/\*\*

\* @param {number} numCourses

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

\* @return {boolean}

\*/

var canFinish = function(numCourses, prerequisites) {

// 1. 构建邻接表和入度数组

const adjList = new Array(numCourses).fill(0).map(() => []);

const inDegree = new Array(numCourses).fill(0);

for (const [course, prereq] of prerequisites) {

adjList[prereq].push(course);

inDegree[course]++;

}

// 2. 初始化队列，将所有入度为0的节点加入队列

const queue = [];

for (let i = 0; i < numCourses; i++) {

if (inDegree[i] === 0) {

queue.push(i);

}

}

// 3. 执行拓扑排序

let count = 0;

while (queue.length > 0) {

const current = queue.shift();

count++;

// 减少当前节点的所有邻居的入度

for (const neighbor of adjList[current]) {

inDegree[neighbor]--;

// 如果邻居的入度变为0，加入队列

if (inDegree[neighbor] === 0) {

queue.push(neighbor);

}

}

}

// 4. 如果所有节点都被访问过，说明没有环

return count === numCourses;

};