类内声明：

int findParent(const vector<int> &parent, int v) const;

void SollinAlgorithm(vector<Edge<WeightType> >& mstEdges);

类外定义：

//Solin算法求最小生成树

template <class ElemType, class WeightType>

int AdjListUndirNetwork<ElemType, WeightType>::findParent(const vector<int> &parent, int v) const {

while (parent[v] != v) {

v = parent[v];

}

return v;

}

template <class ElemType, class WeightType> void AdjListUndirNetwork<ElemType, WeightType>::SollinAlgorithm(vector<Edge<WeightType> >& mstEdges) {

// 初始化组件（每个顶点是一个组件）

vector<int> parent(vexNum);

iota(parent.begin(), parent.end(), 0); // 用0至vexNum-1的连续整数填充parent

int numComponents = vexNum;

while (numComponents > 1) {

// 每个组件选择连接到其他组件的最小边

vector<Edge<WeightType> > cheapest(vexNum, Edge<WeightType>(-1, -1, (WeightType)DEFAULT\_INFINITY));

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

for (int w = FirstAdjVex(v); w >= 0; w = NextAdjVex(v, w)) {

int rootV = findParent(parent, v);

int rootW = findParent(parent, w);

if (rootV != rootW) {

WeightType weight = GetWeight(v, w);

if ((weight < cheapest[rootV].weight) || ((weight == cheapest[rootV].weight) && ((v < cheapest[rootV].v1) || ((v == cheapest[rootV].v1) && (w < cheapest[rootV].v2)))))

{

cheapest[rootV] = Edge<WeightType>(v, w, weight);

}

}

}

}

// 合并组件并添加边到MST

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

if (cheapest[i].v1 != -1) {

int rootV1 = findParent(cheapest[i].v1);

int rootV2 = findParent(cheapest[i].v2);

if (rootV1 != rootV2) {

mstEdges.push\_back(cheapest[i]);

parent[rootV1] = rootV2;

--numComponents;

}

}

}

}

}