1. Course Schedule

There are a total of numCourses courses you have to take, labeled from 0 to numCourses-1.

Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: [0,1]

Given the total number of courses and a list of prerequisite **pairs**, is it possible for you to finish all courses?

**Example 1:**

Input: numCourses = 2, prerequisites = [[1,0]]  
Output: true  
Explanation: There are a total of 2 courses to take.   
 To take course 1 you should have finished course 0. So it is possible.

**Example 2:**

Input: numCourses = 2, prerequisites = [[1,0],[0,1]]  
Output: false  
Explanation: There are a total of 2 courses to take.   
 To take course 1 you should have finished course 0, and to take course 0 you should  
 also have finished course 1. So it is impossible.

**Constraints:**

* The input prerequisites is a graph represented by **a list of edges**, not adjacency matrices. Read more about [how a graph is represented](https://www.khanacademy.org/computing/computer-science/algorithms/graph-representation/a/representing-graphs).
* You may assume that there are no duplicate edges in the input prerequisites.
* 1 <= numCourses <= 10^5

**解法1** 拓扑排序。如果能完整排序，说明可以修完所有的课程

class Solution {  
public:  
 bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {  
 vector<vector<int>>g(numCourses);  
 vector<int>d(numCourses, 0);  
 build\_graph(numCourses, prerequisites, g, d);  
   
 return topological\_sort(numCourses, g, d);  
 }  
 void build\_graph(int numCourses, vector<vector<int>>& prerequisites,  
 vector<vector<int>>&g, vector<int>&d){  
 for(auto x: prerequisites){  
 g[x[1]].push\_back(x[0]);  
 d[x[0]]++;  
 }  
 }  
 bool topological\_sort(int numCourses, vector<vector<int>> &g, vector<int>&d){  
   
 vector<bool>vis(numCourses, false);  
 bool flag;  
 while(1){  
 int u = -1;  
 flag = false;  
 for(int i = 0; i < numCourses; ++i){  
 if(d[i] == 0 && vis[i] == false)u = i;  
 if(d[i] != 0)flag = true;  
 }  
 if(u == -1)break;  
 vis[u] = true;  
 for(auto v : g[u])d[v]--;  
 }  
 return !flag;  
 }  
};

**解法2** dfs。枚举每个节点，看该节点能不能形成环

class Solution {  
public:  
 bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {  
 vector<vector<int>>g(numCourses);  
 vector<int>d(numCourses, 0);  
 build\_graph(numCourses, prerequisites, g, d);  
   
 vector<bool>tested(numCourses, false); // 避免对同一条路径上的节点重复测试  
 for(int s = 0; s < numCourses; ++s){  
 if(!tested[s]){  
 vector<bool>vis(numCourses, false);  
 bool flag = false;  
 dfs(g, s, vis, tested, flag);  
 if(flag)return false;  
 }  
 }  
 return true;  
 }  
 void dfs(vector<vector<int>> &g, int s, vector<bool>&vis, vector<bool> &tested, bool &flag){  
 if(flag)return;  
 for(auto v: g[s]){  
 if(vis[v]){  
 flag = true;  
 return;  
 }  
 vis[v] = true;  
 dfs(g, v, vis, tested, flag);  
 vis[v] = false;  
 }  
 tested[s] = true;  
 }  
 void build\_graph(int numCourses, vector<vector<int>>& prerequisites,  
 vector<vector<int>>&g, vector<int>&d){  
 for(auto x: prerequisites){  
 g[x[1]].push\_back(x[0]);  
 d[x[0]]++;  
 }  
 }  
};