Transportation Hub

Date:2024-04-24

Chapter 1:Introduction

Problem Description

The problem is essentially about identifying key nodesin a graph which are traversed by multiple shortest paths between various pairs of nodes (cities). The graph is undirected, with nodes representing cities and edges representing roads between those cities. The weight on each edge corresponds to the length of the road. For each query (which specifies a source and a destination city), the task is to determine which cities qualify as transportation hubs based on the number of times they appear on any shortest path between the source and destination, excluding the source and destination themselves.

Background of the Algorithms

1. **Shortest Path Algorithm (Dijkstra's Algorithm)**:

This is a fundamental graph algorithm used to find the shortest paths from a given source vertex to all other vertices in a graph with non-negative edge weights.

The algorithm works by iteratively expanding the closest vertex not yet processed until all vertices have been processed or the shortest distances to all reachable vertices are known.

It utilizes a priority queue to efficiently fetch the next vertex with the smallest distance.

2. **Depth-First Search (DFS) for Path Counting**:

Once the shortest path distances are computed, DFS can be used to explore all possible paths from the source to the destination that match the shortest path length. During this exploration, nodes are counted whenever they appear in such paths.

DFS is adapted here to only follow paths that are part of the shortest routes from the source to the destination. This ensures that only relevant paths contribute to the hub determination.

3. **Path Counting Strategy**:

For each city on a path that is part of a shortest route (determined by DFS using the results of Dijkstra's algorithm), a count is maintained.

Post-DFS, any city that appears in these counts at least k times (as specified by the problem) and is not the source or destination is labeled a transportation hub.

Chapter 2: Algorithm Specification

Main Data Structures:

1. **Graph Representation**:

Adjacency List: **graph[city1][city2] = length**

Where city1 and city2 are cities connected by a road of given length.

This representation is efficient for sparse graphs and is helpful for graph traversal algorithms like Dijkstra's and DFS.

2. **Distance Array**:

Array: dist[n]

Where dist[i] holds the shortest distance from the current source city to city i.

3. **Path Count Array**:

Array: path_count[n]

Where path_count[i] keeps track of how many times city i appears on shortest paths between queried source and destination cities.

4. **Visited Nodes Tracker**:

Array: visited[n]

Helps to track which cities have been visited during DFS to avoid cycles and redundant paths.

Algorithms:

Dijkstra's Algorithm (Pseudo-code)

```
function dijkstra(src, n, graph, dist)
        for i = 0 to n-1
            dist[i] = INT_MAX
            visited[i] = false
        dist[src] = 0

        for count = 0 to n-1
            u = findMinVertex(dist, visited, n)
            visited[u] = true
            for v = 0 to n-1
```

```
                    if !visited[v] and graph[u][v] > 0 and dist[u] + graph[u][v] <
dist[v]
                        dist[v] = dist[u] + graph[u][v]

    function findMinVertex(dist, visited, n)
        min = INT_MAX
        min_index = -1
        for v = 0 to n-1
            if !visited[v] and dist[v] < min
                min = dist[v]
                min_index = v
        return min_index
```

Depth-First Search for Path Counting (Pseudo-code)

```
function dfs(current, dest, path_length, n, graph, visited, dist, path_count)
    if current == dest
        if path_length == dist[dest]
            for i = 0 to n-1
                if visited[i]
                    path_count[i] += 1
        return
    visited[current] = true
    for i = 0 to n-1
```

```
            if !visited[i] and graph[current][i] > 0 and path_length + graph[current][i]
== dist[i]
            dfs(i, dest, path_length + graph[current][i], n, graph, visited,
dist,path_count)
        visited[current] = false
```

Chapter 3: Testing Results

Sample Input/Output 1: Basic Functionality Test

Input:

10 16 2

1 2 1

1 3 1

1 4 2

2 4 1

2 5 2

3 4 1

3 0 1

4 5 1

4 6 2

5 6 1

7 3 2

7 8 1

7 0 3

8 9 1

9 0 2

0 6 2

3

1 6

7 0

5 5

Output:

2 3 4 5

None

None

Purpose:

This test verifies that the program can correctly identify cities that appear in at least k shortest paths between given source and destination pairs.

Query 1 (0 to 4): Cities 2 and 3 are on at least 2 shortest paths.

Query 2 (2 to 4): Only city 3 meets the condition.

Query 3 (1 to 4): There are no transportation hubs as there is only a direct path from 1 to 4, with no intermediate nodes being used more than once.

Status:Passed

Sample Input/Output 2: Boundary Condition Test

Input:

3 3 2

0 1 1

1 2 2

0 2 3

1

0 2

Output:

None

Purpose:

This test checks the program's response to a graph where potential hub cities do not meet the threshold k despite multiple paths existing.

It tests the edge case where cities are part of multiple paths but none are used sufficiently frequently to qualify as hubs.

Status:Passed

Sample Input/Output 3: Complex Connectivity Graph

Input:

6 8 1

0 1 7

0 2 9

0 5 14

1 2 10

1 3 15

2 3 11

2 5 2

3 4 6

4

0 4

2 4

0 3

1 4

Output:

2 3

3

2

3

Purpose:

Tests the program's ability to handle a graph with a dense network of paths and correctly compute hub status across multiple queries.

Ensures that the algorithm consistently identifies hubs across different source-

destination pairs in a complex network.

Status:Passed

Chapter 4: Analysis and Comments

Time and Space Complexity Analysis

Dijkstra's Algorithm

Time Complexity: The time complexity of Dijkstra's algorithm when implemented using an adjacency matrix and a simple linear array to find the minimum distance vertex is O(n^2), where n is the number of cities or nodes in the graph. This complexity arises because each node is extracted from the priority queue (or checked in an array for the minimum distance) once, and for each node, all other nodes are inspected to update distances.

Space Complexity: The space complexity is O(n) for the dist array, O(n) for the visited array, plus O(n^2) for the adjacency matrix, totaling O(n^2).

Depth-First Search (DFS)

Time Complexity: In the worst case, DFS explores every vertex and every edge coming out of these vertices once, leading to a complexity of O(n+m), where mm is the number of edges. However, since the DFS is used to find all paths from source to destination matching the shortest path criteria, its complexity might approach O(n×m) in dense graphs due to repeated explorations of paths.

Space Complexity: The space usage primarily comes from the recursion stack

(which in the worst case could be as deep as n), the visited array, and the path_count array, all of which contribute to a complexity of O(n).

Constructing Output for Hubs

Time Complexity: Simply iterating through the list of nodes to check path counts against the threshold k, which is O(n).

Space Complexity: This requires O(n) space to maintain the path_count and other auxiliary data structures for storing nodes meeting hub criteria.

Possible Improvements

1. **Efficient Graph Representations**:

Switch from an adjacency matrix to an adjacency list, which can save significant space and reduce the complexity of operations in sparse graphs. The adjacency list is particularly advantageous when the graph has far fewer edges than n2.

2. **Optimized Priority Queue:**

Implement Dijkstra's algorithm using a binary heap (or other efficient priority queue structures like Fibonacci heaps) to decrease the time complexity of the priority queue operations to $O((n+m)\log n)$, which is beneficial especially for dense graphs.

3. **Algorithmic Improvements for Specific Graph Types:**

Adapt algorithms based on the characteristics of the graph applying specialized algorithms that can handle such cases more efficiently.

Declaration:I hereby declare that all the work done in this project titled "

Transportation Hub " is of my independent effort.