MINI PROJECT
On

Travel Plan Optimizer

In
Data Structure & Algorithms

BACHELOR OF TECHNOLOGY
IN
Artificial Intelligence and Machine Learning

SUBMITTED BY

Ramitha V
(PRN - 22070126082)

Under the Supervision of

Dr. Hema Karande,
Assistant Professor

SYMBIOSIS INSTITUTE OF TECHNOLOGY, PUNE – 412115

(A CONSTITUENT OF SYMBIOSIS INTERNATIONAL (DEEMED UNIVERSITY))

2023-24

Problem Statement:

The problem at hand is to create a program that assists travelers in planning their trips effectively. This program should allow users to input various destinations, allocate a certain number of days and a budget to each destination, and further specify activities within those destinations. The challenge is to manage these inputs while ensuring that the plan remains within the user's budget and time frame. The program should also present the travel plan in an organized way and provide warnings if there are any issues with the plan, such as remaining cost with no days left or remaining days with no budget left.

Motivation:

The motivation behind this program is to simplify the process of travel planning for individuals who want to create detailed itineraries. Travel planning can be a complex task involving decisions about destinations, duration, and budget, as well as the inclusion of various activities. This program aims to address the following motivations:

Streamlining the travel planning process: The program helps users efficiently organize their trips by breaking them down into destinations, days, and activities.

Cost and time management: It ensures that users can allocate their budgets and time effectively by providing real-time feedback on the plan's status.

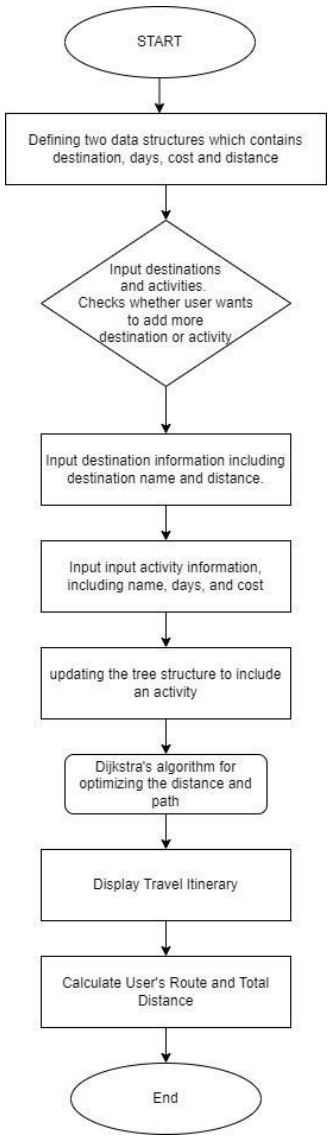
Customization: The program allows for flexibility in planning by accommodating different destinations, activities, and budgets.

Objective :

The main objective of this program is to assist users in creating optimized travel itineraries. The specific objectives are as follows:

- Allow users to input destination details, including name, days, and budget.
- Enable users to add various activities within each destination.
- Maintain a structured tree-based representation of the travel plan.
- Ensure that the plan remains within the specified budget and duration.
- Present the travel itinerary in an organized and readable format.
- Calculate and display the total days and costs for the entire trip.
- Provide warnings in cases where there is a mismatch between remaining cost and remaining days.

Methodology of Implementation:



Hardware/Software Used:

Software used: VSCode

Device Used: HP Pavilion 15

Device RAM: 16 GB

Executable Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>

#define MAX_NAME_LENGTH 50
#define MAX_DESTINATIONS 10

// Structure to represent a tree node
struct TreeNode {
    char name[MAX_NAME_LENGTH];
    int days;
    int cost;
    struct TreeNode* activities;
    struct TreeNode* next;
};

// Structure to represent a destination and its distance
struct Destination {
    char name[MAX_NAME_LENGTH];
    int distance;
};
```

```
// Function to add a node to the tree and update remaining days and cost
void addNode(struct TreeNode* parent, const char* name, int days, int cost, int*
remainingDays, int* remainingCost) {
    struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    strncpy(newNode->name, name, sizeof(newNode->name) - 1);
    newNode->name[sizeof(newNode->name) - 1] = '\0';
    newNode->days = days;
    newNode->cost = cost;
    newNode->activities = NULL;
    newNode->next = NULL;

    if (parent->activities == NULL) {
        parent->activities = newNode;
    } else {
        struct TreeNode* current = parent->activities;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
    }

    // Update remaining days and cost
    *remainingDays -= days;
    *remainingCost -= cost;
}

// Function to display the tree structure in an organized way
void displayTree(struct TreeNode* root, int level, int *totalDays, int *totalCost) {
    if (root == NULL) {
        return;
    }

    // Print only activities, not destinations
    if (level > 0) {
        for (int i = 0; i < level; i++) {
            printf("  ");
```

```
        }
        printf("Activity: %s (Days: %d, Cost: $%d)\n", root->name, root->days, root->cost);
    }

    // Recursively display sub-activities with increased indentation
    struct TreeNode* current = root->activities;
    int activityNum = 1;
    while (current != NULL) {
        displayTree(current, level + 1, totalDays, totalCost);
        current = current->next;
        activityNum++;
    }

    // Update totalDays and totalCost
    *totalDays += root->days;
    *totalCost += root->cost;
}

int main() {
    printf("----------------------TRAVEL PLAN OPTIMIZER-----------------------\n");

    struct TreeNode root;
    strncpy(root.name, "Trip", sizeof(root.name) - 1);
    root.name[sizeof(root.name) - 1] = '\0'; // Ensure null-termination
    root.days = 0;
    root.cost = 0;
    root.activities = NULL;
    root.next = NULL;

    int numDestinations = 0;
    char addAnotherDestination;

    do {
        printf("\n--- Enter Destination #%d ---\n", numDestinations + 1);
        char name[MAX_NAME_LENGTH];
        int days, cost;
```

```
printf("Enter destination name: ");
fgets(name, sizeof(name), stdin);
name[strcspn(name, "\n")] = 0; // Remove the trailing newline character

do {
    printf("Enter number of days: ");
    scanf("%d", &days);
    if (days <= 0) {
        printf("Error: Number of days must be greater than zero.\n");
    }
} while (days <= 0);

do {
    printf("Enter cost: $");
    scanf("%d", &cost);
    if (cost <= 0) {
        printf("Error: Cost must be greater than zero.\n");
    }
} while (cost <= 0);

// Consume the newline character left in the input buffer
getchar();

struct TreeNode* destination = &root;
int remainingDays = days;
int remainingCost = cost;

int numActivities = 0;
char addAnotherActivity;

do {
    char activityName[MAX_NAME_LENGTH];
    int activityDay, activityCost;

    printf("\n--- Enter Activity for %s ---\n", name);
```

```
        printf("Enter activity name: ");
        fgets(activityName, sizeof(activityName), stdin);
        activityName[strcspn(activityName, "\n")] = 0;

       do {
           printf("Enter day allocated (1-%d): ", remainingDays);
           scanf("%d", &activityDay);
           if (activityDay < 1 || activityDay > remainingDays) {
               printf("Error: Invalid day. Please choose between 1 and %d.\n", remainingDays);
           }
        } while (activityDay < 1 || activityDay > remainingDays);

        do {
           printf("Enter cost: $");
           scanf("%d", &activityCost);
           if (activityCost <= 0 || activityCost > remainingCost) {
               printf("Error: Cost must be greater than zero and not exceed the remaining budget
of $%d.\n", remainingCost);
           }
        } while (activityCost <= 0 || activityCost > remainingCost);

        // Consume the newline character left in the input buffer
        getchar();

        addNode(destination, activityName, activityDay, activityCost, &remainingDays,
&remainingCost);

        numActivities++;

        if (remainingDays == 0) {
           printf("No more days left for this destination.\n");
           break;
        } else if (remainingCost == 0) {
           printf("No more budget left for this destination.\n");
           break;
```

```
            } else if (remainingDays < 0 || remainingCost < 0) {
                printf("Error: Days or budget exceeded for this destination.\n");
                break;
            }

            printf("Remaining days: %d, Remaining cost: $%d\n", remainingDays, remainingCost);

            printf("Add another activity for %s? (y/n): ", name);
            scanf(" %c", &addAnotherActivity);
            getchar(); // Consume the newline character

        } while ((addAnotherActivity == 'y' || addAnotherActivity == 'Y') && remainingDays > 0
&& remainingCost > 0);

        numDestinations++;

        printf("Add another destination? (y/n): ");
        scanf(" %c", &addAnotherDestination);
        getchar(); // Consume the newline character

    } while (addAnotherDestination == 'y' || addAnotherDestination == 'Y');

    int totalDays = 0;
    int totalCost = 0;

    printf("\n--- Your Travel Itinerary ---\n");
    displayTree(&root, 0, &totalDays, &totalCost); // Start with level 0

    printf("\nTotal Days Spent: %d\n", totalDays);
    printf("Total Cost Spent: $%d\n", totalCost);

    // Check for the edge cases where there's remaining cost but no days left, and vice versa
    if (totalDays == 0 && totalCost > 0) {
        printf("Warning: There is remaining cost, but no more days left for your trip.\n");
    } else if (totalDays > 0 && totalCost == 0) {
        printf("Warning: There are remaining days, but no more budget left for your trip.\n");
```

```
    }

    // Deallocate memory
    struct TreeNode* current = root.activities;
    while (current != NULL) {
        struct TreeNode* next = current->next;
        free(current);
        current = next;
    }

    return 0;
}
```

Result Analysis with Output Screenshot:

```
--------------------TRAVEL PLAN OPTIMIZER-----------------------

--- Enter Destination #1 ---
Enter destination name: New York
Enter number of days: 2
Enter cost: $800

--- Enter Activity for New York ---
Enter activity name: Empire State Building
Enter day allocated (1-2): 1
Enter cost: $300
Remaining days: 1, Remaining cost: $500
Add another activity for New York? (y/n): y

--- Enter Activity for New York ---
Enter activity name: Ice Skating
Enter day allocated (1-1): 1
Enter cost: $200
No more days left for this destination.
Add another destination? (y/n): y

--- Enter Destination #2 ---
Enter destination name: Paris
Enter number of days: 1
Enter cost: $400

--- Enter Activity for Paris ---
Enter activity name: Eiffel Tower
Enter day allocated (1-1): 1
Enter cost: $400
No more days left for this destination.
Add another destination? (y/n): n
```

```
--- Your Travel Itinerary ---
  Activity: Empire State Building (Days: 1, Cost: $300)
  Activity: Ice Skating (Days: 1, Cost: $200)
  Activity: Eiffel Tower (Days: 1, Cost: $400)

Total Days Spent: 3
Total Cost Spent: $900
```

```
--- Input Destination Information ---

--- Enter Destination #1 ---
Enter destination name: Central Park
Enter distance to the next destination: 120
Add another destination? (y/n): y

--- Enter Destination #2 ---
Enter destination name: Statue of Liberty
Enter distance to the next destination: 100
Add another destination? (y/n): y

--- Enter Destination #3 ---
Enter destination name: Manhattan City Park
Enter distance to the next destination: 50
Add another destination? (y/n): y

--- Enter Destination #4 ---
Enter destination name: Empire State Building
Enter distance to the next destination: 250
Add another destination? (y/n): y

--- Enter Destination #5 ---
Enter destination name: Ice Skating Range
Enter distance to the next destination: 30
Add another destination? (y/n): n

--- Preferred Route Input ---
Enter your preferred source destination: Central Park
Enter your preferred destination: Ice Skating Range

User's Route: Central Park -> Ice Skating Range
Total Distance: 150
Optimized Distance for the user's route: 120
```

Edge Cases:

- **Invalid Input Handling:** The code ensures that user inputs are validated. It checks for conditions such as negative days, zero cost, and other constraints, and prompts the user for valid inputs.
- **Data Structure Size Limitations**: The code limits the size of character arrays.This prevents buffer overflows and handles cases where the user might input long strings.
- **Remaining Budget and Days:** The code tracks and displays the remaining budget and days for each destination, and it raises errors when either budget or days are exceeded.
- **User Route Calculation**: The code calculates the user's preferred route distance by summing the distances between the source and destination, handling cases where these locations are not in the list of destinations.
- **Preferred Route Input:** The code allows the user to input their preferred source and destination. It calculates the total distance for the user's chosen route. If the source or destination is not found in the list of destinations, it raises an error.
- **Case Sensitive:** The code is not case sensitive when taking inputs for source and destination.

How can you implement Symbol Table and AVL Tree in your Projects? (Justify)

Symbol Table for Destinations:

A Symbol Table, often implemented as a hashtable or dictionary, can be used to store destination names as keys and their associated data (e.g., distance to the next destination) as values. This structure can be added to the project as follows:

- When users input destination information, store it in the Symbol Table, allowing fast lookup by destination name.
- Whenever you need to retrieve or update destination details (e.g., distance between destinations), you can directly access the Symbol Table, eliminating the need for linear searches.

AVL Tree for Destinations and Activities:

An AVL Tree is a self-balancing binary search tree that maintains balance, making it useful for sorting and efficient searching of destinations and activities. You can use an AVL Tree to store destination names and their associated activities. Here's how to integrate it:

- Each node in the AVL Tree represents a destination with its activities as children nodes.
- Destinations can be sorted based on their names, which allows for alphabetical sorting of destinations.
- Activities under each destination can also be sorted based on their attributes (e.g., days or cost).

References

[1] https://www.geeksforgeeks.org/project-idea-trip-planner/

[2] https://www.programiz.com/dsa/trees

[3] https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

[4] https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/