**《算法与数据结构》**

**实 验 指 导 手 册**

# 北京邮电大学软件学院

# 2020-2021学年第1学期实验报告

**课程名称： 算法与数据结构**

**实验名称： 实验五、图及其应用**

**实验完成人：**

**姓名：**\_\_\_王宇涵\_\_\_\_\_**学号：**\_\_\_\_2020211730\_\_\_\_**成绩：**\_\_\_\_\_\_\_\_

**指导教师：**\_\_\_\_\_\_\_**\_\_贾红娓** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**日 期： 2021 年 12 月 1 日**

1. **实验目的**

树和图是两种应用极为广泛的数据结构，也是这门课程的重点。它们的特点在于非线性。本章实验继续突出了数据结构加操作的程序设计观点，但根据这种结构的非线性特点，将操作进一步集中在遍历操作上，因为遍历操作是其它众多操作的基础。遍历逻辑的（或符号形式的）结构，访问动作可是任何操作。本次实验希望帮助学生熟悉各种存储结构的特征，以及如何应用图结构解决具体问题（即原理与应用的结合）。

1. **实验内容**
   * + 1. **图遍历的演示**

**[问题描述]**

　　很多涉及图上操作的算法都是以图的遍历操作为基础的。试写一个程序，演示无向图的遍历操作。

**[基本要求]**

　　以邻接表为存储结构，实现连通无向图的深度优先和广度优先遍历。以用户指定的结点为起点，分别输出每种遍历下的结点访问序列和相应生成树的边集。

**[测试数据]**

　　由学生依据软件工程的测试技术自己确定。注意测试边界数据，如单个结点。

**[实现提示]**

　　设图的结点不超过30个，每个结点用一个编号表示（如果一个图有n个结点，则它们的编号分别为1,2,…,n）。通过输入图的全部边输入一个图，每条边为一个数对，可以对边的输入顺序做出某种限制。注意，生成树的边是有向边，端点顺序不能颠倒。

**选做内容**

1. 借助于栈类型（自己定义和实现）将深度优先遍历用非递归算法实现。
2. 实现有向图的遍历操作。
3. **实验环境**

VC6.0

1. **实验过程和实验结果**

**1.图遍历的演示**

**1.1问题分析**

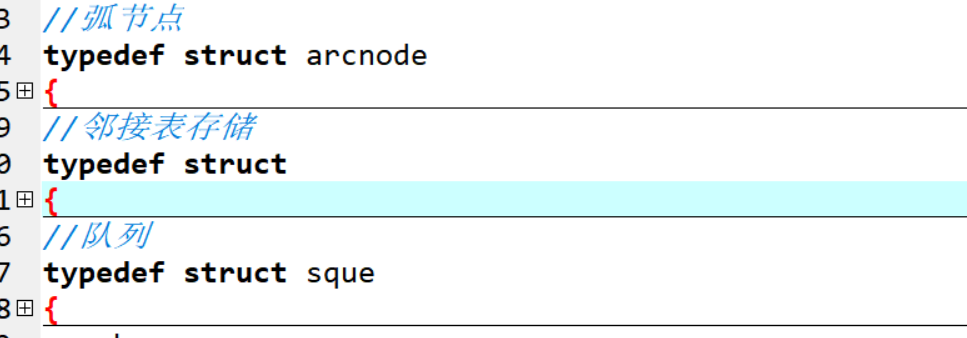
考察的是图的创建和遍历,以及在遍历的时候记录顺序,并且由这些结点的遍历顺序创建生成树。以邻接表为存储结构，实现连通无向图的深度优先和广度优先遍历。

1. 邻接表：类似于树的孩子链表。顺序存储结构与链式存储相结合。
2. 深度优先：类似于树的先序遍历。从顶点搜索，访问顶点v1，选择未曾访问的邻接点v2。从v2出发，选择未曾访问的邻接点v4，以此类推，并有回溯。
3. 广度优先遍历：类似于树的层序遍历。访问一个点，将该店未被访问的邻接点依次全部访问，记录最早被访问的点，从这个点出发，该点未被访问的邻接点依次全部访问。循环直到最终都被访问过。

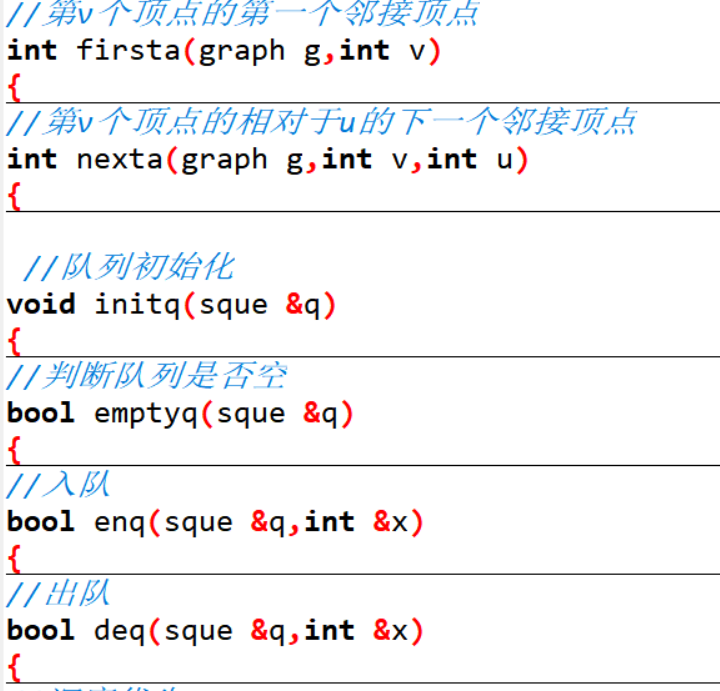
**1.2设计方案**

1. 创建节点的结构体
2. 创建图的函数
3. 创建队列结点（bfs）
4. 队列相关函数
5. 有邻接表查找第一个邻接结点
6. 对于u的下一个邻接顶点
7. Bfs
8. Dfs
9. 主函数

结点：

****

**函数：**

****

**1.3算法**

**1）图结点**

//弧节点

typedef struct arcnode

{

int adjvex;//顶点位置

struct arcnode \*next;//指向下一条弧

}arcnode;

//邻接表存储

typedef struct

{

arcnode\* adjlist[MAXSIZE];//指向第一条依附该顶点的

int vexnum, arcnum; //结点数，边的个数

}graph;

1. **图功能实现**

//构造邻接表结构的图g

void createg(graph &g)

{

int beg,end;

arcnode \*s;

//初始化

for(int i=1;i<=g.vexnum;i++)

{

g.adjlist[i]=NULL;

}

if (g.arcnum == 0)

{

cout << "没有边" << endl;

}

else

{

for(int i=1;i<=g.arcnum;i++)

{

cin >> beg >> end; //输入弧的起点和终点

s = (arcnode \*)malloc(sizeof(arcnode));

s->next = g.adjlist[end];

s->adjvex = beg;

g.adjlist[end] = s;

//无向图，插入两次

s = (arcnode \*)malloc(sizeof(arcnode)); //生成弧结点

s->next = g.adjlist[beg]; //插入到邻接表中

s->adjvex = end;

g.adjlist[beg] = s;

}

/\*s = (arcnode \*)malloc(sizeof(arcnode));

for(int i=1;i<=g.vexnum;i++)

{

s=g.adjlist[i];

cout<<i<<":";

while(s)

{

cout<<s->adjvex<<" ";

s=s->next;

}

cout<<endl;

}\*/

}

}

//第v个顶点的第一个邻接顶点

int firsta(graph g,int v)

{

if(!g.adjlist[v])

{

return 0;

}

else

{

return (g.adjlist[v]->adjvex);

}

}

//第v个顶点的相对于u的下一个邻接顶点

int nexta(graph g,int v,int u)

{

arcnode\* p=g.adjlist[v];

while(p->adjvex!=u)

{

p=p->next;

}

if(p->next==NULL)//可能是最后一个结点

{

return 0;

}

else

{

return(p->next->adjvex);

}

}

1. **队列结构体**

//队列

typedef struct sque

{

int data[MAXSIZE];

int front,rear;

}sque;

1. **队列相关功能**

//队列初始化

void initq(sque &q)

{

q.rear=q.front=0;

}

//判断队列是否空

bool emptyq(sque &q)

{

if(q.rear==q.front)

return true;

else

return false;

}

//入队

bool enq(sque &q,int &x)

{

if((q.rear+1)% MAXSIZE==q.front)//队满报错

{

return false;

}

q.data[q.rear]=x;//新元素入队尾

q.rear=(q.rear+1)%MAXSIZE;//队尾指针+1取模

return true;

}

//出队

bool deq(sque &q,int &x)

{

if(q.rear==q.front)//队空报错

{

return false;

}

x=q.data[q.front];

q.front=(q.front+1)%MAXSIZE;//队头指针+1取模

return true;

}

1. **Dfs算法（防止图为非连通图）**

//深度优先

void dfs(graph g,int v)

{

cout<<v<<" ";//访问初始v

visited[v]=true;//

for(int w=firsta(g,v);w>0;w=nexta(g,v,w))

{

//cout<<"w"<<w<<endl;

if(!visited[w])//w为v的尚未访问结点

{

//cout<<w<<" ";

dfs(g,w);

}

}

}

//对g深度优先

void dfstra(graph g)

{

cout<<"dfs 结果："<<endl;

//标记初始化

memset(visited,false,sizeof(visited));

for(int i=1;i<=g.vexnum;i++)//从0开始遍历

{

if(!visited[i])//对每个连通分量分别用bfs

{

dfs(g,i);

}

}

cout<<endl;

}

1. **Bfs算法**

//对g广度优先

void bfstra(graph g)

{

cout<<"bfs 结果："<<endl;

memset(visited,false,sizeof(visited));//初始化

initq(q);

for(int i=1;i<=g.vexnum;i++)

{

if(!visited[i])

{

visited[i]=true;

cout<<i<<" ";

enq(q,i);//和i相连的放入队列

while(!emptyq(q))//队列非空

{

int u;

deq(q,u);//将队头元素出队列

//对u的尚未访问的邻接顶点访问并入队

for(int w=firsta(g,u);w>0;w=nexta(g,u,w))

{

if(!visited[w])//w为v的尚未访问结点

{

visited[w]=true;

cout<<w<<" ";

enq(q,w);

}

}

}

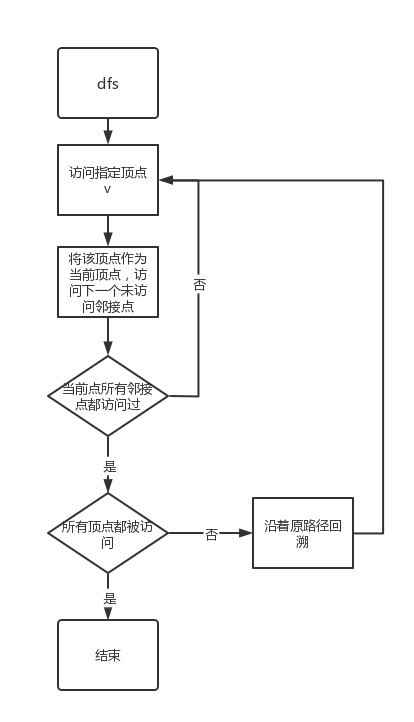
}

}

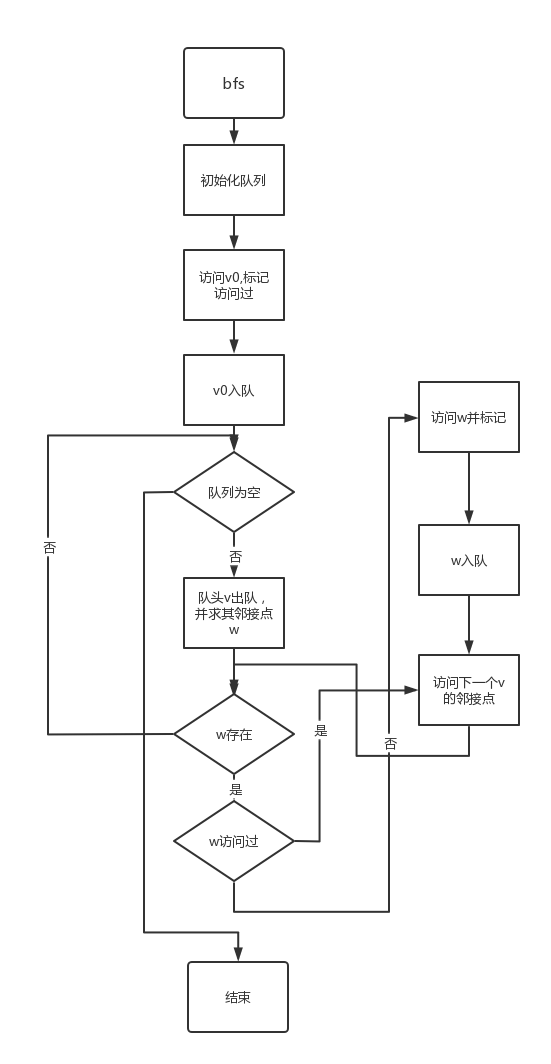
cout<<endl;

}**1.4设计图**

1)dfs



1. Bfs



**1.5程序**

#include<cstdio>

#include<cstdlib>

#include<cstring>

#include<iostream>

#include<cmath>

#define MAXSIZE 10086

using namespace std;

//8 9

//1 2 1 3 2 4 2 5 5 8 4 8 3 7 3 6 6 7

//1 0

//5 7

//1 2 1 5 2 5 2 3 2 4 3 4 4 5

//弧节点

typedef struct arcnode

{

int adjvex;//顶点位置

struct arcnode \*next;//指向下一条弧

}arcnode;

//邻接表存储

typedef struct

{

arcnode\* adjlist[MAXSIZE];//指向第一条依附该顶点的

int vexnum, arcnum; //结点数，边的个数

}graph;

//队列

typedef struct sque

{

int data[MAXSIZE];

int front,rear;

}sque;

graph g;

sque q;

bool visited[MAXSIZE];//访问标记数组

//构造邻接表结构的图g

void createg(graph &g)

{

int beg,end;

arcnode \*s;

//初始化

for(int i=1;i<=g.vexnum;i++)

{

g.adjlist[i]=NULL;

}

if (g.arcnum == 0)

{

cout << "没有边" << endl;

}

else

{

for(int i=1;i<=g.arcnum;i++)

{

cin >> beg >> end; //输入弧的起点和终点

s = (arcnode \*)malloc(sizeof(arcnode));

s->next = g.adjlist[end];

s->adjvex = beg;

g.adjlist[end] = s;

//无向图，插入两次

s = (arcnode \*)malloc(sizeof(arcnode)); //生成弧结点

s->next = g.adjlist[beg]; //插入到邻接表中

s->adjvex = end;

g.adjlist[beg] = s;

}

/\*s = (arcnode \*)malloc(sizeof(arcnode));

for(int i=1;i<=g.vexnum;i++)

{

s=g.adjlist[i];

cout<<i<<":";

while(s)

{

cout<<s->adjvex<<" ";

s=s->next;

}

cout<<endl;

}\*/

}

}

//第v个顶点的第一个邻接顶点

int firsta(graph g,int v)

{

if(!g.adjlist[v])

{

return 0;

}

else

{

return (g.adjlist[v]->adjvex);

}

}

//第v个顶点的相对于u的下一个邻接顶点

int nexta(graph g,int v,int u)

{

arcnode\* p=g.adjlist[v];

while(p->adjvex!=u)

{

p=p->next;

}

if(p->next==NULL)//可能是最后一个结点

{

return 0;

}

else

{

return(p->next->adjvex);

}

}

//队列初始化

void initq(sque &q)

{

q.rear=q.front=0;

}

//判断队列是否空

bool emptyq(sque &q)

{

if(q.rear==q.front)

return true;

else

return false;

}

//入队

bool enq(sque &q,int &x)

{

if((q.rear+1)% MAXSIZE==q.front)//队满报错

{

return false;

}

q.data[q.rear]=x;//新元素入队尾

q.rear=(q.rear+1)%MAXSIZE;//队尾指针+1取模

return true;

}

//出队

bool deq(sque &q,int &x)

{

if(q.rear==q.front)//队空报错

{

return false;

}

x=q.data[q.front];

q.front=(q.front+1)%MAXSIZE;//队头指针+1取模

return true;

}

//深度优先

void dfs(graph g,int v)

{

cout<<v<<" ";//访问初始v

visited[v]=true;//

for(int w=firsta(g,v);w>0;w=nexta(g,v,w))

{

//cout<<"w"<<w<<endl;

if(!visited[w])//w为v的尚未访问结点

{

//cout<<w<<" ";

dfs(g,w);

}

}

}

//对g深度优先

void dfstra(graph g)

{

cout<<"dfs 结果："<<endl;

//标记初始化

memset(visited,false,sizeof(visited));

for(int i=1;i<=g.vexnum;i++)//从0开始遍历

{

if(!visited[i])//对每个连通分量分别用bfs

{

dfs(g,i);

}

}

cout<<endl;

}

//对g广度优先

void bfstra(graph g)

{

cout<<"bfs 结果："<<endl;

memset(visited,false,sizeof(visited));//初始化

initq(q);

for(int i=1;i<=g.vexnum;i++)

{

if(!visited[i])

{

visited[i]=true;

cout<<i<<" ";

enq(q,i);//和i相连的放入队列

while(!emptyq(q))//队列非空

{

int u;

deq(q,u);//将队头元素出队列

//对u的尚未访问的邻接顶点访问并入队

for(int w=firsta(g,u);w>0;w=nexta(g,u,w))

{

if(!visited[w])//w为v的尚未访问结点

{

visited[w]=true;

cout<<w<<" ";

enq(q,w);

}

}

}

}

}

cout<<endl;

}

int main()

{

cout << "输入结点数和边数:" << endl;

cin >> g.vexnum >> g.arcnum;

cout << "输入边:" << endl;

createg(g);

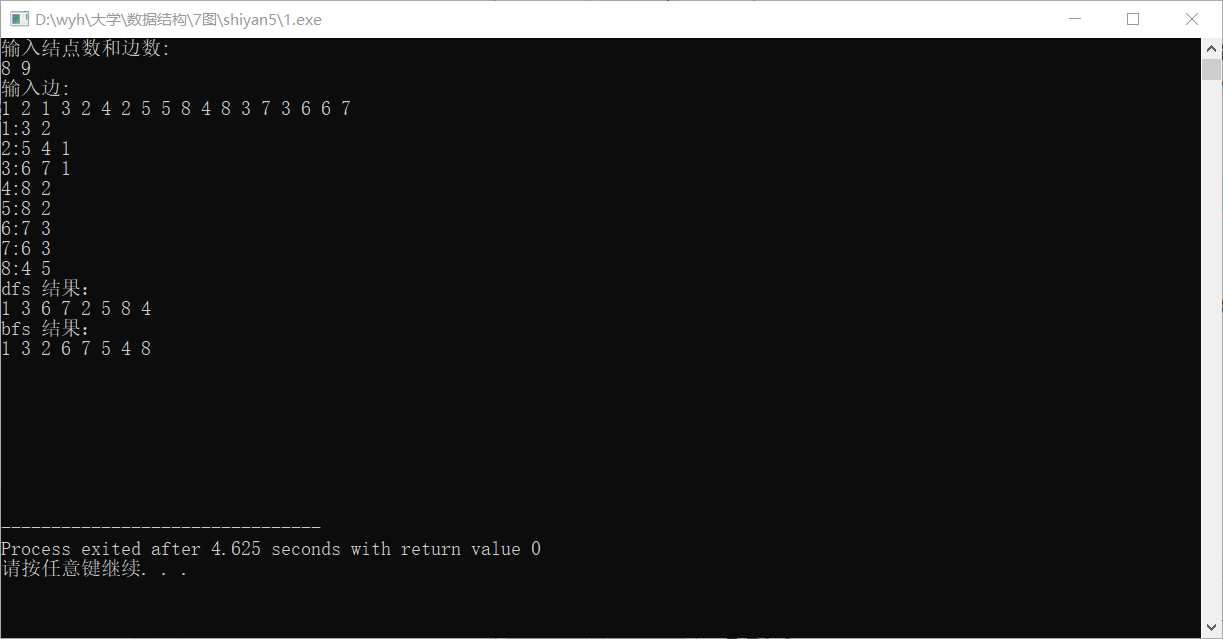
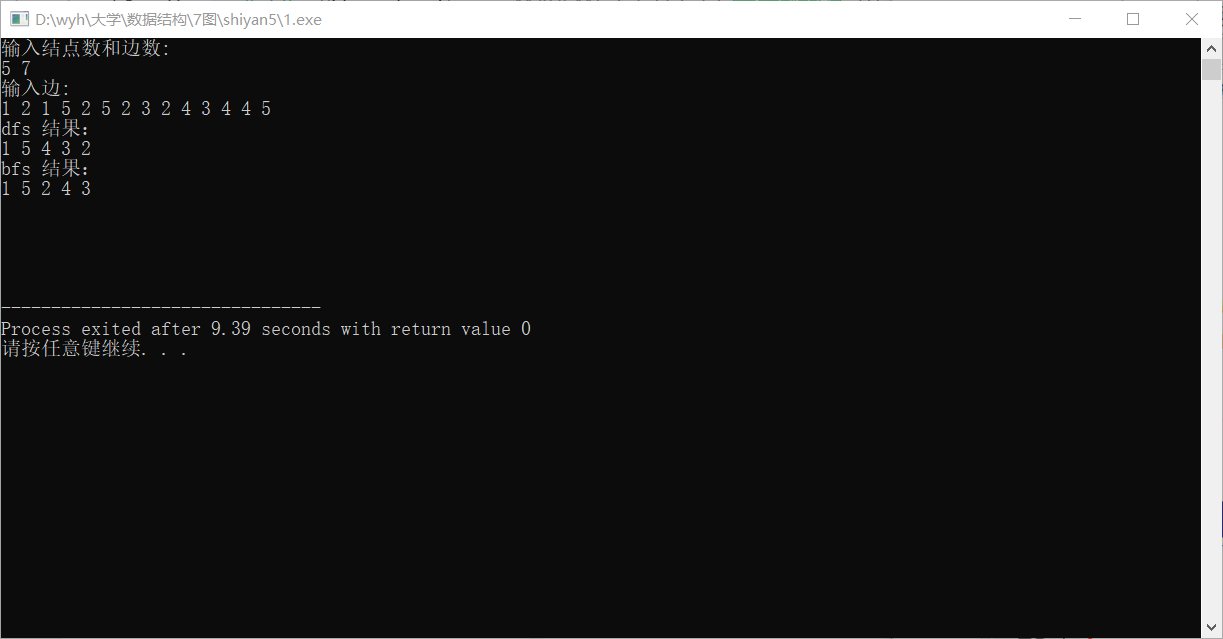
dfstra(g);

bfstra(g);

return 0;

}

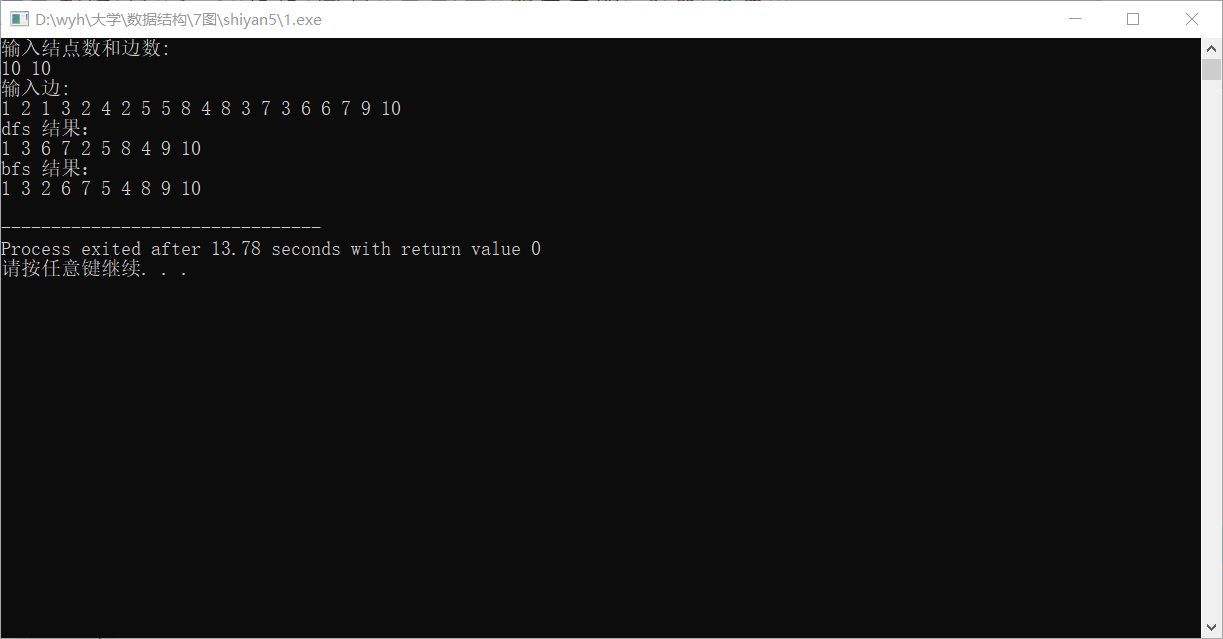
**1.6调试过程截图**

****

**查看邻接表的创建**

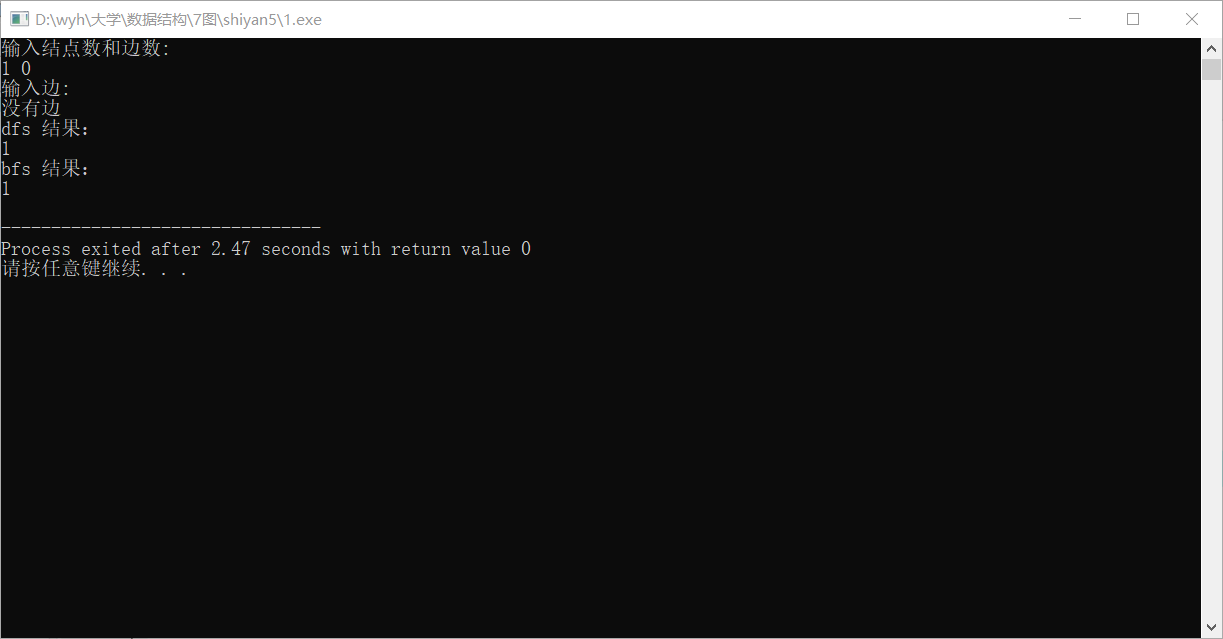
**1.7结果截图**

1. **非连通图**

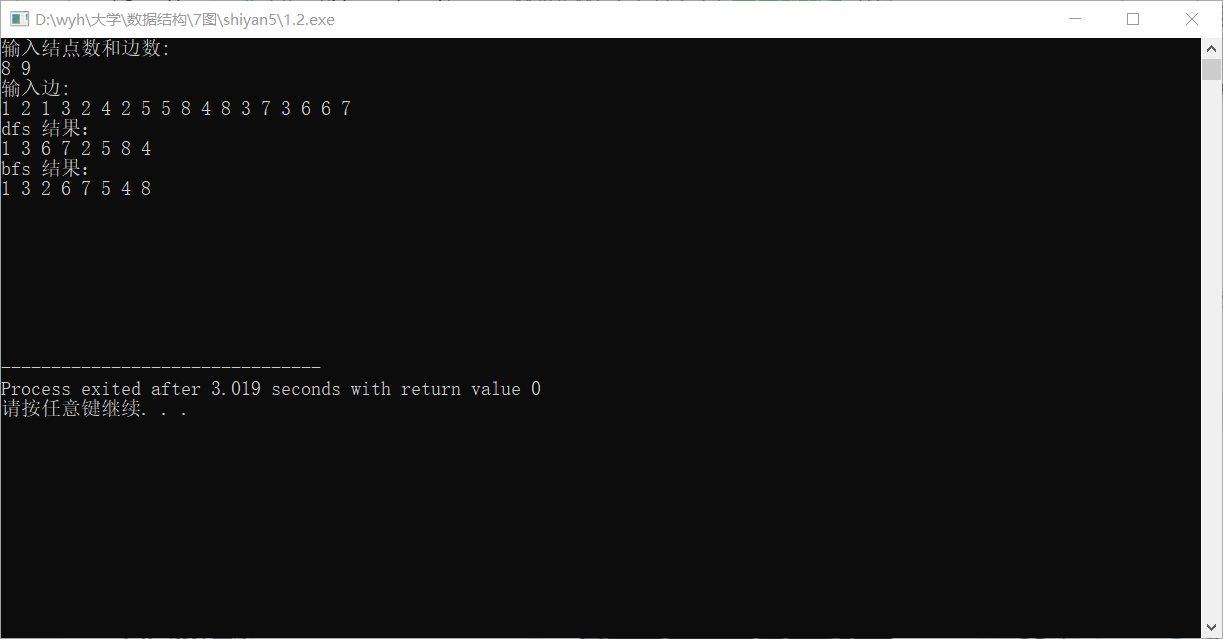
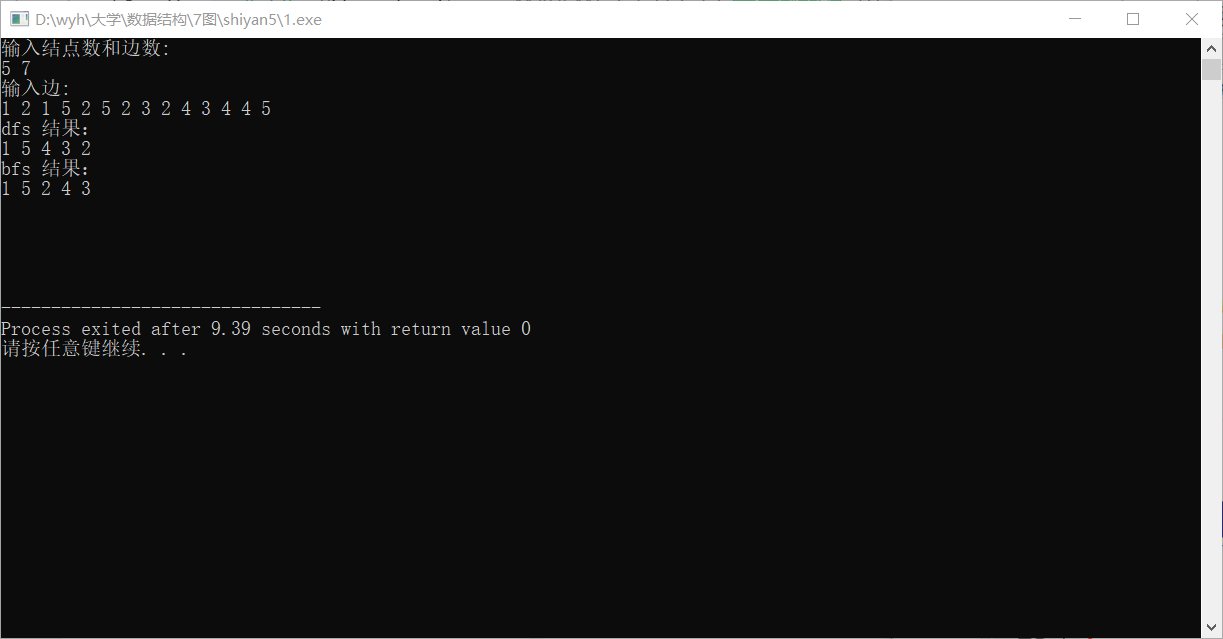
****

****

1. **没有边**

****

1. **连通图**

****

1. **选做**

**2.1问题分析**

1. 有向图：与无向图不同的是，创建的时候只要在开始的结点创捷一条链就行了
2. 借助于栈类型（自己定义和实现）将深度优先遍历用非递归算法实现。需要增加栈的操作

**2.2设计方案**

**1.仅需要改变创建图的函数**

无向图：

s = (arcnode \*)malloc(sizeof(arcnode));

s->next = g.adjlist[end];

s->adjvex = beg;

g.adjlist[end] = s;

//无向图，插入两次

s = (arcnode \*)malloc(sizeof(arcnode)); //生成弧结点

s->next = g.adjlist[beg]; //插入到邻接表中

s->adjvex = end;

g.adjlist[beg] = s;

有向图：

//有向图，插入1次

s = (arcnode \*)malloc(sizeof(arcnode)); //生成弧结点

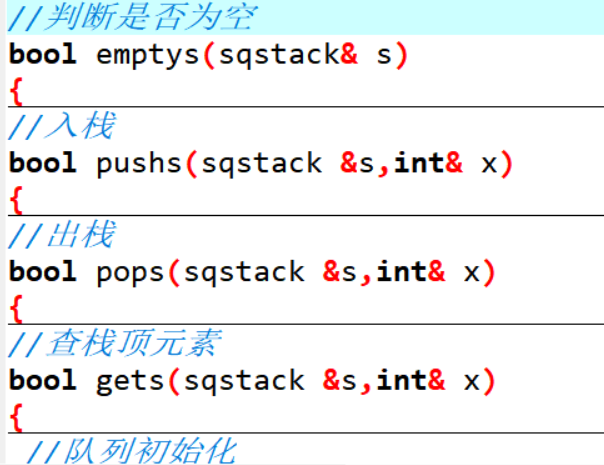
s->next = g.adjlist[beg]; //插入到邻接表中

s->adjvex = end;

g.adjlist[beg] = s;

}

1. **栈操作**

****

**3.dfs递归用栈代替**

**2.3算法**

**1.//构造邻接表结构的图g**

**void createg(graph &g)**

**{**

**int beg,end;**

**arcnode \*s;**

**//初始化**

**for(int i=1;i<=g.vexnum;i++)**

**{**

**g.adjlist[i]=NULL;**

**}**

**if (g.arcnum == 0)**

**{**

**cout << "没有边" << endl;**

**}**

**else**

**{**

**for(int i=1;i<=g.arcnum;i++)**

**{**

**cin >> beg >> end; //输入弧的起点和终点**

**//有向图，插入1次**

**s = (arcnode \*)malloc(sizeof(arcnode)); //生成弧结点**

**s->next = g.adjlist[beg]; //插入到邻接表中**

**s->adjvex = end;**

**g.adjlist[beg] = s;**

**}**

**s = (arcnode \*)malloc(sizeof(arcnode));**

**for(int i=1;i<=g.vexnum;i++)**

**{**

**s=g.adjlist[i];**

**cout<<i<<":";**

**while(s)**

**{**

**cout<<s->adjvex<<" ";**

**s=s->next;**

**}**

**cout<<endl;**

**}**

**}**

**}**

**2.栈操作**

**//初始化**

**void inits(sqstack &s)**

**{**

**s.top=-1;**

**}**

**//判断是否为空**

**bool emptys(sqstack& s)**

**{**

**if(s.top==-1)**

**return true;**

**else**

**return false;**

**}**

**//入栈**

**bool pushs(sqstack &s,int& x)**

**{**

**if(s.top==MAXSIZE-1)//栈满报错**

**{**

**return false;**

**}**

**s.top++;//指针先加**

**s.data[s.top]=x;//新元素入栈**

**return true;**

**}**

**//出栈**

**bool pops(sqstack &s,int& x)**

**{**

**if(s.top==-1)//栈空报错**

**{**

**return false;**

**}**

**x=s.data[s.top];//栈顶元素出栈**

**s.top--;//指针再减**

**return true;**

**}**

**//查栈顶元素**

**bool gets(sqstack &s,int& x)**

**{**

**if(s.top==-1)//栈空报错**

**{**

**return false;**

**}**

**x=s.data[s.top];//栈顶元素出栈**

**return true;**

**}**

**3.dfs（非递归）**

**//对g深度优先（栈）**

**void dfstra(graph g)**

**{**

**sqstack s;**

**cout<<"dfs 结果："<<endl;**

**//标记初始化**

**memset(visited,false,sizeof(visited));**

**inits(s);**

**for(int v=1;v<=g.vexnum;v++)//从1开始遍历**

**{**

**if(!visited[v])//对每个连通分量分别用dfs**

**{**

**visited[v]=true;**

**cout<<v<<" ";**

**pushs(s,v);**

**while(!emptys(s))**

**{**

**int w,cishu=0;**

**gets(s,v);**

**for(w=firsta(g,v);w>0;w=nexta(g,v,w))**

**{**

**//cout<<"w"<<w<<endl;**

**if(!visited[w])//w为v的尚未访问结点**

**{**

**cout<<w<<" " ;**

**visited[w]=1;**

**pushs(s,w);**

**break;**

**}**

**}**

**if(w==0)**

**{**

**pops(s,v);**

**}**

**}**

**}**

**}**

**cout<<endl;**

**}**

**2.4设计图**

**同第一题**

**2.5程序**

1. **有向图：**

**#include<cstdio>**

**#include<cstdlib>**

**#include<cstring>**

**#include<iostream>**

**#include<cmath>**

**#define MAXSIZE 10086**

**using namespace std;**

**//8 9**

**//1 2 1 3 2 4 2 5 5 8 4 8 3 7 3 6 6 7**

**//1 0**

**//5 7**

**//1 2 1 5 2 5 2 3 2 4 3 4 4 5**

**//弧节点**

**typedef struct arcnode**

**{**

**int adjvex;//顶点位置**

**struct arcnode \*next;//指向下一条弧**

**}arcnode;**

**//邻接表存储**

**typedef struct**

**{**

**arcnode\* adjlist[MAXSIZE];//指向第一条依附该顶点的**

**int vexnum, arcnum; //结点数，边的个数**

**}graph;**

**//队列**

**typedef struct sque**

**{**

**int data[MAXSIZE];**

**int front,rear;**

**}sque;**

**graph g;**

**sque q;**

**bool visited[MAXSIZE];//访问标记数组**

**//构造邻接表结构的图g**

**void createg(graph &g)**

**{**

**int beg,end;**

**arcnode \*s;**

**//初始化**

**for(int i=1;i<=g.vexnum;i++)**

**{**

**g.adjlist[i]=NULL;**

**}**

**if (g.arcnum == 0)**

**{**

**cout << "没有边" << endl;**

**}**

**else**

**{**

**for(int i=1;i<=g.arcnum;i++)**

**{**

**cin >> beg >> end; //输入弧的起点和终点**

**//有向图，插入1次**

**s = (arcnode \*)malloc(sizeof(arcnode)); //生成弧结点**

**s->next = g.adjlist[beg]; //插入到邻接表中**

**s->adjvex = end;**

**g.adjlist[beg] = s;**

**}**

**s = (arcnode \*)malloc(sizeof(arcnode));**

**for(int i=1;i<=g.vexnum;i++)**

**{**

**s=g.adjlist[i];**

**cout<<i<<":";**

**while(s)**

**{**

**cout<<s->adjvex<<" ";**

**s=s->next;**

**}**

**cout<<endl;**

**}**

**}**

**}**

**//第v个顶点的第一个邻接顶点**

**int firsta(graph g,int v)**

**{**

**if(!g.adjlist[v])**

**{**

**return 0;**

**}**

**else**

**{**

**return (g.adjlist[v]->adjvex);**

**}**

**}**

**//第v个顶点的相对于u的下一个邻接顶点**

**int nexta(graph g,int v,int u)**

**{**

**arcnode\* p=g.adjlist[v];**

**while(p->adjvex!=u)**

**{**

**p=p->next;**

**}**

**if(p->next==NULL)//可能是最后一个结点**

**{**

**return 0;**

**}**

**else**

**{**

**return(p->next->adjvex);**

**}**

**}**

**//深度优先**

**void dfs(graph g,int v)**

**{**

**cout<<v<<" ";//访问初始v**

**visited[v]=true;//**

**for(int w=firsta(g,v);w>0;w=nexta(g,v,w))**

**{**

**if(!visited[w])//w为v的尚未访问结点**

**{**

**dfs(g,w);**

**}**

**}**

**}**

**//队列初始化**

**void initq(sque &q)**

**{**

**q.rear=q.front=0;**

**}**

**//判断队列是否空**

**bool emptyq(sque &q)**

**{**

**if(q.rear==q.front)**

**return true;**

**else**

**return false;**

**}**

**//入队**

**bool enq(sque &q,int &x)**

**{**

**if((q.rear+1)% MAXSIZE==q.front)//队满报错**

**{**

**return false;**

**}**

**q.data[q.rear]=x;//新元素入队尾**

**q.rear=(q.rear+1)%MAXSIZE;//队尾指针+1取模**

**return true;**

**}**

**//出队**

**bool deq(sque &q,int &x)**

**{**

**if(q.rear==q.front)//队空报错**

**{**

**return false;**

**}**

**x=q.data[q.front];**

**q.front=(q.front+1)%MAXSIZE;//队头指针+1取模**

**return true;**

**}**

**//对g深度优先**

**void dfstra(graph g)**

**{**

**cout<<"dfs 结果："<<endl;**

**for(int i=1;i<=g.vexnum;i++)**

**{**

**visited[i]=false;//标记初始化**

**}**

**for(int i=1;i<=g.vexnum;i++)//从0开始遍历**

**{**

**if(!visited[i])//对每个连通分量分别用bfs**

**{**

**dfs(g,i);**

**}**

**}**

**cout<<endl;**

**}**

**//对g广度优先**

**void bfstra(graph g)**

**{**

**cout<<"bfs 结果："<<endl;**

**memset(visited,false,sizeof(visited));//初始化**

**initq(q);**

**for(int i=1;i<=g.vexnum;i++)**

**{**

**if(!visited[i])**

**{**

**visited[i]=true;**

**cout<<i<<" ";**

**enq(q,i);//和i相连的放入队列**

**while(!emptyq(q))//队列非空**

**{**

**int u;**

**deq(q,u);//将队头元素出队列**

**//对u的尚未访问的邻接顶点访问并入队**

**for(int w=firsta(g,u);w>0;w=nexta(g,u,w))**

**{**

**if(!visited[w])//w为v的尚未访问结点**

**{**

**visited[w]=true;**

**cout<<w<<" ";**

**enq(q,w);**

**}**

**}**

**}**

**}**

**cout<<endl;**

**}**

**}**

**int main()**

**{**

**cout << "输入结点数和边数:" << endl;**

**cin >> g.vexnum >> g.arcnum;**

**cout << "输入边:" << endl;**

**createg(g);**

**dfstra(g);**

**bfstra(g);**

**return 0;**

**}**

1. **非递归实现：**

**#include<cstdio>**

**#include<cstdlib>**

**#include<cstring>**

**#include<iostream>**

**#include<cmath>**

**#define MAXSIZE 10086**

**using namespace std;**

**//8 9**

**//1 2 1 3 2 4 2 5 5 8 4 8 3 7 3 6 6 7**

**//1 0**

**//5 7**

**//1 2 1 5 2 5 2 3 2 4 3 4 4 5**

**//弧节点**

**typedef struct arcnode**

**{**

**int adjvex;//顶点位置**

**struct arcnode \*next;//指向下一条弧**

**}arcnode;**

**//邻接表存储**

**typedef struct**

**{**

**arcnode\* adjlist[MAXSIZE];//指向第一条依附该顶点的**

**int vexnum, arcnum; //结点数，边的个数**

**}graph;**

**//队列**

**typedef struct sque**

**{**

**int data[MAXSIZE];**

**int front,rear;**

**}sque;**

**//栈**

**typedef struct**

**{**

**int data[MAXSIZE];**

**int top;//栈顶指针**

**}sqstack;**

**graph g;**

**sque q;**

**int count[10086];**

**bool visited[MAXSIZE];//访问标记数组**

**//构造邻接表结构的图G**

**void createg(graph &g)**

**{**

**int beg,end;**

**arcnode \*s;**

**//初始化**

**for(int i=1;i<=g.vexnum;i++)**

**{**

**g.adjlist[i]=NULL;**

**}**

**if (g.arcnum == 0)**

**{**

**cout << "没有边" << endl;**

**}**

**else**

**{**

**for(int i=1;i<=g.arcnum;i++)**

**{**

**cin >> beg >> end; //输入弧的起点和终点**

**s = (arcnode \*)malloc(sizeof(arcnode));**

**s->next = g.adjlist[end];**

**s->adjvex = beg;**

**g.adjlist[end] = s;**

**//无向图，插入两次**

**s = (arcnode \*)malloc(sizeof(arcnode)); //生成弧结点**

**s->next = g.adjlist[beg]; //插入到邻接表中**

**s->adjvex = end;**

**g.adjlist[beg] = s;**

**}**

**s = (arcnode \*)malloc(sizeof(arcnode));**

**for(int i=1;i<=g.vexnum;i++)**

**{**

**s=g.adjlist[i];**

**cout<<i<<":";**

**while(s)**

**{**

**cout<<s->adjvex<<" ";**

**s=s->next;**

**count[i]++;**

**}**

**cout<<endl;**

**}**

**}**

**}**

**//第v个顶点的第一个邻接顶点**

**int firsta(graph g,int v)**

**{**

**if(!g.adjlist[v])**

**{**

**return 0;**

**}**

**else**

**{**

**return (g.adjlist[v]->adjvex);**

**}**

**}**

**//第v个顶点的相对于u的下一个邻接顶点**

**int nexta(graph g,int v,int u)**

**{**

**arcnode\* p=g.adjlist[v];**

**while(p->adjvex!=u)**

**{**

**p=p->next;**

**}**

**if(p->next==NULL)//可能是最后一个结点**

**{**

**return 0;**

**}**

**else**

**{**

**return(p->next->adjvex);**

**}**

**}**

**//初始化**

**void inits(sqstack &s)**

**{**

**s.top=-1;**

**}**

**//判断是否为空**

**bool emptys(sqstack& s)**

**{**

**if(s.top==-1)**

**return true;**

**else**

**return false;**

**}**

**//入栈**

**bool pushs(sqstack &s,int& x)**

**{**

**if(s.top==MAXSIZE-1)//栈满报错**

**{**

**return false;**

**}**

**s.top++;//指针先加**

**s.data[s.top]=x;//新元素入栈**

**return true;**

**}**

**//出栈**

**bool pops(sqstack &s,int& x)**

**{**

**if(s.top==-1)//栈空报错**

**{**

**return false;**

**}**

**x=s.data[s.top];//栈顶元素出栈**

**s.top--;//指针再减**

**return true;**

**}**

**//查栈顶元素**

**bool gets(sqstack &s,int& x)**

**{**

**if(s.top==-1)//栈空报错**

**{**

**return false;**

**}**

**x=s.data[s.top];//栈顶元素出栈**

**return true;**

**}**

**//队列初始化**

**void initq(sque &q)**

**{**

**q.rear=q.front=0;**

**}**

**//判断队列是否空**

**bool emptyq(sque &q)**

**{**

**if(q.rear==q.front)**

**return true;**

**else**

**return false;**

**}**

**//入队**

**bool enq(sque &q,int &x)**

**{**

**if((q.rear+1)% MAXSIZE==q.front)//队满报错**

**{**

**return false;**

**}**

**q.data[q.rear]=x;//新元素入队尾**

**q.rear=(q.rear+1)%MAXSIZE;//队尾指针+1取模**

**return true;**

**}**

**//出队**

**bool deq(sque &q,int &x)**

**{**

**if(q.rear==q.front)//队空报错**

**{**

**return false;**

**}**

**x=q.data[q.front];**

**q.front=(q.front+1)%MAXSIZE;//队头指针+1取模**

**return true;**

**}**

**//对g深度优先（栈）**

**void dfstra(graph g)**

**{**

**sqstack s;**

**cout<<"dfs 结果："<<endl;**

**//标记初始化**

**memset(visited,false,sizeof(visited));**

**inits(s);**

**for(int v=1;v<=g.vexnum;v++)//从1开始遍历**

**{**

**if(!visited[v])//对每个连通分量分别用dfs**

**{**

**visited[v]=true;**

**cout<<v<<" ";**

**pushs(s,v);**

**while(!emptys(s))**

**{**

**int w,cishu=0;**

**gets(s,v);**

**for(w=firsta(g,v);w>0;w=nexta(g,v,w))**

**{**

**//cout<<"w"<<w<<endl;**

**if(!visited[w])//w为v的尚未访问结点**

**{**

**cout<<w<<" " ;**

**visited[w]=1;**

**pushs(s,w);**

**break;**

**}**

**}**

**//gets(s,v);**

**if(w==0)**

**{**

**pops(s,v);**

**}**

**/\*//gets(s,v);**

**int data=nexta(g,v,w);**

**cout<<"data"<<data<<endl;**

**if(data==0)**

**{**

**pops(s,v);**

**}\*/**

**}**

**}**

**}**

**cout<<endl;**

**}**

**//对g广度优先**

**void bfstra(graph g)**

**{**

**cout<<"bfs 结果："<<endl;**

**memset(visited,false,sizeof(visited));//初始化**

**initq(q);**

**for(int i=1;i<=g.vexnum;i++)**

**{**

**if(!visited[i])**

**{**

**visited[i]=true;**

**cout<<i<<" ";**

**enq(q,i);//和i相连的放入队列**

**while(!emptyq(q))//队列非空**

**{**

**int u;**

**deq(q,u);//将队头元素出队列**

**//对u的尚未访问的邻接顶点访问并入队**

**for(int w=firsta(g,u);w>0;w=nexta(g,u,w))**

**{**

**if(!visited[w])//w为v的尚未访问结点**

**{**

**visited[w]=true;**

**cout<<w<<" ";**

**enq(q,w);**

**}**

**}**

**}**

**}**

**cout<<endl;**

**}**

**}**

**int main()**

**{**

**cout << "输入结点数和边数:" << endl;**

**cin >> g.vexnum >> g.arcnum;**

**cout << "输入边:" << endl;**

**createg(g);**

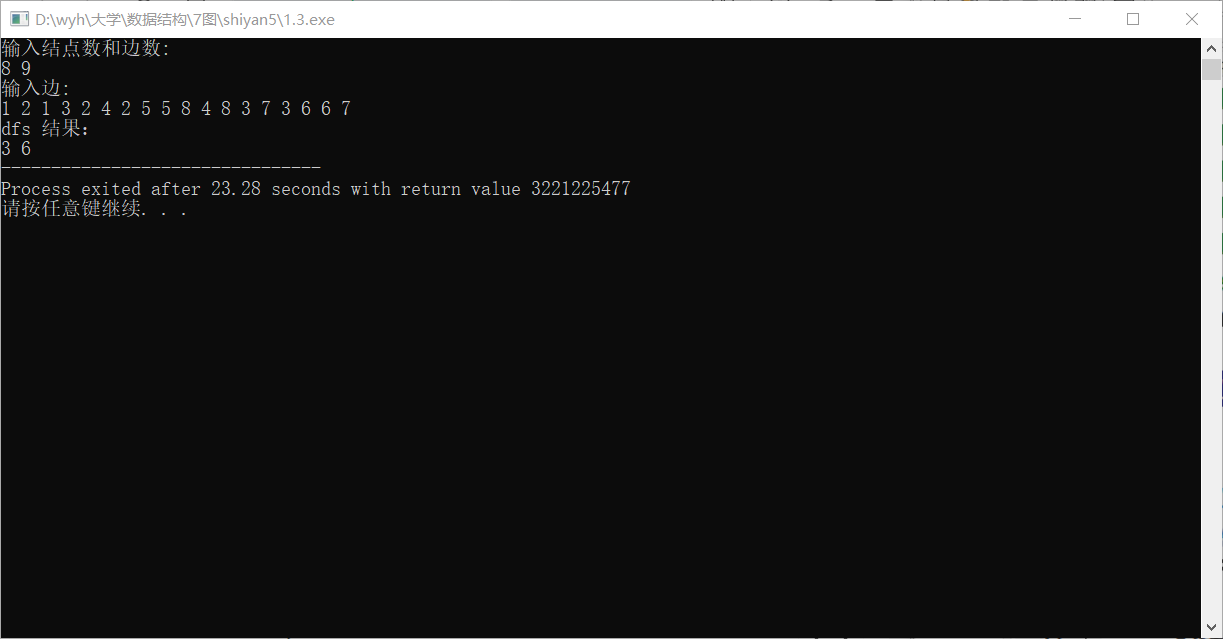
**dfstra(g);**

**bfstra(g);**

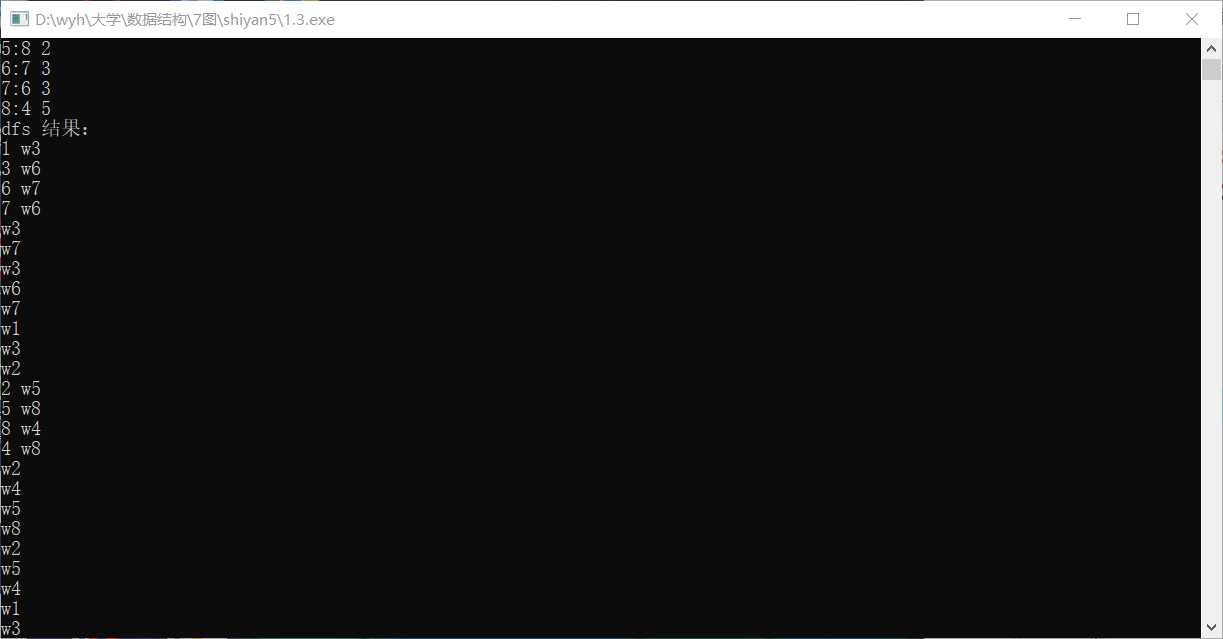
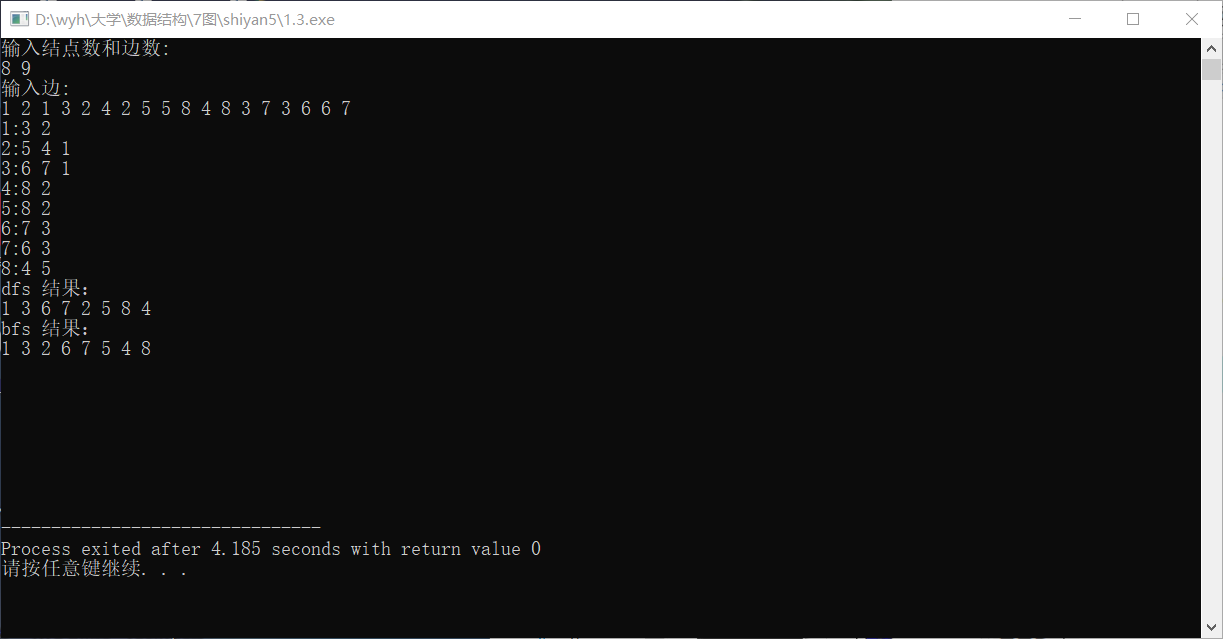
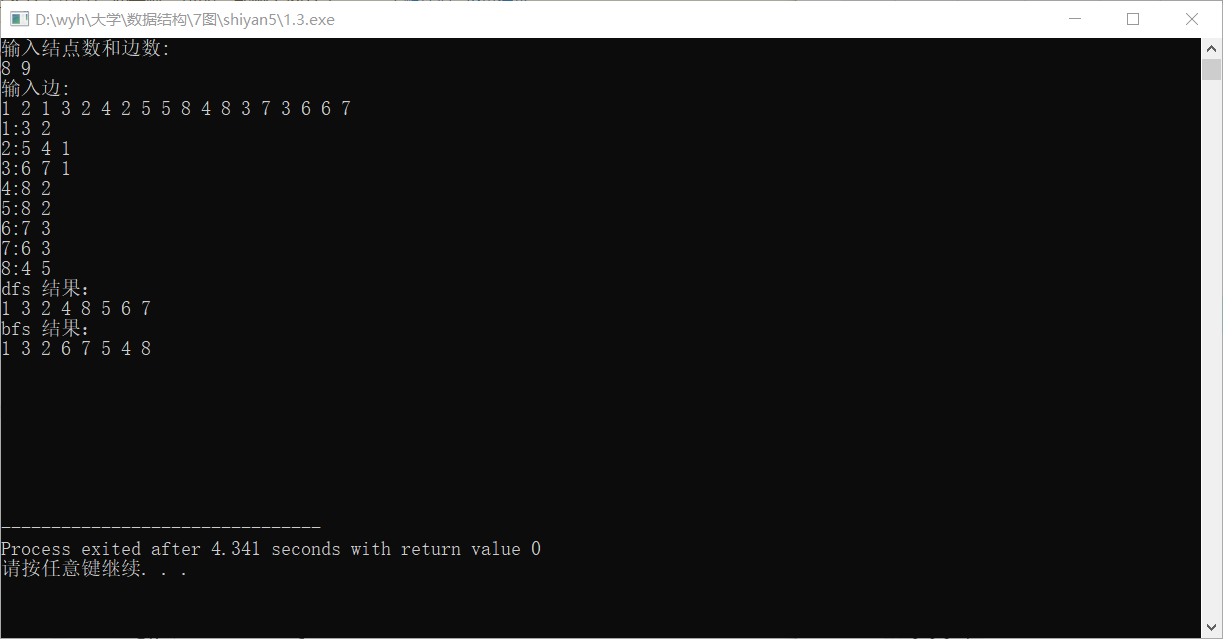
**return 0;**

**}**

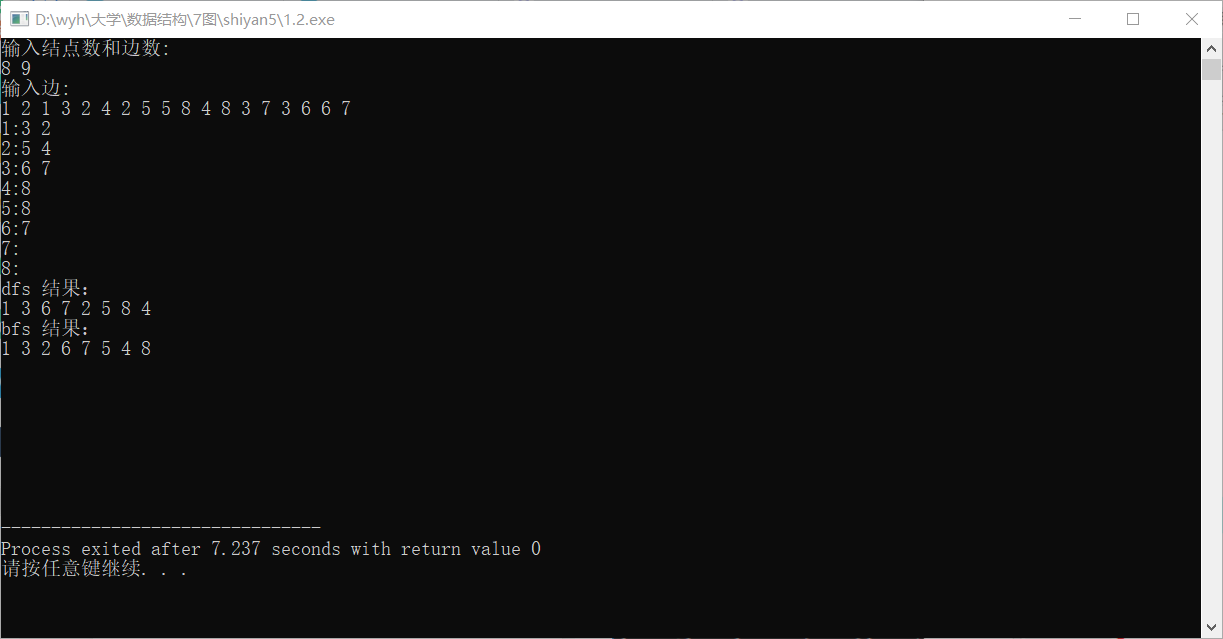
**2.6调试过程截图**

****

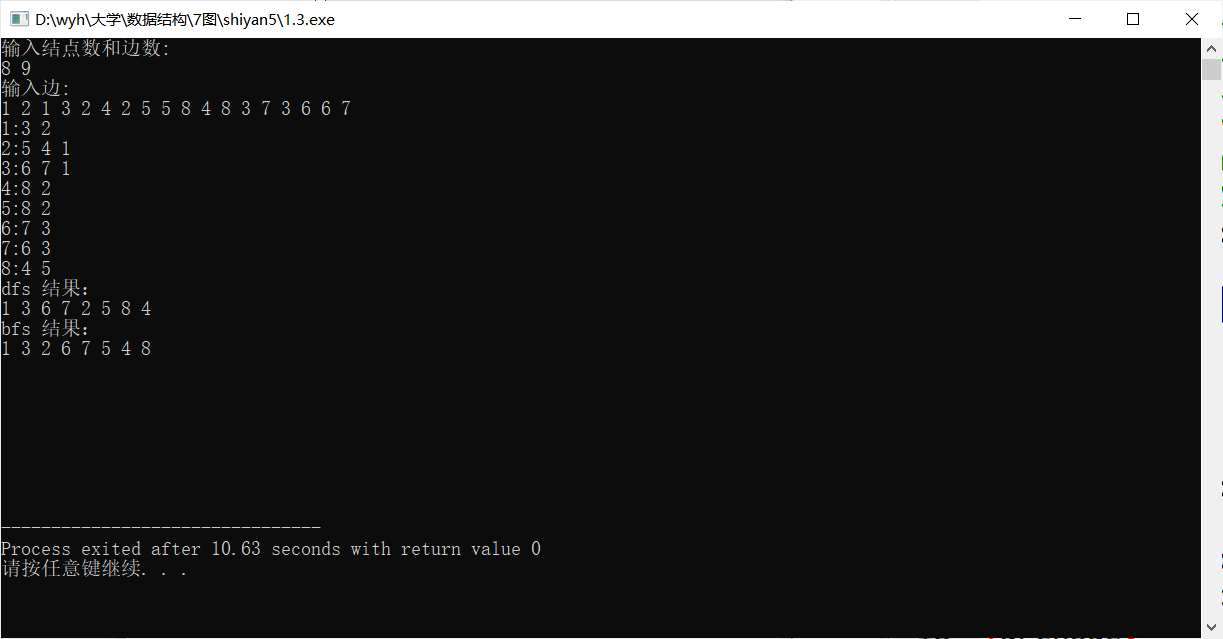
**栈的实现有错，没有遍历完**

****

**2.7运行结果截图**

****

**有向图（可以看出邻接表不同）**

****

**栈的实现**

1. **实验心得**

DFS用的是递归遍历，也就是访问一个结点，之后要访问这个结点所连接的第一个结点,之后再访问这个所连接的第一个结点所连接的第一个节点…直到一个结点没有连接的下一个结点或者所连接的下一个结点已经在这次遍历中被访问到，之后回退，寻找某一个结点所连接的第二个结点(该节点没有被访问过)，并且继续这个过程。

BFS不需要递归，但是需要一个队列，把第一个结点放入队列,之后开始一个while循环，如果队列不为空就从队列中拿出一个结点(访问该结点)，之后把这个结点所连接到的所有节点再次放入队列，重复循环，操作更复杂但是不需要递归。且实现了队列的功能，更进一步理解和掌握队列的使用。

不同的DFS、BFS算法可能会导致输出的结果不一样，但是都符合预期，因为选择同级顶点时选择方法可能不一样。从哪个点开始遍历可以通过改变for循环的初始值。

通过这次实验，我明白只有多练习上机，并且想好总体的思路，才能更好的理解掌握数据结构的精髓，从而避免基础语法错误，让代码变得更简洁高效。如此才能准确高效的解决问题。在今后的编程过程中要更注重代码整体设计的提纲记录，用更多的注释，让自己的思路更清晰。