

```

/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
// 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; //顶点表一边已经
    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
    //-----为拓扑排序而设置-----
    int * InDegree; //入度数组，记录每一个顶点的入度--增加后，一种方法是修改
    构造函数，构造时处理该成员，另一种方法是通过函数专门进行处理
    //此处采用第二种处理方式，下面就是定义的处理函数
    void InitialInDegree(int *begin, int *end); //初始化入度数组，顺带将源点和
    汇点序号放入 begin 和 end 中

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, vertexType 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 CurrentNumVertexes == 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 Visit( vertexType v, int mode, int *visited); //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 DFS(const int v, int *visited, int *count, int mode); //深度优
先搜索, 用于习题 31
void DFS(const int v, arcType *visited, int mode); //深度优先搜索,
用于 28 的习题
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
void TopologicalSort (int *nCycle); //拓扑排序算法
void DFSTopologicalSort (int *nCycle); //采用深度优先搜索的拓扑排序算法
bool IsDAG(); //判断是否为有向无环图
void Kruskal(); //最小生成树的克鲁斯卡尔算法
void Floyd(); //弗洛伊德(Floyd)算法

```

```
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
int GetRoot(); // 求有向无环图的根
void DFSCriticalPath(); // 关键路径算法
};
```