哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称：数据结构与算法

课程类型：必修

实验项目：图型结构的建立与搜索

实验题目：图的存储结构的建立与搜索

实验日期：12 月 3 日

设计成绩	报告成绩	指导老师
		张岩

一、实验目的

图的搜索（遍历）算法是图型结构相关算法的基础，本实验要求编写程序演示无向图三种典型存储结构的建立和搜索（遍历）过程。

二、实验要求及实验环境

图的搜索（遍历）算法是图型结构相关算法的基础，本实验要求编写程序演示有向图三种典型存储结构的建立和搜索（遍历）过程。

实验要求：

1．分别实现有向图的邻接矩阵、邻接表和十字链表存储结构的建立算法，分析和比较各建立算法的时间复杂度以及存储结构的空间占用情况；（**见第五项·经验体会与不足**）

2．实现有向图的邻接矩阵、邻接表和十字链表三种存储结构的相互转换算法；

3．在上述三种存储结构上，分别实现有向图的深度优先搜索（递归和非递归）和广度优先搜索算法。并以适当的方式存储和显示相应的搜索结果（深度优先或广度优先生成森林（或生成树）、深度优先或广度优先序列和编号）；

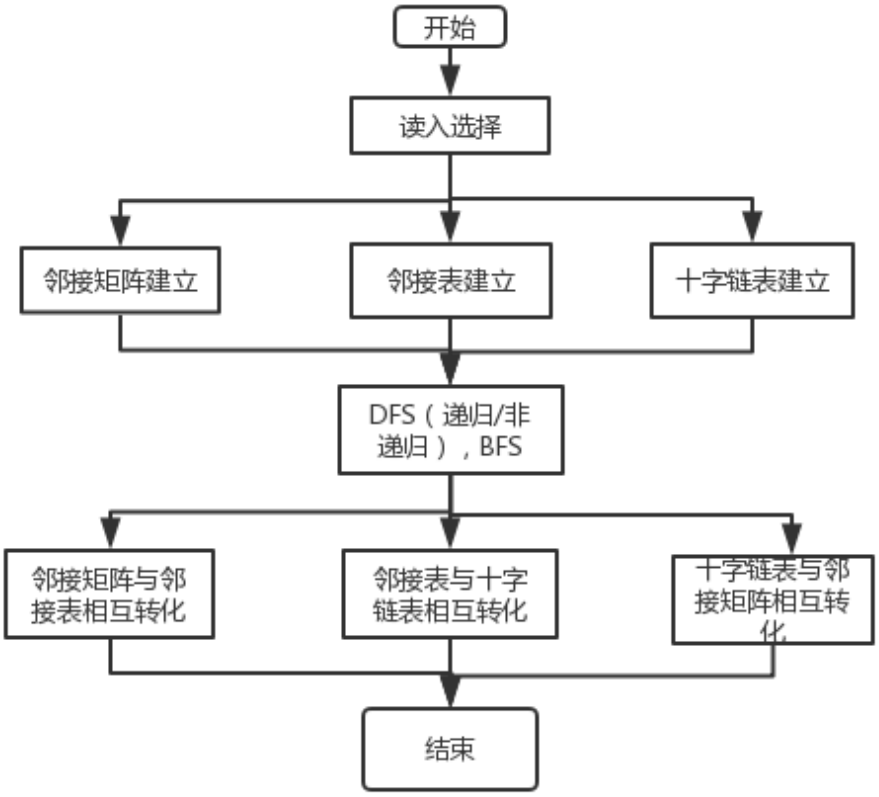
4．分析搜索算法的时间复杂度；（**见第五项·经验体会与不足**）

5．以文件形式输入图的顶点和边，并显示相应的结果。要求顶点不少于 10 个，边不少于 13 个；

6．软件功能结构安排合理，界面友好，便于使用。

三、设计思想（本程序中的用到的所有数据类型的定义，主程序的流程图及各程序模块之间的调用关系）

1．逻辑设计



2. 物理设计

①邻接矩阵结构

```
typedef struct{
    int vertex[VertexNum];//顶点数组
    int edge[VertexNum][VertexNum];//从第一个顶点到第二个顶点边的权重
    int n,e;//边和顶点的数目
}MTGraph;
```

②

```
typedef struct node{
    int adjvex;//顶点编号
    int cost;//从头结点到该结点的边的权重
    struct node *next;//下一个与头结点关联的结点
}EdgeNode;
```

```
typedef struct{
    int vertex;//头结点编号
    EdgeNode *firstedge;//与头结点相关联的第一个结点
}VertexNode;//头结点结构
typedef struct{
    VertexNode verlist[VertexNum];//头结点数组
    int n,e;//边和结点的数目
}AdjGraph;//邻接表
③
typedef struct ArcBox{
    int tailvex,headvex
//tailvex ：尾域，指示弧尾顶点在图中的位置
//headvex: 头域，指示弧头顶点在图中的位置
    struct ArcBox *hlink,*tlink;
//hlink ：链域，指向弧头相同的下一条弧
//tlink ：链域，指向弧尾相同的下一条弧
    int info;//信息
}ArcBox;//结点结构
typedef struct VexNode{
    int data;//顶点编号
    ArcBox *firstin,*firstout;
//firstin ：链域，指向以该顶点为弧头的第一个弧结点。firstout ：链域，指向以该顶点为弧尾的第一个弧结点
}VexNode;//头结点结构
typedef struct{
    VexNode xlist[VertexNum];//头结点数组
    int vexnum,arcnum;//边和结点数目
}OLGraph;//多重邻接表
```

四、测试结果

测试用例：

test.txt

```
10 20
1 2 3 4 5 6 7 8 9 10
1 2 1
2 3 1
3 4 1
4 5 1
5 6 1
6 7 1
7 8 1
8 9 1
9 10 1
10 1 1
2 10 1
6 2 1
10 6 1
1 5 1
7 1 1
5 7 1
9 3 1
4 9 1
8 3 1
4 8 1

```

测试结果：

（1）

```
1.邻接矩阵建立
2.邻接表建立
3.邻接多重表建立
4.先用邻接矩阵建立，再转化到邻接表
5.先用邻接表建立，再转化到邻接矩阵
6.先用邻接表建立，再转化到十字链表
7.先用十字链表建立，再转化到邻接表
8.先用邻接矩阵建立，再转化到十字链表
9.先用十字链表建立，再转化到邻接矩阵
1
      1  2  3  4  5  6  7  8  9 10

1     0  1  0  0  1  0  0  0  0  0
2     0  0  1  0  0  0  0  0  0  1
3     0  0  0  1  0  0  0  0  0  0
4     0  0  0  0  1  0  0  1  1  0
5     0  0  0  0  0  1  1  0  0  0
6     0  1  0  0  0  0  1  0  0  0
7     1  0  0  0  0  0  0  1  0  0
8     0  0  1  0  0  0  0  0  1  0
9     0  0  1  0  0  0  0  0  0  1
10     1  0  0  0  0  1  0  0  0  0
邻接矩阵的深度优先遍历（递归）
2  3  4  5  6  7  1  8  9  10
邻接矩阵的深度优先遍历（非递归）
2  3  4  5  6  7  1  8  9  10
邻接矩阵的广度优先遍历
2  3  10  4  1  6  5  8  9  7
```

（2）

```
2
1-> 5|1-> 2|1
2-> 10|1-> 3|1
3-> 4|1
4-> 8|1-> 9|1-> 5|1
5-> 7|1-> 6|1
6-> 2|1-> 7|1
7-> 1|1-> 8|1
8-> 3|1-> 9|1
9-> 3|1-> 10|1
10-> 6|1-> 1|1
邻接表的深度优先遍历（递归）
2  10  6  7  1  5  8  3  4  9
邻接表的深度优先遍历（非递归）
2  10  6  7  1  5  8  3  4  9
邻接表的广度优先遍历
2  10  3  6  1  4  7  5  8  9
Process returned 0 (0x0)   execution time : 1.235 s
Press any key to continue.
```

（3）

```
3
1入边
w:(1)7->1  w:(1)10->1
1出边
w:(1)1->5  w:(1)1->2
2入边
w:(1)6->2  w:(1)1->2
2出边
w:(1)2->10  w:(1)2->3
3入边
w:(1)8->3  w:(1)9->3  w:(1)2->3
3出边
w:(1)3->4
4入边
w:(1)3->4
4出边
w:(1)4->8  w:(1)4->9  w:(1)4->5
5入边
w:(1)1->5  w:(1)4->5
5出边
w:(1)5->7  w:(1)5->6
6入边
w:(1)10->6  w:(1)5->6
6出边
w:(1)6->2  w:(1)6->7
7入边
```

```
w:(1)5->7  w:(1)6->7
7出边
w:(1)7->1  w:(1)7->8
8入边
w:(1)4->8  w:(1)7->8
8出边
w:(1)8->3  w:(1)8->9
9入边
w:(1)4->9  w:(1)8->9
9出边
w:(1)9->3  w:(1)9->10
10入边
w:(1)2->10  w:(1)9->10
10出边
w:(1)10->6  w:(1)10->1
多重邻接表的深度优先遍历（递归）
2  10  6  7  1  5  8  3  4  9
多重邻接表的深度优先遍历（非递归）
2  10  6  7  1  5  8  3  4  9
多重邻接表的广度优先遍历
2  10  3  6  1  4  7  5  8  9
Process returned 0 (0x0)   execution time : 1.075 s
Press any key to continue.
```

（4）

```
4
转化前
      1  2  3  4  5  6  7  8  9 10

1     0  1  0  0  1  0  0  0  0  0
2     0  0  1  0  0  0  0  0  0  1
3     0  0  0  1  0  0  0  0  0  0
4     0  0  0  0  1  0  0  1  1  0
5     0  0  0  0  0  1  1  0  0  0
6     0  1  0  0  0  0  1  0  0  0
7     1  0  0  0  0  0  0  1  0  0
8     0  0  1  0  0  0  0  0  1  0
9     0  0  1  0  0  0  0  0  0  1
10     1  0  0  0  0  1  0  0  0  0
转化后
1-> 5|1-> 2|1
2-> 10|1-> 3|1
3-> 4|1
4-> 9|1-> 8|1-> 5|1
5-> 7|1-> 6|1
6-> 7|1-> 2|1
7-> 8|1-> 1|1
8-> 9|1-> 3|1
9-> 10|1-> 3|1
10-> 6|1-> 1|1
```

（5）

```
5
转化前
1-> 5|1-> 2|1
2-> 10|1-> 3|1
3-> 4|1
4-> 8|1-> 9|1-> 5|1
5-> 7|1-> 6|1
6-> 2|1-> 7|1
7-> 1|1-> 8|1
8-> 3|1-> 9|1
9-> 3|1-> 10|1
10-> 6|1-> 1|1
转化后
      1  2  3  4  5  6  7  8  9 10

1     0  1  0  0  1  0  0  0  0  0
2     0  0  1  0  0  0  0  0  0  1
3     0  0  0  1  0  0  0  0  0  0
4     0  0  0  0  1  0  0  1  1  0
5     0  0  0  0  0  1  1  0  0  0
6     0  1  0  0  0  0  1  0  0  0
7     1  0  0  0  0  0  0  1  0  0
8     0  0  1  0  0  0  0  0  1  0
9     0  0  1  0  0  0  0  0  0  1
10     1  0  0  0  0  1  0  0  0  0
```

(6)

```
6
转化前
1-> 5|1-> 2|1
2-> 10|1-> 3|1
3-> 4|1
4-> 8|1-> 9|1-> 5|1
5-> 7|1-> 6|1
6-> 2|1-> 7|1
7-> 1|1-> 8|1
8-> 3|1-> 9|1
9-> 3|1-> 10|1
10-> 6|1-> 1|1
转化后
1入边
w:(1)10->1  w:(1)7->1
1出边
w:(1)1->2  w:(1)1->5
2入边
w:(1)6->2  w:(1)1->2
2出边
w:(1)2->3  w:(1)2->10
3入边
w:(1)9->3  w:(1)8->3  w:(1)2->3
3出边
w:(1)3->4
4入边
w:(1)3->4
4出边
w:(1)4->5  w:(1)4->9  w:(1)4->8
5入边
w:(1)4->5  w:(1)1->5
5出边
w:(1)5->6  w:(1)5->7
6入边
w:(1)10->6  w:(1)5->6
6出边
w:(1)6->7  w:(1)6->2
7入边
w:(1)6->7  w:(1)5->7
7出边
w:(1)7->8  w:(1)7->1
8入边
w:(1)7->8  w:(1)4->8
8出边
w:(1)8->9  w:(1)8->3
9入边
w:(1)8->9  w:(1)4->9
9出边
w:(1)9->10  w:(1)9->3
10入边
w:(1)9->10  w:(1)2->10
10出边
w:(1)10->1  w:(1)10->6
```

（7）

```
7
转化前
1入边
w:(1)7->1  w:(1)10->1
1出边
w:(1)1->5  w:(1)1->2
2入边
w:(1)6->2  w:(1)1->2
2出边
w:(1)2->10  w:(1)2->3
3入边
w:(1)8->3  w:(1)9->3  w:(1)2->3
3出边
w:(1)3->4
4入边
w:(1)3->4
4出边
w:(1)4->8  w:(1)4->9  w:(1)4->5
5入边
w:(1)1->5  w:(1)4->5
5出边
w:(1)5->7  w:(1)5->6
6入边
w:(1)10->6  w:(1)5->6
6出边
w:(1)6->2  w:(1)6->7
7入边
w:(1)5->7  w:(1)6->7
7出边
w:(1)7->1  w:(1)7->8
8入边
w:(1)4->8  w:(1)7->8
8出边
w:(1)8->3  w:(1)8->9
9入边
w:(1)4->9  w:(1)8->9
9出边
w:(1)9->3  w:(1)9->10
10入边
w:(1)2->10  w:(1)9->10
10出边
w:(1)10->6  w:(1)10->1
转化后
1-> 5|1-> 2|1
2-> 10|1-> 3|1
3-> 4|1
4-> 9|1-> 8|1-> 5|1
5-> 7|1-> 6|1
6-> 7|1-> 2|1
7-> 8|1-> 1|1
8-> 9|1-> 3|1
9-> 10|1-> 3|1
10-> 6|1-> 1|1
```

（8）

```
8
转化前
      1  2  3  4  5  6  7  8  9 10

1     0  1  0  0  1  0  0  0  0  0
2     0  0  1  0  0  0  0  0  0  1
3     0  0  0  1  0  0  0  0  0  0
4     0  0  0  0  1  0  0  1  1  0
5     0  0  0  0  0  1  1  0  0  0
6     0  1  0  0  0  0  1  0  0  0
7     1  0  0  0  0  0  0  1  0  0
8     0  0  1  0  0  0  0  0  1  0
9     0  0  1  0  0  0  0  0  0  1
10     1  0  0  0  0  1  0  0  0  0
转化后
1入边
w:(1)10->1  w:(1)7->1
1出边
w:(1)1->5  w:(1)1->2
2入边
w:(1)6->2  w:(1)1->2
2出边
w:(1)2->10  w:(1)2->3
3入边
w:(1)9->3  w:(1)8->3  w:(1)2->3
3出边
w:(1)3->4
4入边
w:(1)3->4
4出边
w:(1)4->9  w:(1)4->8  w:(1)4->5
5入边
w:(1)4->5  w:(1)1->5
5出边
```

```
w:(1)5->7  w:(1)5->6
6入边
w:(1)10->6  w:(1)5->6
6出边
w:(1)6->7  w:(1)6->2
7入边
w:(1)6->7  w:(1)5->7
7出边
w:(1)7->8  w:(1)7->1
8入边
w:(1)7->8  w:(1)4->8
8出边
w:(1)8->9  w:(1)8->3
9入边
w:(1)8->9  w:(1)4->9
9出边
w:(1)9->10  w:(1)9->3
10入边
w:(1)9->10  w:(1)2->10
10出边
w:(1)10->6  w:(1)10->1
```

（9）

```
9
转化前
1入边
w:(1)7->1  w:(1)10->1
1出边
w:(1)1->5  w:(1)1->2
2入边
w:(1)6->2  w:(1)1->2
2出边
w:(1)2->10  w:(1)2->3
3入边
w:(1)8->3  w:(1)9->3  w:(1)2->3
3出边
w:(1)3->4
4入边
w:(1)3->4
4出边
w:(1)4->8  w:(1)4->9  w:(1)4->5
5入边
w:(1)1->5  w:(1)4->5
5出边
w:(1)5->7  w:(1)5->6
6入边
w:(1)10->6  w:(1)5->6
6出边
w:(1)6->2  w:(1)6->7
7入边
w:(1)5->7  w:(1)6->7
7出边
w:(1)7->1  w:(1)7->8
8入边
w:(1)4->8  w:(1)7->8
8出边
```

```
w:(1)8->3  w:(1)8->9
9入边
w:(1)4->9  w:(1)8->9
9出边
w:(1)9->3  w:(1)9->10
10入边
w:(1)2->10  w:(1)9->10
10出边
w:(1)10->6  w:(1)10->1
转化后
      1  2  3  4  5  6  7  8  9 10

1     0  1  0  0  1  0  0  0  0  0
2     0  0  1  0  0  0  0  0  0  1
3     0  0  0  1  0  0  0  0  0  0
4     0  0  0  0  1  0  0  1  1  0
5     0  0  0  0  0  1  1  0  0  0
6     0  1  0  0  0  0  1  0  0  0
7     1  0  0  0  0  0  0  1  0  0
8     0  0  1  0  0  0  0  0  1  0
9     0  0  1  0  0  0  0  0  0  1
10     1  0  0  0  0  1  0  0  0  0
```

五、经验体会与不足

（1）分析和比较各建立算法的时间复杂度以及存储结构的空间占用情况；

假设图 G 有 n 个顶点 e 条边，则该图的空间和时间占用情况为：

① 邻接矩阵：空间需求 O(n+n^2) = O(n^2),与边的条数 e 无关。

时间需求：O（n^2）

② 邻接表：空间需求 O(n + e)

时间需求 O（n + e ）

（2）分析搜索算法的时间复杂度；

深度优先搜索的问题规模 n、e，对每个结点都要访问 O(n)，对每个结点的邻接表进行扫描 O(2e)，所以时间复杂度为 O(n+2e)，广度优先搜索深度优先搜索的非递归情况类似，也是 O(n+2e)

（3）通过实验我更加深入的理解了有向图的三种存储方式之间的联系和区别。邻接矩阵适合边稠密的图，顶点之间的关系清晰明确，但是当边稀疏的时候空间浪费比较多，表示方法唯一。邻接表适合边稀疏的图，但是查找的时候就不是很方便。十字链表就解决了这个问题，方便了入度和出度的计算。三种存储方式在一定程度上来说是很类似的，通过相互转化算法可以看出来。

六、附录：源代码（带注释）

```
#include<iostream>
#include<cstdio>
#include<fstream>
#include<queue>
#include<stack>
#define VertexNum 30
using namespace std;

/*
邻接矩阵的结构
*/
typedef struct{
    int vertex[VertexNum];//顶点数组
    int edge[VertexNum][VertexNum];//从第一个顶点到第二个顶点边的权重
    int n,e;//边和顶点的数目
}MTGraph;
/*
邻接表结构
```

```
*/
typedef struct node{
    int adjvex;//顶点编号
    int cost;//从头结点到该结点的边的权重
    struct node *next;//下一个与头结点关联的结点
}EdgeNode;
typedef struct{
    int vertex;//头结点编号
    EdgeNode *firstedge;//与头结点相关联的第一个结点
}VertexNode;//头结点结构
typedef struct{
    VertexNode verlist[VertexNum];//头结点数组
    int n,e;//边和结点的数目
}AdjGraph;//邻接表
/*
十字链表结构
*/
typedef struct ArcBox{
    int tailvex,headvex;
//tailvex : 尾域, 指示弧尾顶点在图中的位置
//headvex: 头域, 指示弧头顶点在图中的位置
    struct ArcBox *hlink,*tlink;
//hlink : 链域, 指向弧头相同的下一条弧
//tlink : 链域, 指向弧尾相同的下一条弧
    int info;//信息
}ArcBox;//结点结构
typedef struct VexNode{
    int data;//顶点编号
    ArcBox *firstin,*firstout;
// firstin : 链域, 指向以该顶点为弧头的第一个弧结点。firstout : 链域, 指向以该顶点为弧尾的第一个弧结点
}VexNode;//头结点结构
typedef struct{
    VexNode xlist[VertexNum];//头结点数组
    int vexnum,arcnum;//边和结点数目
}OLGraph;//多重邻接表

/*
建立邻接矩阵
*/
void Create0(MTGraph *G)
{
    int i,j,k,w;
    scanf("%d",&G->n);
```

```
    scanf("%d",&G->e);
    for(i=1;i<=G->n;i++)
    {
        scanf("%d",&G->vertex[i]);
    }
    for(i=1;i<=G->n;i++)
    {
        for(j=1;j<=G->n;j++)
        {
            G->edge[i][j]=0;
        }
    }
    for(k=1;k<=G->e;k++)
    {
        scanf("%d %d %d",&i,&j,&w);
        G->edge[i][j]=w;
        //G->edge[j][i]=w;
    }
}
/*
打印邻接矩阵
*/
void Print0(MTGraph *G)
{
    cout<<"    ";
    for(int i=1;i<=G->n;i++)
    {
        printf("%3d",G->vertex[i]);
    }
    cout<<endl<<endl;
    for(int i=1;i<=G->n;i++)
    {
        cout<<i<<"   ";
        for(int j=1;j<=G->n;j++)
        {
            printf("%3d",G->edge[i][j]);
        }
        cout<<endl;
    }
}
/*
建立邻接表
*/
void Create1(AdjGraph &G)
```

```
{
    cin>>G.n>>G.e;
    int m,n;
    int w;
    for(int i=1;i<=G.n;i++)
    {
        cin>>G.verlist[i].vertex;
        G.verlist[i].firstedge=NULL;
    }
    for(int i=1;i<=G.e;i++)
    {
        cin>>m>>n>>w;
        //m
        EdgeNode *p=new EdgeNode;
        p->cost=w;
        p->adjvex=n;
        EdgeNode *tmp;
        tmp=G.verlist[m].firstedge;
        G.verlist[m].firstedge=p;
        p->next=tmp;
    }
}
/*
打印邻接表
*/
void Print1(AdjGraph &G)
{
    for(int i=1;i<=G.n;i++)
    {
        cout<<G.verlist[i].vertex;
        EdgeNode *tmp=G.verlist[i].firstedge;
        while(tmp!=NULL)
        {
            cout<<"-> ";
            cout<<tmp->adjvex<<"|"<<tmp->cost;//"|"后面是边权重

            tmp=tmp->next;
        }
        cout<<endl;
    }
}
/*
建立十字链表
*/
```

```
150. void Create2(OLGraph &G)
151. {
152.     scanf("%d %d",&G.vexnum,&G.arcnum);
153.     for(int i=1;i<=G.vexnum;i++)
154.     {
155.         scanf("%d",&G.xlist[i].data);
156.         G.xlist[i].firstin=NULL;
157.         G.xlist[i].firstout=NULL;
158.     }
159.     for(int k=1;k<=G.arcnum;k++)
160.     {
161.         int m,n,w;
162.         scanf("%d %d %d",&m,&n,&w);
163.         ArcBox *p=new ArcBox;
164.
165.         p->tailvex=m;
166.         p->headvex=n;
167.         p->info=w;
168.         p->tlink=G.xlist[m].firstout;
169.         p->hlink=G.xlist[n].firstin;
170.         G.xlist[m].firstout=p;
171.         G.xlist[n].firstin=p;
172.     }
173. }
174. /*
175. 打印十字链表
176. */
177. void Print2(OLGraph &G)
178. {
179.     for(int i=1;i<=G.vexnum;i++)
180.     {
181.
182.         cout<<G.xlist[i].data<<"入边"<<endl;
183.         ArcBox *tmp=G.xlist[i].firstin;
184.         while(tmp!=NULL)
185.         {
186.             cout<<"w:"<<"("<<tmp->info<<")";
187.             cout<<tmp->tailvex<<"->"<<tmp->headvex<<"  ";//<<"w:"<<tmp->inf
    o<<"  ";
188.             tmp=tmp->hlink;
189.         }
190.         cout<<endl;
191.         cout<<G.xlist[i].data<<"出边"<<endl;
192.         ArcBox *tmp1=G.xlist[i].firstout;
```

```
193.         while(tmp1!=NULL)
194.          {
195.              cout<<"w:"<<"("<<tmp1->info<<")";
196.              cout<<tmp1->tailvex<<"->"<<tmp1->headvex<<"  ";//<<"w:"<<tmp->i
    nfo<<"  ";
197.              tmp1=tmp1->tlink;
198.          }
199.          cout<<endl;
200.     }
201.     return;
202. }
203. /*
204. 邻接矩阵转邻接表
205. */
206. void turn0to1(MTGraph &G0,AdjGraph &G1)
207. {
208.     G1.e=G0.e;
209.     G1.n=G0.n;
210.     for(int i=1;i<=G0.n;i++)
211.     {
212.         G1.verlist[i].vertex=G0.vertex[i];
213.         G1.verlist[i].firstedge=NULL;
214.     }
215.     for(int i=1;i<=G0.n;i++)
216.     {
217.         for(int j=1;j<=G0.n;j++)
218.         {
219.             if(G0.edge[i][j]!=0)
220.             {
221.                 EdgeNode *p=new EdgeNode;
222.                 p->cost=G0.edge[i][j];
223.                 p->adjvex=j;
224.                 EdgeNode *tmp;
225.                 tmp=G1.verlist[i].firstedge;
226.                 G1.verlist[i].firstedge=p;
227.                 p->next=tmp;
228.             }
229.         }
230.     }
231. }
232. /*
233. 邻接表转邻接矩阵
234. */
235. void turn1to0(MTGraph &G0,AdjGraph &G1)
```

```
{

    G0.e=G1.e;
    G0.n=G1.n;
    for(int i=1;i<=G1.n;i++)
    {
        G0.vertex[i]=G1.verlist[i].vertex;
    }
    for(int i=1;i<=G0.n;i++)
    {
        for(int j=1;j<=G0.n;j++)
        {
            G0.edge[i][j]=0;
        }
    }
    for(int i=1;i<=G1.n;i++)
    {
        EdgeNode *tmp=G1.verlist[i].firstedge;
        while(tmp!=NULL)
        {
            G0.edge[i][tmp->adjvex]=tmp->cost;
            tmp=tmp->next;
        }
    }
}
/*
邻接表转十字链表
*/
void turn1to2(AdjGraph &G1,OLGraph &G2)
{

    G2.arcnum=G1.e;
    G2.vexnum=G1.n;
    for(int i=1;i<=G2.vexnum;i++)
    {
        G2.xlist[i].data=G1.verlist[i].vertex;
        G2.xlist[i].firstin=NULL;
        G2.xlist[i].firstout=NULL;
    }
    for(int k=1;k<=G2.vexnum;k++)
    {
        EdgeNode *tmp=G1.verlist[k].firstedge;
        while(tmp!=NULL)
        {
```

```
            int m,n,w;
            m=G1.verlist[k].vertex;
            n=tmp->adjvex;
            w=tmp->cost;
            ArcBox *p=new ArcBox;
            p->tailvex=m;
            p->headvex=n;
            p->info=w;
            p->tlink=G2.xlist[m].firstout;
            p->hlink=G2.xlist[n].firstin;
            G2.xlist[m].firstout=p;
            G2.xlist[n].firstin=p;
            tmp=tmp->next;
        }
    }
}
/*
十字链表转邻接表
*/
void turn2to1(AdjGraph &G1,OLGraph &G2)
{

    G1.e=G2.arcnum;
    G1.n=G2.vexnum;
    for(int i=1;i<=G1.n;i++)
    {
        G1.verlist[i].vertex=G2.xlist[i].data;
        G1.verlist[i].firstedge=NULL;
    }

        int m,n,w;
        for(int j=1;j<=G1.n;j++)
        {
            ArcBox *tmp=G2.xlist[j].firstin;
            while(tmp!=NULL)
            {
                w=tmp->info;
                m=tmp->tailvex;
                n=tmp->headvex;
                EdgeNode *p=new EdgeNode;
                p->cost=w;
                p->adjvex=n;
                EdgeNode *tmp1;
                tmp1=G1.verlist[m].firstedge;
```

```
                G1.verlist[m].firstedge=p;
                p->next=tmp1;
                tmp=tmp->hlink;
            }
        }
}
/*
邻接矩阵转十字链表
*/
void turn0to2(MTGraph &G0,OLGraph &G2)
{

    G2.arcnum=G0.e;
    G2.vexnum=G0.n;
    for(int i=1;i<=G2.vexnum;i++)
    {
        G2.xlist[i].data=G0.vertex[i];
        G2.xlist[i].firstin=NULL;
        G2.xlist[i].firstout=NULL;
    }

     for(int i=1;i<=G0.n;i++)
    {
        for(int j=1;j<=G0.n;j++)
        {
            if(G0.edge[i][j]!=0)
            {
                int m,n,w;
                m=i;
                n=j;
                w=G0.edge[i][j];
                ArcBox *p=new ArcBox;

                p->tailvex=m;
                p->headvex=n;
                p->info=w;
                p->tlink=G2.xlist[m].firstout;
                p->hlink=G2.xlist[n].firstin;
                G2.xlist[m].firstout=p;
                G2.xlist[n].firstin=p;
            }
        }
    }

```

```
}
/*
十字链表转邻接矩阵
*/
void turn2to0(MTGraph &G0,OLGraph &G2)
{

    G0.e=G2.arcnum;
    G0.n=G2.vexnum;
    for(int i=1;i<=G0.n;i++)
    {
        G0.vertex[i]=G2.xlist[i].data;
    }
    for(int i=1;i<=G0.n;i++)
    {
        for(int j=1;j<=G0.n;j++)
        {
            G0.edge[i][j]=0;
        }
    }
        for(int j=1;j<=G0.n;j++)
        {
            int m,n,w;
            ArcBox *tmp=G2.xlist[j].firstin;
            while(tmp!=NULL)
            {
                w=tmp->info;
                m=tmp->tailvex;
                n=tmp->headvex;
                G0.edge[m][n]=w;
                tmp=tmp->hlink;
            }
        }
}
int visited[VertexNum];//访问标记数组是全局变量
int dfn[VertexNum];//顶点先深编号
/*
初始化
*/
void initial()
{
    for(int i=0;i<VertexNum;i++)
    {
        visited[i]=0;
```

```
412.         dfn[i]=0;
413.     }
414. }
415. /*
416. 在邻接矩阵上做深度优先搜索递归
417. */
418. void DFS0(MTGraph *G,int i)
419. {
420.     cout<<G->vertex[i]<<"  ";
421.     visited[i]=1;
422.     for(int j=1;j<=G->n;j++)
423.     {
424.         if((G->edge[i][j]!=0)&&visited[j]==0)
425.         {
426.             DFS0(G,j);
427.         }
428.     }
429. }
430. /*
431. 在邻接表上做深度优先搜索递归
432. */
433. void DFS1(AdjGraph *G,int i)
434. {
435.     EdgeNode *p;
436.     cout<<G->verlist[i].vertex<<"  ";
437.     visited[i]=1;
438.     p=G->verlist[i].firstedge;
439.     while(p)
440.     {
441.         if(!visited[p->adjvex])
442.             DFS1(G,p->adjvex);
443.         p=p->next;
444.     }
445. }
446. /*
447. 在十字链表上做深度优先搜索递归
448. */
449. void DFS2(OLGraph *G,int i)
450. {
451.     ArcBox *p;
452.     cout<<G->xlist[i].data<<"  ";
453.     visited[i]=1;
454.     p=G->xlist[i].firstout;
455.     while(p)
```

```
456.     {
457.         if(!visited[p->headvex])
458.             DFS2(G,p->headvex);
459.         p=p->tlink;
460.     }
461. }
462. /*
463. 在邻接矩阵上做广度优先搜索
464. */
465. void BFS0(MTGraph *G,int k)
466. {
467.     int i,j;
468.     queue<int>q;
469.     cout<<G->vertex[k]<<"  ";
470.     visited[k]=1;
471.     q.push(G->vertex[k]);
472.     while(!q.empty())
473.     {
474.         i=q.front();
475.         q.pop();
476.         for(j=1;j<=G->n;j++)
477.         {
478.             if(G->edge[i][j]!=0&&visited[j]==0)
479.             {
480.                 cout<<G->vertex[j]<<"  ";
481.                 visited[j]=1;
482.                 q.push(G->vertex[j]);
483.             }
484.         }
485.     }
486. }
487. /*
488. 在邻接表上做广度优先搜索
489. */
490. void BFS1(AdjGraph *G,int k)
491. {
492.     int i;
493.     EdgeNode *p;
494.     queue<int>q;
495.     cout<<G->verlist[k].vertex<<"  ";
496.     visited[k]=1;
497.     q.push(G->verlist[k].vertex);
498.     while(!q.empty())
499.     {
```

```
500.         i=q.front();
501.         q.pop();
502.         p=G->verlist[i].firstedge;
503.         while(p)
504.         {
505.             if(!visited[p->adjvex])
506.             {
507.                 cout<<G->verlist[p->adjvex].vertex<<"  ";
508.                 visited[p->adjvex]=1;
509.                 q.push(p->adjvex);
510.             }
511.             p=p->next;
512.         }
513.     }
514. }
515. /*
516. 在十字链表上做广度优先搜索
517. */
518. void BFS2(OLGraph *G,int k)
519. {
520.     int i;
521.     ArcBox *p;
522.     queue<int>q;
523.     cout<<G->xlist[k].data<<"  ";
524.     visited[k]=1;
525.     q.push(G->xlist[k].data);
526.     while(!q.empty())
527.     {
528.         i=q.front();
529.         q.pop();
530.         p=G->xlist[i].firstout;
531.         while(p)
532.         {
533.             if(!visited[p->headvex])
534.             {
535.                 cout<<G->xlist[p->headvex].data<<"  ";
536.                 visited[p->headvex]=1;
537.                 q.push(p->headvex);
538.
539.             }
540.             p=p->tlink;
541.         }
542.     }
543. }
```

```
/*
在邻接矩阵上做深度优先搜索非递归
*/
void DFS_0(MTGraph *G,int i)
{
    stack<int>s;
    s.push(G->vertex[i]);
    cout<<s.top()<<"  ";
    visited[s.top()]=1;
    while(!s.empty())
    {
        i=s.top();
        for(int j=1;j<=G->n;j++)
        {
            if((G->edge[i][j]!=0)&&visited[j]==0)
            {
                s.push(G->vertex[j]);
                cout<<s.top()<<"  ";
                visited[j]=1;
                i=j;j=0;
            }
        }
        if(!s.empty())
        {
            s.pop();
        }
    }

}
/*
在邻接表上做深度优先搜索非递归
*/
void DFS_1(AdjGraph *G,int i)
{
   // cout<<G->verlist[10].firstedge->adjvex<<endl;
    stack<int>s;
    s.push(G->verlist[i].vertex);
    cout<<s.top()<<"  ";
    visited[s.top()]=1;
    int flag=0;
    while(!s.empty())
    {

        i=s.top();
```

```
588.         for(EdgeNode *tmp=G->verlist[i].firstedge;tmp!=NULL;)
589.         {
590.            // cout<<"&"<<tmp->adjvex<<" ";
591.             if(visited[tmp->adjvex]==0)
592.             {
593.                 s.push(tmp->adjvex);
594.                 cout<<s.top()<<"  ";
595.                 visited[tmp->adjvex]=1;
596.                 i=s.top();
597.                 flag=1;
598.
599.              }
600.              tmp=tmp->next;
601.              if(flag==1){flag=0;tmp=G->verlist[i].firstedge;}
602.
603.         }
604.         if(!s.empty())
605.         {
606.             s.pop();
607.         }
608.     }
609.
610. }
611. /*
612. 在十字链表上做深度优先搜索非递归
613. */
614. void DFS_2(OLGraph *G,int i)
615. {
616.     stack<int>s;
617.     s.push(G->xlist[i].data);
618.     cout<<s.top()<<"  ";
619.     visited[s.top()]=1;
620.     int flag=0;
621.     while(!s.empty())
622.     {
623.         i=s.top();
624.
625.         for(ArcBox *tmp=G->xlist[i].firstout;tmp!=NULL;)
626.         {
627.             if(visited[tmp->headvex]==0)
628.             {
629.                 s.push(tmp->headvex);
630.                 cout<<s.top()<<"  ";
631.                 visited[tmp->headvex]=1;
```

```
                i=s.top();
                flag=1;

            }
            tmp=tmp->tlink;
            if(flag==1){flag=0;tmp=G->xlist[i].firstout;}

        }
        if(!s.empty())
        {
            s.pop();
        }
    }
}
int main()
{
    int option;
    cout<<"1.邻接矩阵建立"<<endl;
    cout<<"2.邻接表建立"<<endl;
    cout<<"3.邻接多重表建立"<<endl;
    cout<<"4.先用邻接矩阵建立，再转化到邻接表"<<endl;
    cout<<"5.先用邻接表建立，再转化到邻接矩阵"<<endl;
    cout<<"6.先用邻接表建立，再转化到十字链表"<<endl;
    cout<<"7.先用十字链表建立，再转化到邻接表"<<endl;
    cout<<"8.先用邻接矩阵建立，再转化到十字链表"<<endl;
    cout<<"9.先用十字链表建立，再转化到邻接矩阵"<<endl;
    scanf("%d",&option);

    if(option==1)
    {
        freopen("test.txt","r",stdin);
        MTGraph Graph;
        Create0(&Graph);
        Print0(&Graph);
        initial();
        cout<<"邻接矩阵的深度优先遍历（递归）"<<endl;
        DFS0(&Graph,2);
        cout<<endl;
        initial();
        cout<<"邻接矩阵的深度优先遍历（非递归）"<<endl;
        DFS_0(&Graph,2);
        cout<<endl;
        initial();
        cout<<"邻接矩阵的广度优先遍历"<<endl;
```

```
676.         BFS0(&Graph,2);
677.         fclose(stdin);
678.     }
679.
680.     if(option==2)
681.     {
682.         freopen("test.txt","r",stdin);
683.         AdjGraph Graph;
684.         Create1(Graph);
685.         Print1(Graph);
686.         initial();
687.         cout<<"邻接表的深度优先遍历（递归）"<<endl;
688.         DFS1(&Graph,2);
689.         cout<<endl;
690.         initial();
691.         cout<<"邻接表的深度优先遍历（非递归）"<<endl;
692.         DFS_1(&Graph,2);
693.         cout<<endl;
694.         initial();
695.         cout<<"邻接表的广度优先遍历"<<endl;
696.         BFS1(&Graph,2);
697.         fclose(stdin);
698.     }
699.
700.     if(option==3)
701.     {
702.         freopen("test.txt","r",stdin);
703.         OLGraph Graph;
704.         Create2(Graph);
705.         Print2(Graph);
706.         initial();
707.         cout<<"多重邻接表的深度优先遍历（递归）"<<endl;
708.         DFS2(&Graph,2);
709.         cout<<endl;
710.         initial();
711.         cout<<"多重邻接表的深度优先遍历（非递归）"<<endl;
712.         DFS_2(&Graph,2);
713.         cout<<endl;
714.         initial();
715.         cout<<"多重邻接表的广度优先遍历"<<endl;
716.         BFS2(&Graph,2);
717.         fclose(stdin);
718.     }
719.
```

```
720.     if(option==4)
721.     {
722.         freopen("test.txt","r",stdin);
723.         MTGraph Graph0;
724.         AdjGraph Graph1;
725.         Create0(&Graph0);
726.         cout<<"转化前"<<endl;
727.         Print0(&Graph0);
728.         turn0to1(Graph0,Graph1);
729.         cout<<"转化后"<<endl;
730.         Print1(Graph1);
731.         fclose(stdin);
732.     }
733.
734.     if(option==5)
735.     {
736.         freopen("test.txt","r",stdin);
737.         MTGraph Graph0;
738.         AdjGraph Graph1;
739.         Create1(Graph1);
740.         cout<<"转化前"<<endl;
741.         Print1(Graph1);
742.         turn1to0(Graph0,Graph1);
743.         cout<<"转化后"<<endl;
744.         Print0(&Graph0);
745.         fclose(stdin);
746.
747.     }
748.
749.     if(option==6)
750.     {
751.         freopen("test.txt","r",stdin);
752.         AdjGraph Graph1;
753.         OLGraph Graph2;
754.         Create1(Graph1);
755.         cout<<"转化前"<<endl;
756.         Print1(Graph1);
757.         turn1to2(Graph1,Graph2);
758.         cout<<"转化后"<<endl;
759.         Print2(Graph2);
760.         fclose(stdin);
761.     }
762.
763.      if(option==7)
```

```
764.     {
765.         freopen("test.txt","r",stdin);
766.         AdjGraph Graph1;
767.         OLGraph Graph2;
768.         Create2(Graph2);
769.         cout<<"转化前"<<endl;
770.         Print2(Graph2);
771.         turn2to1(Graph1,Graph2);
772.         cout<<"转化后"<<endl;
773.         Print1(Graph1);
774.         fclose(stdin);
775.     }
776.
777.      if(option==8)
778.     {
779.         freopen("test.txt","r",stdin);
780.         MTGraph Graph0;
781.         OLGraph Graph2;
782.         Create0(&Graph0);
783.         cout<<"转化前"<<endl;
784.         Print0(&Graph0);
785.         turn0to2(Graph0,Graph2);
786.         cout<<"转化后"<<endl;
787.         Print2(Graph2);
788.         fclose(stdin);
789.     }
790.
791.      if(option==9)
792.     {
793.         freopen("test.txt","r",stdin);
794.         MTGraph Graph0;
795.         OLGraph Graph2;
796.         Create2(Graph2);
797.         cout<<"转化前"<<endl;
798.         Print2(Graph2);
799.         turn2to0(Graph0,Graph2);
800.         cout<<"转化后"<<endl;
801.         Print0(&Graph0);
802.         fclose(stdin);
803.     }
804.     return 0;
805. }
```