沈阳航空航天大学

人工智能学院

**实验报告**

|  |  |
| --- | --- |
| 课程名称 | 算法导论 |
| 专 业 | 物联网工程 |
| 班 级 | 物联网2202 |
| 学 号 | 223428010210 |
| 学生姓名 | 陈梓欣 |
| 指导教师 | 孙恩岩 |
| 实验时间 | 2024年6月2日 3、4节 |
| 实验地点 | 机械馆410-3 |

# 一、实验名称

图的建立和应用

# 二、实验目的

掌握图的存储结构，实现图的深度和广度优先遍历算法。

# 三、实验内容和要求

建立二叉树，并实现二叉树的遍历，先序遍历采用递归算法实现，层次遍历用非递归算法来实现。

# 四、实验环境

Microsoft Visual Studio 2022

# 五、实验设计

算法和数据结构设计

1、数据结构设计:

图的表示：采用邻接表来表示无向图。邻接表使用map存储，其中键为图的顶点，值为与该顶点相连的其他顶点的列表。这样可以高效地存储图结构并进行查找操作。

邻接表：使用C++中的map<char, list<char>>来表示，其中char表示顶点，list<char>表示与该顶点相连的所有顶点。

2、算法设计:

添加边（addEdge）：在邻接表中为两个顶点添加边。由于是无向图，所以需要在两个顶点的列表中分别添加对方。

打印邻接表（printAdjList）：遍历邻接表并打印每个顶点及其相邻的顶点，便于直观查看图结构。

深度优先搜索（DFS）：采用递归方式实现。使用一个辅助函数DFSUtil，从给定顶点开始遍历，标记已访问的顶点，并继续访问未被访问的相邻顶点。

广度优先搜索（BFS）：采用队列实现。从给定顶点开始遍历，使用队列保存待访问的顶点，并按层次访问所有相邻顶点。

3、网络拓扑结构设计

在本实验中，设计了一个包含9个顶点的无向图，其网络拓扑结构如下：

顶点：A, B, C, D, E, F, G, H, I

边：

A - B

A - D

A - E

A - F

B - C

C - D

E - D

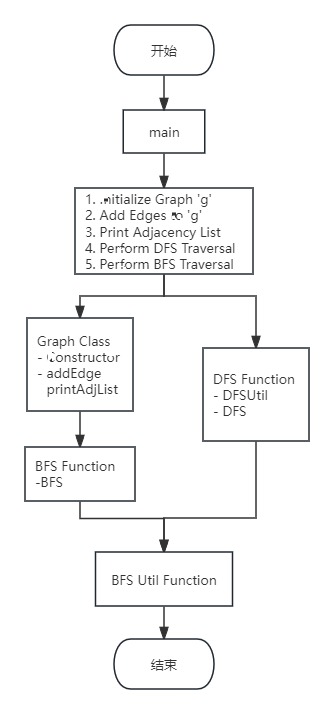
F - G

G - H

G - I

H - I

通过以上实验设计，直观地展示了图的邻接表表示、深度优先搜索和广度优先搜索的过程和结果，有助于理解图的基本操作及其遍历算法。



**图1 主模块流程图**

程序代码要加入必要注释，程序示例如下：

#include <iostream>

#include <list>

#include <queue>

#include <vector>

#include <map>

using namespace std;

class Graph {

int V;

map<char, list<char>> adj;

public:

Graph(int V) {

this->V = V;

}

void addEdge(char v, char w) {

adj[v].push\_back(w);

adj[w].push\_back(v);

}

void printAdjList() {

cout << "Adjacency List:" << endl;

for (auto const& pair : adj) {

char v = pair.first;

cout << v << " -> ";

for (auto x : adj[v])

cout << x << " ";

cout << endl;

}

}

void DFSUtil(char v, map<char, bool>& visited, vector<char>& result) {

visited[v] = true;

result.push\_back(v);

for (auto i = adj[v].begin(); i != adj[v].end(); ++i)

if (!visited[\*i])

DFSUtil(\*i, visited, result);

}

vector<char> DFS(char v) {

vector<char> result;

map<char, bool> visited;

for (auto const& pair : adj)

visited[pair.first] = false;

DFSUtil(v, visited, result);

return result;

}

vector<char> BFS(char s) {

vector<char> result;

map<char, bool> visited;

for (auto const& pair : adj)

visited[pair.first] = false;

queue<char> queue;

visited[s] = true;

queue.push(s);

while (!queue.empty()) {

s = queue.front();

result.push\_back(s);

queue.pop();

for (auto i = adj[s].begin(); i != adj[s].end(); ++i) {

if (!visited[\*i]) {

visited[\*i] = true;

queue.push(\*i);

}

}

}

return result;

}

};

int main() {

Graph g(9);

g.addEdge('A', 'B');

g.addEdge('A', 'D');

g.addEdge('A', 'E');

g.addEdge('A', 'F');

g.addEdge('B', 'C');

g.addEdge('C', 'D');

g.addEdge('E', 'D');

g.addEdge('F', 'G');

g.addEdge('G', 'H');

g.addEdge('G', 'I');

g.addEdge('H', 'I');

g.printAdjList();

char startVertex = 'A';

cout << "\nDepth First Traversal (starting from vertex " << startVertex << "): ";

vector<char> dfsResult = g.DFS(startVertex);

for (char v : dfsResult)

cout << v << " ";

cout << "\nBreadth First Traversal (starting from vertex " << startVertex << "): ";

vector<char> bfsResult = g.BFS(startVertex);

for (char v : bfsResult)

cout << v << " ";

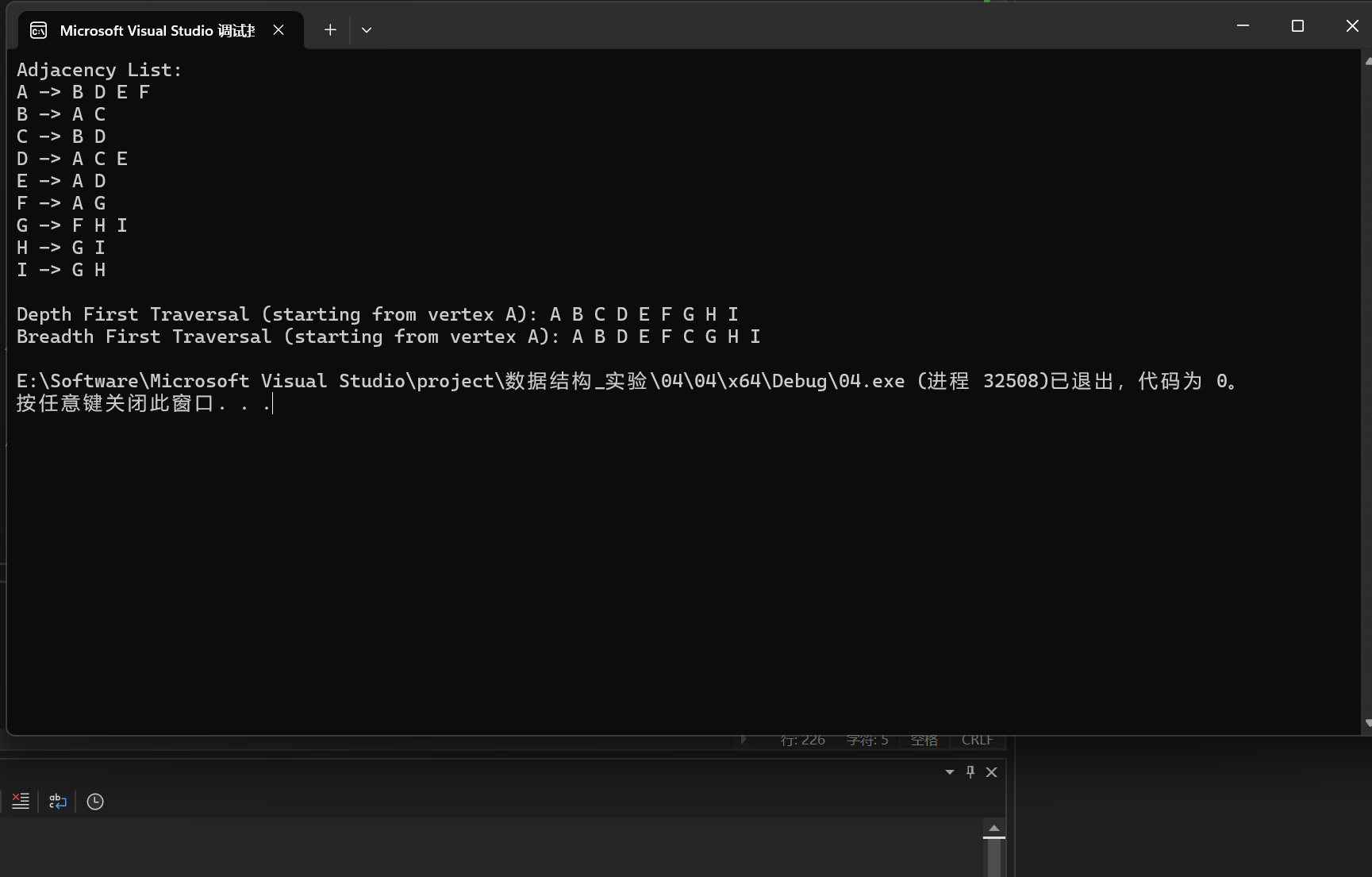
cout << endl;

return 0;

}

# 六、实验步骤及实验结果

本节用文字和屏幕截图详实记录实验的设计结果、完成过程（步骤）和每一步的测试结果



**图2 打印信息**

# 七、结论

本次实验通过构建无向图并实现深度优先搜索（DFS）和广度优先搜索（BFS）算法，加深了对图数据结构及其遍历方法的理解。实验采用C++的邻接表来表示图结构，并通过DFS和BFS算法进行遍历，从而验证了它们在不同应用场景中的特点。

邻接表是一种高效表示稀疏图的数据结构，能够支持快速的边操作。DFS利用递归实现，适用于路径探测、连通性分析等问题，但由于其栈空间消耗较大，适合处理规模较小的图。BFS利用队列实现，适用于层次遍历和寻找最短路径等问题，具有较高的时间复杂度，但由于其广度优先的特性，更适合处理大规模图。

实验结果表明，DFS和BFS各有优劣，具体选择应根据问题的特性进行。

总结如下：邻接表是表示稀疏图的优选数据结构；DFS适用于解决深度问题，如路径探测和连通性分析，BFS适用于广度问题，如层次遍历和最短路径寻找；DFS在空间复杂度上劣于BFS，适用规模较小的图，BFS在时间复杂度上劣于DFS，适用大规模图的广度问题。

此次实验验证了图论及相关算法在实际应用中的有效性，并通过实验结果加深了对图论及相关算法的理解。