

```

/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
// 用邻接矩阵表示图的类定义
// 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 AdjacencyMatrixGraph
{
private:
    static const int MaxVertexes = 20;           //结点的最大数目
    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
    SeqList<vertexType> Vertexes;                //图的结点
    arcType Arcs[MaxVertexes][MaxVertexes];     //图的邻接矩阵
    int CurrentNumArcs;                          //边数
    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 * dist;                             //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
    //----- 为 求 最 小 生 成 树 而 设 置
    -----

public:
    //----- 基 本 操 作
    -----

    AdjacencyMatrixGraph(int num, int type1, int type2);
    AdjacencyMatrixGraph(int num, int type1, int type2, arcType max);

```

```

~AdjacencyMatrixGraph( ) {} ;

int FindVertex(vertexType &v)           //查找顶点 v 是否存在，返回结
点位置 (序号+1)
{ return Vertexes.Locate(v); }
int GetVertexPos (vertexType &v )       //取顶点 v 在数组中的位置
{ return Vertexes.Locate(v)-1; }
int IsEmpty()const {return Vertexes.IsEmpty();} //图是否为空
int NumberOfVertexes() {return Vertexes.Length();} //返回顶点数
int NumberOfArcs() {return CurrentNumArcs;} //返回边数
vertexType GetValue(int v) { return Vertexes.Get(v); } //返回指定位置上
(下标+1) 的顶点的值

void SetVertex(vertexType *a, int num); //设置顶点值
int InsertArc(vertexType v1, vertexType v2, arcType weight); //图插入
边
int InsertArc(vertexType v1, vertexType v2); //无权值
的图插入边
void Display(); //显示图信息
arcType GetWeight(vertexType v1, vertexType v2); //给出始点和终点的边的权
值
int DeleteArc(vertexType v1, vertexType v2); //删除始点和终点的边
int GetFirstNeighbor(vertexType v); //查找顶点 v 的第一条边
int GetNextNeighbor(vertexType v1, vertexType v2); //查找顶点 v1 与 v2 构成边
的下一条边
int InsertVertex(vertexType &v); //插入顶点 v，置于顺序表
的最后
int DeleteVertex(vertexType v); //删除顶点 v
//----- 扩展和图中各类算法实现的操作
-----

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 DFS(vertexType v, int *visited, int *count); //深度优先搜索
// void BFS(vertexType 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
bool IsConnected(); //判断图是否连通
bool HavePostiveEdge(); //判断图中边的权值是否含有负数
void ShortestPath (int type, int begin, vertexType &v); //求单源点最
短路径
void Dijkstra(int v); //迪杰斯特拉算法
void BellmanFord(int v); //贝尔曼—福特算法
void Floyd(); //弗洛伊德(Floyd)算法

```

```
void TopologicalSort ( );    //拓扑排序算法
void CriticalPathQuestion();    //有问题的关键路径算法
void CriticalPath();        //关键路径算法
void Kruskal();             //最小生成树的克鲁斯卡尔算法
void Prime(vertexType temp);    //最小生成树的普里姆算法
};
```