河南工业大学_数据结构_实验报告

课程名称 数据结构 实验项目 实验四　图的典型算法实现

院　　系 信息学院计科系 专业班级 计科 xxxx 班

姓　　名 xxxxx 学　　号 xxxxxxxxxxxxx

指导老师 xxxxxx 日　　期 2020.11.16

批改日期 成　　绩

一 实验目的

1．掌握图的相关概念。

2．掌握用邻接矩阵和邻接表的方法描述图的存储结构。

3．掌握图的深度优先搜索和广度优先搜索遍历的方法及其计算机的实现。

4．理解最小生成树的有关算法

二 实验内容及要求

实验内容：

选择邻接矩阵或邻接链表存储结构，掌握图的创建、遍历、最小生成树、拓扑排序、关键路径、最短路径等典型操作。

1．编程实现如下功能：

（1）输入有向图的顶点数、边数及各条边的顶点对，建立用邻接表存储的有向图。

（2）输出有向图的邻接表

（3）对有向图进行深度优先搜索和广度优先搜索遍历，并分别输出其遍历序列。

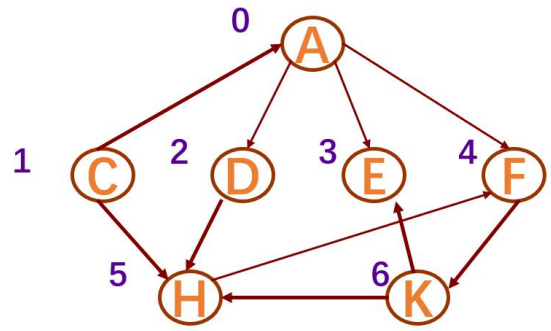
```
//-----图的邻接表存储表示-----
typedef char VerTexType;                    //假设顶点的数据类型为字符型
typedef struct ArcNode{                     //边结点
    int adjvex;                             //该边所指向的顶点的位置
    struct ArcNode *nextarc;                //指向下一条边的指针
}ArcNode;

typedef struct VNode{
    VerTexType data;                        //顶点信息
    ArcNode *firstarc;                      //指向第一条依附该顶点的边的指针
}VNode, AdjList[MVNum];                     //AdjList 表示邻接表类型

typedef struct{
    AdjList vertices;                       //邻接表
    int vexnum, arcnum;                     //图的当前顶点数和边数
}ALGraph;
//----队列的定义
```

```
typedef struct
{
   int *base;          //初始化的动态分配存储空间，注意队列中存储的是顶点的位置
   int front;          //头指针，若队列不空，指向队头元素
   int rear;           //尾指针，若队列不空，指向队尾元素的下一个位置
}sqQueue;
//队列的相关操作
void InitQueue(sqQueue &Q)
{//构造一个空队列 Q
  Q.base = new VerTexType [MAXQSIZE];
  if(!Q.base)       exit(1);      //存储分配失败
  Q.front = Q.rear = 0;
}//InitQueue
#define MAXQSIZE 100          //最大队列长度
void EnQueue(sqQueue &Q, int e){
  //插入元素 e 为 Q 的新的队尾元素
  if((Q.rear + 1) % MAXQSIZE == Q.front)
    return;
  Q.base[Q.rear] = e;
  Q.rear = (Q.rear + 1) % MAXQSIZE;
}//EnQueue

bool QueueEmpty(sqQueue Q){
  //判断是否为空队
  if(Q.rear == Q.front)
    return true;
  return false;
}//QueueEmpty

void DeQueue(sqQueue &Q, int &u){
  //队头元素出队并置为 u
  u = Q.base[Q.front];
  Q.front = (Q.front + 1) % MAXQSIZE;
}//DeQueue
```

提示：

测试，可以仿照下图和运行结果

有向图如下图所示：



其中 0、1、2。。。6 代表顶点的存储位置下标

A、C、D 等表示顶点名称

输入信息建议如下图所示：

```
请输入总顶点数，总边数，以空格隔开:7 10

输入点的名称，如a
请输入第1个点的名称:A
请输入第2个点的名称:C
请输入第3个点的名称:D
请输入第4个点的名称:E
请输入第5个点的名称:F
请输入第6个点的名称:H
请输入第7个点的名称:K

输入边依附的顶点，如a b
请输入第1条边依附的顶点:CA
请输入第2条边依附的顶点:CH
请输入第3条边依附的顶点:AD
请输入第4条边依附的顶点:AE
请输入第5条边依附的顶点:AF
请输入第6条边依附的顶点:DH
请输入第7条边依附的顶点:KE
请输入第8条边依附的顶点:FK
请输入第9条边依附的顶点:HF
请输入第10条边依附的顶点:KH
```

输出结果：

有向图G创建完成！

有向图的邻接表如下：
A F E D
C H A
D H
E
F K
H F
K H E
请输入遍历有向图的起始点：C

深度优先搜索遍历有向图结果：
C H F K E A D
广度优先搜索遍历有向图结果：
C H A F E D K

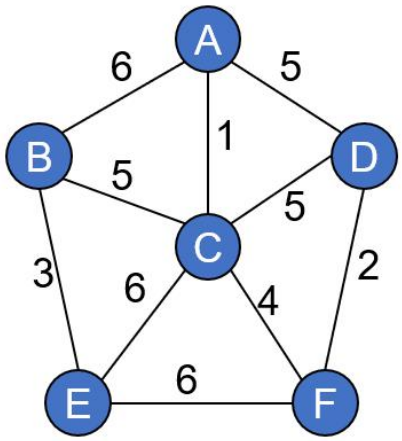
邻接矩阵的输出语句建议如下：　　cout<<setw(7)<<G.arcs[j][k];

要求：能输入不同的起点，得到相应的 DFS 和 BFS 的结果

2. 最小生成树

采用邻接矩阵的存储结构，编写 prim 算法完成最小生成树，输出最小生成树的代价之和

```
typedef char VerTexType;                    //假设顶点的数据类型为字符型
typedef int ArcType;                        //表示边的权值，类型为整形
typedef struct{
    VerTexType vexs[MVNum];                 //顶点表
    ArcType arcs[MVNum][MVNum];             //邻接矩阵
   int vexnum,arcnum;                       //图的当前点数和边数
}AMGraph;
  //辅助数组的定义，用来记录从顶点集 U 到 V-U 的权值最小的边
  Struct
  {
   VerTexType adjvex;          //最小边在 U 中的那个顶点
   ArcType lowcost;          //最小边上的权值
  }closedge[MVNum];
```



输入信息建议如下图所示：

```
请输入总顶点数，总边数，以空格隔开:6 10

输入点的名称，如a
请输入第1个点的名称:A
请输入第2个点的名称:B
请输入第3个点的名称:C
请输入第4个点的名称:D
请输入第5个点的名称:E
请输入第6个点的名称:F

输入边依附的顶点及权值，如AB 5
请输入第1条边依附的顶点及权值:AB 6
请输入第2条边依附的顶点及权值:AC 1
请输入第3条边依附的顶点及权值:AD 5
请输入第4条边依附的顶点及权值:BC 5
请输入第5条边依附的顶点及权值:BE 3
请输入第6条边依附的顶点及权值:CD 5
请输入第7条边依附的顶点及权值:CE 6
请输入第8条边依附的顶点及权值:CF 4
请输入第9条边依附的顶点及权值:DF 2
请输入第10条边依附的顶点及权值:EF 6
```

输出结果：

```
无向网G的邻接矩阵如下：
输出无向网的邻接矩阵
  32767      6      1      5  32767  32767
      6  32767      5  32767      3  32767
      1      5  32767      5      6      4
      5  32767      5  32767  32767      2
  32767      3      6  32767  32767      6
  32767  32767      4      2      6  32767
请输入起点：C
******利用普里姆算法构造最小生成树结果：******
边   C--->A       1
边   C--->F       4
边   F--->D       2
边   C--->B       5
边   B--->E       3

最小生成树的代价之和=15
```

要求：能输入不同的起点

实验要求：

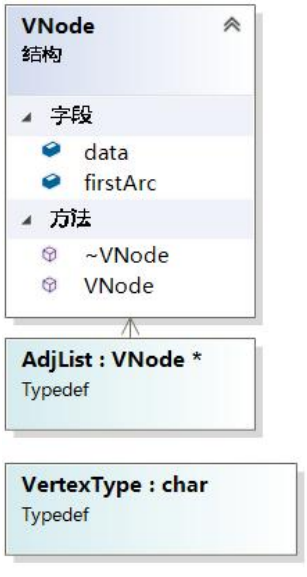
1. 键盘输入数据，并给出必要的输入提示，当输入起始顶点信息有误（即图中查无此顶点时，能提示继续输入）
2. 屏幕输出运行结果。
3. 要求记录实验源代码及运行结果。
4. 运行环境：CodeBlocks/Dev c++/VC6.0 等 C 编译环境

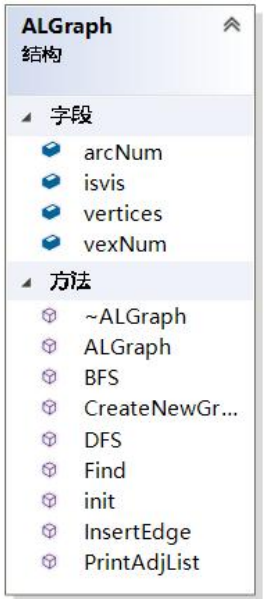
三 实验过程及运行结果

1. 编程实现如下功能：输入有向图的顶点数、边数及各条边的顶点对，建立用邻接表存储的有向图，输出有向图的邻接表，对有向图进行深度优先搜索和广度优先搜索遍历，并分别输出其遍历序列

(1) 算法设计思路

类图：





```
const int Null = -1;     //顶点类型
const int MVNum = 1000; //最大顶点数
typedef char VertexType;
struct VNode;
struct ArcNode {     //边结点定义
    int vexindex;    //该边所指向的顶点位置下标
    ArcNode* nextArc;    //指向下一条边的指针
    ArcNode() :vexindex(Null), nextArc(nullptr) {}  //用构造函数进行初始化
};
typedef ArcNode* ArcPtr;

struct VNode {
    VertexType data;//顶点数据类型
    ArcPtr firstArc;     //指向第一条依附该顶点的边的指针
    VNode() :firstArc(nullptr) {}
    ~VNode();
};

typedef VNode* AdjList;
struct ALGraph {
    int vexNum;      //该图顶点数
    int arcNum;      //该图边数
    bool isvis[MVNum];       // DFS/BFS中是否被访问过
    AdjList vertices;    //邻接表
    ALGraph() :vexNum(0), arcNum(0), vertices(nullptr) { init(); }  //构造函数初始化
    ~ALGraph(); //构造函数初始化
    int Find(VertexType val);    //查找值为val的顶点下标 没有则返回-1
    void CreateNewGraph();  //输入有向图的顶点数、边数及各条边的顶点对
    void InsertEdge(VNode& u, int vindex);  //插入边<u,v>
    void PrintAdjList();     //展示邻接表
    void init();         //遍历前的初始化操作，将isvis全部置否
    void DFS(int v);     //以顶点v为起点进行dfs遍历
    void BFS(int v);     //以顶点v为起点进行bfs遍历
};
```

(2) 源程序代码

1. 构造函数和析构函数

构造函数都内联实现，要注意的就是指针的初始化，核心代码如下

```
ALGraph() :vexNum(0), arcNum(0), vertices(nullptr) { init(); }  //构造函数初始化
ArcNode() :vexindex(Null), nextArc(nullptr) {}  //用构造函数进行初始化
VNode() :firstArc(nullptr) {}
ALGraph::~ALGraph() {
    if (vertices)
        delete[] vertices;
}
VNode::~VNode() {
    if (firstArc) {
        while (firstArc->nextArc) {
            ArcPtr p = firstArc->nextArc;
            while (p->nextArc)
                p = p->nextArc;
            delete p;
        }
        delete firstArc;
    }
}
```

2. 根据顶点的值寻找顶点在顶点表中的对应下标（Find）

核心思想：没找到则返回-1，因为我们输入顶点时是直接输入顶点类型的数据而不是下标，故需要这个函数进行查找返回下标。

核心代码：

```
//查找值为val的顶点下标 没有则返回-1
int ALGraph::Find(VertexType val) {
    for (int i = 0; i < vexNum; ++i) {
        if (vertices[i].data == val) return i;
    }
    return -1;
}
```

3.创建新图（CreateNewGraph）

核心思想：输入顶点数、边数，根据顶点数给顶点表分配适当空间，输入其他数据，插入有向边。

核心代码：

```
//输入顶点数、边数与顶点和边创建一个新图
void ALGraph::CreateNewGraph() {
    cout << "请输入总顶点数、总边数,以空格隔开:";
    cin >> vexNum >> arcNum;
    getchar();
    cout << endl;
    if (vertices) delete[] vertices;
    vertices = new VNode[vexNum];
    cout << "输入点的名称,如A" << endl;
    for (int i = 0; i < vexNum; ++i) {
        cout << "请输入第" << i + 1 << "个结点的名称:";
        cin >> vertices[i].data;
        getchar();
    }
    cout << endl;
    cout << "输入每个有向边两端的结点，空格间隔，如AB" << endl;
    for (int i = 0; i < arcNum; ++i) {
        VertexType s, e;
        int u, v;
        u = v = -1;
        cout << "请输入第" << i + 1 << "条边两端的顶点:";
        cin >> s >> e;
        getchar();
        for (int j = 0; j < vexNum; ++j) {
            if (vertices[j].data == s)
                u = j;
            else if (vertices[j].data == e)
                v = j;
        }
        InsertEdge(vertices[u], v);
    }
    cout << endl;

    cout << "有向图创建完成！" << endl;
}
```

4.插入有向边<u，v>（InsertEdge）

核心思想：因为是有向边，所以只需将顶点 v（终点）加入到顶点 u（始点）的邻接链表中~注意判断第一个边界点是否为空

核心代码：

```
//插入边<u,v>
void ALGraph::InsertEdge(VNode& u, int vindex) {
    //将v加入u的邻接链表中
    if (!u.firstArc) {  //结点u暂时没有邻接边
        u.firstArc = new ArcNode;
        u.firstArc->vexindex = vindex;
        u.firstArc->nextArc = nullptr;
    }
    else {
        ArcPtr temp = new ArcNode;
        temp->vexindex = vindex;
        temp->nextArc = u.firstArc;
        u.firstArc = temp;
    }
}
```

5.打印邻接表（PrintAdjList）

核心思想：输出该图的邻接表，简单的链表遍历

核心代码：

```
void ALGraph::PrintAdjList() {
    cout << "有向图的邻接表如下:" << endl;
    for (int i = 0; i < vexNum; ++i) {
        cout << setw(7) << vertices[i].data << "顶点  ->  ";
        ArcPtr p = vertices[i].firstArc;
        if (p) {
            cout << vertices[p->vexindex].data;
            while (p->nextArc) {
                p = p->nextArc;
                cout << setw(7) << vertices[p->vexindex].data;
            }
        }
        cout << endl;
    }
    cout << endl;
}
```

6.深度优先搜索遍历该图（DFS）

核心思想：需要标记结点是否访问过，所以应先初始化，然后开始遍历，对每个邻接点一路遍历下去

核心代码：

```
void ALGraph::init() {
    for (int i = 0; i < vexNum; ++i)
        isvis[i] = false;
}
void ALGraph::DFS(int v) {
    VNode& vex = vertices[v];
    cout << setw(7) << vex.data;
    isvis[v] = true;
    ArcPtr p = vex.firstArc;
    while (p) {
        int w = p->vexindex;
        if (!isvis[w])    //若这个邻接点未访问过
            DFS(w);
        p = p->nextArc;
    }
}
```

7.广度优先搜索遍历该图（BFS）

核心思想：同 DFS 一样的是也应先初始化，然后开始遍历，用队列暂存邻接点，对每个结点先访问，再将每个邻接点存到队列中。

核心代码：

```
void ALGraph::BFS(int v) {
    queue<int> q;//存每次访问的下标
    isvis[v] = true;
    q.push(v);
    while (!q.empty()) {
        int t = q.front();
        q.pop();
        VNode& vex = vertices[t];
        cout << setw(7) << vex.data;
        ArcPtr p = vex.firstArc;
        while (p) {
            int w = p->vexindex;
            if (!isvis[w]) {  //若这个邻接点未访问过 加入队列
                isvis[w] = true;
                q.push(w);
            }
            p = p->nextArc;
        }
    }
}
```

(3) 运行结果截图

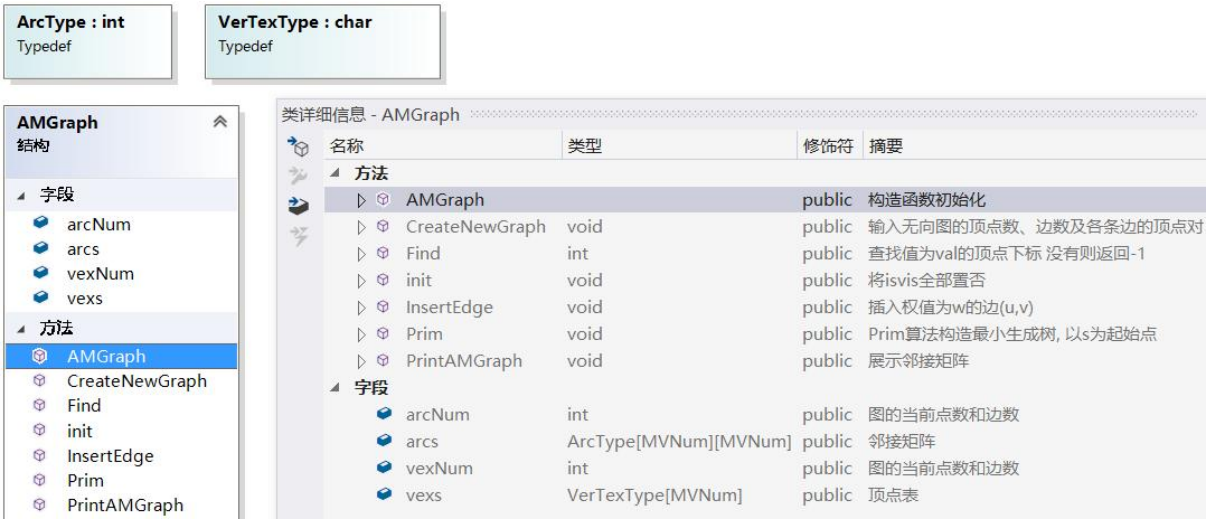
```
选择E:\Programming\数据结构\class\实验\计科201916010728余思娴实验4\Graph_Traversal
请输入总顶点数、总边数,以空格隔开:7 10

输入点的名称,如A
请输入第1个结点的名称:A
请输入第2个结点的名称:C
请输入第3个结点的名称:D
请输入第4个结点的名称:E
请输入第5个结点的名称:F
请输入第6个结点的名称:H
请输入第7个结点的名称:K

输入每个有向边两端的结点，空格间隔，如AB
请输入第1条边两端的顶点:CA
请输入第2条边两端的顶点:CH
请输入第3条边两端的顶点:AD
请输入第4条边两端的顶点:AE
请输入第5条边两端的顶点:AF
请输入第6条边两端的顶点:DH
请输入第7条边两端的顶点:KE
请输入第8条边两端的顶点:FK
请输入第9条边两端的顶点:HF
请输入第10条边两端的顶点:KH

有向图创建完成!
有向图的邻接表如下:
      A顶点  ->  F      E      D
      C顶点  ->  H      A
      D顶点  ->  H
      E顶点  ->
      F顶点  ->  K
      H顶点  ->  F
      K顶点  ->  H      E
```

```
请输入遍历有向图的起始点:C

深度优先搜索遍历有向图结果:
      C      H      F      K      E      A      D
广度优先搜索遍历有向图结果:
      C      H      A      F      E      D      K

请输入遍历有向图的起始点:F

深度优先搜索遍历有向图结果:
      F      K      H      E
广度优先搜索遍历有向图结果:
      F      K      H      E

请输入遍历有向图的起始点:K

深度优先搜索遍历有向图结果:
      K      H      F      E
广度优先搜索遍历有向图结果:
      K      H      E      F

请输入遍历有向图的起始点:A

深度优先搜索遍历有向图结果:
      A      F      K      H      E      D
广度优先搜索遍历有向图结果:
      A      F      E      D      K      H

请输入遍历有向图的起始点:E

深度优先搜索遍历有向图结果:
      E
广度优先搜索遍历有向图结果:
      E

请输入遍历有向图的起始点:D

深度优先搜索遍历有向图结果:
      D      H      F      K      E
广度优先搜索遍历有向图结果:
      D      H      F      K      E
```

2 编程实现如下功能：采用邻接矩阵的存储结构，编写 prim 算法完成最小生成树，输出最小生成树的代价之和

(1) 算法设计思路

类图：




```
const int inf = 32767;     //无穷大
const int Null = -1;
const int MVNum = 500;     //最大顶点数
typedef char VerTexType;   //假设顶点的数据类型为字符型
typedef int ArcType;       //表示边的权值，类型为整形
//辅助数组的定义，用来记录从顶点集U到V-U的权值最小的边
bool isvis[MVNum];     //顶点i是否被收录过
ArcType dist[MVNum];   //到该节点路径最短的路径长度
int fa[MVNum];         //到该节点路径最短的顶点下标
```

(2) 源程序代码

1. 构造函数和析取函数

核心思想：初始化邻接矩阵为全为最大值 inf,并将辅助数组全部初始化

核心代码：

```
AMGraph() : vexNum(0), arcNum(0) { //构造函数初始化
    for (int i = 0; i < MVNum; ++i) {
        for (int j = 0; j < MVNum; ++j) {
            arcs[i][j] = inf;
        }
        dist[i] = inf;
        fa[i] = Null;
        isvis[i] = false;
    }
};
```

2.根据顶点的值寻找顶点在顶点表中的对应下标（Find）

核心思想：没找到则返回-1，因为我们输入顶点时是直接输入顶点类型的数据而不是下标，故需要这个函数进行查找返回下标。

核心代码：

```
int AMGraph::Find(VerTexType val) {
    for (int i = 0; i < vexNum; ++i) {
        if (vexs[i] == val) {
            return i;
        }
    }
    return Null;
}
```

3.创建新图（CreateNewGraph）

核心思想：输入顶点数、边数和点的名称，将顶点存在顶点表里，输入其他数据，插入无向边。

核心代码:

```
void AMGraph::CreateNewGraph() {
    cout << "请输入总顶点数、总边数,以空格隔开:";
    cin >> vexNum >> arcNum;
    getchar();
    cout << endl;
    cout << "输入点的名称,如A" << endl;
    for (int i = 0; i < vexNum; ++i) {
        cout << "请输入第" << i + 1 << "个结点的名称:";
        cin >> vexs[i];
        getchar();
    }
    cout << endl;
    cout << "输入每个无向边两端的结点名称及权值，空格间隔，如AB 5" << endl;
```

```
    for (int i = 0; i < arcNum; ++i) {
        VerTexType s, e;
        ArcType w;
        int u, v;
        u = v = -1;
        cout << "请输入第" << i + 1 << "条边两端的顶点及其权值:";
        cin >> s >> e >> w;
        getchar();
        InsertEdge(s, e, w);
    }
    cout << endl;
    cout << "有向图创建完成！" << endl;
}
```

4.插入无向边(u,v)（InsertEdge）

核心思想：邻接矩阵存储，因为是无向图，所以 u 到 v，v 到 u 都得存

核心代码:

```
void AMGraph::InsertEdge(VerTexType s, VerTexType e, ArcType w) {
    int u, v;
    for (int i = 0; i < vexNum; ++i) {
        if (vexs[i] == s)
            u = i;
        if (vexs[i] == e)
            v = i;
    }
    arcs[u][v] = w;
    arcs[v][u] = w;
}
```

5.展示邻接矩阵（PrintAMGraph）

核心思想：简单的遍历

核心代码:

```
void AMGraph::PrintAMGraph() {
    cout << "该无向图的邻接矩阵如下:" << endl;
    for (int i = 0; i < vexNum; ++i) {
        for (int j = 0; j < vexNum; ++j) {
            cout << setw(7) << arcs[i][j];
        }
        cout << endl;
    }
    cout << endl;
}
```

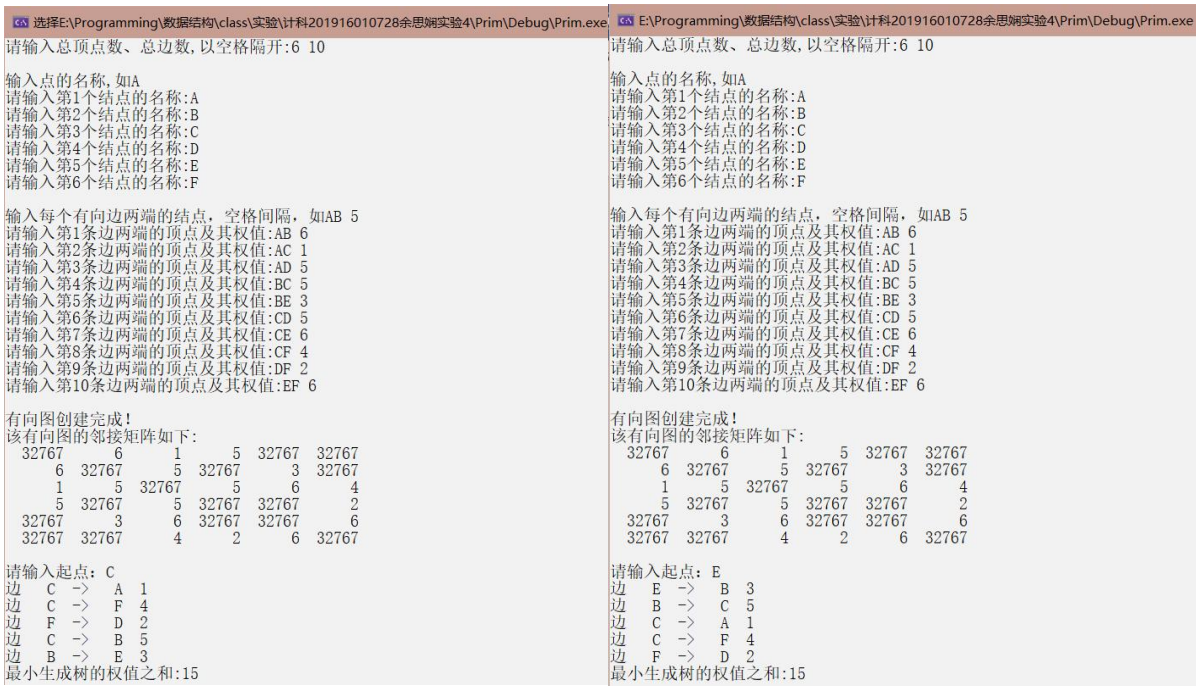
6.Prim 算法求解最小生成树（Prim）

核心思想：先初始化，传入一个结点 s 作为起始点，找其下标 si,先将这个结点收录（isvis 置为 true），将辅助数组初始化，做 n-1 次循环，每次找出未收录顶点中 dist 最小者（即距离起始点 s 距离最小），将其收录，并更新 dist 数组和 fa 数组。

核心代码:

```
void AMGraph::Prim(VerTexType s) {
    int ans = 0;
    int si = Find(s);
    init();
    isvis[si] = true;
    for (int i = 0; i < vexNum; ++i) {
        dist[i] = arcs[si][i];  //初始每个顶点到根结点s的距离为他们边，若无边则为最大值
        fa[i] = si;    //每个结点最短边的另一个端点
    }
    for (int i = 1; i <= vexNum - 1; ++i) {  // n个顶点的最小生成树需要找n-1条边
        int minw = inf;
        int v = Null;
        //找最小且未找过的顶点
        for (int j = 0; j < vexNum; ++j) {
            if (!isvis[j] && dist[j] < minw) {
                minw = dist[j];
                v = j;
            }
        }
        if (v != Null) {  //找到了最小的，收入
            isvis[v] = true;
            ans += dist[v];
            cout << "边   " << vexs[fa[v]] << "  ->   " << vexs[v] << "\t"
                 << dist[v] << endl;
            //更新距离dist,
            for (int j = 0; j < vexNum; ++j) {
                if (!isvis[j] && arcs[v][j] < dist[j]) {
                    dist[j] = arcs[v][j];
                    fa[j] = v;
                }
            }
        } else {
            cout << "未能找到最小生成树，该图不为连通图！" << endl;
            return;
        }
    }
    cout << "最小生成树的权值之和:" << ans << endl;
}
```

运行结果截图



```
选择E:\Programming\数据结构\class\实验\计科201916010728余思婉实验4\Prim\Debug\Prim.exe
请输入总顶点数、总边数,以空格隔开:6 10

输入点的名称,如A
请输入第1个结点的名称:A
请输入第2个结点的名称:B
请输入第3个结点的名称:C
请输入第4个结点的名称:D
请输入第5个结点的名称:E
请输入第6个结点的名称:F

输入每个有向边两端的结点，空格间隔，如AB 5
请输入第1条边两端的顶点及其权值:AB 6
请输入第2条边两端的顶点及其权值:AC 1
请输入第3条边两端的顶点及其权值:AD 5
请输入第4条边两端的顶点及其权值:BC 5
请输入第5条边两端的顶点及其权值:BE 3
请输入第6条边两端的顶点及其权值:CD 5
请输入第7条边两端的顶点及其权值:CE 6
请输入第8条边两端的顶点及其权值:CF 4
请输入第9条边两端的顶点及其权值:DF 2
请输入第10条边两端的顶点及其权值:EF 6

有向图创建完成!
该有向图的邻接矩阵如下:
  32767      6      1      5  32767  32767
      6  32767      5  32767      3  32767
      1      5  32767      5      6      4
      5  32767      5  32767  32767      2
  32767      3      6  32767  32767      6
  32767  32767      4      2      6  32767

请输入起点：C
边   C  ->   A  1
边   C  ->   F  4
边   F  ->   D  2
边   C  ->   B  5
边   B  ->   E  3
最小生成树的权值之和:15
```

```
E:\Programming\数据结构\class\实验\计科201916010728余思婉实验4\Prim\Debug\Prim.exe
请输入总顶点数、总边数,以空格隔开:6 10

输入点的名称,如A
请输入第1个结点的名称:A
请输入第2个结点的名称:B
请输入第3个结点的名称:C
请输入第4个结点的名称:D
请输入第5个结点的名称:E
请输入第6个结点的名称:F

输入每个有向边两端的结点，空格间隔，如AB 5
请输入第1条边两端的顶点及其权值:AB 6
请输入第2条边两端的顶点及其权值:AC 1
请输入第3条边两端的顶点及其权值:AD 5
请输入第4条边两端的顶点及其权值:BC 5
请输入第5条边两端的顶点及其权值:BE 3
请输入第6条边两端的顶点及其权值:CD 5
请输入第7条边两端的顶点及其权值:CE 6
请输入第8条边两端的顶点及其权值:CF 4
请输入第9条边两端的顶点及其权值:DF 2
请输入第10条边两端的顶点及其权值:EF 6

有向图创建完成!
该有向图的邻接矩阵如下:
  32767      6      1      5  32767  32767
      6  32767      5  32767      3  32767
      1      5  32767      5      6      4
      5  32767      5  32767  32767      2
  32767      3      6  32767  32767      6
  32767  32767      4      2      6  32767

请输入起点：E
边   E  ->   B  3
边   B  ->   C  5
边   C  ->   A  1
边   C  ->   F  4
边   F  ->   D  2
最小生成树的权值之和:15
```

四 调试情况、设计技巧及体会

此次实验未遇到太多困难，主要问题是要注意辅助数组的初始化，dist 数组要置为 inf，还有就是顶点名称和下标的转换，通过此次实验，我掌握了图的相关概念以及用邻接矩阵和邻接表的方法描述图的存储结构，还有图的深度优先搜索和广度优先搜索遍历的方法及其实现，理解了最小生成树的 Prim 算法。