模板+解析

DFS（深度优先搜索✦）和BFS（广度优先搜索）是图论中两个重要的算法。

dfs

其中DFS是一种用于遍历或搜索树✦或图的算法，BFS则是一种用于搜索或遍历树或图的算法。两种算法都有其自身的优点和缺点，应用于不同的场景中。 DFS（深度优先搜索） 深度优先搜索是一种用于**遍历**或**搜索树或图**的算法，其基本思路是从起始节点开始，沿着一条路径一直走到底，直到无法再走下去为止，然后回溯到上一个节点，继续走到另外一个路径，重复上述过程，直到遍历完所有节点。

DFS的实现方式可以采用**递归**或者**栈**来实现。下面是一个采用**递归方式**实现的DFS代码示例（C++）：

```
void dfs(int cur, vector<int>& visited, vector<vector<int>>& graph) {
    visited[cur] = 1; // 标记当前节点已经被访问
    // 处理当前节点cur
    for (int i = 0; i < graph[cur].size(); i++) {
        int next = graph[cur][i];
        if (!visited[next]) { // 如果下一个节点未被访问
            dfs(next, visited, graph); // 继续访问下一个节点
        }
    }
}
void dfsTraversal(vector<vector<int>>& graph) {
    int n = graph.size();
    vector<int> visited(n, 0); // 初始化访问数组
    for (int i = 0; i < n; i++) {
        if (!visited[i]) { // 如果当前节点未被访问
            dfs(i, visited, graph); // 从当前节点开始进行深度优先遍历✦
        }
    }
}
```

bfs

BFS（广度优先搜索） 广度优先搜索是一种用于搜索或遍历树或图的算法，其基本思路是从起始节点开始，依次遍历当前节点的所有**邻居节点**，然后再依次遍历邻居节点的所有邻居节点，直到遍历到目标节点或者遍历完所有节点。 BFS的实现方式可以采用队列来实现。下面是一个采用队列方式实现的BFS代码示例（C++）：

```
void bfsTraversal(vector<vector<int>>& graph) {
    int n = graph.size();
    vector<int> visited(n, 0); // 初始化访问数组
    queue<int> q;
    for (int i = 0; i < n; i++) {
        if (!visited[i]) { // 如果当前节点未被访问
            q.push(i); // 将当前节点加入队列
```

```
            visited[i] = 1; // 标记当前节点已经被访问
            while (!q.empty()) { // 循环遍历队列中的节点
                int cur = q.front();
                q.pop();
                // 处理当前节点cur
                for (int j = 0; j < graph[cur].size(); j++) {
                    int next = graph[cur][j];
                    if (!visited[next]) { // 如果下一个节点未被访问
                        q.push(next); // 将下一个节点加入队列
                        visited[next] = 1; // 标记下一个节点已经被访问
                    }
                }
            }
        }
    }
}
```

总结 DFS和BFS都是图论中常用的搜索算法✦，其应用广泛，例如在寻路、迷宫问题、拓扑排序✦、连通性等问题中都有应用。两种算法的实现方式不同，DFS采用递归或者栈实现，而BFS采用队列实现。在应用场景中，需要根据实际情况选择合适的算法来解决问题。