哈尔滨工业大学计算学部

实验报告

课程名称：数据结构与算法

课程类型：专业基础（必修）

实验项目：图型结构及其应用

实验题目：最短路径算法

实验日期：2023.10.25

班级：2203102

学号：2022111570

姓名：王乾宇

|  |  |  |
| --- | --- | --- |
| 设计成绩 | 报告成绩 | 指导老师 |
|  |  | 张岩 |

1. **实验目的**

最短路径问题研究的主要有：单源最短路径问题和所有顶点对之间的最短路径问题。在计算机领域和实际工程中具有广泛的应用，如集成电路设计、GPS/游戏地图导航、智能交通、路由选择、铺设管线等。理解、实现并运用两种最短路径算法：Dijkstra算法和Floyd-Warshall算法，求解图的最短路径。

1. **实验要求及实验环境**

实验要求：

1.实现单源最短路径的 Dijkstra 算法，输出源点及其到其他顶点的最短路径长

度和最短路径。

2.实现全局最短路径的 Floyd-Warshall算法。计算任意两个顶点间的最短距离

矩阵和最短路径矩阵，并输出任意两个顶点间的最短路径长度和最短路径。

3.利用 Dijkstra 或 Floyd-Warshall 算法解决单目标最短路径问题：找出图中每

个顶点 v 到某个指定顶点 c 最短路径。

4.利用 Dijkstra 或 Floyd-Warshall算法解决单顶点对间最短路径问题：对于某

对顶点 u 和 v，找出 u 到 v 和 v 到 u 的一条最短路径。

5.以文件形式输入图的顶点和边，并以适当的方式展示相应的结果。要求顶点

不少于 10 个，边不少于 13 个。

6.（选做）实现 Warshall 算法，计算有向图的可达矩阵，理解可达矩阵的含义；

7.（选做）利用堆结构（优先级队列）改进和优化 Dijkstra 算法，实现改进和

优化的 Dijkstra 算法，并与原算法进行实验比较；

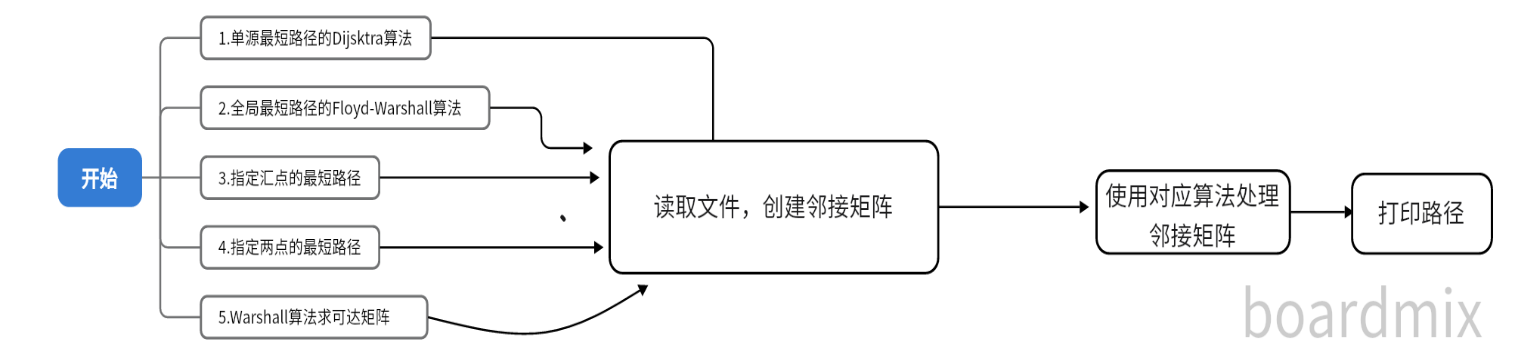
实验环境：  
Visual Studio 2022；Windows 10

**三、设计思想**（本程序中的用到的所有数据类型的定义，主程序的流程图及各程序模块之间的调用关系、核心算法的主要步骤）

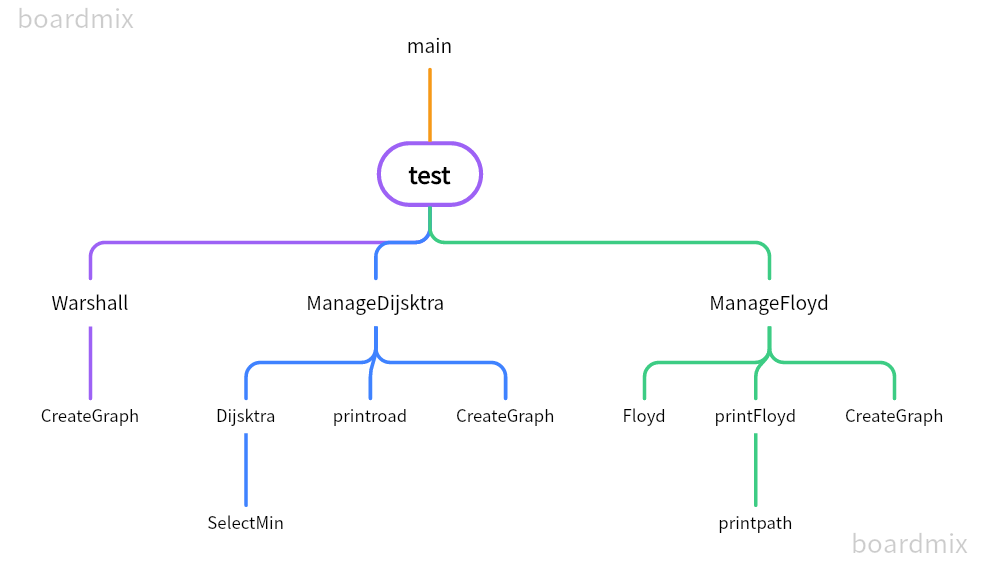
1.逻辑设计：利用邻接矩阵存储有向图

2.物理设计：

程序流程图如下：

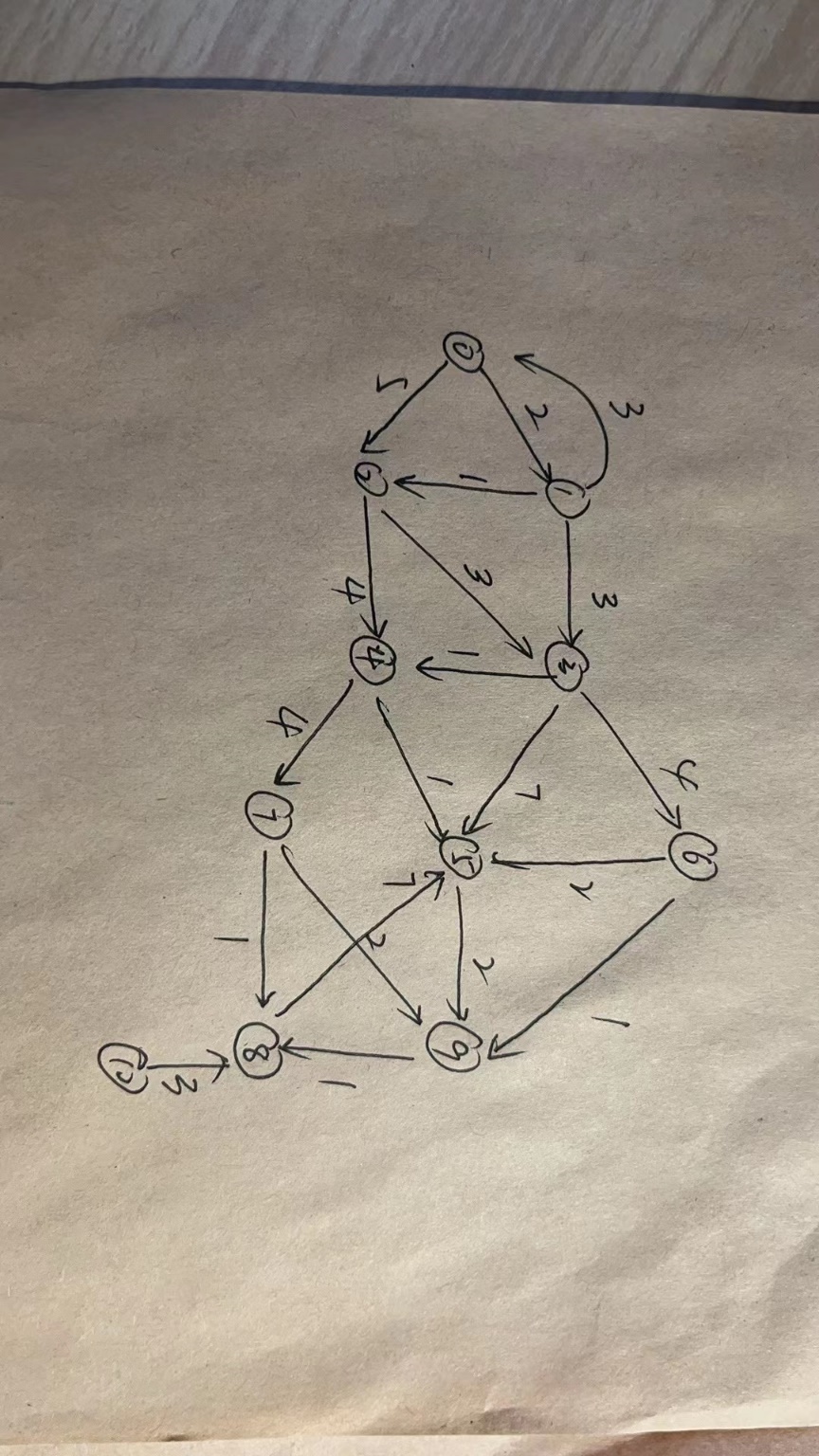


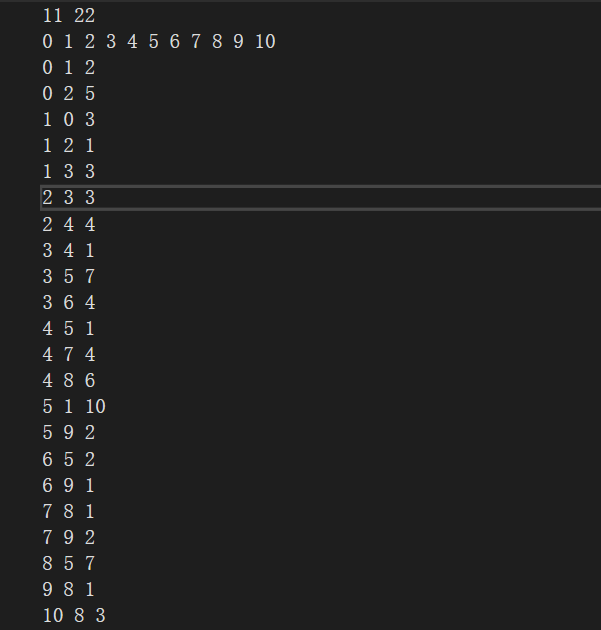
函数调用如下：



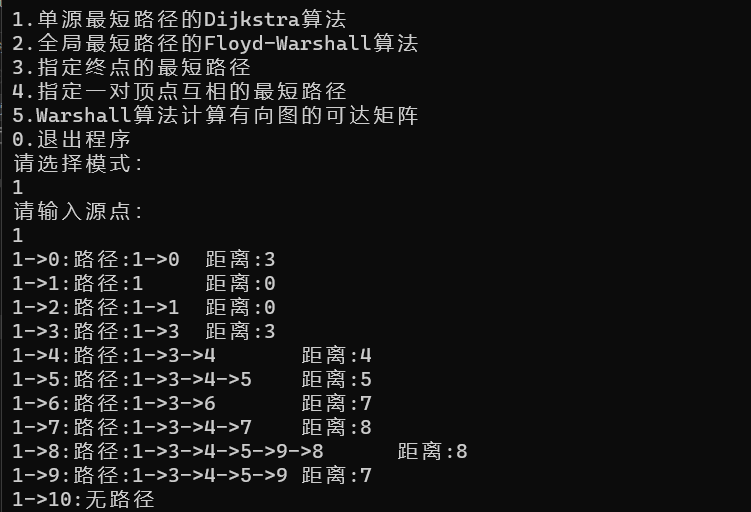
四、测试结果

测试数据如下：

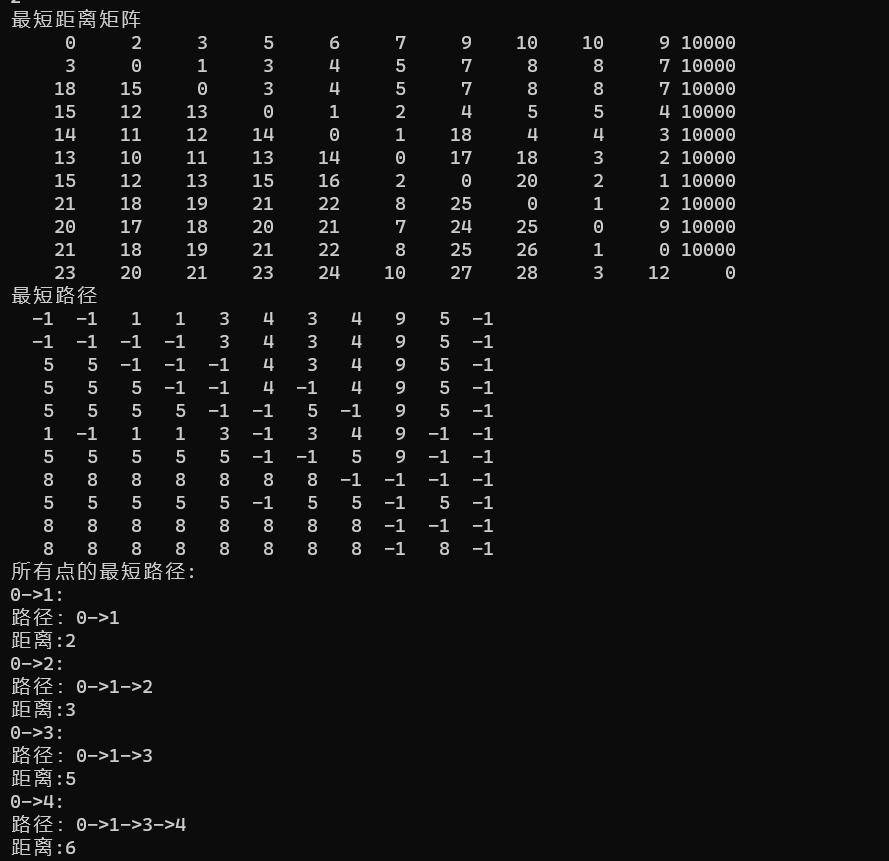


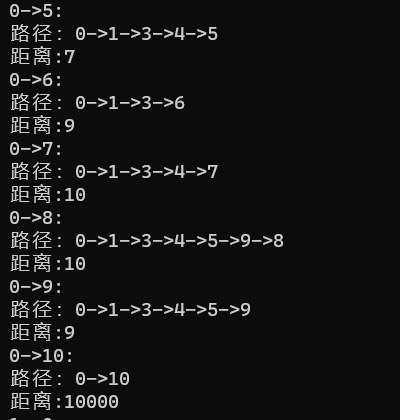


1. 单源最短路径的Dijkstra算法

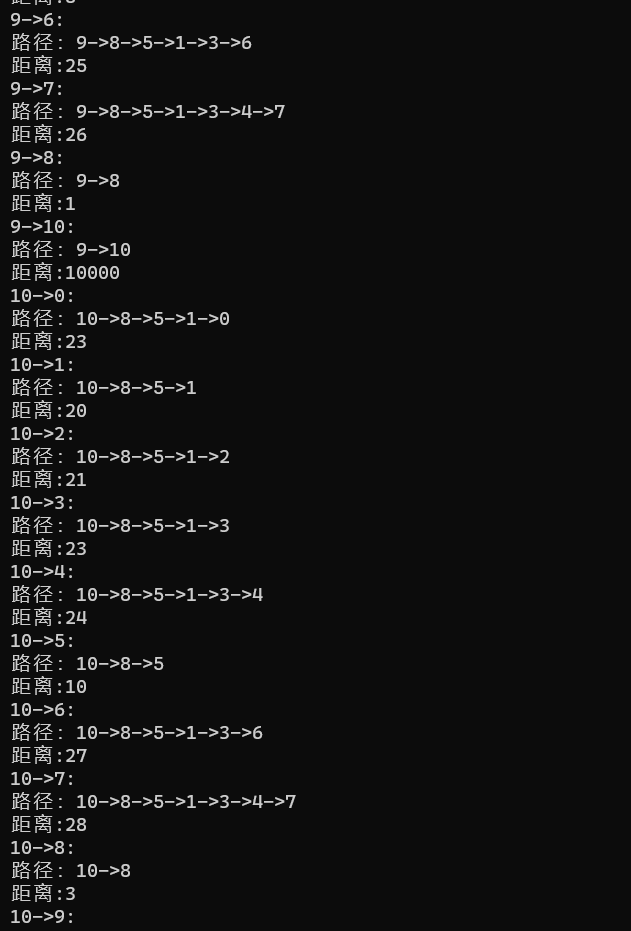


1. 全局最短路径的Floyd-Warshall算法

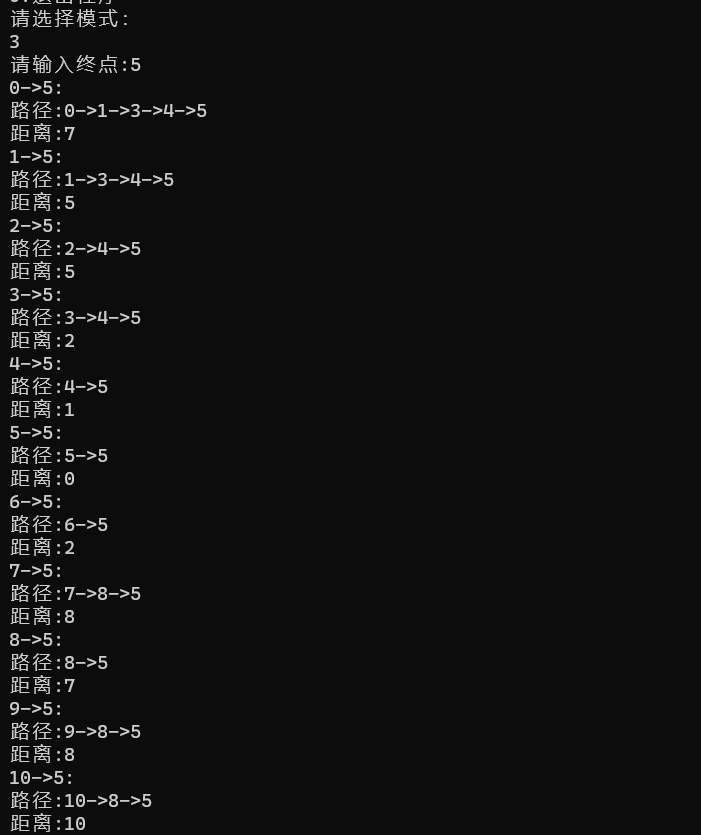




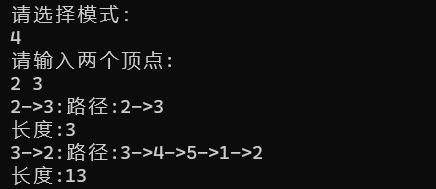
省略中间部分



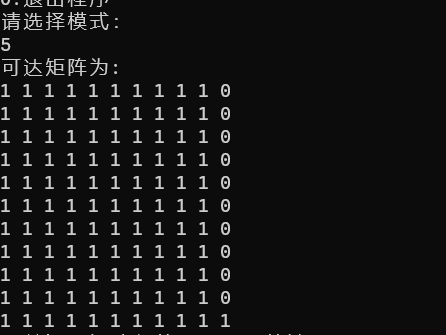
3.指定终点的最短路径



4.指定一堆顶点互相的路径



5.Warshall算法计算有向图的可达矩阵



五、经验体会和不足

本次实验进一步加深了我对Dijkstra和Floyd算法的理解，但在一些函数的实现上代码冗长，性能方面还有欠缺，可移植性较差。

六、附录：源代码（带注释）

1. #pragma once
2. #include<iostream>
3. #include<iomanip>
4. #include<fstream>
5. #include<time.h>
6. using namespace std;
7. #define inf 10000*//最大值*
8. *//邻接矩阵表示有向图*
9. #define VertexData int
10. #define EdgeData int
11. #define NumVertices 20
12. typedef struct MTGraph
13. {
14. VertexData vertex[NumVertices];
15. EdgeData edge[NumVertices][NumVertices];
16. int n;  *//顶点数*
17. int e;  *//边数*
18. }MTGraph;
19. #define \_CRT\_SECURE\_NO\_WARNINGS
20. #include"graph.h"
21. void CreateGraph(MTGraph\* G)
22. {
23. ifstream file("test.txt");
24. file >> G->n;
25. file >> G->e;
26. *//初始化*
27. for (int i = 0; i < G->n; i++)
28. {
29. for (int j = 0; j < G->n; j++)
30. G->edge[i][j] = inf;
31. }
32. for (int i = 0; i < G->n; i++)
33. {
34. G->edge[i][i] = 0; *//对角线初始化为0*
35. }
36. for (int i = 0; i < G->n; i++)
37. {
38. file >> G->vertex[i];
39. }
40. for (int i = 0; i < G->e; i++)
41. {
42. int a, b, w;
43. file >> a >> b >> w;
44. G->edge[a][b] = w;
45. }
46. }
47. int SelectMin(MTGraph\* G,int dist[], bool s[])*//在V-s中找到w，使d[w]最小*
48. {
49. int temp = inf;
50. int w = 0;
51. for (int i = 0; i < G->n; i++)
52. {
53. if (!s[i] && dist[i] < temp)
54. {
55. temp = dist[i];
56. w = i;
57. }
58. }
59. return w;
60. }
61. void Dijkstra(MTGraph\* G, int start, int dist[], int path[], bool s[])
62. {
63. for (int i = 0; i < G->n; i++)
64. {
65. dist[i] = G->edge[start][i];
66. path[i] = start;
67. s[i] = false;
68. }
69. s[start] = true;
70. for (int i = 0; i < G->n - 1; i++)
71. {
72. int w = SelectMin(G, dist, s);
73. s[w] = true;
74. for (int v = 0; v < G->n; v++)*//加入顶点w后，更新dist[]和path[]*
75. {
76. if (!s[v])
77. {
78. int sum = dist[w] + G->edge[w][v];;
79. if (sum < dist[v])
80. {
81. dist[v] = sum;
82. path[v] = w;
83. }
84. }
85. }
87. }
88. }
89. *//打印路径*
90. void printroad(MTGraph\* G, int start, int dist[], int path[])
91. {
92. for (int i = 0; i < G->n; i++)
93. {
94. cout << start << "->" << i << ":";
95. if (dist[i] == inf)
96. {
97. cout << "无路径";
98. cout << endl;
99. }
100. else if (dist[i] == start)
101. {
102. cout << "路径:";
103. cout << start << "->" << start;
104. cout << "\t距离:";
105. cout << 0 << endl;
106. }
107. else
108. {
109. int j = 0;
110. int r[NumVertices];
111. int temp = i;
112. while (temp != start)
113. {
114. r[j++] = temp;
115. temp = path[temp];
116. }
117. r[j++] = start;
118. cout << "路径:";
119. while (j--)
120. {
121. cout << r[j];
122. if (j != 0)
123. cout << "->";
124. }
125. cout << "\t距离:" << dist[i] << endl;
126. }
127. }
128. }
129. void ManageDijkstra(int mode)
130. {
131. MTGraph\* G = new MTGraph;
132. CreateGraph(G);
133. int dist[NumVertices]; *//从源点到该节点i的当前最短路径长*
134. int path[NumVertices]; *//最短路径上最后一个节点*
135. bool s[NumVertices]; *//记录是否在已生成的最短路经图*
136. int start;
137. cout << "请输入源点：" << endl;
138. cin >> start;
139. if (mode == 1)
140. {
141. Dijkstra(G, start, dist, path, s);
142. printroad(G, start, dist, path);
143. }
144. }
145. void Floyd(MTGraph\* G, int dist[][NumVertices], int path[][NumVertices])
146. {
147. for (int i = 0; i < G->n; i++)
148. {
149. for (int j = 0; j < G->n; j++)
150. {
151. dist[i][j] = G->edge[i][j];
152. path[i][j] = -1;
153. }
154. }
155. for (int k = 0; k < G->n; k++)
156. {
157. for (int i = 0; i < G->n; i++)
158. {
159. for (int j = 0; j < G->n; j++)
160. {
161. if (dist[i][k] + dist[k][j] < dist[i][j])
162. {
163. dist[i][j] = dist[i][k] + dist[k][j];
164. path[i][j] = k;
165. }
166. }
167. }
168. }
169. }
170. *//递归打印路径*
171. void printpath(int path[][NumVertices], int i, int j)
172. {
173. int k = path[i][j];
174. if (k != -1)
175. {
176. printpath(path, i, k);
177. cout << k << "->";
178. printpath(path, k, j);
179. }
180. }
181. *//输出最短距离矩阵、最短路径矩阵，*
182. *//任意两个顶点间的最短路径长度和最短路径*
183. void printFloyd\_2(MTGraph\* G, int dist[][NumVertices], int path[][NumVertices])
184. {
185. cout << "最短距离矩阵" << endl;
186. for (int i = 0; i < G->n; i++)
187. {
188. for (int j = 0; j < G->n; j++)
189. {
190. cout << setw(6) << dist[i][j];
191. }
192. cout << endl;
193. }
194. cout << "最短路径" << endl;
195. for (int i = 0; i < G->n; i++)
196. {
197. for (int j = 0; j < G->n; j++)
198. {
199. cout << setw(4) << path[i][j];
200. }
201. cout << endl;
202. }
203. cout << "所有点的最短路径:" << endl;
204. for (int i = 0; i < G->n; i++)
205. {
206. for (int j = 0; j < G->n; j++)
207. {
208. if (i != j)
209. {
210. cout << i << "->" << j << ":" << endl;
211. cout << "路径：";
212. cout << i<<"->";
213. printpath(path, i, j);
214. cout << j << endl;
215. cout << "距离:";
216. cout << dist[i][j] << endl;
217. }
218. }
219. }
220. }
221. *//每个顶点到指定终点c的所有路径和长度*
222. void printFloyd\_3(MTGraph\* G, int dist[][NumVertices], int path[][NumVertices])
223. {
224. int c;
225. cout << "请输入终点:";
226. cin >> c;
227. for (int i = 0; i < G->n; i++)
228. {
229. if (dist[i][c] == inf)
230. {
231. cout << i << "->" << c << ":";
232. cout << "无路径" << endl;
233. }
234. cout << i << "->" << c << ":"<<endl;
235. cout << "路径:";
236. cout << i;
237. cout << "->";
238. printpath(path, i, c);
239. cout << c << endl;
240. cout << "距离:" << dist[i][c] << endl;
241. }
242. }
243. *//指定起点和终点，输出最短路径和长度*
244. void printFloyd\_4(MTGraph\* G, int dist[][NumVertices], int path[][NumVertices])
245. {
246. int a, b;
247. cout << "请输入两个顶点:" << endl;
248. cin >> a >> b;
249. if (dist[a][b] == inf)
250. cout << a << "->" << b << " 无路径";
251. else
252. {
253. cout << a << "->" << b << ":";
254. cout << "路径:";
255. cout << a << "->";
256. printpath(path, a, b);
257. cout << b << endl;
258. cout << "长度:" << dist[a][b] << endl;
259. }
260. if (dist[b][a] == inf)
261. cout << b << "->" << a << " 无路径";
262. else
263. {
264. cout << b << "->" << a << ":";
265. cout << "路径:";
266. cout << b << "->";
267. printpath(path, b, a);
268. cout << a << endl;
269. cout << "长度:" << dist[b][a] << endl;
270. }
271. }
272. void ManageFloyd(int mode)
273. {
274. MTGraph\* G = new MTGraph;
275. CreateGraph(G);
276. int dist[NumVertices][NumVertices]; *//最短距离矩阵*
277. int path[NumVertices][NumVertices]; *//最短路径矩阵*
278. Floyd(G, dist, path);
279. if (mode == 2)
280. printFloyd\_2(G, dist, path);
281. else if (mode == 3)
282. printFloyd\_3(G, dist, path);
283. else
284. printFloyd\_4(G, dist, path);
285. }
286. void Warshall()
287. {
288. MTGraph\* G = new MTGraph;
289. CreateGraph(G);
290. int nn[NumVertices][NumVertices];
291. *//初始化可达矩阵*
292. for (int i = 0; i < G->n; i++)
293. {
294. for (int j = 0; j < G->n; j++)
295. {
296. if (G->edge[i][j] == inf)
297. nn[i][j] = 0;
298. else
299. nn[i][j] = 1;
300. }
301. }
302. *//动态规划问题*
303. for (int k = 0; k < G->n; k++)
304. {
305. for (int i = 0; i < G->n; i++)
306. {
307. for (int j = 0; j < G->n; j++)
308. {
309. if (nn[i][k] && nn[k][j])
310. nn[i][j] = 1;
311. }
312. }
313. }
314. *//输出可达矩阵*
315. cout<< "可达矩阵为:" << endl;
316. for (int i = 0; i < G->n; i++)
317. {
318. for (int j = 0; j < G->n; j++)
319. {
320. cout << nn[i][j] << " ";
321. }
322. cout << endl;
323. }
324. }
325. void test()
326. {
327. while (1)
328. {
329. cout << "1.单源最短路径的Dijkstra算法" << endl;
330. cout << "2.全局最短路径的Floyd-Warshall算法" << endl;
331. cout << "3.指定终点的最短路径" << endl;
332. cout << "4.指定一对顶点互相的最短路径" << endl;
333. cout << "5.Warshall算法计算有向图的可达矩阵" << endl;
334. cout << "0.退出程序" << endl;
335. int mode;
336. cout << "请选择模式：" << endl;
337. cin >> mode;
338. switch (mode)
339. {
340. case 1:
341. ManageDijkstra(1);
342. break;
343. case 2:
344. ManageFloyd(2);
345. break;
346. case 3:
347. ManageFloyd(3);
348. break;
349. case 4:
350. ManageFloyd(4);
351. break;
352. case 5:
353. Warshall();
354. break;
355. case 0:
356. return;
357. }
359. }
360. }
361. int main()
362. {
363. test();
364. return 0;
365. }