

Leetcode 每日一题

2023.09.12

1462. 课程表 IV

已解答

中等

相关标签

相关企业

提示

你总共需要上 `numCourses` 门课，课程编号依次为 `0` 到 `numCourses-1`。你会得到一个数组 `prerequisite`，其中 `prerequisites[i] = [ai, bi]` 表示如果你想选 `bi` 课程，你必须先选 `ai` 课程。

- 有的课会有直接的先修课程，比如如果想上课程 `1`，你必须先上课程 `0`，那么会以 `[0,1]` 数对的形式给出先修课程数对。

先决条件也可以是间接的。如果课程 `a` 是课程 `b` 的先决条件，课程 `b` 是课程 `c` 的先决条件，那么课程 `a` 就是课程 `c` 的先决条件。

你也得到一个数组 `queries`，其中 `queries[j] = [uj, vj]`。对于第 `j` 个查询，您应该回答课程 `uj` 是否是课程 `vj` 的先决条件。

返回一个布尔数组 `answer`，其中 `answer[j]` 是第 `j` 个查询的答案。

最初解法：纯建图后DFS

```

vector<bool> checkIfPrerequisite(int numCourses, vector<vector<int>>& prerequisites, vector<vect
vector<vector<int>> graph(numCourses);
for(int i = 0; i < prerequisites.size(); i++){
    int u = prerequisites[i][0];
    int v = prerequisites[i][1];
    graph[u].emplace_back(v);
}
vector<bool> ans;
for(int i = 0; i < queries.size(); i++){
    int u = queries[i][0];
    int v = queries[i][1];
    vector<bool> visited(numCourses, false);
    dfs(graph, u, v, ans, visited, false);
    if(ans.size() < i+1) {ans.emplace_back(false);}
}
return ans;
}

void dfs(vector<vector<int>>& graph, int u, int v, vector<bool>& ans, vector<bool>& visited, bo
if(u == v) {
    ans.emplace_back(true);
    flag = true;
    return;
}
if(graph[u].size() == 0) {return;}
for(int j = 0; j<graph[u].size(); j++){
    if(flag) {break;}
    int point = graph[u][j];
    if(visited[point] != true){
        visited[point] = true;
        dfs(graph, point, v, ans, visited, flag);
    }
}
}
}

```

更好解法：拓扑排序+BFS/DFS

- 拓扑排序：1.有向无环图； 2.序列里的每一个点只能出现一次； 3.任何一对 u 和 v ， u 总在 v 之前（这里的两个字母分别表示的是一条线段的两个端点， u 表示起点， v 表示终点）
- DFS+拓扑排序

我们从编号小到遍历全部节点，若节点 i 未被访问，则进入「深度优先搜索」流程：

- 若当前节点 x 已被访问，则直接返回。
- 若当前节点 x 未被访问，将访问状态设为已访问，然后继续对其全部后继节点递归进行「深度优先搜索」流程。将节点 x 置为其每一个后继节点 y 的先决条件，有 $isPre[x][y] = \text{True}$ ，以及对于每一个以 y 为先决条件的节点 t ，节点 x 同样为 t 的先决条件，有 $isPre[x][t] = \text{True}$ 。

```

vector<bool> checkIfPrerequisite(int numCourses, vector<vector<int>>& prerequisites, vector<vect
vector<vector<int>> graph(numCourses);
vector<vector<bool>> judge(numCourses, vector<bool>(numCourses, false));
for(int i = 0; i < prerequisites.size(); i++){
    int u = prerequisites[i][0];
    int v = prerequisites[i][1];
    graph[u].emplace_back(v);
}
vector<bool> visited(numCourses, false);
for(int i = 0; i < numCourses; i++){
    dfs(i, graph, judge, visited);
}
vector<bool> ans;
for(int i = 0; i < queries.size(); i++){
    ans.emplace_back(judge[queries[i][0]][queries[i][1]]);
}
return ans;
}

void dfs(int cur, vector<vector<int>>& graph, vector<vector<bool>>& judge, vector<bool>& vis
if(visited[cur]) {return;}
visited[cur] = true;
for(int j = 0; j<graph[cur].size(); j++){ //遍历cur的子树
    int next = graph[cur][j];
    dfs(next, graph, judge, visited);
    judge[cur][next] = true;
    for(int nnext = 0; nnext < judge[cur].size(); nnext++){//遍历cur的子树的子树
        judge[cur][nnext] = judge[cur][nnext] | judge[next][nnext];
    }
}
}
}

```

• BFS+拓扑排序

现在有 m 个查询 $queries$ ，其中对于第 i 个查询 $queries[i] = [u_i, v_i]$ ，我们需要判断课程 u_i 是否是课程 v_i 的直接或间接先决条件。我们创建一个 $numCourses \times numCourses$ 的矩阵 $isPre$ ，其中 $isPre[x][y]$ 表示课程 x 是否是课程 y 的直接或间接先决条件，若是则 $isPre[x][y] = True$ ，否则 $isPre[x][y] = False$ 。在完成 $isPre$ 计算后，我们对于每一个查询就可以在 $O(1)$ 时间得到结果。对于 $isPre$ 的计算，我们可以通过「广度优先搜索」+「拓扑排序」来对矩阵 $isPre$ 进行计算：

首先我们需要计算有向图中每一个节点的入度，并对入度为 0 的节点加入队列。然后只要队列非空，就进行以下操作：

- 取出队列队首元素，同时，将这个节点及其所有前置条件节点设置为所有后续节点的前置条件节点，然后对每一个后续节点入度进行 -1 操作，若操作后的节点入度为 0，则继续将该节点加入队列。

```

vector<bool> checkIfPrerequisite(int numCourses, vector<vector<int>>& prerequisites, vector<vector<vector<int>>> graph(numCourses);
vector<int> degree(numCourses, 0);
vector<vector<bool>> judge(numCourses, vector<bool>(numCourses, false));
for(int i = 0; i < prerequisites.size(); i++){
    int u = prerequisites[i][0];
    int v = prerequisites[i][1];
    graph[u].emplace_back(v);
    degree[v]++;
}
queue<int> que;
for(int i = 0; i < numCourses; i++){
    if(degree[i] == 0){
        que.push(i); //入度为0的入队列
    }
}
while(!que.empty()){
    int cur = que.front();
    que.pop();
    for(int j = 0; j < graph[cur].size(); j++){ //遍历cur的子树
        int next = graph[cur][j];
        judge[cur][next] = true;
        for(int pre = 0; pre < judge[cur].size(); pre++){ //遍历cur的父节点
            judge[pre][next] = judge[pre][next] | judge[pre][cur];
        }
        degree[next]--;
        if(degree[next] == 0){
            que.push(next);
        }
    }
}
vector<bool> ans;
for(int i = 0; i < queries.size(); i++){
    ans.emplace_back(judge[queries[i][0]][queries[i][1]]);
}
return ans;
}

```

- 注意：由于DFS是从最深回溯到最浅的结点的，所以遍历的是cur的子树的子树，而BFS是从浅入深地，所以遍历的是cur的父节点

2596. 检查骑士巡视方案

已解答

中等

相关标签

相关企业

提示

骑士在一张 $n \times n$ 的棋盘上巡视。在 有效 的巡视方案中，骑士会从棋盘的 左上角 出发，并且访问棋盘上的每个格子 恰好一次 。

给你一个 $n \times n$ 的整数矩阵 `grid`，由范围 $[0, n * n - 1]$ 内的不同整数组成，其中 `grid[row][col]` 表示单元格 (row, col) 是骑士访问的第 `grid[row][col]` 个单元格。骑士的行动是从下标 **0** 开始的。

如果 `grid` 表示了骑士的有效巡视方案，返回 `true`；否则返回 `false`。

注意，骑士行动时可以垂直移动两个格子且水平移动一个格子，或水平移动两个格子且垂直移动一个格子。下图展示了骑士从某个格子出发可能的八种行动路线。



方法一：深度优先搜索

```

bool checkValidGrid(vector<vector<int>>& grid) {
    if(grid[0][0] != 0) {return false;}
    int g_size = grid.size();
    vector<vector<bool>> visited(g_size, vector<bool>(g_size, false));
    vector<vector<int>> direction = {{1, -2}, {2, -1}, {2, 1}, {1, 2}, {-1, 2}, {-2, 1}, {-2, -1}
    bool ans = false;
    dfs(0, 0, 0, visited, direction, g_size, ans, grid);
    return ans;
}

void dfs(int x, int y, int index, vector<vector<bool>>& visited, vector<vector<int>>& direction,
    if(index == g_size*g_size-1) {ans = true; return;}
    if(visited[x][y]) {return;}
    visited[x][y] = true;
    for(int i = 0; i < 8; i++){
        int new_x = x + direction[i][0];
        int new_y = y + direction[i][1];
        if(new_x < g_size && new_x >= 0 && new_y < g_size && new_y >= 0 && grid[new_x][new_y] ==
            dfs(new_x, new_y, index+1, visited, direction, g_size, ans, grid);
        }
    }
}

```