# CAPSTONEPROJECT

# Minimum Cost to Reach Destination in Time

# CSA0695- Design and Analysis of Algorithms for Open Addressing Techniques

Kesavulu Y
(192210585)
Department of Computer Science and Engineering
Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences Saveetha University, Chennai, Tamil Nadu, India Pincode:602105.
yerremachettykesavulu0585.sse@saveetha.com

Dr R Dhanalakshmi,
Project guide, Corresponding Author, Department of Computer Science and Engineering, Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, Tamil Nadu, India. Pincode:602105.

## PROBLEM STATEMENT:

Given a country with `n` cities numbered from 0 to n-1, all connected by bi-directional roads, find the minimum cost to travel from the city `0` to city `n-1` within a given time limit `maxTime`. Roads between cities are described by a 2D array of `edges`, where each edge connects two cities and has a travel time. Additionally, each city has a passing fee, represented by `passing Fees`, which must be paid when passing through the city. The goal is to compute the minimum cost of passing fees to reach the destination city within the time limit or return `-1` if it is not possible.

## ABSTRACT:

The problem of finding the minimum cost to reach a destination city within a given time limit can be mapped to a graph traversal problem. Each city represents a node, and roads between cities are weighted edges. The weights on the edges correspond to the travel time, and each node has an associated cost represented by the passing fee. Using Dijkstra's algorithm or a similar shortest path algorithm that considers both time and cost constraints, we can compute the minimum total passing fee required to complete the journey within the given time.

## INTRODUCTION:

Traveling between cities while minimizing costs is a classical optimization problem in graph theory. In this scenario, each city has a passing fee, and roads between cities take a specific amount of time to travel. The objective is to find the most cost-efficient route to travel from the start city to the destination city while staying within a strict time limit.

This problem can be visualized as a graph where cities are nodes, roads are edges with weights representing travel time, and each node has an additional fee. The challenge lies in finding a path from city `0` to city `n-1` that minimizes the total passing fees and ensures the total time is within the given limit.

## Key aspects:

- Cities are connected by roads with varying travel times.
- Each city has an associated passing fee.
- The goal is to minimize the total cost (passing fees) while traveling within a time limit.

## SOLUTION APPROACH:

The problem can be tackled using a modified version of Dijkstra's algorithm or another shortest path algorithm that incorporates both time and cost constraints. The algorithm will prioritize nodes based on the accumulated passing fees while also ensuring that the travel time does not exceed the given limit.

### Step 1: Graph Construction:

We first represent the cities and roads as a graph. The edges array provides the travel times between cities, and each node (city) has an associated cost (passing fee).

### Step 2: Priority Queue for Minimum Cost Search:

We employ a priority queue to keep track of the minimum-cost route to each city, prioritizing cities that can be reached within the least cost. The algorithm processes each city in ascending order of passing fees and ensures that the total time spent on the journey is within the allowed `maxTime`.

### Step 3: Relaxation of Edges:

For each city, we explore all possible adjacent cities connected by roads. If the total time to reach an adjacent city is within the time limit and the cost is reduced, we update the cost and time for that city.

### Step 4: Termination and Result:

If the destination city `n-1` is reached within the given `maxTime`, the minimum cost is returned. If no such path exists, the function returns `-1`.

## Coding Implementation:

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

#define MAX_CITIES 1000
#define INF INT_MAX

// Structure to represent edges
typedef struct Edge {
    int city;
    int time;
    struct Edge* next;
} Edge;

typedef struct {
    int city;
    int time;
    int cost;
} State;

// Compare function for priority queue (min-heap)
int compare(const void* a, const void* b) {
    return ((State*)a)->cost - ((State*)b)->cost;
}

Edge* createEdge(int city, int time) {
    Edge* newEdge = (Edge*)malloc(sizeof(Edge));
    newEdge->city = city;
    newEdge->time = time;
    newEdge->next = NULL;
```

```
    return newEdge;
}

void addEdge(Edge* graph[], int from, int to, int time) {
    Edge* newEdge = createEdge(to, time);
    newEdge->next = graph[from];
    graph[from] = newEdge;
}

int minCostToReachDestination(int maxTime, int edges[][3], int numEdges, int
passingFees[], int numCities) {
    Edge* graph[MAX_CITIES] = {NULL};
    for (int i = 0; i < numEdges; i++) {
        addEdge(graph, edges[i][0], edges[i][1], edges[i][2]);
        addEdge(graph, edges[i][1], edges[i][0], edges[i][2]);
    }

    int minCost[MAX_CITIES][MAX_CITIES];
    for (int i = 0; i < numCities; i++) {
        for (int j = 0; j <= maxTime; j++) {
            minCost[i][j] = INF;
        }
    }
    minCost[0][0] = passingFees[0];

    State pq[MAX_CITIES * MAX_CITIES];
    int pqSize = 0;
    pq[pqSize++] = (State){0, 0, passingFees[0]};

    while (pqSize > 0) {
        State current = pq[0];
        pq[0] = pq[--pqSize];
        qsort(pq, pqSize, sizeof(State), compare);
if (current.city == numCities - 1) {
            return current.cost;
```

```
        }

        Edge* adj = graph[current.city];
        while (adj) {
            int newTime = current.time + adj->time;
            int newCost = current.cost + passingFees[adj->city];

            if (newTime <= maxTime && newCost < minCost[adj->city][newTime]) {
                minCost[adj->city][newTime] = newCost;
                pq[pqSize++] = (State){adj->city, newTime, newCost};
                qsort(pq, pqSize, sizeof(State), compare);
            }
            adj = adj->next;
        }
    }
    return -1;
}

int main() {
    int maxTime = 30;
    int edges[][3] = {{0, 1, 10}, {1, 2, 10}, {2, 5, 10}, {0, 3, 1}, {3, 4, 10}, {4, 5, 15}};
    int passingFees[] = {5, 1, 2, 20, 20, 3};
    int numEdges = sizeof(edges) / sizeof(edges[0]);
    int numCities = sizeof(passingFees) / sizeof(passingFees[0]);

    int result = minCostToReachDestination(maxTime, edges, numEdges, passingFees,
numCities);
    printf("Minimum cost: %d\n", result);

    return  0;
}
```

## OUTPUT:

For the input `maxTime = 30`, `edges = [[0,1,10],[1,2,10],[2,5,10],[0,3,1],[3,4,10],[4,5,15]]`, and passing Fees = [5, 1, 2, 20, 20, 3]`, the output is : **Minimum cost**: 11

## COMPLEXITY ANALYSIS:

**Time Complexity**: O(E log V), where E is the number of edges and V is the number of cities. The time complexity arises from processing each edge and updating the minimum cost using the priority queue (min-heap).

**Space Complexity**: O(E + V), where E is the number of edges and V is the number of cities. This includes space for the adjacency list and the priority queue.

## CONCLUSION:

The problem of finding the minimum cost to reach a destination city within a time limit can be effectively solved using a graph traversal algorithm like Dijkstra's with a priority queue. By balancing both time and cost constraints, the algorithm efficiently computes the most cost-effective route. This approach emphasizes the importance of priority-based pathfinding algorithms in solving real-world optimization problems.