实验报告（week-09）

2020111235 马靖淳

一．实验任务

1. 邻接矩阵存储结构
2. 邻接表存储结构
3. 图的遍历

二．实验上机时间

三．知识点

1. 邻接矩阵存储结构
2. 邻接表存储结构
3. 图的遍历
   (1) 深度优先遍历（DFS）
   (2) 广度优先遍历（BFS）
4. 最小生成树

四．源代码及结果截屏

1. 邻接矩阵存储结构

（1）AdjacencyMatrix.h

```
#pragma once
#ifndef ADJACENCYMATRIX_H
#define ADJACENCYMATRIX_H

#include <stdlib.h>
#include "Status.h"

#define INFINITY INT_MAX//最大值
#define MAX_VERTEX_NUM 20 //最大顶点个数

typedef int VRType;
typedef int InfoType;
typedef int VertexType;
typedef enum{ DG,DN,UDG,UDN } GraphKind;
//{DG有向图，DN有向网，UDG无向图，UDN无向网}
typedef struct ArcCell
{
    VRType adj;//VRType是顶点关系类型
    //对无权图，用1或0表示相邻否；对带权图，则为权值类型
    InfoType* info;//该弧相关信息的指针
}ArcCell,AdjMatrix[MAX_VERTEX_NUM][MAX_VERTEX_NUM];

typedef struct//图的类型定义
{
    VertexType vexs[MAX_VERTEX_NUM];//顶点向量
    AdjMatrix arcs;//邻接矩阵
    int vexnum, arcnum;//图的当前顶点数和弧数
```

```
    GraphKind kind;//图的种类标志
}MGraph;

Status CreateUDN(MGraph* G);
#endif
```

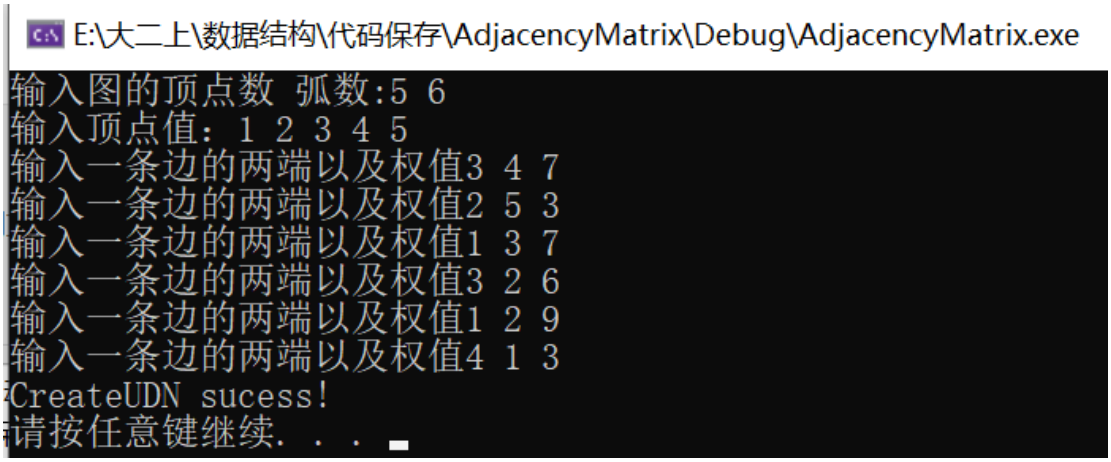
(2) AdjacencyMatrix.c

```
#define _CRT_SECURE_NO_WARNINGS
#include "AdjacencyMatrix.h"
#include "Status.h"
#include <stdio.h>
#include <malloc.h>

int LocateVex(MGraph G, VRType point)   //确定某个顶点在图G中的位置
{
    int i, j;
    for (i = 0; i < G.vexnum; i++)
        if (G.vexs[i] == point)
            return i;
}//LocateVex

Status CreateUDN(MGraph* G)
{
    int i, j, k;
    VRType v1, v2, w;
    printf("输入图的顶点数 弧数:");
    scanf("%d %d", &((*G).vexnum), &((*G).arcnum));
    printf("输入顶点值：");
    for (i = 0; i < (*G).vexnum; i++)
    {
        scanf("%d", &((*G).vexs[i]));
    }
    for (i = 0; i < (*G).vexnum; i++)
    {
        //初始化邻接矩阵
        for (j = 0; j < (*G).vexnum; j++)
        {
            (*G).arcs[i][j].adj = INFINITY;
            (*G).arcs[i][j].info = NULL;
        }
    }
    for (k = 0; k < (*G).arcnum; k++)
    {
        printf("输入一条边的两端以及权值");
```

```
            scanf("%d %d %d", &v1, &v2, &w);
            i = LocateVex(*G, v1);//查找v1在数组vexs中的序号i
            j = LocateVex(*G, v2);//查找v1在数组vexs中的序号i
            (*G).arcs[i][j].adj = w;
            (*G).arcs[j][i].adj = w;
            (*G).arcs[i][j].info = (InfoType*)malloc(sizeof(InfoType));
            (*G).arcs[j][i].info = (InfoType*)malloc(sizeof(InfoType));
            *((*G).arcs[i][j].info) = w;
            *((*G).arcs[j][i].info) = w;
        }
        return OK;
}//CreateUDN
```

(3) test.c

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include "AdjacencyMatrix.h"

int main()
{
    //InitList test
    MGraph G;
    if (CreateUDN(&G))
        printf("CreateUDN sucess!\n");
    else
        printf("CreateUDN unsucess!\n");
    system("pause");
    return 0;
}
```



2. **邻接表存储结构**
   (1) **邻接表表示的存储实现**
   (2) **建立无向图 G 的邻接表**
   (3) **深度优先遍历（DFS）**

(4) 广度优先遍历（BFS）

a. AdjacencyList.h

```
#pragma once
#ifndef ADJACENCYLIST_H
#define ADJACENCYLIST_H

#include <stdlib.h>
#include "Status.h"

typedef int InfoType;
typedef int VertexType;
typedef int Boolean;
#define MAX_VERTEX_NUM 20
//边表的存储类型
typedef struct ArcNode
{
    int adjvex;//该弧所指向的顶点的位置
    struct ArcNode* nextarc;//指向下一条弧的指针
    InfoType* info;//该弧相关信息的指针
}ArcNode;
//顶点表的存储类型
typedef struct VNode
{
    VertexType data;//顶点信息
    ArcNode* firstarc;//指向第一条依附该顶点的弧
}VNode,AdjList[MAX_VERTEX_NUM];
//图的邻接表存储类型
typedef struct
{
    AdjList vertices;
    int vexnum, arcnum;
    int kind;//图的种类标志
}ALGraph;

int LocateVex(ALGraph G, VertexType u);
Status CreateUDG(ALGraph* G);
void DFSTraverse(ALGraph G);
void DFS(ALGraph G, int v);
void Visit(ALGraph G, int v);
int FirstAdjVex(ALGraph G, int v);
int NextAdjVex(ALGraph G, int v, int w);
void BFSTraverse(ALGraph G);

#endif
```

b. AdjacencyList.c

```
#define _CRT_SECURE_NO_WARNINGS
#include "AdjacencyList.h"
#include "LinkQueue.h"
#include "Status.h"
#include <stdio.h>
#include <malloc.h>

int LocateVex(ALGraph G, VertexType u)//返回顶点u在vertices数组中的位置
{
    for (int i = 0; i <= G.vexnum; i++)
    {
        if (G.vertices[i].data == u)
            return i;
    }
    return ERROR;
}

Status CreateUDG(ALGraph* G)
{
    ArcNode* p, * q;
    int i, j, k;
    VertexType v1, v2;
    ArcNode* r[MAX_VERTEX_NUM + 1];
    printf("输入图的顶点数 边数:");
    scanf("%d %d", &((*G).vexnum), &((*G).arcnum));//顶点和边个数
    getchar();
    printf("输入每个顶点的信息：\n");
    for (i = 1; i <= (*G).vexnum; i++)
    {
        printf("输入第%d个顶点的信息：",i);
        scanf("%c", &((*G).vertices[i].data));//顶点信息
        getchar();
        (*G).vertices[i].firstarc = NULL;
        r[i] = NULL;
    }

    for (k = 1; k <= (*G).arcnum; k++)
    {
        //依次输入每个边信息
        printf("输入一条边的起点和终点的编号：");
        scanf("%c %c", &v1, &v2);
        getchar();
```

```
            i = LocateVex(*G, v1);//需查找顶点编号
            j = LocateVex(*G, v2);

            p = (ArcNode*)malloc(sizeof(ArcNode));
            if (!p)
                exit(OVERFLOW);
            p->adjvex = j;
            p->nextarc = NULL;
            if (r[i] == NULL)//邻接表中第一个结点
                (*G).vertices[i].firstarc = p;//加入到邻接表
            else
                r[i]->nextarc = p;
            r[i] = p;

            q = (ArcNode*)malloc(sizeof(ArcNode));
            if (!q)
                exit(OVERFLOW);
            q->adjvex = i;
            q->nextarc = NULL;
            if (r[j] == NULL)//邻接表中第一个结点
                (*G).vertices[j].firstarc = q;//加入到邻接表
            else
                r[j]->nextarc = q;
            r[j] = q;
        }
        return OK;
}//CreateUDG

Boolean visited[MAX_VERTEX_NUM+1];
void DFSTraverse(ALGraph G)
{
    //对图G作深度优先遍历
    int v = 0;
    printf("DFSGraph:\n");
    for (v = 1; v < G.vexnum; ++v)
    {
        visited[v] = FALSE;//访问标志数组初始化
    }
    for (v = 1; v < G.vexnum; ++v)
    {
        if (!visited[v])//对尚未访问的顶点调用DFS
            DFS(G, v);
    }
}
```

```
void DFS(ALGraph G, int v)
{
    //从顶点v出发，深度优先搜索遍历连通图G
    visited[v] = TRUE;
    Visit(G, v);//访问v
    int w;
    for (w = FirstAdjVex(G, v); w >= 0; w = NextAdjVex(G, v, w))
    {
        if (!visited[w])//对v的尚未访问的邻接顶点w
            DFS(G, w);//递归调用DFS
    }
}//DFS

void Visit(ALGraph G, int v)
{
    printf("%c ", G.vertices[v].data);
}

int FirstAdjVex(ALGraph G, int v)
{
    if (G.vertices[v].firstarc != NULL)
        return G.vertices[v].firstarc->adjvex;
    return OVERFLOW;
}

int NextAdjVex(ALGraph G, int v, int w)
{
    ArcNode* p;
    p = G.vertices[v].firstarc;
    while (p->adjvex != w)
        p = p->nextarc;
    if (p->nextarc != NULL)
        return p->nextarc->adjvex;
    return OVERFLOW;
}

void BFSTraverse(ALGraph G)
{
    LinkQueue Q;
    int v, w;
    QElemType u = 0;

    for (v = 0; v <= G.vexnum; ++v)
```

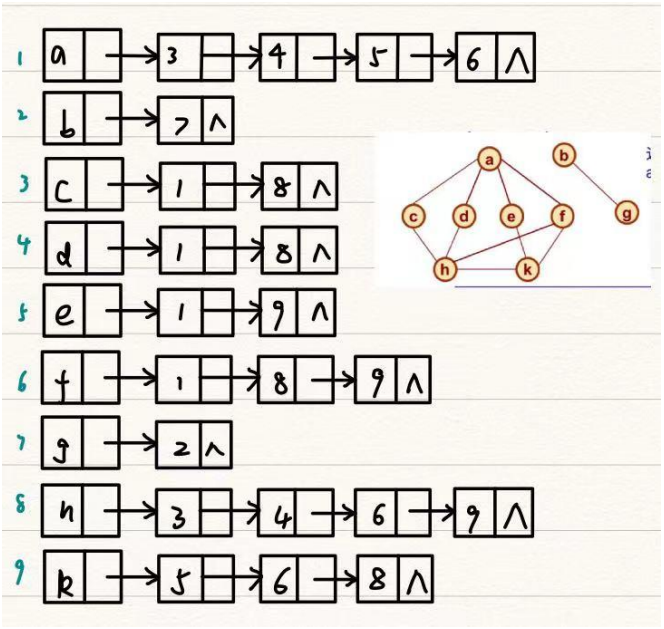
```
        visited[v] = FALSE;//初始化标志
    InitQueue(&Q);//置空的辅助队列Q
    for (v = 1; v <= G.vexnum; ++v)
        if (!visited[v])
        {
            //v未被访问
            visited[v] = TRUE;
            Visit(G, v);//访问v
            EnQueue(&Q, v);//v入队列
            while (!QueueEmpty(Q))
            {
                DeQueue(&Q, &u);//队头元素出队并置为u
                for(w=FirstAdjVex(G,u);w>=0;w=NextAdjVex(G,u,w))
                    if (!visited[w])
                    {
                        //w为u的尚未被访问的邻接顶点
                        visited[w] = TRUE;
                        Visit(G, w);//访问w
                        EnQueue(&Q, w);//访问的顶点w入队列
                    }//if
            }//while
        }//if
}//BFSTraverse
```

c. test.c

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include "AdjacencyList.h"

int main()
{
    //InitList test
    ALGraph G;
    if (CreateUDG(&G))
        printf("CreateUDG sucess!\n");
    else
        printf("CreateUDG unsucess!\n");

    //DFSTraverse test
    DFSTraverse(G);
    printf("\n");

    //BFSTraverse test
    BFSTraverse(G);
```

```
    printf("\n");

    system("pause");
    return 0;
}
```

```
E:\大二上\数据结构\代码保存\AdjacencyList\Debug\AdjacencyList.exe
输入图的顶点数 边数:9 11
输入每个顶点的信息:
输入第1个顶点的信息: 1
输入第2个顶点的信息: 2
输入第3个顶点的信息: 3
输入第4个顶点的信息: 4
输入第5个顶点的信息: 5
输入第6个顶点的信息: 6
输入第7个顶点的信息: 7
输入第8个顶点的信息: 8
输入第9个顶点的信息: 9
输入一条边的起点和终点的编号: 1 3
输入一条边的起点和终点的编号: 1 4
输入一条边的起点和终点的编号: 1 5
输入一条边的起点和终点的编号: 1 6
输入一条边的起点和终点的编号: 2 7
输入一条边的起点和终点的编号: 3 8
输入一条边的起点和终点的编号: 4 8
输入一条边的起点和终点的编号: 5 9
输入一条边的起点和终点的编号: 6 8
输入一条边的起点和终点的编号: 6 9
输入一条边的起点和终点的编号: 8 9
CreateUDG sucess!
DFSGraph:
1 3 8 4 6 9 5 2 7
1 3 4 5 6 8 9 2 7
请按任意键继续. . .
```



3. 最小生成树

a. Prim.h

```
#pragma once
```

```
#ifndef PRIM_H
#define PRIM_H

#include <stdlib.h>
#include "Status.h"
#include "AdjacencyMatrix.h"

typedef struct{
    VertexType adjvex;//U集中的顶点序号
    VRType lowcost;//边的权值
}Edge[MAX_VERTEX_NUM];

void MiniSpanTree_PRIM(MGraph G, VertexType u);

#endif
```

b. Prim.c

```
#define _CRT_SECURE_NO_WARNINGS
#include "Prim.h"
#include "AdjacencyMatrix.h"
#include "Status.h"
#include <stdio.h>
#include <malloc.h>

void MiniSpanTree_PRIM(MGraph G, VertexType u)
{
    int j = 0;
    int k = LocateVex(G, u);
    Edge closedge;
    for (j = 0; j < G.vexnum; ++j)//辅助数组初始化
        if (j != k)
        {
            closedge[j].adjvex = u;
            closedge[j].lowcost = G.arcs[k][j].adj;
        }//if
    closedge[k].lowcost = 0;//初始，U={u}
    printf("最小代价生成树的各条边为:\n");
    for (int i = 1; i < G.vexnum; ++i)
    {
        //选择其余G.vexnum-1个顶点
        k = minimum(closedge,G);//求出T的下一个结点：第k顶点
        //此时closegde[k].lowcost=MIN{closedge[vi].lowcost|closedge[vi].lowcost>0}
        printf("%d %d\n",closedge[k].adjvex, G.vexs[k]);//输出边
        closedge[k].lowcost = 0;//第k顶点并入U集
```

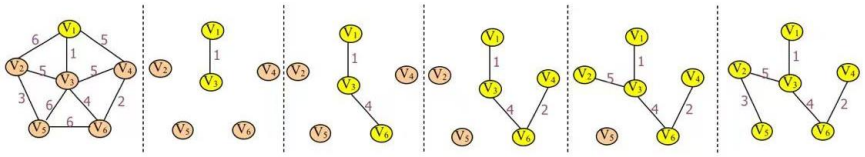
```
        for(int j=0;j<G.vexnum;++j)
            if (G.arcs[k][j].adj < closedge[j].lowcost)
            {
                //新顶点并入U后重新选择最小边
                closedge[j].adjvex = G.vexs[k];
                closedge[j].lowcost = G.arcs[k][j].adj;
            }//if
    }//for
}//MiniSpanTree

int minimum(Edge SZ, MGraph G)
{
    //求closedge.lowcost的最小正值
    int i = 0;
    int j, k, min;
    while (!SZ[i].lowcost)
        i++;
    min = SZ[i].lowcost; //第一个不为0的值
    k = i;
    for (j = i + 1; j < G.vexnum; j++)
        if (SZ[j].lowcost > 0)
            if (min > SZ[j].lowcost)
            {
                min = SZ[j].lowcost;
                k = j;
            }
    return k;
}
```

c. Test.c

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include "Prim.h"

int main()
{
    MGraph G;
    CreateUDN(&G);
    VertexType u = 1;
    MiniSpanTree_PRIM(G, u);

    system("pause");
    return 0;
}
```

```
E:\大二上\数据结构\代码保存\Prim\Debug\Prim.exe
输入图的顶点数 弧数:6 10
输入顶点值：1
输入顶点值：2
输入顶点值：3
输入顶点值：4
输入顶点值：5
输入顶点值：6
输入一条边的两端以及权值1 2 6
输入一条边的两端以及权值1 3 1
输入一条边的两端以及权值1 4 5
输入一条边的两端以及权值2 3 5
输入一条边的两端以及权值2 5 3
输入一条边的两端以及权值3 4 5
输入一条边的两端以及权值3 5 6
输入一条边的两端以及权值3 6 4
输入一条边的两端以及权值4 6 2
输入一条边的两端以及权值5 6 6
最小代价生成树的各条边为:
1 3
3 6
6 4
3 2
2 5
请按任意键继续. . .
```

- 普里姆 (Prim) 算法



顶点i / closedge	1	2	3	4	5	U	V-U	k
adjvex lowcost	v1 6	v1 1	v1 5			{v1}	{v2,v3,v4,v5,v6}	3
adjvex lowcost	v3 5	0	v1 5	v3 6	v3 4	{v1,v3}	{v2,v4,v5,v6}	6
adjvex lowcost	v3 5	0	v6 2	v3 6	0	{v1,v3,v6}	{v2,v4,v5 }	4
adjvex lowcost	v3 5	0	0	v3 6	0	{v1,v3,v6,v4}	{v2,v5 }	2
adjvex lowcost	0	0	0	v2 3	0	{v1,v3,v6,v4,v2}	{v5 }	5
adjvex lowcost	0	0	0	0	0	{v1,v3,v6,v4,v2,v5}	{}	

五．实验总结

1. 算法逻辑很难理解，调了很多次断点

   在输入 scanf 后要添加 getchar()吸收空格符

2. 在编写 DFS 算法时

   在调试阶段输入时发生错误

   以后循环输入时要回车

3.写 for 循环的时候要注意细节是否取等，初始化的值是 0 还是 1