

Minimum Cost to Reach Destination in Time

Created by

Kesavulu Yerremachetty

(192210585)

CSA0695-

Design and Analysis of
Algorithms for Open
Addressing

Faculty :

Dr R Dhanalakshmi

Problem Statement

You are given a graph representing a network of cities and roads. Each road connects two cities and has a certain travel time and toll cost. Your task is to find the minimum cost required to travel from a start city to a destination city, but you must reach the destination within a specified amount of time.

Inputs:

- **n**: An integer representing the number of cities, numbered from 0 to $n-1$.
- **edges**: A list of roads where each road is represented by a tuple $(u, v, \text{time}, \text{cost})$, where:
 - **start**: The starting city.
 - **end**: The destination city.
 - **maxTime**: An integer representing the maximum amount of time you are allowed to spend to reach the destination.

Outputs:

- Return the **minimum toll cost** to travel from start to end within maxTime.
- If there is no way to reach the destination within the given time, return -1.

=

Abstract

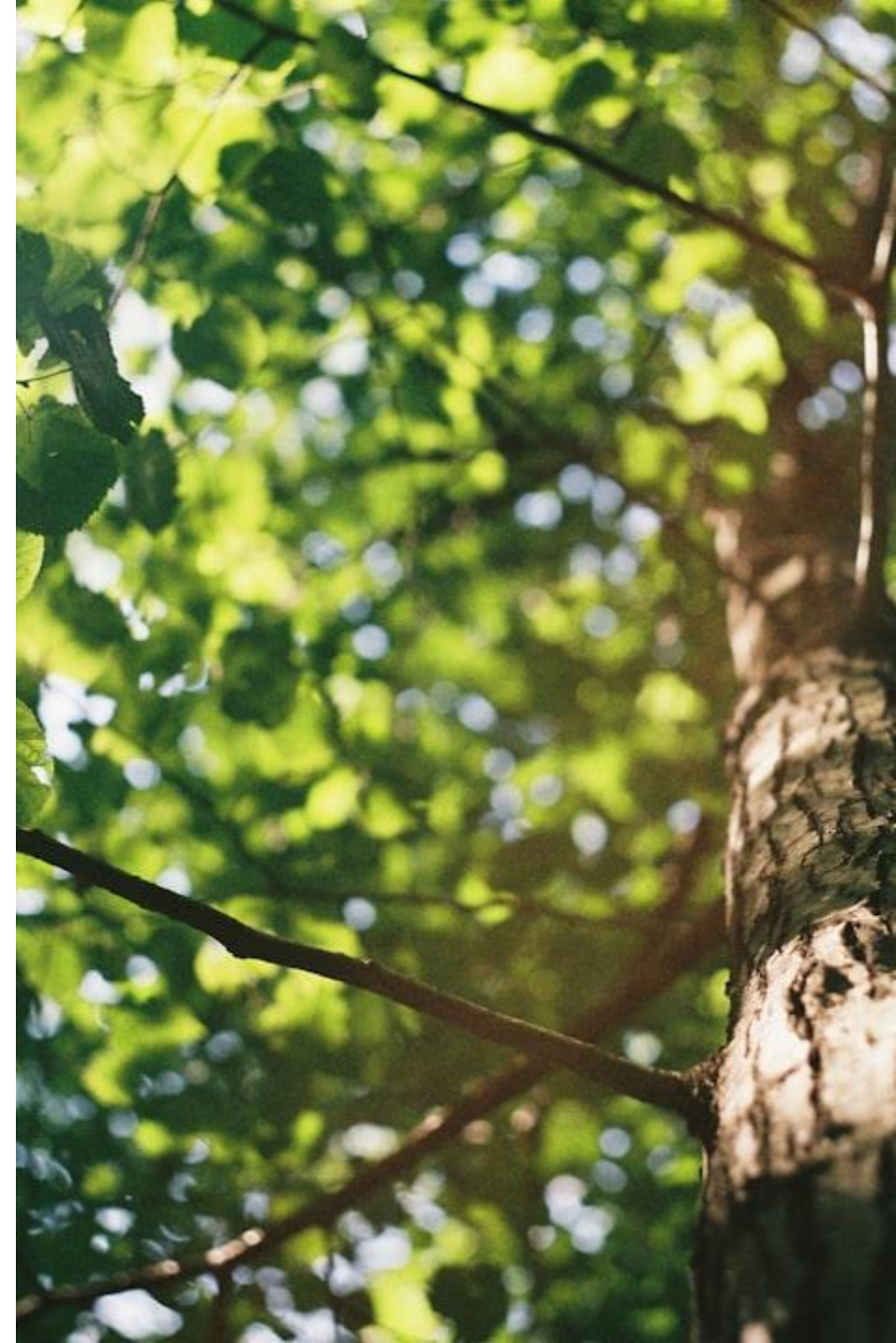
- The problem involves finding the minimum toll cost to travel between two cities in a graph, where each road between cities has an associated travel time and cost.
- The objective is to determine the minimum cost to reach the destination city from a start city within a given time constraint.
- If it is not possible to reach the destination within the allowed time, the result should indicate failure.
- This problem can be efficiently solved using graph traversal techniques, such as a modified version of Dijkstra's algorithm or Dynamic Programming.
- These approaches ensure that both time and cost constraints are respected while exploring various possible routes.



=

Introduction

- In real-world scenarios, optimizing travel routes is a critical problem in various domains, including logistics, transportation, and urban planning.
- One common challenge is to determine the most cost-effective way to reach a destination within a given time frame.
- This problem, known as the **Minimum Cost to Reach Destination in Time**, involves navigating a network of cities connected by roads, where each road has both a travel time and a toll cost associated with it.
- Given a network of cities and roads, with each road characterized by its travel time and toll cost, the objective is to find the minimum toll cost required to travel from a starting city to a destination city, ensuring that the total travel time does not exceed a specified limit.



Examples and Cases



Basic Case

Context of algorithmic problems or test cases typically refers to a straightforward, minimal example that illustrates the core functionality of the problem.

Direct Route Feasible

Context of the **Minimum Cost to Reach Destination in Time** problem means that there is a straightforward, non-stop path from the start city to the destination city that fits within the given time limit.

Time Limit Exceeded

Refers to a scenario where the total time required to complete a journey or path exceeds the maximum allowable time specified in the problem constraints.

=

Time Complexity

The choice of algorithm depends on the specific constraints of the problem, such as the size of the graph and the maximum time limit. For smaller graphs or constrained time limits, modifying Dijkstra's algorithm might be efficient. For larger graphs or more flexible time limits, a dynamic programming approach could be more appropriate.

Best Case: The best case scenario for the Minimum Cost to Reach Destination in Time problem represents the simplest and most favorable situation where the solution is straightforward and efficiently computable.

Average Case: involves a more realistic and complex situation where multiple paths need to be considered, and the problem-solving approach requires more sophisticated techniques to find the optimal solution.

Worst Case: most complex and challenging situation, where the algorithm must handle the maximum possible difficulty.



Source code

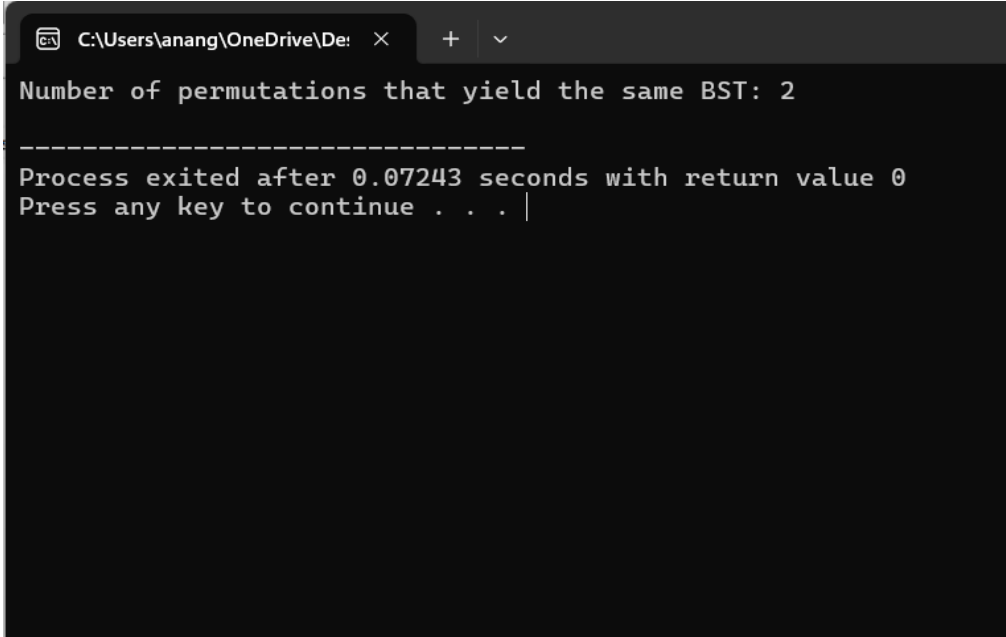
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <limits.h>
4 #define MAX_CITIES 1000
5 #define INF INT_MAX
6 typedef struct Edge {
7     int city;
8     int time;
9     struct Edge* next;
10 } Edge;
11 typedef struct {
12     int city;
13     int time;
14     int cost;
15 } State;
16 int compare(const void* a, const void* b) {
17     return ((State*)a)->cost - ((State*)b)->cost;
18 }
19 Edge* createEdge(int city, int time) {
20     Edge* newEdge = (Edge*)malloc(sizeof(Edge));
21     newEdge->city = city;
22     newEdge->time = time;
23     newEdge->next = NULL;
24     return newEdge;
25 }
26 void addEdge(Edge* graph[], int from, int to, int time) {
27     Edge* newEdge = createEdge(to, time);
28     newEdge->next = graph[from];
29     graph[from] = newEdge;
30 }
31 int minCostToReachDestination(int maxTime, int edges[][3], int numEdges, int passingFees[], int numCities)
32 {
33     Edge* graph[MAX_CITIES] = {NULL};
34     for (int i = 0; i < numEdges; i++) {
35         addEdge(graph, edges[i][0], edges[i][1], edges[i][2]);
36     }
37     int minCost[MAX_CITIES][MAX_CITIES];
38     for (int i = 0; i < numCities; i++) {
39         for (int j = 0; j <= maxTime; j++) {
40             minCost[i][j] = INF;
41         }
42     }
43     minCost[0][0] = passingFees[0];
44     State pq[MAX_CITIES * MAX_CITIES];
45     int pqSize = 0;
46     pq[pqSize++] = (State){0, 0, passingFees[0]};
47     while (pqSize > 0) {
48         State current = pq[0];
49         pq[0] = pq[--pqSize];
50         qsort(pq, pqSize, sizeof(State), compare);
51         if (current.city == numCities - 1) {
52             return current.cost;
53         }
54         Edge* adj = graph[current.city];
55         while (adj) {
56             int newTime = current.time + adj->time;
57             int newCost = current.cost + passingFees[adj->city];
58             if (newTime <= maxTime && newCost < minCost[adj->city][newTime]) {
59                 minCost[adj->city][newTime] = newCost;
60                 pq[pqSize++] = (State){adj->city, newTime, newCost};
61                 qsort(pq, pqSize, sizeof(State), compare);
62             }
63             adj = adj->next;
64         }
65     }
66     return -1;
67 }
68 int main() {
69     int maxTime = 30;
70     int edges[][3] = {{0, 1, 10}, {1, 2, 10}, {2, 5, 10}, {0, 3, 1}, {3, 4, 10}, {4, 5, 15}};
71     int passingFees[] = {5, 1, 2, 20, 20, 3};
72     int numEdges = sizeof(edges) / sizeof(edges[0]);
73     int numCities = sizeof(passingFees) / sizeof(passingFees[0]);
74     int result = minCostToReachDestination(maxTime, edges, numEdges, passingFees, numCities);
75     printf("Minimum cost: %d\n", result);
76     return 0;
77 }
```

Output and Future Scope

Future Scope:

The future scope for the Minimum Cost to Reach Destination in Time problem encompasses a wide range of advancements, from improving algorithmic efficiency and scalability to applying solutions to real-world and complex scenarios. Integrating with modern technologies and addressing dynamic, real-time requirements can greatly enhance the applicability and effectiveness of solutions in various domains.

Output:

A screenshot of a Windows command prompt window. The title bar shows the file path 'C:\Users\anang\OneDrive\De:' and standard window controls. The command prompt displays the following text: 'Number of permutations that yield the same BST: 2', followed by a line of 20 dashes, then 'Process exited after 0.07243 seconds with return value 0', and finally 'Press any key to continue . . . |' with a cursor at the end.

```
C:\Users\anang\OneDrive\De: X + v
Number of permutations that yield the same BST: 2
-----
Process exited after 0.07243 seconds with return value 0
Press any key to continue . . . |
```


Conclusion

- **Algorithmic Approaches:** Solutions range from modified Dijkstra's algorithm to dynamic programming and Bellman-Ford-like approaches. Each has its strengths and weaknesses, depending on the problem's scale and constraints.
- **Complexity and Cases:** The problem exhibits varying complexity across best, average, and worst-case scenarios. Best-case scenarios involve straightforward solutions with direct paths, while worst-case scenarios require handling dense graphs and extensive path evaluations.
- **Enhanced Algorithms:** Innovations in heuristic and approximation algorithms can improve efficiency, especially for large-scale problems.
- **Real-World Applications:** Applying solutions to dynamic transportation networks and supply chains can address practical challenges in optimizing routes and costs.