数据结构与算法 第四次实验

学号：37220232203808　　姓名：宋浩元

一、 实验目的

1. 了解分别使用邻接矩阵与邻接表存储图的基础实现方法与原理，理解存储图的基本操作的代码编写方式
2. 了解分别使用 Prim 算法 Kruskal 算法的实现方法与原理，在理解的基础上学习代码编写
3. 学会灵活按照邻接矩阵与邻接表的存储方式自由编写存储结构代码
4. 通过实验探索图的深度优先搜索方法，理解回溯的意义

二、 实验内容

4-1 假设带权连通图 G 有 n 个顶点，用邻接矩阵 A[n][n]表示存储结构，u 为指定顶点的序号。试设计 Prim 算法，用于从顶点 u 出发构造连通图 G 的最小生成树。

- 图的基本操作包括在 MGraph.h 中

4-1.cpp | MGraph.cpp | MGraph.h

```
#ifndef   __MGraph_H__
#define   __MGraph_H__
typedef int VRType;     // 顶点关系类型
typedef char VertexType[20]; // 顶点类型
// 图的数组(邻接矩阵)存储表示
#define INFINITY 4270000 // 用整型最大值代替∞
#define MAX_VERTEX_NUM 20 // 最大顶点个数
typedef enum {DG,DN,UDG,UDN} GraphKind; // {有向图,有向网,无向图,无向网}

typedef struct {
    VRType adj; // 顶点关系类型。对无权图，用1(是)或0(否)表示相邻否；对带权图，则
} ArcCell,AdjMatrix[MAX_VERTEX_NUM][MAX_VERTEX_NUM]; // 二维数组

typedef struct {    // 图的数组(邻接矩阵)存储
    VertexType vexs[MAX_VERTEX_NUM]; // 顶点矢量
    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;

/*邻接矩阵的8个基本操作函数声明*/
int LocateVex(MGraph G,VertexType u);//若图G中存在顶点u，则返回该顶点在图中位置；
VertexType* GetVex(MGraph G,int v);// 根据图G中某个顶点的序号v，返回该顶点的值
void visit(VertexType i);// 访问输出顶点的值
int FirstAdjVex(MGraph G,VertexType v);// v是图G中某个顶点，返回v的第一个邻接顶点
int NextAdjVex(MGraph G,VertexType v,VertexType w);//v是图G中某个顶点，w是v的邻接
void CreateGraphF(MGraph &G);//采用数组(邻接矩阵)表示法，由文档构造无向网G
void DestroyGraph(MGraph &G);//销毁图G
void Display(MGraph G);//输出邻接矩阵存储表示的图G
#endif
```

- 记录从顶点集 U 到 V-U 的代价最小的边的辅助数组定义

```
typedef struct min { //记录从顶点集U到V-U的代价最小的边的辅助数组定义
    VertexType adjvex;
    VRType lowcost;
} minside[MAX_VERTEX_NUM];
```

- 求 SZ.lowcost 的最小正值，并返回其在 SZ 中的序号

```
int minimum(minside SZ,MGraph G) { //求SZ.lowcost的最小正值，并返回其在SZ中
    int i=0,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&&min>SZ[j].lowcost) { // 找到新的大于0的最小值
            min=SZ[j].lowcost;
            k=j;
        }
    return k;
}
```

- 用 Prim 算法从第 u 个顶点出发构造网 G 的最小生成树 T，输出 T 的各条边

```
void MiniSpanTree_PRIM(MGraph G,VertexType u) { // 用Prim算法从第u个顶点出发构造网G的最小
    int i,j,k;
    minside closedge;
    k=LocateVex(G,u);
    for(j=0; j<G.vexnum; ++j) {
        strcpy(closedge[j].adjvex,u);
        closedge[j].lowcost=G.arcs[k][j].adj;
    }
    closedge[k].lowcost=0;
    printf("最小代价生成树的各条边为:\n");
    for(i=1; i<G.vexnum; ++i) {
        k=minimum(closedge,G);
        printf("边(%s,%s),权值为%d\n",closedge[k].adjvex,G.vexs[k],closedge[k].lowcost);
        closedge[k].lowcost=0;
        for(j=0; j<G.vexnum; ++j)
            if(G.arcs[k][j].adj<closedge[j].lowcost) {
                strcpy(closedge[j].adjvex,G.vexs[k]);
                closedge[j].lowcost=G.arcs[k][j].adj;
            }
    }
}
```

- Main 函数创建无向图、构造最小生成树

```
int main() {
    MGraph g;
    int n;
    CreateGraphF(g);  // 利用数据文档创建无向图
    Display(g);       // 输出无向图
    printf("请输入构造最小生成树的起点: \n");
    scanf("%d",&n);
    printf("用普里姆算法从g的第%d个顶点出发输出最小生成树的各条边:\n",n);
    MiniSpanTree_PRIM(g,g.vexs[n]);  // Prim算法从第1个顶点构造最小生成树
    return 0;
}
```

4-2 采用邻接表存储结构，设计一个算法，判别无向图 G 中指定的两个顶点之间是否存在一条长度为 k 的简单路径。
注：简单路径是指顶点序列中不含有重复的顶点。

- 编写邻接表存储结构

```
#define MAX_VERTEX_NUM 30

typedef struct ArcNode {
    int adjvex;
    struct ArcNode *nextarc;
} ArcNode;

typedef struct VNode {
    int data;
    ArcNode *firstarc;
} VNode,AdjList[MAX_VERTEX_NUM];

typedef struct {
    AdjList vertices;
    int vexnum,arcnum;
} ALGraph;
```

- 创建图 存储为邻接表

```
void creat_DG_ALGraph(ALGraph *G) {
    int i,j,k;
    ArcNode *p,*q;
    p=q=NULL;
    printf("Please input vexnum,arcnum=:\n");
    scanf("%d %d",&G->vexnum,&G->arcnum);
    printf("Please input VNode:\n");
    for(i=0; i<G->vexnum; i++) {
        scanf("%d",&G->vertices[i].data);
        G->vertices[i].firstarc=NULL;
    }
//  for(i=0; i<G->vexnum; i++)
//      printf("%d ",G->vertices[i].data);
    printf("\n");
    for(k=0; k<G->arcnum; k++) {
        p=(ArcNode*)malloc(sizeof(ArcNode));
        q=(ArcNode*)malloc(sizeof(ArcNode));
        printf("please input edge <i,j>: \n");
        scanf("%d %d", &i, &j);
//      无向图
        p->adjvex = j;
        p->nextarc = G->vertices[i].firstarc;
        G->vertices[i].firstarc=p;
        q->adjvex = i;
        q->nextarc = G->vertices[j].firstarc;
        G->vertices[j].firstarc=q;
    }
}
```

- DFS 找路径

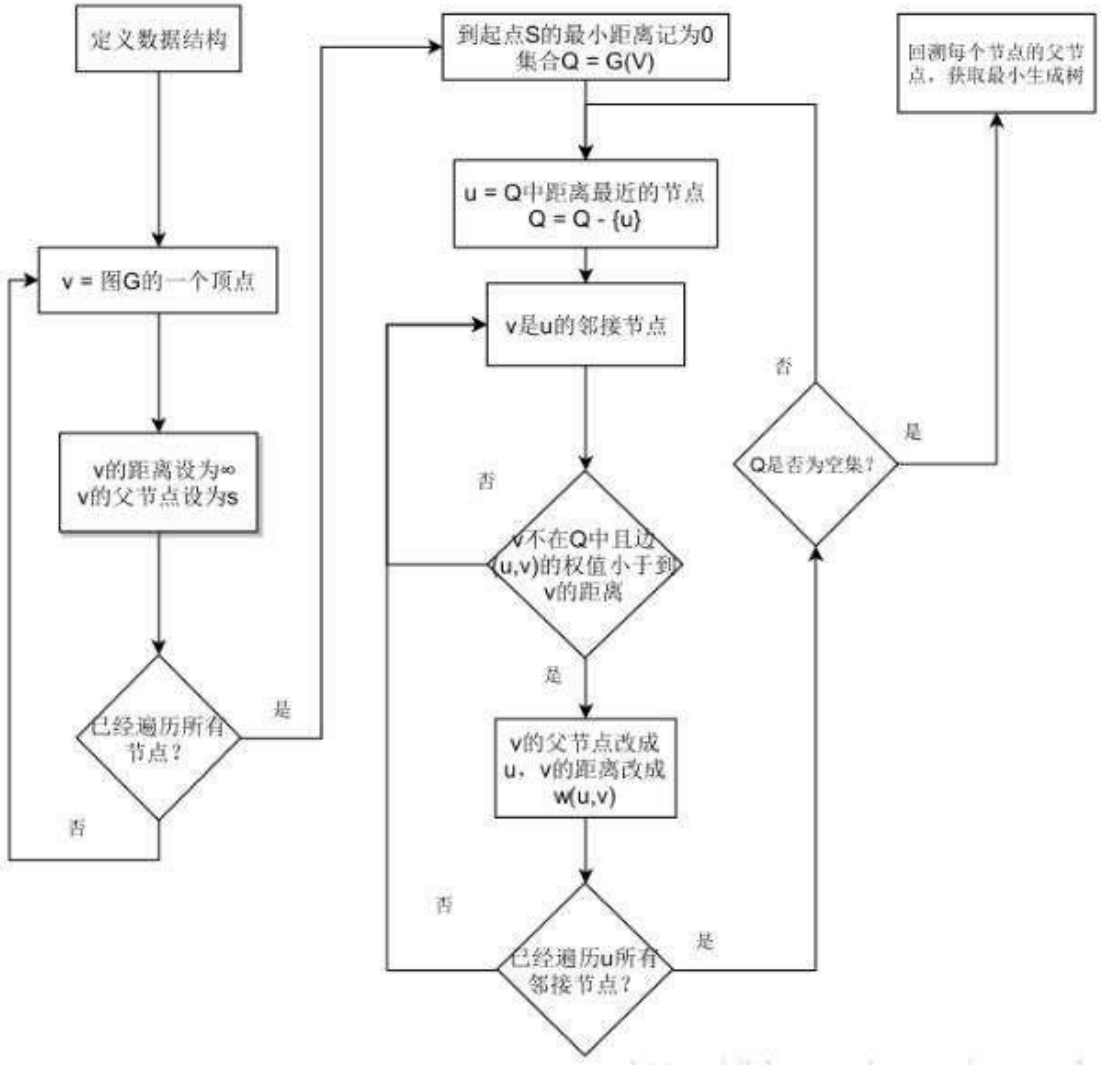
```
50  int visited[MAX_VERTEX_NUM]={0},length;
51  int exist_path_DFS(ALGraph G,int i,int j,int k) {
52      ArcNode *p;
53      int tmp;
54      p=NULL;
55      if(i==j&&k==length) return 1;
56      else {
57          visited[i]=1;
58          for(p=G.vertices[i].firstarc; p; p=p->nextarc) {
59              tmp=p->adjvex;
60              if(!visited[tmp]&&exist_path_DFS(G,tmp,j,k+1)) {
61                  return 1;
62              }
63          }
64          visited[i]=0;
65      }
66      return 0;
67  }
```

- 主函数交互

```
69  int main() {
70      ALGraph *G;
71      int i,j;
72      G=(ALGraph*)malloc(sizeof(ALGraph));
73      creat_DG_ALGraph(G);
74      while(1) {
75          memset(visited,0,sizeof(visited));
76          printf("Please input i->j and length k you want to find:\n");
77          scanf("%d %d %d",&i,&j,&length);
78          if(i==j&&i==0)return 0;
79          if(exist_path_DFS(*G,i,j,0)) printf("Exist the path!\n");
80          else printf("Not exist the path!\n");
81      }
82      return 0;
83  }
```

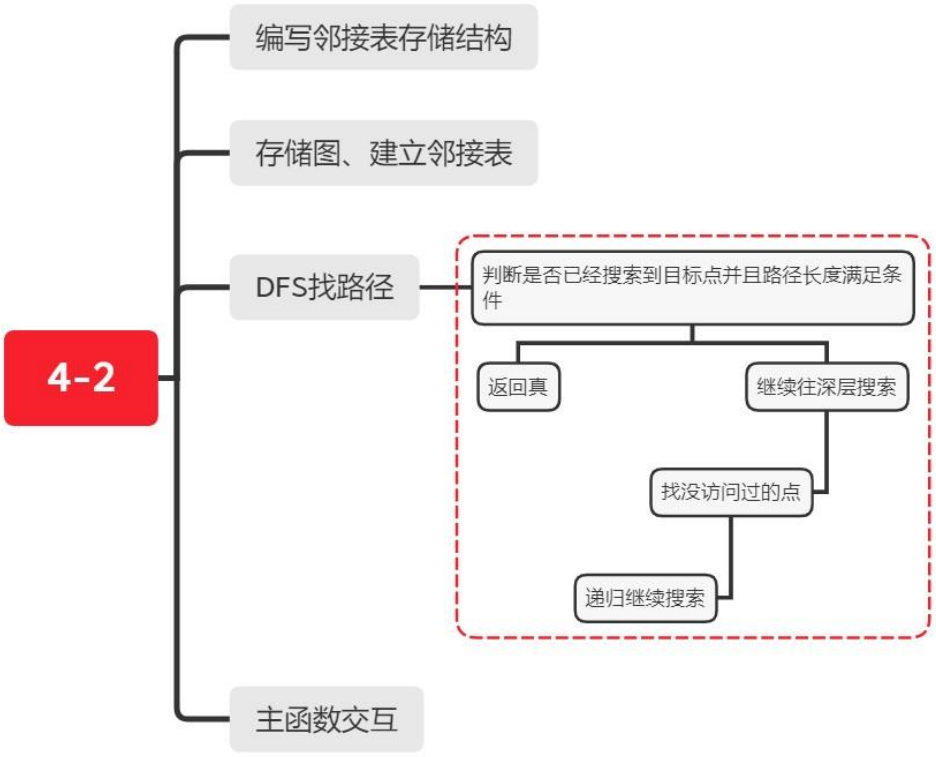
三、 主要算法流程图

4-1 算法流程



（图片太小可见目录下的 4-1 算法图.jpg）

4-2 算法流程



（可见目录下的 4-2 算法图.jpg）

四、 实验结果：

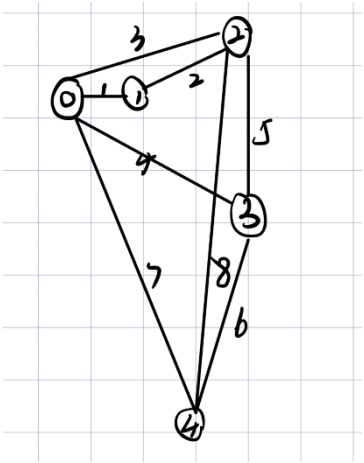
（结合截图说明算法的输入输出）

1、关于 4-1 的输入与输出：

```
请输入图的类型(有向图:0,有向网:1,无向图:2,带权连通图:3):
3
图的顶点的个数n:
5
图的边的条数m:
8
n个顶点的数据:
0
1
2
3
4
m条边的数据:
0 1 1
0 2 3
0 3 4
0 4 7
1 2 2
2 3 5
2 4 8
3 4 6
带权连通图
5个顶点8条边。顶点依次是: 0 1 2 3 4
图的邻接矩阵:
∞       1       3       4       7
1       ∞       2       ∞       ∞
3       2       ∞       5       8
4       ∞       5       ∞       6
7       ∞       8       6       ∞
请输入构造最小生成树的起点:
0
用普里姆算法从g的第0个顶点出发输出最小生成树的各条边:
最小代价生成树的各条边为:
边(0,1),权值为1
边(1,2),权值为2
边(0,3),权值为4
边(3,4),权值为6

--------------------------------
Process exited after 3.318 seconds with return value 0
请按任意键继续. . .
```

在实际运行中，我创建了如下的图，并生成了正确的最小生成树：



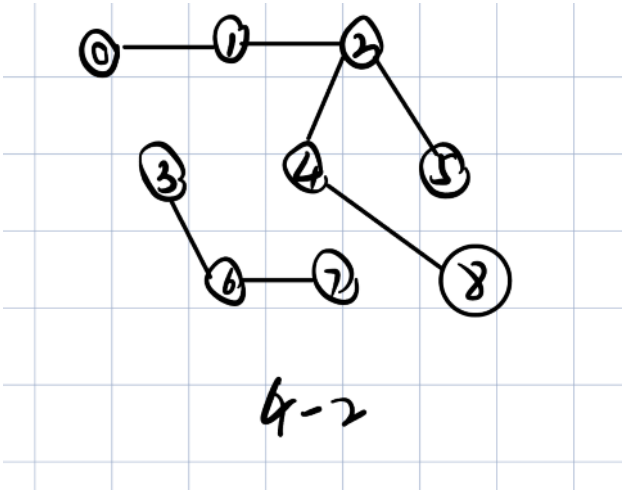
2、关于 4-2 的输入与输出

```
Please input vexnum,arcnum=:
9 7
Please input VNode:
0 1 2 3 4 5 6 7 8

please input edge <i,j>:
0 1
please input edge <i,j>:
1 2
please input edge <i,j>:
2 4
please input edge <i,j>:
2 5
please input edge <i,j>:
4 8
please input edge <i,j>:
3 6
please input edge <i,j>:
6 7
Please input i->j and length k you want to find:

0 1 2
Not exist the path!
Please input i->j and length k you want to find:
1 3 3
Not exist the path!
Please input i->j and length k you want to find:
1 8 3
Exist the path!
Please input i->j and length k you want to find:
3 7 2
Exist the path!
Please input i->j and length k you want to find:
0 0 0
```

在实际运行中，我创建了如下的图，并找到了正确的路径:



五、 实验小结（即总结本次实验所得到的经验与启发等）:

在本次实验中，我尝试具体运用了图，在实体机的实验中我能够更深刻地理解对这一部分数据结构的执行方式与特点，并且在编写代码的过程中，我通过不断的调试去寻找语句之间的问题和不足，在潜移默化中提高了我的代码编写能力，这是一次完成效果良好的实验！