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**

**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．逻辑设计

（1）主程序流程图

（2）各程序模块间调用关系：

**四、测试结果**

**五、经验体会与不足**

**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
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. **break**;
170. S[min] = **true**;
171. **for** (**int** i = 0; i < adjm->n; ++i)
172. {
173. **if** (!S[i])
174. {
175. **if** ((dis[min] + adjm->edge[min][i]) < dis[i])
176. {
177. pre[i] = min;
178. dis[i] = dis[min] + adjm->edge[min][i];  //更新距离
179. temp.num = i;
180. temp.weight = dis[i];
181. HeapInsert(heap, temp);
182. }
183. }
184. }
185. }
186. }
188. //处理并屏幕显示最短路径中各点（递归）
189. **void** PrintPath(AdjMatrix\* adjm, **int** path[][Max\_N], **int** i, **int** j)
190. {
191. **int** k;
192. k = path[i][j];
193. **if** (k != -1)
194. {
195. PrintPath(adjm, path, i, k);
196. cout << adjm->vertex[k] << " ";
197. PrintPath(adjm, path, k, j);
198. }
199. }
201. //Floyd-Warshall算法
202. **void** Floyd\_Warshall(AdjMatrix\* adjm, EdgeData e[][Max\_N], **int** path[][Max\_N])
203. {
204. //设定最短路径矩阵初值
205. **for** (**int** i = 0; i < adjm->n; ++i)
206. {
207. **for** (**int** j = 0; j < adjm->n; ++j)
208. {
209. **if** (i == j)
210. e[i][j] = 0;
211. **else**
212. e[i][j] = adjm->edge[i][j];
213. path[i][j] = -1;
214. }
215. }
216. **for** (**int** k = 0; k < adjm->n; ++k)
217. {
218. **for** (**int** i = 0; i < adjm->n; ++i)
219. {
220. **for** (**int** j = 0; j < adjm->n; ++j)
221. {
222. **if** (e[i][k] + e[k][j] < e[i][j])
223. {
224. e[i][j] = e[i][k] + e[k][j];
225. path[i][j] = k;
226. }
227. }
228. }
229. }
230. }
232. //单目标最短路径（Dijkstra算法修改）
233. **void** Single\_Target(AdjMatrix\* adjm, MinHeap& heap, **bool** S[], **int** pre[], **int** dis[], **int** target)
234. {//主要是逆向找路径
235. NodeData temp;
236. **int** min = target;
237. InitHeap(heap);
238. **for** (**int** i = 0; i < adjm->n; ++i)
239. S[i] = **false**;  //集合S初始化
240. S[target] = **true**;
241. **for** (**int** i = 0; i < adjm->n; ++i)
242. {
243. pre[i] = target;
244. dis[i] = adjm->edge[i][target];
245. **if** (dis[i] < Inf)
246. {
247. temp.weight = dis[i];
248. temp.num = i;
249. HeapInsert(heap, temp);
250. }
251. }
252. **while** (1)
253. {
254. **if** (HeapEmpty(heap))
255. **break**;
256. HeapDeleteMin(heap, min);
257. **if** (S[min])
258. **break**;
259. S[min] = **true**;
260. **for** (**int** i = 0; i < adjm->n; ++i)
261. {
262. **if** (!S[i])
263. {
264. **if** ((dis[min] + adjm->edge[i][min]) < dis[i])
265. {
266. pre[i] = min;
267. dis[i] = dis[min] + adjm->edge[i][min];  //更新距离
268. temp.num = i;
269. temp.weight = dis[i];
270. HeapInsert(heap, temp);
271. }
272. }
273. }
274. }
275. }
277. //单顶点对最短路径（Dijkstra算法修改）
278. **void** Single\_Vertex\_Pair(AdjMatrix\* adjm, MinHeap& heap, **bool** S[], **int** pre[], **int** dis[], **int** s, **int** t)
279. {
280. NodeData temp;
281. **int** min;
282. min = s;
283. InitHeap(heap);
284. **for** (**int** i = 0; i < adjm->n; ++i)
285. S[i] = **false**;  //集合S初始化
286. S[s] = **true**;
287. **for** (**int** i = 0; i < adjm->n; ++i)
288. {
289. pre[i] = s;
290. dis[i] = adjm->edge[s][i];
291. **if** (dis[i] < Inf)
292. {
293. temp.weight = dis[i];
294. temp.num = i;
295. HeapInsert(heap, temp);
296. }
297. }
298. **while** (1)
299. {
300. **if** (HeapEmpty(heap))
301. **break**;
302. HeapDeleteMin(heap, min);
303. **if** (S[min] || min == t)
304. **break**;
305. S[min] = **true**;
306. **for** (**int** i = 0; i < adjm->n; ++i)
307. {
308. **if** (!S[i])
309. {
310. **if** ((dis[min] + adjm->edge[min][i]) < dis[i])
311. {
312. pre[i] = min;
313. dis[i] = dis[min] + adjm->edge[min][i];  //更新距离
314. temp.num = i;
315. temp.weight = dis[i];
316. HeapInsert(heap, temp);
317. }
318. }
319. }
320. }
321. }
323. //计算并显示可达矩阵（Floyd-Warshall算法修改）
324. **void** Reachable\_Matrix(AdjMatrix\* adjm, EdgeData m[][Max\_N])
325. {
326. //设定可达矩阵初值
327. **for** (**int** i = 0; i < adjm->n; ++i)
328. {
329. **for** (**int** j = 0; j < adjm->n; ++j)
330. {
331. **if** (i == j)
332. m[i][j] = 1;
333. **else**
334. m[i][j] = (adjm->edge[i][j] < Inf) ? 1 : 0;
335. }
336. }
337. **for** (**int** k = 0; k < adjm->n; ++k)
338. {
339. **for** (**int** i = 0; i < adjm->n; ++i)
340. {
341. **for** (**int** j = 0; j < adjm->n; ++j)
342. m[i][j] = (m[i][j] || (m[i][k] && m[k][j]));
343. }
344. }
345. //屏幕打印可达矩阵
346. cout << "--------------------Reachable Matrix--------------------" << endl;
347. **for** (**int** i = 0; i < adjm->n; ++i)
348. {
349. **for** (**int** j = 0; j < adjm->n; ++j)
350. cout << m[i][j] << " ";
351. cout << endl;
352. }
353. }
355. //屏幕显示Dijkstra算法结果函数
356. **void** PrintResult\_D(AdjMatrix\* adjm, **int** start, **bool** S[], **int** pre[], **int** dis[])
357. {
358. **char** path[Max\_N];
359. **int** count;
360. cout << "--------------------Result--------------------" << endl;
361. **for** (**int** i = 0; i < adjm->n; ++i)
362. {
363. **if** (i != start)
364. {
365. printf("The shortest path from V%d(%c) to V%d(%c):\n", start, adjm->vertex[start], i, adjm->vertex[i]);
366. **if** (dis[i] == (**int**)Inf)
367. printf("No Path\n");
368. **else**
369. {
370. count = 0;
371. path[count] = adjm->vertex[i];
372. **for** (**int** j = i; j != start; )
373. {
374. path[++count] = adjm->vertex[pre[j]];
375. j = pre[j];
376. }
377. **for** (**int** j = count; j >= 0; --j)
378. cout << path[j] << " ";
379. printf("\nDistance:%d\n", dis[i]);
380. }
381. }
382. }
383. }
385. //屏幕显示Floyd-Warshall算法结果函数
386. **void** PrintResult\_F(AdjMatrix\* adjm, EdgeData e[][Max\_N], **int** path[][Max\_N])
387. {
388. cout << "--------------------Result--------------------" << endl;
389. cout << "Begin\tEnd\tDistance\tPath" << endl;
390. **for** (**int** i = 0; i < adjm->n; ++i)
391. {
392. **for** (**int** j = 0; j < adjm->n; ++j)
393. {
394. cout << "V" << i << ":" << adjm->vertex[i] << "\t";
395. cout << "V" << j << ":" << adjm->vertex[j] << "\t";
396. **if** (e[i][j] < (**int**)Inf)
397. {
398. cout << e[i][j] << "\t\t";
399. cout << adjm->vertex[i] << " ";
400. PrintPath(adjm, path, i, j);
401. cout << adjm->vertex[j] << endl;
402. }
403. **else**
404. {
405. cout << "inf" << "\t\t";
406. cout << "NULL" << endl;
407. }
408. }
409. }
410. }
412. //屏幕显示单目标最短路径结果函数
413. **void** PrintResult\_ST(AdjMatrix\* adjm, **bool** S[], **int** pre[], **int** dis[], **int** target)
414. {
415. cout << "--------------------Result--------------------" << endl;
416. **for** (**int** i = 0; i < adjm->n; ++i)
417. {
418. **if** (i != target)
419. {
420. printf("The shortest path from V%d(%c) to V%d(%c):\n", i, adjm->vertex[i], target, adjm->vertex[target]);
421. **if** (dis[i] == (**int**)Inf)
422. printf("No Path\n");
423. **else**
424. {
425. cout << adjm->vertex[i] << " ";
426. **for** (**int** j = i; j != target; )
427. {
428. cout << adjm->vertex[pre[j]] << " ";
429. j = pre[j];
430. }
431. printf("\nDistance:%d\n", dis[i]);
432. }
433. }
434. }
435. }
437. //屏幕显示单顶点对最短路径结果函数
438. **void** PrintResult\_SVP(AdjMatrix\* adjm, **bool** S[], **int** pre[], **int** dis[], **int** s, **int** t)
439. {
440. **int** count;
441. **char** path[Max\_N];
442. printf("The shortest path from V%d(%c) to V%d(%c):\n", s, adjm->vertex[s], t, adjm->vertex[t]);
443. **if** (dis[t] == (**int**)Inf)
444. printf("No Path\n");
445. **else**
446. {
447. count = 0;
448. path[count] = adjm->vertex[t];
449. **for** (**int** j = t; j != s; )
450. {
451. path[++count] = adjm->vertex[pre[j]];
452. j = pre[j];
453. }
454. **for** (**int** j = count; j >= 0; --j)
455. cout << path[j] << " ";
456. printf("\nDistance:%d\n", dis[t]);
457. }
458. }
460. **int** main(**void**)
461. {
462. AdjMatrix\* adjm = NULL;
463. ifstream Input;
464. //Floyd-Warshall及其相关算法
465. EdgeData e[Max\_N][Max\_N];  //最短路径长度矩阵
466. **int** path[Max\_N][Max\_N];    //最短路径矩阵
467. //Dijkstra及其相关算法
468. MinHeap heap;
469. **int** start;                 //起始顶点
470. **int** target;                //目标顶点
471. **bool** S[Max\_N];             //标记集合S
472. **int** pre[Max\_N];            //记录最短路径（保存直接前导结点）
473. **int** dis[Max\_N];            //记录路径长度
474. **char** ch;
475. cout << "--------------------Read File--------------------" << endl;
476. Input.open(InputDir);
477. **if** (!Input)
478. {
479. cout << "File open failed!" << endl;
480. system("pause");
481. **return** 0;
482. }
483. streambuf\* stream\_buffer\_cin = cin.rdbuf();
484. streambuf\* stream\_buffer\_file = Input.rdbuf();
485. cin.rdbuf(stream\_buffer\_file);
486. adjm = CreateAdjM();
487. Input.close();
488. cin.rdbuf(stream\_buffer\_cin);
489. system("pause");
490. **do** {
491. system("cls");
492. cout << "--------------------Shortest Path--------------------" << endl;
493. cout << "[1]Dijkstra(single source)" << endl;
494. cout << "[2]Floyd-Warshall(all)" << endl;
495. cout << "[3]Single Target" << endl;
496. cout << "[4]Single Vertex Pair" << endl;
497. cout << "[5]Reachable matrix" << endl;
498. cout << "Input choice(other characters to quit):";
499. cin >> ch;
500. **switch** (ch)
501. {
502. **case** '1': {
503. cout << "Enter the starting vertex:";
504. cin >> start;
505. **if** (start >= adjm->n || start < 0)
506. cout << "Illegal input!" << endl;
507. **else**
508. {
509. Dijkstra(adjm, heap, start, S, pre, dis);
510. PrintResult\_D(adjm, start, S, pre, dis);
511. }
512. }
513. **break**;
514. **case** '2': {
515. Floyd\_Warshall(adjm, e, path);
516. PrintResult\_F(adjm, e, path);
517. }
518. **break**;
519. **case** '3': {
520. cout << "Enter the target vertex:";
521. cin >> target;
522. **if** (target >= adjm->n || target < 0)
523. cout << "Illegal input!" << endl;
524. **else**
525. {
526. Single\_Target(adjm, heap, S, pre, dis, target);
527. PrintResult\_ST(adjm, S, pre, dis, target);
528. }
529. }
530. **break**;
531. **case** '4': {
532. cout << "Enter the vertex pair:";
533. cin >> start >> target;
534. **if** (start >= adjm->n || start < 0 || target >= adjm->n || target < 0)
535. cout << "Illegal input!" << endl;
536. **else** **if** (start == target)
537. cout << "The two vertices should be different!" << endl;
538. **else**
539. {
540. Single\_Vertex\_Pair(adjm, heap, S, pre, dis, start, target);
541. PrintResult\_SVP(adjm, S, pre, dis, start, target);
542. Single\_Vertex\_Pair(adjm, heap, S, pre, dis, target, start);
543. PrintResult\_SVP(adjm, S, pre, dis, target, start);
544. }
545. }
546. **break**;
547. **case** '5': Reachable\_Matrix(adjm, e);
548. **break**;
549. **default**:
550. cout << "Program terminated." << endl;
551. }
552. system("pause");
553. } **while** (ch >= '1' && ch <= '5');
554. **return** 0;
555. }