```
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
//   用邻接表表示图的类定义，包括顶点结点的类/结构定义、边结点类/结构定义和图类定义
//   Author：Melissa M.CAO
//   Belong：Section of software theory，School of Computer Engineering & Science，Shanghai University
//   Version： 1.0
//////////////////////////////////////////////////////////////////////////////
//////////////////////////

#pragma once

#include "SeqList.h"

template <class vertexType, class arcType> class AdjacencyListGraph;// 图的前置声明

//////////////////////////////////////////////////////////////////////////////
//////////////////////////
//边结点的类/结构定义
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
template < class arcType> struct ArcNode // 定义边（或弧）结点
{
    friend class AdjacencyListGraph <class vertexType, arcType>;
    int adjvex;  //和边（或弧）相关联的另一个顶点序号
    arcType weight; //边（或弧）上的信息（权）
    ArcNode<arcType> *nextarc ; //指向下一个边（或弧）结点的指针
    ArcNode( ) { }  //构造函数
    ArcNode( int v , arcType w ) : adjvex( v ) , weight( w ) , nextarc( NULL ){ }
    ArcNode( int v ) : adjvex( v ) , nextarc( NULL ){ }
    //构造函数
};

//////////////////////////////////////////////////////////////////////////////
//////////////////////////
//顶点结点的类/结构定义
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
template < class vertexType, class arcType> struct VertexNode // 定义顶点结点
{
    friend class AdjacencyListGraph <vertexType, arcType>;
    vertexType data; //顶点的信息
```

```
    ArcNode<arcType> *firstarc ; //指向依附该顶点的边链表的头
};

////////////////////////////////////////////////////////////////////////////////
//////////////////////////
//图类定义
////////////////////////////////////////////////////////////////////////////////
//////////////////////////
template <class vertexType, class arcType> class AdjacencyListGraph
{
    private:
        static const int MaxVertexes = 20;  //最大的顶点数

     //   VertexNode <arcType, vertexType> * VertexesTable; //顶点表
        SeqList<VertexNode<vertexType, arcType> > VertexesTable; //顶点表--边已经由每个结点的 firstarc 指定
        int CurrentNumVertexes;                         //当前的顶点数----该变量已经没有存在的必要，即为顶点表的长度
        int CurrentNumArcs;                             //当前的边（或弧）数
        int graphType;                                  //图类型：1 表示无向图，2 表示有向图
        int weightGraph;                                //图类型：1 表示无权值的图，2 表示带权图
        arcType edgeMaxValue;                           //带权图中无边时权值的最大值
        //-----------------------为求最短路径而设置----------------------------
        arcType * distarc;   //最短路径长度数组
        int *path;     //最短路径数组
        int *s;        //最短路径顶点集

    public:
        AdjacencyListGraph() : CurrentNumVertexes (0), CurrentNumArcs (0) {}
        AdjacencyListGraph(int type1, int type2);
        AdjacencyListGraph ( vertexType v[ ] , int num, int type1, int type2);
        ~AdjacencyListGraph ( );//析构函数
        void Display();     //显示图的基本信息
        //------------------------------        基        本        操        作
-----------------------------------------------
        int GetVertexPos( const vertexType &v );    // 取顶点 v 在数组中的位置
        int IsEmpty( ) const { return CurrentNumVertexes==0; }
        int NumberOfVertexes ( ) { return CurrentNumVertexes; }
        int NumberOfArcs ( ) { return CurrentNumArcs; }
        vertexType GetValue ( int v );
        arcType GetWeight ( int v1, int v2 );
        arcType GetWeight ( vertexType v1, vertexType v2 );
```

```
        void InsertArc( int v1,  int v2, arcType w, int insertpos );
        void InsertArc( int v1,  int v2, int insertpos );
        void InsertArc( vertexType v1,  vertex Type v2, arcType w, int insertpos );
        void InsertArc( vertexType v1,  vertexType v2, int insertpos );
        int GetFirstNeighbor ( int v );
        int GetNextNeighbor ( int v1,  int v2 );
        void DeleteArc ( int v1, int v2 );
        void DeleteVertex ( int v );
        int InsertVertex ( vertexType & v );

        int IsFull() const { return CurrentNumVertexs == MaxVertexes; }
        ArcNode<arcType>* GetAdj(int v);//获得指向序号 v 的结点的第一邻接边的指针
        ArcNode<arcType>*GetAdj(int v, int u);//获得指向序号 v 的结点的邻接边(v,u)
的指针
        //--------------------------        扩        展        操        作
-----------------------------------------------------------
        void Visit( vertexType v ); //定义的抽象的“访问/遍历”函数
        void DFS(const int v, int *visited, int *count);        //深度优先搜索
        void BFS(int v, int *visited, int *count);      //广度优先搜索
        void DFTraverse( );             //深度优先遍历
        void BFTraverse( );             //广度优先遍历
        void Prim(vertexType temp); //普里姆算法求最小生成树
        void ShortestPath (int type, int begin, vertexType &v);     //求单源点最
短路径
        void BellmanFord(int v);    //贝尔曼—福特算法
        bool IsConnected();         //判断图是否连通
        void Dijkstra(int v);       //迪杰斯特拉算法
        bool HavePostiveEdge();     //判断图中边的权值是否含有负数
        //--------------------------                  习                  题
-----------------------------------------------------------
        int ExistPath(vertexType vi, vertexType vj, int op, int l); //判断顶点
vi 与 vj 之间有无路径[是否相通]
        int DFS_ExistPath(vertexType vi, vertexType vj, int visited[]); //深度优
先搜索判断顶点 vi 与 vj 之间有无路径[是否相通]
        int BFS_ExistPath(vertexType vi, vertexType vj);    //广度优先搜索判断顶
点 vi 与 vj 之间有无路径[是否相通]
        int DFS_ExistPath(vertexType vi, vertexType vj, int L, int visited[]);
    //深度优先搜索判断顶点 vi 与 vj 之间有无长度为 L 的简单路径
};
```