**东北大学计算机科学与工程学院**

**数据结构课程设计报告**

题目 无向图的关节点问题

课题组长 于梓元

课题组成员 张伟颖 孟霞

专业名称 计算机科学与技术

班级 计1504

指导教师 张晓红

2017 年 1月

**课程设计任务书**

|  |
| --- |
| **题目：**  无向图的关节点问题 |
| **问题描述：**  对无向连通图，若删除某个结点使其成为非连通图，则称该结点为关节点。假设某一地区公路交通网，求解关节点。 |
| **设计要求：**  设计基于哈夫曼编码的压缩软件。  （1）采用静态链表的二叉树等数据结构的类实现。  （2）创建哈夫曼树。  （3）哈夫曼编码和译码。  （4）源码、编码和压缩后的信息均以文件形式保存。  （5）软件时间和空间性能分析。  （6）基于哈夫曼编码的位图压缩软件设计（可选）。 |
| **指导教师签字：**  **年　　月　　日** |

**课题程序设计分工**

|  |  |  |  |
| --- | --- | --- | --- |
| 学号 | 姓名 | 程序设计函数原型、类 | 功能说明 |
| 20154561 | 于梓元 | class Error  enum StatusCode;  int main(void);  template<class ElemType>  class LinkQueue  template<class ElemType>  void DFSArticul(const AdjListUndirGraph<ElemType> &g,int v0,int Visited[],int low[],int couted[])  template<class ElemType>  void FindArticul(const AdjListUndirGraph<ElemType> &G)  template <class ElemType>  void Display(const AdjListUndirGraph<ElemType> &g, bool showVexElem); | //通用异常类  //表明状态的枚举  //主函数  //链队列类  //深度优先遍历关节点  //深度优先遍历关节点  // 显示邻接矩阵无向图 |
| 20154379 | 张伟颖 | template<class ElemType>  class LinkStack  template <class ElemType>  void BFSTraverse(const AdjListUndirGraph<ElemType> &g, void (\*visit)(const ElemType &))  template<class ElemType>  void CorrectArticul( AdjListUndirGraph<ElemType> &G)  static char GetChar(istream &inStream = cin);  static bool UserSaysYes();  template <class ElemType >  void Swap(ElemType &e1, ElemType &e2); | //链栈类  // 对图g进行广度优先遍历  //深度优先遍历关节点并修正关节点（使关节点变成非关节点）  // 从输入流inStream中跳过空格及制表符获取一字符  // 当用户肯定回答(yes)时, 返回true, 用户否定回答(no)时,返回false  // 交换e1, e2之值 |
| 20154508 | 孟霞 | template<class ElemType> class AdjListGraphVexNode  template<class ElemType>  class AdjListUndirGraph  template <class ElemType>  struct Node  template<class ElemType>  class LinkList  template<class ElemType>  void CorrectArticul( AdjListUndirGraph<ElemType> &G)  static void SetRandSeed();  static int GetRand(int n);  static int GetRand(); | //邻接表图定点结点类  //无向图邻接表类  //结点类  //链表类  //深度优先遍历关节点并修正关节点（使关节点变成非关节点）  // 设置当前时间为随机数种子  // 生成0 ~ n-1之间的随机数  // 生成随机数 |

# 课题报告分工

|  |  |  |
| --- | --- | --- |
| 章节 | 内容 | 完成人 |
| 1 课题概述 | 1.1 课题任务  1.2 课题原理  1.3 相关知识 | 张伟颖 |
| 2 需求分析 | 2.1 课题调研  2.2 用户需求分析 | 于梓元 |
| 3 方案设计 | 3.1 总体功能设计  3.2 数据结构设计  3.3 函数原型设计  3.4 主算法设计  3.5 用户界面设计 | 孟霞 |
| 4 方案实现 | 4.1 开发环境与工具  4.2 程序设计关键技术  4.3 个人设计实现（按组员分工）  4.3.1于梓元设计实现  4.3.2张伟颖设计实现  4.3.3孟霞设计实现 | 于梓元  张伟颖  孟霞 |
| 5 测试与调试 | 5.1 个人测试（按组员分工）  5.1.1于梓元测试  5.1.2张伟颖测试  5.1.3孟霞测试  5.2 组装与系统测试  5.3 系统运行 | 孟霞 |
| 6 课题总结 | 6.1 课题评价  6.2 团队协作  6.3 下一步工作  6.4 个人设计心得（按组员分工）  6.4.1于梓元小结  6.4.2张伟颖小结  6.4.3孟霞小结 | 于梓元  孟霞  张伟颖 |

目录

[课题报告分工 4](#_Toc471983732)

[1 课题概述 5](#_Toc471983733)

[1.1 课题任务 5](#_Toc471983734)

[1.2 课题原理 5](#_Toc471983735)

[1.3 相关知识 6](#_Toc471983736)

[2 需求分析 6](#_Toc471983737)

[2.1 课题调研 6](#_Toc471983738)

[2.2 用户需求分析 7](#_Toc471983739)

[3 方案设计 7](#_Toc471983740)

[3.1 总体功能设计 7](#_Toc471983741)

[3.2 数据结构设计 7](#_Toc471983742)

[3.3 函数原型设计 10](#_Toc471983743)

[3.4 主算法设计 12](#_Toc471983744)

[3.5 用户界面设计 17](#_Toc471983745)

[3.6 输入输出设计 18](#_Toc471983746)

[4 方案实现 23](#_Toc471983747)

[4.1 开发环境与工具 23](#_Toc471983748)

[4.2 程序设计关键技术 23](#_Toc471983749)

[4.3 个人设计实现（按组员分工） 23](#_Toc471983750)

[4.3.1 于梓元设计实现： 23](#_Toc471983751)

[4.3.2 张伟颖设计实现： 43](#_Toc471983752)

[4.3.3 孟霞设计实现： 49](#_Toc471983753)

[5 测试与调试 56](#_Toc471983754)

[5.1 个人测试（按组员分工） 56](#_Toc471983755)

[5.1.1 于梓元测试 56](#_Toc471983756)

[5.1.2张伟颖测试 57](#_Toc471983757)

[5.1.3孟霞测试 58](#_Toc471983758)

[5.2 组装与系统测试 60](#_Toc471983759)

[5.3 系统运行 62](#_Toc471983760)

[6 课题总结 65](#_Toc471983761)

[6.1 课题评价 65](#_Toc471983762)

[6.2 团队协作 66](#_Toc471983763)

[6.3 个人设计小结（按组员分工） 66](#_Toc471983764)

[6.3.1 于梓元小结 66](#_Toc471983765)

[6.3.2 张伟颖小结 66](#_Toc471983766)

[6.3.3 孟霞小结 66](#_Toc471983767)

# 1 课题概述

**无向图的关节点问题**

对无向连通图，若删除某个结点使其成为非连通图，则称该结点为关节点。假设某一地区公路交通网，求解关节点。

1.1 课题任务

设计求解无向连通图关节点的模拟程序。

（1）采用邻接表或邻接矩阵存储结构。

（2）可以随机、文件及人工输入数据。

（3）采用深度优先遍历求解关节点。

（4）实现关节点的查询和统计功能。

（5）实现将关节点改造为非关节的功能。

（6）其它完善性或扩展性功能。

1.2 课题原理

求无向图的关节点：

从任一点出发深度优先遍历得到优先生成树，对于树中任一顶点V而言，其孩子节点为邻接点。由深度优先生成树可得出两类关节点的特性：

（1）若生成树的根有两棵或两棵以上的子树，则此根顶点必为关节点。因为图中不存在连接不同子树顶点的边，若删除此节点，则树便成为森林。

（2）若生成树中某个非叶子节点V，其某棵子树与V的祖先节点无连接，则V为关节点。

对图G重新定义遍历时的访问函数visited，并引入一个新函数low，则由以此深度优先遍历便可求得连通图中存在的所有关节点。

定义low[v]取下述三个结点中的最小者：

(1)visited[v]；

(2) visited[k]，凡是有回退边（v,k）的任何结点k；

(3) low[w]，对v的任何儿子w。

若对于某个顶点v，存在孩子节点w且low[w]>=visited[v]，则该节点v必为关节点。

1.3 相关知识

（1）数据结构相关知识：

无向图的邻接表存储结构、链表相关知识、无向图的深度优先遍历、求无向图关节点算法等。

（2）C++程序设计相关知识：

程序编写了C++的类模板，并使用了C++中异常处理的功能，封装了异常类，并使用了动态存储结构。

# 2 需求分析

2.1 课题调研

关节点和连通图在实际中有较多的应用。显然，一个表示通信网络的图的连通度越高，其系统越可靠，无论是哪一站点出现故障或遭到外界破坏，都不影响系统正常工作；又如：一个航空网络若是重联通的，则当某条航线因天气等某种原因关闭时，旅客仍可以从别的航线绕道而行；再如，若将大规模集成电路的关键电路设计成重连通的话，则某些元件失效的情况下，整个片子的功能不受影响，反之，在战争中，若要摧毁敌方的运输线，仅需破坏其运输网中的关节点即可。

本课题对在校大学生们对图的学习、作业有很大帮助、可以帮助大学生了解关节点的具体内涵。本程序可以帮助同学们快速地求出关节点，方便了同学们对关节点的搜寻。

同时在校外的交通运输等方面本程序也有着极大的便利作用，可以简便地求出关键的枢纽，快速地修正，避免了因为一处崩溃导致大量瘫痪，节省了大量的人力物力，有极大的应用价值。

2.2 用户需求分析

根据本课题的要求进行分析、调查、研究，结果表明我们需要做出将图中关节点求出，并统计、反馈、修正（比如说在交通图中求出关键车站，并找出何处建立新的公交路线是不至于一个车站出问题后，整个交通网络全面瘫痪）。需要有找出关键点，并修正的功能。

另外，用户更喜欢简洁的外形，方便、易懂的使用流程。

# 3 方案设计

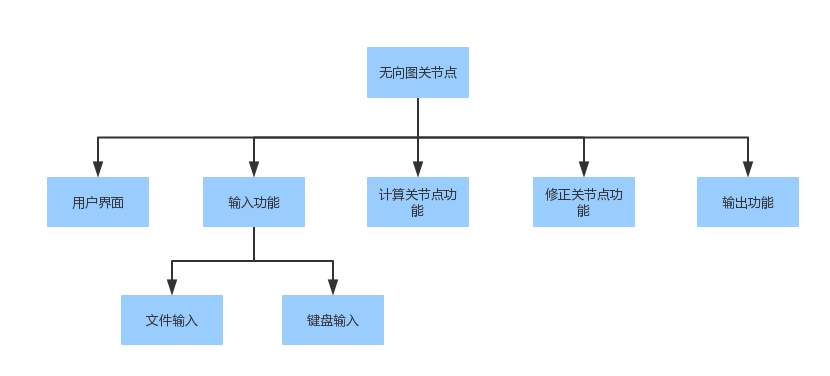
3.1 总体功能设计

进入程序之后，可以进行功能选择：从文件输入上次保存的图或者自己手动输入一个新的无向连通图；

若选择自动读入上次输入的图后，将文件中的图读入程序，若选择自己输入一个新的图，则按步骤输入图的结点以及图的每一条边；

之后根据程序步骤，即可得到图中的所有关节点；

若在选择将图中关节点修正（即使图中不存在关节点），即可得到修正后的无关节点的无向连通图。



3.2 数据结构设计

本程序设计了无向图的邻接表的数据结构；

template<class ElemType> //邻接表图定点结点类

class AdjListGraphVexNode

{

public:

ElemType data;

LinkList<int>\* adjLink;

AdjListGraphVexNode();

AdjListGraphVexNode(ElemType item, LinkList<int>\* adj=NULL);

};

template<class ElemType> //无向图邻接表类

class AdjListUndirGraph

{

protected:

int vexNum,edgeNum;

AdjListGraphVexNode<ElemType>\* vexTable;

mutable StatusCode\* tag;

void DestroyHelp();

int IndexHelp(const LinkList<int>\* la,int v) const;

public:

AdjListUndirGraph(ElemType es[],int vertexNum=DEFAULT\_SIZE);

AdjListUndirGraph(int vertexNum=DEFAULT\_SIZE);

AdjListUndirGraph(const AdjListUndirGraph<ElemType>& copy);

~AdjListUndirGraph();

StatusCode GetElem(int v,ElemType &e) const;

StatusCode SetElem(int v,const ElemType &e);

ElemType GetInfinity() const;

int GetVexNum() const;

int GetEdgeNum() const;

int FirstAdjVex(int v) const;

int NextAdjVex(int v1,int v2) const;

void InsertEdge(int v1,int v2);

void DeleteEdge(int v1,int v2);

StatusCode GetTag(int v) const;

void SetTag(int v,StatusCode val) const;

AdjListUndirGraph<ElemType> &operator = (const AdjListUndirGraph<ElemType>&copy);

};

设计了链、栈和队列的数据结构；

template <class ElemType> //结点类

struct Node

{

ElemType data;

Node<ElemType>\* next;

Node();

Node(ElemType item, Node<ElemType>\*link=NULL);

};

template<class ElemType> //链表类

class LinkList

{

protected:

Node<ElemType> \*head;

mutable int curPosition;

mutable Node<ElemType>\* curPtr;

int count;

Node<ElemType>\* GetElemPtr(int position) const;

void Init();

public:

LinkList();

virtual ~ LinkList();

int Length() const;

int GetCurPosition() const;

bool Empty() const;

void Clear();

void Traverse(void (\* Visit)(ElemType &));

StatusCode GetElem (int position, ElemType &e) const;

StatusCode SetElem (int position, const ElemType &e);

StatusCode Delete (int position, ElemType &e);

StatusCode Insert (int position, const ElemType &e);

LinkList (const LinkList<ElemType> &copy);

LinkList<ElemType> &operator=(const LinkList<ElemType> &copy);

};

template<class ElemType> //链栈类

class LinkStack

{

protected:

Node<ElemType>\* top;

void Init();

public:

LinkStack();

virtual ~LinkStack();

bool Empty() const;

void Clear();

bool Push(const ElemType &e); //入栈

bool Top(ElemType &e) const; //返回栈顶元素

bool Pop(ElemType &e); //出栈(e返回所删除的栈顶元素)

};

template<class ElemType>//链队列类，注！此队列构造对头头节点不存放数据

class LinkQueue

{

protected:

Node<ElemType> \*rear, \*front;

void Init();

public:

LinkQueue();

LinkQueue(const LinkQueue<ElemType> &copy);

virtual ~LinkQueue();

bool Empty() const;

void Clear();

int Length() const; //求队列长度

bool InQueue(const ElemType &e); //入队

bool GetHead(ElemType &e) const; //取队头

bool OutQueue(ElemType &e); //出队(e返回所删除的队头元素)

friend void PrintPostfix(LinkQueue<char> L); //输出后缀表达式

//已写复制构造函数

};

3.3 函数原型设计

template <class ElemType>

void Display(const AdjListUndirGraph<ElemType> &g, bool showVexElem); // 显示邻接矩阵无向图

template <class ElemType>

void FindArticul(const AdjListUndirGraph<ElemType> &G);

template <class ElemType>

void BFSTraverse(const AdjListUndirGraph<ElemType> &g, void (\*visit)(const ElemType &));

// 操作结果：对图g进行广度优先遍历

template <class ElemType>

void BFS(const AdjListUndirGraph<ElemType> &g, int v, void (\*visit)(const ElemType &));

// 操作结果：从第顶点v出发进行广度优先搜索图g

template <class ElemType>

void DFSTraverse(const AdjListUndirGraph<ElemType> &g, void (\*visit)(const ElemType &))

// 操作结果：对图g进行深度优先遍历;

template <class ElemType>

void DFS(const AdjListUndirGraph<ElemType> &g, int v, void (\*visit)(const ElemType &));

// 操作结果：从顶点v出发进行深度优先搜索图g

static bool UserSaysYes();

static char GetChar(istream &inStream = cin);

static char GetChar(istream &inStream);

template <class ElemType>

AdjListGraphVexNode<ElemType>::AdjListGraphVexNode();

template <class ElemType>

AdjListGraphVexNode<ElemType>::AdjListGraphVexNode(ElemType item, LinkList<int> \*adj);

template<class ElemType>

int AdjListUndirGraph<ElemType>::IndexHelp(const LinkList<int> \*la, int v) const;

template <class ElemType>

void AdjListUndirGraph<ElemType>::DestroyHelp();

AdjListUndirGraph(ElemType es[],int vertexNum=DEFAULT\_SIZE);

AdjListUndirGraph(int vertexNum=DEFAULT\_SIZE);

AdjListUndirGraph(const AdjListUndirGraph<ElemType>& copy);

~AdjListUndirGraph();

StatusCode GetElem(int v,ElemType &e) const;

StatusCode SetElem(int v,const ElemType &e);

ElemType GetInfinity() const;

int GetVexNum() const;

int GetEdgeNum() const;

int FirstAdjVex(int v) const;

int NextAdjVex(int v1,int v2) const;

void InsertEdge(int v1,int v2);

void DeleteEdge(int v1,int v2);

StatusCode GetTag(int v) const;

void SetTag(int v,StatusCode val) const;

AdjListUndirGraph<ElemType> &operator = (const AdjListUndirGraph<ElemType>&copy);

Node();

Node(ElemType item, Node<ElemType>\*link=NULL);

LinkList();

virtual ~ LinkList();

int Length() const;

int GetCurPosition() const;

bool Empty() const;

void Clear();

void Traverse(void (\* Visit)(ElemType &));

StatusCode GetElem (int position, ElemType &e) const;

StatusCode SetElem (int position, const ElemType &e);

StatusCode Delete (int position, ElemType &e);

StatusCode Insert (int position, const ElemType &e);

LinkList (const LinkList<ElemType> &copy);

LinkList<ElemType> &operator=(const LinkList<ElemType> &copy);

LinkStack();

virtual ~LinkStack();

bool Empty() const;

void Clear();

bool Push(const ElemType &e); //入栈

bool Top(ElemType &e) const; //返回栈顶元素

bool Pop(ElemType &e); //出栈(e返回所删除的栈顶元素)

LinkQueue();

LinkQueue(const LinkQueue<ElemType> &copy);

virtual ~LinkQueue();

bool Empty() const;

void Clear();

int Length() const; //求队列长度

bool InQueue(const ElemType &e); //入队

bool GetHead(ElemType &e) const; //取队头

bool OutQueue(ElemType &e); //出队(e返回所删除的队头元素)

friend void PrintPostfix(LinkQueue<char> L); //输出后缀表达式

int main();

3.4 主算法设计

template<class ElemType>

void DFSArticul(const AdjListUndirGraph<ElemType> &g,int v0,int Visited[],int low[],int couted[])

{

static int count=1;

int min=0;

Visited[v0]=min=++count;

g.SetTag(v0,VISITED);

for (int w = g.FirstAdjVex(v0); w != -1; w = g.NextAdjVex(v0, w))

{ // 对v的尚未访问过的邻接顶点w递归调用DFS

if (g.GetTag(w) == UNVISITED)

{ // 从w开始进行深度优先搜索

#ifdef \_MSC\_VER

DFS<ElemType>(g, w , Visited,low,couted); // VC需<ElemType>确定函数模板参数

if(low[w]<min) min=low[w];

if(low[w]>=Visited[v0]&&couted[v0]==0)

{

couted[v0]=1;

cout <<" "<< v0 <<" ";

ElemType e;

g.GetElem(v0, e);

cout << e << " ";

}

#else

DFSArticul(g, w , Visited,low,couted);

if(low[w]<min) min=low[w];

if(low[w]>=Visited[v0]&&couted[v0]==0)

{

couted[v0]=1;

cout <<" "<< v0 <<" ";

ElemType e;

g.GetElem(v0, e);

cout << e << " ";

}

#endif

}

else if(g.GetTag(w)==VISITED)

{

if (min>Visited[w])min=Visited[w];

}

}

low[v0]=min;

if(v0==0)

{

for(int i=0;i<g.GetVexNum();i++)

{

if(Visited[i]!=0) Visited[i]=1;

}

}

}

template<class ElemType>

void FindArticul(const AdjListUndirGraph<ElemType> &G)

{

int Visited[G.GetVexNum()];

int low[G.GetVexNum()];

int couted[G.GetVexNum()];

Visited[0]=1;

couted[0]=0;

for(int i=1;i<G.GetVexNum();++i) { G.SetTag(i,UNVISITED);Visited[i]=0; couted[i]=0;}

G.SetTag(0,VISITED);

int v;

v=G.FirstAdjVex(0);

DFSArticul(G,v,Visited,low,couted);

int count=0;

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

{

if(G.GetTag(i)==VISITED)

count++;

}

if(count<G.GetVexNum()&&couted[0]==0)

{

couted[0]=1;

cout <<" "<< "0" <<" ";

ElemType e;

G.GetElem(0, e);

cout << e << " ";

while(G.NextAdjVex(0,v)>0)

{

v=G.NextAdjVex(0,v);

if(Visited[v]==0) DFSArticul(G,v,Visited,low,couted);

}

}

}

template<class ElemType>

void DFSCArticul(AdjListUndirGraph<ElemType> &g,int v0,int Visited[],int low[],int Articul[])

{

static int count=1;

int min=0;

Visited[v0]=min=++count;

g.SetTag(v0,VISITED);

for (int w = g.FirstAdjVex(v0); w != -1; w = g.NextAdjVex(v0, w))

{ // 对v的尚未访问过的邻接顶点w递归调用DFS

if (g.GetTag(w) == UNVISITED)

{ // 从w开始进行深度优先搜索

#ifdef \_MSC\_VER

DFS<ElemType>(g, w , visit,Articul); // VC需<ElemType>确定函数模板参数

if(low[w]<min) min=low[w];

if(low[w]>=Visited[v0])

{

Articul[v0]=1;

}

#else

DFSCArticul(g, w , Visited,low,Articul);

if(low[w]<min) min=low[w];

if(low[w]>=Visited[v0])

{

g.InsertEdge(w,0);

min=1;

}

#endif

}

else if(g.GetTag(w)==VISITED)

{

if (min>Visited[w]) min=Visited[w];

}

}

low[v0]=min;

if(v0==0)

{

for(int i=0;i<g.GetVexNum();i++)

{

if(Visited[i]!=0) Visited[i]=1;

}

}

}

template<class ElemType>

void CorrectArticul( AdjListUndirGraph<ElemType> &G)

{

int Visited[G.GetVexNum()];

int low[G.GetVexNum()];

int Articul[G.GetVexNum()];

Visited[0]=1;

Articul[0]=0;

for(int i=1;i<G.GetVexNum();++i) { G.SetTag(i,UNVISITED);Visited[i]=0; Articul[i]=0;}

G.SetTag(0,VISITED);

int v;

v=G.FirstAdjVex(0);

DFSCArticul(G,v,Visited,low,Articul);

int count=0;

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

{

if(G.GetTag(i)==VISITED)

count++;

}

if(count<G.GetVexNum())

{

Articul[0]=1;

while(G.NextAdjVex(0,v)>0)

{

v=G.NextAdjVex(0,v);

if(Visited[v]==0)

{

DFSCArticul(G,v,Visited,low,Articul);

for (int w = G.FirstAdjVex(0); w != -1; w = G.NextAdjVex(0, w))

{

if(G.NextAdjVex(0,w)==v)

{

G.InsertEdge(v,w);

break;

}

}

}

}

}

}

3.5 用户界面设计

while(option!=1&&option!=2)

{

cout <<"^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^"<<endl ;

cout <<"^ ^"<<endl ;

cout <<"^ --This is Articulation Point System-- ^"<<endl ;

cout <<"^ ^"<<endl ;

cout <<"^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^"<<endl ;

cout <<"^ ^"<<endl ;

cout <<"^ \_\_\_\_First, We need know the graph ! ^"<<endl ;

cout <<"^ ^"<<endl ;

cout <<"^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^"<<endl ;

cout <<"^ ^"<<endl ;

cout <<"^ \_\_\_\_We can get this information in File~ ^"<<endl ;

cout <<"^ ^"<<endl ;

cout <<"^ \_\_\_\_Or you should be entered manually~ ^"<<endl ;

cout <<"^ ^"<<endl ;

cout <<"^ ^"<<endl ;

cout <<"^ >>>>>>>>Please Enter 1. To get information in file<<<<<<<< ^"<<endl ;

cout <<"^ ^"<<endl ;

cout <<"^ >>>>>>>>Please Enter 2. To get information in Yourself<<<<<<<< ^"<<endl ;

cout <<"^ ^"<<endl ;

cout <<" Please Enter : ";

cin >> option;

}

3.6 输入输出设计

ifstream file;

file.open("Y:\\Work\\data\_structure\\data\_Stucture\_Super\_Big\\Articul\\data.txt");

char line[1000]={0};

int NodeNum=0;

file.getline(line, sizeof(line));

istringstream word(line);

word>>NodeNum;

char vexs[NodeNum] ;

char c='A';

for(int i=0;i<NodeNum&&c<='Z';i++,c++)

{

vexs[i]=c;

if (c=='Z') throw Error("顶点个数溢出!");

}

int m[NodeNum][NodeNum];

int i=0;

while(file.getline(line, sizeof(line)))

{

istringstream word(line);

for(int j=0; j<NodeNum; j++)

{

word>>m[i][j];

}

i++;

}

int u; // 临时变量

for (u = 0; u < n; u++)

{ // 生成邻接矩阵的行

for (int v = 0; v < n; v++)

{ // 生成邻接矩阵元素的值

if (m[u][v] == 1) g.InsertEdge(u, v);

}

}

cout << "Super undirected graph:" << endl;

Display(g);

cout << endl;

system("PAUSE");

system("CLS");

cout << "Super undirected graph:" << endl;

Display(g);

cout << endl;

cout << "Articulation Points are:";

FindArticul(g);

DFSTraverse<char>(g, Write<char>);// <char>用于确定函数模板参数

cout << endl;

cout << endl;

system("PAUSE");

system("CLS");

cout << "Correct Articulation Point ..........";

CorrectArticul(g);

cout << endl;

cout << endl;

cout << endl;

cout << endl;

system("PAUSE");

system("CLS");

cout << "Super undirected graph(NO ARTICULATION POINT) :" << endl;

Display(g);

cout << endl;

system("PAUSE");

break;

ofstream file1;

file1.open("Y:\\Work\\data\_structure\\data\_Stucture\_Super\_Big\\Articul\\data.txt");

if(file1.is\_open())

{

file1<<n;

file1<<endl;

for(int i=0;i<n;i++)

{

for(int j=0;j<n;j++)

{

file1<<m[i][j]<<'\t';

}

file1<<endl;

}

}

cout << "Please Enter how many Node in the Graph:" <<endl;

int NodeNum;

cin>>NodeNum;

cout << endl;

char vexs[NodeNum] ;

har c='A';

for(int i=0;i<NodeNum&&c<='Z';i++,c++)

{

vexs[i]=c;

if (c=='Z') throw Error("顶点个数溢出!");

}

int m[NodeNum][NodeNum];

for(int i=0;i<NodeNum;i++)

{

for(int j=0;j<NodeNum;j++)

m[i][j]=0;

}

bool YoN=true;

for(;YoN==true;)

{

cout<<"Do you Want Enter a Edge? (Yes or No)" <<endl;

YoN=UserSaysYes();

if(YoN==false) break;

cout<<"Node 1 is ( :"<<vexs[0]<<"~"<<vexs[NodeNum-1]<<"):"<<endl;

char node1;

cin>>node1;

cout<<"Node 2 is ( :"<<vexs[0]<<"~"<<vexs[NodeNum-1]<<"):"<<endl;

char node2;

cin>>node2;

int N1=0,N2=0;

for(int i=0;i<=NodeNum;i++)

{

if(node1==vexs[i]) {N1=i; break;}

if(i==NodeNum) { cout<<"顶点输入错误!"; N1=-1; break;}

}

if(N1==-1) continue;

for(int i=0;i<=NodeNum;i++)

{

if(node2==vexs[i]) {N2=i; break;}

if(i==NodeNum) {cout<<"顶点输入错误!"; N2=-1; break;}

}

if(N2==-1) continue;

m[N1][N2]=1;

m[N2][N1]=1;

}

int n = NodeNum;

AdjListUndirGraph<char> g(vexs, n);

int u; // 临时变量

for (u = 0; u < n; u++)

{ // 生成邻接矩阵的行

for (int v = 0; v < n; v++)

{ // 生成邻接矩阵元素的值

if (m[u][v] == 1) g.InsertEdge(u, v);

}

}

cout << "Super undirected graph:" << endl;

Display(g);

cout << endl;

system("PAUSE");

system("CLS");

cout << "Super undirected graph:" << endl;

Display(g);

cout << endl;

cout << "Articulation Points are:";

FindArticul(g);

// DFSTraverse<char>(g, Write<char>);// <char>用于确定函数模板参数

cout << endl;

cout << endl;

system("PAUSE");

system("CLS");

cout << "Correct Articulation Point ..........：";

CorrectArticul(g);

cout << endl;

cout << endl;

cout << endl;

cout << endl;

system("PAUSE");

system("CLS");

cout << "Super undirected graph(NO ARTICULATION POINT):" << endl;

Display(g);

cout << endl;

system("PAUSE");

ofstream file;

file.open("Y:\\Work\\data\_structure\\data\_Stucture\_Super\_Big\\Articul\\data.txt");

if(file.is\_open())

{

file<<n<<endl;

for(int i=0;i<n;i++)

{

for(int j=0;j<n;j++)

{

file<<m[i][j]<<'\t';

}

file<<endl;

}

}

file.close();

# 4 方案实现

4.1 开发环境与工具

开发环境code::blocks IDE，工具code::blocks 13.12

4.2 程序设计关键技术

深度遍历搜寻关节点；

深度遍历修正关节点（消除图中关节点）；

4.3 个人设计实现（按组员分工）

4.3.1 于梓元设计实现：

template<class ElemType>//链队列类，注！此队列构造对头头节点不存放数据

class LinkQueue

{

protected:

Node<ElemType> \*rear, \*front;

void Init();

public:

LinkQueue();

LinkQueue(const LinkQueue<ElemType> &copy);

virtual ~LinkQueue();

bool Empty() const;

void Clear();

int Length() const; //求队列长度

bool InQueue(const ElemType &e); //入队

bool GetHead(ElemType &e) const; //取队头

bool OutQueue(ElemType &e); //出队(e返回所删除的队头元素)

friend void PrintPostfix(LinkQueue<char> L); //输出后缀表达式

//已写复制构造函数

};

template <class ElemType>

void LinkQueue<ElemType>::Init()

{

rear=front=new Node<ElemType>;

}

template <class ElemType>

LinkQueue<ElemType>::LinkQueue()

{

Init();

}

template <class ElemType>

LinkQueue<ElemType>::LinkQueue(const LinkQueue<ElemType> &copy)

{

Init();

for(Node<ElemType>\*tmpPtr=copy.front->next;tmpPtr!=NULL;tmpPtr=tmpPtr->next)

{

InQueue(tmpPtr->data);

}

}

template <class ElemType>

LinkQueue<ElemType>::~LinkQueue()

{

Clear();

}

template <class ElemType>

int LinkQueue<ElemType>::Length() const

{

int count=0;

for(Node<ElemType>\* tmpPtr=front->next; tmpPtr!=NULL;tmpPtr=tmpPtr->next)

count++;

return count;

}

template <class ElemType>

void LinkQueue<ElemType>::Clear()

{

ElemType tmpElem;

while (Length()>0)

OutQueue(tmpElem);

}

template <class ElemType>

bool LinkQueue<ElemType>::Empty() const

{

return rear==front;

}

template <class ElemType>

bool LinkQueue<ElemType>::InQueue(const ElemType &e)

{

Node<ElemType>\* tmpPtr=new Node<ElemType>(e);

if(tmpPtr==NULL) return false;

else

{

rear->next=tmpPtr;

rear=tmpPtr;

return true;

}

}

template <class ElemType>

bool LinkQueue<ElemType>::GetHead(ElemType &e) const

{

if(Empty()) return false;

else

{

e=front->next->data;

return true;

}

}

template <class ElemType>

bool LinkQueue<ElemType>::OutQueue(ElemType &e)

{

if(Empty()) return false;

else

{

Node<ElemType>\* temPtr=front->next;

e=temPtr->data;

front->next=temPtr->next;

if(rear==temPtr)

{

rear=front;

}

delete temPtr;

return true;

}

}

template<class ElemType> //邻接表图定点结点类

class AdjListGraphVexNode

{

public:

ElemType data;

LinkList<int>\* adjLink;

AdjListGraphVexNode();

AdjListGraphVexNode(ElemType item, LinkList<int>\* adj=NULL);

};

template <class ElemType>

AdjListGraphVexNode<ElemType>::AdjListGraphVexNode()

{

adjLink = NULL;

}

template <class ElemType>

AdjListGraphVexNode<ElemType>::AdjListGraphVexNode(ElemType item, LinkList<int> \*adj)

{

data = item;

adjLink = adj;

}

template<class ElemType> //无向图邻接表类

class AdjListUndirGraph

{

protected:

int vexNum,edgeNum;

AdjListGraphVexNode<ElemType>\* vexTable;

mutable StatusCode\* tag;

void DestroyHelp();

int IndexHelp(const LinkList<int>\* la,int v) const;

public:

AdjListUndirGraph(ElemType es[],int vertexNum=DEFAULT\_SIZE);

AdjListUndirGraph(int vertexNum=DEFAULT\_SIZE);

AdjListUndirGraph(const AdjListUndirGraph<ElemType>& copy);

~AdjListUndirGraph();

StatusCode GetElem(int v,ElemType &e) const;

StatusCode SetElem(int v,const ElemType &e);

ElemType GetInfinity() const;

int GetVexNum() const;

int GetEdgeNum() const;

int FirstAdjVex(int v) const;

int NextAdjVex(int v1,int v2) const;

void InsertEdge(int v1,int v2);

void DeleteEdge(int v1,int v2);

StatusCode GetTag(int v) const;

void SetTag(int v,StatusCode val) const;

AdjListUndirGraph<ElemType> &operator = (const AdjListUndirGraph<ElemType>&copy);

};

template<class ElemType>

int AdjListUndirGraph<ElemType>::IndexHelp(const LinkList<int> \*la, int v) const

{

int curPos, adjVex;

curPos=la->GetCurPosition();

la->GetElem(curPos,adjVex);

if(adjVex==v) return curPos;

curPos=1;

for(curPos=1;curPos<=la->Length();curPos++)

{

la->GetElem(curPos,adjVex);

if(adjVex==v) break;

}

return curPos;

}

template <class ElemType>

void AdjListUndirGraph<ElemType>::DestroyHelp()

{

delete []tag; // 释放标志

for (int iPos = 0; iPos < vexNum; iPos++)

{ // 释放链表

if (vexTable[iPos].adjLink != NULL)

delete vexTable[iPos].adjLink;

}

delete []vexTable; // 释放邻接表

}

template <class ElemType>

AdjListUndirGraph<ElemType>::AdjListUndirGraph(int vertexNum)

{

if (vertexNum < 0) throw "顶点个数不能为负!";// 抛出异常

vexNum = vertexNum;

edgeNum = 0; // 边数为0

tag = new StatusCode[vexNum];

int curPos;

for (curPos = 0; curPos < vexNum; curPos++)

{

tag[curPos] = UNVISITED;

}

vexTable = new AdjListGraphVexNode<ElemType>[vexNum];// 生成邻接表

}

template <class ElemType>

AdjListUndirGraph<ElemType>::AdjListUndirGraph(ElemType es[], int vertexNum)

// 操作结果：构造顶点数为numVex,顶点数据为es[],顶点个数为vertexNum,边数为0的无向图

{

if (vertexNum < 0) throw Error("顶点个数不能为负!");// 抛出异常

vexNum = vertexNum;

edgeNum = 0;

tag = new StatusCode[vexNum];

int curPos;

for (curPos = 0; curPos < vexNum; curPos++)

{

tag[curPos] = UNVISITED;

}

vexTable = new AdjListGraphVexNode<ElemType>[vexNum];// 生成邻接表

for (curPos = 0; curPos < vexNum; curPos++)

{

vexTable[curPos].data = es[curPos];

}

}

template<class ElemType>

AdjListUndirGraph<ElemType>::AdjListUndirGraph(const AdjListUndirGraph<ElemType> &copy)

{

int curPos; // 临时变量

vexNum = copy.vexNum; // 复制顶点数

edgeNum = copy.edgeNum; // 复制边数

tag = new bool[vexNum]; // 生成标志数组

for (curPos = 0; curPos < vexNum; curPos++)

{ // 复制标志数组

tag[curPos] = copy.tag[curPos];

}

vexTable = new AdjListGraphVexNode<ElemType>[vexNum];// 生成邻接表

for (curPos = 0; curPos < vexNum; curPos++)

{ // 复制邻接链表

vexTable[curPos].data = copy.vexTable[curPos].data; // 复制顶点数据

vexTable[curPos].adjLink = new LinkList<int>(\*copy.vexTable[curPos].adjLink);

}

}

template <class ElemType>

AdjListUndirGraph<ElemType>::~AdjListUndirGraph()

{

DestroyHelp();

}

template <class ElemType>

int AdjListUndirGraph<ElemType>::GetVexNum() const

// 操作结果：返回顶点个数

{

return vexNum;

}

template <class ElemType>

int AdjListUndirGraph<ElemType>::GetEdgeNum() const

// 操作结果：返回边数个数

{

return edgeNum;

}

template <class ElemType>

StatusCode AdjListUndirGraph<ElemType>::GetElem(int v, ElemType &e) const

// 操作结果：求顶点v的元素, v的取值范围为0 ≤ v ＜ vexNum, v合法时返回

{

if (v < 0 || v >= vexNum)

{ // v范围错

return RANGE\_ERROR; // 元素不存在

}

else

{ // v合法

e = vexTable[v].data; // 将顶点v的元素值赋给e

return ENTRY\_FOUND; // 元素存在

}

}

template<class ElemType>

StatusCode AdjListUndirGraph<ElemType>::SetElem(int v, const ElemType &e)

{

if (v < 0 || v >= vexNum)

{ // v范围错

return RANGE\_ERROR; // 位置错

}

else

{ // v合法

vexTable[v].data = e; // 顶点元素

return SUCCESS; // 成功

}

}

template<class ElemType>

ElemType AdjListUndirGraph<ElemType>::GetInfinity() const

{}

template <class ElemType>

StatusCode AdjListUndirGraph<ElemType>::GetTag(int v) const

// 操作结果：返回顶点v的标志

{

if (v < 0 || v >= vexNum) throw Error("v不合法!"); // 抛出异常

return tag[v];

}

template <class ElemType>

void AdjListUndirGraph<ElemType>::SetTag(int v, StatusCode val) const

// 操作结果：设置顶点v的标志为val

{

if (v < 0 || v >= vexNum) throw Error("v不合法!"); // 抛出异常

tag[v] = val;

}

template <class ElemType>

int AdjListUndirGraph<ElemType>::FirstAdjVex(int v) const

// 操作结果：返回顶点v的第一个邻接点

{

if (v < 0 || v >= vexNum) throw Error("v不合法!");// 抛出异常

if (vexTable[v].adjLink == NULL)

{ // 空邻接链表，无邻接点

return -1;

}

else

{ // 非空邻接链表，存在邻接点

int adjVex;

vexTable[v].adjLink->GetElem(1, adjVex);

return adjVex;

}

}

template<class ElemType>

int AdjListUndirGraph<ElemType>::NextAdjVex(int v1, int v2) const

{

if(v1<0||v1>=vexNum) throw Error("v1不合法!");;

if(v2<0||v2>=vexNum) throw Error("v2不合法!");

if(v1==v2) throw Error("v1不能等于v2!");

if(vexTable[v1].adjLink==NULL) return -1;

int curPos=IndexHelp(vexTable[v1].adjLink,v2);

if(curPos<vexTable[v1].adjLink->Length())

{

int adjVex;

vexTable[v1].adjLink->GetElem(curPos+1,adjVex);

return adjVex;

}

else return -1;

}

template <class ElemType>

void AdjListUndirGraph<ElemType>::InsertEdge(int v1, int v2)

// 操作结果：插入顶点为v1和v2的边

{

if (v1 < 0 || v1 >= vexNum) throw Error("v1不合法!"); // 抛出异常

if (v2 < 0 || v2 >= vexNum) throw Error("v2不合法!"); // 抛出异常

if (v1 == v2) throw Error("v1不能等于v2!"); // 抛出异常

// 插入<v1, v2>

if (vexTable[v1].adjLink == NULL)

{ // 空链表

vexTable[v1].adjLink = new LinkList<int>;

}

int curPos = IndexHelp(vexTable[v1].adjLink, v2); // 取出v2在邻接链表中的位置

if (curPos > vexTable[v1].adjLink->Length())

{ // 不存在边<v1, v2>

vexTable[v1].adjLink->Insert(curPos, v2); // 插入边

edgeNum++; // 边数自增1

}

// 插入<v2, v1>

if (vexTable[v2].adjLink == NULL)

{ // 空链表

vexTable[v2].adjLink = new LinkList<int>;

}

curPos = IndexHelp(vexTable[v2].adjLink, v1); // 取出v1在邻接链表中的位置

if (curPos > vexTable[v2].adjLink->Length())

{ // 不存在边<v1, v2>

vexTable[v2].adjLink->Insert(curPos, v1); // 插入边

}

}

template <class ElemType>

void AdjListUndirGraph<ElemType>::DeleteEdge(int v1, int v2)

// 操作结果：删除顶点为v1和v2的边

{

if (v1 < 0 || v1 >= vexNum) throw Error("v1不合法!"); // 抛出异常

if (v2 < 0 || v2 >= vexNum) throw Error("v2不合法!"); // 抛出异常

if (v1 == v2) throw Error("v1不能等于v2!"); // 抛出异常

int curPos = IndexHelp(vexTable[v1].adjLink, v2); // 取出v2在邻接链表中的位置

if (curPos <= vexTable[v1].adjLink->Length())

{ // 存在边<v1, v2>

vexTable[v1].adjLink->Delete(curPos, v2); // 删除<v1, v2>

edgeNum--; // 边数自减1

}

curPos = IndexHelp(vexTable[v2].adjLink, v1); // 取出v1在邻接链表中的位置

if (curPos <= vexTable[v2].adjLink->Length())

{ // 存在边<v2, v1>

vexTable[v2].adjLink->Delete(curPos, v1); // 删除<v2, v1>

}

}

template <class ElemType>

AdjListUndirGraph<ElemType> &AdjListUndirGraph<ElemType>::operator =(const AdjListUndirGraph<ElemType> &copy)

// 操作结果：将无向图的邻接矩阵copy赋值给当前无向图的邻接矩阵——重载赋值运算符

{

if (&copy != this)

{

DestroyHelp(); // 释放当前无向图占用空间

int curPos; // 临时变量

vexNum = copy.vexNum; // 复制顶点数

edgeNum = copy.edgeNum; // 复制边数

tag = new bool[vexNum]; // 生成标志数组

for (curPos = 0; curPos < vexNum; curPos++)

{ // 复制标志数组

tag[curPos] = copy.tag[curPos];

}

vexTable = new AdjListGraphVexNode<ElemType>[vexNum];// 生成邻接表

for (curPos = 0; curPos < vexNum; curPos++)

{ // 复制邻接链表

vexTable[curPos].data = copy.vexTable[curPos].data; // 复制顶点数据

vexTable[curPos].adjLink = new LinkList<int>(\*copy.vexTable[curPos].adjLink);

}

}

return \*this;

}

template<class ElemType>

void DFSArticul(const AdjListUndirGraph<ElemType> &g,int v0,int Visited[],int low[],int couted[])

{

static int count=1;

int min=0;

Visited[v0]=min=++count;

g.SetTag(v0,VISITED);

for (int w = g.FirstAdjVex(v0); w != -1; w = g.NextAdjVex(v0, w))

{ // 对v的尚未访问过的邻接顶点w递归调用DFS

if (g.GetTag(w) == UNVISITED)

{ // 从w开始进行深度优先搜索

#ifdef \_MSC\_VER

DFS<ElemType>(g, w , Visited,low,couted); // VC需<ElemType>确定函数模板参数

if(low[w]<min) min=low[w];

if(low[w]>=Visited[v0]&&couted[v0]==0)

{

couted[v0]=1;

cout <<" "<< v0 <<" ";

ElemType e;

g.GetElem(v0, e);

cout << e << " ";

}

#else

DFSArticul(g, w , Visited,low,couted);

if(low[w]<min) min=low[w];

if(low[w]>=Visited[v0]&&couted[v0]==0)

{

couted[v0]=1;

cout <<" "<< v0 <<" ";

ElemType e;

g.GetElem(v0, e);

cout << e << " ";

}

#endif

}

else if(g.GetTag(w)==VISITED)

{

if (min>Visited[w])min=Visited[w];

}

}

low[v0]=min;

if(v0==0)

{

for(int i=0;i<g.GetVexNum();i++)

{

if(Visited[i]!=0) Visited[i]=1;

}

}

}

template<class ElemType>

void FindArticul(const AdjListUndirGraph<ElemType> &G)

{

int Visited[G.GetVexNum()];

int low[G.GetVexNum()];

int couted[G.GetVexNum()];

Visited[0]=1;

couted[0]=0;

for(int i=1;i<G.GetVexNum();++i) { G.SetTag(i,UNVISITED);Visited[i]=0; couted[i]=0;}

G.SetTag(0,VISITED);

int v;

v=G.FirstAdjVex(0);

DFSArticul(G,v,Visited,low,couted);

int count=0;

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

{

if(G.GetTag(i)==VISITED)

count++;

}

if(count<G.GetVexNum()&&couted[0]==0)

{

couted[0]=1;

cout <<" "<< "0" <<" ";

ElemType e;

G.GetElem(0, e);

cout << e << " ";

while(G.NextAdjVex(0,v)>0)

{

v=G.NextAdjVex(0,v);

if(Visited[v]==0) DFSArticul(G,v,Visited,low,couted);

}

}

}

template<class ElemType>

void DFSCArticul(AdjListUndirGraph<ElemType> &g,int v0,int Visited[],int low[],int Articul[])

{

static int count=1;

int min=0;

Visited[v0]=min=++count;

g.SetTag(v0,VISITED);

for (int w = g.FirstAdjVex(v0); w != -1; w = g.NextAdjVex(v0, w))

{ // 对v的尚未访问过的邻接顶点w递归调用DFS

if (g.GetTag(w) == UNVISITED)

{ // 从w开始进行深度优先搜索

#ifdef \_MSC\_VER

DFS<ElemType>(g, w , visit,Articul); // VC需<ElemType>确定函数模板参数

if(low[w]<min) min=low[w];

if(low[w]>=Visited[v0])

{

Articul[v0]=1;

}

#else

DFSCArticul(g, w , Visited,low,Articul);

if(low[w]<min) min=low[w];

if(low[w]>=Visited[v0])

{

g.InsertEdge(w,0);

min=1;

}

#endif

}

else if(g.GetTag(w)==VISITED)

{

if (min>Visited[w]) min=Visited[w];

}

}

low[v0]=min;

if(v0==0)

{

for(int i=0;i<g.GetVexNum();i++)

{

if(Visited[i]!=0) Visited[i]=1;

}

}

}

// 通用异常类

class Error

{

private:

// 数据成员

char message[MAX\_ERROR\_MESSAGE\_LEN];// 异常信息

public:

// 方法声明

Error(char mes[] = "一般性异常!") // 构造函数

{

strcpy(message, mes); // 复制异常信息

}

~Error(void) {}; // 析构函数

void Show() const // 显示异常信息

{

cout << message << endl; // 显示异常信息

}

};

int main()

{

int option=0;

while(option!=1&&option!=2)

{

cout <<"^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^"<<endl ;

cout <<"^ ^"<<endl ;

cout <<"^ --This is Articulation Point System-- ^"<<endl ;

cout <<"^ ^"<<endl ;

cout <<"^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^"<<endl ;

cout <<"^ ^"<<endl ;

cout <<"^ \_\_\_\_First, We need know the graph ! ^"<<endl ;

cout <<"^ ^"<<endl ;

cout <<"^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^"<<endl ;

cout <<"^ ^"<<endl ;

cout <<"^ \_\_\_\_We can get this information in File~ ^"<<endl ;

cout <<"^ ^"<<endl ;

cout <<"^ \_\_\_\_Or you should be entered manually~ ^"<<endl ;

cout <<"^ ^"<<endl ;

cout <<"^ ^"<<endl ;

cout <<"^ >>>>>>>>Please Enter 1. To get information in file<<<<<<<< ^"<<endl ;

cout <<"^ ^"<<endl ;

cout <<"^ >>>>>>>>Please Enter 2. To get information in Yourself<<<<<<<< ^"<<endl ;

cout <<"^ ^"<<endl ;

cout <<" Please Enter : ";

cin >> option;

}

try // 用try封装可能出现异常的代码

{

system("CLS");

switch (option)

{

case 1:

{

ifstream file;

file.open("Y:\\Work\\data\_structure\\data\_Stucture\_Super\_Big\\Articul\\data.txt");

char line[1000]={0};

int NodeNum=0;

file.getline(line, sizeof(line));

istringstream word(line);

word>>NodeNum;

char vexs[NodeNum] ;

char c='A';

for(int i=0;i<NodeNum&&c<='Z';i++,c++)

{

vexs[i]=c;

if (c=='Z') throw Error("顶点个数溢出!");

}

int m[NodeNum][NodeNum];

int i=0;

while(file.getline(line, sizeof(line)))

{

istringstream word(line);

for(int j=0; j<NodeNum; j++)

{

word>>m[i][j];

}

i++;

}

file.close();

int n = NodeNum;

AdjListUndirGraph<char> g(vexs, n);

int u; // 临时变量

for (u = 0; u < n; u++)

{ // 生成邻接矩阵的行

for (int v = 0; v < n; v++)

{ // 生成邻接矩阵元素的值

if (m[u][v] == 1) g.InsertEdge(u, v);

}

}

cout << "Super undirected graph:" << endl;

Display(g);

cout << endl;

system("PAUSE");

system("CLS");

cout << "Super undirected graph:" << endl;

Display(g);

cout << endl;

cout << "Articulation Points are:";

FindArticul(g);

// DFSTraverse<char>(g, Write<char>);// <char>用于确定函数模板参数

cout << endl;

cout << endl;

system("PAUSE");

system("CLS");

cout << "Correct Articulation Point ..........";

CorrectArticul(g);

cout << endl;

cout << endl;

cout << endl;

cout << endl;

system("PAUSE");

system("CLS");

cout << "Super undirected graph(NO ARTICULATION POINT) :" << endl;

Display(g);

cout << endl;

system("PAUSE");

break;

ofstream file1;

file1.open("Y:\\Work\\data\_structure\\data\_Stucture\_Super\_Big\\Articul\\data.txt");

if(file1.is\_open())

{

file1<<n;

file1<<endl;

for(int i=0;i<n;i++)

{

for(int j=0;j<n;j++)

{

file1<<m[i][j]<<'\t';

}

file1<<endl;

}

}

file.close();

}

case 2:

{

cout << "Please Enter how many Node in the Graph:" <<endl;

int NodeNum;

cin>>NodeNum;

cout << endl;

char vexs[NodeNum] ;

char c='A';

for(int i=0;i<NodeNum&&c<='Z';i++,c++)

{

vexs[i]=c;

if (c=='Z') throw Error("顶点个数溢出!");

}

int m[NodeNum][NodeNum];

for(int i=0;i<NodeNum;i++)

{

for(int j=0;j<NodeNum;j++)

m[i][j]=0;

}

bool YoN=true;

for(;YoN==true;)

{

cout<<"Do you Want Enter a Edge? (Yes or No)" <<endl;

YoN=UserSaysYes();

if(YoN==false) break;

cout<<"Node 1 is ( :"<<vexs[0]<<"~"<<vexs[NodeNum-1]<<"):"<<endl;

char node1;

cin>>node1;

cout<<"Node 2 is ( :"<<vexs[0]<<"~"<<vexs[NodeNum-1]<<"):"<<endl;

char node2;

cin>>node2;

int N1=0,N2=0;

for(int i=0;i<=NodeNum;i++)

{

if(node1==vexs[i]) {N1=i; break;}

if(i==NodeNum) { cout<<"顶点输入错误!"; N1=-1; break;}

}

if(N1==-1) continue;

for(int i=0;i<=NodeNum;i++)

{

if(node2==vexs[i]) {N2=i; break;}

if(i==NodeNum) {cout<<"顶点输入错误!"; N2=-1; break;}

}

if(N2==-1) continue;

m[N1][N2]=1;

m[N2][N1]=1;

}

int n = NodeNum;

AdjListUndirGraph<char> g(vexs, n);

int u; // 临时变量

for (u = 0; u < n; u++)

{ // 生成邻接矩阵的行

for (int v = 0; v < n; v++)

{ // 生成邻接矩阵元素的值

if (m[u][v] == 1) g.InsertEdge(u, v);

}

}

cout << "Super undirected graph:" << endl;

Display(g);

cout << endl;

system("PAUSE");

system("CLS");

cout << "Super undirected graph:" << endl;

Display(g);

cout << endl;

cout << "Articulation Points are:";

FindArticul(g);

// DFSTraverse<char>(g, Write<char>);// <char>用于确定函数模板参数

cout << endl;

cout << endl;

system("PAUSE");

system("CLS");

cout << "Correct Articulation Point ..........：";

CorrectArticul(g);

cout << endl;

cout << endl;

cout << endl;

cout << endl;

system("PAUSE");

system("CLS");

cout << "Super undirected graph(NO ARTICULATION POINT):" << endl;

Display(g);

cout << endl;

system("PAUSE");

ofstream file;

file.open("Y:\\Work\\data\_structure\\data\_Stucture\_Super\_Big\\Articul\\data.txt");

if(file.is\_open())

{

file<<n<<endl;

for(int i=0;i<n;i++)

{

for(int j=0;j<n;j++)

{

file<<m[i][j]<<'\t';

}

file<<endl;

}

}

file.close();

break;

}

}

}

catch (Error err) // 捕捉并处理异常

{

err.Show(); // 显示异常信息

}

return 0;

}

4.3.2 张伟颖设计实现：

template<class ElemType> //链栈类

class LinkStack

{

protected:

Node<ElemType>\* top;

void Init();

public:

LinkStack();

virtual ~LinkStack();

bool Empty() const;

void Clear();

bool Push(const ElemType &e); //入栈

bool Top(ElemType &e) const; //返回栈顶元素

bool Pop(ElemType &e); //出栈(e返回所删除的栈顶元素)

};

template <class ElemType>

void LinkStack<ElemType>::Init()

{

top=NULL;

}

template <class ElemType>

LinkStack<ElemType>::LinkStack()

{

Init();

}

template <class ElemType>

LinkStack<ElemType>::~LinkStack()

{

Clear();

}

template <class ElemType>

void LinkStack<ElemType>::Clear()

{

ElemType tmpElem;

while (Empty()==false)

Pop(tmpElem);

}

template <class ElemType>

bool LinkStack<ElemType>::Empty() const

{

return top==NULL;

}

template <class ElemType>

bool LinkStack<ElemType>::Push(const ElemType &e)

{

Node<ElemType>\* new\_top=new Node<ElemType>(e,top);

if(new\_top==NULL) return false;

else

{

top=new\_top;

return true;

}

}

template <class ElemType>

bool LinkStack<ElemType>::Top(ElemType &e) const

{

if(Empty()) return false;

else

{

e=top->data;

return true;

}

}

template <class ElemType>

bool LinkStack<ElemType>::Pop(ElemType &e)

{

if(Empty()) return false;

else

{

Node<ElemType>\* old\_top=top;

e=old\_top->data;

top=old\_top->next;

delete old\_top;

return true;

}

}

template<class ElemType>

void CorrectArticul( AdjListUndirGraph<ElemType> &G)

{

int Visited[G.GetVexNum()];

int low[G.GetVexNum()];

int Articul[G.GetVexNum()];

Visited[0]=1;

Articul[0]=0;

for(int i=1;i<G.GetVexNum();++i) { G.SetTag(i,UNVISITED);Visited[i]=0; Articul[i]=0;}

G.SetTag(0,VISITED);

int v;

v=G.FirstAdjVex(0);

DFSCArticul(G,v,Visited,low,Articul);

int count=0;

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

{

if(G.GetTag(i)==VISITED)

count++;

}

if(count<G.GetVexNum())

{

Articul[0]=1;

while(G.NextAdjVex(0,v)>0)

{

v=G.NextAdjVex(0,v);

if(Visited[v]==0)

{

DFSCArticul(G,v,Visited,low,Articul);

for (int w = G.FirstAdjVex(0); w != -1; w = G.NextAdjVex(0, w))

{

if(G.NextAdjVex(0,w)==v)

{

G.InsertEdge(v,w);

break;

}

}

}

}

}

}

static char GetChar(istream &inStream = cin); // 从输入流inStream中跳过空格及制表符获取一字符

template <class ElemType>

void BFSTraverse(const AdjListUndirGraph<ElemType> &g, void (\*visit)(const ElemType &))

// 操作结果：对图g进行广度优先遍历

{

int v;

for (v = 0; v < g.GetVexNum(); v++)

{ // 对每个顶点作访问标志

g.SetTag(v, UNVISITED);

}

for (v = 0; v < g.GetVexNum(); v++)

{ // 对尚未访问的顶点按BFS进行深度优先搜索

if (g.GetTag(v) == UNVISITED)

{

#ifdef \_MSC\_VER

BFS<ElemType>(g, v , visit); // VC需<ElemType>确定函数模板参数

#else

BFS(g, v , visit);

#endif

}

}

}

template <class ElemType>

void BFS(const AdjListUndirGraph<ElemType> &g, int v, void (\*visit)(const ElemType &))

// 操作结果：从第顶点v出发进行广度优先搜索图g

{

g.SetTag(v, VISITED); // 作访问标志

ElemType e; // 临时变量

g.GetElem(v, e); // 顶点v的数据元素

(\*visit)(e); // 访问顶点v的数据元素

LinkQueue<int> q; // 定义队列

q.InQueue(v); // v入队

while (!q.Empty())

{ // 队列q非空, 进行循环

int u, w; // 临时顶点

q.OutQueue(u); // 出队

for (w = g.FirstAdjVex(u); w >= 0; w = g.NextAdjVex(u, w))

{ // 对u尚未访问过的邻接顶点w进行访问

if (g.GetTag(w) == UNVISITED)

{ // 对w进行访问

g.SetTag(w, VISITED); // 作访问标志

g.GetElem(w, e); // 顶点w的数据元素

(\*visit)(e); // 访问顶点w的数据元素

q.InQueue(w); // w入队

}

}

}

}

template <class ElemType>

void DFSTraverse(const AdjListUndirGraph<ElemType> &g, void (\*visit)(const ElemType &))

// 操作结果：对图g进行深度优先遍历

{

int v;

for (v = 0; v < g.GetVexNum(); v++)

{ // 对每个顶点作访问标志

g.SetTag(v, UNVISITED);

}

for (v = 0; v < g.GetVexNum(); v++)

{ // 对尚未访问的顶点按DFS进行深度优先搜索

if (g.GetTag(v) == UNVISITED)

{ // 从v开始进行深度优先搜索

#ifdef \_MSC\_VER

DFS<ElemType>(g, v , visit); // VC需<ElemType>确定函数模板参数

#else

DFS(g, v , visit);

#endif

}

}

}

template <class ElemType>

void DFS(const AdjListUndirGraph<ElemType> &g, int v, void (\*visit)(const ElemType &))

// 操作结果：从顶点v出发进行深度优先搜索图g

{

g.SetTag(v, VISITED); // 作访问标志

ElemType e; // 临时变量

g.GetElem(v, e); // 顶点v的数据元素

(\*visit)(e); // 访问顶点v的数据元素

for (int w = g.FirstAdjVex(v); w != -1; w = g.NextAdjVex(v, w))

{ // 对v的尚未访问过的邻接顶点w递归调用DFS

if (g.GetTag(w) == UNVISITED)

{ // 从w开始进行深度优先搜索

#ifdef \_MSC\_VER

DFS<ElemType>(g, w , visit); // VC需<ElemType>确定函数模板参数

#else

DFS(g, w , visit);

#endif

}

}

}

static char GetChar(istream &inStream)

// 操作结果：从输入流inStream中跳过空格及制表符获取一字符

{

char ch; // 临时变量

while ((ch = (inStream).peek()) != EOF // 文件结束符(peek()函数从输入流中接受1

// 字符,流的当前位置不变)

&& ((ch = (inStream).get()) == ' ' // 空格(get()函数从输入流中接受1字符,流

// 的当前位置向后移1个位置)

|| ch == '\t')); // 制表符

return ch; // 返回字符

}

static bool UserSaysYes()

// 操作结果: 当用户肯定回答(yes)时, 返回true, 用户否定回答(no)时,返回false

{

char ch; // 用户回答字符

bool initialResponse = true; // 初始回答

do

{ // 循环直到用户输入恰当的回答为止

if (initialResponse)

{ // 初始回答

cout << "(y, n)?";

}

else

{ // 非初始回答

cout << "用y或n回答:";

}

while ((ch = GetChar()) == '\n'); // 跳过空格,制表符及换行符获取一字符

initialResponse = false;

} while (ch != 'y' && ch != 'Y' && ch != 'n' && ch != 'N');

while (GetChar() != '\n'); // 跳过当前行后面的字符

if (ch == 'y' || ch == 'Y') return true;

else return false;

}

4.3.3 孟霞设计实现：

template <class ElemType> //结点类

struct Node

{

ElemType data;

Node<ElemType>\* next;

Node();

Node(ElemType item, Node<ElemType>\*link=NULL);

};

template <class ElemType>

Node<ElemType>::Node()

{

next=NULL;

}

template <class ElemType>

Node<ElemType>::Node(ElemType item, Node<ElemType>\*link)

{

data=item;

next=link;

}

template<class ElemType> //链表类

class LinkList

{

protected:

Node<ElemType> \*head;

mutable int curPosition;

mutable Node<ElemType>\* curPtr;

int count;

Node<ElemType>\* GetElemPtr(int position) const;

void Init();

public:

LinkList();

virtual ~ LinkList();

int Length() const;

int GetCurPosition() const;

bool Empty() const;

void Clear();

void Traverse(void (\* Visit)(ElemType &));

StatusCode GetElem (int position, ElemType &e) const;

StatusCode SetElem (int position, const ElemType &e);

StatusCode Delete (int position, ElemType &e);

StatusCode Insert (int position, const ElemType &e);

LinkList (const LinkList<ElemType> &copy);

LinkList<ElemType> &operator=(const LinkList<ElemType> &copy);

};

template<class ElemType>

Node<ElemType>\* LinkList<ElemType>::GetElemPtr(int position) const

{

if(curPosition>position)

{

curPosition=0;

curPtr=head;

}

for(;curPosition<position;curPosition++)

{

curPtr=curPtr->next;

}

return curPtr;

}

template<class ElemType>

void LinkList<ElemType>::Init()

{

head=new Node<ElemType>;

curPtr=head; curPosition=0;

count=0;

}

template<class ElemType>

LinkList<ElemType>::LinkList()

{

Init();

}

template<class ElemType>

LinkList<ElemType>::LinkList(const LinkList<ElemType> &copy)

{

int copyLength=copy.Length();

ElemType e;

head=NULL;

Init();

for(int curPosition=1;curPosition<=copyLength;curPosition++)

{

copy.GetElem(curPosition,e);

Insert(Length()+1,e);

}

}

template<class ElemType>

LinkList<ElemType>::~LinkList()

{

Clear();

delete head;

}

template<class ElemType>

int LinkList<ElemType>::Length() const

{

return count;

}

template<class ElemType>

int LinkList<ElemType>::GetCurPosition() const

{

return curPosition;

}

template<class ElemType>

bool LinkList<ElemType>::Empty() const

{

return head->next==NULL;

}

template<class ElemType>

void LinkList<ElemType>::Clear()

{

ElemType tmpElem;

while(Length()>0)

{

Delete(1,tmpElem);

}

}

template<class ElemType>

void LinkList<ElemType>::Traverse(void (\*Visit)(ElemType &)) //此(\*Visit)为需要实现功能函数，按需使用

{

for(Node<ElemType>\*tmpPtr=head->next;tmpPtr!=NULL;tmpPtr=tmpPtr->next)

{

(\*Visit)(tmpPtr->data);

}

}

template<class ElemType>

StatusCode LinkList<ElemType>::GetElem(int position, ElemType &e) const

{

if(position<1||position>Length())

{

return RANGE\_ERROR; //范围错

}

else

{

Node<ElemType>\* tmpPtr;

tmpPtr=GetElemPtr(position);

e=tmpPtr->data;

return ENTRY\_FOUND;

}

}

template<class ElemType>

StatusCode LinkList<ElemType>::SetElem(int position, const ElemType &e)

{

if(position<1||position>Length())

{

return RANGE\_ERROR; //范围错

}

else

{

Node<ElemType>\* tmpPtr;

tmpPtr=GetElemPtr(position);

tmpPtr->data=e;

return SUCCESS;

}

}

template<class ElemType>

StatusCode LinkList<ElemType>::Insert(int position, const ElemType &e)

{

if(position<1||position>Length()+1)

{

return RANGE\_ERROR;

}

else

{

Node<ElemType>\* tmpPtr;

tmpPtr=GetElemPtr(position-1);

Node<ElemType>\* newPtr;

newPtr=new Node<ElemType>(e,tmpPtr->next);

tmpPtr->next=newPtr;

curPosition=position;

curPtr=newPtr;

count++;

return SUCCESS;

}

}

template<class ElemType>

StatusCode LinkList<ElemType>::Delete(int position, ElemType &e)

{

if(position<1||position>Length())

{

return RANGE\_ERROR;

}

else

{

Node<ElemType>\* tmpPtr;

tmpPtr=GetElemPtr(position-1);

Node<ElemType>\* nextPtr=tmpPtr->next;

tmpPtr->next=nextPtr->next;

e=nextPtr->data;

if(position==Length())

{

curPosition=0;

curPtr=head;

}

else

{

curPosition=position;

curPtr=tmpPtr->next;

}

count--;

delete nextPtr;

return SUCCESS;

}

}

template<class ElemType>

LinkList<ElemType> &LinkList<ElemType>::operator =(const LinkList<ElemType>& copy)

{

if(&copy!=this)

{

int copyLength=copy.Length();

ElemType e;

head=NULL;

Init();

for(int curPosition=1;curPosition<=copyLength;curPosition++)

{

copy.GetElem(curPosition,e);

Insert(Length()+1,e);

}

}

return \*this;

}

template<class ElemType>

void CorrectArticul( AdjListUndirGraph<ElemType> &G)

{

int Visited[G.GetVexNum()];

int low[G.GetVexNum()];

int Articul[G.GetVexNum()];

Visited[0]=1;

Articul[0]=0;

for(int i=1;i<G.GetVexNum();++i) { G.SetTag(i,UNVISITED);Visited[i]=0; Articul[i]=0;}

G.SetTag(0,VISITED);

int v;

v=G.FirstAdjVex(0);

DFSCArticul(G,v,Visited,low,Articul);

int count=0;

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

{

if(G.GetTag(i)==VISITED)

count++;

}

if(count<G.GetVexNum())

{

Articul[0]=1;

while(G.NextAdjVex(0,v)>0)

{

v=G.NextAdjVex(0,v);

if(Visited[v]==0)

{

DFSCArticul(G,v,Visited,low,Articul);

for (int w = G.FirstAdjVex(0); w != -1; w = G.NextAdjVex(0, w))

{

if(G.NextAdjVex(0,w)==v)

{

G.InsertEdge(v,w);

break;

}

}

}

}

}

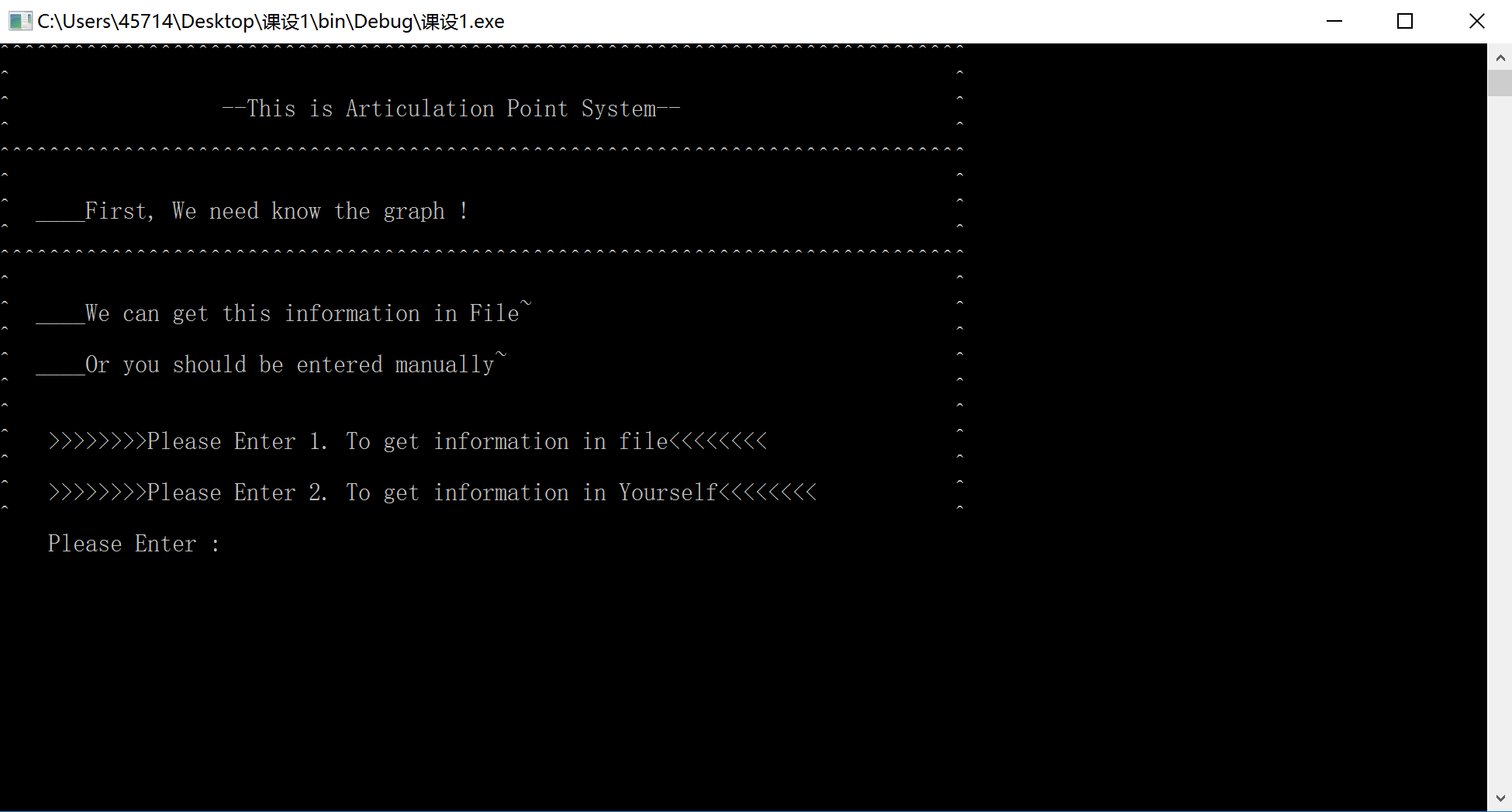
}

static char GetChar(istream &inStream = cin); // 从输入流inStream中跳过空格及制表符获取一字符

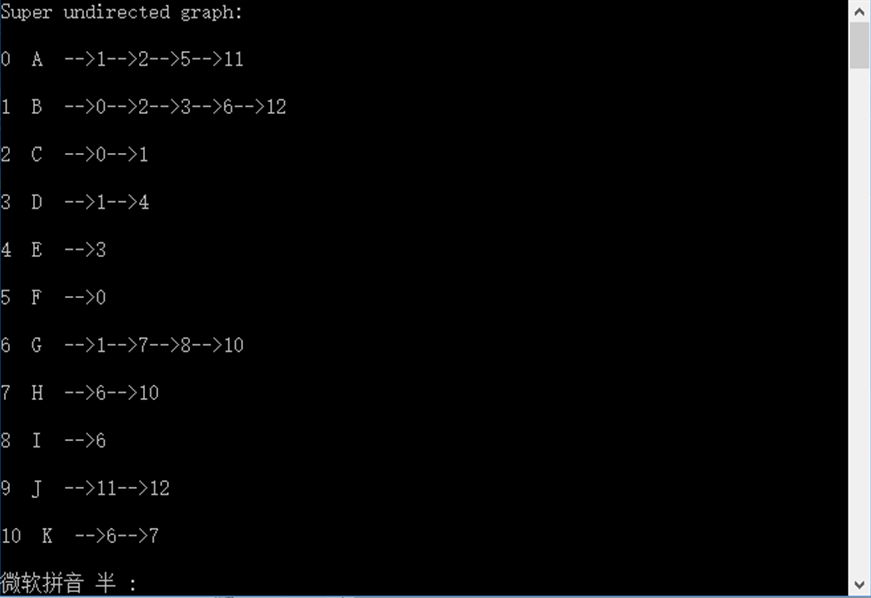
# 5 测试与调试

5.1 个人测试（按组员分工）

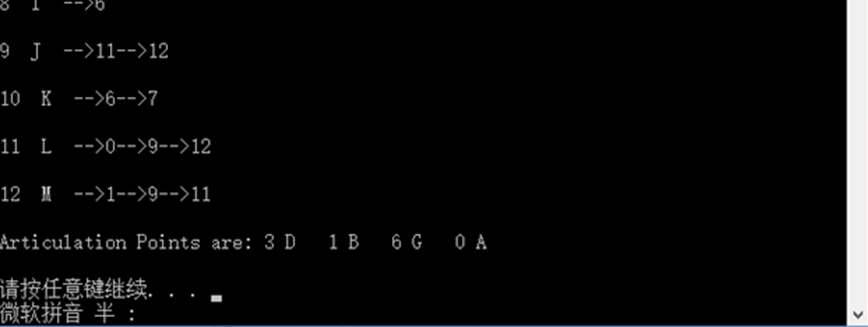
5.1.1 于梓元测试



主界面，选择文件输入或键盘输入



输入如下

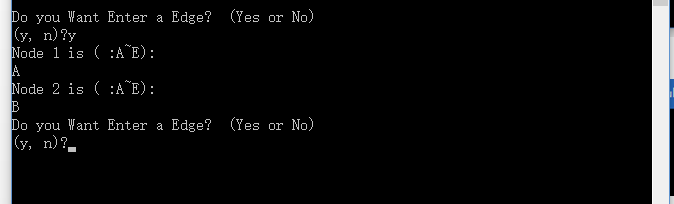


显示关节点

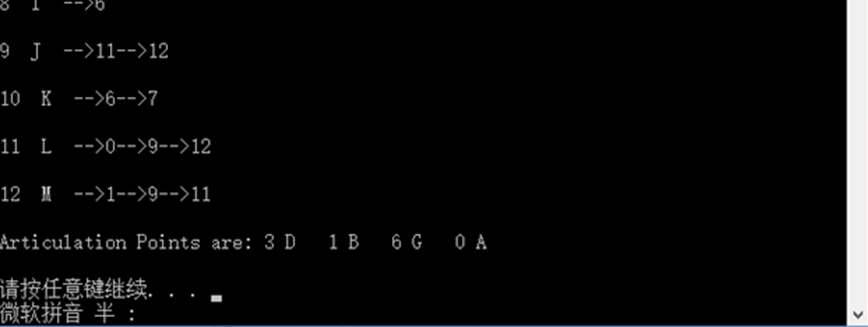
5.1.2张伟颖测试



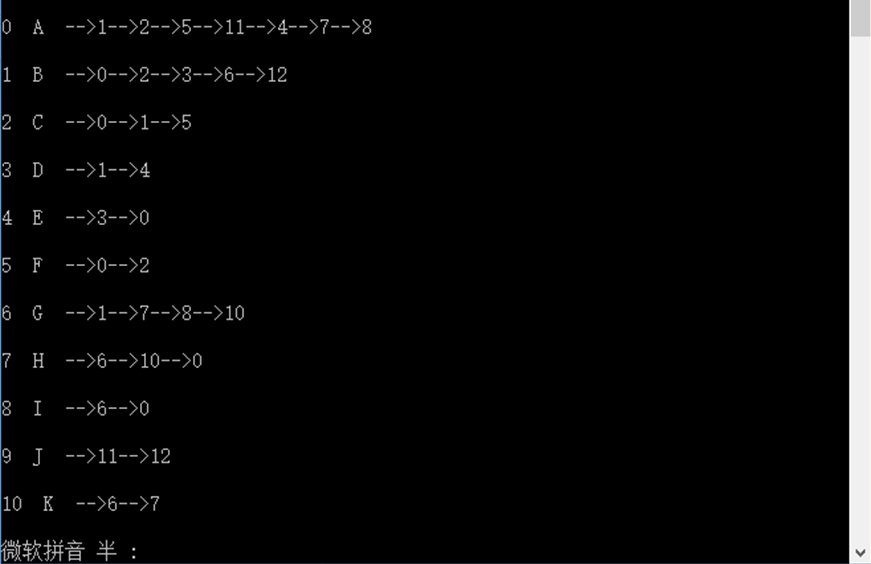
手动输入数据中



判断是否继续输入



显示关节点

修正后如图

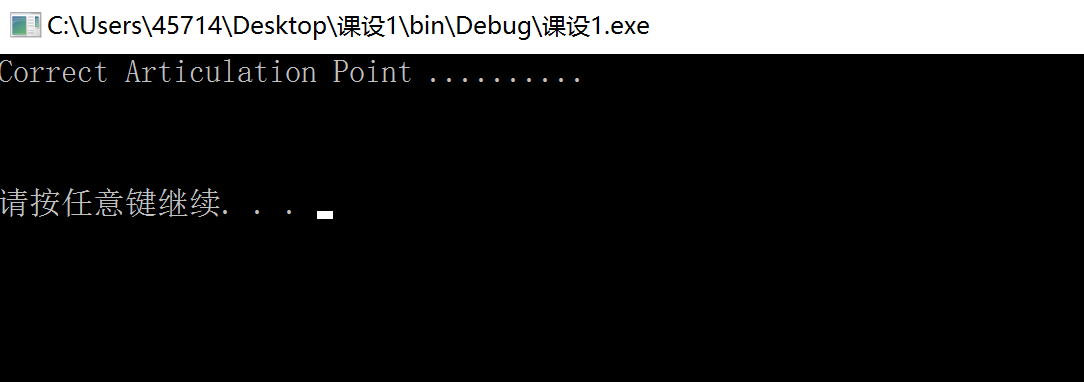
5.1.3孟霞测试

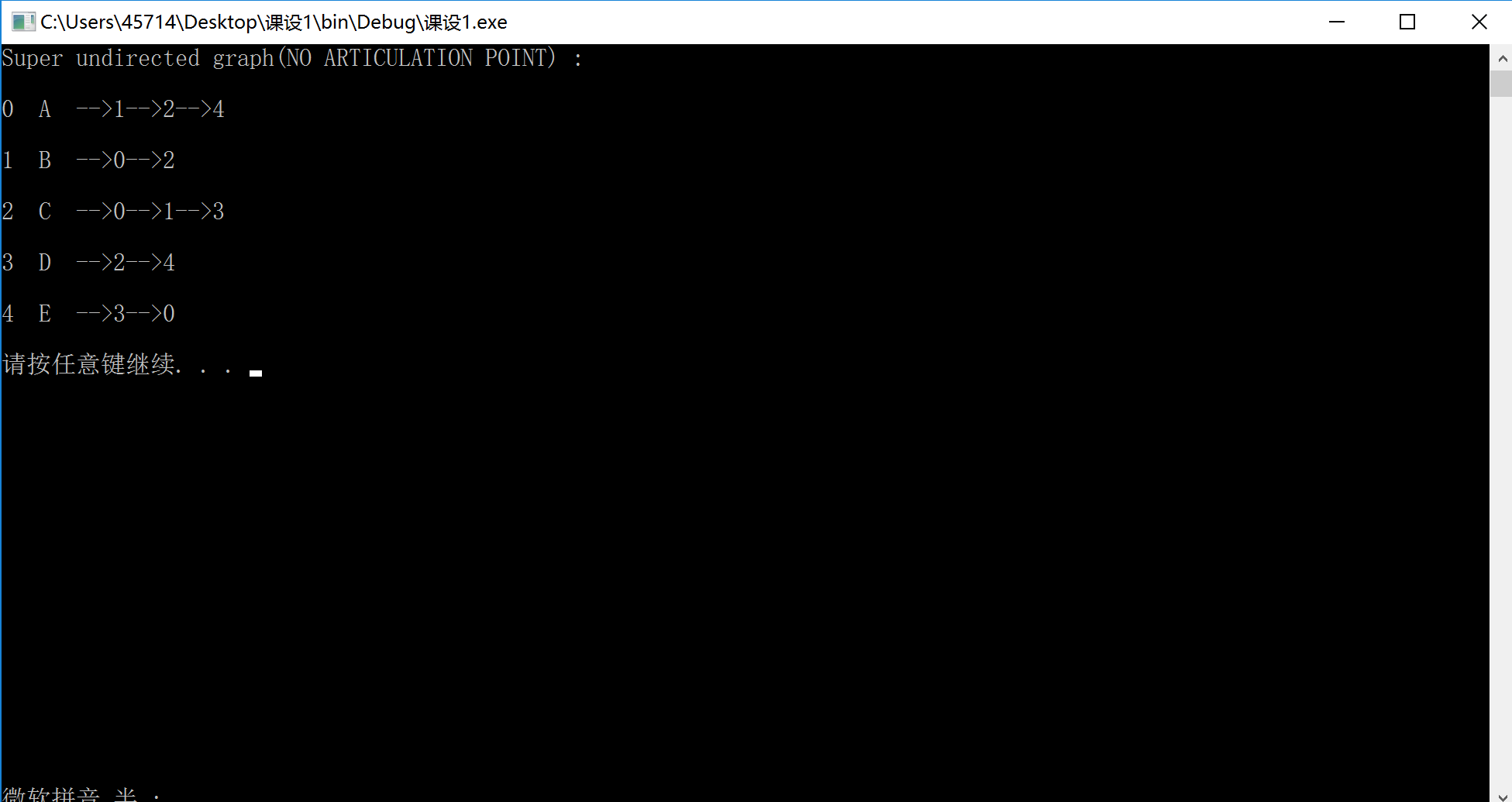


从外部文件读入数据



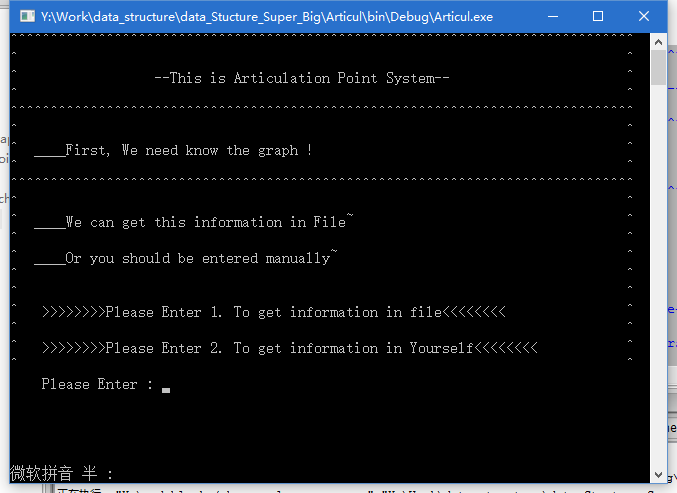
输出关节点

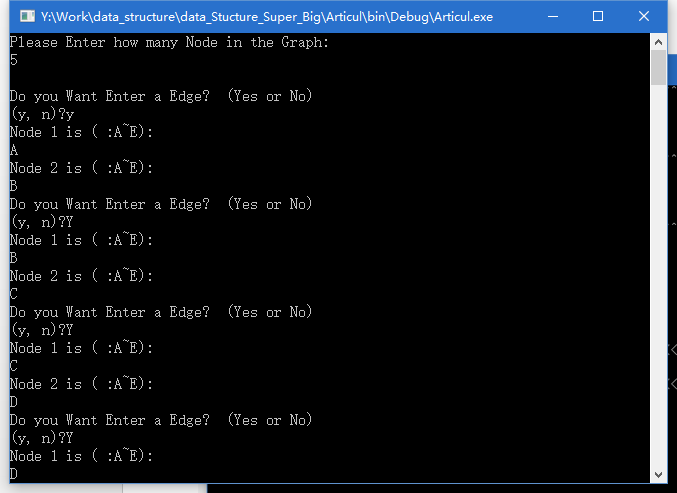


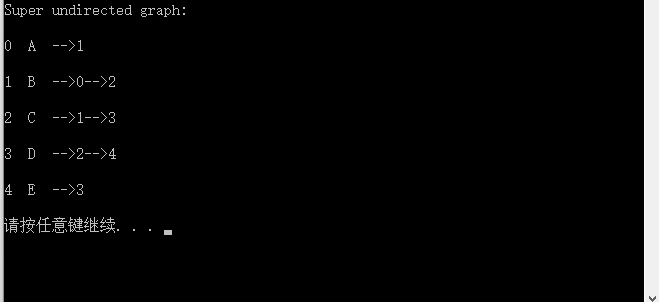


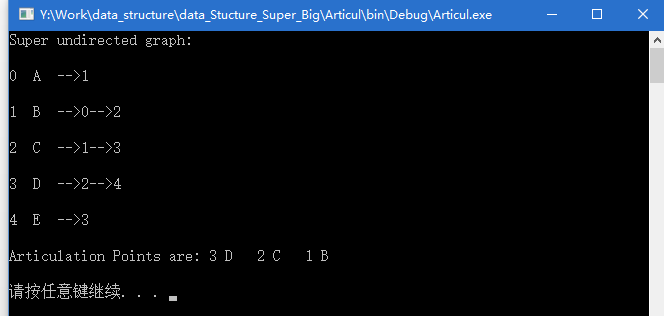
修正后

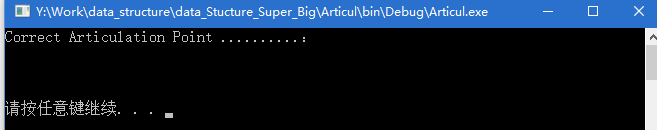
5.2 组装与系统测试

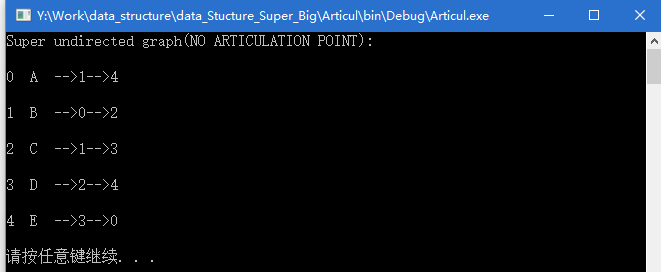








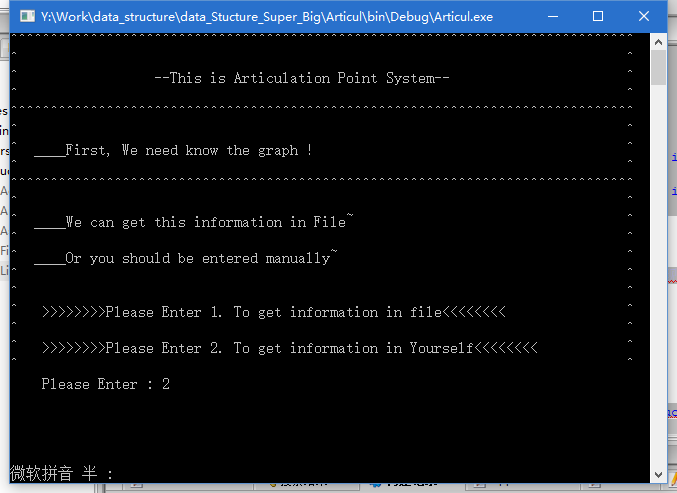




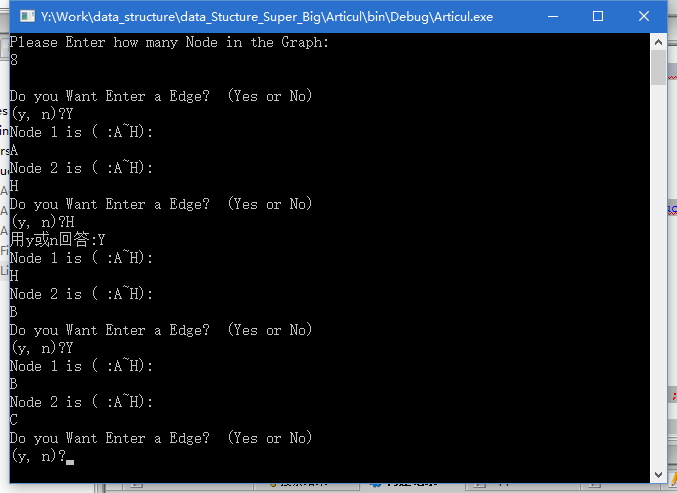
5.3 系统运行



打开程序



输入数据



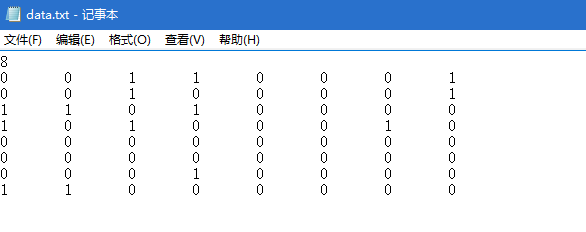
显示图之后求出关节点



修正



程序成功运行



数据存储。

# 6 课题总结

6.1 课题评价

以深度遍历求无向图的关节点，此问题看上去有些复杂难以理解，在刚开始构思之初，也因为对递归的运用不是特别熟练，对标记数组的不会使用导致编程遇到层层阻挠；

然而研究到后来，发现其实并不是特别困难，只要对递归的思路清晰，对标志数组熟练运用，便可以实现很多难以攻破的问题；

在关节点的修正问题上也遇到很大困难，此问题优化空间极大，想找到最优的方案实属不易，需要对函数之间关系，每个步骤，每个数据的熟练掌握，很费脑筋；

因为用到了递归，空间复杂度不是特别小，但也不是大到无法承受，且运行时间极短，一切都可以接受。

6.2 团队协作

我们团队的成员各司其职，每个人都尽自己所能地为整个团队，为整个项目付出着，在实现程序的日子里，我们相互帮助；在一个项目中，每个人能力强固然重要，但是对于一个团队，心往一处想，劲往一处使更是重中之重！我们小组很好的做到了这一点，在我们互相的协助之下，我们都最大限度地发挥了自己的能力。努力地将整个项目，整个程序做到我们所能做的最好！

6.3 个人设计小结（按组员分工）

6.3.1 于梓元小结

转眼，《数据结构》的课程设计接近尾声了，在这次实战中感觉到自己的在编程中的各个不足，只看书不行，只编也不行，而是都要结合起来，才能提高你的编程能力。总结起来，主要是牢固掌握自己的基础知识，没有C语言的知识，数据结构怎么可能学好，C语言在大一学的虽然有些遗忘，但慢慢查查资料，问问老师同学，然后慢慢有思路。所以我告诫自己在学好英语同时一定要牢固掌握专业基础知识。还有要有耐心，细心，严谨的态度。在编程过程中由于少了分号的小错误而导致整个程序运行不起来，不够认真细微，给自己带来了许多麻烦。编程是一件十分严谨的事情，容不得马虎。所以在今后自己一定要培养严谨，细心的态度，这不仅是对程序设计还是做任何事情都应如此。再次充分认识到《数据结构》的重要性，它可以给我们大纲和思想。

6.3.2 张伟颖小结

此次实验我主要负责将关节点转化为非关节点。实验初期阶段试图找其连通分量和其中的结点将其连接即可，后发现其可操作性较差，效率较低，于是对实验不断进行改善，最终通过队友的启发找到了适合于整个实验过程的算法设计，即运用深度优先遍历的算法对图进行改变，提高了运行效率的同时节省了空间存储量。通过此次实验，我对深度优先遍历的算法有了更深的了解和体会，同时认识到了团队合作的重要性。A类题目与B类题目相比较为基础，但同时更考验个人以及团队的编程基本功，实验初始阶段，我们根据每个人所擅长以及掌握的不同部分进行合理分工，遇到问题共同解决，这才使得实验能够顺利进行。在以后的实验中，应充分借鉴之前实验所积累的经验教训，以便更高效的实现算法设计。

6.3.3 孟霞小结

图，是数据结构整门课程中比较重要也比较复杂的部分，正是在这次实验设计之中，我结合书本、课件，把以往模模糊糊，不甚清楚的知识搞懂，非常有成就感。在这次实验设计中，我深深认识到了合作的重要性，这个实验的整个思路都是在我们整个小组的讨论中逐渐明晰、敲定的，这三周，我和小伙伴们互为师友，共同进步，当思路进入死胡同时，是我们之间的互相帮助和互相激励让我们成功渡过困难。此外，也要感谢老师和学长的耐心指导，为我们提供了很大的帮助，感谢小伙伴们和老师3周的陪伴。这次实验能顺利通过，除了高兴外，感激是占据我心情的最大部分，我想这也来源于3周来我性格的一种成长。