**207. 课程表**

你这个学期必须选修 numCourse 门课程，记为 0 到 numCourse-1 。

在选修某些课程之前需要一些先修课程。 例如，想要学习课程 0 ，你需要先完成课程 1 ，我们用一个匹配来表示他们：[0,1]

给定课程总量以及它们的先决条件，请你判断是否可能完成所有课程的学习？

输入: 2, [[1,0]]

输出: true

解释: 总共有 2 门课程。学习课程 1 之前，你需要完成课程 0。所以这是可能的。

输入: 2, [[1,0],[0,1]]

输出: false

解释: 总共有 2 门课程。学习课程 1 之前，你需要先完成​课程 0；并且学习课程 0 之前，你还应先完成课程 1。这是不可能的。

|  |
| --- |
| class Solution {  public:  bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {  if (prerequisites.empty()) {  return true;  }  vector<set<int>> adjGraph(numCourses, set<int>());  vector<int> inDegrees(numCourses, 0);  for (auto& relation : prerequisites) {  int from = relation[1];  int to = relation[0];  adjGraph[from].insert(to);  inDegrees[to]++;  }  queue<int> zeroDegrees;  for (size\_t i = 0; i < inDegrees.size(); i++) {  if (inDegrees[i] == 0) {  zeroDegrees.push(i);  }  }  vector<int> courses;  while (!zeroDegrees.empty()) {  int availabe = zeroDegrees.front();  zeroDegrees.pop();  courses.push\_back(availabe);  for (auto next : adjGraph[availabe]) {  inDegrees[next]--;  if (inDegrees[next] == 0) {  zeroDegrees.push(next);  }  }  }    return courses.size() == numCourses;  }  }; |

**210. 课程表 II**

现在你总共有 n 门课需要选，记为 0 到 n-1。

在选修某些课程之前需要一些先修课程。 例如，想要学习课程 0 ，你需要先完成课程 1 ，我们用一个匹配来表示他们: [0,1]

给定课程总量以及它们的先决条件，返回你为了学完所有课程所安排的学习顺序。

可能会有多个正确的顺序，你只要返回一种就可以了。如果不可能完成所有课程，返回一个空数组。

输入: 2, [[1,0]]

输出: [0,1]

解释: 总共有 2 门课程。要学习课程 1，你需要先完成课程 0。因此，正确的课程顺序为 [0,1] 。

输入: 4, [[1,0],[2,0],[3,1],[3,2]]

输出: [0,1,2,3] or [0,2,1,3]

解释: 总共有 4 门课程。要学习课程 3，你应该先完成课程 1 和课程 2。并且课程 1 和课程 2 都应该排在课程 0 之后。因此，一个正确的课程顺序是 [0,1,2,3] 。另一个正确的排序是 [0,2,1,3] 。

|  |
| --- |
| class Solution {  public:  vector<int> findOrder(int numCourses, vector<vector<int>>& prerequisites) {  if (numCourses == 0) {  return {};  }  vector<int> res;  if (prerequisites.empty()) {  res.resize(numCourses);  iota(res.begin(), res.end(), 0);  return res;  }  vector<set<int>> adjGraph(numCourses, set<int>());  vector<int> inDegree(numCourses, 0);  for (auto& relation : prerequisites) {  int from = relation[1];  int to = relation[0];  inDegree[to]++;  adjGraph[from].insert(to);  }  queue<int> zeroQueue;  for (size\_t i = 0; i < inDegree.size(); i++) {  if (inDegree[i] == 0) {  zeroQueue.push(i);  }  }  while (!zeroQueue.empty()) {  int top = zeroQueue.front();  zeroQueue.pop();  res.push\_back(top);  for (auto item : adjGraph[top]) {  inDegree[item]--;  if (inDegree[item] == 0) {  zeroQueue.push(item);  }  }  }  if (res.size() != numCourses) {  return {};  }  return res;  }  }; |

**329. 矩阵中的最长递增路径**

给定一个整数矩阵，找出最长递增路径的长度。

对于每个单元格，你可以往上，下，左，右四个方向移动。 你不能在对角线方向上移动或移动到边界外（即不允许环绕）。

输入: nums =

[

[9,9,4],

[6,6,8],

[2,1,1]

]

输出: 4

解释: 最长递增路径为 [1, 2, 6, 9]。

输入: nums =

[

[3,4,5],

[3,2,6],

[2,2,1]

]

输出: 4

解释: 最长递增路径是 [3, 4, 5, 6]。注意不允许在对角线方向上移动。

|  |
| --- |
| class Solution {  public:  int longestIncreasingPath(vector<vector<int>>& matrix)  {  int row = matrix.size();  if(row <= 0)return 0;  int col = matrix[0].size();  vector<vector<int>>isVisited(row, vector<int>(col, 0));  int res = 0;  //依次求取每一个节点为起点的最长增长序列，并保存最大值。  for(int i = 0; i < row; ++i)  {  for(int j = 0; j < col; ++j)  res = max(res, dfs(matrix, i, j, isVisited, INT\_MIN));  }  return res;  }  int dfs(vector<vector<int>>& matrix, int row, int col, vector<vector<int>>& isVisited, int pre)  {  if(row<0 || col<0|| row>= matrix.size() || col>=matrix[0].size() || matrix[row][col]<=pre)  return 0;  //已经对该节点进行过求取，直接返回即可  if(isVisited[row][col] > 0)  return isVisited[row][col];  //遍历上下左右节点  int l = dfs(matrix, row, col-1, isVisited, matrix[row][col]);  int r = dfs(matrix, row, col+1, isVisited, matrix[row][col]);  int up = dfs(matrix, row-1, col, isVisited, matrix[row][col]);  int dow = dfs(matrix, row+1, col, isVisited, matrix[row][col]);  //存储以该节点为起点的最长增长序列的长度  isVisited[row][col] = 1 + max(max(l, r), max(up, dow));  return isVisited[row][col];  }  }; |

**LeetCode 269. 火星词典（拓扑排序）**

现有一种使用字母的全新语言，这门语言的字母顺序与英语顺序不同。

假设，您并不知道其中字母之间的先后顺序。但是，会收到词典中获得一个 不为空的 单词列表。

因为是从词典中获得的，所以该单词列表内的单词已经 按这门新语言的字母顺序进行了排序。您需要根据这个输入的列表，还原出此语言中已知的字母顺序。

输入:

["wrt", "wrf", "er", "ett", "rftt"]

输出: "wertf"

输入:[ "z", "x" ]

输出: "zx"

输入:

["z", "x", "z"]

输出: ""

解释: 此顺序是非法的，因此返回 ""。

提示：

你可以默认输入的全部都是小写字母

若给定的顺序是不合法的，则返回空字符串即可

若存在多种可能的合法字母顺序，请返回其中任意一种顺序即可

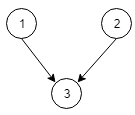
|  |
| --- |
| class Solution {  public:  string alienOrder(vector<string>& words) {  unordered\_set<char> allchar; //记下所有的字符  for(string& w : words)  {  allchar.insert(w.begin(), w.end());  }  unordered\_map<char,int> indegree;  unordered\_map<char,unordered\_set<char>> graph;  int n1, n2, n;  for(int i = 1; i < words.size(); ++i)  {  if(words[i-1] == words[i])  continue;  n1 = words[i-1].size();  n2 = words[i].size();  n = min(n1, n2);  int j = 0;  for(j = 0; j < n; ++j)  { //第一个不相等的字符构成有向图的边  if(words[i-1][j] != words[i][j])  {//防止重复添加同一条边导致度计算错误 如：a->b a不存在 或 a->b连接不存在  if(!graph.count(words[i-1][j]) || !graph[words[i-1][j]].count(words[i][j]))  {  graph[words[i-1][j]].insert(words[i][j]);  indegree[words[i][j]]++;  }  break;  }  }  if(j == n && n1 > n2)  return "";//前面相等，前者长，输入有误  }  queue<char> q;  for(auto it = indegree.begin(); it != indegree.end(); ++it)  {  if(it->second == 0)//入度为0的入队  q.push(it->first);  }  string ans;  while(!q.empty())  {  char ch = q.front();  allchar.erase(ch);  q.pop();  ans += ch;  for(auto c : graph[ch].second)  {  --indegree[c]；  if(indegree[c] == 0)  q.push(c);  }  }  if(ans.size() != indegree.size())  return "";// 存在边关系的节点没用完，存在环  while(allchar.size())  { //剩余字符随便放，放在ans的后面  ans += \*allchar.begin();  allchar.erase(allchar.begin());  }  return ans;  }  }; |

**1136. 平行课程（拓扑排序）**

已知有 N 门课程，它们以 1 到 N 进行编号。

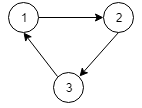
给你一份课程关系表 relations[i] = [X, Y]，用以表示课程 X 和课程 Y 之间的先修关系：课程 X 必须在课程 Y 之前修完。

假设在一个学期里，你可以学习任何数量的课程，但前提是你已经学习了将要学习的这些课程的所有先修课程。请你返回学完全部课程所需的最少学期数。如果没有办法做到学完全部这些课程的话，就返回 -1。

输入：N = 3, relations = [[1,3],[2,3]]

输出：2

解释：在第一个学期学习课程 1 和 2，在第二个学期学习课程 3。

输入：N = 3, relations = [[1,2],[2,3],[3,1]]

输出：-1

解释：没有课程可以学习，因为它们相互依赖。

提示：1 <= N <= 5000, 1 <= relations.length <= 5000, relations[i][0] != relations[i][1]

输入中没有重复的关系

|  |
| --- |
| class Solution {  public:  int minimumSemesters(int N, vector<vector<int>>& relations) {  unordered\_map<int,unordered\_set<int>> m;  vector<int> indegree(N+1, 0);  for(auto& re:relations)  {  indegree[re[1]]++;  m[re[0]].insert(re[1]);  }  queue<int> q;  int cur, size, sum = 0, finish = 0;  for(int i = 1; i <= N; ++i)  if(indegree[i] == 0)  q.push(i);  while(!q.empty())  {  size = q.size();  while(size--)  {  cur = q.front();  q.pop();  finish++;  for(auto it = m[cur].begin(); it != m[cur].end(); ++it)  {  if(--indegree[\*it] == 0)  q.push(\*it);  }  }  **sum++;**  }  return finish==N ? sum : -1;  }  }; |

**444. 序列重建（拓扑排序）**

验证原始的序列 org 是否可以从序列集 seqs 中唯一地重建。序列 org 是 1 到 n 整数的排列，其中 1 ≤ n ≤ 104。

重建是指在序列集 seqs 中构建最短的公共超序列。（即使得所有 seqs 中的序列都是该最短序列的子序列）。确定是否只可以从 seqs 重建唯一的序列，且该序列就是 org 。

输入：org: [1,2,3], seqs: [[1,2],[1,3]]

输出：false

解释：[1,2,3] 不是可以被重建的唯一的序列，因为 [1,3,2] 也是一个合法的序列。

输入：org: [1,2,3], seqs: [[1,2]]

输出：false

解释：可以重建的序列只有 [1,2]。

输入：org: [1,2,3], seqs: [[1,2],[1,3],[2,3]]

输出：true

解释：序列 [1,2], [1,3] 和 [2,3] 可以被唯一地重建为原始的序列 [1,2,3]。

输入：org: [4,1,5,2,6,3], seqs: [[5,2,6,3],[4,1,5,2]]

输出：true

|  |
| --- |
| class Solution {  public:  bool sequenceReconstruction(vector<int>& org, vector<vector<int>>& seqs) {  int n = org.size();  vector<bool> exit(n+1, false);//是否存在  vector<vector<int>> g(n+1);//图  vector<int> indegree(n+1, 0);//入度  for(auto& s : seqs)  {  int from = -1;  for(int i = 0; i < s.size(); ++i)  {  if(s[i] <= 0 || s[i] > n) return false;//编号超了  exit[s[i]] = true;  if(from != -1)  {  g[from].push\_back(s[i]);//边  indegree[s[i]]++;//入度+1  }  from = s[i];  }  }  queue<int> q;  for(int i = 1; i <= n; ++i)  {  if(!exit[i]) return false;//有的点不存在  if(indegree[i]==0)//入度为0的入队  q.push(i);  }  int i = 0;  while(!q.empty())  {  if(q.size() != 1) return false;//选择不唯一  int cur = q.front();  if(cur != org[i++]) return false;//跟序列不匹配  q.pop();  for(int i = 0; i < g[cur].size(); ++i)  {  if(--indegree[g[cur][i]] == 0)  q.push(g[cur][i]);//入队为0 的入队  }  }  if(i != n) return false;//有环  return true;  }  }; |

<https://www.cnblogs.com/grandyang/p/6032498.html> 哈希方式