

```

1 //before
2 #include "StackAndQueue.h"
3
4 //-----图的邻接表法
5 #include <iostream>
6 #include <stdio.h>
7 using namespace std;
8
9 //-----写在前边
10 #define InfoType int
11 #define VertexType char
12 #define INFINITY INT_MAX //最大值表示无联通
13
14 typedef enum {DG, DN, UDG, UDN} GraphKind; //枚举型变量: 有向图、有向网、无向图、无向网
15 typedef enum {ERROR, OK} Status; //枚举型, 函数状态变量
16
17 #define MAX_VERTEX_NUM 20
18
19 typedef struct ArcNode{
20     int adjvex; //该弧所指向的顶点位置
21     struct ArcNode *nextarc; //指向下一条弧的指针
22
23     InfoType *info; //该弧相关信息的指针
24 }ArcNode;
25
26 typedef struct VNode{
27     VertexType data; //顶点信息
28     ArcNode *firstarc; //指向第一条依附该顶点的弧的指针
29 }VNode, AdjList[MAX_VERTEX_NUM];
30
31 typedef struct{
32     AdjList vertices; //顶点数组
33     int vexnum, arcnum; //图中顶点及弧的数目
34     GraphKind kind; //图的种类
35 }ALGraph;
36
37 //-----基本函数
38 //若c中存在顶点u, 则返回u在c中的位置, 若没找到则返回-1
39 int LocateVex(ALGraph G, VertexType v){
40     int i=0;
41     for(i=0; i<G.vexnum; i++){
42         if(G.vertices[i].data==v)
43             break;
44     }
45     if(i<G.vexnum)
46         return i;
47     else
48         return -1;
49 }
50
51 //构造有向图
52 Status CreatedG(ALGraph &G){
53     char IncInfo; //0/1用来表示弧边是否含有辅助信息
54     cout << "Please input: vexnum(no more than 20) arcnum(no more than vexnum*vexnum)
55     IncInfo(default 0)" << endl;
56     cin >> G.vexnum >> G.arcnum >> IncInfo; //0/1用来表示弧边是否含有辅助信息
57     cout << "构造顶点向量" << endl;
58     for(int i=0; i<G.vexnum; i++){
59         //构造顶点向量并初始化第一条依附顶点的指针
60         cin >> G.vertices[i].data;
61         G.vertices[i].firstarc = NULL;
62     }
63
64     VertexType v1, v2; //顶点
65     int v1_int, v2_int;
66     ArcNode *p;
67     cout << "以'起点 终点'的方式依次输入每一条边(例如: ab\t起点: a, 终点: b): " << endl;
68     for(int i=0; i<G.arcnum; i++){ //构造邻接表
69         cin >> v1 >> v2;
70         //定位v1, v2在G中的位置
71         v1_int=LocateVex(G, v1);
72         v2_int=LocateVex(G, v2);
73         p = new ArcNode;
74         p->adjvex = v2_int;
75         p->info = NULL;
76         p->nextarc = G.vertices[v1_int].firstarc; //以头插法的方式将p插入到v1的第一个弧边上
77         G.vertices[v1_int].firstarc = p;
78         // if(IncInfo)
79         //     input(*G.arcs[i][j].info);
80         // 若弧边上有辅助信息, 则输入
81     }
82     return OK;
83 }

```

```

83 //构造有向网
84 Status CreateDN(ALGraph &G){
85     char IncInfo; //0/1用来表示弧边是否含有辅助信息
86     cout << "Please input: vexnum(no more than 20) arcnum(no more than vexnum*vexnum)
IncInfo(default 0)" << endl;
87     cin >> G.vexnum >> G.arcnum >> IncInfo; //0/1用来表示弧边是否含有辅助信息
88     cout << "构造顶点向量" << endl;
89     for(int i=0; i<G.vexnum; i++){
//构造顶点向量并初始化第一条依附顶点的指针
90         cin >> G.vertices[i].data;
91         G.vertices[i].firstarc = NULL;
92     }
93
94     VertexType v1,v2; //顶点
95     int v1_int,v2_int;
96     InfoType w; //顶点信息
97     ArcNode *p;
98     cout << "以起点 终点
权值'的方式依次输入每一条边(例如: ab\t起点:a, 终点:b, 权值:9): " << endl;
99     for(int i=0; i<G.arcnum; i++){ //构造邻接表
100         cin >> v1 >> v2 >> w;
101         //定位v1,v2在G中的位置
102         v1_int=LocateVex(G,v1);
103         v2_int=LocateVex(G,v2);
104         p = new ArcNode;
105         p->adjvex = v2_int;
106         p->info = new InfoType;
107         *p->info = w;
108         p->nextarc = G.vertices[v1_int].firstarc; //以头插法的方式讲p插入到v1的第一个弧边上
109         G.vertices[v1_int].firstarc = p;
110         // if(IncInfo)
111         //     Input(*G.arcs[i][j].info);
112         //     若弧边上有辅助信息, 则输入
113     }
114     return OK;
115 }
116
117 //构造无向图
118 Status CreateUDG(ALGraph &G){
119     char IncInfo; //0/1用来表示弧边是否含有辅助信息
120     cout << "Please input: vexnum(no more than 20) arcnum(no more than vexnum*vexnum)
IncInfo(default 0)" << endl;
121     cin >> G.vexnum >> G.arcnum >> IncInfo; //0/1用来表示弧边是否含有辅助信息
122     cout << "构造顶点向量" << endl;
123     for(int i=0; i<G.vexnum; i++){
//构造顶点向量并初始化第一条依附顶点的指针
124         cin >> G.vertices[i].data;
125         G.vertices[i].firstarc = NULL;
126     }
127
128     VertexType v1,v2; //顶点
129     int v1_int,v2_int;
130     ArcNode *p;
131     cout << "以起点 终点'的方式依次输入每一条边(例如: ab\t起点:a, 终点:b): " << endl;
132     for(int i=0; i<G.arcnum; i++){ //构造邻接表
133         cin >> v1 >> v2;
134         //定位v1,v2在G中的位置
135         v1_int=LocateVex(G,v1);
136         v2_int=LocateVex(G,v2);
137         //无向图中单方向
138         p = new ArcNode;
139         p->adjvex = v2_int;
140         p->info = NULL;
141         p->nextarc =
G.vertices[v1_int].firstarc; //以头插法的方式讲p插入到v1的第一个弧边上
142         G.vertices[v1_int].firstarc = p;
143         // if(IncInfo)
144         //     Input(*G.arcs[i][j].info);
145         //     若弧边上有辅助信息, 则输入
146     }
147     //无向图中反方向
148     p = new ArcNode;
149     p->adjvex = v1_int;
150     p->info = NULL;
151     p->nextarc =
G.vertices[v2_int].firstarc; //以头插法的方式讲p插入到v2的第一个弧边上
152     G.vertices[v2_int].firstarc = p;
153     // if(IncInfo)
154     //     Input(*G.arcs[i][j].info);
155     //     若弧边上有辅助信息, 则输入
156 }
157 }
158 return OK;
159 }

```

```

160
161 //构造无向网
162 Status CreateUDN(ALGraph &G){
163     char IncInfo; //0/1用来表示弧边是否含有辅助信息
164     cout << "Please input: vexnum(no more than 20) arcnum(no more than vexnum*vexnum)
IncInfo(default 0)" << endl;
165     cin >> G.vexnum >> G.arcnum >> IncInfo; //0/1用来表示弧边是否含有辅助信息
166     cout << "构造顶点向量" << endl;
167     for(int i=0; i<G.vexnum; i++){
//构造顶点向量并初始化第一条依附顶点的指针
168         cin >> G.vertices[i].data;
169         G.vertices[i].firstarc = NULL;
170     }
171
172     VertexType v1,v2; //顶点
173     int v1_int,v2_int;
174     InfoType w; //顶点信息
175     ArcNode *p;
176     cout << "以起点 终点
权值的方式依次输入每一条边(例如: ab9\t起点: a, 终点: b, 权值: 9): " << endl;
177     for(int i=0; i<G.arcnum; i++){ //构造邻接表
178         cin >> v1 >> v2 >> w;
179         //定位v1,v2在G中的位置
180         v1_int=LocateVex(G,v1);
181         v2_int=LocateVex(G,v2);
182         //无向图中单方向
183         p = new ArcNode;
184         p->adjvex = v2_int;
185         p->info = new InfoType;
186         *p->info = w;
187         p->nextarc =
G.vertices[v1_int].firstarc; //以头插法的方式将p插入到v1的第一个弧边上
188         G.vertices[v1_int].firstarc = p;
189         // if(IncInfo)
190         //     Input(*G.arcs[i][j].info);
191         // 若弧边上有辅助信息, 则输入
192     }
193     //无向图中反方向
194     p = new ArcNode;
195     p->adjvex = v1_int;
196     p->info = new InfoType;
197     *p->info = w;
198     p->nextarc =
G.vertices[v2_int].firstarc; //以头插法的方式将p插入到v2的第一个弧边上
199     G.vertices[v2_int].firstarc = p;
200     // if(IncInfo)
201     //     Input(*G.arcs[i][j].info);
202     // 若弧边上有辅助信息, 则输入
203 }
204 }
205 return OK;
206 }
207
208 //创建ALGraph
209 Status CreateGraph(ALGraph &G){
210     cout << "选择图的类型: " << endl;
211     cout << "DG,DN,UDG,UDN 有向图、有向网、无向图、无向网" << endl;
212     scanf("%d",&G.kind);
213     switch(G.kind){
214         case DG: return CreatedG(G); //构造有向图
215         case DN: return CreatedN(G); //构造有向网
216         case UDG: return CreateUDG(G); //构造无向图
217         case UDN: return CreateUDN(G); //构造无向网
218         default: return ERROR;
219     }
220 }
221
222 //打印邻接表
223 Status ALGraphShow(ALGraph G){
224
225     cout << "ALGraph 顶点信息: " << endl;
226     for(int i=0; i<G.vexnum; i++){
227         cout << G.vertices[i].data << "\t";
228     }
229     cout << endl << endl;
230
231     cout << "ALGraph 邻接链表: " << endl;
232     ArcNode *p;
233     switch(G.kind){
234         case DG:
235         case UDG:
236         {
237             for(int i=0; i<G.vexnum; i++){
238                 p=G.vertices[i].firstarc;
239                 cout << G.vertices[i].data << " | \t";

```

```

239         while(p){
240             cout << G.vertices[p->adjvex].data;
241             if( p->nextarc )
242                 cout << " --> ";
243             p=p->nextarc;
244         }
245         cout << endl;
246     }
247     break;
248 }
249 case DN:
250 case UDN:
251 {
252     for(int i=0; i<G.vexnum; i++){
253         p=G.vertices[i].firstarc;
254         cout << G.vertices[i].data << "  |\\t";
255         while(p){
256             cout << G.vertices[p->adjvex].data << "_" << *p->info;
257             if( p->nextarc )
258                 cout << " --> ";
259             p=p->nextarc;
260         }
261         cout << endl;
262     }
263 }
264
265     default: return ERROR;
266 }
267 return OK;
268 }
269
270 //求某个顶点的入度
271 int FindInDegree(ALGraph G, int v){
272     ArcNode *p;
273     int indegree=0;
274     for(int i=0; i<G.vexnum; i++){
275         p = G.vertices[i].firstarc;
276         while(p){
277             if(p->adjvex == v)
278                 indegree++;
279             p=p->nextarc;
280         }
281     }
282     return indegree;
283 }
284
285 //-----拓扑排序
286
287 //入度数组，存放每个顶点的入度
288 int indegree[MAX_VERTEX_NUM];
289
290 //对有向图进行拓扑排序，图G采用邻接表法的存储方式
291 //图中无回路，返回OK，并输出一条拓扑排序序列，若有回路则返回ERROR
292 Status TopologicalSort(ALGraph G){
293     for(int i=0; i<G.vexnum; i++) //求每个顶点的入度
294         indegree[i] = FindInDegree(G, i);
295
296     SqStack S;
297     InitStack_Sq(S);
298     for(int i=0; i<G.vexnum; i++)
299         if(indegree[i]==0)
300             Push_Sq(S, i); //入度为0者进栈
301
302     int count = 0; //对输出顶点计数
303     int i, k;
304     ArcNode *p;
305     while(!StackEmpty_Sq(S)){ //栈不为空
306         Pop_Sq(S, i);
307         //输出第i个顶点并计数
308         cout << i << ": " << G.vertices[i].data << endl;
309         count ++;
310         for(p=G.vertices[i].firstarc; p; p=p->nextarc){
311             k = p->adjvex;
312             if(0==(--indegree[k])) //k的入度不为0，k入栈
313                 Push_Sq(S, k);
314         }
315     }
316     if(count<G.vexnum)
317         return ERROR;
318     else
319         return OK;
320 }
321

```