Homework 11.18

PB19010450 和泳毅

```c
//----图的邻接表存储方式----
#define MAX_VERTEX_NUM  20
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
}ArcNode;
typedef struct VHode{
    VertexType      data;       //顶点信息
    ArcNode         *firstarc;  //指向第一条依附该顶点的弧的指针
}VNode, AdjList[MAX_VERTEX_NUM];
typedef struct{
    AdjList         vertices;
    int             vexnum,arcnum;//图的当前顶点数和弧数
    int             kind;       //图的种类标志
}ALGraph;
```

7.16 试在**邻接表**存储结构上实现图的基本操作: *InsertVex*(G,v)，*InsertArc*(G,v ,w)， *DeleteVex*(G,v)和*DeleteArc*(G,v,w)。

类C描述:

```c
//----处理有向图，无权值----
Status InsertVex(ALGraph &G,VertexType v){
//在有向图G中增添新顶点v
    if(!G) return ERROR;
    if(G.vexnum >= MAX_VERTEX_NUM)
        return ERROR;   //溢出
    G.vexnum++;         //顶点数+1
    G.vertices[G.vexnum].data = v;
    G.vertices[G.vexnum].firstarc = NULL;
    return OK;
}

Status InsertArc(ALGraph &G,VertexType v,VertexType w){
//在有向图G中增添弧<v,w>
    if(!G) return ERROR;
    k1 = LocateVex(G,v);
    k2 = LocateVex(G,w);            //找到顶点位置
    if(!k1 || !k2)  return ERROR;   //某顶点不存在
    p = (ArcNode *)malloc(sizeof(ArcNode));
    if(!p) exit(OVERFLOW);
    p->nextarc = NULL;
    p->adjvex = k2;                 //该弧指向w
    q = G.vertices[k1].firstarc;    //依赖v的第一个弧
    if(!q || q->adjvex > k2){       //弧不存在 或 依赖v的第一个弧指向顶点存储位置在w之后
```

```
        p->nextarc = G.vertices[k1].firstarc;
        G->vertices[k1].firstarc = p;
    }
    else{
        while(q->nextarc && q->adjvex < k2) //找到合适插入位置
            q = q->nextarc;
        p->nextarc = q->nextarc;
        q->nextarc = p;                     //插入
    }
    G.arcnum ++;                            //弧数+1
    return OK;
}

Status DeleteVex(ALGraph &G,VertexType v){
//删除有向图G中顶点v极其相关的弧
    if(!G) return ERROR;
    if(G.vexnum <= 0)   return  ERROR;  //已无顶点
    k = LocateVex(G,v);                 //找到顶点位置
    if(!k)  return ERROR;               //顶点不存在
    p = G.vertices[k].firstarc;         //依赖v的第一个弧
    while(p){                           //删除由此出发的弧
        q = p;
        p = p->nextarc;
        DeleteArc(G,v,G.vertices[q->adjvex].data);//删除以v为起点的弧
    }
    for(i = k+1; i <= G.vexnum; i++){   //移动存储位置
        G.vertices[i - 1].data = G.vertices[i].data;
        G.vertices[i - 1].firstarc = G.vertices[i].firstarc
    }
    G.vexnum--;                         //顶点数-1
    for(i = 1; i <= G.vexnum; i++){     //处理终点为v的弧
        p = G.vertices[i].firstarc;
        while(p && p->adjvex < k){      //遍历寻找指向该顶点的弧
            q = p;
            p = p->nextarc;
        }
        if(p &&  p->adjvex == k){
            if(p == G.vertices[i].firstarc;)//若是首弧
                G.vertices[i].firstarc = p->nextarc;
            else    q->nextarc = p->nextarc;
            free(p);
            G.arcnum--;                 //弧数-1
        }
    }
    return OK;
}

Status DeleteArc(ALGraph &G,VertexType v,VertexType w){
//在有向图G中删除弧<v,w>
    if(!G) return ERROR;
    if(G.arcnum <= 0)   return  ERROR;      //已无弧
    k1 = LocateVex(G,v);
    k2 = LocateVex(G,w);                    //找到顶点位置
    if(!k1 || !k2)  return ERROR;           //某顶点不存在
    p = G.vertices[k1].firstarc;            //依赖v的第一个弧
    if(p && p->adjvex == k2){               //若是首弧
        G.vertices[k1].firstarc = p->nextarc;
        free(p);
```

```
    }
    else{
        while(p && p->adjvex != k2){        //遍历寻找<v,w>
            q = p;
            p = p->nextarc;
        }
        if(!p)  return ERROR;               //无此弧
        else{
            q->nextarc = p->nextarc;
            free(p);
        }
    }
    G.arcnum--;                             //弧数-1
    return OK;
}
```

7.22 试基于图的**深度优先**搜索策略写一算法，判别以**邻接表**方式存储的**有向图**中是否存在由顶点v_i到顶点v_j的路径(i ≠ j)。注意：算法中涉及的图的基本操作必须在此存储结构上实现。

类C描述:

```
void DFS_FindPath(ALGraph G, int i,int j){
    for(v = FristAdjVex(G,G.vertices[i].data); v;
                    v = NextAdjVex(G,G.vertices[i].data,G.vertices[v].data))
{
        if(flag)    return;
        if(!visited[v]){
            visited[v] = TRUE;      //标记已访问
            if(v == j)  flag = TRUE;//找到终点顶点
            else    DFS(G,v,j);     //以新的起点DFS
        }
    }
}

Status FindPath(ALGraph G, int i,int j){
//判别有无vi->vj的路径，存在返回TRUE，反之FALSE
    if(i == j)  return ERROR;
    for(v = 1; v <= G.vexnum; v++)  //初始化访问数组
        visited[v] = FALSE;
    flag = FALSE;
    visited[i] = TRUE;              //排除vi顶点
    DFS_FindPath(G,i,j);
    if(flag)    return TRUE;
    else    return FALSE;
}
```