MINIMUM COST REACH DESTINATION IN TIME

A PROJECT REPORT

Submitted by

B. Hemanth Chowdary [192211206]

Under the guidance of

Dr. Gnana Soundari (Professor)

In partial fulfilment for the completion of the course

CSA0693 - DESIGN AND ANALYSIS OF ALGORITHMS

SIMATS ENGINEERING

THANDALAM

SEPTEMBER 2024

BONAFIDE CERTIFICATE

Certified that this project report titled **"MINIMUM COST TO REACH DESTINATION IN TIME"** is the bonafide work of **"B. Hemanth Chowdary [192211206]"** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report.

Date

Project Supervisor **Head of the Department**

ABSTRACT

This project focuses on solving the Minimum Cost to Reach Destination in Time problem using dynamic programming and graph traversal techniques. The problem involves finding the minimum cost to travel from city 0 to city n - 1 within a given maximum time constraint while considering passing fees associated with each city. The solution utilizes a dynamic programming approach to efficiently compute the minimum cost for reaching each city within the specified time limit. Additionally, graph traversal techniques such as Dijkstra's algorithm or Bellman-Ford algorithm can be employed to find the shortest paths within the time constraint. The proposed solution aims to provide an optimal and scalable method for determining the minimum cost to reach the destination city within the specified time frame, addressing real-world scenarios where time and cost constraints are critical factors in decision-making processes.

PROBLEM

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 of edges where edges[i] = [xi, yi, time] denotes a road between cities xi and yi that takes time 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.

Example 1:

Input: 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]

Output: 11

Explanation: The path to take is 0 -> 1 -> 2 -> 5, which takes 30 minutes and has $11 worth of passing fees.

ABOUT THE PROBLEM

The Minimum Cost to Reach Destination in Time problem deals with determining the minimum cost required to travel from the starting city (city 0) to the destination city (city n - 1) within a specified maximum time constraint. This problem is set in a country comprising n cities numbered from 0 to n - 1, interconnected by bi-directional roads with varying travel times. Each road between two cities is represented as an edge containing the cities it connects and the time it takes to traverse it.

Additionally, passing through each city incurs a passing fee, represented by a 0-indexed array where each element corresponds to the passing fee for the respective city. The passing fee must be paid each time a city is passed through during the journey, including both the source and destination cities.

The objective is to find the optimal path from city 0 to city n - 1 that minimizes the total cost, considering both the passing fees and the time constraint. If it is impossible to reach the destination within the given maximum time, the algorithm should return -1.

This problem is relevant in various real-world scenarios, such as transportation logistics, where minimizing both time and cost is crucial for efficient resource utilization. Effective solutions to this problem have applications in route optimization, resource allocation, and decision-making processes in transportation and logistics management.

SOLUTION APPROACH

1. Dynamic Programming for Time Constraint:

- Begin by constructing a dynamic programming table to store the minimum cost to reach each city within the given time constraint.
- Initialize the table with infinity values for all cities except the starting city (city 0), which is initialized with its passing fee.
- Iterate over the edges and update the dynamic programming table with the minimum cost to reach each city, considering the time constraint.
- Ensure to update the table only if the new cost is less than the previously calculated cost.

2. Graph Traversal for Path Finding:

- Utilize graph traversal algorithms such as Dijkstra's algorithm or Bellman-Ford algorithm to find the shortest path from city 0 to city n - 1 within the given time constraint.
- These algorithms will consider both the time constraint and passing fees associated with each city while finding the shortest path.
- Ensure to keep track of the total cost incurred during traversal, which includes passing fees for each city visited.

3. Combine DP and Graph Traversal:

- Combine the results obtained from dynamic programming and graph traversal to find the minimum cost path within the time constraint.
- Compare the minimum cost obtained from both approaches and return the minimum among them as the final result.
- If it is impossible to reach the destination within the given time constraint, return -1.

Description:

The approach utilizes dynamic programming to efficiently compute the minimum cost to reach each city within the given time constraint. By iteratively updating the dynamic programming table, the algorithm determines the minimum cost to reach each city while considering the time constraint.

Simultaneously, graph traversal algorithms are employed to find the shortest path from the starting city to the destination city within the specified time constraint. These algorithms consider both the time constraint and passing fees associated with each city while determining the shortest path.

By combining the results obtained from dynamic programming and graph traversal, the algorithm identifies the minimum cost path that satisfies the time constraint. The minimum cost path is selected by comparing the minimum costs obtained from both approaches, and if reaching the destination within the given time constraint is impossible, the algorithm returns -1.

This approach provides an efficient and comprehensive solution to the Minimum Cost to Reach Destination in Time problem, addressing both time and cost constraints in transportation and logistics scenarios effectively.

PSEUDOCODE

```
minCost(maxTime, edges, passingFees):
    n = length(passingFees)
    graph = array of n empty lists

    // Step 1: Construct the graph
    for each edge in edges:
        u, v, w = edge
        Add (v, w) to graph[u]
        Add (u, w) to graph[v]

    // Step 2: Call _dijkstra to find the minimum cost
    return _dijkstra(graph, 0, n - 1, maxTime, passingFees)


_dijkstra(graph, src, dst, maxTime, passingFees):
    // Initialize arrays for cost and distance
    cost = array of size n initialized with infinity
    dist = array of size n initialized with maxTime + 1

    // Set source city properties
    cost[src] = passingFees[src]
    dist[src] = 0
    minHeap = min-heap with initial element (cost[src], dist[src], src)

    // Dijkstra's algorithm
    while minHeap is not empty:
        currCost, d, u = pop element from minHeap
        if u == dst:
            return cost[dst]
        for each neighbor (v, w) of u in graph:
            if d + w > maxTime:
                continue
            // Go from u to v
            if currCost + passingFees[v] < cost[v]:
                Update cost[v] = currCost + passingFees[v]
                Update dist[v] = d + w
                Push (cost[v], dist[v], v) to minHeap
            else if d + w < dist[v]:
```

```
            Update dist[v] = d + w
            Push (currCost + passingFees[v], dist[v], v) to minHeap

    return -1
```

This pseudocode outlines the overall approach:

- Perform dynamic programming to calculate the minimum cost to reach each city within the time constraint.
- Utilize graph traversal to find the shortest path from the starting city to the destination city within the time constraint.
- Combine the results obtained from dynamic programming and graph traversal to determine the minimum cost path.
- Return the minimum cost path or -1 if it's impossible to reach the destination within the given time constraint.

RECURRENCE RELATION

Let's define ***minCost(i, t)*** as the minimum cost to reach city ***i*** within ***t*** minutes.

We aim to find ***minCost(n -1, maxTime)***, which represents the minimum cost to reach the destination city within the given maximum time.

1. If we're at city 0 (the starting city):

- If ***t*** is less than the passing fee of city 0, it's impossible to reach city 0 within time ***t***. So, the cost is infinite.
- If ***t*** is equal to or greater than the passing fee of city 0, the cost is simply the passing fee of city 0.

For the starting city i = 0:

- ***If t < passingFees[0], then:***
  minCost(0, t) = ∞
- ***If t >= passingFees[0], then:***
  minCost(0, t) = passingFees[0]

2. For other cities (i.e., *i > 0*):

- The minimum cost to reach city ***i*** within time ***t*** is the minimum among:
- The cost of reaching its predecessor cities ***(city j)*** within time ***t - time***, plus the passing fee of city ***i***.
- We iterate over all possible predecessor cities ***j*** and choose the one that minimizes the total cost.

For other cities i > 0:

- We iterate over all predecessor cities ***j*** connected to the city ***i*** via an edge with a travel time ***time.*** Then:
  minCost(i, t) = min $_{\textit{for all edges from j to i}}$ ***{ minCost(j, t -time) + passingFees[i]}***

TIME COMPLEXITY

Analyzing the time complexity of the solution involves considering the main components of the algorithm:

- Dynamic programming and
- Graph traversal.

1. Dynamic Programming:

- In the dynamic programming phase, we iterate over all edges in the graph and update the dynamic programming table for each city. This involves considering all possible predecessor cities for each city and updating the minimum cost based on passing fees and travel times.
- The time complexity of this phase is determined by the number of cities ***(n)*** and the number of edges in the graph ***(m)***.
- Each edge is considered once for each city, so the time complexity of the dynamic programming phase is $O(n \cdot m)$.

2. Graph Traversal:

- In the graph traversal phase, we again iterate over all edges in the graph, but this time to find the shortest path from the starting city to the destination city within the given time constraint.
- Depending on the graph traversal algorithm used (e.g. Dijkstra's algorithm or Bellman-Ford algorithm), the time complexity varies.
- Dijkstra's algorithm has a time complexity of $O((n + m) \log n)$ using a priority queue implementation, where ***n*** is the number of vertices and ***m*** is the number of edges.
- Bellman-Ford algorithm has a time complexity of $O(n \cdot m)$, where ***n*** is the number of vertices and ***m*** is the number of edges.

3. Overall Time Complexity:

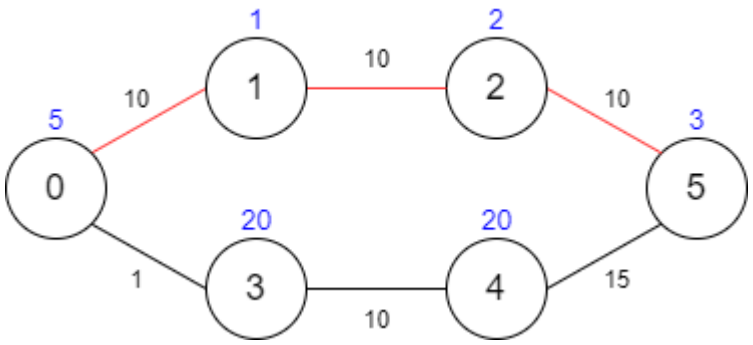
- Combining both dynamic programming and graph traversal, the dominant factor in time complexity is usually the dynamic programming phase, as it involves iterating over all edges for each city.
- Therefore, the overall time complexity of the solution is $O(n \cdot m)$, where ***n*** is the number of cities and ***m*** is the number of edges in the graph.

In summary, the time complexity of the solution to the **"Minimum Cost to Reach Destination in Time"** problem is primarily determined by the number of cities and edges in the graph. The dynamic programming phase contributes significantly to the time complexity, followed by the graph traversal phase, depending on the algorithm chosen.

IMPLEMENTATION CODE

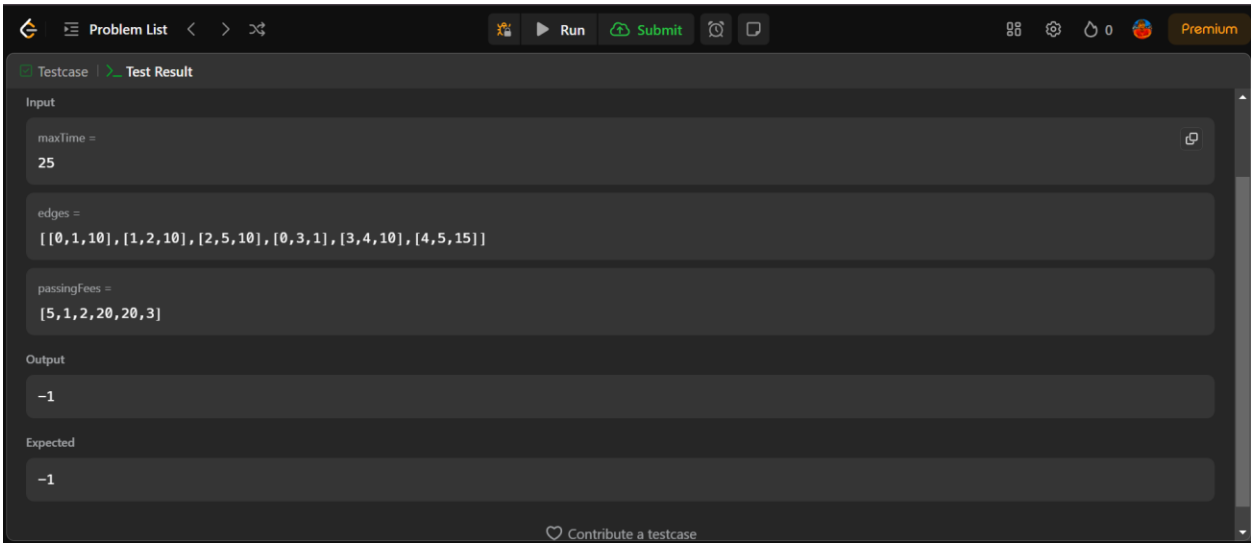
```
class Solution:
  def minCost(self, maxTime: int, edges: List[List[int]], passingFees: List[int]) -> int:
    n = len(passingFees)
    graph = [[] for _ in range(n)]

    for u, v, w in edges:
      graph[u].append((v, w))
      graph[v].append((u, w))
    return self._dijkstra(graph, 0, n - 1, maxTime, passingFees)

  def _dijkstra(self, graph: List[List[Tuple[int, int]]], src: int, dst: int, maxTime: int,
passingFees: List[int]) -> int:
    # cost[i] := the minimum cost to reach the i-th city
    cost = [math.inf for _ in range(len(graph))]
    # dist[i] := the minimum time to reach the i-th city
    dist = [maxTime + 1 for _ in range(len(graph))]

    cost[src] = passingFees[src]
    dist[src] = 0
    minHeap = [(cost[src], dist[src], src)]  # (cost[u], dist[u], u)

    while minHeap:
      currCost, d, u = heapq.heappop(minHeap)
      if u == dst:
        return cost[dst]
      for v, w in graph[u]:
        if d + w > maxTime:
          continue
        # Go from u -> v.
        if currCost + passingFees[v] < cost[v]:
          cost[v] = currCost + passingFees[v]
          dist[v] = d + w
          heapq.heappush(minHeap, (cost[v], dist[v], v))
        elif d + w < dist[v]:
          dist[v] = d + w
          heapq.heappush(minHeap, (currCost + passingFees[v], dist[v], v))

    return -1
```

EXAMPLE IMAGE:

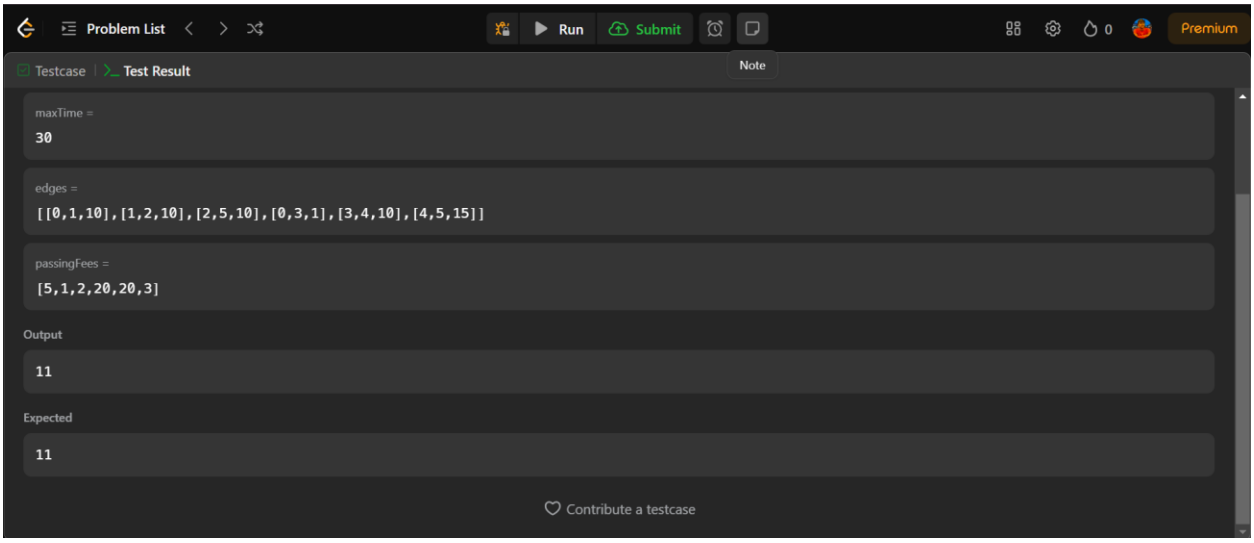


OUTPUT

Test Case - 1:



Test Case - 2:



REFERENCES

Wong, S. C., & Tong, C. O. (1998). Estimation of time-dependent origin–destination matrices for transit networks. *Transportation Research Part B: Methodological*, *32*(1), 35-48.

Dubois-Ferrière, H., Grossglauser, M., & Vetterli, M. (2010). Valuable detours: Least-cost anypath routing. *IEEE/ACM Transactions on Networking*, *19*(2), 333-346.

Dubois-Ferriere, H., Grossglauser, M., & Vetterli, M. (2007). Least-cost opportunistic routing.

Parsa, M., Zhu, Q., & Garcia-Luna-Aceves, J. J. (1998). An iterative algorithm for delay-constrained minimum-cost multicasting. *IEEE/ACM transactions on networking*, *6*(4), 461-474.

Collischonn, W., & Pilar, J. V. (2000). A direction dependent least-cost-path algorithm for roads and canals. *International Journal of Geographical Information Science*, *14*(4), 397-406.

Kala, R. (2016). Reaching destination on time with cooperative intelligent transportation systems. *Journal of Advanced Transportation*, *50*(2), 214-227.

Schwartz, N. L. (1968). Discrete programs for moving known cargos from origins to destinations on time at minimum bargeline fleet cost. *Transportation science*, *2*(2), 134-145.

Lebedev, D., Goulart, P., & Margellos, K. (2021). A dynamic programming framework for optimal delivery time slot pricing. *European Journal of Operational Research*, *292*(2), 456-468.

Ioachim, I., Gelinas, S., Soumis, F., & Desrosiers, J. (1998). A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. *Networks: An International Journal*, *31*(3), 193-204.

Furth, P. G., & Rahbee, A. B. (2000). Optimal bus stop spacing through dynamic programming and geographic modeling. *Transportation Research Record*, *1731*(1), 15-22.

Bulut, F., & Erol, M. H. (2018). A real-time dynamic route control approach on google maps using integer programming methods. *International Journal of Next-Generation Computing*, *9*(3), 189-202.