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：图型结构的应用算法

（实验选题1：最短路径算法的实现）

实验日期：2020.11.17

班级：1903002

学号：1190200526

姓名：沈城有

|  |  |  |
| --- | --- | --- |
| 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。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。

**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 Visual Studio 2019

**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

（1）使用邻接矩阵存储有向图；

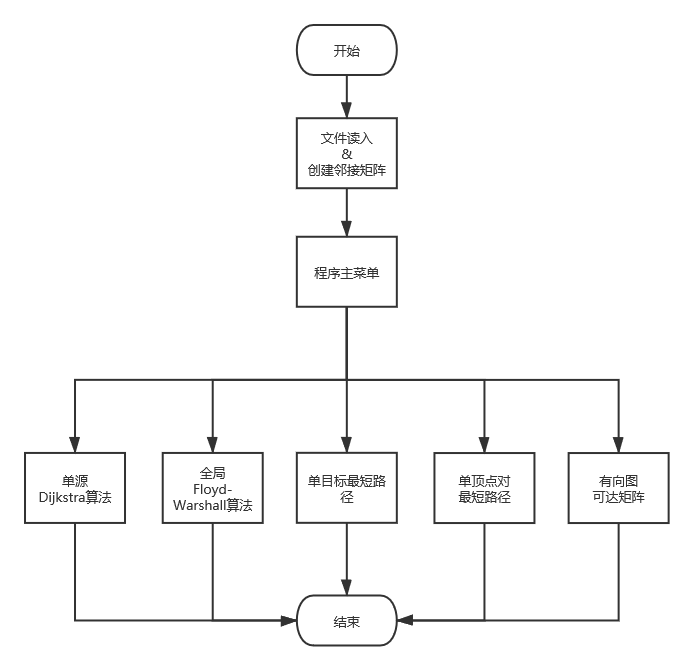
（2）使用最小堆存储权值和顶点序号，优化了Dijkstra算法；

（3）各算法实现函数使用一维或二维数组作为辅助变量，并存储最短路径相关信息。

注：具体实现请查看附录源代码。

2. 逻辑设计

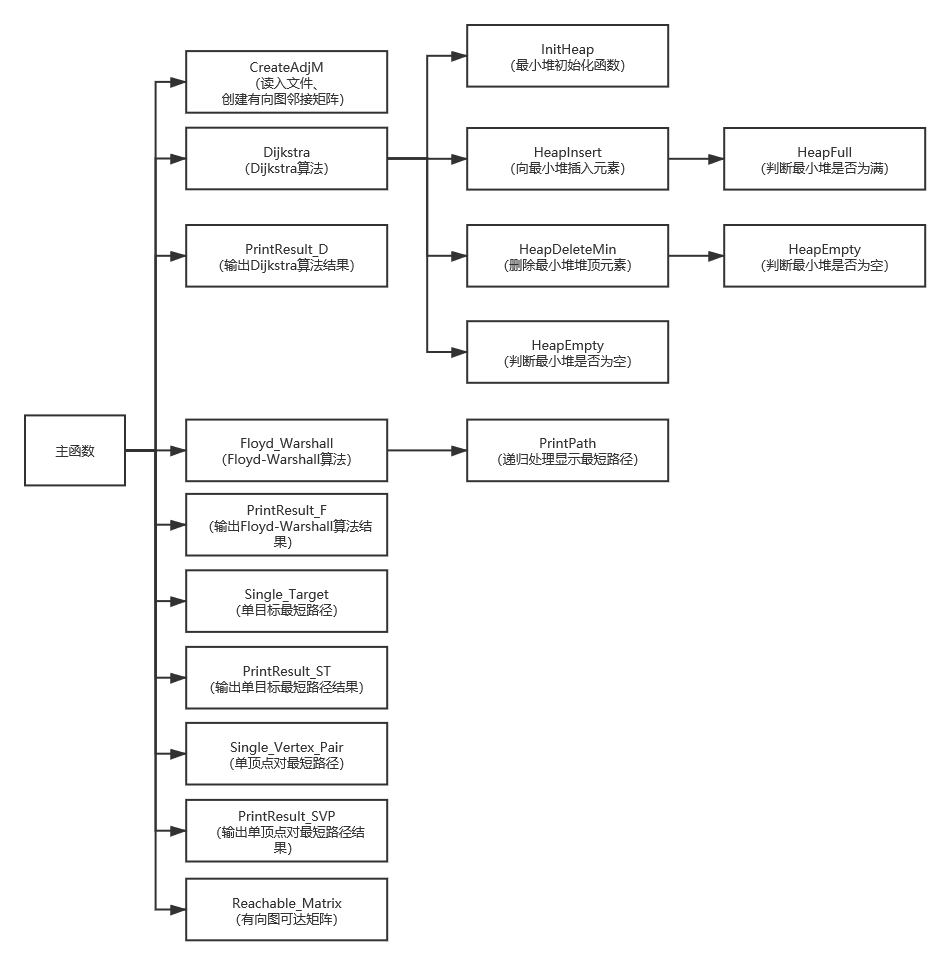
（1）主程序流程图



**图1 主程序流程图**

（2）各程序模块间调用关系：

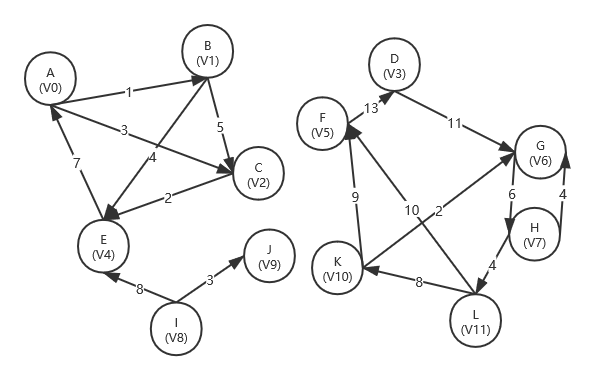
调用关系图见下页。

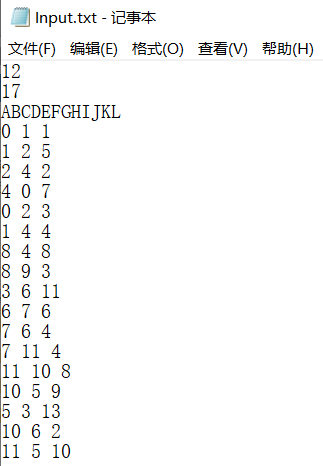


**图2 调用关系图**

**四、测试结果**

1、输入数据

 按照作业要求，自己设计了12个顶点、17条边的有向图如下图。

输入文件内容：

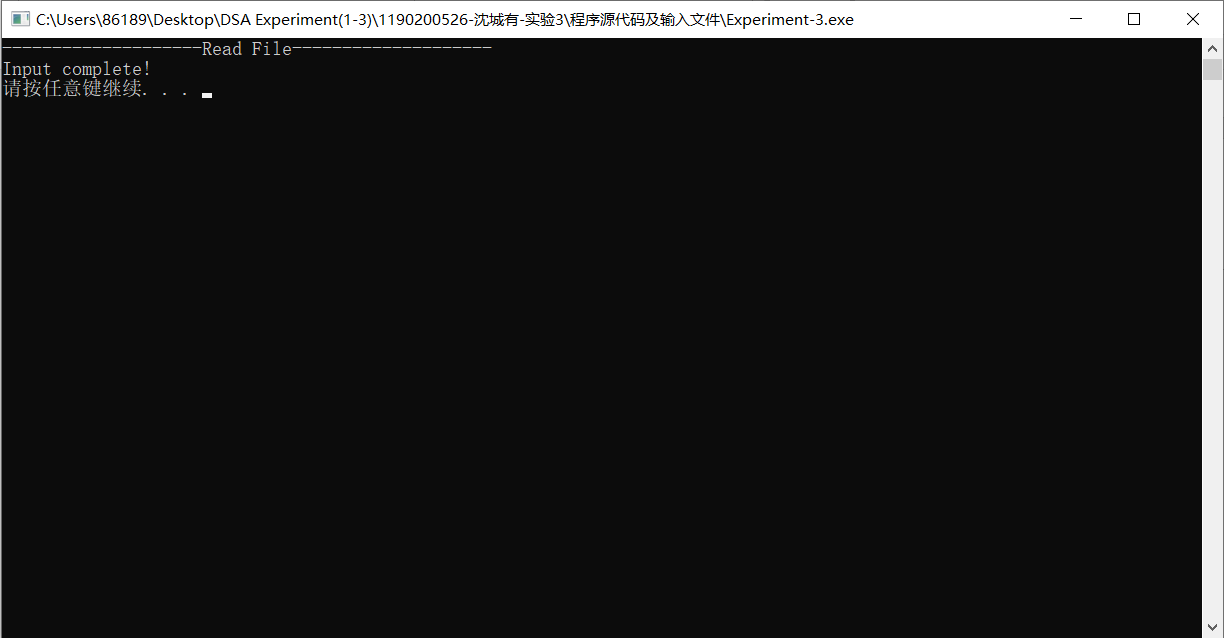
数据含义：

第一行、第二行：顶点数、边数；

第三行：各顶点名称（或数据，程序中定义为char）；

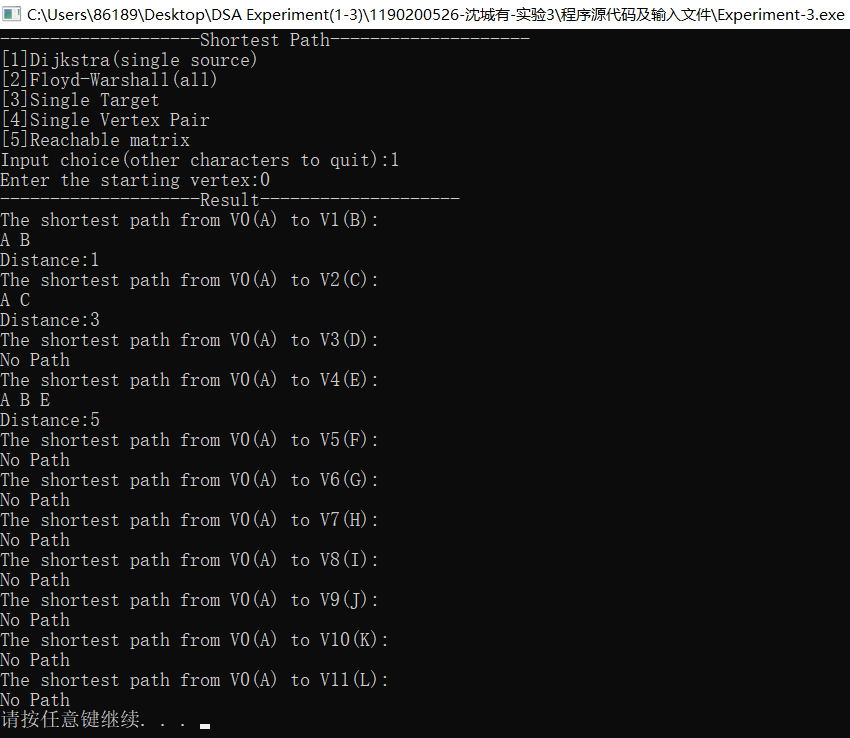
第四行至末尾：边的出发顶点、边的指向顶点、边的权值。

2、文件读入数据

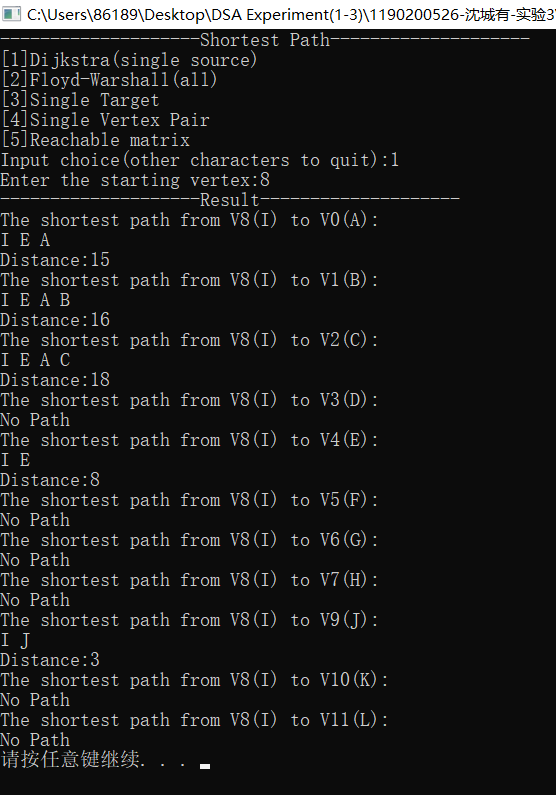


3、Dijkstra算法

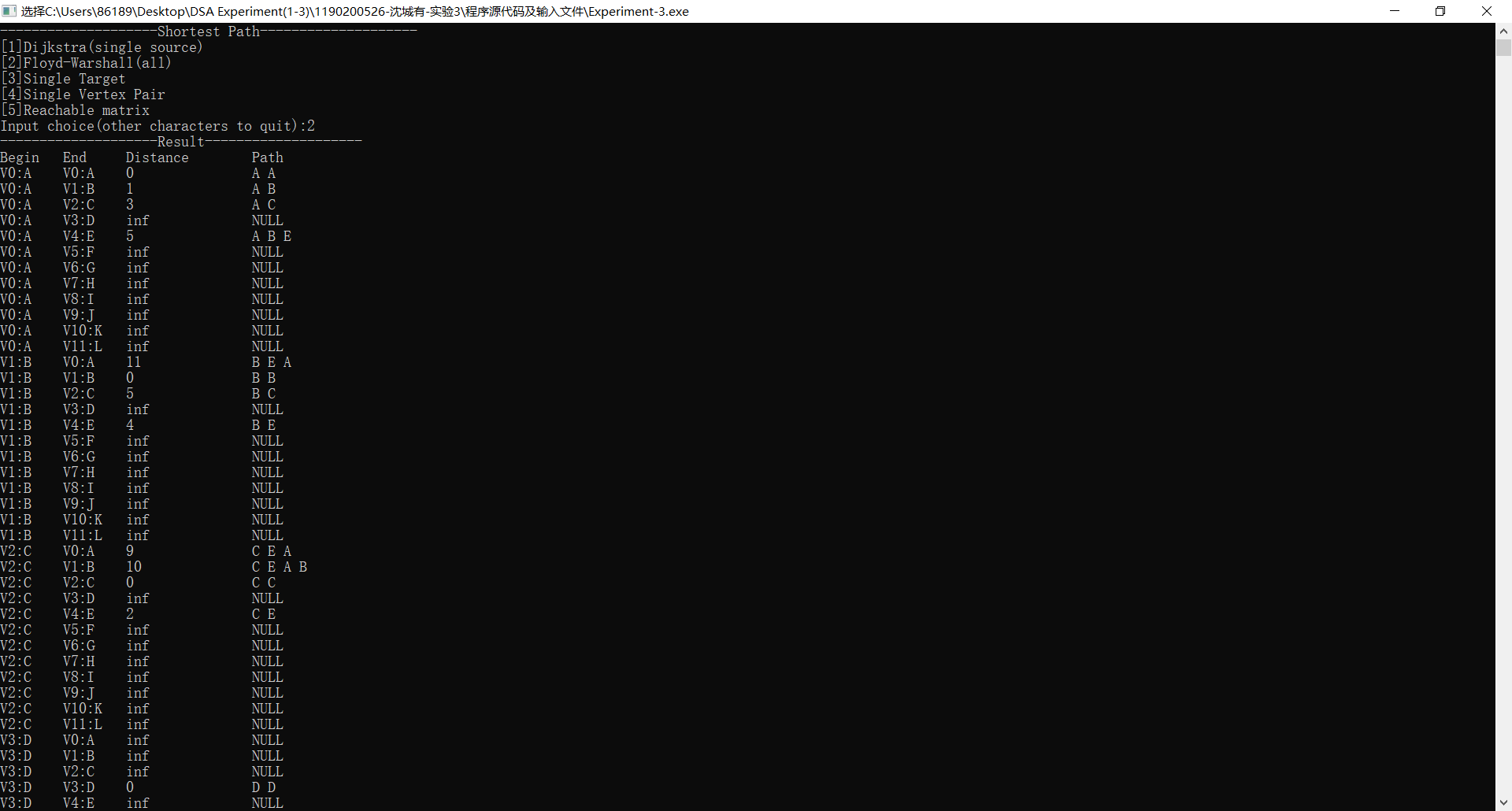
（1）0号顶点为起始点



（2）8号顶点为起始点

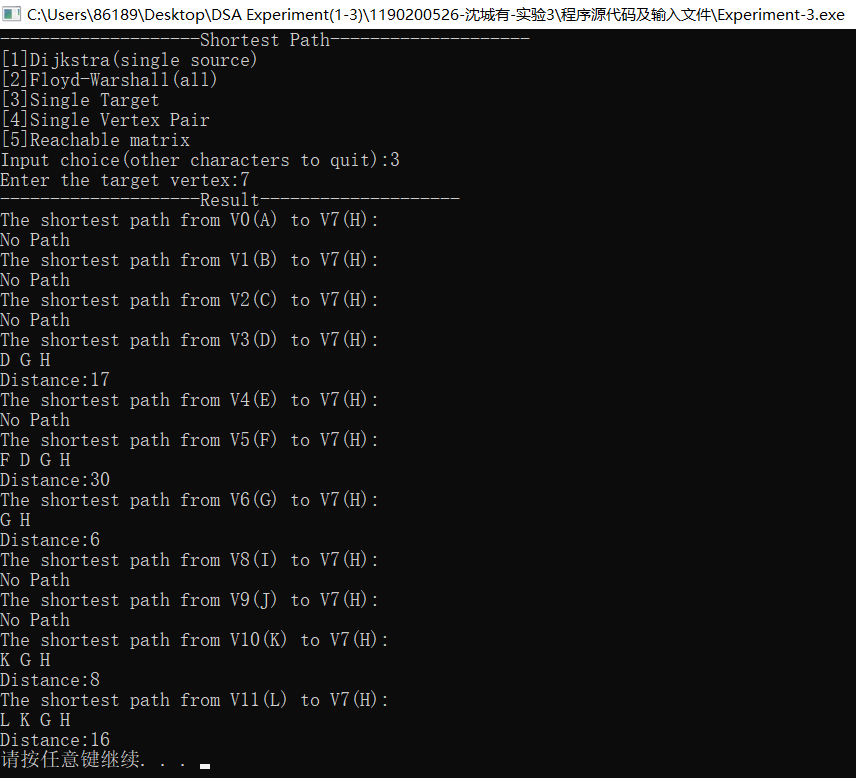


4、Floyd-Warshall算法

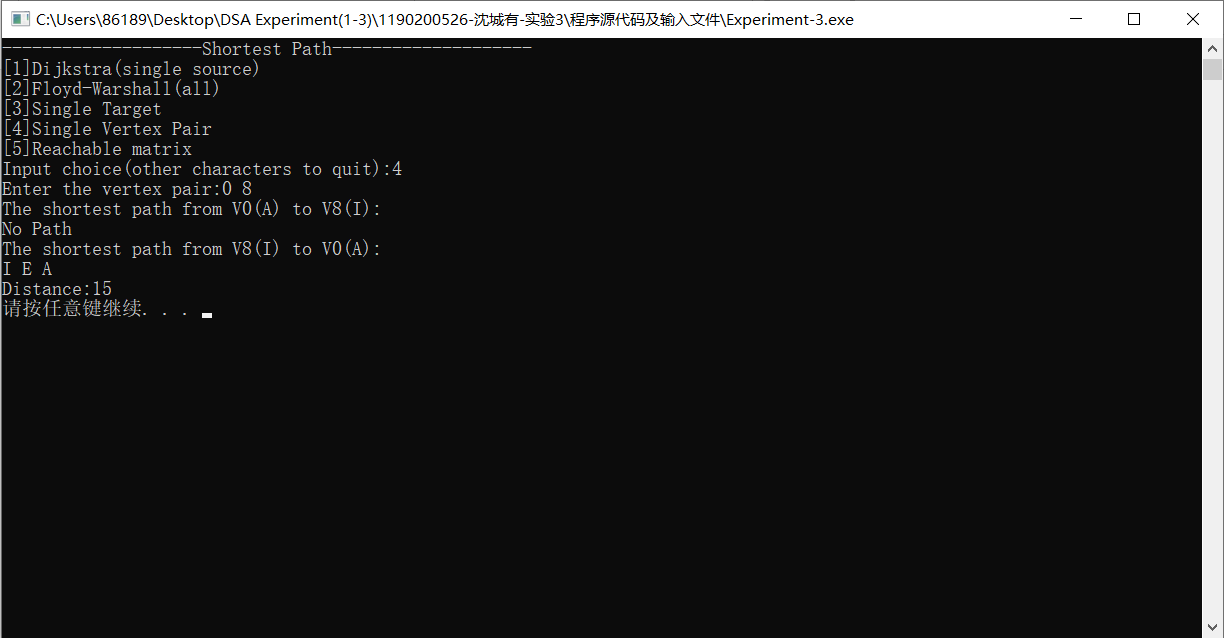


结果较长，此处不再展示。

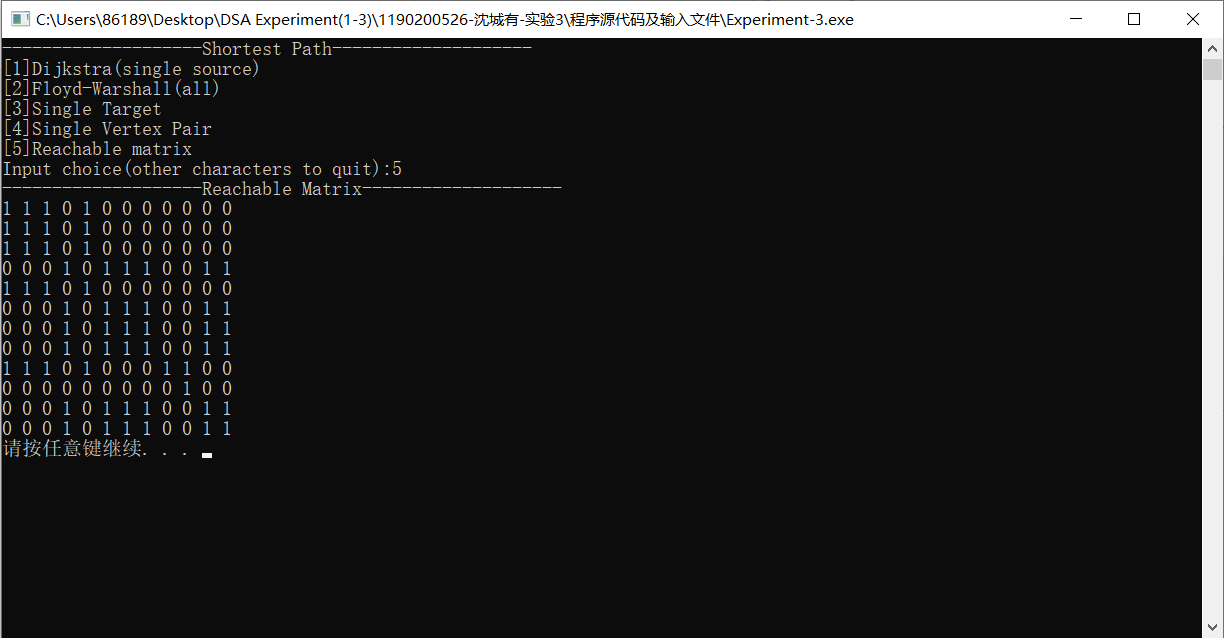
5、单目标最短路径



6、单顶点对最短路径



7、有向图可达矩阵



**五、经验体会与不足**

收获：1、理解了求最短路径的两大算法，并初步实现了灵活运用；

2、通过堆优化提高了Dijkstra算法的效率，掌握了如何用最小堆实现优先队列；

3、锻炼了调试和分析代码的能力，减少了算法示例中的部分不必要操作，提高了整体性能。

不足：1、部分代码内容重复，有缩减内容的空间；

2、Floyd-Warshall算法处理显示最短路径时递归消耗内存较多。

**六、附录：源代码（带注释）**

1. /\*
2. \* 实验3 图型结构及其应用
3. \* 实验题目：图型结构的应用算法
4. \* 实验选题1：最短路径算法的实现
5. \* 学号：1190200526
6. \* 姓名：沈城有
7. \*/
8. #include <iostream>
9. #include <fstream>
10. #include <cstdio>
11. **using** **namespace** std;
13. #define Max\_N 50
14. #define VertexData char         //顶点数据类型
15. #define EdgeData int            //边数据类型
16. #define Inf 1e8                 //无穷大定义
17. #define InputDir "Input.txt"    //文件输入路径
19. //邻接矩阵数据结构
20. **typedef** **struct**
21. {
22. VertexData vertex[Max\_N];     //顶点表
23. EdgeData edge[Max\_N][Max\_N];  //边表
24. **int** n, e;                     //图的顶点数和边数
25. }AdjMatrix;
27. //最小堆数据结构实现
28. **typedef** **struct** nodedata
29. {
30. **int** num;
31. **int** weight;
32. }NodeData;
34. **typedef** **struct** minheap
35. {
36. NodeData elem[Max\_N + 1];   //编号+权值
37. **int** n;
38. }MinHeap;
40. //最小堆结构（优先级队列）相关基本操作声明
41. **void** InitHeap(MinHeap& heap);
42. **bool** HeapEmpty(MinHeap heap);
43. **bool** HeapFull(MinHeap heap);
44. **void** HeapInsert(MinHeap& heap, NodeData elem);
45. **void** HeapDeleteMin(MinHeap& heap, **int**& num);
46. //功能函数声明
47. AdjMatrix\* CreateAdjM(**void**);                                                                            //创建用邻接矩阵表示的有向图
48. **void** Dijkstra(AdjMatrix\* adjm, MinHeap& heap, **int** start, **bool** S[], **int** pre[], **int** dis[]);               //Dijkstra算法（堆优化）
49. **void** Floyd\_Warshall(AdjMatrix\* adjm, EdgeData e[][Max\_N], **int** path[][Max\_N]);                           //Floyd-Warshall算法
50. **void** Single\_Target(AdjMatrix\* adjm, MinHeap& heap, **bool** S[], **int** pre[], **int** dis[], **int** target);         //单目标最短路径（Dijkstra算法修改）
51. **void** Single\_Vertex\_Pair(AdjMatrix\* adjm, MinHeap& heap, **bool** S[], **int** pre[], **int** dis[], **int** s, **int** t);  //单顶点对最短路径（Dijkstra算法修改）
52. **void** Reachable\_Matrix(AdjMatrix\* adjm, EdgeData m[][Max\_N]);                                            //计算并显示可达矩阵（Floyd-Warshall算法修改）
53. **void** PrintResult\_D(AdjMatrix\* adjm, **int** start, **bool** S[], **int** pre[], **int** dis[]);                         //屏幕显示Dijkstra算法结果函数
54. **void** PrintPath(AdjMatrix\* adjm, **int** path[][Max\_N], **int** i, **int** j);                                       //处理并屏幕显示最短路径中各点（递归）
55. **void** PrintResult\_F(AdjMatrix\* adjm, EdgeData e[][Max\_N], **int** path[][Max\_N]);                            //屏幕显示Floyd-Warshall算法结果函数
56. **void** PrintResult\_ST(AdjMatrix\* adjm, **bool** S[], **int** pre[], **int** dis[], **int** target);                       //屏幕显示单目标最短路径结果函数
57. **void** PrintResult\_SVP(AdjMatrix\* adjm, **bool** S[], **int** pre[], **int** dis[], **int** s, **int** t);                    //屏幕显示单顶点对最短路径结果函数
59. **void** InitHeap(MinHeap& heap)
60. {
61. heap.n = 0;
62. }
64. **bool** HeapEmpty(MinHeap heap)
65. {
66. **return** (!heap.n);
67. }
69. **bool** HeapFull(MinHeap heap)
70. {
71. **return** (heap.n == Max\_N);
72. }
74. **void** HeapInsert(MinHeap& heap, NodeData e)
75. {
76. **int** i;
77. **if** (HeapFull(heap))
78. {
79. cout << "Heap is full!" << endl;
80. **return**;
81. }
82. i = ++heap.n;
83. **while** ((i != 1) && (e.weight < heap.elem[i / 2].weight))
84. {
85. heap.elem[i] = heap.elem[i / 2];
86. i /= 2;
87. }
88. heap.elem[i] = e;
89. }
91. **void** HeapDeleteMin(MinHeap& heap, **int**& num)
92. {
93. **int** parent = 1, child = 2;
94. NodeData temp;
95. **if** (!HeapEmpty(heap))
96. {
97. num = heap.elem[1].num;
98. temp = heap.elem[heap.n--];
99. **while** (child <= heap.n)
100. {
101. **if** ((child < heap.n) && (heap.elem[child].weight > heap.elem[child + 1].weight))
102. ++child;  //选更小的
103. **if** (temp.weight <= heap.elem[child].weight)
104. **break**;
105. heap.elem[parent] = heap.elem[child];
106. parent = child;
107. child \*= 2;
108. }
109. }
110. **else**
111. {
112. cout << "Heap is empty!" << endl;
113. **return**;
114. }
115. heap.elem[parent] = temp;
116. }
118. //创建用邻接矩阵表示的有向图
119. AdjMatrix\* CreateAdjM(**void**)
120. {
121. AdjMatrix\* adj = **new** AdjMatrix();
122. cin >> adj->n;
123. cin >> adj->e;
124. **for** (**int** i = 0; i < adj->n; ++i)
125. cin >> adj->vertex[i];
126. **for** (**int** i = 0; i < adj->n; ++i)
127. {
128. **for** (**int** j = 0; j < adj->n; ++j)
129. {
130. adj->edge[i][j] = (**int**)Inf;  //邻接矩阵初始化
131. }
132. }
133. **for** (**int** i = 0; i < adj->e; ++i)
134. {
135. **int** a, b, c;
136. cin >> a >> b >> c;
137. adj->edge[a][b] = c;
138. }
139. cout << "Input complete!" << endl;
140. **return** adj;
141. }
143. //Dijkstra算法（堆优化）
144. **void** Dijkstra(AdjMatrix\* adjm, MinHeap& heap, **int** start, **bool** S[], **int** pre[], **int** dis[])
145. {
146. NodeData temp;
147. **int** min = start;
148. InitHeap(heap);
149. **for** (**int** i = 0; i < adjm->n; ++i)
150. S[i] = **false**;  //集合S初始化
151. S[start] = **true**;
152. **for** (**int** i = 0; i < adjm->n; ++i)
153. {
154. pre[i] = start;
155. dis[i] = adjm->edge[start][i];
156. **if** (dis[i] < Inf)
157. {
158. temp.weight = dis[i];
159. temp.num = i;
160. HeapInsert(heap, temp);
161. }
162. }
163. **while**(1)
164. {
165. **if** (HeapEmpty(heap))
166. **break**;
167. HeapDeleteMin(heap, min);
168. **if** (S[min])
169. {
170. **if** (HeapEmpty(heap))
171. **break**;
172. **else**
173. HeapDeleteMin(heap, min);
174. }
175. S[min] = **true**;
176. **for** (**int** i = 0; i < adjm->n; ++i)
177. {
178. **if** (!S[i])
179. {
180. **if** ((dis[min] + adjm->edge[min][i]) < dis[i])
181. {
182. pre[i] = min;
183. dis[i] = dis[min] + adjm->edge[min][i];  //更新距离
184. temp.num = i;
185. temp.weight = dis[i];
186. HeapInsert(heap, temp);
187. }
188. }
189. }
190. }
191. }
193. //处理并屏幕显示最短路径中各点（递归）
194. **void** PrintPath(AdjMatrix\* adjm, **int** path[][Max\_N], **int** i, **int** j)
195. {
196. **int** k;
197. k = path[i][j];
198. **if** (k != -1)
199. {
200. PrintPath(adjm, path, i, k);
201. cout << adjm->vertex[k] << " ";
202. PrintPath(adjm, path, k, j);
203. }
204. }
206. //Floyd-Warshall算法
207. **void** Floyd\_Warshall(AdjMatrix\* adjm, EdgeData e[][Max\_N], **int** path[][Max\_N])
208. {
209. //设定最短路径矩阵初值
210. **for** (**int** i = 0; i < adjm->n; ++i)
211. {
212. **for** (**int** j = 0; j < adjm->n; ++j)
213. {
214. **if** (i == j)
215. e[i][j] = 0;
216. **else**
217. e[i][j] = adjm->edge[i][j];
218. path[i][j] = -1;
219. }
220. }
221. **for** (**int** k = 0; k < adjm->n; ++k)
222. {
223. **for** (**int** i = 0; i < adjm->n; ++i)
224. {
225. **for** (**int** j = 0; j < adjm->n; ++j)
226. {
227. **if** (e[i][k] + e[k][j] < e[i][j])
228. {
229. e[i][j] = e[i][k] + e[k][j];
230. path[i][j] = k;
231. }
232. }
233. }
234. }
235. }
237. //单目标最短路径（Dijkstra算法修改）
238. **void** Single\_Target(AdjMatrix\* adjm, MinHeap& heap, **bool** S[], **int** pre[], **int** dis[], **int** target)
239. {//主要是逆向找路径
240. NodeData temp;
241. **int** min = target;
242. InitHeap(heap);
243. **for** (**int** i = 0; i < adjm->n; ++i)
244. S[i] = **false**;  //集合S初始化
245. S[target] = **true**;
246. **for** (**int** i = 0; i < adjm->n; ++i)
247. {
248. pre[i] = target;
249. dis[i] = adjm->edge[i][target];
250. **if** (dis[i] < Inf)
251. {
252. temp.weight = dis[i];
253. temp.num = i;
254. HeapInsert(heap, temp);
255. }
256. }
257. **while** (1)
258. {
259. **if** (HeapEmpty(heap))
260. **break**;
261. HeapDeleteMin(heap, min);
262. **if** (S[min])
263. {
264. **if** (HeapEmpty(heap))
265. **break**;
266. **else**
267. HeapDeleteMin(heap, min);
268. }
269. S[min] = **true**;
270. **for** (**int** i = 0; i < adjm->n; ++i)
271. {
272. **if** (!S[i])
273. {
274. **if** ((dis[min] + adjm->edge[i][min]) < dis[i])
275. {
276. pre[i] = min;
277. dis[i] = dis[min] + adjm->edge[i][min];  //更新距离
278. temp.num = i;
279. temp.weight = dis[i];
280. HeapInsert(heap, temp);
281. }
282. }
283. }
284. }
285. }
287. //单顶点对最短路径（Dijkstra算法修改）
288. **void** Single\_Vertex\_Pair(AdjMatrix\* adjm, MinHeap& heap, **bool** S[], **int** pre[], **int** dis[], **int** s, **int** t)
289. {
290. NodeData temp;
291. **int** min;
292. min = s;
293. InitHeap(heap);
294. **for** (**int** i = 0; i < adjm->n; ++i)
295. S[i] = **false**;  //集合S初始化
296. S[s] = **true**;
297. **for** (**int** i = 0; i < adjm->n; ++i)
298. {
299. pre[i] = s;
300. dis[i] = adjm->edge[s][i];
301. **if** (dis[i] < Inf)
302. {
303. temp.weight = dis[i];
304. temp.num = i;
305. HeapInsert(heap, temp);
306. }
307. }
308. **while** (1)
309. {
310. **if** (HeapEmpty(heap))
311. **break**;
312. HeapDeleteMin(heap, min);
313. **if** (S[min])
314. {
315. **if** (HeapEmpty(heap) || min == t)
316. **break**;
317. **else**
318. HeapDeleteMin(heap, min);
319. }
320. S[min] = **true**;
321. **for** (**int** i = 0; i < adjm->n; ++i)
322. {
323. **if** (!S[i])
324. {
325. **if** ((dis[min] + adjm->edge[min][i]) < dis[i])
326. {
327. pre[i] = min;
328. dis[i] = dis[min] + adjm->edge[min][i];  //更新距离
329. temp.num = i;
330. temp.weight = dis[i];
331. HeapInsert(heap, temp);
332. }
333. }
334. }
335. }
336. }
338. //计算并显示可达矩阵（Floyd-Warshall算法修改）
339. **void** Reachable\_Matrix(AdjMatrix\* adjm, EdgeData m[][Max\_N])
340. {
341. //设定可达矩阵初值
342. **for** (**int** i = 0; i < adjm->n; ++i)
343. {
344. **for** (**int** j = 0; j < adjm->n; ++j)
345. {
346. **if** (i == j)
347. m[i][j] = 1;
348. **else**
349. m[i][j] = (adjm->edge[i][j] < Inf) ? 1 : 0;
350. }
351. }
352. **for** (**int** k = 0; k < adjm->n; ++k)
353. {
354. **for** (**int** i = 0; i < adjm->n; ++i)
355. {
356. **for** (**int** j = 0; j < adjm->n; ++j)
357. m[i][j] = (m[i][j] || (m[i][k] && m[k][j]));
358. }
359. }
360. //屏幕打印可达矩阵
361. cout << "--------------------Reachable Matrix--------------------" << endl;
362. **for** (**int** i = 0; i < adjm->n; ++i)
363. {
364. **for** (**int** j = 0; j < adjm->n; ++j)
365. cout << m[i][j] << " ";
366. cout << endl;
367. }
368. }
370. //屏幕显示Dijkstra算法结果函数
371. **void** PrintResult\_D(AdjMatrix\* adjm, **int** start, **bool** S[], **int** pre[], **int** dis[])
372. {
373. **char** path[Max\_N];
374. **int** count;
375. cout << "--------------------Result--------------------" << endl;
376. **for** (**int** i = 0; i < adjm->n; ++i)
377. {
378. **if** (i != start)
379. {
380. printf("The shortest path from V%d(%c) to V%d(%c):\n", start, adjm->vertex[start], i, adjm->vertex[i]);
381. **if** (dis[i] == (**int**)Inf)
382. printf("No Path\n");
383. **else**
384. {
385. count = 0;
386. path[count] = adjm->vertex[i];
387. **for** (**int** j = i; j != start; )
388. {
389. path[++count] = adjm->vertex[pre[j]];
390. j = pre[j];
391. }
392. **for** (**int** j = count; j >= 0; --j)
393. cout << path[j] << " ";
394. printf("\nDistance:%d\n", dis[i]);
395. }
396. }
397. }
398. }
400. //屏幕显示Floyd-Warshall算法结果函数
401. **void** PrintResult\_F(AdjMatrix\* adjm, EdgeData e[][Max\_N], **int** path[][Max\_N])
402. {
403. cout << "--------------------Result--------------------" << endl;
404. cout << "Begin\tEnd\tDistance\tPath" << endl;
405. **for** (**int** i = 0; i < adjm->n; ++i)
406. {
407. **for** (**int** j = 0; j < adjm->n; ++j)
408. {
409. cout << "V" << i << ":" << adjm->vertex[i] << "\t";
410. cout << "V" << j << ":" << adjm->vertex[j] << "\t";
411. **if** (e[i][j] < (**int**)Inf)
412. {
413. cout << e[i][j] << "\t\t";
414. cout << adjm->vertex[i] << " ";
415. PrintPath(adjm, path, i, j);
416. cout << adjm->vertex[j] << endl;
417. }
418. **else**
419. {
420. cout << "inf" << "\t\t";
421. cout << "NULL" << endl;
422. }
423. }
424. }
425. }
427. //屏幕显示单目标最短路径结果函数
428. **void** PrintResult\_ST(AdjMatrix\* adjm, **bool** S[], **int** pre[], **int** dis[], **int** target)
429. {
430. cout << "--------------------Result--------------------" << endl;
431. **for** (**int** i = 0; i < adjm->n; ++i)
432. {
433. **if** (i != target)
434. {
435. printf("The shortest path from V%d(%c) to V%d(%c):\n", i, adjm->vertex[i], target, adjm->vertex[target]);
436. **if** (dis[i] == (**int**)Inf)
437. printf("No Path\n");
438. **else**
439. {
440. cout << adjm->vertex[i] << " ";
441. **for** (**int** j = i; j != target; )
442. {
443. cout << adjm->vertex[pre[j]] << " ";
444. j = pre[j];
445. }
446. printf("\nDistance:%d\n", dis[i]);
447. }
448. }
449. }
450. }
452. //屏幕显示单顶点对最短路径结果函数
453. **void** PrintResult\_SVP(AdjMatrix\* adjm, **bool** S[], **int** pre[], **int** dis[], **int** s, **int** t)
454. {
455. **int** count;
456. **char** path[Max\_N];
457. printf("The shortest path from V%d(%c) to V%d(%c):\n", s, adjm->vertex[s], t, adjm->vertex[t]);
458. **if** (dis[t] == (**int**)Inf)
459. printf("No Path\n");
460. **else**
461. {
462. count = 0;
463. path[count] = adjm->vertex[t];
464. **for** (**int** j = t; j != s; )
465. {
466. path[++count] = adjm->vertex[pre[j]];
467. j = pre[j];
468. }
469. **for** (**int** j = count; j >= 0; --j)
470. cout << path[j] << " ";
471. printf("\nDistance:%d\n", dis[t]);
472. }
473. }
475. **int** main(**void**)
476. {
477. AdjMatrix\* adjm = NULL;
478. ifstream Input;
479. //Floyd-Warshall及其相关算法
480. EdgeData e[Max\_N][Max\_N];  //最短路径长度矩阵
481. **int** path[Max\_N][Max\_N];    //最短路径矩阵
482. //Dijkstra及其相关算法
483. MinHeap heap;
484. **int** start;                 //起始顶点
485. **int** target;                //目标顶点
486. **bool** S[Max\_N];             //标记集合S
487. **int** pre[Max\_N];            //记录最短路径（保存直接前导结点）
488. **int** dis[Max\_N];            //记录路径长度
489. **char** ch;
490. cout << "--------------------Read File--------------------" << endl;
491. Input.open(InputDir);
492. **if** (!Input)
493. {
494. cout << "File open failed!" << endl;
495. system("pause");
496. **return** 0;
497. }
498. streambuf\* stream\_buffer\_cin = cin.rdbuf();
499. streambuf\* stream\_buffer\_file = Input.rdbuf();
500. cin.rdbuf(stream\_buffer\_file);
501. adjm = CreateAdjM();
502. Input.close();
503. cin.rdbuf(stream\_buffer\_cin);
504. system("pause");
505. **do** {
506. system("cls");
507. cout << "--------------------Shortest Path--------------------" << endl;
508. cout << "[1]Dijkstra(single source)" << endl;
509. cout << "[2]Floyd-Warshall(all)" << endl;
510. cout << "[3]Single Target" << endl;
511. cout << "[4]Single Vertex Pair" << endl;
512. cout << "[5]Reachable matrix" << endl;
513. cout << "Input choice(other characters to quit):";
514. cin >> ch;
515. **switch** (ch)
516. {
517. **case** '1': {
518. cout << "Enter the starting vertex:";
519. cin >> start;
520. **if** (start >= adjm->n || start < 0)
521. cout << "Illegal input!" << endl;
522. **else**
523. {
524. Dijkstra(adjm, heap, start, S, pre, dis);
525. PrintResult\_D(adjm, start, S, pre, dis);
526. }
527. }
528. **break**;
529. **case** '2': {
530. Floyd\_Warshall(adjm, e, path);
531. PrintResult\_F(adjm, e, path);
532. }
533. **break**;
534. **case** '3': {
535. cout << "Enter the target vertex:";
536. cin >> target;
537. **if** (target >= adjm->n || target < 0)
538. cout << "Illegal input!" << endl;
539. **else**
540. {
541. Single\_Target(adjm, heap, S, pre, dis, target);
542. PrintResult\_ST(adjm, S, pre, dis, target);
543. }
544. }
545. **break**;
546. **case** '4': {
547. cout << "Enter the vertex pair:";
548. cin >> start >> target;
549. **if** (start >= adjm->n || start < 0 || target >= adjm->n || target < 0)
550. cout << "Illegal input!" << endl;
551. **else** **if** (start == target)
552. cout << "The two vertices should be different!" << endl;
553. **else**
554. {
555. Single\_Vertex\_Pair(adjm, heap, S, pre, dis, start, target);
556. PrintResult\_SVP(adjm, S, pre, dis, start, target);
557. Single\_Vertex\_Pair(adjm, heap, S, pre, dis, target, start);
558. PrintResult\_SVP(adjm, S, pre, dis, target, start);
559. }
560. }
561. **break**;
562. **case** '5': Reachable\_Matrix(adjm, e);
563. **break**;
564. **default**:
565. cout << "Program terminated." << endl;
566. }
567. system("pause");
568. } **while** (ch >= '1' && ch <= '5');
569. **return** 0;
570. }