**实 验 报 告**

|  |  |  |
| --- | --- | --- |
| **课程名称**：数据结构 | **班级**：软工21101 | **实验成绩**： |
| **实验名称**：图的应用-邻接表存储结构 | **学号**：2105006207 | **批阅教师签字：董傲霜** |
| **实验编号**：实验十 | **姓名**：方福涛 | **实验日期：**2022 年 11 月 18 日 |
| **指导教师**：董傲霜 | **组号**：wuyanzu | **实验时间**： 时 分－ 时 分 |

1. **实验设计思想**

说明图的邻接表存储结构的设计思想。

1、图中顶点用一个一维数组存储，另外，对于顶点数组中，每个数据元素还需要存储指向第一个邻接点的指针，以便于查找该顶点的边信息。

2、图中每个顶点vi的所有邻接点构成一个线性表，由于邻接点的个数不定，所以用单链表存储，无向图称为顶点vi的边表，有向图称为顶点vi作为弧尾的出边表。

**二、程序说明**

1. 给出图的邻接表存储结构定义及其属性含义的说明；

#define  MAX\_VERTEX\_NUM 20//最大顶点个数

#define  VertexType int//顶点数据的类型

#define  InfoType int//图中弧或者边包含的信息的类型

typedef struct ArcNode{

    int adjvex;//邻接点在数组中的位置下标

    struct ArcNode \* nextarc;//指向下一个邻接点的指针

    InfoType \* info;//信息域

}ArcNode;

typedef struct VNode{

    VertexType data;//顶点的数据域

    ArcNode \* firstarc;//指向邻接点的指针

}VNode,AdjList[MAX\_VERTEX\_NUM];//存储各链表头结点的数组

typedef struct {

    AdjList vertices;//图中顶点的数组

    int vexnum,arcnum;//记录图中顶点数和边或弧数

    int kind;//记录图的种类

}ALGraph;

1. 给出你设计的函数的结构说明（函数，参数，执行的结果）

bool visited[MaxVex] ;   // 主要针对的是图递归的算法

void InitStack( Stack \* S )

bool is\_stack\_empty( Stack \* S )  // 判断栈是否为空

bool is\_stack\_full( Stack \* S )  // 判断栈是否已满

void Push( Stack \* S , int elem )  // 入栈

void Pop( Stack \* S , int \* elem ) // 出栈

void InitQueue( Queue \* Q )  // 初始化队列

bool is\_empty( Queue \* Q )  // 判断队列是否为空

bool is\_full( Queue \* Q )  // 判断队列是否已满

void EnQueue( Queue \* Q , int elem )  // 入队

void DeQueue( Queue \* Q , int \* elem )  // 出队

// 初始化图

void InitGraph( ALGraph \* G )

// 创建图

void CreateGraph( ALGraph \* G )

// 广度优先遍历

void bfs( ALGraph \* G )

//利用队列实现邻接表的广度优先遍历

void BFS\_Queue( ALGraph \* G )

//利用栈实现图的邻接表结构的深度优先遍历  非递归

void DFS( ALGraph \* G )

// 初始化全局变量visited

void InitVisit( ALGraph \* G , bool \* visited )

// 存储结构为邻接表的图的深度优先遍历的递归实现

// 从下标为v的结点开始遍历

void DFS\_Recursion( ALGraph \* G , int v )

// 图的递归遍历

// 为防止图为非连通图，必须要有此函数

void DFS\_Traverse\_Recursion( ALGraph \* G )

1. 说明图的深度优先和广度优先遍历实现步骤（处理步骤，可用文字也可画流程图）。

**深度优先遍历：**

（1）首先任意选定一个未被访问过的顶点V作为起始顶点，并将其标记为已访问过；想一下我们如何知道这个点是否访问呢。我们可以加一个数组，数组中的元素来表示元素是否访问。

（2）然后搜索与顶点V邻接的所有顶点，判断这些顶点是否被访问过，如果有未被访问过的顶点，则任选一个顶点W进行访问；再选取与顶点W邻接的未被访问过的任一个顶点并进行访问，依次重复进行。当一个顶点的所有的邻接顶点都被访问过时，则依次回退到最近被访问的顶点。若该顶点还有其他邻接顶点未被访问，则从这些未被访问的顶点中取出一个并重复上述过程，直到与起始顶点V相通的所有顶点都被访问过为止。

（3）若此时图中依然有顶点未被访问，则再选取其中一个顶点作为起始顶点并访问之，转（2）。反之，则遍历结束。

**广度优先遍历：**

（1）首先第一个要访问的节点入队，也就是A入队，开始访问A，A出队A的邻接点B,C入队。

（2）开始访问B,B出队，B的所有未访问过的邻接点入E,F入队

（3）开始访问C,C出队，C的所有未访问过的邻接点入D入队.

（4）开始访问E，E出队，

（5）开始访问F,F出队。

（6）开始访问D,D,出队

**三、实验环境**

Windows10、DEV-C++

**四、实验过程分析**

实验调试过程中的问题及改正的事件举例说明（只要调程序就不可能没有错误，现象、原因、修正方法，可以截图但不能过多）。

1.遍历边某个顶点边的时候的遍历顺序正好与读入时候的顺序相反，所以在输入的时候要注意。

2.因为邻接结点存储结构是链表的形式，且我们能通过顶点的指针域（firstedge）以及边表结点间的指针域（next）找到所以边表结点，故在邻接表中不需要定义邻接结点的结构。

**五、实验结果总结**

1、阐述图的邻接表存储结构上实现深度优先和广度优先遍历操作的算法时间复杂度和空间复杂度。

**深度优先遍历和广度优先遍历算法分析：**

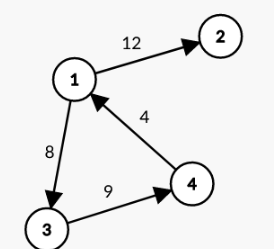
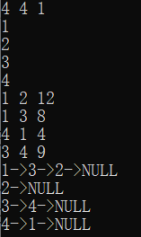
**时间复杂度：O(n+e)**

**空间复杂度：O(n)**

2、给出你的测试方法和测试截图（不能超过三个截图）；

**利用问题引出测试：**  
设有一个带权图G，设计一个算法，输入n个顶点数据和e条边的端顶点号 i、j及其权值w，构建它的邻接表存储表示。

**运行结果（对代码进行一定更改）：**  
第一行输入：结点数n 边数e 有向图还是无向图d  
接下来n行：输入顶点值  
接下来e行：输入边的顶点及对应权值  
最后输出该图对应的邻接表



**六、附录**

1. 意见和建议（没有可不写）。

2. 思考题：

回答以下问题：

1. 图的邻接表存储结构适合什么样的图进行存储？

**稀疏图**

1. 无向图和有向图的邻接表各有什么特点？

**无向图：**

存在逆反的点，比如，有1，2，3，4这4个结点，有 1 ->2,指向  
也可能有 2-1>的指向。

**有向图**：

不存在逆反的点，比如，有1，2，3，4这4个结点，假如有1->2这个指向的话便不会存在 2->1这个指向。

**七、打分表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 项目 | 考核点 | 分数 | **得分** | 备注 |
| 程序 | 逻辑是否正确  程序可读性  创新点 | 50 | **0** |  |
| 报告完整性 | 实验过程阐述是否完整  测试数据设计是否合理  运行结果是否正确 | 40 | **0** |  |
| 调试问题及解决方法 | 是否对调试过程问题进行阐述 | 5 | **0** |  |
| 思考题目 | 回答是否正确 | 5 | **0** |  |
| 合计 |  | 100 | **0** |  |