Shortest Distance After Road Addition Queries I

solved by	Senan
Platform	LeetCode
difficulty	Medium
# Serial	3243
tags	Dijkstras Algorithm
language	C++
solved on	@27/11/2024
link	https://leetcode.com/problems/shortest-distance-after-road-addition-queries-i/description/
Completion	☑

Intuition

This problem requires finding the shortest path in a graph with edges that may change after each query. The goal is to handle the changes efficiently and compute the shortest path from a starting node to the last node for each query.

Approach

1. **Graph Representation**: Use an adjacency list to represent the graph. Each query dynamically adds an edge to the graph.
2. **Dijkstra's Algorithm**: Implement Dijkstra's algorithm using a `set` to maintain the current shortest distance and node. This ensures we always process the node with the smallest distance first.
3. **Edge Updates**: For each query, update the adjacency list and recompute the shortest path from node `0` to node `n-1`.
4. **Return Results**: Store the result of each shortest path calculation and return it as a vector.

Complexity

Time Complexity:

- **Dijkstra's Algorithm**: **O((V+E)logV)** per execution. Here, **V** is the number of vertices and **E** is the number of edges.
- **Graph Updates**: **O(Q)** for **Q** queries where each update takes **O(1)**.

Since Dijkstra is run for every query, the total complexity becomes **O(Q×(V+E)logV)**.

Space Complexity:

- **Graph Storage**: **O(V + E)** for the adjacency list.
- **Distance Array**: **O(V)** for storing distances.
- **Set Storage**: **O(V)**.

Overall space complexity is **O(V+E)**.

Code

```
class Solution {
public:
    vector<int> dijkstra(int V, vector<vector<int>>& adj, int S) {
        vector<int> dist(V, 1e9);
        set<pair<int, int>> st;

        st.insert({0, S});
        dist[S] = 0;

        while (!st.empty()) {
            auto it = *(st.begin());
            int node = it.second;
            int dis = it.first;
            st.erase(it);

            for (auto adjNode : adj[node]) {
                int edgW = 1;
                if (dis + edgW < dist[adjNode]) {
                    if (dist[adjNode] != 1e9)
                        st.erase({dist[adjNode], adjNode});

                    dist[adjNode] = dis + edgW;
                    st.insert({dist[adjNode], adjNode});
                }
            }
        }

        return dist;
    }

    vector<int> shortestDistanceAfterQueries(int n, vector<vector<int>>& queries) {
        vector<int> ans;
        vector<vector<int>> adj(n);

        for (int i = 0; i < n - 1; i++)
            adj[i].push_back(i + 1);

        for (auto it : queries) {
            adj[it[0]].push_back(it[1]);
            auto temp = dijkstra(n, adj, 0);
            ans.push_back(temp[n - 1]);
        }
        return ans;
    }
};
```