**《数据结构》上机报告**

**2018 年 12 月 1 日**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **姓名：** | **赵得泽** | **学号：** | **1753642** | **班级：** | **电子2班** | **得分：** |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **实验题目** | | **图** | | |
| **问题描述** | | 图是一种描述多对多关系的数据结构, 图中的数据元素称作顶点，具有关系的两个顶点形成的一个二元组称作边或弧，顶点的集合V和关系的集合R构成了图，记作G=（V,R）。图又分成有向图，无向图，有向网，无向网。图的常用存储结构有邻接矩阵、邻接表、十字链表、邻接多重表。图的基本操作包括图的创建、销毁、添加顶点、删除顶点、插入边、删除边、图的遍历。 | | |
| **基本要求** | | 1. 练习邻接矩阵和邻接表的创建。 2. 遍历图的路径有深度优先搜索dfs和广度优先搜索bfs。 本题给定一个无向图，用dfs和bfs找出图的所有连通子集。存储结构采用邻接矩阵表示。 3. 本题给定一个无向图，用邻接表作存储结构，用dfs和bfs找出图的所有连通子集。 | | |
| **已完成基本内容（序号）：** | 1，2，3 | |
| **选做要求** | |  | | |
| **已完成选做内容（序号）** | |  |
| **数据结构**  **设计** | | **typedef char VertexType;**  **#define MAX\_VERTEX\_NUM 20**  **//typedef enum { DG, DN, UDG, UDN } GrapgKind;**  **typedef int AdjMatrix[MAX\_VERTEX\_NUM][MAX\_VERTEX\_NUM];**  **//邻接矩阵类型**  **typedef struct {**  **VertexType vexs[MAX\_VERTEX\_NUM]; //顶点表**  **AdjMatrix arcs; //邻接矩阵**  **int vexnum, arcnum; //图的顶点数和边/弧数**  **int Graphkind;**  **} MGraph;**  **typedef struct ArcNode**  **{**  **int adjvex;//弧指向的顶点的位置**  **ArcNode \*nextarc;//指向下一个与该顶点邻接的顶点**  **int info;//弧的相关信息**  **}ArcNode;//边表结点**  **typedef struct VNode**  **{**  **VertexType data;//用于存储顶点**  **ArcNode \*firstarc;//指向第一个与该顶点邻接的顶点**  **}VNode, AdjList[MAX\_VERTEX\_NUM];//表头节点，顺序表存储**  **typedef struct**  **{**  **AdjList vertices;//邻接表**  **int vexnum, arcnum;//边数，顶点数**  **int kind;//图的种类**  **}ALGraph;**  **struct SqQueue {**  **VertexType \*base;**  **int front, rear;**  **};**  本实验主要运用的数据结构是图，通过图来存储数据，一种方法是邻接矩阵法，适用于边比较稠密的图，另一种方法是邻接表法，主要适用于边比较稀疏的图。邻接表法是用一个顶点表来存放顶点数据，其相当于一个顺序表；还有一个边结点表用来存储与顶点相邻的顶点的数据，结构体包括顶点信息info和顶点下标adjvex和指向下一个与该顶点邻接的顶点的指针nextarc其相当于一个链式表。还有一个结构体表示图，用来将前两个表封装起来使用，便于操作图。  邻接矩阵法则是将有相邻关系的vi,vj顶点对应矩阵的i行j列或j行i列置1或者权值；  本实验还用到的数据结构是队列，队列的使用主要是bfs，不能用递归实现，所以用队列实现。 | | |
| **功能(函数)说明** | | **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **函数功能：定位顶点**  **函数说明：通过遍历查找与目标定点相同的顶点的编号返回即可。**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **int LocateVertex(Graph& G, VertexType v)**  **{**  **int i;**  **for (i = 1; i <= G.vexnum; i++)**  **if (G.vertices[i].data == v)**  **return i;**  **return -1;**  **}**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **函数功能：创建图**  **函数说明：创建图有邻接表法和邻接矩阵法；邻接表法就是用一个结构数**  **组即顺序表存储顶点，称为顶点表；创建边结点表存储在与顶点相关联的**  **一条边（弧）的另一个顶点，以链表的方式进行头插法建立。邻接矩阵法**  **就是用一个方阵来存储两个顶点是否相邻的信息，若i,j顶点相邻，就将**  **矩阵的i行j列元素置1，（或者为权值，若是无向图，j行i列也做此操作，**  **有向网，无向网做相应的操作）**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **void CreateGraph(MGraph &G1, ALGraph &G2)**  **{**  **int i, j, k;**  **VertexType v1, v2;**  **ArcNode \*p;**  **int w;**  **cin >> G1.Graphkind;**  **G2.kind = G1.Graphkind;**  **int n, m;**  **cin >> n >> m;**  **G1.vexnum = n;**  **G2.vexnum = n;**  **G1.arcnum = m;**  **G2.arcnum = m;**  **cin >> G1.vexs;**  **for (i = 0; i < G1.vexnum; i++)**  **{**  **G2.vertices[i].data = G1.vexs[i];**  **G2.vertices[i].firstarc = NULL;**  **}**  **for (i = 0; i < G1.vexnum; i++)**  **for (j = 0; j < G1.vexnum; j++)**  **G1.arcs[i][j] = 0;**  **if (G1.Graphkind == 1 || G1.Graphkind == 3)**  **for (k = 0; k < G1.arcnum; k++)**  **{**  **cin >> v1 >> v2;**  **i = LocateVertex\_S(G1, v1);**  **j = LocateVertex\_S(G1, v2);**  **if (G1.Graphkind == 1)**  **{ /\*j为入i为出创建邻接链表\*/**  **G1.arcs[i][j] = 1;**  **p = new ArcNode;**  **p->adjvex = j;**  **p->nextarc = G2.vertices[i].firstarc;**  **G2.vertices[i].firstarc = p;**  **}**  **if (G1.Graphkind == 3)**  **{**  **G1.arcs[i][j] = 1;**  **G1.arcs[j][i] = 1;**  **/\*j为入i为出创建邻接链表\*/**  **p = new ArcNode;**  **p->adjvex = j;**  **p->nextarc = G2.vertices[i].firstarc;**  **G2.vertices[i].firstarc = p;**  **/\*i为入j为出创建邻接链表\*/**  **p = new ArcNode;**  **p->adjvex = i;**  **p->nextarc = G2.vertices[j].firstarc;**  **G2.vertices[j].firstarc = p;**  **}**  **}**  **if (G1.Graphkind == 2 || G1.Graphkind == 4)**  **for (k = 0; k < G1.arcnum; k++)**  **{**  **cin >> v1 >> v2 >> w;**  **i = LocateVertex\_S(G1, v1);**  **j = LocateVertex\_S(G1, v2);**  **if (G1.Graphkind == 2)**  **{/\*j为入i为出创建邻接链表\*/**  **G1.arcs[i][j] = w;**  **p = new ArcNode;**  **p->adjvex = j;**  **p->info = w;**  **p->nextarc = G2.vertices[i].firstarc;**  **G2.vertices[i].firstarc = p;**  **}**  **if (G1.Graphkind == 4)**  **{**  **G1.arcs[i][j] = w;**  **G1.arcs[j][i] = w;**  **/\*j为入i为出创建邻接链表\*/**  **p = new ArcNode;**  **p->adjvex = j;**  **p->info = w;**  **p->nextarc = G2.vertices[i].firstarc;**  **G2.vertices[i].firstarc = p;**  **/\*i为入j为出创建邻接链表\*/**  **p = new ArcNode;**  **p->adjvex = i;**  **p->info = w;**  **p->nextarc = G2.vertices[j].firstarc;**  **G2.vertices[j].firstarc = p;**  **}**  **}**  **}**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **函数功能：打印图**  **函数说明：打印图有两种类型，一种是邻接矩阵类型DisplayMGraph，一**  **类是邻接表类型DisplayLGraph,两种类型的图打印方式均需要做一些变动，**  **见下代码。**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **void DisplayLGraph(ALGraph &G)**  **{**  **int i;**  **ArcNode \*p;**  **cout << endl;**  **for (i = 0; i < G.vexnum; i++)**  **{**  **cout << G.vertices[i].data << "-->";**  **p = G.vertices[i].firstarc;**  **while (p)**  **{**  **if (G.kind == 1 || G.kind == 3)**  **cout << p->adjvex << " ";**  **else if (G.kind == 2 || G.kind == 4)**  **cout << p->adjvex << "," << p->info << " ";**  **p = p->nextarc;**  **}**  **cout << endl;**  **}**  **}**  **void DisplayMGraph(MGraph &G)**  **{**  **int i, j;**  **for (i = 0; i < G.vexnum; i++)**  **cout << G.vexs[i] << " ";**  **cout << endl;**  **for (i = 0; i < G.vexnum; i++)**  **{**  **for (j = 0; j < G.vexnum; j++)**  **cout << setw(4) << G.arcs[i][j];**  **cout << endl;**  **}**  **}**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **函数功能：DFS递归遍历图（邻接矩阵/邻接表）**  **函数说明：建立一个标记数组vis[],若该顶点访问过就标记为1，否则置零；**  **递归的时候，从某个指定定点开始进行，遍历第一个邻接点，然后再遍历**  **邻接点的第一个邻接点......直到某个顶点没有未访问的邻接点，就退**  **出该层递归，然后再进入上一层递归，直到所有的顶点都没有未访问的顶点**  **dfs遍历结束。每一次大的for循环结束就找到了一个连通子集。**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **void DFS\_recur(MGraph &G, int v, int count1)**  **{**  **int w;**  **vis[v] = 1;**  **count1++;**  **if (count1 == 1)**  **cout << G.vexs[v];**  **else**  **cout << " " << G.vexs[v];**  **for (w = 0; w < G.vexnum; w++)**  **if (!vis[w] && G.arcs[v][w] == 1)**  **DFS\_recur(G, w, count1);**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **邻接表法：**  **p = G.vertices[v].firstarc;**  **while (p)**  **{**  **if (!vis[p->adjvex])**  **DFS\_recur(G, p->adjvex, count1);**  **p = p->nextarc;**  **}**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **}**  **void DFSTraverse(MGraph &G, int count1)**  **{**  **int v;**  **for (v = 0; v < G.vexnum; v++)**  **vis[v] = 0;**  **for (v = 0; v < G.vexnum; v++)**  **{**  **if (!vis[v])**  **{**  **cout << "{";**  **count1 = 0;**  **DFS\_recur(G, v, count1);**  **cout << "}";**  **}**  **}**  **}**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **函数功能：BFS非递归遍历图（邻接矩阵/邻接表）**  **函数说明：建立一个标记数组vis[],若该顶点访问过就标记为1，否则置零；**  **遍历的时候，首先找到一个未访问的顶点，入队，然后如果队列不空，就**  **一直循环，并且每次循环都要出队一个元素，再在以该元素为顶点的链表**  **或者矩阵的某一行中进行查找，如果某个顶点未访问，则进行访问，然后**  **vis置1，表示已经访问，再入队，直到队为空，退出本次循环，再查找顶**  **点表中的第二个顶点或者矩阵中的第二行，以此类推得到广度优先搜索序**  **列，每次大的循环其实就是一个连通子集。**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **void BFSTraverse(MGraph &G)**  **{**  **int v;**  **SqQueue Q;**  **int w = 0;**  **VertexType u = 0;**  **Q.base = new VertexType[MAX\_VERTEX\_NUM];**  **Q.front = Q.rear = 0;**  **for (v = 0; v < G.vexnum; v++)**  **vis[v] = 0;**  **for (v = 0; v < G.vexnum; v++)**  **{**  **if (!vis[v])**  **{**  **cout << "{";**  **cout << G.vexs[v];**  **vis[v] = 1;**  **EnQueue(Q, G.vexs[v]);**  **while (!QueueEmpty(Q))**  **{**  **DeQueue(Q, u);**  **int u1 = LocateVertex(G, u);**  **for (w = 0; w < G.vexnum; w++)**  **if (!vis[w] && G.arcs[u1][w] == 1)**  **{**  **vis[w] = 1;**  **cout << " " << G.vexs[w];**  **EnQueue(Q, G.vexs[w]);**  **}**  **}**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **邻接表法：**  **p = G.vertices[u1].firstarc;**  **while (p)**  **{**  **if (!vis[p->adjvex])**  **{**  **vis[p->adjvex] = 1;**  **cout << " " << G.vertices[p->adjvex].data;**  **EnQueue(Q, G.vertices[p->adjvex].data);**  **}**  **p = p->nextarc;**  **}**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **cout << "}";**  **}**  **}**  **}** | | |
| **开发环境** | | **Win10,vs2017,C++高级程序语言设计** | | |
| **调试分析** | | **Problem1：**    **Problem2：**    **Problem3：** | | |
| **心得体会** | 本次实验主要是图的建立及其遍历，其中建立有邻接表法和邻接矩阵法，图的遍历有深度优先和广度优先法。这四种方法的结合使用让我充分认识到图本质意义，对图的结构有了更加深刻的认识。  其中两种建立方法所得到的连通子集的序列是不一样的，对于两种方法种的任何一种方法，其DFS递归遍历图的方法是一样的，都是通过两层循环，一层遍历顶点表/按行遍历，一层就是递归遍历  直到一个连通子集被遍历结束退出这层循环。而对于BFS遍历来说，主要是利用队列来实现，这和二叉树的层次遍历极为相似，通过对每个顶点相邻的顶点全部访问完并入队，然后再一一出队访问其相邻顶点，如此循环，直到所有顶点访问到，结束广度优先遍历。  这两种方法是图的两种重要遍历方法，仔细理解并区分它们之间的不同，在以后问题的解决中将非常方便。 | | | |