6442. Modify Graph Edge Weights

My Submissions (/contest/weekly-contest-346/problems/modify-graph-edge-weights/submissions/) Back to Contest (/contest/weekly-contest-346/)

User Accepted:	0
User Tried:	1
Total Accepted:	0
Total Submissions:	1
Difficulty:	Hard

You are given an **undirected weighted connected** graph containing `n` nodes labeled from `0` to `n - 1`, and an integer array `edges` where `edges[i] = [`a_i`, `b_i`, `w_i`]` indicates that there is an edge between nodes a_i and b_i with weight w_i.

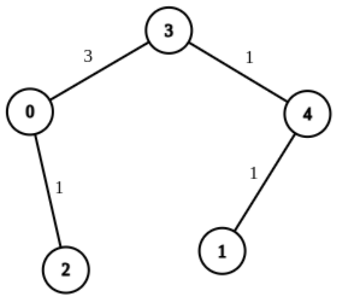
Some edges have a weight of `-1` ($w_i = -1$), while others have a **positive** weight ($w_i > 0$).

Your task is to modify **all edges** with a weight of `-1` by assigning them **positive integer values** in the range $[1, 2 * 10^9]$ so that the **shortest distance** between the nodes `source` and `destination` becomes equal to an integer `target`. If there are **multiple modifications** that make the shortest distance between `source` and `destination` equal to `target`, any of them will be considered correct.

Return *an array containing all edges (even unmodified ones) in any order if it is possible to make the shortest distance from* `source` *to* `destination` *equal to* `target`*, or an* ***empty array*** *if it's impossible.*

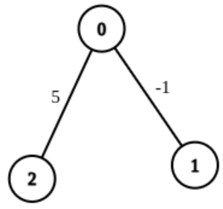
Note: You are not allowed to modify the weights of edges with initial positive weights.

Example 1:



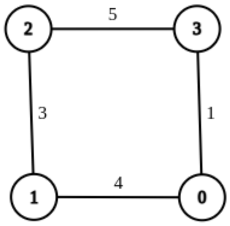
```
Input: n = 5, edges = [[4,1,-1],[2,0,-1],[0,3,-1],[4,3,-1]], source = 0, destination = 1, target = 5
Output: [[4,1,1],[2,0,1],[0,3,3],[4,3,1]]
Explanation: The graph above shows a possible modification to the edges, making the distance from 0 to 1 equal to 5.
```

Example 2:



```
Input: n = 3, edges = [[0,1,-1],[0,2,5]], source = 0, destination = 2, target = 6
Output: []
Explanation: The graph above contains the initial edges. It is not possible to make the distance from 0 to 2 equal to 6 by modi
```

Example 3:



```
Input: n = 4, edges = [[1,0,4],[1,2,3],[2,3,5],[0,3,-1]], source = 0, destination = 2, target = 6
Output: [[1,0,4],[1,2,3],[2,3,5],[0,3,1]]
Explanation: The graph above shows a modified graph having the shortest distance from 0 to 2 as 6.
```

Constraints:

- `1 <= n <= 100`
- `1 <= edges.length <= n * (n - 1) / 2`
- `edges[i].length == 3`
- `0 <=` a_i, b_i `< n`
- w_i `= -1` or `1 <=` w_i `<=` 10^7
- a_i `!=` b_i
- `0 <= source, destination < n`
- `source != destination`
- `1 <= target <=` 10^9
- The graph is connected, and there are no self-loops or repeated edges

JavaScript

```
const initializeGraphMap = (n) => { let g = []; for (let i = 0; i < n; i++) { g.push(new Map()); } return g; };
const packUGCost = (g, edges) => {
    for (let i = 0; i < edges.length; i++) {
        let [u, v, cost] = edges[i];
        if (cost == -1) {
            canModify.push([u, v, cost]);
            edges[i][2] = 2e9;
        }
        g[u].set(v, edges[i][2])
        g[v].set(u, edges[i][2])
    }
};


////////////////////////////////////////////////////////////////////////////////////////////////////////
let m, n, canModify;
const modifiedGraphEdges = (N, edges, start, dest, target) => {
    m = new Map(), n = N, canModify = [];
    let g = initializeGraphMap(n);
    packUGCost(g, edges);
    let d = dijkstra(g, start)
    if (d[dest] == target) return go(g, edges);
    if (d[dest] < target) return [];
    for (const [u, v, cost] of canModify) {
        g[u].set(v, 1);
        g[v].set(u, 1);
        let d = dijkstra(g, start);
        if (d[dest] <= target) {
            let gap = target - d[dest];
            let pre = g[u].get(v), update = pre + gap;
            g[u].set(v, update);
            g[v].set(u, update);
            return go(g, edges);
        }
    }
    return [];
```

```javascript
};

const dijkstra = (g, start) => {
    let n = g.length, dis = Array(n).fill(Number.MAX_SAFE_INTEGER);
    let pq = new MinPriorityQueue({
        compare: (x, y) => {
            if (x[0] != y[0]) return x[0] - y[0];
            return x[1] - y[1];
        }
    });
    dis[start] = 0;
    pq.enqueue([0, start]);
    while (pq.size()) {
        let [d, cur] = pq.dequeue();
        if (d > dis[cur]) continue;
        for (const [child, cost] of g[cur]) {
            let toChildCost = d + cost;
            if (toChildCost < dis[child]) {
                dis[child] = toChildCost;
                pq.enqueue([toChildCost, child]);
            }
        }
    }
    return dis; // min distance: start -> all other nodes
};

const go = (g) => {
    let res = new Set();
    for (let i = 0; i < n; i++) {
        for (const [child, cost] of g[i]) {
            res.add(JSON.stringify([Math.min(i, child), Math.max(i, child), cost]))
        }
    }
    return [...res].map(e => JSON.parse(e));
};
```

□ **Custom Testcase** Use Example Testcases

Run Submit

Submission Result: Accepted (/submissions/detail/954376520/) More Details > (/submissions/detail/954376520/)

Share your acceptance!

Copyright © 2023 LeetCode

Help Center (/support) | Jobs (/jobs) | Bug Bounty (/bugbounty) | Online Interview (/interview/) | Students (/student) | Terms (/terms) | Privacy Policy (/privacy)

United States (/region)