Destinations are stored in an array and linked to the travel graph.

- Route Graph (Adjacency Matrix):

  Implements a directed graph structure where nodes represent destinations and edges represent travel routes with attributes like cost, time and transport type.

- Dijkstra's Algorithm:

  Used to compute the shortest or least expensive route between 2 locations, depending on user preference.

- DFS(Depth-First Search):

  Used to list all possible travel paths between two destinations, showing detailed breakdowns for analysis.

- File Handling:

  Supports reading from and writing to binary files to save/load traveler profiles, destination lists, and route graphs.

- Console Interface (Windows API):

Utilizes functions like gotoxy() and SetConsoleCursorPosition() to create and organized, interactive menu-driven console UI.

4.4 Testing

Integration Testing:

- Unit Testing:

  Each module—such as profile management, route graph, destination handling—is tested separately to validate functionality.

- Integration Testing:

  After module testing, components are combined to ensure consistent data flow and expected behavior across the system.

- Functional Testing:

Tests include adding multiple destinations and routes, finding optimal paths, and verifying that budget analysis and file persistence work correctly.

- Boundary Testing:

Edge cases such as exceeding attraction limits, entering invalid input, or searching for non-existent locations are tested for proper handling.

4.5 Deployment

The final application is a console-based C++ program designed to run in a Windows environment.

Deployment steps include:

- Compiling the source code using a standard C++ compiler (e.g., g++, Visual Studio).
- Running the compiled executable in Command Prompt or a terminal emulator.
- The system can be extended for cross-platform compatibility or adapted into a GUI-based or web-enabled travel assistant in future iterations.

Chapter 5: Results and Discussion

5.1 Results

- Successfully created and stored traveler profiles using file I/O.
- Added and displayed multiple destinations, each with detailed attractions.
- Defined transportation routes and stored them in a graph structure.
- Applied Dijkstra's algorithm to find optimal travel paths.
- Displayed all possible paths using DFS, with cost and time breakdowns.
- Provided budget analysis, highlighting surplus or deficit conditions.
- Enabled saving and loading of complete trip plans (destinations, routes, profile).

5.2 Discussion

The Tour Planner project achieved its objectives by integrating user profile management, destination tracking, route optimization, and budget analysis into a unified, interactive application. The use of object-oriented programming allowed modular and reusable code. Graph algorithms such as Dijkstra's and DFS were implemented successfully to handle pathfinding and route evaluations. File operations ensured that travel data could be preserved across sessions. The system offers an intuitive, structured interface while demonstrating real-world applications of core C++ programming concepts. Future versions could benefit from features like graphical visualization of maps, real-time API integration for live data, or mobile compatibility.

Chapter 6: Conclusion

In conclusion, the Tour Planner system demonstrates a practical and effective application of C++ for organizing travel plans. By combining structured data handling, graph algorithms, and file management, the project allows users to manage profiles, add destinations and attractions, define routes, and analyze trip feasibility based on cost and time. The use of Dijkstra's algorithm ensures route efficiency, while DFS provides a broader view of travel options. Through modular development and testing, the system proves to be reliable, user-friendly, and extensible. With potential for future enhancements such as GUI integration, real-time travel APIs, and mobile deployment, this project lays a strong foundation for more advanced trip planning applications.

References

- GeeksforGeeks. (2023, November 2). How to use gotoxy() in codeblocks?
- https://www.geeksforgeeks.org/how-to-use-gotoxy-in-codeblocks/
- GeeksforGeeks. (2025, April 11). How to find Shortest Paths from Source to all Vertices using Dijkstra's Algorithm
- https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
- GeeksforGeeks. (2025, March 29). Depth First Search or DFS for a Graph
- https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

Source Code

```
#include<iostream>
#include<iomanip>
#include<fstream>
#include<string.h>
#include<windows.h>
using namespace std;
#define MAX_DESTINATIONS 20
#define MAX_ATTRACTIONS 5
#define MAX_TRANSPORTATION 3
#define INF 999999.0
int *posx, *posy;
void dash() {
   CONSOLE_SCREEN_BUFFER_INFO csbi;
   int width;
   if(!GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE),&csbi))
      cout<<"Error getting console screen buffer info.";
   else
   {
            width=csbi.dwSize.X;
               cout<<endl;
               for(int i=0;i<width;i++)
                  cout<<"-";
               cout<<endl<<endl;
               fflush(stdout);
            }
}
void gotoxy(int x, int y) {
   HANDLE hConsole=GetStdHandle(STD_OUTPUT_HANDLE);
   if(hConsole == INVALID_HANDLE_VALUE)
      cout<<"Error getting console handle.";
   else
   {
            COORD cursorPos;
               cursorPos.X=x;
               cursorPos.Y=y;
               if(!SetConsoleCursorPosition(hConsole,cursorPos))
                  cout<<"Error setting console cursor position.";
            }
}
void drawHeader(const char* title,int& x,int& y) {
   system("cls");
            gotoxy(x,y);
   dash();
   gotoxy(x,y+=3);
   cout << title;
   gotoxy(x, ++y);
   dash();
}
template<typename T>
bool saveToFile(const T& data, const char* filename) {
   ofstream file(filename,ios::binary);
   if(!file.is_open()) {
      cout<<"Error opening file for writing: "<<filename<<endl;
      return false;
   }
   file.write(reinterpret_cast<const char*>(&data),sizeof(T));
   file.close();
   return true;
}
template<typename T>
bool loadFromFile(T& data, const char* filename) {
   ifstream file(filename,ios::binary);
   gotoxy(*posx,*posy);
   if(!file.is_open()) {
      cout<<"Error opening file for reading: "<<filename<<endl;
      return false;
   }
   file.read(reinterpret_cast<char*>(&data),sizeof(T));
   file.close();
   return true;
}
```

```
enum TransportType {
   AIR,
   BUS,
   CAR,
   OTHER
};
class Attraction {
        protected:
           char name[50];
           double cost;
           int timeRequired;
        public:
           Attraction() {
              strcpy(name,"N/A");
              cost=0;
              timeRequired=0;
           }
           Attraction(const char* n,double c,int time) {
              strncpy(name,n,sizeof(name)-1);
              name[sizeof(name)-1]='\0';
              cost=c;
              timeRequired=time;
           }
           double getCost() const { return cost; }
           int getTimeRequired() const { return timeRequired; }
           const char* getName() const { return name; }
           void updateAttraction(const char* newName,double newCost,int newTime) {
              strncpy(name,newName,sizeof(name)-1);
              name[sizeof(name)-1]='\0';
              cost=newCost;
              timeRequired=newTime;
           }
           virtual ~Attraction() {}
};
class Transportation {
        private:
           TransportType type;
           char from[50],to[50];
           double cost;
           int duration;
        public:
           Transportation() {
              type=OTHER;
              strcpy(from,"N/A");
              strcpy(to,"N/A");
              cost=0;
              duration=0;
           }
           Transportation(TransportType t,const char* f,const char* t_dest,double c,int d) {
              type=t;
              strncpy(from,f,sizeof(from)-1);
              from[sizeof(from)-1]='\0';
              strncpy(to,t_dest,sizeof(to)-1);
              to[sizeof(to)-1]='\0';
              cost=c;
              duration=d;
           }
           const char* getTypeString() const {
              switch(type) {
                 case AIR: return "Air";
                 case BUS: return "Bus";
                 case CAR: return "Car";
                 case OTHER: return "Other";
                 default: return "Unknown";
              }
           }
           double getCost() const { return cost; }
           int getDuration() const { return duration; }
           TransportType getType() const { return type; }
           const char* getFrom() const { return from; }
           const char* getTo() const { return to; }
};
class Destination {
        private:
           char name[50];
```

```
    int stayDuration,attractionCount,x,y;
    Attraction* attractions[MAX_ATTRACTIONS];
public:
    Destination() {
        strcpy(name,"N/A");
        stayDuration=0;
        attractionCount=0;
        for(int i=0;i<MAX_ATTRACTIONS;i++)
                                        attractions[i]=NULL;
    }
    Destination& operator=(const Destination& other) {
                if(this!=&other) {
                    strncpy(name,other.name,sizeof(name)-1);
                    name[sizeof(name)-1]='\0';
                    stayDuration=other.stayDuration;
                    for(int i=0;i<attractionCount;++i) {
                        delete attractions[i];
                        attractions[i]=NULL;
                    }
                    attractionCount=other.attractionCount;
                    for(int i=0;i<MAX_ATTRACTIONS;++i)
                        if(i< attractionCount && other.attractions[i])
                            attractions[i]=new Attraction(*other.attractions[i]);
                        else
                            attractions[i]=NULL;
                }
                return *this;
            }
    ~Destination() {
        for(int i=0;i<attractionCount;i++)
                                        delete attractions[i];
    }
    Attraction* getAttraction(int i) const {
                if (i>=0 && i<attractionCount)
                                        return attractions[i];
                return NULL;
            }
            void setName(const char* n) {
        strncpy(name,n,sizeof(name)-1);
        name[sizeof(name)-1]='\0';
    }
    void setStayDuration(int days) {
        stayDuration=days;
            }
    void addAttraction(Attraction* attraction) {
        if(attractionCount < MAX_ATTRACTIONS)
            attractions[attractionCount++]=attraction;
        else
            cout << "Maximum attractions reached for this destination.";
    }
    double calculateTotalCost() {
        double totalCost=0;
        for(int i=0;i<attractionCount;i++)
            totalCost+=attractions[i]->getCost();
        return totalCost;
    }
    void displayDestination() {
                x=10;
                            y=*posy+1;
        gotoxy(x,y+=2);
        cout<<"Destination : "<<name;
        gotoxy(x,++y);
        cout<<"Duration of Stay : "<<stayDuration<<" days";
        gotoxy(x,y+=2);
        cout<<"Attractions :";
        if(attractionCount>0)
            for(int i=0;i<attractionCount;i++) {
                Attraction* a=attractions[i];
                if(a) {
                                                gotoxy(x+5,++y);
                                                cout<<i+1<<". Attraction : "<< a->getName();
                                                gotoxy(x+12,++y);
                                                cout<<"Cost : Rs "<< a->getCost();
                                                gotoxy(x+12,++y);
                                                cout<<"Time Required : " <<a->getTimeRequired()<<" hrs";
```

```
                                        }
                    }
                                else
                cout<<"No attractions added yet.";
                gotoxy(x,y+=2);
                cout<<"Total Estimated Cost : Rs "<<calculateTotalCost();
                gotoxy(x,++y);
                                dash();
                                (*posy)=y;
            }
            const char* getName() const { return name; }
            int getStayDuration() const { return stayDuration; }
            int getAttractionCount() const { return attractionCount; }
};
struct AttractionData {
    char name[50];
    double cost;
    int timeRequired;
};
struct DestinationRawData {
    char name[50];
    int stayDuration,attractionCount;
    AttractionData attractions[MAX_ATTRACTIONS];
};
struct DestinationData {
    int count;
    DestinationRawData data[MAX_DESTINATIONS];
};
class Route {
        private:
            char from[50];
            char to[50];
            double cost;
            int time,x,y;
            TransportType transportType;
        public:
            Route() {
                strcpy(from,"");
                strcpy(to,"");
                cost=0;
                time=0;
                transportType=OTHER;
            }
            Route(const char* f,const char* t,double c,int tm,TransportType type) {
                strncpy(from,f,sizeof(from)-1);
                from[sizeof(from)-1]='\0';
                strncpy(to,t,sizeof(to)-1);
                to[sizeof(to)-1]='\0';
                cost=c;
                time=tm;
                transportType=type;
            }
            const char* getFrom() const { return from; }
            const char* getTo() const { return to; }
            double getCost() const { return cost; }
            int getTime() const { return time; }
            TransportType getTransportType() const { return transportType; }
            const char* getTypeString() const {
                switch(transportType) {
                    case AIR: return "Air";
                    case BUS: return "Bus";
                    case CAR: return "Car";
                    case OTHER: return "Other";
                    default: return "Unknown";
                }
            }
            void displayRoute() {
                    x=10;
                                y=*posy+1;
                gotoxy(x,y+=2);
                cout<<" Route : "<<from<<" -> "<<to;
                gotoxy(x+3,++y);
                cout<<"\tTransport Type : "<<getTypeString();
                gotoxy(x+3,++y);
                cout<<"\tCost : Rs "<<cost;
```

```
                gotoxy(x+3,++y);
                cout<<"\tTravel Time : "<<time/60<<" hrs "<<time%60<<" min";
                                gotoxy(x,++y);
                                dash();
                                (*posy)=y;
                        }
};
struct RouteGraphData {
   int locationCount,routeCount,timeMatrix[MAX_DESTINATIONS][MAX_DESTINATIONS];
   char locations[MAX_DESTINATIONS][50];
   Route routes[MAX_DESTINATIONS*MAX_DESTINATIONS];
   double costMatrix[MAX_DESTINATIONS][MAX_DESTINATIONS];
   TransportType transportMatrix[MAX_DESTINATIONS][MAX_DESTINATIONS];
};
class RouteGraph {
        private:
           double costMatrix[MAX_DESTINATIONS][MAX_DESTINATIONS];
           int timeMatrix[MAX_DESTINATIONS][MAX_DESTINATIONS],locationCount,routeCount,x,y;
           TransportType transportMatrix[MAX_DESTINATIONS][MAX_DESTINATIONS];
           char locations[MAX_DESTINATIONS][50];
           Route routes[MAX_DESTINATIONS * MAX_DESTINATIONS];
        public:
           RouteGraph() {
              locationCount=0;
              routeCount=0;
              for(int i=0;i<MAX_DESTINATIONS;i++) {
                 strcpy(locations[i],"");
                 for(int j=0;j<MAX_DESTINATIONS;j++) {
                    if(i==j) {
                       costMatrix[i][j]=0;
                       timeMatrix[i][j]=0;
                    } else {
                       costMatrix[i][j]=INF;
                       timeMatrix[i][j]=INF;
                    }
                    transportMatrix[i][j]= OTHER;
                 }
              }
           }
           int addLocation(const char* name) {
              for(int i = 0; i < locationCount; i++)
                 if(strcmp(locations[i], name) == 0)
                    return i;
              if(locationCount<MAX_DESTINATIONS) {
                 strncpy(locations[locationCount],name,49);
                 locations[locationCount][49]='\0';
                 return locationCount++;
              } else {
                 cout<<"Maximum destinations reached.";
                 return -1;
              }
           }
           void addRoute(const char* from,const char* to,double cost,int time,TransportType type) {
              int fromIndex=addLocation(from),toIndex=addLocation(to);
              if(fromIndex!=-1 && toIndex!=-1) {
                 costMatrix[fromIndex][toIndex]=cost;
                 timeMatrix[fromIndex][toIndex]=time;
                 transportMatrix[fromIndex][toIndex]=type;
                 if(routeCount<MAX_DESTINATIONS*MAX_DESTINATIONS)
                    routes[routeCount++]=Route(from,to,cost,time,type);
              }
           }
           void displayLocations() {
                    x=*posx;
                                y=*posy;
                    gotoxy(x,y+=3);
              if(locationCount==0)
                 cout<<"No locations added yet.";
              else
              {
                    cout<<"Available Locations :";
                    for(int i=0;i<locationCount;i++)
                    {
                                gotoxy(x+5,++y);
                                cout<<i+1<<". "<<locations[i];
```

```
                                        }
                                }
                                (*posx)=x;
                                (*posy)=y+=1;
        }
        void displayRoutes() {
                x=*posx;
                                y=*posy;
           if(routeCount==0) {
                gotoxy(x+=5,y+=3);
              cout<<"No routes added yet.";
           }
           else
                for(int i=0;i<routeCount;i++) {
                cout<<"\t"<<i+1<<". ";
                routes[i].displayRoute();
                                        }
                                (*posy)+=3;
        }
        int findLocationIndex(const char* name) {
           for(int i=0;i<locationCount;i++)
              if(strcmp(locations[i],name)==0)
                 return i;
           return -1;
        }
        void findShortestPath(const char* start,const char* end,bool useCost) {
           int startIndex=findLocationIndex(start),endIndex=findLocationIndex(end);
           drawHeader("Budget Analysis : Least Expensive Path",x=45,y=0);
           if(startIndex==-1 || endIndex==-1)
           {
                gotoxy(x=44,y+=3);
                cout<<"One or both locations not found.";
                (*posy)=y;
                        }
                        else
                        {
                                        double weight[MAX_DESTINATIONS][MAX_DESTINATIONS];
                   for(int i=0;i<locationCount;i++)
                      for(int j=0;j<locationCount;j++)
                         if(useCost)
                            weight[i][j]=costMatrix[i][j];
                         else
                            weight[i][j]=(double)timeMatrix[i][j];
                   double dist[MAX_DESTINATIONS];
                   bool visited[MAX_DESTINATIONS];
                   int prev[MAX_DESTINATIONS];
                   for(int i=0;i<locationCount;i++) {
                      dist[i]=INF;
                      visited[i]=false;
                      prev[i]=-1;
                   }
                   dist[startIndex]=0;
              for(int count=0;count<locationCount-1; count++) {
                      double min=INF;
                      int minIndex=-1;
                      for(int v = 0; v < locationCount; v++)
                         if(!visited[v] && dist[v]<=min) {
                            min=dist[v];
                            minIndex=v;
                         }
                      visited[minIndex]=true;
                      for(int v=0;v<locationCount;v++)
                         if(!visited[v]  &&  weight[minIndex][v]!=INF  &&  dist[minIndex]!=INF  &&
dist[minIndex]+weight[minIndex][v]<dist[v]) {
                            dist[v]=dist[minIndex]+weight[minIndex][v];
                            prev[v]=minIndex;
                         }
                          }
                   if(dist[endIndex]==INF)
                            cout<<"No path exists between "<<start<<" and "<<end;
                   else
                            {
                                 cout<<"\tLeast Expensive Path : "<<endl;
                                 int path[MAX_DESTINATIONS];
                                 int pathLen=0;
```

```
                                    for(int at=endIndex;at!=-1;at=prev[at])
                                       path[pathLen++]=at;
                                    cout<<"\t\t";
                                    for(int i=pathLen-1;i>=0;i--) {
                                       cout<<locations[path[i]];
                                       if(i>0)cout<<" -> ";
                                    }
                                    cout<<endl;
                                    if(useCost)
                                       cout<<"\t\tTotal Cost : Rs "<<dist[endIndex]<<endl;
                                    else {
                                       int hours=(int)dist[endIndex]/60;
                                       int mins=(int)dist[endIndex]%60;
                                       cout<<"\t\tTotal Time : "<< hours<<" hrs "<<mins<<" min"<<endl;
                                    }
                                    y=6;
                                                cout<<endl<<"\tRoute Details :"<<endl;
                                    for(int i=pathLen-1;i>0;i--) {
                                       int from=path[i];
                                       int to=path[i-1];
                                       cout<<"\t\t"<<locations[from]<<" -> " << locations[to]<<endl;
                                       cout<<"\t\tTransport Type : "<<getTransportTypeName(transportMatrix[from][to])<<endl;
                                       cout<<"\t\tCost : Rs "<<costMatrix[from][to]<<endl;
                                       cout<<"\t\tTime    :    "<<timeMatrix[from][to]/60<<"    hrs    "<<timeMatrix[from][to]%60<<"
min"<<endl;
                                       if(i>1) cout<<"\t    ----------"<<endl;
                                       y+=6;
                                    }
                                    dash();
                                          }
                                          (*posy)=y+=3;
                                }
                        }
                const char* getTransportTypeName(TransportType type) {
                   switch(type) {
                      case AIR: return "Air";
                      case BUS: return "Bus";
                      case CAR: return "Car";
                      case OTHER: return "Other";
                      default: return "Unknown";
                   }
                }
                int getLocationCount() const { return locationCount; }
                int getRouteCount() const { return routeCount; }
                const char* getLocationName(int index) const {
                   if(index>=0 && index<locationCount)
                      return locations[index];
                   return "";
                }
                void saveToFile(const char* filename) {
                        int x=*posx,y=*posy;
                           RouteGraphData data;
                           data.locationCount=locationCount;
                           data.routeCount=routeCount;
                           for(int i=0;i<locationCount;++i) {
                              strcpy(data.locations[i],locations[i]);
                              for(int j=0;j<locationCount;++j) {
                                 data.costMatrix[i][j]=costMatrix[i][j];
                                 data.timeMatrix[i][j]=timeMatrix[i][j];
                                 data.transportMatrix[i][j]=transportMatrix[i][j];
                              }
                           }
                           for(int i=0;i<routeCount;++i)
                              data.routes[i]=routes[i];
                           gotoxy(x-5,y+=2);
                           if(::saveToFile(data,filename))
                              cout<<"Routes saved to file "<<filename<<" successfully";
                           else
                              cout<<"Failed to save route graph.";
                        }
                void loadFromFile(const char* filename) {
                        (*posy)-=3;
                                int x=30,y=*posy;
                           RouteGraphData data;
                           (*posx)=x;
```

```
                if(::loadFromFile(data,filename)) {
                        locationCount=data.locationCount;
                        routeCount=data.routeCount;
                        for (int i=0;i<locationCount;++i) {
                           strcpy(locations[i],data.locations[i]);
                           for (int j=0;j<locationCount;++j) {
                              costMatrix[i][j]=data.costMatrix[i][j];
                              timeMatrix[i][j]=data.timeMatrix[i][j];
                              transportMatrix[i][j]=data.transportMatrix[i][j];
                           }
                        }
                        for(int i=0;i<routeCount;++i)
                           routes[i]=data.routes[i];
                        cout<<"Routes loaded successfully from "<<filename;
                     }
                }
             void dfsAllPaths(int current,int end,bool visited[],int path[],int pathIndex,double currentCost,int currentTime) {
                x=10;
                        visited[current]=true;
                path[pathIndex]=current;
                pathIndex++;
                if(current==end) {
                        (*posy)+=1;
                y=*posy;
                        gotoxy(x,++y);
                cout<<"Path : ";
                for(int i=0;i<pathIndex;i++) {
                cout<<locations[path[i]];
                if(i<pathIndex-1)
                                                        cout<<" -> ";
                }
                gotoxy(x+3,++y);
                cout<<"Total Cost : Rs "<<currentCost;
                gotoxy(x+3,++y);
                cout<<"Total Time : "<<currentTime/60<< " hrs "<<currentTime%60<<" min";
                                gotoxy(x+3,++y);
                cout<<"Routes : ";
                gotoxy(x,++y);
                cout<<"+------------------+------------------+----------+-------------+-----------+";
                gotoxy(x,++y);
                cout<<"| From             | To               | Cost     | Transport   | Time      |";
                gotoxy(x,++y);
                cout<<"+------------------+------------------+----------+-------------+-----------+";
                for(int i=0;i<pathIndex-1;i++) {
                int u=path[i],v=path[i + 1];
                gotoxy(x,++y);
                cout<<"|              "<<left<<setw(16)<<locations[u]<<"              |              "<<left<<setw(16)<<locations[v]<<"              |
$"<<right<<setw(7)<<costMatrix[u][v]<<          "         |         "<<left<<setw(11)<<getTransportTypeName(transportMatrix[u][v])<<"         |
"<<right<<setw(2)<<timeMatrix[u][v]/60<<" h "<<right<<setw(2)<<timeMatrix[u][v]%60<<" m |";
                }
                     gotoxy(x,++y);
                     cout<<"+------------------+------------------+----------+-------------+-----------+";
                           (*posy)=++y;
                }
                           else {
                     for(int i=0;i<locationCount;i++)
                        if(!visited[i] && costMatrix[current][i]!=INF)

dfsAllPaths(i,end,visited,path,pathIndex,currentCost+costMatrix[current][i],currentTime+timeMatrix[current][i]);
                }
                   visited[current]=false;
                   pathIndex--;
                }
                void findAllRoutesWithCost(const char* start,const char* end) {
                int startIndex=findLocationIndex(start),endIndex=findLocationIndex(end);
             drawHeader("Budget Analysis : All Paths",x=45,y=0);
                if (startIndex == -1 || endIndex == -1) {
                        gotoxy(x,y+=2);
                cout<<"Invalid locations.";
                }
                else
                {
                        bool visited[MAX_DESTINATIONS]={false};
                        int path[MAX_DESTINATIONS],pathIndex=0;
                        gotoxy(10,y+=3);
```

```
                                cout<<"All Paths from "<<start<<" to "<<end<<" :";
                                (*posy)=++y;
                                dfsAllPaths(startIndex,endIndex,visited,path,pathIndex,0.0,0);
                                }
                                gotoxy(x,*posy);
                                        dash();
                                (*posy)+=3;
                        }
};
class TravellerProfile {
        private:
           char name[50];
           int age,plannedDays,x,y;
           double budget;
        public:
           TravellerProfile() {
              name[0]='\0';
              age=0;
              budget=0;
              plannedDays=0;
           }
           void setProfile() {
              drawHeader("Traveller Profile : Setup",x=45,y=4);
              gotoxy(x=44,y+=3);
              cout<<"Enter your name : ";
              cin.ignore();
              cin.getline(name,sizeof(name));
              gotoxy(x,++y);
              cout<<"Enter age : ";
              cin>>age;
              cin.ignore();
              gotoxy(x,++y);
              cout<<"Enter total travel budget (Rs) : ";
              cin>>budget;
              gotoxy(x,++y);
              cout<<"Enter planned days for travel : ";
              cin>>plannedDays;
              (*posx)=x;
              (*posy)=y+2;
              saveProfile();
           }
           void saveProfile() {
                   gotoxy(*posx,*posy);
              if(saveToFile(*this,"traveller_profile.txt"))
                                cout<<"Profile saved successfully!";
                           else
                              cout<<"Unable to save profile.";
                           (*posy)=y+5;
           }
                   bool loadProfile() {
                   return loadFromFile(*this,"traveller_profile.txt");
                   }
           void viewProfile() {
              drawHeader("Traveller Profile : View",x=45,y=4);
              (*posx)=x-=10;
                      (*posy)=y+=3;
                      if(loadProfile()) {
                                        gotoxy(x=44,y-=3);
                         cout<<"Name : "<<name;
                         gotoxy(x,++y);
                         cout<<"Age : "<<age;
                         gotoxy(x,++y);
                         cout<<"Budget : Rs "<<budget;
                         gotoxy(x,++y);
                         cout<<"Planned Days : "<<plannedDays;
                      } else {
                         gotoxy(x,y+=3);
                         cout<<"No profile found. Please set up a profile first.";
                      }
                      (*posx)=44;
                              (*posy)=y+3;
                   }
           double getBudget() const { return budget; }
           int getPlannedDays() const { return plannedDays; }
           const char* getName() const { return name; }
```

```
};
class TripPlanner {
        private:
          Destination destinations[MAX_DESTINATIONS];
          int destinationCount,x,y;
          RouteGraph routeGraph;
          TravellerProfile traveller;
        public:
          TripPlanner() {
            destinationCount = 0;
          }
          void saveDestinations() {
                char filename[100];
            drawHeader("Destinations : Save",x=45,y=4);
            gotoxy(x=35,y+=3);
                        cout<<"Enter filename to save destinations : ";
                        cin>>filename;
              DestinationData dd;
              dd.count=destinationCount;
                for (int i=0;i<destinationCount;++i) {
                  Destination& d=destinations[i];
                  DestinationRawData& raw=dd.data[i];
                  strcpy(raw.name,d.getName());
                  raw.stayDuration=d.getStayDuration();
                  raw.attractionCount=d.getAttractionCount();
                  for(int j=0;j<raw.attractionCount;++j) {
                    Attraction* a=d.getAttraction(j);
                    if(a) {
                      strcpy(raw.attractions[j].name,a->getName());
                      raw.attractions[j].cost=a->getCost();
                      raw.attractions[j].timeRequired=a->getTimeRequired();
                    }
                  }
                }
                        gotoxy(x,y+=2);
                if(::saveToFile(dd,filename))
                  cout<<"Destinations saved to "<<filename<<" successfully!";
                else
                  cout<<"Failed to save destinations.";
                (*posy)=y+=3;
              }
          void loadDestinations() {
                char filename[100];
            drawHeader("Destinations : Load",x=45,y=4);
            gotoxy(x=30,y+=3);
                        cout<<"Enter filename to load destinations from : ";
                        cin>>filename;
            DestinationData dd;
            (*posx)=x;
                        (*posy)=y+=2;
                if (!::loadFromFile(dd,filename))
                {
                        gotoxy(x,y+=2);
                        cout<<"Failed to load destinations.";
                        }
                        else
                        {
                                destinationCount=dd.count;
                          for (int i=0;i<destinationCount;++i) {
                            DestinationRawData& raw=dd.data[i];
                            Destination d;
                            d.setName(raw.name);
                            d.setStayDuration(raw.stayDuration);
                            for (int j=0;j<raw.attractionCount;++j) {
                              AttractionData& a=raw.attractions[j];
                              Attraction* attraction=new Attraction(a.name, a.cost, a.timeRequired);
                              d.addAttraction(attraction);
                            }
                            destinations[i]=d;
                            routeGraph.addLocation(d.getName());
                          }
                                gotoxy(x,y);
                          cout<<"Destinations loaded successfully from "<<filename;
                        }
                        (*posx)=x+=10;
```

```
            (*posy)=y+=3;
            }
void addDestination(const Destination& dest) {
   if(destinationCount<MAX_DESTINATIONS)
      destinations[destinationCount++]=dest;
   else
      cout<<"Maximum destinations reached.";
}
void createProfile() {
   traveller.setProfile();
}
void viewProfile() {
   traveller.viewProfile();
}
void addRoute() {
   char from[50],to[50];
   double cost;
   int time,transportChoice;
   drawHeader("Routes : Add",x=45,y=0);
   (*posx)=x-3;
   (*posy)=y;
   routeGraph.displayLocations();
   x=(*posx);
   y=(*posy);
   gotoxy(x,++y);
   cout<<"Enter new route details :";
   gotoxy(x,++y);
   cout<<"Enter source name : ";
   cin.ignore();
   cin.getline(from,sizeof(from));
   gotoxy(x,++y);
   cout<<"Enter destination name : ";
   cin.getline(to,sizeof(to));
   gotoxy(x,y+=2);
   cout<<"Transportation Type :";
   gotoxy(x,++y);
   cout<<"1. Air";
   gotoxy(x,++y);
   cout<<"2. Bus";
   gotoxy(x,++y);
   cout<<"3. Car";
   gotoxy(x,++y);
   cout<<"4. Other";
   gotoxy(x,++y);
   cout<<"Enter choice : ";
                        cin>>transportChoice;
                        gotoxy(x,y+=2);
                        cout<<"Cost (Rs) : ";
   cin>>cost;
   gotoxy(x,++y);
   cout<<"Time (in minutes) : ";
   cin>>time;
   TransportType type;
   switch(transportChoice) {
      case 1: type=AIR; break;
      case 2: type=BUS; break;
      case 3: type=CAR; break;
           case 4: type=OTHER; break;
      default: type=OTHER;
   }
   routeGraph.addRoute(from,to,cost,time,type);
   gotoxy(x,y+=2);
   cout<<"Route added successfully.";
   (*posy)=y+=3;
}
void displayRoutes() {
   drawHeader("Routes : View",x=35,y=0);
   (*posy)=y;
   routeGraph.displayRoutes();
}
void findBestRoute() {
   char from[50],to[50];
   int preference;
   drawHeader("Budget Analysis : For Best Path",x=45,y=0);
   (*posy)=y;
```

```
    routeGraph.displayLocations();
    y=*posy;
                gotoxy(x,++y);
                cout<<"Enter source : ";
    cin.ignore();
    cin.getline(from,sizeof(from));
    gotoxy(x,++y);
    cout<<"Enter destination : ";
    cin.getline(to,sizeof(to));
    gotoxy(x,y+=2);
    cout<<"Optimization preference :";
    gotoxy(x,++y);
    cout<<"1. Optimize for cost";
    gotoxy(x,++y);
    cout<<"2. Optimize for time";
    gotoxy(x,++y);
    cout<<"Enter your choice : ";
    cin>>preference;
    bool optimizeForCost=(preference==1);
    routeGraph.findShortestPath(from,to,optimizeForCost);
                (*posy)+=3;
        }
void saveRoutes() {
        char filename[100];
    drawHeader("Routes : Save",x=45,y=4);
    gotoxy(x=35,y+=3);
                cout<<"Enter filename to save routes : ";
                cin>>filename;
                (*posx)=x+5;
                (*posy)=y;
        routeGraph.saveToFile(filename);
                (*posx)=x;
                (*posy)=y+5;
}
        void loadRoutes() {
                char filename[100];
    drawHeader("Routes : Load",x=45,y=4);
    gotoxy(x=30,y+=3);
                cout<<"Enter filename to load routes from : ";
                cin>>filename;
        routeGraph.loadFromFile(filename);
                (*posy)+=3;
        }
void budgetAnalysis() {
    drawHeader("Budget Analysis : All Paths",x=45,y=4);
    double totalBudget=traveller.getBudget();
    if(totalBudget==0)
    {
        gotoxy(x=44,y+=3);
        cout<<"Please set up your traveller profile first.";
                }
    else
    {
        double totalCost=0;
            double attractionCost=0;
            for(int i=0;i<destinationCount;i++) {
               double destCost=destinations[i].calculateTotalCost();
               attractionCost+=destCost;
               totalCost+=destCost;
            }
            cout<<"\tTotal Budget : Rs "<<totalBudget<<endl;
            cout<<"\tEstimated Total Cost : Rs "<<totalCost<<endl;
            if(totalCost>totalBudget)
               cout<<"\tBudget Deficit : Rs "<<totalCost-totalBudget<<endl<<"\tWarning: Your current plan exceeds your
budget."<<endl;
            else
               cout<<"\tBudget Surplus: Rs "<<totalBudget-totalCost<<endl<<"\tGood news! Your plan is within
budget."<<endl;
            cout<<endl<<"\tCost Breakdown for All Paths :"<<endl;
            cout<<"\t\tTotal cost for visiting all attractions : Rs "<<attractionCost<<endl;
            char start[50],end[50];
            cout<<endl<<"\t\tEnter source : ";
            cin.ignore();
            cin.getline(start, sizeof(start));
            cout<<"\t\tEnter destination : ";
```

```
                cin.getline(end, sizeof(end));
                routeGraph.findAllRoutesWithCost(start,end);
                    }
        }
        void addDestinationDetails() {
            char name[50];
            int days;
            drawHeader("Destinations : Add",x=45,y=4);
            gotoxy(x=44,y+=3);
            cout<<"Enter destination name : ";
            cin.ignore();
            cin.getline(name,sizeof(name));
            gotoxy(x,++y);
            cout<<"Enter stay duration (days) : ";
            cin>>days;
            Destination newDest;
            newDest.setName(name);
            newDest.setStayDuration(days);
            char addMore;
            do {
                char attractionName[50];
                double cost;
                int time;
                cin.ignore();
                gotoxy(x,y+=2);
                cout<<"Enter attraction name : ";
                cin.getline(attractionName,sizeof(attractionName));
                gotoxy(x,++y);
                cout<<"Enter cost (Rs) : ";
                cin>>cost;
                gotoxy(x,++y);
                cout<<"Enter time required (hours) : ";
                cin>>time;
                Attraction* attraction=NULL;
                attraction=new Attraction(attractionName,cost,time);
                if(attraction)
                    newDest.addAttraction(attraction);
                gotoxy(x,++y);
                cout<<"Add another attraction? [Y/N] : ";
                cin>>addMore;
            } while(toupper(addMore)=='Y');
            addDestination(newDest);
            gotoxy(x,y+=2);
            cout<<"Destination added successfully!";
            routeGraph.addLocation(name);
                        (*posy)=y+=3;
            }
        void displayDestinations() {
            drawHeader("Destinations : View",x=45,y=0);
            if(destinationCount==0) {
                gotoxy(x=44,y+=3);
                (*posy)=y;
                cout<<"No destinations added yet!";
                        }
            else {
                (*posy)=y;
                for(int i=0;i<destinationCount;i++)
                        destinations[i].displayDestination();
                        }
            (*posy)+=3;
        }
};
int main() {
    TripPlanner planner;
    int choice1,choice2,x,y;
    bool exit=false;
    char response;
    posx=&x;
    posy=&y;
    while(!exit) {
        drawHeader("TOUR PLANNER MENU",x=45,y=4);
        gotoxy(x=44,y+=3);
        cout<<"1. Traveller Profile";
        gotoxy(x,++y);
        cout<<"2. Destination";
```

```
gotoxy(x,++y);
cout<<"3. Route";
gotoxy(x,++y);
cout<<"4. Budget Analysis";
gotoxy(x,++y);
cout<<"5. Exit";
gotoxy(x,y+=2);
cout<<"Enter your choice: ";
cin>>choice1;
switch(choice1) {
   case 1:
       drawHeader("Traveller Profile : Options",x=45,y=4);
                        gotoxy(x=44,y+=3);
                        cout<<"1. Setup Traveller Profile";
                        gotoxy(x,++y);
                        cout<<"2. View Traveller Profile";
                        gotoxy(x,y+=2);
                        cout<<"Enter your choice: ";
                        cin>>choice2;
                        if(choice2==1)
       planner.createProfile();
      else if(choice2==2)
       planner.viewProfile();
      else
       cout<<"Invalid choice. Please try again."<<endl;
      break;
   case 2:
       drawHeader("Destinations : Options",x=45,y=4);
                        gotoxy(x=44,y+=3);
                        cout<<"1. Add Destination";
                        gotoxy(x,++y);
                        cout<<"2. View Destinations";
                        gotoxy(x,++y);
                        cout<<"3. Save Destinations";
                        gotoxy(x,++y);
                        cout<<"4. Load Destinations";
                        gotoxy(x,y+=2);
                        cout<<"Enter your choice: ";
                        cin>>choice2;
                        if(choice2==1)
       planner.addDestinationDetails();
      else if(choice2==2)
       planner.displayDestinations();
      else if(choice2==3)
                                                planner.saveDestinations();
      else if(choice2==4)
       planner.loadDestinations();
      else
       cout<<"Invalid choice. Please try again."<<endl;
      break;
   case 3:
       drawHeader("Routes : Options",x=45,y=4);
                        gotoxy(x=44,y+=3);
                        cout<<"1. Add Routes";
                        gotoxy(x,++y);
                        cout<<"2. View Routes";
                        gotoxy(x,++y);
                        cout<<"3. Save Routes";
                        gotoxy(x,++y);
                        cout<<"4. Load Routes";
                        gotoxy(x,y+=2);
                        cout<<"Enter your choice: ";
                        cin>>choice2;
                        if(choice2==1)
       planner.addRoute();
      else if(choice2==2)
       planner.displayRoutes();
      else if(choice2==3)
                                                planner.saveRoutes();
      else if(choice2==4)
       planner.loadRoutes();
      else
       cout<<"Invalid choice. Please try again."<<endl;
      break;
   case 4:
```

```
            drawHeader("Budget Analysis : Options",x=45,y=4);
                        gotoxy(x=44,y+=3);
                        cout<<"1. For Best Path";
                        gotoxy(x,++y);
                        cout<<"2. For All Paths";
                        gotoxy(x,y+=2);
                        cout<<"Enter your choice: ";
                        cin>>choice2;
                        if(choice2==1)
            planner.findBestRoute();
          else if(choice2==2)
                                                    planner.budgetAnalysis();
          else
            cout<<"Invalid choice. Please try again."<<endl;
          break;
        case 5:
            system("cls");
          exit=true;
          break;
        default:
          cout<<"Invalid choice. Please try again."<<endl;
      }
      if(!exit) {
                                gotoxy(*posx,*posy);
                                cout<<"Return to Main Menu? [Y/N] : ";
                                cin>>response;
        if(toupper(response)!='Y')
          exit=true;
      }
    }
    drawHeader("Thank you for using Tour Planner",x=40,y=4);
    return 0;
}
```

Output

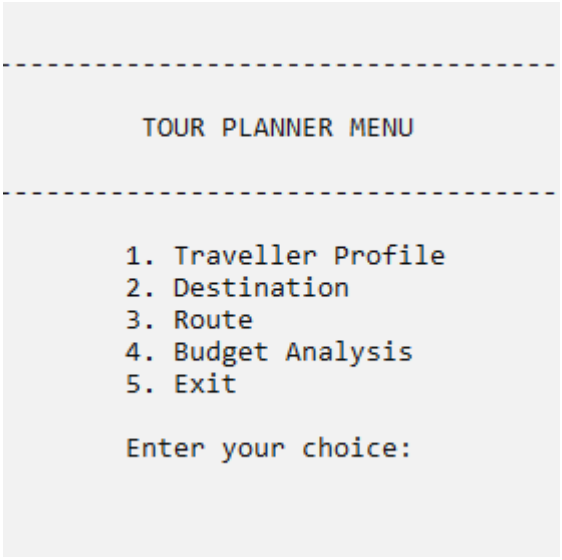


Fig 1 : Main Menu

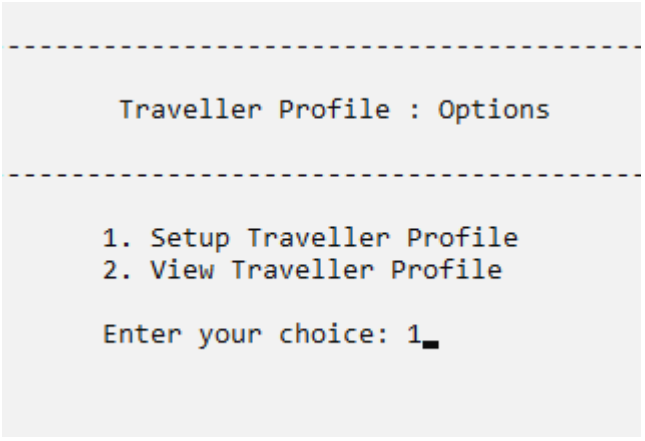


Fig 2 : Traveller Profile

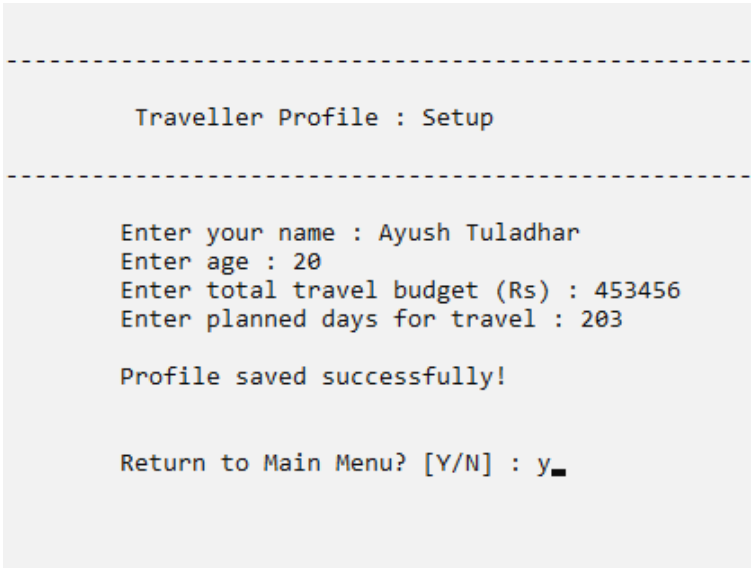


Fig 2.1 : Traveller Profile Setup

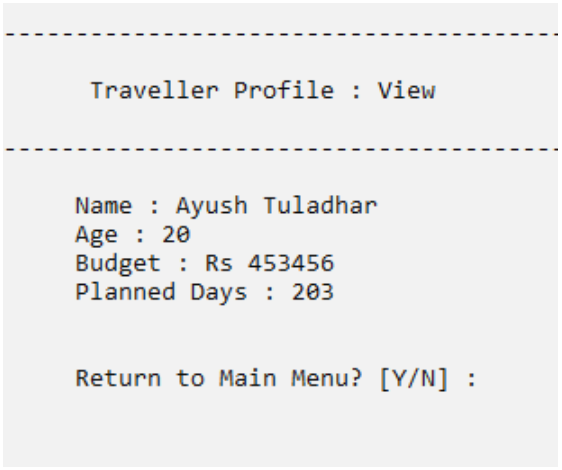


Fig 2.2 : Traveller Profile View

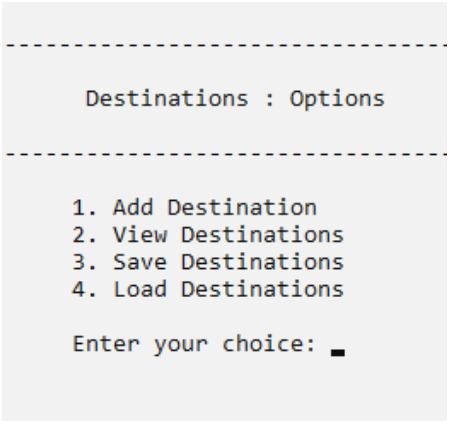


Fig 3 : Destinations

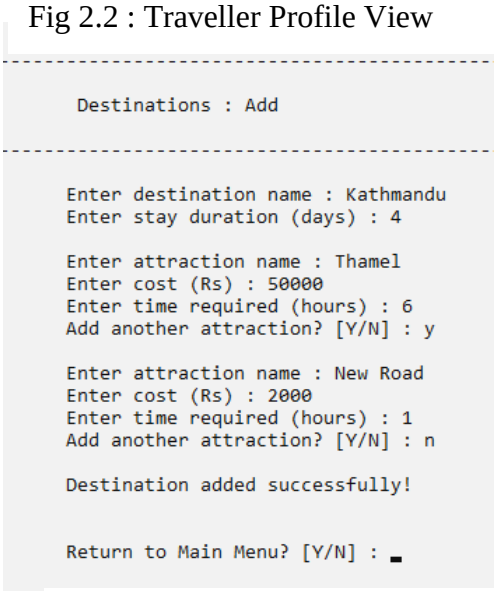


Fig 3.1 : Destinations Add

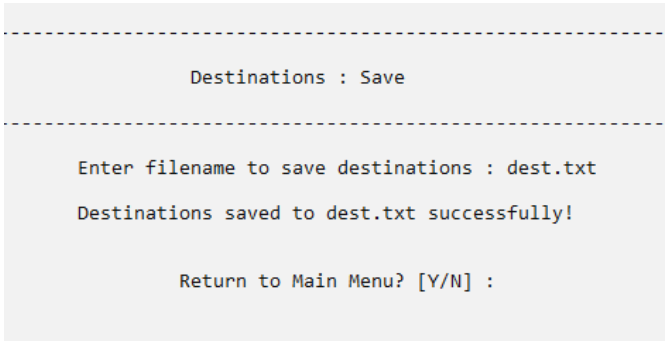


Fig 3.3 : Destinations Save

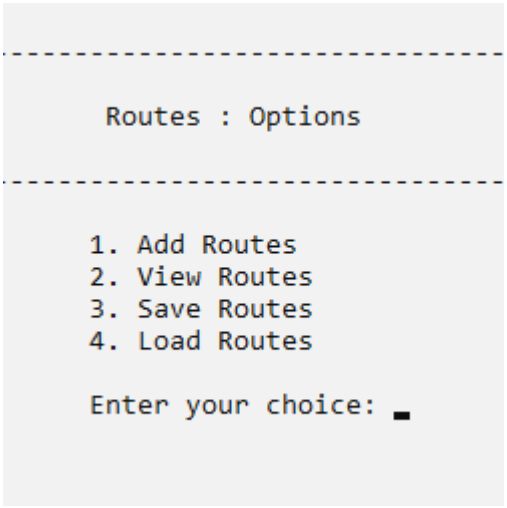


Fig 4 : Routes

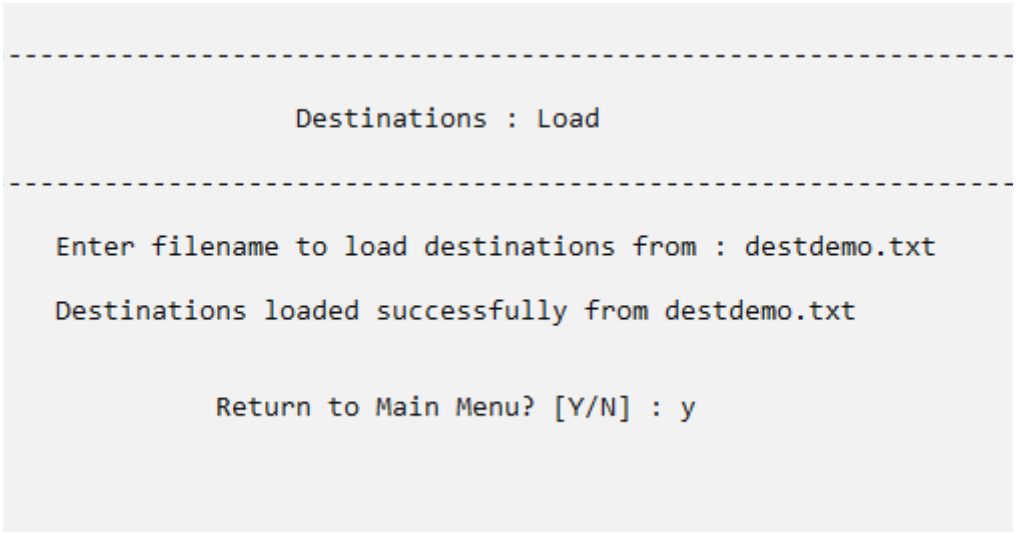


Fig 3.3 : Destinations Load

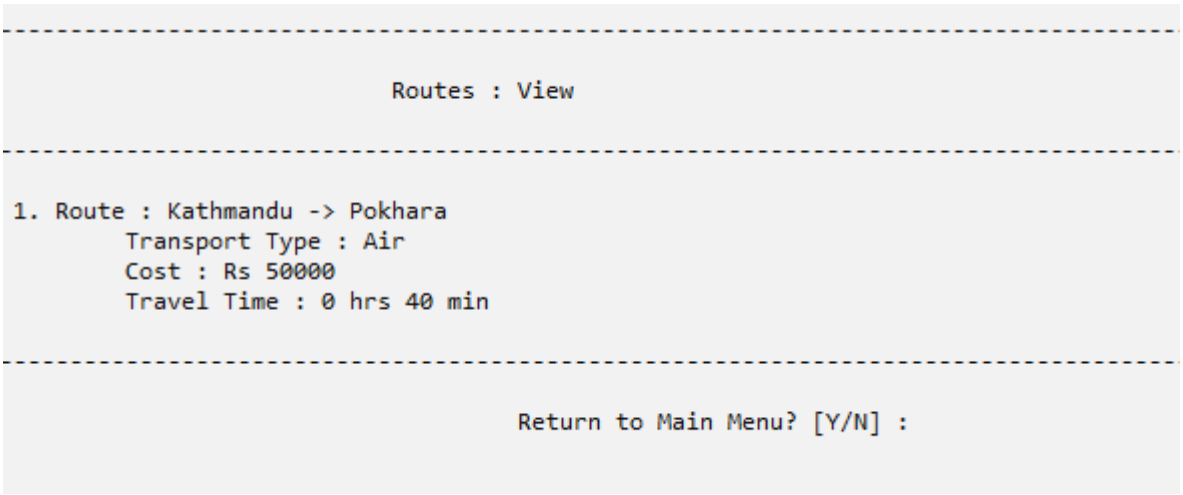


Fig 4.3 : Routes Add

```
-----------------------------------------

     Budget Analysis : For Best Path

-----------------------------------------

    Available Locations :
         1. Kathmandu
         2. Lumbini
         3. Dhading
         4. Gorkha
         5. Pokhara

     Enter source : Kathmandu
     Enter destination : Pokhara

     Optimization preference :
     1. Optimize for cost
     2. Optimize for time
     Enter your choice : 1
```

Fig 5.1 : Best Path Analysis

```
-----------------------------------------------

               Routes : Load

-----------------------------------------------

  Enter filename to load routes from : routdemo.txt

  Routes loaded successfully from routdemo.txt


  Return to Main Menu? [Y/N] : y
```

Fig 3.4 : Routes Load

```
-----------------------------------------------

          Routes : Add

-----------------------------------------------

       Available Locations :
            1. Kathmandu
            2. Lumbini
            3. Dhading
            4. Gorkha
            5. Pokhara

       Enter new route details :
       Enter source name : Kathmandu
       Enter destination name : Pokhara

       Transportation Type :
       1. Air
       2. Bus
       3. Car
       4. Other
       Enter choice : 1

       Cost (Rs) : 50000
       Time (in minutes) : 40

       Route added successfully.


       Return to Main Menu? [Y/N] :
```

Fig 4.1 : Routes Add

```
-------------------------------------------------------------------------------------

                                  Budget Analysis : Least Expensive Path

-------------------------------------------------------------------------------------

 Least Expensive Path :
         Kathmandu -> Pokhara
         Total Time : 1 hrs 0 min

 Route Details :
         Kathmandu -> Pokhara
         Transport Type : Other
         Cost : Rs 20000
         Time : 1 hrs 0 min

-------------------------------------------------------------------------------------

                                  Return to Main Menu? [Y/N] : _
```

Fig 5.3 : Best Path Analysis (Time)

```
------------------------------------------------

            Routes : Save

------------------------------------------------

   Enter filename to save routes : route.txt

   Routes saved to file route.txt successfully


   Return to Main Menu? [Y/N] :
```

Fig 4.3 : Routes Save

```
-----------------------------------

     Budget Analysis : Options

-----------------------------------

    1. For Best Path
    2. For All Paths

    Enter your choice: ▬
```

Fig 5 : Budget Analysis

```
------------------------------------------------------------------------------------------------------------

                                    Budget Analysis : All Paths

------------------------------------------------------------------------------------------------------------

  Total Budget : Rs 453456
  Estimated Total Cost : Rs 199000
  Budget Surplus: Rs 254456
  Good news! Your plan is within budget.

  Cost Breakdown for All Paths :
         Total cost for visiting all attractions : Rs 199000

         Enter source : Kathmandu
         Enter destination : Pokhara▬
```

Fig 5.4 : Budget Analysis (All Paths)

```
------------------------------------------------------------------------------------------

                                    Budget Analysis : Least Expensive Path

------------------------------------------------------------------------------------------

Least Expensive Path :
        Kathmandu -> Dhading -> Gorkha -> Pokhara
        Total Cost : Rs 8700

Route Details :
        Kathmandu -> Dhading
        Transport Type : Car
        Cost : Rs 6300
        Time : 5 hrs 0 min
    ----------
        Dhading -> Gorkha
        Transport Type : Car
        Cost : Rs 1400
        Time : 0 hrs 50 min
    ----------
        Gorkha -> Pokhara
        Transport Type : Bus
        Cost : Rs 1000
        Time : 1 hrs 10 min

------------------------------------------------------------------------------------------


                                    Return to Main Menu? [Y/N] :
```

Fig 5.2 : Best Path Analysis (Cost)

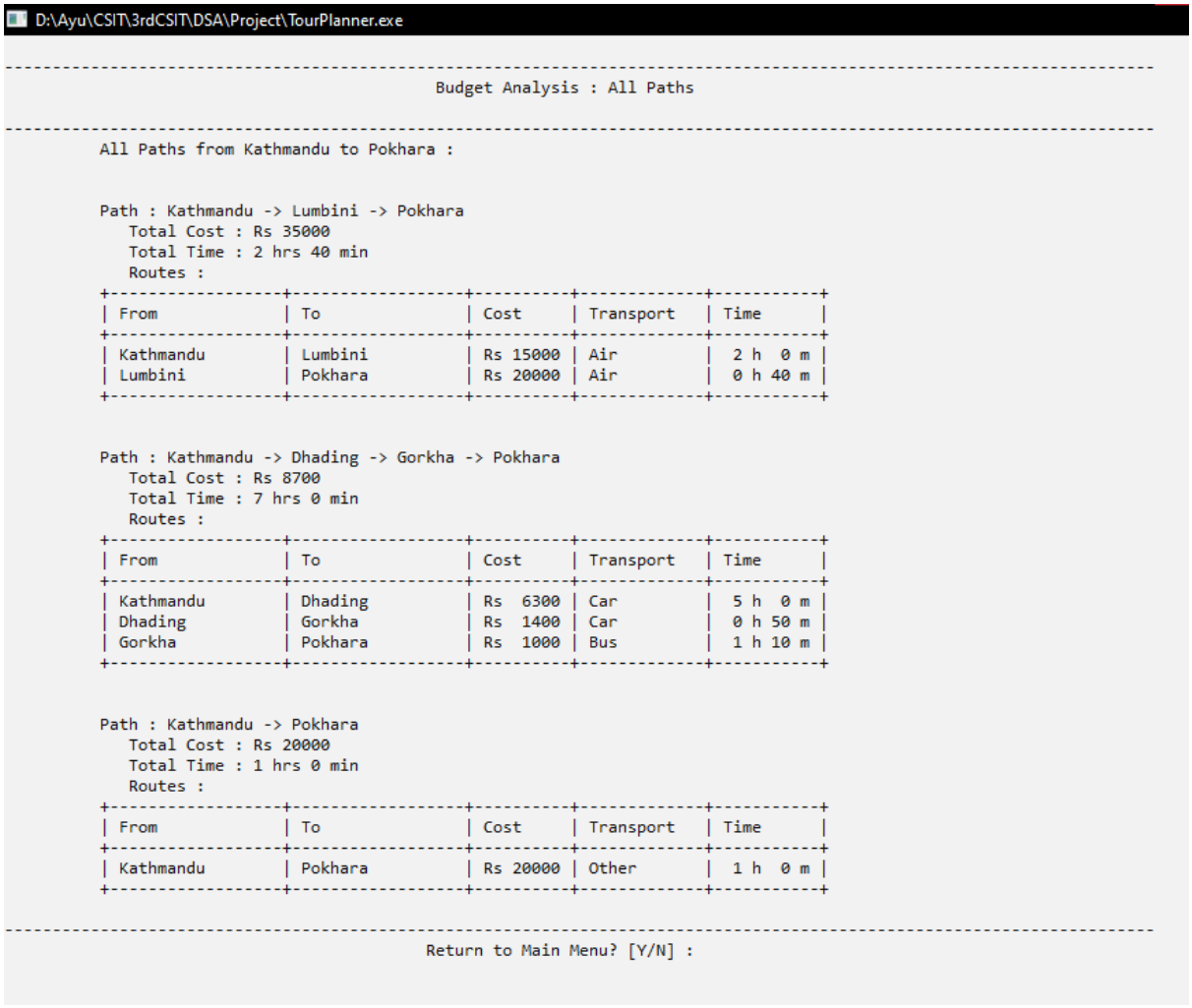


Fig 5.5 : Budget Analysis (All Paths)

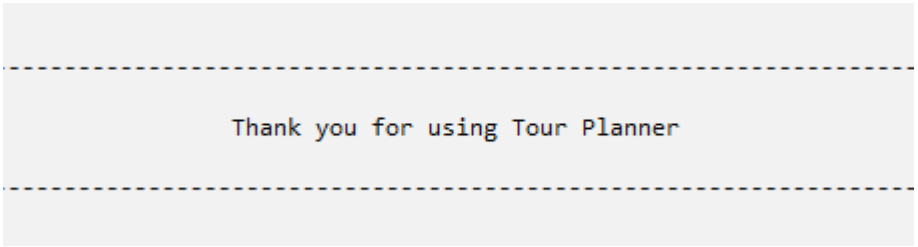


Fig 6 : Exit