Algorithm

Name: Koushik Roy
Id: 0122230054

Table of Contents

Question 1 Solution

Problem Statement

Given a city map with junctions and roads, where each junction has a traffic light that alternates between green and red with a fixed period, the task is to find the minimum time required to travel from a given source junction to a given destination junction, considering the traffic light timings.

Approach

The problem can be solved using a modified version of Dijkstra's algorithm. The algorithm is modified to consider the waiting time due to traffic lights at each junction.

Algorithm

1. Create a graph to represent the city map with junctions and roads.
2. Populate the graph with road information.
3. Initialize a visited set to keep track of visited junctions.
4. Initialize a dictionary to store the minimum time taken to reach each junction.

5. Initialize a priority queue to efficiently find the junction with the minimum time.
6. Set the minimum time of the source junction to 0 and add it to the priority queue.
7. Process the priority queue until it is empty.
8. Pop the junction with the minimum time from the priority queue.
9. If the current junction is the destination, return the minimum time.
10. If the current junction has already been visited, skip it.
11. Mark the current junction as visited.
12. Process each neighboring junction of the current junction.
13. Calculate the waiting time at the current junction's traffic light.
14. Calculate the total time to reach the neighboring junction.
15. If the calculated time is less than the current minimum time for the neighboring junction, update the minimum time and add the junction to the priority queue.
16. If the destination junction is not reached, return -1 (or an appropriate value indicating no path found).

Explanation

1. A graph is created to represent the city map with junctions and roads. The graph is created using an adjacency list, where each junction is a key and its value is a list of tuples containing the neighboring junction and the distance to it.
2. The graph is populated with road information using the input data.
3. A visited set is initialized to keep track of visited junctions. This set is used to avoid revisiting the same junction multiple times.
4. A dictionary is initialized to store the minimum time taken to reach each junction. The key is the junction number, and the value is initialized to infinity, except for the source junction, which is initialized to 0.
5. A priority queue is initialized to efficiently find the junction with the minimum time. The priority queue is implemented using a heap data structure, where each element is a tuple containing the time and the junction number.
6. The minimum time of the source junction is set to 0 and added to the priority queue.

7. The priority queue is processed until it is empty. This loop extracts the junction with the minimum time from the priority queue and processes its neighboring junctions.
8. The junction with the minimum time is popped from the priority queue.
9. If the current junction is the destination, the minimum time is returned.
10. If the current junction has already been visited, it is skipped.
11. The current junction is marked as visited.
12. Each neighboring junction of the current junction is processed.
13. The waiting time at the current junction's traffic light is calculated by finding the remainder of the current time divided by the traffic light period and subtracting it from the period.
14. The total time to reach the neighboring junction is calculated by adding the distance to the neighboring junction, the waiting time at the current junction, and the current time.
15. If the calculated time is less than the current minimum time for the neighboring junction, the minimum time is updated and the neighboring junction is added to the priority queue.
16. If the destination junction is not reached, -1 (or an appropriate value indicating no path found) is returned.

Question 2 Solution

Problem Statement

The problem describes a land called Marvel with N cities connected by M bidirectional roads. Dr. Strange has to visit a sequence of cities (A1 to AK) to spread a secret magic. He travels from one city to another through the shortest path between them. The government wants to track which cities Dr. Strange targets, but they do not know the sequence A. Instead, they have tracked Dr. Strange's movement as a sequence of cities visited by him during his mission (B1 to BL). Dr. Strange's sequence A is a subsequence of B, and the problem is to find the minimum possible number of targeted cities, or determine that no such sequence A exists.

Algorithm

1. Parse the input for the number of test cases (T).
2. For each test case, perform the following steps:
   a. Parse the input for the number of cities (N), number of roads (M), and the number of cities visited by Dr. Strange (L).
   b. Parse the input for the sequence of visited cities (B) and the road information (roads).
   c. Create an empty graph (graph) to represent the city map with junctions and roads.
   d. Populate the graph with road information.
3. For each test case, find the minimum possible number of targeted cities by following these steps:
   a. Initialize an empty list (targeted) to store the targeted cities.
   b. Initialize two pointers, left and right, to iterate through the sequence of visited cities (B).
   c. Calculate the shortest distance between the cities pointed by left and right pointers using Dijkstra's algorithm.
   d. Check the current shortest distance and update the distance list and flag accordingly. Three type of scenario could happen here: Current distance between left and right is less, more or equal to the last value of distance list. My main idea here is to find out when the distance between left and right becomes less than the last distance in the distance list. That is the point I find a targeted city and move the left pointer in search of the next targeted city.
   e. Based on the value of the flag, update the targeted list and pointers. Here, the assignment asked to find the minimum number of targeted city. So, in case to distance list, when I find current distance is higher than last then I consider the last city as a stopover city and not the actual targeted city. Then again when the current distance is less than last then I consider the last city to be targeted and I start finding the next targeted city by updating left and right pointer.
   f. If the targeted list has only one city, return -1. Otherwise, return the number of targeted cities.
4. Print the output for each test case.