/\*\*

\* @param {number[][]} times

\* @param {number} n

\* @param {number} k

\* @return {number}

\*/

var networkDelayTime = function(times, n, k) {

// 1. 构建图的邻接表

const graph = new Array(n + 1).fill().map(() => []);

for (const [u, v, w] of times) {

graph[u].push([v, w]);

}

// 2. 初始化距离数组

const dist = new Array(n + 1).fill(Infinity);

dist[k] = 0;

// 3. 优先队列（这里用数组模拟，实际应用中可以用更高效的结构）

const queue = [[k, 0]];

while (queue.length > 0) {

// 找到当前距离最小的节点

let minIndex = 0;

for (let i = 1; i < queue.length; i++) {

if (queue[i][1] < queue[minIndex][1]) {

minIndex = i;

}

}

const [current, currentDist] = queue.splice(minIndex, 1)[0];

// 遍历邻居

for (const [neighbor, time] of graph[current]) {

const newDist = currentDist + time;

if (newDist < dist[neighbor]) {

dist[neighbor] = newDist;

queue.push([neighbor, newDist]);

}

}

}

// 4. 找出最大距离

let maxTime = 0;

for (let i = 1; i <= n; i++) {

if (dist[i] === Infinity) {

return -1; // 有节点不可达

}

maxTime = Math.max(maxTime, dist[i]);

}

return maxTime;

};