# 4基于邻接表的图实现

## 4.1 问题描述

**4.1.1 总述**

通过邻接表作为图的物理结构，创建一个简易的菜单演示系统。该系统定义了创建无向图，销毁图，查找顶点，顶点赋值，获得第一邻接点，获得下一邻接点，插入顶点，删除顶点，插入删除弧，深度优先搜索，广度优先搜索等运算对应的函数，并给出了相应的操作提示，并可选择以文件的形式进行存储和加载。在该系统中，可以通过选择序号来初始化并操作不同的图。

**4.1.2附加功能**

在本程序中，除了完成了无向无权图的基本功能外，还定义了在无向网，有向网上的操作；

附加功能：

1）选择15号功能，可以以邻接表的形式打印图的信息，包括图的所有顶点的关键字，以及每一个结点后继的邻接点的位序；

2）选择16号功能，输入一个结点作为源点，输出所有其他点到源点的最短路径长，如果没有路径会打印信息；

3）选择17号功能，判断当前操作的图是否是一个二分图，在二分图中，所有的顶点可以被分为两个点集，要求每一个弧的两端的结点都能恰好分别进入两个点集；

4）选择18号功能，可以计算当前操作的图的连通分量的个数；

5）选择19号功能，输入有向网中的所有结点的关键字和每一条边的起点和终点，边权，以此来构建一个有向带权图；

6）选择20号功能，通过迪杰斯特拉算法计算所选择的源点到其他所有点的最短路径，当结点与源点间没有有向通路时，会打印相对信息；

7）选择21号功能，输入无向网中的所有结点的关键字和所有边连接的端点的位序，边权，以此来构建一个无向带权图；

8）选择22号功能，通过prim算法得到这个无向带权图的最小生成树的总权值；

9）选择23号功能，可以对当前有向网求出拓扑排序，如果网中有环则没有拓扑排序。

**4.2 系统设计**

**4.2.1 数据的物理结构**

1）无向无权图：

typedef struct {

KeyType key;

char others[20];

} VertexType; //顶点类型定义

图的顶点类型包括顶点的关键字和顶点的内容others数组；

typedef struct ArcNode { //表结点类型定义

int adjvex; //顶点位置编号

struct ArcNode\* nextarc; //下一个表结点指针

} ArcNode;

表结点结构应包含顶点的位序，以及指向下一个结点的指针；

typedef struct VNode { //头结点及其数组类型定义

VertexType data; //顶点信息

ArcNode\* firstarc; //指向第一条弧

} VNode, AdjList[MAX\_VERTEX\_NUM];

头结点应包含顶点的data域，以及指向第一个邻接点的指针；

typedef struct { //邻接表的类型定义

AdjList vertices; //头结点数组

int vexnum, arcnum; //顶点数、弧数

GraphKind kind; //图的类型

char name[20];

} ALGraph;

无向图结构应包含头结点数组，图中的顶点数，弧数，图的类型，图的名称；

typedef struct {

ALGraph G[20];

int listsize;

}GList;

图列表结构应包含ALGraph型的数组，以及列表的长度。

2）无向网：

typedef struct {

int key;//结点关键字

}UDNnode;

无向网结点的内容应是结点的关键字；

typedef struct {

int node1, node2;//边连接的两结点位序

int length;//边权

}UDNedge;

无向网的边应包含边所连接的两个结点的位序以及边权；

typedef struct {

int NodeSum = 0;//结点总数

int EdgeSum = 0;//边总数

UDNnode n[100];

UDNedge e[500];

}UDNet;

无向网类型应包括图的结点总数，边的总数，结点类型的数组，边类型的数组；

3）有向网

typedef struct {

int key;//结点的关键字

}DNnode;

有向网结点结构包含结点的关键字；

typedef struct {

int start;//边的起点

int end;//边的终点

int length;//边的权重

}Edge;

边的结构应包含有向边的起点结点的位序，终点结点的位序，以及权重；

typedef struct {

int NodeSum = 0;//有向带权图中的结点数目，初始值为0

int EdgeSum = 0;

DNnode n[100];//结点数组

Edge e[500];//边的数组

}Net;

有向网结构应包含结点总数，边总数，结点类型的数组，边类型的数组。

**4.2.2 函数运算**

程序以函数的形式定义了创建无向图，销毁图，查找顶点，顶点赋值，获得第一邻接点，获得下一邻接点，插入顶点，删除顶点，插入删除弧，深度优先搜索，广度优先搜索，判断二分图，求最小路径，生成有向网，生成无向网，迪杰斯特拉算法等运算对应的函数，在进行函数功能的实现时都会有对应的输入提示与结果表现。

函数的定义：

1）创建无向图（CreateGraph）：由存储着图中的结点关键字和所有边的连接信息的数组来构建一个图，图的表现形式为邻接表，要求图的所有顶点关键字具有唯一性；

2）销毁图（DestroyGraph）；将邻接表表示的无向图销毁，释放所有结点和指针域的空间；

3）查找顶点（LocateVex）：根据给定的结点的关键字，在已构建的图G中查找这个结点，查找成功返回位序下标，否则返回-1；

4）顶点赋值（PutVex）：首先根据所给定的结点关键字查找到要重新赋值的结点，如果能够找到结点，那么用新的VertexType型变量进行更换完成赋值，否则返回ERROR；

5）获得第一邻接点（FirstAdjvex）；首先根据给定的关键字在邻接表中进行查找，如果能够查找到对应结点，当它有邻接结点时返回第一邻接点下标否则返回-1；

6）获得下一邻接点（NextAdjvex）：首先根据给定关键字1查找对应结点，然后在后继邻接结点的链表中找到关键字2所在结点，返回它的下一个结点位序；

7）插入结点（InsertVex）：在图的邻接表中的最后位序新添加一个结点变量，同时图的结点总数增加1；

8）删除顶点（DeleteVex）：删除所给定的关键字所在的结点，包括出现在其他结点的邻接点域；

9）插入弧（InsertArc）：根据给定的两个关键字，连接这两个结点作为一条新的边，修改后继邻接结点域，图的弧总数增加1；

10）删除弧（DeleteArc）：首先根据给定的两个关键字找到对应结点的位序，删除他们间的弧，清除后继邻接结点的链表中的结点域，图的弧总数减少1；

11）深度优先搜索（DFSTraverse）：以位序为0的结点为起点，输出这个图的深度优先搜索序列；

12）广深度优先搜索（BFSTraverse）：以位序为0的结点为起点，输出这个图的广度优先搜索遍历的序列；

13）文件保存（SaveGraph）：将图的所有顶点的关键字，所有边所连接的结点的位序对存储在一个文件里；

14）文件读取（LoadGraph）：根据一个存储了图的结点关键字以及所有边的连接信息的文件来构建一个无向图；

15）打印图（PrintGraph）：打印图的邻接表信息，每一行包含对应位序的结点关键字与内容，后面接着打印邻接顶点的位序（如果有的话）；

16）最短路径（getShortestPath）：以一各给定了关键字的点为源点，输出其他所有结点与源点间的最短路径列表；

17）二分图判断（BipartiteJudge）：判断当前操作的图是否是一个二分图；

18）连通分量数量（getComponentNum）：计算当前的图的连通分量的数量；

19）创建有向带权图（CreateDN）：根据存储了有向图的所有结点关键字，边连接的起点结点位序，终点结点位序，边权的数组来创建一个有向带权图；

20）迪杰斯特拉算法（Dijkstra）：使用迪杰斯特拉算法得到所有其它结点待给定源点的最短路径长度，若结点与源点间没有有向通路，会打印相应的信息；

21）创建无向有权图（CreateUDN）：根据存储了无向有权图的所有结点关键字，所有边连接的结点位序以及边权的数组来创建一个无向有权图（无向网）；

22）普利姆算法(Prim)：根据prim算法得到当前操作的无向有权图的最小生成树的总权值。

23）拓扑排序（TopoSort）：计算有向网的拓扑排序结果，给出拓扑排序序列。

**4.2.3 输入与输出说明**

进入系统后首先出现功能菜单，首先输入所要操作的线性表的序号，这里允

许在1~20号上操作；菜单会罗列23个选择项，系统会提示首先选择功能23来选择所要操作的图的序号，选择完毕后就可以选择功能对应的序号进行相应操作，倘若不进行所要操作的图的序号选择，则会默认在序号为1的图上进行操作，创建的无向网与有向网作为附加功能不在图数组中存储而是独立在外。

系统通过switch-case语句进行对应的功能操作，每一个操作完毕后按回车或其他字符跳出当前操作，继续执行while循环选择功能并实现，在每一次操作结束后系统都会显示操作的结果。对于图的各种操作，可以通过功能15的打印图的邻接表来验证每一个功能实现是否成功。当在系统中输入0后会退出系统。

当要在另一个图中操作时只要输入23号重新选择操作位序num的值即可。

## 4.3 系统实现

**4.3.1 环境说明**

本系统在Microsoft Visual Studio2019上执行，编译器为MSVC++ 14.2 \_MSC\_VER == 1920.并在其上编译运行。

**4.3.2 预定义变量**

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define MAX\_VERTEX\_NUM 20

#define inf 1000

typedef int status;

typedef int KeyType;

typedef enum { DG, DN, UDG, UDN } GraphKind;

变量说明：

VertexType V[30], value;//V用于存储输入，value记录单个结点的内容

KeyType VR[100][2]; char FileName[20]; //VR记录边连接的顶点的位序，FileName为文件读写时的文件名

GList Graph; //GList型对象

Net net; UDNet UDnet; //Net型和UDNet型对象

int i; int num = 1;//所操作的图的序号，默认为1

int u;//用来表示结点的序号

int v, w;//用来表示单个结点的关键字

int ShortestPath[100];//记录最短路径的数组

int DNkey[100]; //记录有向(无向)带权图的结点的关键字

int se[500][2];//记录有向边的起点与终点或无向边连接的结点位序

int weight[500];//记录权值

**4.3.3 基于邻接表的图的实现**

本实验通过结构体GList来存储多个图的信息，进入系统后选择23号功能来对所要操作的图的序号进行选择，选择完毕后即可进行多种功能的演示。在每一次选择功能进入case语句时都会首先对图是否已经创建进行判断，若是则会显示“图中无结点”，下述对各个函数算法设计过程的描述过程除创建图外均建立在前提条件满足，即已构建的图上操作。对于无向网和有向网的操作同样需要判断网是否构建。

1）status CreateGraph(ALGraph& G, VertexType V[], KeyType VR[][2])

首先将图中所有结点的关键字存入VertexType类型的数组V，包含结点关键字和内容，将图中所有边连接的两个顶点位序存储入KeyType类型的关键字VR，然后将空的图G的引用，两个数组作为参数传入函数CreateGraph，首先借助辅助函数检查构建图的过程中是否会出现关键字的重复；

bool CheckKey(VertexType V[], KeyType VR[][2])

在函数CheckKey中，定义一个数组sum存储关键字的出现次数，初始时每一个下标对应大小为0，对V数组进行遍历，每一个出现关键字对应出现次数增加1，当出现某一个关键字出现次数大于1时说明出现重复，返回false;

循环结束后，再进行一次遍历对所有次数为1的元素，结点总数增加1，当总数大于20时说明超限，返回false;

最后对VR函数进行遍历，当VR中每一对结点组中有在图中不存在的关键字时，说明边信息输入错误，返回false;

检查完毕后说明信息输入无误，通过do-while循环对图G的vertices数组进行关键字赋值，初始时指向邻接结点的指针域firstarc置为NULL；循环完毕后对G的结点数目进行填充；同样借助do-while循环对邻接结点域进行填补，对VR进行循环，使用两个整型变量one,two分别代表每一个边连接的点的位序，此举通过对vertices进行遍历同时比较关键字来找到对应的位序，然后定义ArcNode型的指针并申请空间，然后将他们的关键字分别赋为two，one，然后将结点通过首插法插入链表，先将p，q的nextarc指向结点的firstarc域，然后修改对应的firstarc指针域指向p,q从而成为新的首结点，增边操作结束后给弧的数目进行赋值；

2）status DestroyGraph(ALGraph& G)

销毁图的操作过程，对vertices数组的每一个Vnode型变量，定义一个ArcNode型的指针p指向后接邻接结点，当它存在后继时定义一个指针类型tem指向nextarc域，使用free释放p的空间后借助tem后移；空间释放完毕后将图的结点数目和边的数目都置为0；

3）int LocateVex(ALGraph G, KeyType u)

查找结点，直接遍历vertices数组，比较传入参数u和结点的关键字key，当相等时返回对应位序下标，遍历完毕后若没有返回下标则返回-1；

4）status PutVex(ALGraph& G, KeyType u, VertexType value)

结点赋值，首先借助辅助函数判断异常情况；

status check(ALGraph& G, KeyType u, VertexType value)

对vertices数组进行循环遍历，当有结点的关键字等于u时说明查找到了待赋值的结点，用于做判断的整形变量tag赋值为1，遍历时记录关键字的出现次数，当结点关键字等于u时不是u对应个数加1而是被替换后的value里的关键字出现次数加1；当有关键字的出现次数大于1时说明赋值后出现关键字重复，返回FALSE;当tag仍为0时说明未找到待赋值结点返回INFEASIBLE;检测没有异常情况则返回TRUE;

PutVex根据check函数的返回值进行判断，为INFEASIBLE或ERROR时也返回对应值；检查返回TRUE时对vertices数组进行循环遍历，当找到关键字为u的结点时将这个VNode类型的data域修改为新的VertexType类型value；修改完毕后返回OK；

5）int FirstAdjVex(ALGraph G, KeyType u)

找到第一个邻接点，对G的vertices数组进行遍历，当找到关键字大小为u的顶点时，对这个VNode类型的firstarc指针域进行判断，若它指向空则返回-1否则返回它指向的结点的数据域adjvex，即第一个邻接点的位序；

6）int NextAdjVex(ALGraph G, KeyType v, KeyType w)

对G的vertices数组进行遍历，对每一个VNode类型的数据域关键字与u比较，若相等则定义ArcNode型指针p等于这个结点的firstar指针域，然后不断判断p指向的结点的表示邻接点位序的adjvex与w是否相等，不相等则后移，若找不到与w相等的位序，则返回-1，若找到与w相等的位序所在的结点，判断它的nextarc域是否指向空，不为空则返回它的nextarc域的adjvex大小；

7）status InsertVex(ALGraph& G, VertexType v)

插入结点，首先判断新结点的插入是否会造成异常；

bool CheckDouble(ALGraph& G, VertexType v)

判断G的结点数是否超过上限20，若是则无法进入新结点返回false;对G的vertices数组进行遍历，当有一个VNode的关键字与v中的关键字相等时，说明插入后有关键字重复，返回false;遍历完毕后没有返回false则返回true;

当检测返回false时InsertVex函数返回ERROR；否则给vertices下标为结点数目大小的VNode 数据域赋值为v，指针域firstarc置为空；插入完毕结点总数增加1，返回OK；

8）status DeleteVex(ALGraph& G, KeyType v)

当图中仅有一个结点并且结点关键字正好为v时返回ERROR；定义一个整形变量x记录下标，对vertices数组进行遍历，当有一个VNode 的数据域的关键字大小为v时则说明找到待删除的结点，用x记录当前下标，并且对当前下标之后的所有VNode ,进行位置前移（因为当前位置i的结点被删除了，后继结点的位序都应该减少1），且G的结点数目减少1，若循环结束对x的大小进行判断仍,为-1则说明图中没有这个结点，直接返回ERROR；否则说明可以删除，对新的vertices数组进行遍历，对每一个结点后的记录邻接结点位序的链表都要进行遍历，其中出现删除的结点的位序，则需要删除它所在的结点，当firstarc域指向的结点的位序就满足条件则直接释放空间，否则定义ArcNode型指针指向firstarc的结点，对nextarc域的存在性与其位序大小进行判断，当nextarc域不指向空且位序大小恰为删除结点，则释放它的空间，p的nextarc域重新指向后继结点的下一个结点（可以为空）；在遍历每一层链表的同时，对结点中的adjvex大小进行判断，若大于删除结点位序，则都应该减1；

9）status InsertArc(ALGraph& G, KeyType v, KeyType w)

插入弧，首先根据给定的关键字，在vertices数组中找到关键字所在结点对应的下标，定义两个整型变量x1,x2表示两个结点的下标，初始化时为-1，遍历数组，找到data域关键字相等的，则将x1或x2赋为对应下标的值，遍历结束后，若x1或x2有为-1的则说明所要插入边连接的点的关键字有不存在图中的，返回ERROR；否则定义一个ArcNode型的指针变量指向x1位序的VNode 的firstarc指针域，对后继邻接结点的链表进行遍历，当发现邻接结点中有位序为x2的结点时，说明这个边在图中已经存在，返回INFEASIBLE;

能够插入时，定义两个ArcNode型指针，申请空间后分别将位序变量赋值为x1,x2然后通过首插法将结点插入在当前结点VNode 的firstarc所指向的位置，插入完毕后图的弧的总数增加1，返回OK；

10）status DeleteArc(ALGraph& G, KeyType v, KeyType w)

删除弧，首先根据两个关键字找到对应的位序下标，定义两个整型变量x1,x2表示两个结点的下标，初始化时为-1，遍历数组，找到data域关键字相等的，则将xw1或x2赋为对应下标的值，遍历结束后，若x1或x2有为-1的则说明所要插入边连接的点的关键字有不存在图中的，返回ERROR；否则定义一个ArcNode型的指针变量指向x1位序的VNode 的firstarc指针域，对后继邻接结点的链表进行遍历，当发现邻接结点中有位序为x2的结点时，说明这个边在图中存在，否则返回INFEASIBLE；找到弧，则定义两个ArcNode型的指针变量，指向x1，x2号结点的指针域，进行遍历，当在x1号结点的邻接点集中找到结点位序为x2的结点时将其空间释放，在x2号结点的邻接点集中找到结点位序为x1的结点时将其空间释放；操作完毕后图的弧总数减少1，返回OK；

11）status DFSTraverse(ALGraph& G, void (\*visit)(VertexType))

定义一个整形数组vis记录结点的访问情况，0代表下标为i的结点没有被访问，1代表下标为i的结点已经被访问过；然后将参数ALGraph类型的图G，初始访问下标0，访问数组vis，访问结点函数 visit传入辅助函数DFS进行深度优先搜索遍历；

void visit(VertexType v)

访问VertexType类型变量v，打印它的关键字和内容others;

void DFS(ALGraph& G, int i, int vis[], void (\*visit)(VertexType))

i是当前访问的结点，对它进行判断，若vis[i]为0，则说明i号结点尚未访问，则用visit函数输出结点的内容，然后将vis数组vis[i]置为1，表示已访问，然后定义一个ArcNode型的指针变量p指向当前结点的firstarc指针域，对邻接结点进行遍历，每遍历到一个结点，深度由于深度优先搜索，对当前搜索的结点进行DFS调用，此时传入的位序变量为结点中的adjvex，深层遍历退出后再进入p后移的语句，完成深度优先搜索；

而当当前访问的i号结点对应vis中为1时说明结点已经被访问过，直接返回；打印完所有结点信息后DFSTraverse函数返回OK；

12）status BFSTraverse(ALGraph& G, void (\*visit)(VertexType))

定义一个整形数组vis记录结点的访问情况，0代表下标为i的结点没有被访问，1代表下标为i的结点已经被访问过；定义一个整形数组来模拟队列，定义两个整型变量left和right分别表示队尾和队首，新的结点下标在右边增加，right增加，访问过的结点在左端，访问后left增加；首先位序为0的结点入队，right自增；然后进入while循环，当left小于right说明有结点没有访问完毕，进行循环，当left所指向的结点位序在vis数组中对应值为0时说明尚未访问该结点，使用visit函数进行访问，访问完毕vis数组对应位置标为1，然后left自增，定义一个ArcNode\*的指针指向当前结点的后继的表示邻接点的结点，当其指向不为空时，判断它的表示位序的变量adjvex对应的vis的值，若为0则说明尚未访问，将其值赋给a[right]，即入队操作，然后right自增，p指向下一个指针域；而当当前结点对应的vis值为1时说明已经访问过该结点，则直接left自增；这样就能以位序为0的结点开始逐层地去访问结点，即宽度优先遍历，输出所有序列后返回OK；

13）status SaveGraph(ALGraph G, char FileName[])

定义一个文件指针fp，写入图的信息，文件名就是选择图序号时输入的图的名称；首先读入图G的结点数目，然后用for循环写入G的vertices数组每一项VNode 的数据域的关键字和内容others，然后写入当前结点的所有邻接结点的位序，定义结点指针依次访问并读入即可；在一行的结束时读入一个-1作为一个邻接结点链表的结束，方便后续读取文件创建图；写入完毕时返回OK；

14）status LoadGraph(ALGraph& G, char FileName[])

定义一个文件指针，性质为”r+”，定义一个整形常量len，首先读取的结点数目用len存储，G的结点数目vexnum=len，然后通过for循环，上限就是len，每次先读取结点的关键字和内容others，然后读取后继的邻接点的位序，每读取一个就创建一个指针域申请空间后存储结点位序，当读取到-1时说明这个结点的邻接点的输入到此为止，最后一个指针域指向空，值得注意的是不能先动态申请空间，然后判断读取的位序是否为-1，若是再将新建的空间置为空，这样在打印邻接表信息时会出错，因为动态申请空间后再赋为NULL不是直接变成空（0）而是会变成某个不可访问的值，因此先判断是否读取到-1，若不是再为结点的nextarc指针域指向赋予空间去存储位序值；

15）void getShortestPath(ALGraph G, int u, int ShortestPath[])

对于一个给定的结点位序，求出所有其他结点到这个指定结点的最短路径，定义一个整形数组vis存储访问情况，为0时表示未访问，1表示访问了，初始时vis全部置为0，定义一个存储最短路径长度的数组ShortestPath，初始时所有值置为上限值100000，表示距离目标点无限远；借助辅助函数getPath来深度优先搜索并更新ShortestPath数组；

void getPath(ALGraph G, int u, int vis[], int ShortestPath[], int dis)

函数的参数为ALGraph型的变量G，当前访问结点位序，访问数组vis，存储最短路径的数组 ShortestPath,当前与目标节点距离dis；初始时访问结点为位序0，距离dis也为0，对dis进行判断，若小于位序i对应的最短路径的值，则更新ShortestPath[i]的大小，然后对vis[i]进行判断，若不为0说明访问过直接return；否则访问结点，vis数组置为1，定义ArcNode\*型的指针变量p指向当前结点的firstarc域，当p指向不为空时，调用自身进行深度搜索，因为每一个结点的邻接点到自身的距离肯定是最短的，此时修改参数，当前访问的结点位序应该为p中的adjvex大小，而距离大小应该传入dis+1，搜索完一个结点后p再后移；当所有结点全部搜索完毕后则ShortestPath更新完毕；getShortestPath函数返回OK；

16）bool BipartiteJudge(ALGraph G)

判断是否为二分图，可以通过染色法，即判断这个图能不能将每一条边的端点都染成不同的两个颜色，且共只有两种颜色,定义一个整形数组color记录结点所染的颜色，且只有0和1两种，初始化时所有元素都置为-1，表示未上色状态，递归调用辅助函数来判断是否为二分图；

bool isBipartite(ALGraph G, int i, int color[], int c)

首先定义一个整形常量next表示与当前结点i邻接的点应该染的色，它的值取决于c，若c为0，即位序为i的结点染为0，则next应该为1，c为1则next为0；然后对color[i]进行判断，若它不为-1且不等于c说明这个结点已被染色且不为应该被染上的色，说明出现相邻的点染色相同的情况，返回false；

否则，当color[i]为-1时说明未染色，color[i]赋值为c，对位序为i的结点的所有后继的邻接结点位序进行调用isBipartite，传入的位序i改为访问的adjvex的值，c更换为next即所有邻接点应该染的色为next，当有邻接点的调用返回false时说明不为二分图，返回false；深度搜索会遍历并为所有的点染色完毕，在遍历完毕后若不返回false则说明为二分图，返回true；最后一种情况，color[i]为c，则说明这个点染色正确，直接返回true；isBipartite的返回值作为BipartiteJudge的返回值进行返回；

17）int getComponentNum(ALGraph G)

得到连通分量数目，定义一个整型变量count作为计数，定义一个整形数组vis存储访问信息，初始化时所有元素初始化为0，表示结点都未被访问，对于一个连通分量，以一个结点为起点进行深度优先搜索必能访问到这个连通分量中的所有结点，对vertices数组进行遍历，遍历到一个结点，判断这个位序的结点是否被访问过，若没有则联通分量数目增加1，然后调用Sign函数对这个结点所在的连通分量的结点进行标记；

void Sign(ALGraph G, int vis[], int i)

对于当前访问的结点对应的vis置为1，定义ArcNode\*型的指针p指向首结点，当p不为空时，若p中的adjvex指向的结点尚未进行访问，则调用Sign函数访问，传入参数为p中的adjvex，DFS完毕后，p后移；所有结点访问完毕后返回计数count；

18）status CreateDN(Net& net, int DNkey[], int se[][2], int weight[])

首先输入有向网的所有结点的关键字存储入Dnkey数组，然后输入所有边连接的顶点，先输入起点，后输入终点，存入二维数组欧se,以及每一条边的边权存储入weight数组；然后根据数组来创建有向带权图net，先对DNkey 数组循环遍历，记录每一个元素出现的次数，当出现有元素出现次数大于1时说明关键字重复，返回ERROR;对DNkey数组循环遍历，在读取到-1之前存入所有读取的结点关键字，数组以-1作为结尾，有向网中的结点数量为遍历到-1时的下标i；对se数组进行遍历，当发现有边连接的结点位序不在0-net.NodeSum-1间时说明有结点不存在这个图中，返回INFEASIBLE;否则对se数组遍历，每一个结点对都存入有向网net的顶点类型对象e[i]的起点位序start和终点位序end，第i条边的边权e[i]的length为weight[i]，读取到-1作为三个数组的结尾，网中的边数为遍历到-1时的下标i；成功构建后返回OK；

19）void Dijkstra(Net net, int vertex)

首先根据构建的有向图构建邻接矩阵二维数组adjacent，初始化时adjacent中的所有元素置为inf表示所有结点间的距离为无限远，遍历有向网中的所有边集，对于e[i]，读取它的start和end作为二维数组的位置下标，将对应的邻接矩阵中的大小置为这个有向边的长度；定义整型变量count作为访问过的结点的数目，定义一个bool型的数组find表示已经找到最短路径的结点，定义整形数组pre记录每一个当前结点的前驱结点的最短路径，定义整形数组distance记录结点的最短路径大小；初始化时pre的所有元素均为所选择的源点的位序vertex，distance的初始值为邻接矩阵中存放的源点到各个结点的距离；初始时所有结点都没有找到最短路径，故find数组均值为false；源点的find值置为true，distance为0，定义一个用来迭代顶点的整型变量v，初始时赋值为源点位序，定义一个整型变量d表示距离；当已找到最短路径结点数count小于总结点数时执行while循环，d置为inf极限值，对所有结点进行遍历，找到离最初结点最近的一个未被访问的结点，将v置为这个节点的位序，d置为源点和这个结点的距离，修改find[v]为true；再将上面的找到最短路径的点作为源点，连接到其他未被访问过的点上，仍是遍历结点数组，当结点未被访问时，比较新的源点的最短路径值distance[v]与点v到遍历到的点的距离之和（即一条新路径）与该遍历到的点与最初所选择的源点间的直接距离的大小，distance更新为两者中的较小者；遍历结束后新增了一个新的结点被找到了最短路径，则count自增1；访问完毕后，i从0遍历到net.NodeSum，若distance[i]为inf的值说明没有更新距离，即这个点与源点间无有向通路，打印对应信息，否则打印源点和i号结点的最短路径值；

20）status CreateUDN(UDNet& UDnet, int DNkey[], int se[][2], int weight[])

根据提示首先输入UDnet的所有结点的关键字存储在DNkey数组中，然后输入所有无向边所连接的点的位序，以及边的权重，传入这几个数组作为参数，在函数中，先对DNkey 数组循环遍历，记录每一个元素出现的次数，当出现有元素出现次数大于1时说明关键字重复，返回ERROR;否则根据DNkey 数组把结点的关键字写入n[i]的关键字中，当读取到-1时作为数组的结尾，然后退出循环；定义无向网中的结点数目NodeSum就是循环结束时当前的位序下标；对se数组进行遍历，当发现有边连接的结点位序不在0-net.NodeSum-1间时说明有结点不存在这个图中，返回INFEASIBLE;否则根据se数组构建边，当读取到-1前给网中的边类型数组赋值，边e[i]的node1和node2分别为se[i]的第一个和第二个元素；同时从weight数组中写入边权length；最后以-1 -1 -1结束循环；写入图的边总数EdgeSum为遍历结束时当前位序下标；创建成功后返回OK；

21）int prim(UDNet UDnet)

定义一个整形二维数组m作为邻接矩阵，定义一个整形数组vis记录结点的访问信息，定义整形数组low记录每一个结点到所选择的第一个点的最短路径；初始化时m中的所有元素置为10000表示初始时所有结点间距离为无限大，然后根绝UDnet 中的边属性来构建邻接矩阵，每一个边记录的node1，node2分别作为一维和二维坐标在m中对应位置将m[node1][node2]和m[node2][node1]大小置为这个边的边权，表示结点间距离；我们选择0号结点为起始点，故vis[0]大小置为1；定义一个整形变量sum记录最小生成树的总边权，表示位置的变量pos，初始时为位序0，首先遍历所有结点，将low数组的值更新为每一个结点到初始点的距离作为最短路径，即先把第一个点放入树中，然后再找剩下所有点到这个点的距离，不直接相连的点距离暂时仍为无限大，接下来执行UDnet.NodeSum-1次循环，将剩下的所有结点全部加入树中，在每一次循环中，我们的任务是找到离现在在树中的任意结点路径的最短的结点，先定义一个整形变量minn为10000，然后遍历结点，当结点位序对应的vis为0即未访问时且对应low的值比minn小时更新minn和pos，循环结束后总权值sum加上最终的minn，并且pos对应的vis的值变为1，赋值完毕后，再进行一次循环去更新所有结点的low值，此时比较的是刚刚新增入树中的位序为pos的结点到其他结点的距离与其他结点本来的low值，而low值更新为两者中的较小者，这个操作的意义是每新加入一个结点，就把其他结点到这个新的树的最短路径更新，然后下个循环再取low值最小的新加入树，重复完结点树-1次循环后，所有结点均加入树中，这棵树就是最小生成树，此时的总权值sum作为返回值返回。

22）void TopoSort(Net net)

定义整形数组in记录每一个结点的入度，一个整形二维数组记录邻接矩阵，一个整形数组记录可能存在的拓扑排序结果序列，初始化时所有结点入度为0，邻接矩阵的所有元素为0，对边数组遍历，更新邻接矩阵，有向边的起点为i,j，则adjacent[i][j]=1;终点的入度也加1，然后进行net.NodeSum循环，一次循环加入一个结点到结果里，每一次循环，首先对in数组遍历，找到第一个入度为0的结点加入结果的queue数组，然后这个结点的入度-1（变为负），这样下次就不会搜索到已加入结果的点，然后对所有以这个结点为起点的有向边的终点的入度减1，相当于删边操作，在循环中如果出现找不到入度为0的结点的情况则说明没有拓扑排序，否则输出结果queue。

**4.4测试用例**

在命令行中执行生成的exe文件，对每一个函数进行测试。

首先在图未创建下选择函数执行；

空图测试用例表

|  |  |  |
| --- | --- | --- |
| 测试函数 | 预期结果 | 操作结果 |
| DestroyGraph | 显示“图中无结点” |  |
| LocateVex | 显示“图中无结点” |  |
| PutVex | 显示“图中无结点” |  |
| FirstAdjVex | 显示“图中无结点” |  |
| NextAdjVex | 显示“图中无结点” |  |
| InsertNode | 显示“图中无结点” |  |
| DeleteNode | 显示“图中无结点” |  |
| InsertArc | 显示“图中无结点” |  |
| DeleteArc | 显示“图中无结点” |  |
| DFSTraverse | 显示“图中无结点” |  |
| BFSTraverse | 显示“图中无结点” |  |
| SaveGraph | 显示“图中无结点” |  |
| LoadGraph | 显示“文件不存在” |  |
| PrintGraph | 显示“图中无结点” |  |
| ShortestPath | 显示“图中无结点” |  |
| BipartiteJudge | 显示“图中无结点” |  |
| getComponentNum | 显示“图中无结点” |  |
| Dijkstra | 显示“图不存在结点或无边连接” |  |
| prim | 显示“图不存在结点或无边连接” |  |

以下函数的操作与结果验证都在构建图的条件下完成

1）CreateGraph测试

测试结果用15号功能打印邻接表来验证正确性

|  |  |
| --- | --- |
| 用例 | 建图序列 |
| 1 | 5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1 |
| 2 | 5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 10 8 -1 -1 |
| 3 | 5 线性表 7 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 10 8 -1 -1 |
| 4 | 1 G1 2 G2 3 G3 4 G4 5 G5 6 G6 7 G7 8 G8 9 G9 10 G10 11 G11 12 G12 13 G13 14 G14 15 G15 16 G16 17 G17 18 G18 19 G19 20 G20 21 G21 -1 nil -1 -1 |
| 5 | 3 Lisa 4 Rose 1 Jisoo 2 Jennie -1 nil 1 2 1 3 1 4 2 3 2 4 3 4 -1 -1 |

结果验证：

|  |  |  |
| --- | --- | --- |
| 用例 | 理论结果 | 操作结果 |
| 1 | 5 线性表 2 3 8 集合 2  7 二叉树 1 3 0  6 无向图 2 0 |  |
| 2 | 创建失败（关键字重复） |  |
| 3 | 创建失败（不存在结点） |  |
| 4 | 创建失败（超限） |  |
| 5 | 3 Lisa 1 3 2  4 Rose 0 3 2  1 Jisoo 1 0 3  2 Jennie 1 0 2 |  |

2）DestroyGraph测试

测试后使用其他操作理论上会输出图中无结点

首先建图，建图语句为：5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1 

|  |  |  |
| --- | --- | --- |
| 用例 | 使用函数 | 操作结果 |
| 1 | DestroyGraph |  |
| 2 | PutVex |  |
| 3 | InsertArc |  |
| 4 | BFSTraverse |  |

3）LocateVex测试

|  |  |
| --- | --- |
| 用例 | 建图语句 |
| 1 | 5 线性表8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1 |
| 2 | 5 线性表8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1 |
| 3 | 3 Lisa 4 Rose 1 Jisoo 2 Jennie -1 nil 1 4 2 3 2 4 3 4 -1 -1 |
| 4 | 3 Lisa 4 Rose 1 Jisoo 2 Jennie -1 nil 1 4 2 3 2 4 3 4 -1 -1 |

结果验证

|  |  |  |  |
| --- | --- | --- | --- |
| 用例 | 结点关键字 | 理论结果 | 操作结果 |
| 1 | 8 | 8 集合 |  |
| 2 | 10 | 查找失败 |  |
| 3 | 1 | 1 Jisoo |  |
| 4 | 5 | 查找失败 |  |

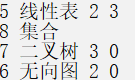
4）PutVex测试

|  |  |  |  |
| --- | --- | --- | --- |
| 用例 | 建图语句 | 结点关键字 | 新的key,others |
| 1 | 5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1 | 6 | 9 有向图 |
| 2 | 5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1 | 9 | 9 有向图 |
| 3 | 5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1 | 6 | 7 有向图 |
| 4 | 10 王宝强 15 管虎 20 冯小刚 25 变形金刚 -1 nil 10 20 15 25 -1 -1 | 20 | 30 霸天虎 |

结果验证

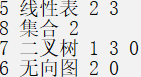
|  |  |  |
| --- | --- | --- |
| 用例 | 理论结果 | 操作结果 |
| 1 | 5 线性表 2 3  8 集合 2  7 二叉树 1 3 0  9 有向图 2 0 |  |
| 2 | 查找关键字不存在 |  |
| 3 | 赋值后出现关键字重复 |  |
| 4 | 10 王宝强 2  15 管虎 3  30 霸天虎 0  25 变形金刚 1 |  |

5）FirstAdjVex验证

创建图的语句：5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 -1 -1 它的邻接表为：

|  |  |  |  |
| --- | --- | --- | --- |
| 用例 | 关键字 | 理论结果 | 操作结果 |
| 1 | 7 | 第一邻接点为6 无向图 |  |
| 2 | 8 | 查找失败！ |  |
| 3 | 10 | 查找失败！ |  |

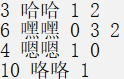
6）NextAdjvex验证

建图语句：5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1，它的邻接表为：

|  |  |  |  |
| --- | --- | --- | --- |
| 用例 | v,w | 理论结果 | 操作结果 |
| 1 | 7 8 | 6 无向图 |  |
| 2 | 8 7 | 无下一邻接顶点 |  |
| 3 | 10 7 | 无下一邻接顶点 |  |
| 4 | 7 7 | 无下一邻接顶点 |  |
| 5 | 6 7 | 5 线性表 |  |

7）InsertNode验证

建图语句：3 哈哈 6 嘿嘿 4 嗯嗯 10 咯咯 -1 nil 3 4 6 4 6 10 3 6 -1 -1

邻接表为：

|  |  |  |  |
| --- | --- | --- | --- |
| 用例 | 插入结点 | 理论结果 | 操作结果 |
| 1 | 8 呼呼 | 3 哈哈 1 2  6 嘿嘿 0 3 2  4 嗯嗯 1 0  10 咯咯 1 8 呼呼 |  |
| 2 | 4 擦擦 | 插入结点失败! |  |

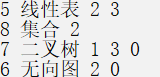
建图语句：1 G1 2 G2 3 G3 4 G4 5 G5 6 G6 7 G7 8 G8 9 G9 10 G10 11 G11 12 G12 13 G13 14 G14 15 G15 16 G16 17 G17 18 G18 19 G19 20 G20 -1 nil -1 -1 20个结点不能再插入，应该返回插入失败！

|  |  |  |
| --- | --- | --- |
| 插入结点 | 理论结果 | 操作结果 |
| 21 G21 | 插入结点失败! |  |

建图语句：1 G1 2 G2 3 G3 4 G4 5 G5 6 G6 7 G7 8 G8 9 G9 10 G10 11 G11 12 G12 13 G13 14 G14 15 G15 16 G16 17 G17 18 G18 19 G19 -1 nil -1 -1

|  |  |  |
| --- | --- | --- |
| 插入结点 | 理论结果 | 操作结果 |
| 20 G20 | 插入成功 |  |

8）DeleteNode验证

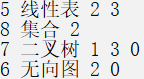
建图语句：5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1，邻接表为：

|  |  |  |  |
| --- | --- | --- | --- |
| 用例 | 删除结点 | 理论结果 | 操作结果 |
| 1 | 7 | 5 线性表 2  8 集合  6 无向图 0 |  |
| 2 | 10 | 删除失败！ |  |
| 3 | 6 | 5 线性表 2  8 集合 2  7 二叉树 1 0 |  |

建图语句：5 线性表 -1 nil -1 -1 仅有一个结点，没有边

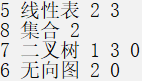
|  |  |  |
| --- | --- | --- |
| 删除结点 | 理论结果 | 操作结果 |
| 5 | 删除失败！ |  |

9）InsertArc检测

建图语句：5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1，邻接表为：

|  |  |  |  |
| --- | --- | --- | --- |
| 用例 | 端点关键字 | 理论结果 | 操作结果 |
| 1 | 6 8 | 5 线性表 2 3  8 集合 3 2  7 二叉树 1 3 0  6 无向图 1 2 0 |  |
| 2 | 6 5 | 该弧已存在 |  |
| 3 | 16 15 | 结点有的不存在此图中 |  |
| 4 | 5 9 | 结点有的不存在此图中 |  |
| 5 | 8 5 | 5 线性表 1 2 3  8 集合 0 2  7 二叉树 1 3 0  6 无向图 2 0 |  |

10）DeleteArc检测

建图语句：5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1 邻接表为：

|  |  |  |  |
| --- | --- | --- | --- |
| 用例 | 端点 | 理论结果 | 操作结果 |
| 1 | 5 6 | 5 线性表 2  8 集合 2  7 二叉树 1 3 0  6 无向图 2 |  |
| 2 | 5 7（接着上例） | 5 线性表  8 集合 2  7 二叉树 1 3  6 无向图 2 |  |
| 3 | 5 8 | 图中不存在这条弧！ |  |
| 4 | 5 16 | 结点有的不存在此图中！ |  |

建图语句：1 G1 2 G2 3 G3 4 G4 5 G5 -1 nil 1 2 2 3 3 4 -1 -1

邻接表为：

|  |  |  |  |
| --- | --- | --- | --- |
| 用例 | 端点 | 理论结果 | 操作结果 |
| 1 | 1 2 | 1 G1  2 G2 2  3 G3 3 1  4 G4 2  5 G5 |  |
| 2 | 1 5 | 不存在这条弧！ |  |

11）DFSTraverse检测

|  |  |
| --- | --- |
| 用例 | 建图语句 |
| 1 | 5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1 |
| 2 | 2 a 4 b 7 c 10 d 8 e 3 e -1 nil 2 4 2 7 2 10 4 8 7 3 3 10 -1 -1 |
| 3 | 3 a 6 b 9 c 2 m 4 n -1 nil 3 6 3 9 2 4 -1 -1 |

结果验证：

|  |  |  |
| --- | --- | --- |
| 用例 | 理论结果 | 操作结果 |
| 1 | 5 线性表 7 二叉树 8 集合 6 无向图 |  |
| 2 | 2 a 10 d 3 e 7 c 4 b 8 e |  |
| 3 | 图不为连通图 |  |

12）BFSTraverse检测

|  |  |
| --- | --- |
| 用例 | 建图语句 |
| 1 | 5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1 |
| 2 | 5 a 10 b 12 c 7 d 8 e 6 f 4 g -1 nil 5 10 5 12 5 7 10 8 12 6 7 4 -1 -1 |
| 3 | 5 a 4 b 3 c 7 d -1 nil 5 4 5 3 3 4 -1 -1 |

结果验证

|  |  |  |
| --- | --- | --- |
| 用例 | 理论结果 | 操作结果 |
| 1 | 5 线性表 7 二叉树 6 无向图 8 集合 |  |
| 2 | 5 a 7 d 12 c 10 b 4 g 6 f 8 e |  |
| 3 | 图不为连通图! |  |

13）SaveGraph检测

|  |  |  |
| --- | --- | --- |
| 用例 | 图名称 | 建图语句 |
| 1 | aa | 5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1 |
| 2 | Haha | 3 Lisa 4 Rose 1 Jisoo 2 Jennie -1 nil 1 4 2 3 2 4 3 4 -1 -1 |

结果验证

|  |  |
| --- | --- |
| 用例 | 文件 |
| 1 |  |
| 2 |  |
| 3 | 不存在的文件“hhhhh” |

14）LoadGraph检测

重新在空图中读取文件

|  |  |  |  |
| --- | --- | --- | --- |
| 用例 | 文件名 | 理论结果 | 操作结果 |
| 1 | aa | 5 线性表 2 3  8 集合 2  7 二叉树 1 3 0  6 无向图 2 0 |  |
| 2 | Haha | 3 Lisa 1 3  4 Rose 0 3 2  1 Jisoo 1  2 Jennie 1 0 |  |
| 3 | Hhhhh | 文件不存在！ |  |

15）ShortestPath检测

建图语句：5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1

|  |  |  |  |
| --- | --- | --- | --- |
| 用例 | 源点位序 | 理论结果（结点：路长） | 操作结果 |
| 1 | 0 | 1:2 2:1 3:1 |  |
| 2 | 3 | 0:1 1:2 2:1 |  |

建图语句：3 a 4 b 7 c 8 d 5 e 10 f -1 nil 3 4 3 8 3 7 3 10 5 8 5 10 -1 -1

|  |  |  |  |
| --- | --- | --- | --- |
| 用例 | 源点序 | 理论结果（结点：路长） | 操作结果 |
| 1 | 10 | 不存在该结点 |  |
| 2 | 0 | 1:1 2:1 3:1 4:2 5:1 |  |
| 3 | 2 | 0:1 1:2 3:2 4:3 5:2 |  |

16）BipartiteJudge检测

|  |  |
| --- | --- |
| 用例 | 建图语句 |
| 1 | 5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1 |
| 2 | 3 a 5 b 4 c 6 d 8 m 10 n -1 nil 3 4 3 5 4 6 5 6 4 8 8 10 6 10 -1 -1 |
| 3 | 3 A 4 C 5 自 6 动 8 机 -1 nil 3 8 4 8 5 8 6 8 3 4 3 5 3 6 -1 -1 |

结果验证

|  |  |  |
| --- | --- | --- |
| 用例 | 理论结果 | 操作结果 |
| 1 | 不是二分图！ |  |
| 2 | 是二分图! |  |
| 3 | 不是二分图! |  |

17)getComponentNum检测

|  |  |
| --- | --- |
| 用例 | 建图语句 |
| 1 | 5 线性表 8 集合 7 二叉树 6 无向图 -1 nil 5 6 5 7 6 7 7 8 -1 -1 |
| 2 | 5 鸡 10 鸭 15 鱼 20 猪 -1 nil -1 -1 |
| 3 | 4 Jisoo 8 Jennie 7 Tayon 9 Rose 11 Lisa -1 nil 4 8 7 9 9 11 7 11 -1 -1 |

结果验证

|  |  |  |
| --- | --- | --- |
| 用例 | 理论结果 | 操作结果 |
| 1 | 该图有1个连通分量 |  |
| 2 | 该图有4个连通分量 |  |
| 3 | 该图有2个连通分量 |  |

18）CreateDN检测

|  |  |
| --- | --- |
| 用例 | 建网语句 |
| 1 | 2 4 6 8 10 -1 0 1 10 0 3 30 0 4 90 1 2 50 3 1 10 3 2 20 2 4 10 3 4 60 -1 -1 -1 |
| 2 | 2 4 6 8 8 -1 0 1 10 0 3 30 0 4 90 1 2 50 3 1 10 3 2 20 2 4 10 3 4 60 -1 -1 -1 |
| 3 | 2 4 6 8 10 -1 0 1 10 0 3 30 0 4 90 1 2 50 3 9 10 3 2 20 2 4 10 3 4 60 -1 -1 -1 |

结果验证

|  |  |
| --- | --- |
| 用例 | 操作结果 |
| 1 |  |
| 2 |  |
| 3 |  |

19）Dijkstra检测

建图语句：2 4 6 8 10 -1 0 1 10 0 3 30 0 4 90 1 2 50 3 1 10 3 2 20 2 4 10 3 4 60 -1 -1 -1

结果验证

|  |  |  |
| --- | --- | --- |
| 用例 | 源点位序 | 操作结果 |
| 1 | 0 |  |
| 2 | 1 |  |
| 3 | 3 |  |

20）CreateUDN检测

|  |  |
| --- | --- |
| 用例 | 建网语句 |
| 1 | 1 3 5 7 10 -1 0 1 2 0 2 1 0 3 3 2 3 5 1 4 2 2 4 3 -1 -1 -1 |
| 2 | 1 3 5 7 5 -1 0 1 2 0 2 1 0 3 3 2 3 5 1 4 2 2 4 3 -1 -1 -1 |
| 3 | 1 3 5 7 10 -1 0 1 2 0 2 1 0 3 3 2 3 5 1 4 2 6 4 3 -1 -1 -1 |

结果验证

|  |  |
| --- | --- |
| 用例 | 操作结果 |
| 1 |  |
| 2 |  |
| 3 |  |

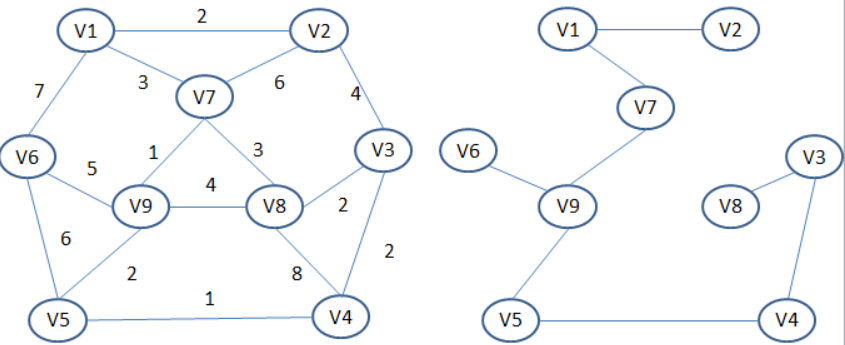
21）prim检测

|  |  |
| --- | --- |
| 用例 | 建图语句 |
| 1 | 1 3 5 7 10 -1 0 1 2 0 2 1 0 3 3 2 3 5 1 4 2 2 4 3 -1 -1 -1 |
| 2 | 2 4 8 9 11 -1 0 1 10 0 2 12 1 3 50 1 2 10 0 3 20 3 4 15 -1 -1 -1 |
| 3 | 1 2 3 4 5 6 7 8 9 -1 0 1 2 0 6 3 1 6 6 0 5 7 5 8 5 6 8 1 6 7 3 8 7 4 1 2 4 2 7 2 5 4 6 4 8 2 4 3 1 7 3 8 2 3 2 -1 -1 -1 |

结果验证

|  |  |  |
| --- | --- | --- |
| 用例 | 理论结果 | 操作结果 |
| 1 | 1+2+2+3=8 |  |
| 2 | 10+20+10+15=55 |  |
| 3 | 2+3+1+5+2+1+2+2=18 |  |

用例3：



22）TopoSort检测

|  |  |
| --- | --- |
| 用例 | 建网序列 |
| 1 | 2 4 6 8 10 -1 0 1 10 0 2 20 1 3 30 2 4 40 3 4 50 4 1 60 -1 –1 -1 |
| 2 | 1 3 5 7 9 -1 0 1 5 0 2 10 0 3 15 1 3 20 2 4 25 3 4 30 -1 -1 -1 |

检测结果

|  |  |  |
| --- | --- | --- |
| 用例 | 理论结果 | 操作结果 |
| 1 | 无拓扑排序序列 |  |
| 2 | 0 1 2 3 4 |  |

**4.5 实验小结**

本次关于基于邻接表的图的实验花费了很多时间，实验题目只要求了在无向无权图上的操作，在实验过程中也有不少的问题，比如删除结点时考虑邻接表中后序的结点的位序问题，同时，在文件读取中，最初的想法是先为ArcNode类型p申请空间，然后判断是否读取到了表示一行结束的-1，再对p指向置为NULL，在提交时出现错误，每一行的最后都出现了一个多余的0，查阅资料后才知道动态申请的空间赋值为NULL，并不会置空，有的地方会输出0，有的时候会变成一个不可访问的值，算是弥补了一个知识空白；此外，在本次实验中结合了以前离散数学中学到的知识补充了二分图的判断，计算连通分量数量等功能，然后为了实现上课时出现的求最小生成树和图中的最短路径问题，添加了构建无向，有向网的功能，并分别实现了Prim算法和Dijkstra算法，算是实现了一些相对较难的操作；图与树是CSP，PAT中经常出现的模型，因此理解和实现一些经典算法是非常有必要的。