6232. Minimum Total Distance Traveled

My Submissions (/contest/weekly-contest-318/problems/minimum-total-distance-traveled/submissions/) Back to Contest (/contest/weekly-contest-318/)

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

There are some robots and factories on the X-axis. You are given an integer array `robot` where `robot[i]` is the position of the i^{th} robot. You are also given a 2D integer array `factory` where $factory[j] = [position_j, limit_j]$ indicates that $position_j$ is the position of the j^{th} factory and that the j^{th} factory can repair at most $limit_j$ robots.

The positions of each robot are **unique**. The positions of each factory are also **unique**. Note that a robot can be **in the same position** as a factory initially.

All the robots are initially broken; they keep moving in one direction. The direction could be the negative or the positive direction of the X-axis. When a robot reaches a factory that did not reach its limit, the factory repairs the robot, and it stops moving.

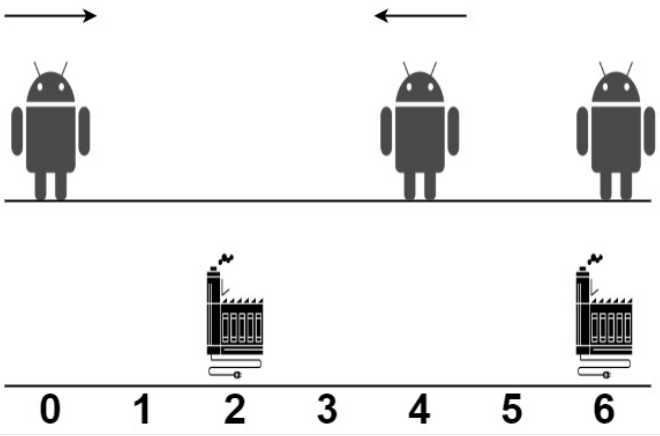
At any moment, you can set the initial direction of moving for **some** robot. Your target is to minimize the total distance traveled by all the robots.

Return *the minimum total distance traveled by all the robots*. The test cases are generated such that all the robots can be repaired.

Note that

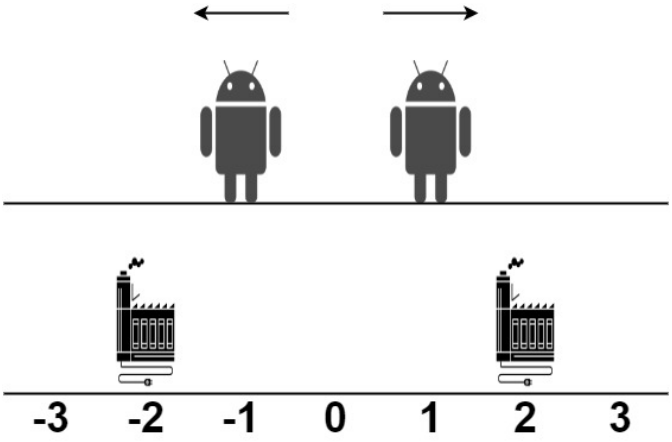
- All robots move at the same speed.
- If two robots move in the same direction, they will never collide.
- If two robots move in opposite directions and they meet at some point, they do not collide. They cross each other.
- If a robot passes by a factory that reached its limits, it crosses it as if it does not exist.
- If the robot moved from a position `x` to a position `y`, the distance it moved is `|y - x|`.

Example 1:




```
Input: robot = [0,4,6], factory = [[2,2],[6,2]]
Output: 4
Explanation: As shown in the figure:
- The first robot at position 0 moves in the positive direction. It will be repaired at the first factory.
- The second robot at position 4 moves in the negative direction. It will be repaired at the first factory.
- The third robot at position 6 will be repaired at the second factory. It does not need to move.
The limit of the first factory is 2, and it fixed 2 robots.
The limit of the second factory is 2, and it fixed 1 robot.
The total distance is |2 - 0| + |2 - 4| + |6 - 6| = 4. It can be shown that we cannot achieve a better total distance than 4.
```

Example 2:



Input: robot = [1,-1], factory = [[-2,1],[2,1]]
Output: 2
Explanation: As shown in the figure:
- The first robot at position 1 moves in the positive direction. It will be repaired at the second factory.
- The second robot at position -1 moves in the negative direction. It will be repaired at the first factory.
The limit of the first factory is 1, and it fixed 1 robot.
The limit of the second factory is 1, and it fixed 1 robot.
The total distance is |2 - 1| + |(-2) - (-1)| = 2. It can be shown that we cannot achieve a better total distance than 2.

Constraints:

- 1 <= robot.length, factory.length <= 100
- factory[j].length == 2
- -10^9 <= robot[i], position$_j$ <= 10^9
- 0 <= limit$_j$ <= robot.length
- The input will be generated such that it is always possible to repair every robot.

JavaScript

```
function edge(from, to, cost, cap) {
    this.from = from;
    this.to = to;
    this.cost = cost;
    this.cap = cap;
}

function MCMF(n) {
    const initializeGraph = (n) => { let g = []; for (let i = 0; i < n; i++) { g.push([]); } return g; };
    let g = initializeGraph(n), h = Array(n).fill(0), dis = Array(n).fill(0), prev_v = Array(n).fill(0), prev_e =
Array(n).fill(0);
    return { addEdge, minCostFlow }
    function addEdge(from, to, cost, cap) {
        g[from].push(new edge(g[to].length, to, cost, cap));
        g[to].push(new edge(g[from].length - 1, from, -cost, 0));
    }
    function minCostFlow(from, to, flow) {
        let res = 0;
        while (flow > 0) {
            let pq = new MinPriorityQueue({
                compare: (x, y) => {
                    if (x[0] != y[0]) return x[0] - y[0];
                    return x[1] - y[1];
                }
            });
            dis.fill(Number.MAX_SAFE_INTEGER);
            dis[from] = 0;
            pq.enqueue([0, from]);
            while (pq.size()) {
                let [curDis, cur] = pq.dequeue();
                if (dis[cur] < curDis) continue;
                for (let i = 0; i < g[cur].length; i++) {
                    let child = g[cur][i];
                    if (child.cap > 0 && dis[child.to] > dis[cur] + child.cost + h[cur] - h[child.to]) {
                        dis[child.to] = dis[cur] + child.cost + h[cur] - h[child.to];
```

```
                        prev_v[child.to] = cur;
                        prev_e[child.to] = i;
                        pq.enqueue([dis[child.to], child.to]);
                    }
                }
            }
            if (dis[to] == Number.MAX_SAFE_INTEGER) return -1;
            for (let i = 0; i < n; i++) h[i] += dis[i];
            let d = flow;
            for (let i = to; i != from; i = prev_v[i]) {
                d = Math.min(d, g[prev_v[i]][prev_e[i]].cap);
            }
            flow -= d;
            res += d * h[to];
            for (let i = to; i != from; i = prev_v[i]) {
                let edge = g[prev_v[i]][prev_e[i]];
                edge.cap -= d;
                g[i][edge.from].cap += d;
            }
        }
        return res;
    }
}

const minimumTotalDistance = (a, b) => {
    let n = a.length, m = b.length, mcmf = new MCMF(n + m + 2), from = n + m, to = from + 1;
    for (let i = 0; i < n; i++) mcmf.addEdge(from, i, 0, 1);
    for (let i = 0; i < m; i++) mcmf.addEdge(n + i, to, 0, b[i][1]);
    for (let i = 0; i < n; i++) {
        for (let j = 0; j < m; j++) {
            mcmf.addEdge(i, n + j, Math.abs(a[i] - b[j][0]), 1);
        }
    }
    return mcmf.minCostFlow(from, to, n);
};
```

☐ **Custom Testcase** Use Example Testcases

Run Submit

Submission Result: Accepted (/submissions/detail/837843796/) More Details > (/submissions/detail/837843796/)

Share your acceptance!

1