拓扑排序

拓扑排序是一种有向无环图（DAG）的顶点线性排序，其中每一条有向边都从排在前面的顶点指向排在后面的顶点。拓扑排序可以应用于任务调度、依赖关系分析等场景。

深度优先搜索（DFS）是一种常用的拓扑排序算法。下面详细解释拓扑排序的基本思路和实现步骤。

拓扑排序的基本思路：

1. **构建有向图：** 将问题中的关系表示成有向图，图的节点表示任务或顶点，有向边表示任务间的依赖关系。
2. **深度优先搜索：** 从图中的任意一个未访问的节点开始，进行深度优先搜索。在搜索的过程中，将已访问的节点标记为“正在访问中”（状态1），递归访问该节点的邻接节点。
3. **检测环：** 如果在搜索的过程中发现当前节点已经处于“正在访问中”的状态，说明图中存在环，拓扑排序不合法。如果发现当前节点已经访问完成（状态2），则说明已经处理过，无需再次访问。
4. **逆序输出：** 在深度优先搜索完成后，逆序输出顶点，即将深度优先搜索的结果保存在一个栈中，然后依次弹出栈顶元素，得到的顺序即为拓扑排序的结果。

深度优先搜索的拓扑排序实现：

```
#include <iostream>
#include <vector>
#include <unordered_set>

using namespace std;

void dfs(int course, vector<vector<int>>& adjList, vector<int>& visited,
vector<int>& result, unordered_set<int>& cycleCheck) {
    visited[course] = 1; // 1 表示正在访问中

    for (int neighbor : adjList[course]) {
        if (visited[neighbor] == 0) {
            dfs(neighbor, adjList, visited, result, cycleCheck);
        } else if (visited[neighbor] == 1) {
            // 检测到环，拓扑排序不合法
            cycleCheck.insert(course);
        }
    }

    visited[course] = 2; // 2 表示已经访问完成
    result.push_back(course);
}

vector<int> topologicalSort(int numCourses, vector<vector<int>>& prerequisites) {
    vector<int> result;
    vector<int> visited(numCourses, 0); // 0 表示未访问

    // 构建邻接表
    vector<vector<int>> adjList(numCourses);
    for (const auto& edge : prerequisites) {
        adjList[edge[1]].push_back(edge[0]);
    }
```

```
    unordered_set<int> cycleCheck;

    // 对每个节点进行深度优先搜索
    for (int i = 0; i < numCourses; ++i) {
        if (visited[i] == 0) {
            dfs(i, adjList, visited, result, cycleCheck);
        }
    }

    // 如果存在环，拓扑排序不合法
    if (!cycleCheck.empty()) {
        return {};
    }

    // 按访问完成的逆序即为拓扑排序的结果
    reverse(result.begin(), result.end());
    return result;
}

int main() {
    int numCourses = 4;
    vector<vector<int>> prerequisites = {{1, 0}, {2, 0}, {3, 1}, {3, 2}};

    vector<int> result = topologicalSort(numCourses, prerequisites);

    if (result.empty()) {
        cout << "There is a cycle in the graph. Topological sort is not
possible." << endl;
    } else {
        cout << "Topological Sort: ";
        for (int course : result) {
            cout << course << " ";
        }
        cout << endl;
    }

    return 0;
}
```

这个实现使用了深度优先搜索，通过 `visited` 数组记录节点的访问状态，同时使用 `cycleCheck` 集合检查是否存在环。在 DFS 过程中，如果发现某个节点已经在访问中，说明存在环，拓扑排序不合法。否则，按照 DFS 结束时的逆序即为拓扑排序的结果。

拓扑排序题目：

考虑一个课程选修系统，每个课程有一个唯一的编号，并且一些课程有先修课程关系。设计一个算法，判断是否可以合理地安排课程的学习顺序，使得每个课程的先修课程都在其之前学习。

例如，有3门课程：1、2、3，它们的先修关系如下：

1. 课程 1 的先修课程为课程 2。
2. 课程 2 的先修课程为课程 3。
3. 课程 3 没有先修课程。

则这组课程的学习顺序为 3 -> 2 -> 1。

拓扑排序题解：

```
#include <iostream>
#include <vector>
#include <queue>

using namespace std;

bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {

    vector<int> inDegree(numCourses, 0);
    vector<vector<int>> adjList(numCourses);

    for (const auto& edge : prerequisites) {
        int from = edge[1];
        int to = edge[0];
        inDegree[to]++;
        adjList[from].push_back(to);
    }


    queue<int> zeroInDegreeQueue;
    for (int i = 0; i < numCourses; ++i) {
        if (inDegree[i] == 0) {
            zeroInDegreeQueue.push(i);
        }
    }


    while (!zeroInDegreeQueue.empty()) {
        int course = zeroInDegreeQueue.front();
        zeroInDegreeQueue.pop();

        for (int neighbor : adjList[course]) {
            inDegree[neighbor]--;
            if (inDegree[neighbor] == 0) {
                zeroInDegreeQueue.push(neighbor);
            }
        }
    }


    for (int degree : inDegree) {
        if (degree > 0) {
            return false;
        }
    }

    return true;
}

int main() {
    int numCourses = 3;
```

```
    vector<vector<int>> prerequisites = {{1, 2}, {2, 3}};

    if (canFinish(numCourses, prerequisites)) {
        cout << "It is possible to finish all courses." << endl;
    } else {
        cout << "It is impossible to finish all courses." << endl;
    }

    return 0;
}
```

这个题解使用了拓扑排序的思想。通过构建入度数组和邻接表，然后使用队列进行广度优先搜索，最终检查是否所有课程都被安排。如果所有课程都能被安排，那么拓扑排序存在，表示可以合理地安排课程的学习顺序。