（一）

修改后的代码如下

template <class ElemType>

class AdjMatrixUndirGraph

{

protected:

// 邻接矩阵的数据成员:

int vexNum, vexMaxNum, arcNum; // 顶点数目、允许的顶点最大数目和边数

int \*\*arcs; // 存放边信息邻接矩阵

ElemType \*vertexes; // 存放顶点信息的数组

mutable Status \*tag; // 标志数组

public:

// 邻接矩阵类型的方法声明:

AdjMatrixUndirGraph(ElemType es[], int vertexNum, int vertexMaxNum = DEFAULT\_SIZE);

// 以数组es[]为顶点,顶点个数为vertexNum,允许的顶点最大数目为vertexMaxNum,边数为0的无向图

AdjMatrixUndirGraph(int vertexMaxNum = DEFAULT\_SIZE);

// 构造允许的顶点最大数目为vertexMaxNum,边数为0的无向图

~AdjMatrixUndirGraph(); // 析构函数

void Clear(); // 清空图

bool IsEmpty(); // 判断无向图是否为空

int GetOrder(ElemType &d) const;// 求顶点的序号

Status GetElem(int v, ElemType &d) const;// 求顶点的元素值

Status SetElem(int v, const ElemType &d);// 设置顶点的元素值

int GetVexNum() const; // 返回顶点个数

int GetArcNum() const; // 返回边数

int FirstAdjVex(int v) const; // 返回顶点v的第一个邻接点

int NextAdjVex(int v1, int v2) const; // 返回顶点v1的相对于v2的下一个邻接点

void InsertVex(const ElemType &d); // 插入元素值为d的顶点

void InsertArc(int v1, int v2); // 插入顶点为v1和v2的边

void DeleteVex(const ElemType &d); // 删除元素值为d的顶点

void DeleteArc(int v1, int v2); // 删除顶点为v1和v2的边

Status GetTag(int v) const; // 返回顶点v的标志

void SetTag(int v, Status val) const; // 设置顶点v的标志为val

AdjMatrixUndirGraph(const AdjMatrixUndirGraph<ElemType> &g); // 复制构造函数

AdjMatrixUndirGraph<ElemType> &operator =(const AdjMatrixUndirGraph<ElemType> &g);

// 赋值语句重载

void Display(); // 显示邻接矩阵无向图

};

// 无向图的邻接矩阵类的实现部分

template <class ElemType>

AdjMatrixUndirGraph<ElemType>::AdjMatrixUndirGraph(ElemType es[], int vertexNum, int vertexMaxNum)

// 操作结果：构造数据元素为es[],顶点个数为vertexNum,允许的顶点最大数目为vertexMaxNum,边数为0的无向图

{

if (vertexMaxNum < 0)

throw Error("允许的顶点最大数目不能为负!"); // 抛出异常

if (vertexMaxNum < vertexNum)

throw Error("顶点数目不能大于允许的顶点最大数目!");// 抛出异常

vexNum = vertexNum;

vexMaxNum = vertexMaxNum;

arcNum = 0;

vertexes = new ElemType[vexMaxNum]; // 生成生成顶点信息数组

tag = new Status[vexMaxNum]; // 生成标志数组

arcs = (int \*\*)new int \*[vexMaxNum]; // 生成邻接矩阵

for (int v = 0; v < vexMaxNum; v++)

arcs[v] = new int[vexMaxNum];

for (int v = 0; v < vexNum; v++) {

vertexes[v] = es[v];

tag[v] = UNVISITED;

for (int u = 0; u < vexNum; u++)

arcs[v][u] = 0;

}

}

template <class ElemType>

AdjMatrixUndirGraph<ElemType>::AdjMatrixUndirGraph(int vertexMaxNum)

// 操作结果：构造允许顶点的最大数目为vertexMaxNum的空无向图

{

if (vertexMaxNum < 0)

throw Error("允许的顶点最大数目不能为负!");// 抛出异常

vexNum = 0;

vexMaxNum = vertexMaxNum;

arcNum = 0;

vertexes = new ElemType[vexMaxNum]; // 生成生成顶点信息数组

tag = new Status[vexMaxNum]; // 生成标志数组

arcs = (int \*\*)new int \*[vexMaxNum]; // 生成邻接矩阵

for (int v = 0; v < vexMaxNum; v++)

arcs[v] = new int[vexMaxNum];

}

template <class ElemType>

void AdjMatrixUndirGraph<ElemType>::Clear()

// 操作结果：把无向图的顶点数和边数设置为0.

{

vexNum = 0;

arcNum = 0;

}

template <class ElemType>

bool AdjMatrixUndirGraph<ElemType>::IsEmpty()

// 操作结果：如果无向图为空返回true,否则返回false.

{

return vexNum == 0;

}

template <class ElemType>

AdjMatrixUndirGraph<ElemType>::~AdjMatrixUndirGraph()

// 操作结果：释放对象所占用的空间

{

delete []vertexes; // 释放顶点数据

delete []tag; // 释放标志

for (int v = 0; v < vexMaxNum; v++) // 释放邻接矩阵的行

delete []arcs[v];

delete []arcs; // 释放邻接矩阵

}

template <class ElemType>

int AdjMatrixUndirGraph<ElemType>::GetOrder(ElemType &d) const

// 操作结果：求顶点d的序号.顶点的序号从0开始，图中不存在顶点d时，返回-1.

{

for (int v = 0; v < vexNum; v++)

if (vertexes[v] == d)

return v; // 顶点d存在,返回它的序号

return -1; // 顶点d不存在,返回-1

}

template <class ElemType>

Status AdjMatrixUndirGraph<ElemType>::GetElem(int v, ElemType &d) const

// 操作结果：求序号为v的顶点值, v的取值范围为0 ≤ v ＜ vexNum, v合法时函数

// 通过d取得顶点值，并返回ENTRY\_FOUND；否则函数返回NOT\_PRESENT

{

if (v < 0 || v >= vexNum)

return NOT\_PRESENT; // v范围错,返回元素不存在

else {

d = vertexes[v]; // 将顶点v的元素值赋给d

return ENTRY\_FOUND; // 返回元素存在

}

}

template <class ElemType>

Status AdjMatrixUndirGraph<ElemType>::SetElem(int v, const ElemType &d)

// 操作结果：设置顶点的元素值v的取值范围为0 ≤ v ＜ vexNum, v合法时函数返回

// SUCCESS, 否则函数返回RANGE\_ERROR

{

if (v < 0 || v >= vexNum)

return RANGE\_ERROR; // v范围错,返回位置错

else {

vertexes[v] = d; // 将顶点元素的值改为d

return SUCCESS; // 返回修改成功

}

}

template <class ElemType>

int AdjMatrixUndirGraph<ElemType>::GetVexNum() const

// 操作结果：返回顶点个数

{

return vexNum;

}

template <class ElemType>

int AdjMatrixUndirGraph<ElemType>::GetArcNum() const

// 操作结果：返回边数

{

return arcNum;

}

template <class ElemType>

int AdjMatrixUndirGraph<ElemType>::FirstAdjVex(int v) const

// 操作结果：返回顶点v的第1个邻接点的序号

{

if (v < 0 || v >= vexNum)

throw Error("v不合法!");// 抛出异常

for (int u = 0; u < vexNum; u++)

if (arcs[v][u] != 0)

return u;

return -1; // 返回-1表示无邻接点

}

template <class ElemType>

int AdjMatrixUndirGraph<ElemType>::NextAdjVex(int v1, int v2) const

// 操作结果：返回顶点v1的相对于v2的下一个邻接点

{

if (v1 < 0 || v1 >= vexNum)

throw Error("v1不合法!"); // 抛出异常

if (v2 < 0 || v2 >= vexNum)

throw Error("v2不合法!"); // 抛出异常

if (v1 == v2) throw

Error("v1不能等于v2!"); // 抛出异常

for (int u = v2 + 1; u < vexNum; u++)

if (arcs[v1][u] != 0)

return u;

return -1; // 返回-1表示无下一个邻接点

}

template <class ElemType>

void AdjMatrixUndirGraph<ElemType>::InsertVex(const ElemType &d)

// 操作结果：插入顶点d

{

if (vexNum == vexMaxNum)

throw Error("图的顶点数不能超过允许的最大数!"); // 抛出异常

vertexes[vexNum] = d;

tag[vexNum] = UNVISITED;

for (int v = 0; v <= vexNum; v++) {

arcs[vexNum][v] = 0;

arcs[v][vexNum] = 0;

}

vexNum++;

}

template <class ElemType>

void AdjMatrixUndirGraph<ElemType>::InsertArc(int v1, int v2)

// 操作结果：插入依附顶点v1和v2的边

{

if (v1 < 0 || v1 >= vexNum)

throw Error("v1不合法!"); // 抛出异常

if (v2 < 0 || v2 >= vexNum)

throw Error("v2不合法!"); // 抛出异常

if (v1 == v2)

throw Error("v1不能等于v2!");// 抛出异常

if (arcs[v1][v2] == 0) { // 原无向图中没有边(v1, v2)

arcNum++;

arcs[v1][v2] = 1;

arcs[v2][v1] = 1;

}

}

template <class ElemType>

void AdjMatrixUndirGraph<ElemType>::DeleteVex(const ElemType &d)

// 操作结果：删除元素为d的顶点

{

int v;

for (v = 0; v < vexNum; v++)

if (vertexes[v] == d)

break;

if (v == vexNum)

throw Error("图中不存在要删除的顶点!"); // 抛出异常

for (int u = 0; u < vexNum; u++) // 删除与顶点d相关联的边

if (arcs[v][u] == 1) {

arcNum--;

arcs[v][u] = 0;

arcs[u][v] = 0;

}

vexNum--;

if (v < vexNum) {

vertexes[v] = vertexes[vexNum];

tag[v] = tag[vexNum];

for (int u = 0; u <= vexNum; u++)

arcs[v][u] = arcs[vexNum][u];

for (int u = 0; u <= vexNum; u++)

arcs[u][v] = arcs[u][vexNum];

}

}

template <class ElemType>

void AdjMatrixUndirGraph<ElemType>::DeleteArc(int v1, int v2)

// 操作结果：删除依附顶点v1和v2的边

{

if (v1 < 0 || v1 >= vexNum)

throw Error("v1不合法!"); // 抛出异常

if (v2 < 0 || v2 >= vexNum)

throw Error("v2不合法!"); // 抛出异常

if (v1 == v2)

throw Error("v1不能等于v2!");// 抛出异常

if (arcs[v1][v2] != 0) { // 原无向图存在边(v1, v2)

arcNum--;

arcs[v1][v2] = 0;

arcs[v2][v1] = 0;

}

}

template <class ElemType>

Status AdjMatrixUndirGraph<ElemType>::GetTag(int v) const

// 操作结果：返回顶点v的标志

{

if (v < 0 || v >= vexNum)

throw Error("v不合法!"); // 抛出异常

return tag[v];

}

template <class ElemType>

void AdjMatrixUndirGraph<ElemType>::SetTag(int v, Status val) const

// 操作结果：设置顶点v的标志为val

{

if (v < 0 || v >= vexNum)

throw Error("v不合法!"); // 抛出异常

tag[v] = val;

}

template <class ElemType>

AdjMatrixUndirGraph<ElemType>::AdjMatrixUndirGraph(const AdjMatrixUndirGraph<ElemType> &g)

// 操作结果：由无向图的邻接矩阵g构造新无向图的邻接矩阵g——复制构造函数

{

vexNum = g.vexNum;

vexMaxNum = g.vexMaxNum;

arcNum = g.arcNum;

vertexes = new ElemType[vexMaxNum]; // 生成顶点数据数组

tag = new Status[vexMaxNum]; // 生成标志数组

arcs = (int \*\*)new int \*[vexMaxNum]; // 生成邻接矩阵

for (int v = 0; v < vexMaxNum; v++)

arcs[v] = new int[vexMaxNum];

for (int v = 0; v < vexNum; v++) { // 复制顶点数据数组

vertexes[v] = g.vertexes[v];

tag[v] = g.tag[v];

for (int u = 0; u < vexNum; u++)

arcs[v][u] = g.arcs[v][u];

}

}

template <class ElemType>

AdjMatrixUndirGraph<ElemType> &AdjMatrixUndirGraph<ElemType>::operator =(const AdjMatrixUndirGraph<ElemType> &g)

// 操作结果：将无向图的邻接矩阵g赋值给当前无向图的邻接矩阵——赋值语句重载

{

if (&g != this) {

delete []vertexes; // 释放顶点数据

delete []tag; // 释放标志

for (int v = 0; v < vexMaxNum; v++) // 释放邻接矩阵的行

delete []arcs[v];

delete []arcs; // 释放邻接矩阵

vexNum = g.vexNum;

vexMaxNum = g.vexMaxNum;

arcNum = g.arcNum;

vertexes = new ElemType[vexMaxNum]; // 生成顶点数据数组

tag = new Status[vexMaxNum]; // 生成标志数组

arcs = (int \*\*)new int \*[vexMaxNum]; // 生成邻接矩阵

for (int v = 0; v < vexMaxNum; v++)

arcs[v] = new int[vexMaxNum];

for (int v = 0; v < vexNum; v++) { // 复制顶点数据数组

vertexes[v] = g.vertexes[v];

tag[v] = g.tag[v];

for (int u = 0; u < vexNum; u++)

arcs[v][u] = g.arcs[v][u];

}

}

return \*this;

}

template <class ElemType>

void AdjMatrixUndirGraph<ElemType>::Display()

// 操作结果: 显示邻接矩阵无向图

{

for (int v = 0; v < vexNum; v++)

cout << "\t" << vertexes[v];

cout << endl;

for (int v = 0; v < vexNum; v++) {

cout << vertexes[v];

for (int u = 0; u < vexNum; u++)

cout << "\t" << arcs[v][u];

cout << endl;

}

}

#endif

