哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称：数据结构与算法

课程类型：必修

实验项目：图型结构及其应用

实验题目：最短路径算法的实现

实验日期：2020.11.17

班级：1903002

学号：1190200523

姓名：石翔宇

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

**一、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。

**二、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. 利用堆结构（实现的优先级队列），改进和优化Dijkstra算法的实现；
3. 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。
4. 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；
5. 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；
6. （选做）利用Floyd-Warshall算法，计算有向图的可达矩阵，理解可达矩阵的含义；
7. 以文件形式输入图的顶点和边，并显示相应的结果。要求顶点不少于10个，边不少于13个；
8. 软件功能结构安排合理，界面友好，便于使用。

**实验环境：**Windows 10 + Intel i7-9750H + 16GB + MinGW-W64-builds-4.3.5 + VSCode

**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）

1. 物理设计
2. 单源最短路搜索的结果：包括起点，每个点到起点的最近距离。
3. 多源最短路搜索的结果：包括每个点到其他点的最近距离。
4. 多源最短路搜索的路径结果：从某个点出发到其他所有点的最短路径。
5. 图：利用链式前向星存储。
6. 点：用于小根堆，包括节点编号和当前的最短距离。
7. 小根堆：包括堆数组，堆长度，利用堆结构优化Dijkstra算法。
8. 逻辑设计
   1. 主程序流程图



图 1主程序流程图

* 1. 程序模块之间调用关系



图 2程序模块之间调用关系

**四、测试结果**

(斜体为备注内容)



图 3输入的图

1. 输入数据

1 *输入图*

12 17 *图有12个点，编号从1到12，有17条边*

1 2 1 *图的描述*

2 3 5

3 5 2

5 1 7

1 3 3

2 5 4

9 5 8

9 10 3

4 7 11

7 8 6

8 7 4

8 12 4

12 11 8

11 6 9

6 4 13

11 7 2

12 6 10

2 *输出任意两个顶点间的最短路径长度和最短路径*

3 *找出图中每个顶点到某个指定顶点7的最短路径*

7

4 *对于某对顶点9和2，找出9到2和2到9的一条最短路径*

9 2

5 *计算有向图的可达矩阵*

0 *退出*

1. 输出数据

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* *图形界面*

\* \*

\* 0.Exit \*

\* 1.Input And Build \*

\* 2.Get Every Shortest Length And Path \*

\* 3.Get Single Target Shortest Path \*

\* 4.Get Single Shortest Path \*

\* 5.Get Reachable Matrix \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* *输入图，没有输出*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* 0.Exit \*

\* 1.Input And Build \*

\* 2.Get Every Shortest Length And Path \*

\* 3.Get Single Target Shortest Path \*

\* 4.Get Single Shortest Path \*

\* 5.Get Reachable Matrix \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0 1 3 -1 5 -1 -1 -1 -1 -1 -1 -1 *任意两个顶点间的最短路径长度，-1为无法到达*

11 0 5 -1 4 -1 -1 -1 -1 -1 -1 -1

9 10 0 -1 2 -1 -1 -1 -1 -1 -1 -1

-1 -1 -1 0 -1 31 11 17 -1 -1 29 21

7 8 10 -1 0 -1 -1 -1 -1 -1 -1 -1

-1 -1 -1 13 -1 0 24 30 -1 -1 42 34

-1 -1 -1 33 -1 20 0 6 -1 -1 18 10

-1 -1 -1 27 -1 14 4 0 -1 -1 12 4

15 16 18 -1 8 -1 -1 -1 0 3 -1 -1

-1 -1 -1 -1 -1 -1 -1 -1 -1 0 -1 -1

-1 -1 -1 22 -1 9 2 8 -1 -1 0 12

-1 -1 -1 23 -1 10 10 16 -1 -1 8 0

1 TO 2 : 1 2 *任意两个顶点间的最短路径*

1 TO 3 : 1 3

1 TO 5 : 1 3 5

2 TO 1 : 2 5 1

2 TO 3 : 2 3

2 TO 5 : 2 5

3 TO 1 : 3 5 1

3 TO 2 : 3 5 1 2

3 TO 5 : 3 5

4 TO 6 : 4 7 8 12 6

4 TO 7 : 4 7

4 TO 8 : 4 7 8

4 TO 11 : 4 7 8 12 11

4 TO 12 : 4 7 8 12

5 TO 1 : 5 1

5 TO 2 : 5 1 2

5 TO 3 : 5 1 3

6 TO 4 : 6 4

6 TO 7 : 6 4 7

6 TO 8 : 6 4 7 8

6 TO 11 : 6 4 7 8 12 11

6 TO 12 : 6 4 7 8 12

7 TO 4 : 7 8 12 6 4

7 TO 6 : 7 8 12 6

7 TO 8 : 7 8

7 TO 11 : 7 8 12 11

7 TO 12 : 7 8 12

8 TO 4 : 8 12 6 4

8 TO 6 : 8 12 6

8 TO 7 : 8 7

8 TO 11 : 8 12 11

8 TO 12 : 8 12

9 TO 1 : 9 5 1

9 TO 2 : 9 5 1 2

9 TO 3 : 9 5 1 3

9 TO 5 : 9 5

9 TO 10 : 9 10

11 TO 4 : 11 6 4

11 TO 6 : 11 6

11 TO 7 : 11 7

11 TO 8 : 11 7 8

11 TO 12 : 11 7 8 12

12 TO 4 : 12 6 4

12 TO 6 : 12 6

12 TO 7 : 12 11 7

12 TO 8 : 12 11 7 8

12 TO 11 : 12 11

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* 0.Exit \*

\* 1.Input And Build \*

\* 2.Get Every Shortest Length And Path \*

\* 3.Get Single Target Shortest Path \*

\* 4.Get Single Shortest Path \*

\* 5.Get Reachable Matrix \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Please input your target: *图中每个顶点到顶点7的最短路径*

4 TO 7 : 4 7

6 TO 7 : 6 4 7

7 TO 7 : 7

8 TO 7 : 8 7

11 TO 7 : 11 7

12 TO 7 : 12 11 7

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* 0.Exit \*

\* 1.Input And Build \*

\* 2.Get Every Shortest Length And Path \*

\* 3.Get Single Target Shortest Path \*

\* 4.Get Single Shortest Path \*

\* 5.Get Reachable Matrix \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Please input your starting point and target:

5 TO 2 : 5 1 2 *对于顶点5和2，找出5到2和2到5的一条最短路径*

2 TO 5 : 2 5

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* 0.Exit \*

\* 1.Input And Build \*

\* 2.Get Every Shortest Length And Path \*

\* 3.Get Single Target Shortest Path \*

\* 4.Get Single Shortest Path \*

\* 5.Get Reachable Matrix \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1 1 1 0 1 0 0 0 0 0 0 0 *可达矩阵*

1 1 1 0 1 0 0 0 0 0 0 0

1 1 1 0 1 0 0 0 0 0 0 0

0 0 0 1 0 1 1 1 0 0 1 1

1 1 1 0 1 0 0 0 0 0 0 0

0 0 0 1 0 1 1 1 0 0 1 1

0 0 0 1 0 1 1 1 0 0 1 1

0 0 0 1 0 1 1 1 0 0 1 1

1 1 1 0 1 0 0 0 1 1 0 0

0 0 0 0 0 0 0 0 0 1 0 0

0 0 0 1 0 1 1 1 0 0 1 1

0 0 0 1 0 1 1 1 0 0 1 1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* 0.Exit \*

\* 1.Input And Build \*

\* 2.Get Every Shortest Length And Path \*

\* 3.Get Single Target Shortest Path \*

\* 4.Get Single Shortest Path \*

\* 5.Get Reachable Matrix \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

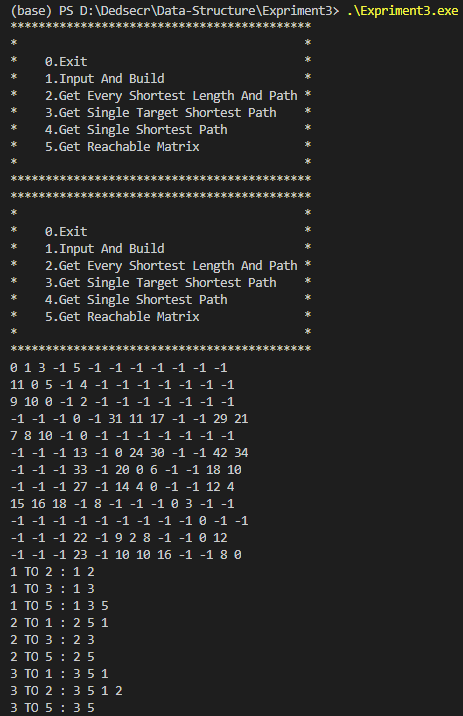


图 4部分运行结果截图

**五、经验体会与不足**

1. 不足
2. 输入格式对用户较不友好，可改进。
3. 程序鲁棒性较弱
4. 特定格式较多，没有报错提示。
5. 部分算法实现不完美，时间复杂度高
6. 经验体会
   1. 要根据不同任务选择不同算法。
   2. 要合理设计以尽量实现程序模块复用。

**六、附录：源代码（带注释）**

1. #include <iostream>
2. #include <cstdio>
3. #include <algorithm>
4. #include <string>
5. #include <cstring>
6. #include <stack>
7. #include <queue>
8. #include <map>
9. #include <vector>
10. #include <cmath>
11. **using** **namespace** std;
12. **const** **int** MAXN = 50;
13. **const** **int** MAXM = MAXN \* MAXN << 1;
14. //单源最短路搜索的结果
15. **struct** Result\_SingleSource
16. {
17. **int** S, Dis[MAXN];
18. Result\_SingleSource()
19. {
20. **for** (**int** i = 0; i < MAXN; ++i)
21. Dis[i] = 1e9;
22. }
23. };
24. //多源最短路搜索的结果
25. **struct** Result\_AllSource
26. {
27. **int** Dis[MAXN][MAXN];
28. Result\_AllSource()
29. {
30. **for** (**int** i = 0; i < MAXN; ++i)
31. **for** (**int** j = 0; j < MAXN; ++j)
32. Dis[i][j] = 1e9;
33. }
34. };
35. //多源最短路搜索的路径结果
36. **struct** Result\_Path
37. {
38. string Path[MAXN];
39. Result\_Path()
40. {
41. **for** (**int** i = 0; i < MAXN; ++i)
42. Path[i] = "";
43. }
44. };
45. //图的存储结构
46. **struct** Graph
47. {
48. **int** n, m;
49. **int** Next[MAXM], To[MAXM], V[MAXM], Head[MAXN];
50. **int** Tot;
51. Graph()
52. {
53. Tot = 1;
54. memset(Next, 0, **sizeof**(Next));
55. memset(To, 0, **sizeof**(To));
56. memset(V, 0, **sizeof**(V));
57. memset(Head, 0, **sizeof**(Head));
58. }
59. **void** Add(**int** x, **int** y, **int** v)
60. {
61. Next[++Tot] = Head[x];
62. Head[x] = Tot;
63. To[Tot] = y;
64. V[Tot] = v;
65. }
66. };
67. //点
68. **struct** Node
69. {
70. **int** x, Dis;
71. Node() {}
72. Node(**int** \_x, **int** \_D)
73. {
74. x = \_x;
75. Dis = \_D;
76. }
77. };
78. **class** Priority\_Queue //小根堆
79. {
80. **private**:
81. Node Heap[MAXN << 2];
82. **int** Length;
84. **public**:
85. Priority\_Queue()
86. {
87. Length = 0;
88. }
89. **bool** empty()
90. {
91. **return** !Length;
92. }
93. **int** size()
94. {
95. **return** Length;
96. }
97. **void** push(Node x)
98. {
99. Heap[++Length] = x;
100. **int** Pos = Length;
101. **while** (Pos > 1)
102. {
103. **if** (Heap[Pos >> 1].Dis > Heap[Pos].Dis)
104. swap(Heap[Pos >> 1], Heap[Pos]);
105. Pos >>= 1;
106. }
107. }
108. Node top()
109. {
110. **return** Heap[1];
111. }
112. **void** pop()
113. {
114. **if** (!Length)
115. {
116. cerr << "Error in" << \_\_LINE\_\_;
117. exit(-1);
118. }
119. swap(Heap[1], Heap[Length]);
120. Length--;
121. **if** (Length == 0)
122. **return**;
123. **int** Pos = 1;
124. **while** (Length >= (Pos << 1))
125. {
126. **if** ((Pos << 1 | 1) <= Length)
127. {
128. **if** (Heap[Pos].Dis > min(Heap[Pos << 1].Dis, Heap[Pos << 1 | 1].Dis))
129. {
130. **if** (Heap[Pos << 1].Dis < Heap[Pos << 1 | 1].Dis)
131. swap(Heap[Pos << 1], Heap[Pos]), Pos <<= 1;
132. **else**
133. swap(Heap[Pos << 1 | 1], Heap[Pos]), Pos = Pos << 1 | 1;
134. }
135. **else**
136. **break**;
137. }
138. **else**
139. {
140. **if** (Heap[Pos].Dis > Heap[Pos << 1].Dis)
141. swap(Heap[Pos << 1], Heap[Pos]), Pos <<= 1;
142. **else**
143. **break**;
144. }
145. }
146. }
147. };
148. Priority\_Queue Q;
149. Graph G;
150. **bool** Vis[MAXN];
151. **int** Stack[MAXN], StackP;
152. //输入图
153. **void** InputAndBuild()
154. {
155. cin >> G.n >> G.m;
156. **for** (**int** i = 1; i <= G.m; ++i)
157. {
158. **int** x, y, v;
159. cin >> x >> y >> v;
160. G.Add(x, y, v);
161. }
162. }
163. //把AllSource转换成SingleSource
164. Result\_SingleSource AllSource2SingleSource(Result\_AllSource AS, **int** S)
165. {
166. Result\_SingleSource Res;
167. Res.S = S;
168. **for** (**int** i = 0; i < MAXN; ++i)
169. Res.Dis[i] = AS.Dis[S][i];
170. **return** Res;
171. }
172. //Dijkstra算法实现
173. Result\_SingleSource Dijkstra(**int** S)
174. {
175. memset(Vis, 0, **sizeof**(Vis));
176. Result\_SingleSource Res;
177. **while** (!Q.empty())
178. Q.pop();
179. Res.S = S;
180. Q.push(Node(S, 0));
181. Res.Dis[S] = 0;
182. **while** (!Q.empty())
183. {
184. Node Now = Q.top();
185. Q.pop();
186. **int** x = Now.x;
187. **if**(Vis[x])
188. **continue**;
189. Vis[x] = 1;
190. **for** (**int** i = G.Head[x]; i; i = G.Next[i])
191. **if** (Res.Dis[G.To[i]] > Res.Dis[x] + G.V[i])
192. {
193. Res.Dis[G.To[i]] = Res.Dis[x] + G.V[i];
194. **if** (!Vis[G.To[i]])
195. Q.push(Node(G.To[i], Res.Dis[G.To[i]]));
196. }
197. }
198. **return** Res;
199. }
200. //Floyd算法实现
201. Result\_AllSource Floyd()
202. {
203. Result\_AllSource Res;
204. **for** (**int** i = 0; i < MAXN; ++i)
205. Res.Dis[i][i] = 0;
206. **for** (**int** x = 0; x < MAXN; ++x)
207. **for** (**int** i = G.Head[x]; i; i = G.Next[i])
208. Res.Dis[x][G.To[i]] = G.V[i];
209. **for** (**int** k = 0; k < MAXN; ++k)
210. **for** (**int** i = 0; i < MAXN; ++i)
211. **for** (**int** j = 0; j < MAXN; ++j)
212. Res.Dis[i][j] = min(Res.Dis[i][j], Res.Dis[i][k] + Res.Dis[k][j]);
213. **return** Res;
214. }
215. //找路径的DFS
216. **void** FindPathDFS(**int** x, Result\_Path &Res, Result\_SingleSource &SS, **int** Sum)
217. {
218. Vis[x] = 1;
219. Stack[++StackP] = x;
221. **for** (**int** i = 1; i <= StackP; ++i)
222. {
223. Res.Path[x] += to\_string(Stack[i]);
224. Res.Path[x] += ' ';
225. }
227. **for** (**int** i = G.Head[x]; i; i = G.Next[i])
228. **if** (Sum + G.V[i] == SS.Dis[G.To[i]] && !Vis[G.To[i]])
229. {
230. FindPathDFS(G.To[i], Res, SS, Sum + G.V[i]);
231. }
233. StackP--;
234. }
235. //找从x开始的所有最短路径
236. Result\_Path FindPath(**int** x, Result\_SingleSource SS)
237. {
238. StackP = 0;
239. memset(Vis, 0, **sizeof**(Vis));
240. Result\_Path Res;
241. FindPathDFS(x, Res, SS, 0);
242. **return** Res;
243. }
244. //输出S到每个顶点的路径
245. **void** Print\_Every\_Result\_Path(**int** S, Result\_Path &Res)
246. {
247. **for** (**int** i = 1; i <= G.n; ++i)
248. {
249. **if** (S == i)
250. **continue**;
251. **if** (Res.Path[i].length() == 0)
252. **continue**;
253. cout << S << " TO " << i << " : "  << Res.Path[i] << "\n";
254. }
255. }
256. //输出S到Target的路径
257. **void** Print\_Single\_Result\_Path(**int** S, **int** Target, Result\_Path &Res)
258. {
259. **if** (Res.Path[Target].length() == 0)
260. **return**;
261. cout << S << " TO " << Target << " : " << Res.Path[Target] << "\n";
262. }
263. //输出任意两个顶点间的最短路径长度和最短路径
264. **void** GetEveryShortestLengthAndPath()
265. {
266. Result\_AllSource Res = Floyd();
267. **for** (**int** i = 1; i <= G.n; ++i, cout << '\n')
268. **for** (**int** j = 1; j <= G.n; ++j)
269. **if**(Res.Dis[i][j] >= 1e9)
270. cout << "-1 ";
271. **else**
272. cout << Res.Dis[i][j] << " ";
274. **for** (**int** x = 1; x <= G.n; ++x)
275. {
276. Result\_Path RP = FindPath(x, AllSource2SingleSource(Res, x));
277. Print\_Every\_Result\_Path(x, RP);
278. }
279. }
280. //找出图中每个顶点到某个指定顶点Target的最短路径
281. **void** GetSingleTargetShortestPath(**int** Target)
282. {
283. Result\_AllSource Res = Floyd();
284. **for** (**int** x = 1; x <= G.n; ++x)
285. {
286. Result\_Path RP = FindPath(x, AllSource2SingleSource(Res, x));
287. Print\_Single\_Result\_Path(x, Target, RP);
288. }
289. }
290. //对于某对顶点S和T，找出S到T和T到S的一条最短路径
291. **void** GetSingleShortestPath(**int** S, **int** T)
292. {
293. Result\_SingleSource Res = Dijkstra(S);
294. Result\_Path RP = FindPath(S, Res);
295. Print\_Single\_Result\_Path(S, T, RP);
297. Res = Dijkstra(T);
298. RP = FindPath(T, Res);
299. Print\_Single\_Result\_Path(T, S, RP);
300. }
301. //计算有向图的可达矩阵
302. **void** GetReachableMatrix()
303. {
304. Result\_AllSource Res = Floyd();
305. **for** (**int** i = 1; i <= G.n; ++i, cout << '\n')
306. **for** (**int** j = 1; j <= G.n; ++j)
307. **if** (Res.Dis[i][j] >= 1e9)
308. cout << "0 ";
309. **else**
310. cout << "1 ";
311. }
312. **int** main()
313. {
314. //freopen("Expriment3\_In.txt", "r", stdin);
315. //freopen("Expriment3\_Out.txt", "w", stdout);
316. **while**(1)
317. {
318. **int** OP;
319. printf(
320. "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"
321. "\*                                         \*\n"
322. "\*    0.Exit                               \*\n"
323. "\*    1.Input And Build                    \*\n"
324. "\*    2.Get Every Shortest Length And Path \*\n"
325. "\*    3.Get Single Target Shortest Path    \*\n"
326. "\*    4.Get Single Shortest Path           \*\n"
327. "\*    5.Get Reachable Matrix               \*\n"
328. "\*                                         \*\n"
329. "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");
330. cin >> OP;
331. **if**(OP == 0)
332. **break**;
333. **else** **if**(OP == 1)
334. {
335. InputAndBuild();
336. }
337. **else** **if**(OP == 2)
338. {
339. GetEveryShortestLengthAndPath();
340. }
341. **else** **if**(OP == 3)
342. {
343. puts("Please input your target:");
344. **int** Target;
345. cin >> Target;
346. GetSingleTargetShortestPath(Target);
347. }
348. **else** **if**(OP == 4)
349. {
350. puts("Please input your starting point and target:");
351. **int** S, T;
352. cin >> S >> T;
353. GetSingleShortestPath(S, T);
354. }
355. **else** **if**(OP == 5)
356. {
357. GetReachableMatrix();
358. }
359. }
360. **return** 0;
361. }