实验-4-图-参考程序

王 新 宇
计算机科学与技术系

```
//读入数据建立图
template <class DataType, class WeightType>
bool LoadData(ALDirNetwork<DataType, WeightType> &graph) {
    ifstream fin("GraphData.txt");
    if(!fin) {
        cout << "File open error!" << endl;
        return false;
    }
    int vexnum,arcnum;
    DataType data;
    int v1, v2, weight;
    fin >> vexnum >> arcnum;          //读入顶点、弧数目
```

```
    for(int i=1; i<=vexnum; i++)  {     //顺序读入顶点信息
        fin >> data;
        graph.InsertVex(data);
    }
    for(int i=1; i<=arcnum; i++)  {     //顺序读入弧信息
        fin >> v1 >> v2 >> weight;
        graph.InsertArc(v1,v2,weight);
    }
    fin.close();
    return true;
}
```

```
//实验题第1题
//重载的DFS函数，供ExistPathDFS函数调用
template <class DataType,class WeightType>
void DFS(ALDirNetwork<DataType,WeightType> &graph, int v)  {
    graph.SetVisitedTag(v, VISITED); //设置顶点v 已访问标记
    for(int w = graph.GetFirstAdjvex(v); w != -1; w = graph.GetNextAdjvex(v, w))
    if(graph.GetVisitedTag(w) == UNVISITED)
        DFS(graph, w);//从v的邻接点w开始深度优先搜索
}
```

```
template <class DataType, class WeightType>
bool ExistPathDFS(ALDirNetwork<DataType, WeightType> &graph, DataType &vi,
DataType &vj)  {
    int i, j;
    i = graph.GetOrder(vi);
    j = graph.GetOrder(vj);
    if(i == -1 || j == -1)  {
        cout << "Vertex is not exist!" << endl;
        return false;
    }
    if(i == j)  {
        cout << "The start cannot be the same as the end!" << endl;
        return false;
    }
    for(int k = 0; k < graph.GetVexNum(); k++)
        graph.SetVisitedTag(k, UNVISITED);
    DFS(graph, i);
    return graph.GetVisitedTag(j) == VISITED;
}
```

```
//实验题第2题
template <class DataType, class WeightType>
bool ExistPathBFS(ALDirNetwork<DataType, WeightType> &graph, DataType &vi,
DataType &vj)  {
    int i, j;
    bool result = false;
    LinkQueue<int> queue;
    i = graph.GetOrder(vi);    j = graph.GetOrder(vj);
    if(i == -1 || j == -1)  {
        cout << "Vertex is not exist!" << endl;
        return false;
    }
    if(i == j)  {
        cout << "The start point cannot be the same as the end point!" << endl;
        return false;
    }
```

```
    for(int k = 0; k < graph.GetVexNum(); k++)
        graph.SetVisitedTag(k, UNVISITED);
    graph.SetVisitedTag(i, VISITED);
    queue.EnQueue(i);
    while(!queue.IsEmpty() && !result) {
        queue.DelQueue(i);
        for(int w = graph.GetFirstAdjvex(i); w != -1; w = graph.GetNextAdjvex(i, w))
            if(graph.GetVisitedTag(w) == UNVISITED) {
                if(w == j) {
                    result = true;
                    break;
                }
                graph.SetVisitedTag(w, VISITED);
                queue.EnQueue(w);
            }
    }
    return result;
}
```

```
//实验题第3题
template <class DataType, class WeightType>
void Dijkstra(const ALDirNetwork<DataType, WeightType> &graph, int v0,
WeightType *dist, int *path)  {
    WeightType mindist, infinity = graph.GetInfinity();
    int v, u;
    for (v = 0; v < graph.GetVexNum(); v++) {
        dist[v] = graph.GetWeight(v0, v);
        if (dist[v] == infinity)  path[v] = -1;
        else  path[v] = v0;
        graph.SetVisitedTag(v, UNVISITED);
    }
    graph.SetVisitedTag(v0, VISITED);
```

```
    for (int i = 1; i < graph.GetVexNum(); i++) {
        u = v0;        mindist = infinity;
        for (v = 0; v < graph.GetVexNum(); v++)
            if (graph.GetVisitedTag(v) == UNVISITED && dist[v] < mindist)  {
                u = v;    mindist = dist[v];
            }
        graph.SetVisitedTag(u, VISITED);
        for (v = graph.GetFirstAdjvex(u); v != -1; v = graph.GetNextAdjvex(u, v))
            if (graph.GetVisitedTag(v) == UNVISITED && mindist +
                graph.GetWeight(u, v) < dist[v]) {
                dist[v] = mindist + graph.GetWeight(u, v);
                path[v] = u;
            }
    }
}
```

```
//输出起点start到所有顶点的最短路径和最短路径长度
template <class DataType, class WeightType>
void OutputShortestPath(const ALDirNetwork<DataType, WeightType> &graph, int
start, WeightType *dist, int *path)  {
    DataType *route = new DataType[graph.GetVexNum()];
    int len;
    char sdata, edata, tdata;
    graph.GetElem(start, sdata);
    for (int i = 0; i < graph.GetVexNum(); i++)  {
        if (i == start)    continue;
        graph.GetElem(i, edata);
        len = 0;    route[len++] = edata;
        for (int j = i; path[j] != -1; j = path[j])  {
            graph.GetElem(path[j], tdata);
            route[len++] = tdata;
        }
```

```
        if (len == 1)
            cout << "There is no path between " << sdata << " and " << edata << endl;
        else  {
            cout << "The shortest path between " << sdata << " and " << edata << "
                        is:" << endl;
             for (int i = len - 1; i >= 0; i--)
                    cout << route[i] << " ";
             cout << endl;
            cout << "The distance is: " << dist[i] << endl;
        }
    }
    delete[]route;
}
```