# Leetcode 题解 - 图

* [Leetcode 题解 - 图](#leetcode-题解---图)
	+ [二分图](#二分图)
		- [1. 判断是否为二分图](#1-判断是否为二分图)
	+ [拓扑排序](#拓扑排序)
		- [1. 课程安排的合法性](#1-课程安排的合法性)
		- [2. 课程安排的顺序](#2-课程安排的顺序)
	+ [并查集](#并查集)
		- [1. 冗余连接](#1-冗余连接)

## 二分图

如果可以用两种颜色对图中的节点进行着色，并且保证相邻的节点颜色不同，那么这个图就是二分图。

### 1. 判断是否为二分图

785. Is Graph Bipartite? (Medium)

[Leetcode](https://leetcode.com/problems/is-graph-bipartite/description/) / [力扣](https://leetcode-cn.com/problems/is-graph-bipartite/description/)

Input: [[1,3], [0,2], [1,3], [0,2]]
Output: true
Explanation:
The graph looks like this:
0----1
| |
| |
3----2
We can divide the vertices into two groups: {0, 2} and {1, 3}.

Example 2:
Input: [[1,2,3], [0,2], [0,1,3], [0,2]]
Output: false
Explanation:
The graph looks like this:
0----1
| \ |
| \ |
3----2
We cannot find a way to divide the set of nodes into two independent subsets.

public boolean isBipartite(int[][] graph) {
 int[] colors = new int[graph.length];
 Arrays.fill(colors, -1);
 for (int i = 0; i < graph.length; i++) { // 处理图不是连通的情况
 if (colors[i] == -1 && !isBipartite(i, 0, colors, graph)) {
 return false;
 }
 }
 return true;
}

private boolean isBipartite(int curNode, int curColor, int[] colors, int[][] graph) {
 if (colors[curNode] != -1) {
 return colors[curNode] == curColor;
 }
 colors[curNode] = curColor;
 for (int nextNode : graph[curNode]) {
 if (!isBipartite(nextNode, 1 - curColor, colors, graph)) {
 return false;
 }
 }
 return true;
}

## 拓扑排序

常用于在具有先序关系的任务规划中。

### 1. 课程安排的合法性

207. Course Schedule (Medium)

[Leetcode](https://leetcode.com/problems/course-schedule/description/) / [力扣](https://leetcode-cn.com/problems/course-schedule/description/)

2, [[1,0]]
return true

2, [[1,0],[0,1]]
return false

题目描述：一个课程可能会先修课程，判断给定的先修课程规定是否合法。

本题不需要使用拓扑排序，只需要检测有向图是否存在环即可。

public boolean canFinish(int numCourses, int[][] prerequisites) {
 List<Integer>[] graphic = new List[numCourses];
 for (int i = 0; i < numCourses; i++) {
 graphic[i] = new ArrayList<>();
 }
 for (int[] pre : prerequisites) {
 graphic[pre[0]].add(pre[1]);
 }
 boolean[] globalMarked = new boolean[numCourses];
 boolean[] localMarked = new boolean[numCourses];
 for (int i = 0; i < numCourses; i++) {
 if (hasCycle(globalMarked, localMarked, graphic, i)) {
 return false;
 }
 }
 return true;
}

private boolean hasCycle(boolean[] globalMarked, boolean[] localMarked,
 List<Integer>[] graphic, int curNode) {

 if (localMarked[curNode]) {
 return true;
 }
 if (globalMarked[curNode]) {
 return false;
 }
 globalMarked[curNode] = true;
 localMarked[curNode] = true;
 for (int nextNode : graphic[curNode]) {
 if (hasCycle(globalMarked, localMarked, graphic, nextNode)) {
 return true;
 }
 }
 localMarked[curNode] = false;
 return false;
}

### 2. 课程安排的顺序

210. Course Schedule II (Medium)

[Leetcode](https://leetcode.com/problems/course-schedule-ii/description/) / [力扣](https://leetcode-cn.com/problems/course-schedule-ii/description/)

4, [[1,0],[2,0],[3,1],[3,2]]
There are a total of 4 courses to take. To take course 3 you should have finished both courses 1 and 2. Both courses 1 and 2 should be taken after you finished course 0. So one correct course order is [0,1,2,3]. Another correct ordering is[0,2,1,3].

使用 DFS 来实现拓扑排序，使用一个栈存储后序遍历结果，这个栈的逆序结果就是拓扑排序结果。

证明：对于任何先序关系：v->w，后序遍历结果可以保证 w 先进入栈中，因此栈的逆序结果中 v 会在 w 之前。

public int[] findOrder(int numCourses, int[][] prerequisites) {
 List<Integer>[] graphic = new List[numCourses];
 for (int i = 0; i < numCourses; i++) {
 graphic[i] = new ArrayList<>();
 }
 for (int[] pre : prerequisites) {
 graphic[pre[0]].add(pre[1]);
 }
 Stack<Integer> postOrder = new Stack<>();
 boolean[] globalMarked = new boolean[numCourses];
 boolean[] localMarked = new boolean[numCourses];
 for (int i = 0; i < numCourses; i++) {
 if (hasCycle(globalMarked, localMarked, graphic, i, postOrder)) {
 return new int[0];
 }
 }
 int[] orders = new int[numCourses];
 for (int i = numCourses - 1; i >= 0; i--) {
 orders[i] = postOrder.pop();
 }
 return orders;
}

private boolean hasCycle(boolean[] globalMarked, boolean[] localMarked, List<Integer>[] graphic,
 int curNode, Stack<Integer> postOrder) {

 if (localMarked[curNode]) {
 return true;
 }
 if (globalMarked[curNode]) {
 return false;
 }
 globalMarked[curNode] = true;
 localMarked[curNode] = true;
 for (int nextNode : graphic[curNode]) {
 if (hasCycle(globalMarked, localMarked, graphic, nextNode, postOrder)) {
 return true;
 }
 }
 localMarked[curNode] = false;
 postOrder.push(curNode);
 return false;
}

## 并查集

并查集可以动态地连通两个点，并且可以非常快速地判断两个点是否连通。

### 1. 冗余连接

684. Redundant Connection (Medium)

[Leetcode](https://leetcode.com/problems/redundant-connection/description/) / [力扣](https://leetcode-cn.com/problems/redundant-connection/description/)

Input: [[1,2], [1,3], [2,3]]
Output: [2,3]
Explanation: The given undirected graph will be like this:
 1
 / \
2 - 3

题目描述：有一系列的边连成的图，找出一条边，移除它之后该图能够成为一棵树。

public int[] findRedundantConnection(int[][] edges) {
 int N = edges.length;
 UF uf = new UF(N);
 for (int[] e : edges) {
 int u = e[0], v = e[1];
 if (uf.connect(u, v)) {
 return e;
 }
 uf.union(u, v);
 }
 return new int[]{-1, -1};
}

private class UF {

 private int[] id;

 UF(int N) {
 id = new int[N + 1];
 for (int i = 0; i < id.length; i++) {
 id[i] = i;
 }
 }

 void union(int u, int v) {
 int uID = find(u);
 int vID = find(v);
 if (uID == vID) {
 return;
 }
 for (int i = 0; i < id.length; i++) {
 if (id[i] == uID) {
 id[i] = vID;
 }
 }
 }

 int find(int p) {
 return id[p];
 }

 boolean connect(int u, int v) {
 return find(u) == find(v);
 }
}