实验报告电子版

**数据结构实训**

**—— 图的遍历与存储**

姓 名： 刘远明

学 号： 2220212113

指导老师： 韩凤

实验日期： 2024年4月15日

**大连海事大学电子信息科学技术专业**

**Academic Honesty Violations in Practice of Data Structures (PDS)**

In **PDS**, the MINIMUM penalty recommended for a violation of the Academic Honesty Policy will be a ZERO ON THE ASSIGNMENT, PROJECT or EXAM and a LOWERING OF YOUR FINAL GRADE for below what is otherwise earned. You may NOT withdraw from the course if found guilty. Some examples of academic misconduct in **PDS** include but are not limited to the following actions:

1. Picking up and using or discarding another student's written or computer output;

2. Using the computer account of another student;

3. Representing as one's own the work of another on assignments, quizzes, and projects;

4. Giving another student a copy of one's work on an assignment before the due date.

5. Copying work from online resources (Baidu,Chegg, google forums, etc.)

6. Posting work to online resources where other students can view your work.

All submissions will be checked for similarity. This code will check each submission for similarity between other student submissions, past student submissions, the solution manual, and online resources and postings. If your submission is flagged for a high level of similarity, it will be turned in for an academic honesty violation if deemed appropriate.

NOTE: Changing variable names, adding comments, or spacing will l result in a violation.

1. **实验目的**

（1）深入理解图的基本概念和程序设计方法。

（2）深入理解图的建立方法和深度优先遍历、图的广度优先遍历方法和实现方法。

（3）提高实际动手进行程序设计的能力。

1. **实验内容与要求**

**实验内容**：

1. 定义邻接矩阵存储结构或邻接表存储结构。
2. 按照建立一个带权有向图的操作需要，编写在邻接矩阵或邻接表存储结构下，带权有向图基本操作的实现函数（如初始化图、在图中插入一个结点、在图中插入一条边、在图中寻找序号为v的结点的第一个邻接结点、在图中寻找序号为v1结点的邻接结点v2的下一个邻接结点、图的深度优先遍历、图的广度优先遍历等）。
3. 设计一个测试主函数，首先创建一个图（有n个结点和e条边），然后打印图的n个结点信息和e条边信息，最后分别打印出图的深度优先遍历和广度优先遍历的结点信息序列。

**实验要求：**

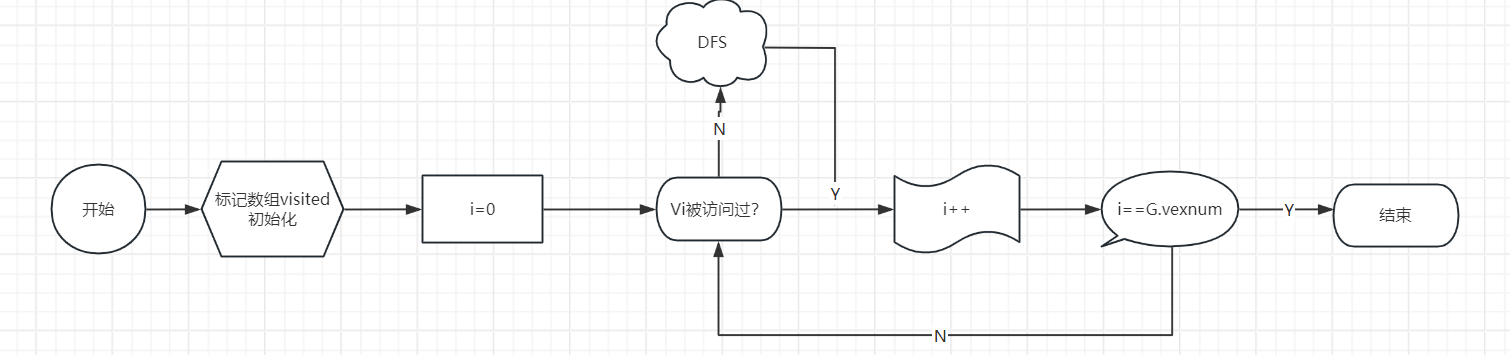
1. 遵守工程伦理和学术规范；
2. 解决的问题要来源于实际工程：要对日常生活中的一个具体应用进行抽象建模与求解，鼓励对国家重大工程中的一个具体点进行抽象建模与求解并融入工程管理理念。
3. 深入理解图的基本概念、建立方法和深度优先遍历、图的广度优先遍历方法，要对各种方案进行分析比较。
4. 要对实验结果进行和解释，并通过信息综合得到合理有效地结论。算法描述与流程
5. **算法描述与流程**
6. DFS遍历算法:

深度优先搜索(DFS)该方法类似树的先序遍历方法

基本思路如下：

从图的某一顶点V0出发，访问此顶点，然后依次从V0未被访问的邻接点出发，深度优先遍历图，直到图中所有与V0相邻的顶点都被访问到为止。

若此时图中仍有结点未被访问到，则选择图中未被访问到的结点作为起点，重复上述过程，直到图中所有顶点均被访问到为止。



1. BFS算法:

广度优先搜索类似树的层序遍历方法，要借助一个辅助队列实现

其基本思路如下:

从图的某个顶点V0出发，访问此结点之后，依次访问V0的所有未被访问过的邻接点，然后再分别从这些邻接点出发，广度优先遍历图，直到图中所有已被访问结点的邻接点都被访问到。

若此时图中还有顶点未被访问到，则另选一个图中未被访问到的顶点作为新起点，重复上述过程，直到图中所有结点都被访问到。

1. Dijkstra算法流程图

实验要求使用迪杰斯特拉算法来寻找最小路径，是从一个顶点到其余各顶点的最短路径算法，解决的是有向图中最短路径问题。迪杰斯特拉算法主要特点是以起始点为中心向外层层扩展，直到扩展到终点为止，是一种广度优先的搜索方法。主要思路就是沿着当前找到的最短路径的点不断延长路径，直到找到全部点的最短路径。完成这个算法需要有三个存储数据的数组，分别是：

1. int Path[MaxVex]：用来存储最短路径的，Path[i]存放的是能够以最短路径到达i的前驱结点
2. int Distance[MaxVex]：用来存放最短路径距离的数组，Distance[i]表示初始结点到结点i的距离。
3. int final[MaxVex]：用来标记是否已经找到到每个结点的最短距离0表示还没找到，1表示已经找到了
4. **算法实现**

**#include <iostream>**

**#include <cstdio>**

**#include <cstring>**

**#include<queue>**

**using namespace std;**

**#define MaxInt 32767**

**#define MVNum 100**

**int count1 = 0;**

**int a3 = 1;**

**typedef char VerTexType;**

**typedef int ArcType;**

**bool visited[MVNum];**

**typedef struct {**

**VerTexType vexs[MVNum];**

**ArcType arcs[MVNum][MVNum];**

**int vexnum, arcnum;**

**} AMGraph;**

**// 初始化图**

**void InitGraph(AMGraph\* G) {**

**memset(G->arcs, 0, sizeof(G->arcs)); // 初始化邻接矩阵，全部置为0**

**G->vexnum = 0;**

**G->arcnum = 0;**

**}**

**// 添加顶点**

**void AddVertex(AMGraph\* G, VerTexType vertex) {**

**int i = 0;**

**if (G->vexnum == MVNum) {**

**printf("图已满，无法添加新顶点！\n");**

**return;**

**}**

**for (i = 0; i < G->vexnum; i++)**

**{**

**if (G->vexs[i] == vertex)**

**{**

**printf("顶点重复\n");**

**return;**

**}**

**}**

**G->vexs[G->vexnum++] = vertex;**

**}**

**// 添加边**

**void AddEdge(AMGraph\* G, VerTexType start, VerTexType end, ArcType weight) {**

**int i, j;**

**for (i = 0; i < G->vexnum; i++) {**

**if (G->vexs[i] == start) {**

**break;**

**}**

**}**

**for (j = 0; j < G->vexnum; j++) {**

**if (G->vexs[j] == end) {**

**break;**

**}**

**}**

**if (i == G->vexnum || j == G->vexnum) {**

**printf("顶点不存在，无法添加边！\n");**

**return;**

**}**

**int temp = G->arcs[i][j];**

**G->arcs[i][j] = weight;**

**if(weight != 0 && temp == 0){ G->arcnum++; }**

**}**

**// 输出邻接矩阵**

**void PrintGraph(AMGraph\* G) {**

**printf("顶点数：%d，边数：%d\n", G->vexnum, G->arcnum);**

**printf(" ");**

**for (int i = 0; i < G->vexnum; i++) {**

**printf("%c ", G->vexs[i]);**

**}**

**printf("\n");**

**for (int i = 0; i < G->vexnum; i++) {**

**printf("%c ", G->vexs[i]);**

**for (int j = 0; j < G->vexnum; j++) {**

**printf("%d ", G->arcs[i][j]);**

**}**

**printf("\n");**

**}**

**}**

**int NextAdjVex(AMGraph\* G, int v, int w) {**

**for (int j = w + 1; j < G->vexnum; j++) {**

**if (G->arcs[v][j] != 0) { // 邻接矩阵中的非零元素表示存在边**

**return j; // 返回下一个邻接结点的序号**

**}**

**}**

**return -1; // 没有下一个邻接结点**

**}**

**int FirstAdjVex(AMGraph\* G, int v) {**

**if (v < 0 || v >= G->vexnum) {**

**printf("结点不存在！\n");**

**a3 = 0;**

**return -1;**

**}**

**else { a3 = 1; }**

**for (int j = 0; j < G->vexnum; j++) {**

**if (G->arcs[v][j] != 0) { // 邻接矩阵中的非零元素表示存在边**

**return j; // 返回邻接结点的序号**

**}**

**}**

**return -1; // 该结点没有邻接结点**

**}**

**void DFS(AMGraph\* G, int v)**

**{ //图G为邻接矩阵类型**

**count1++;**

**if (v > G->vexnum - 1) { printf("输入错误"); return; }**

**if ((count1 - G->vexnum)) { std::cout << G->vexs[v] << "->"; }**

**else { std::cout << G->vexs[v]; }**

**visited[v] = true; //访问第v个顶点**

**//依次检查邻接矩阵中v所在的行，即每个与v相邻的点**

**for (int w = 0; w < G->vexnum; w++)**

**if ((G->arcs[v][w] != MaxInt) && (!visited[w]))**

**DFS(G, w);**

**//G.arcs[v][w]!=MaxInt表示w是v的邻接点，如果w未访问，则递归调用DFS**

**}**

**void BFS(AMGraph\* G, int v) {**

**cout << G->vexs[v] << "->";**

**visited[v] = true;**

**queue<int> Q;**

**Q.push(v);**

**while (!Q.empty()) {**

**int u = Q.front();**

**Q.pop(); // 取出队头元素并删除**

**for (int w = FirstAdjVex(G, u); w >= 0; w = NextAdjVex(G, u, w)) {**

**if (!visited[w]) {**

**count1++;**

**cout << G->vexs[w];**

**if (count1 < G->vexnum-1) { cout << "->"; }**

**visited[w] = true;**

**Q.push(w);**

**}**

**}**

**}**

**}**

**int main() {**

**char input0;**

**char input1;**

**char input2\_0[2];**

**int input2\_1;**

**int input3;**

**int input4;**

**int input5;**

**AMGraph G;**

**InitGraph(&G);**

**// 添加顶点**

**AddVertex(&G, 'A');**

**AddVertex(&G, 'B');**

**AddVertex(&G, 'C');**

**AddVertex(&G, 'D');**

**// 添加边**

**AddEdge(&G, 'A', 'B', 3);**

**AddEdge(&G, 'B', 'C', 5);**

**AddEdge(&G, 'C', 'A', 2);**

**AddEdge(&G, 'C', 'D', 4);**

**// 输出邻接矩阵**

**PrintGraph(&G);**

**printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**printf("1.插入顶点\n2.插入边\n3.寻找序号为v的第一个邻接点\n4.图的深度优先遍历\n5.图的广度优先遍历\n6.打印图\n7.退出\n");**

**printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**while (true)**

**{**

**std::cout << "请输入要进行的操作:\n";**

**std::cin >> input0;**

**if (!(int(input0) - 55)) { break; }**

**switch (int(input0)) {**

**case 49:**

**{**

**std::cout << "请输入要插入的顶点:\n";**

**std::cin >> input1;**

**AddVertex(&G, input1);**

**PrintGraph(&G);**

**}**

**break;**

**case 50:**

**{**

**std::cout << "请输入要插入的边的横坐标(字符串):\n";**

**std::cin >> input2\_0[0];**

**std::cout << "请输入要插入的边的纵坐标(字符串):\n";**

**std::cin >> input2\_0[1];**

**std::cout << "请输入要插入的边的权重:\n";**

**std::cin >> input2\_1;**

**AddEdge(&G, input2\_0[0], input2\_0[1], input2\_1);**

**PrintGraph(&G);**

**}**

**break;**

**case 51:**

**{**

**std::cout << "请输入要查找的节点的序号:\n";**

**std::cin >> input3;**

**int adj = FirstAdjVex(&G, input3 - 1); // 结点 A 对应的序号是 0**

**if (a3)**

**{**

**if (adj != -1) {**

**printf("结点 %c 的第一个邻接结点是：%c\n", G.vexs[input3 - 1], G.vexs[adj]);**

**}**

**else {**

**printf("结点 %c 没有邻接结点！\n", G.vexs[input3 - 1]);**

**}**

**}**

**}**

**break;**

**case 52:**

**{**

**std::cout << "请输入深度优先搜索序号:\n";**

**std::cin >> input4;**

**DFS(&G, input4 - 1);**

**std::cout << "\n";**

**count1 = 0;**

**for (int i = 0; i <= G.vexnum; i++)**

**{**

**visited[i] = false;**

**}**

**}**

**break;**

**case 53:**

**{**

**std::cout << "请输入广度优先搜索序号:\n";**

**std::cin >> input5;**

**BFS(&G, input5 - 1);**

**std::cout << "\n";**

**count1 = 0;**

**for (int i = 0; i <= G.vexnum; i++)**

**{**

**visited[i] = false;**

**}**

**}**

**break;**

**case 54:**

**{**

**PrintGraph(&G);**

**}**

**break;**

**default:**

**printf("输入错误！\n");**

**continue;**

**}**

**}**

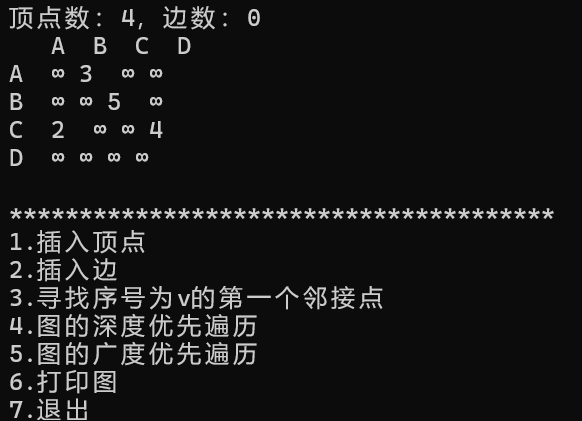
**return 0;**

**}**

1. **实验结果与分析**

**5.1测试用例**

输入数据文件格式如下：



邻接矩阵的无边结点值设为32767

最大结点数设为100

**5.2测试程序**

**int main() {**

**char input0;**

**char input1;**

**char input2\_0[2];**

**int input2\_1;**

**int input3;**

**int input4;**

**int input5;**

**AMGraph G;**

**InitGraph(&G);**

**// 添加顶点**

**AddVertex(&G, 'A');**

**AddVertex(&G, 'B');**

**AddVertex(&G, 'C');**

**AddVertex(&G, 'D');**

**// 添加边**

**AddEdge(&G, 'A', 'B', 3);**

**AddEdge(&G, 'B', 'C', 5);**

**AddEdge(&G, 'C', 'A', 2);**

**AddEdge(&G, 'C', 'D', 4);**

**// 输出邻接矩阵**

**PrintGraph(&G);**

**printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**printf("1.插入顶点\n2.插入边\n3.寻找序号为v的第一个邻接点\n4.图的深度优先遍历\n5.图的广度优先遍历\n6.打印图\n7.退出\n");**

**printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**while (true)**

**{**

**std::cout << "请输入要进行的操作:\n";**

**std::cin >> input0;**

**if (!(int(input0) - 55)) { break; }**

**switch (int(input0)) {**

**case 49:**

**{**

**std::cout << "请输入要插入的顶点:\n";**

**std::cin >> input1;**

**AddVertex(&G, input1);**

**PrintGraph(&G);**

**}**

**break;**

**case 50:**

**{**

**std::cout << "请输入要插入的边的横坐标(字符串):\n";**

**std::cin >> input2\_0[0];**

**std::cout << "请输入要插入的边的纵坐标(字符串):\n";**

**std::cin >> input2\_0[1];**

**std::cout << "请输入要插入的边的权重:\n";**

**std::cin >> input2\_1;**

**AddEdge(&G, input2\_0[0], input2\_0[1], input2\_1);**

**PrintGraph(&G);**

**}**

**break;**

**case 51:**

**{**

**std::cout << "请输入要查找的节点的序号:\n";**

**std::cin >> input3;**

**int adj = FirstAdjVex(&G, input3 - 1); // 结点 A 对应的序号是 0**

**if (a3)**

**{**

**if (adj != -1) {**

**printf("结点 %c 的第一个邻接结点是：%c\n", G.vexs[input3 - 1], G.vexs[adj]);**

**}**

**else {**

**printf("结点 %c 没有邻接结点！\n", G.vexs[input3 - 1]);**

**}**

**}**

**}**

**break;**

**case 52:**

**{**

**std::cout << "请输入深度优先搜索序号:\n";**

**std::cin >> input4;**

**DFS(&G, input4 - 1);**

**std::cout << "\n";**

**count1 = 0;**

**for (int i = 0; i <= G.vexnum; i++)**

**{**

**visited[i] = false;**

**}**

**}**

**break;**

**case 53:**

**{**

**std::cout << "请输入广度优先搜索序号:\n";**

**std::cin >> input5;**

**BFS(&G, input5 - 1);**

**std::cout << "\n";**

**count1 = 0;**

**for (int i = 0; i <= G.vexnum; i++)**

**{**

**visited[i] = false;**

**}**

**}**

**break;**

**case 54:**

**{**

**PrintGraph(&G);**

**}**

**break;**

**default:**

**printf("输入错误！\n");**

**continue;**

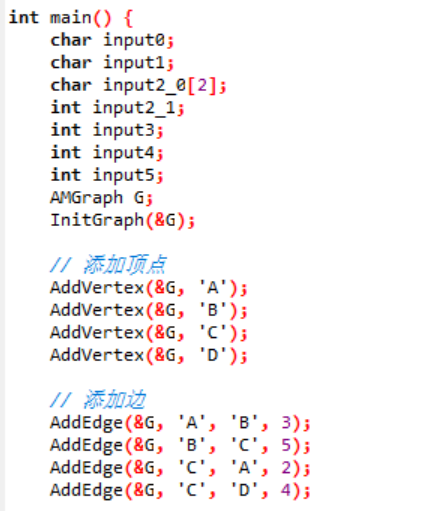
**}**

**}**

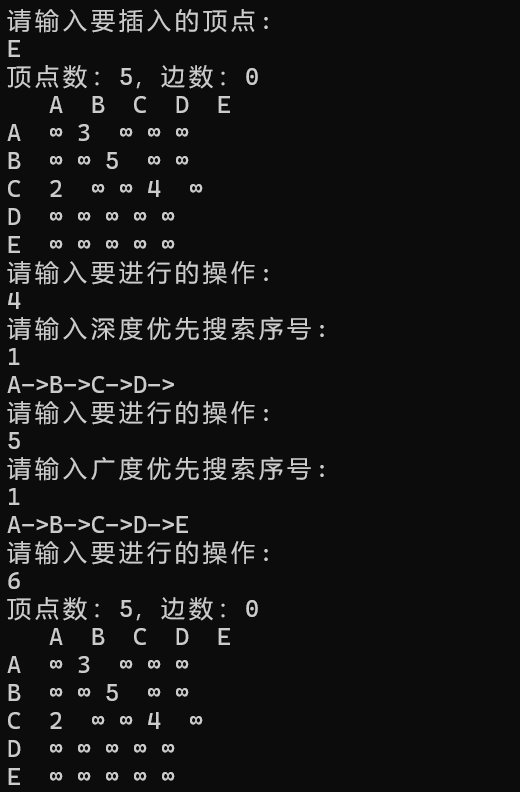
**return 0;**

**}**

**5.3初始参数设置**

****

**5.4实验结果**

****

**5.5实验分析**

1. **时间复杂度：**

①邻接矩阵的创建通常具有O(V^2)的时间复杂度，因为需要遍历每个顶点对并检查它们之间是否存在边。

②邻接表的创建具有O(V + E)的时间复杂度，其中V是顶点数，E是边数。因为对于每个顶点，我们遍历它的所有边并添加到其邻接表中。

③对于连通图，DFS的时间复杂度是O(V + E)，因为每个顶点只被访问一次，并且每条边也只被检查一次。

④对于非连通图，如果遍历所有分量，则最坏情况下的时间复杂度是O(V^2)，因为每个顶点可能需要被访问V次（在极端情况下，每个顶点都在一个单独的分量中）。

1. **空间复杂度：**

①邻接矩阵的空间复杂度是O(V^2)，因为需要存储一个V x V的矩阵，其中V是顶点的数量。

②邻接表的空间复杂度也是O(V + E)，因为我们需要为每个顶点存储一个邻接表，而邻接表的大小取决于该顶点的度数（相邻边的数量）。邻接表

③DFS的空间复杂度取决于递归的深度或显式栈的大小，最坏情况下是O(V)，即当图退化为链表时。

④BFS的空间复杂度主要由队列的大小决定，最坏情况下是O(V)，即当图是一个完全图时，队列中可能存储所有的顶点

1. **边界条件分析：**

①图为空：如果图是空的（没有顶点或边），那么任何遍历操作都应该能够正确处理这种情况，避免产生错误。

②连通性：对于非连通图，遍历算法需要能够处理多个分量，确保所有分量都被访问到。