1. /\* 图的邻接矩阵表示法 \*/
3. #define MaxVertexNum 100    /\* 最大顶点数设为100 \*/
4. #define INFINITY 65535        /\* ∞设为双字节无符号整数的最大值65535\*/
5. typedef int Vertex;         /\* 用顶点下标表示顶点,为整型 \*/
6. typedef int WeightType;        /\* 边的权值设为整型 \*/
7. typedef char DataType;        /\* 顶点存储的数据类型设为字符型 \*/
9. /\* 边的定义 \*/
10. typedef struct ENode \*PtrToENode;
11. struct ENode{
12. Vertex V1, V2;      /\* 有向边<V1, V2> \*/
13. WeightType Weight;  /\* 权重 \*/
14. };
15. typedef PtrToENode Edge;
17. /\* 图结点的定义 \*/
18. typedef struct GNode \*PtrToGNode;
19. struct GNode{
20. int Nv;  /\* 顶点数 \*/
21. int Ne;  /\* 边数   \*/
22. WeightType G[MaxVertexNum][MaxVertexNum]; /\* 邻接矩阵 \*/
23. DataType Data[MaxVertexNum];      /\* 存顶点的数据 \*/
24. /\* 注意：很多情况下，顶点无数据，此时Data[]可以不用出现 \*/
25. };
26. typedef PtrToGNode MGraph; /\* 以邻接矩阵存储的图类型 \*/


30. MGraph CreateGraph( int VertexNum )
31. { /\* 初始化一个有VertexNum个顶点但没有边的图 \*/
32. Vertex V, W;
33. MGraph Graph;
35. Graph = (MGraph)malloc(sizeof(struct GNode)); /\* 建立图 \*/
36. Graph->Nv = VertexNum;
37. Graph->Ne = 0;
38. /\* 初始化邻接矩阵 \*/
39. /\* 注意：这里默认顶点编号从0开始，到(Graph->Nv - 1) \*/
40. for (V=0; V<Graph->Nv; V++)
41. for (W=0; W<Graph->Nv; W++)
42. Graph->G[V][W] = INFINITY;
44. return Graph;
45. }
47. void InsertEdge( MGraph Graph, Edge E )
48. {
49. /\* 插入边 <V1, V2> \*/
50. Graph->G[E->V1][E->V2] = E->Weight;
51. /\* 若是无向图，还要插入边<V2, V1> \*/
52. Graph->G[E->V2][E->V1] = E->Weight;
53. }
55. MGraph BuildGraph()
56. {
57. MGraph Graph;
58. Edge E;
59. Vertex V;
60. int Nv, i;
62. scanf("%d", &Nv);   /\* 读入顶点个数 \*/
63. Graph = CreateGraph(Nv); /\* 初始化有Nv个顶点但没有边的图 \*/
65. scanf("%d", &(Graph->Ne));   /\* 读入边数 \*/
66. if ( Graph->Ne != 0 ) { /\* 如果有边 \*/
67. E = (Edge)malloc(sizeof(struct ENode)); /\* 建立边结点 \*/
68. /\* 读入边，格式为"起点 终点 权重"，插入邻接矩阵 \*/
69. for (i=0; i<Graph->Ne; i++) {
70. scanf("%d %d %d", &E->V1, &E->V2, &E->Weight);
71. /\* 注意：如果权重不是整型，Weight的读入格式要改 \*/
72. InsertEdge( Graph, E );
73. }
74. }
76. /\* 如果顶点有数据的话，读入数据 \*/
77. for (V=0; V<Graph->Nv; V++)
78. scanf(" %c", &(Graph->Data[V]));
80. return Graph;
81. }
82. /\* 图的邻接表表示法 \*/
84. #define MaxVertexNum 100    /\* 最大顶点数设为100 \*/
85. typedef int Vertex;         /\* 用顶点下标表示顶点,为整型 \*/
86. typedef int WeightType;        /\* 边的权值设为整型 \*/
87. typedef char DataType;        /\* 顶点存储的数据类型设为字符型 \*/
89. /\* 边的定义 \*/
90. typedef struct ENode \*PtrToENode;
91. struct ENode{
92. Vertex V1, V2;      /\* 有向边<V1, V2> \*/
93. WeightType Weight;  /\* 权重 \*/
94. };
95. typedef PtrToENode Edge;
97. /\* 邻接点的定义 \*/
98. typedef struct AdjVNode \*PtrToAdjVNode;
99. struct AdjVNode{
100. Vertex AdjV;        /\* 邻接点下标 \*/
101. WeightType Weight;  /\* 边权重 \*/
102. PtrToAdjVNode Next;    /\* 指向下一个邻接点的指针 \*/
103. };
105. /\* 顶点表头结点的定义 \*/
106. typedef struct Vnode{
107. PtrToAdjVNode FirstEdge;/\* 边表头指针 \*/
108. DataType Data;            /\* 存顶点的数据 \*/
109. /\* 注意：很多情况下，顶点无数据，此时Data可以不用出现 \*/
110. } AdjList[MaxVertexNum];    /\* AdjList是邻接表类型 \*/
112. /\* 图结点的定义 \*/
113. typedef struct GNode \*PtrToGNode;
114. struct GNode{
115. int Nv;     /\* 顶点数 \*/
116. int Ne;     /\* 边数   \*/
117. AdjList G;  /\* 邻接表 \*/
118. };
119. typedef PtrToGNode LGraph; /\* 以邻接表方式存储的图类型 \*/


123. LGraph CreateGraph( int VertexNum )
124. { /\* 初始化一个有VertexNum个顶点但没有边的图 \*/
125. Vertex V;
126. LGraph Graph;
128. Graph = (LGraph)malloc( sizeof(struct GNode) ); /\* 建立图 \*/
129. Graph->Nv = VertexNum;
130. Graph->Ne = 0;
131. /\* 初始化邻接表头指针 \*/
132. /\* 注意：这里默认顶点编号从0开始，到(Graph->Nv - 1) \*/
133. for (V=0; V<Graph->Nv; V++)
134. Graph->G[V].FirstEdge = NULL;
136. return Graph;
137. }
139. void InsertEdge( LGraph Graph, Edge E )
140. {
141. PtrToAdjVNode NewNode;
143. /\* 插入边 <V1, V2> \*/
144. /\* 为V2建立新的邻接点 \*/
145. NewNode = (PtrToAdjVNode)malloc(sizeof(struct AdjVNode));
146. NewNode->AdjV = E->V2;
147. NewNode->Weight = E->Weight;
148. /\* 将V2插入V1的表头 \*/
149. NewNode->Next = Graph->G[E->V1].FirstEdge;
150. Graph->G[E->V1].FirstEdge = NewNode;
152. /\* 若是无向图，还要插入边 <V2, V1> \*/
153. /\* 为V1建立新的邻接点 \*/
154. NewNode = (PtrToAdjVNode)malloc(sizeof(struct AdjVNode));
155. NewNode->AdjV = E->V1;
156. NewNode->Weight = E->Weight;
157. /\* 将V1插入V2的表头 \*/
158. NewNode->Next = Graph->G[E->V2].FirstEdge;
159. Graph->G[E->V2].FirstEdge = NewNode;
160. }
162. LGraph BuildGraph()
163. {
164. LGraph Graph;
165. Edge E;
166. Vertex V;
167. int Nv, i;
169. scanf("%d", &Nv);   /\* 读入顶点个数 \*/
170. Graph = CreateGraph(Nv); /\* 初始化有Nv个顶点但没有边的图 \*/
172. scanf("%d", &(Graph->Ne));   /\* 读入边数 \*/
173. if ( Graph->Ne != 0 ) { /\* 如果有边 \*/
174. E = (Edge)malloc( sizeof(struct ENode) ); /\* 建立边结点 \*/
175. /\* 读入边，格式为"起点 终点 权重"，插入邻接矩阵 \*/
176. for (i=0; i<Graph->Ne; i++) {
177. scanf("%d %d %d", &E->V1, &E->V2, &E->Weight);
178. /\* 注意：如果权重不是整型，Weight的读入格式要改 \*/
179. InsertEdge( Graph, E );
180. }
181. }
183. /\* 如果顶点有数据的话，读入数据 \*/
184. for (V=0; V<Graph->Nv; V++)
185. scanf(" %c", &(Graph->G[V].Data));
187. return Graph;
188. }
189. /\* 邻接表存储的图 - DFS \*/
191. void Visit( Vertex V )
192. {
193. printf("正在访问顶点%d\n", V);
194. }
196. /\* Visited[]为全局变量，已经初始化为false \*/
197. void DFS( LGraph Graph, Vertex V, void (\*Visit)(Vertex) )
198. {   /\* 以V为出发点对邻接表存储的图Graph进行DFS搜索 \*/
199. PtrToAdjVNode W;
201. Visit( V ); /\* 访问第V个顶点 \*/
202. Visited[V] = true; /\* 标记V已访问 \*/
204. for( W=Graph->G[V].FirstEdge; W; W=W->Next ) /\* 对V的每个邻接点W->AdjV \*/
205. if ( !Visited[W->AdjV] )    /\* 若W->AdjV未被访问 \*/
206. DFS( Graph, W->AdjV, Visit );    /\* 则递归访问之 \*/
207. }
208. /\* 邻接矩阵存储的图 - BFS \*/
210. /\* IsEdge(Graph, V, W)检查<V, W>是否图Graph中的一条边，即W是否V的邻接点。  \*/
211. /\* 此函数根据图的不同类型要做不同的实现，关键取决于对不存在的边的表示方法。\*/
212. /\* 例如对有权图, 如果不存在的边被初始化为INFINITY, 则函数实现如下:         \*/
213. bool IsEdge( MGraph Graph, Vertex V, Vertex W )
214. {
215. return Graph->G[V][W]<INFINITY ? true : false;
216. }
218. /\* Visited[]为全局变量，已经初始化为false \*/
219. void BFS ( MGraph Graph, Vertex S, void (\*Visit)(Vertex) )
220. {   /\* 以S为出发点对邻接矩阵存储的图Graph进行BFS搜索 \*/
221. Queue Q;
222. Vertex V, W;
224. Q = CreateQueue( MaxSize ); /\* 创建空队列, MaxSize为外部定义的常数 \*/
225. /\* 访问顶点S：此处可根据具体访问需要改写 \*/
226. Visit( S );
227. Visited[S] = true; /\* 标记S已访问 \*/
228. AddQ(Q, S); /\* S入队列 \*/
230. while ( !IsEmpty(Q) ) {
231. V = DeleteQ(Q);  /\* 弹出V \*/
232. for( W=0; W<Graph->Nv; W++ ) /\* 对图中的每个顶点W \*/
233. /\* 若W是V的邻接点并且未访问过 \*/
234. if ( !Visited[W] && IsEdge(Graph, V, W) ) {
235. /\* 访问顶点W \*/
236. Visit( W );
237. Visited[W] = true; /\* 标记W已访问 \*/
238. AddQ(Q, W); /\* W入队列 \*/
239. }
240. } /\* while结束\*/
241. }