SIMATS ENGINEERING

Engineer to Excel

SIMATS

Saveetha Institute of Medical And Technical Sciences

(Declared as Deemed to be University under Section 3 of UGC Act 1956)

A CAPSTONE PROJECT REPORT ON

MINIMUM COST TO REACH DESTINATION IN TIME

Submitted in the partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

Submitted by

R.Santhosh Kumar (192211053)

Under the Supervision of

Dr. R. Dhanalakshmi

SIMATS SCHOOL OF ENGINEERING
THANDALAM CHENNAI-602105

CAPSTONE PROJECT REPORT

Name: R.SANTHOSH KUMAR.

Register Number: 192211053.

Course Code: CSA0656.

Course Name: DESIGN AND ANALYSIS OF ALGORITHM.

Slot: C.

Minimum Cost to Reach Destination in Time

There is a country of n cities numbered from 0 to n - 1 where all the cities are connected by bi-directional roads. The roads are represented as a 2D integer array edges where edges[i] = [xi, yi, timei] denotes a road between cities xi and yi that takes timei minutes to travel. There may be multiple roads of differing travel times connecting the same two cities, but no road connects a city to itself. Each time you pass through a city, you must pay a passing fee. This is represented as a 0-indexed integer array passingFees of length n where passingFees[j] is the amount of dollars you must pay when you pass through city j. In the beginning, you are at city 0 and want to reach city n - 1 in maxTime minutes or less. The cost of your journey is the summation of passing fees for each city that you passed through at some moment of your journey (including the source and destination cities). Given maxTime, edges, and passingFees, return the minimum cost to complete your journey, or -1 if you cannot complete it within maxTime minutes.

DYNAMIC PROGRAMMING

MINIMUM COST TO REACH DESTINATION IN TIME.

ABSTRACT:

The problem involves finding the minimum cost to travel from city 0 to city n-1 within a given maximum time maxTime. Each city has an associated passing fee, and the cities are connected by bi-directional roads with varying travel times. This document presents a dynamic programming solution to this problem, detailing the process step-by-step and providing C code implementation. Additionally, we analyze the complexity of the solution and conclude with the results and findings.

INTRODUCTION:

The task is to find the minimum cost to travel from the starting city (0) to the destination city (n-1) within a specified maximum time. The challenge is to minimize the total cost of passing through cities while adhering to the time constraint. This problem can be approached using dynamic programming to efficiently compute the minimum cost path within the allowed time.

STEP-BY-STEP PROCESS:

1. **Graph Representation**:
   - Represent the cities and roads using an adjacency list.
   - Each edge is represented as a structure containing the destination city and travel time.
2. **DP Array Initialization**:
   - Initialize a 2D array dp where dp[i][t] stores the minimum cost to reach city i using exactly t time.
   - Set all values in dp to infinity (INT_MAX), except for dp[0][0] which is initialized to the passing fee of city 0.

3. **DP Array Update**:
   - Iterate through each possible time from 0 to maxTime.
   - For each city, update the dp values for its neighboring cities based on the travel time and passing fees.
4. **Result Extraction**:
   - The result is the minimum value in dp[n-1][t] for all valid times t from 0 to maxTime.

CODING:

```
#include <stdio.h>

#include <stdlib.h>

#include <limits.h>


#define MAX_CITIES 100

#define MAX_TIME 1000


typedef struct {

    int dest, time;

} Edge;


void addEdge(Edge* graph[], int u, int v, int time, int* graphSize) {

    graph[u] = realloc(graph[u], (graphSize[u] + 1) * sizeof(Edge));

    graph[u][graphSize[u]].dest = v;

    graph[u][graphSize[u]].time = time;

    graphSize[u]++;
```

```
}

int minCost(int maxTime, int** edges, int edgesSize, int* edgesColSize, int* passingFees, int
passingFeesSize) {
    Edge* graph[MAX_CITIES] = {NULL};
    int graphSize[MAX_CITIES] = {0};

    // Build the graph
    for (int i = 0; i < edgesSize; i++) {
        int u = edges[i][0];
        int v = edges[i][1];
        int time = edges[i][2];
        addEdge(graph, u, v, time, graphSize);
        addEdge(graph, v, u, time, graphSize);
    }

    // Initialize the dp array
    int dp[MAX_CITIES][MAX_TIME + 1];
    for (int i = 0; i < MAX_CITIES; i++) {
        for (int j = 0; j <= MAX_TIME; j++) {
            dp[i][j] = INT_MAX;
        }
    }
    dp[0][0] = passingFees[0];
```

```
    // Update the dp array
    for (int t = 0; t <= maxTime; t++) {
        for (int u = 0; u < passingFeesSize; u++) {
            if (dp[u][t] == INT_MAX) continue;
            for (int i = 0; i < graphSize[u]; i++) {
                int v = graph[u][i].dest;
                int travelTime = graph[u][i].time;
                if (t + travelTime <= maxTime) {
                    dp[v][t + travelTime] = fmin(dp[v][t + travelTime], dp[u][t] + passingFees[v]);
                }
            }
        }
    }

    // Extract the result
    int minCost = INT_MAX;
    for (int t = 0; t <= maxTime; t++) {
        minCost = fmin(minCost, dp[passingFeesSize - 1][t]);
    }

    return (minCost == INT_MAX) ? -1 : minCost;
}

int main() {
```

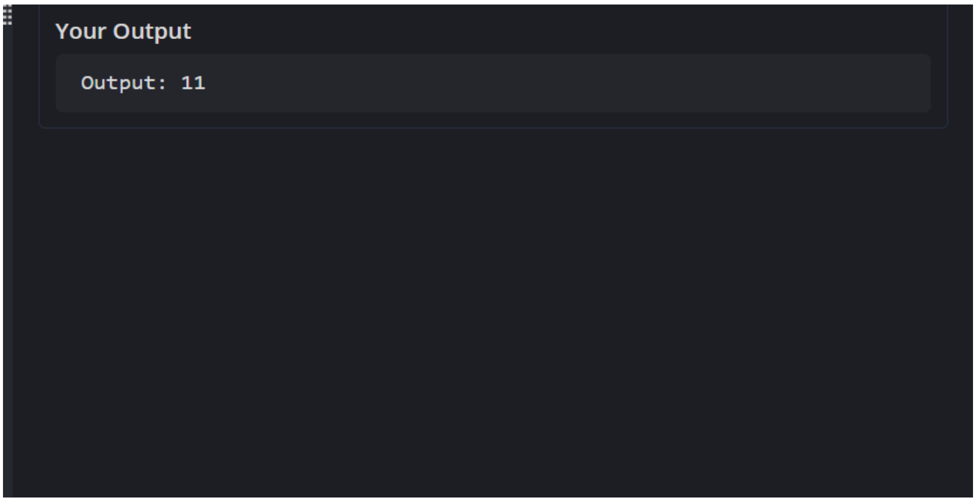
```
    int maxTime = 30;

    int edgesArray[][3] = {{0, 1, 10}, {1, 2, 10}, {2, 5, 10}, {0, 3, 1}, {3, 4, 10}, {4, 5, 15}};

    int edgesSize = sizeof(edgesArray) / sizeof(edgesArray[0]);

    int* edges[edgesSize];

    for (int i = 0; i < edgesSize; i++) {

        edges[i] = edgesArray[i];

    }

    int edgesColSize[] = {3, 3, 3, 3, 3, 3};

    int passingFees[] = {5, 1, 2, 20, 20, 3};

    int passingFeesSize = sizeof(passingFees) / sizeof(passingFees[0]);


    int result = minCost(maxTime, edges, edgesSize, edgesColSize, passingFees, passingFeesSize);

    printf("Result: %d\n", result);


    return 0;

}
```

EXPECTED OUTPUT:

Given the inputs:

- maxTime = 30
- edges = [[0, 1, 10], [1, 2, 10], [2, 5, 10], [0, 3, 1], [3, 4, 10], [4, 5, 15]]
- passingFees = [5, 1, 2, 20, 20, 3]

RESULT:



COMPLEXITY ANALYSIS:

- **Best Case:** O(n×maxTime), where n is the number of cities. This occurs when the optimal path is found early in the iteration.
- **Average Case:** O(n×maxTime), typical performance for dynamic programming with state transitions.
- **Worst Case:** O(n×maxTime) when all possible time states are used.

CONCLUSION:

The dynamic programming approach efficiently finds the minimum cost path within the allowed time by iteratively updating the cost for each city at each time step. This method guarantees that the minimum cost is found if a valid path exists within the given constraints. The solution is robust and scalable for a reasonable number of cities and maximum time, making it suitable for practical applications in route optimization problems.