

《数据结构》上机实验报告

第 7 次上机

学号： 201611130126

姓名： 袁宇昊

学院： 信息科学学院

专业： 计算机科学

教师： 郑新

日期： 2018.11.30

1. 实验要求

1.上机之前应做好充分准备，认真思考所需的上机题目，提高上机效率。

2.独立上机输入和调试自己所编的程序，切忌抄袭、拷贝他人程序。

3.上机结束后，整理出实验报告。书写报告时，重点放在实验的方法、思路以及总结反思上，以达到巩固课堂学习、提高动手能力的目的。

1. 实验内容

一．问题描述

设计、实现一个全国大城市间的交通咨询程序，为旅客提供三种最优决策方案：（1）飞行时间最短（2）总用时最短（3）费用最小（4）中转次数最少。

二、实验要求

（1）选取合适的数据结构存储带权路线图

（2）实现单源最短路径算法

三、数据

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 机 号 | 出 发 地 | 到 达 地 | 出发时间 | 到达时间 | 费 用 |
| 6320 | 北京  上海 | 上海  北京 | 16：20  18：00 | 17：25  19：05 | 680元 |
| 2104 | 北京  乌鲁木齐 | 乌鲁木齐  北京 | 8：00  10：45 | 9：55  11：40 | 1150元 |
| 201 | 北京  西安 | 西安  北京 | 15：25  12：35 | 17：00  14：15 | 930元 |
| 2323 | 西安  广州 | 广州  西安 | 7：15  10：15 | 9：35  11：35 | 1320元 |
| 173 | 拉萨  昆明 | 昆明  拉萨 | 10：20  12：35 | 11：45  14：00 | 830元 |
| 3304 | 拉萨  武汉 | 武汉  拉萨 | 14：15  16：25 | 15：45  17：55 | 890元 |
| 82 | 乌鲁木齐  昆明 | 昆明  乌鲁木齐 | 9：30  13：05 | 12：15  15：50 | 1480元 |
| 4723 | 武汉  广州 | 广州  武汉 | 7：05  11：25 | 8：45  13 ：05 | 810元 |

1. 实验步骤（写出问题分析或者算法思路）
2. 因为问题涉及到的城市不超过10个，规模较小，所以用写法方便的二维矩阵来顺序存储航班信息。

使用了Edge类来表示航班，它存有4个航班信息：机号、起飞时间、到达时间和费用。

时间是用一个Time类来表示，Time类有时、分。这两个类都各自有自己的一些构造函数和其他功能函数。比如Time里类的mi(Time a)函数就是计算两个时间点的分钟差。

最后使用了MGraph类来表示整个图，它包括了一个二维Edge矩阵，一个城市名字数组，以及构造函数、添加边的函数、添加新城市名的函数、打印整个图的信息函数、求最短路函数等等。

1. 读入数据：将上述航班表粘贴到一个FlightData.txt文件中，再使用从文件读入数据的办法读入数据。其中有针对航班信息的字符串处理。在读入航班数据的同时，使用添加边的函数建立出一个完整的图。见程序302行~302行。
2. 建立图后，分别使用了三个显示图信息的函数来检查是否建图成功。见程序307行~309行。
3. 然后使用交互的方式来查询不同类型的最短路。首先输入要查询的起点城市和终点城市，再输入查询的类型（用数字表示），最后调用MGraph类里的Dijkstra（）函数就可以查询起点城市到终点城市在当前type下的最短路了。见程序315行到323行。
4. 程序清单（源程序代码等）

main.cpp

1. #include<iostream>
2. #include<string>
3. #include<string.h>
4. #include<iomanip>
5. #include<fstream>
6. **const** **int** MaxCityNumber=50;
7. **using** **namespace** std;
8. **class** Time
9. {
10. **public**:
11. **int** hour,minute;
12. Time(**int** h,**int** m):hour(h),minute(m){}
13. Time(**int** m)
14. {
15. hour=m/60;
16. minute=m%60;
17. }
18. **void** show()
19. {
20. **if**(hour!=0 || minute!=0)
21. {
22. cout<<setw(5)<<"     ";
23. cout<<setw(2)<<setfill('0')<<hour<<":"<<setw(2)<<minute;
24. }
25. **else**
26. cout<<setfill(' ')<<setw(10)<<"-";
27. }
28. **int** Tminute()
29. {
30. **return** hour\*60-minute;
31. }
32. **int** mi(Time a)
33. {
34. **int** mi=hour\*60+minute;
35. **int** mi2=a.hour\*60+a.minute;
36. **if**(mi2<=mi)
37. mi2+=24\*60;
38. **return** mi2-mi;
39. }
40. //    friend ostream& operator <<(ostream& os,const Time& a );
41. };
43. **class** Edge
44. {
45. **public**:
46. **int** Number;//机号
47. Time Start,End; //时间
48. **int** Cost;//费用
49. Edge(**int** number=0,**int** sh=0,**int** sm=0,**int** eh=0,**int** em=0,**int** cost=0):
50. Number(number),Start(sh,sm),End(eh,em),Cost(cost){}
51. **int** FlightTime(**void**)
52. {
53. **return** Start.mi(End);
54. }
55. };
56. **void** ShowPath(string CityName[],Edge Flight[][MaxCityNumber],**int** path[],**int** ta)
57. {
58. **if**(path[ta]!=-1)
59. {
60. ShowPath(CityName,Flight,path,path[ta]);
61. }
62. **if**(path[ta]!=-1)
63. cout<<"-"<<"("<<Flight[ta][path[ta]].Number<<")->"<<CityName[ta];
64. **else**
65. cout<<CityName[ta];
66. }
67. **class** MGraph
68. {
69. **public**:
70. **int** CityTotalNumber;
71. string CityName[MaxCityNumber];
72. Edge Flight[MaxCityNumber][MaxCityNumber];
74. //type=1:飞行时间最短,type=2:总用时最短,type=3:费用最小,type=4:中转次数最少
75. **void** Dijkstra(string st,string ed,**int** type)
76. {
77. //int parent[MaxCityNumber];
78. **int** path[MaxCityNumber];
79. **int** dis[MaxCityNumber];
80. **int** vis[MaxCityNumber];
81. **int** s=FindCityNumber(st);
82. **int** e=FindCityNumber(ed);
83. **const** **int** Maxn=9999999;
84. **for**(**int** i=0;i<CityTotalNumber;i++)
85. dis[i]=Maxn,vis[i]=0,path[i]=-1;
86. dis[s]=0;
87. **for**(**int** i=1;i<=CityTotalNumber;i++)
88. {
89. **int** minn=Maxn,minth;
90. **for**(**int** j=0;j<CityTotalNumber;j++)
91. {
92. **if**(vis[j]==0 && dis[j]<minn)
93. {
94. minn=dis[j];
95. minth=j;
96. }
97. }
98. vis[minth]=1;
99. **if**(minth==e)
100. **break**;
101. //cout<<"min="<<minn<<" "<<minth<<endl;
102. **for**(**int** j=0;j<CityTotalNumber;j++)
103. {
104. **if**(vis[j]==0 && Flight[minth][j].Cost!=0)
105. {
106. **switch**(type)//type=1:飞行时间最短,type=2:总用时最短,type=3:费用最小,type=4:中转次数最少
107. {
108. **case** 1:
109. dis[j]=min(dis[j],dis[minth]+Flight[minth][j].FlightTime());
110. **if**(dis[j]==dis[minth]+Flight[minth][j].FlightTime())
111. path[j]=minth;
112. **break**;
113. **case** 2:
114. **if**(path[minth]!=-1)
115. {
116. **if**(dis[j]>dis[minth]+Flight[minth][j].End.mi(Flight[path[i]][minth].End));
117. {
118. dis[j]=dis[minth]+Flight[minth][j].End.mi(Flight[path[i]][minth].End);
119. path[j]=minth;
120. }
121. }
122. **else**
123. {
124. **if**(dis[j]>Flight[minth][j].FlightTime())
125. {
126. dis[j]=Flight[minth][j].FlightTime();
127. path[j]=minth;
128. }
129. }
130. **break**;
131. **case** 3:
132. **if**(dis[j]>dis[minth]+Flight[minth][j].Cost)
133. {
134. dis[j]=dis[minth]+Flight[minth][j].Cost;
135. path[j]=minth;
136. }
137. **break**;
138. **case** 4:
139. **if**(dis[j]>1+dis[minth])
140. {
141. dis[j]=1+dis[minth];
142. path[j]=minth;
143. }
144. }
145. }
146. }
147. }
148. **if**(dis[e]<Maxn)
149. {
150. cout<<st<<"到"<<ed<<"的路径";
151. cout<<"找到了\n路径是：";
152. ShowPath(CityName,Flight,path,e);
153. cout<<"\n总消耗";
154. **switch**(type)//type=1:飞行时间最短,type=2:总用时最短,type=3:费用最小,type=4:中转次数最少
155. {
156. **case** 1:
157. cout<<"时间是";
158. Time(dis[e]).show();
159. **break**;
160. **case** 2:
161. cout<<"时间是";
162. Time(dis[e]).show();
163. **break**;
164. **case** 3:
165. cout<<"票价是";
166. cout<<dis[e]<<"元";
167. **break**;
168. **case** 4:
169. cout<<"转机次数是";
170. cout<<dis[e]<<"次";
171. **break**;
172. }
173. cout<<endl;
174. }
175. **else**
176. {
177. cout<<st<<"到"<<ed<<"的路径";
178. cout<<"找不到\n";
179. }
180. }
181. MGraph()
182. {
183. CityTotalNumber=0;
184. }
185. **int** FindCityNumber(string cityname)
186. {
187. **for**(**int** i=0;i<CityTotalNumber;i++)
188. **if**(CityName[i]==cityname)
189. **return** i;
190. **return** -1;
191. }
192. **bool** AddCity(string cityname)
193. {
194. **if**(FindCityNumber(cityname)==-1)
195. {
196. CityName[CityTotalNumber]=cityname;
197. CityTotalNumber++;
198. **return** **true**;
199. }
200. **return** **false**;
201. }
202. **void** AddEdge(**int** number,string st,string ed,**int** sh,**int** sm,**int** eh,**int** em,**int** cost)
203. {
204. AddCity(st),AddCity(ed);
205. **int** s=FindCityNumber(st);
206. **int** e=FindCityNumber(ed);
207. Flight[s][e]=Edge(number,sh,sm,eh,em,cost);
208. }
209. **void** showCost()
210. {
211. **for**(**int** i=0;i<CityTotalNumber;i++)
212. {
213. **if**(i==0)
214. {
215. cout<<setw(10)<<setfill(' ')<<"花费";
216. **for**(**int** j=0;j<CityTotalNumber;j++)
217. cout<<setw(10)<<CityName[j];
218. cout<<endl;
219. }
220. cout<<setw(10)<<setfill(' ')<<CityName[i];
221. **for**(**int** j=0;j<CityTotalNumber;j++)
222. {
223. **if**(Flight[i][j].Cost!=0)
224. cout<<setw(10)<<Flight[i][j].Cost;
225. **else**
226. cout<<setw(10)<<"-";
227. }
228. cout<<endl;
229. }
230. cout<<endl;
231. }
232. **void** showStartTime()
233. {
234. **for**(**int** i=0;i<CityTotalNumber;i++)
235. {
236. **if**(i==0)
237. {
238. cout<<setw(10)<<setfill(' ')<<"出发时间";
239. **for**(**int** j=0;j<CityTotalNumber;j++)
240. cout<<setw(10)<<CityName[j];
241. cout<<endl;
242. }
243. cout<<setw(10)<<setfill(' ')<<CityName[i];
244. **for**(**int** j=0;j<CityTotalNumber;j++)
245. {
246. Flight[i][j].Start.show();
247. }
248. cout<<endl;
249. }
250. cout<<endl;
251. }
252. **void** showEndTime()
253. {
254. **for**(**int** i=0;i<CityTotalNumber;i++)
255. {
256. **if**(i==0)
257. {
258. cout<<setw(10)<<setfill(' ')<<"到达时间";
259. **for**(**int** j=0;j<CityTotalNumber;j++)
260. cout<<setw(10)<<CityName[j];
261. cout<<endl;
262. }
263. cout<<setw(10)<<setfill(' ')<<CityName[i];
264. **for**(**int** j=0;j<CityTotalNumber;j++)
265. {
266. Flight[i][j].End.show();
267. }
268. cout<<endl;
269. }
270. cout<<endl;
271. }
272. };
273. **void** getTime(ifstream &input,**int** &h,**int** &m)
274. {
275. input>>h;
276. input.get();
277. input.get();
278. input>>m;
279. }
280. **int** main ()
281. {
282. ifstream input("FlightData.txt",ios::in);
283. string head;
284. getline(input,head);
285. MGraph g;
286. //g.AddEdge(6320,"北京","上海",16,20,17,25,680);
287. //    cin.getline()
288. **int** number;//机号
289. string st,ed,st2,ed2;//地点
290. **int** sh,sm,eh,em,sh2,sm2,eh2,em2;//时间
291. **int** cost;//费用
292. **while**(input>>number)
293. {
294. cout<<"读取了"<<number<<"航班"<<endl;
295. input>>st>>ed>>st2>>ed2;
296. getTime(input,sh,sm);
297. getTime(input,sh2,sm2);
298. //cout<<sh<<":"<<sm<<endl;
299. getTime(input,eh,em);
300. getTime(input,eh2,em2);
301. input>>cost;
302. g.AddEdge(number,st,ed,sh,sm,eh,em,cost);
303. g.AddEdge(number,st2,ed2,sh2,sm2,eh2,em2,cost);
304. input.get();
305. input.get();
306. }
307. g.showCost();
308. g.showStartTime();
309. g.showEndTime();
310. **char** flag='Y';
311. string St,Ed;
312. **int** Type;
313. **while**(flag=='Y')
314. {
315. cout<<"请输入起点城市名\n";
316. cin>>St;
317. cout<<"请输入起点城市名\n";
318. cin>>Ed;
319. cout<<"请输入查询的类型：type（type=1:飞行时间最短,type=2:总用时最短,type=3:费用最小,type=4:中转次数最少)\n";
320. cin>>Type;
321. g.Dijkstra(St,Ed,Type);
322. cout<<"还要继续查询吗？（Y/N）\n";
323. cin>>flag;
324. }
326. **return** 0;
327. }

数据文件FlightData.txt 直接粘贴航班表格到txt文件里

机 号 出 发 地 到 达 地 出发时间 到达时间 费 用

6320 北京

上海 上海

北京 16：20

18：00 17：25

19：05 680元

2104 北京

乌鲁木齐 乌鲁木齐

北京 8：00

10：45 9：55

11：40 1150元

201 北京

西安 西安

北京 15：25

12：35 17：00

14：15 930元

2323 西安

广州 广州

西安 7：15

10：15 9：35

11：35 1320元

173 拉萨

昆明 昆明

拉萨 10：20

12：35 11：45

14：00 830元

3304 拉萨

武汉 武汉

拉萨 14：15

16：25 15：45

17：55 890元

82 乌鲁木齐

昆明 昆明

乌鲁木齐 9：30

13：05 12：15

15：50 1480元

4723 武汉

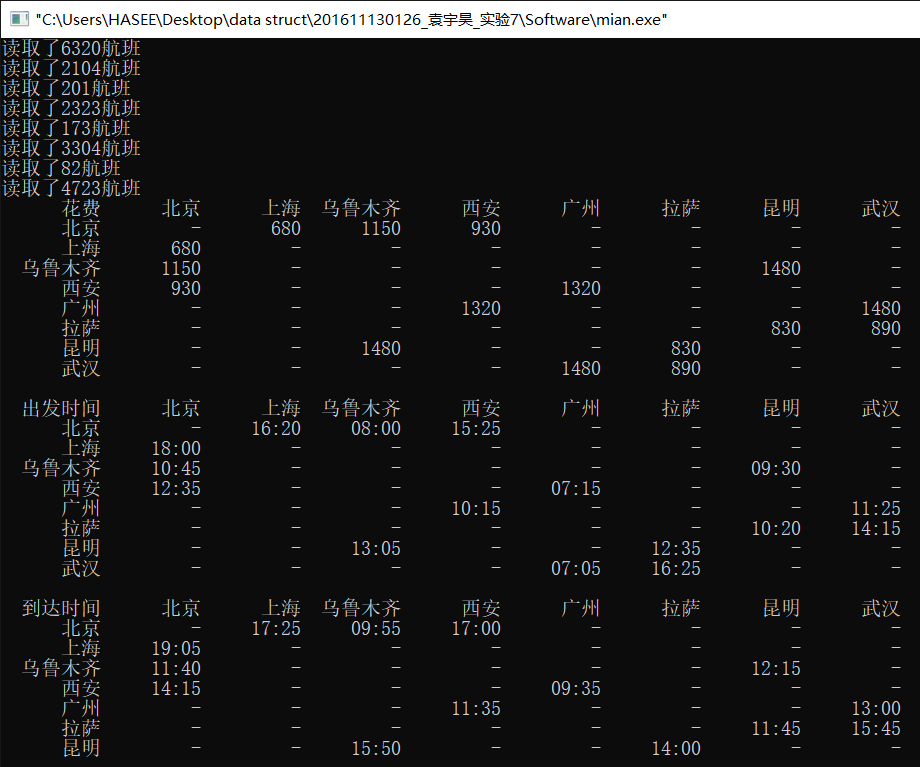
广州 广州

武汉 7：05

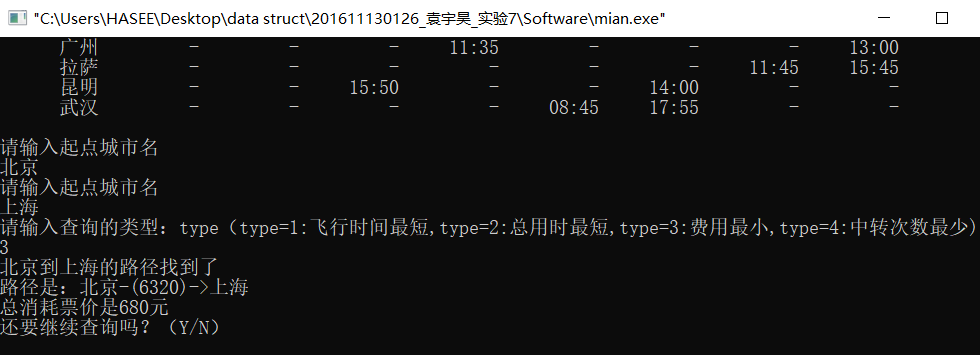
11：25 8：45

13 ：05 810元

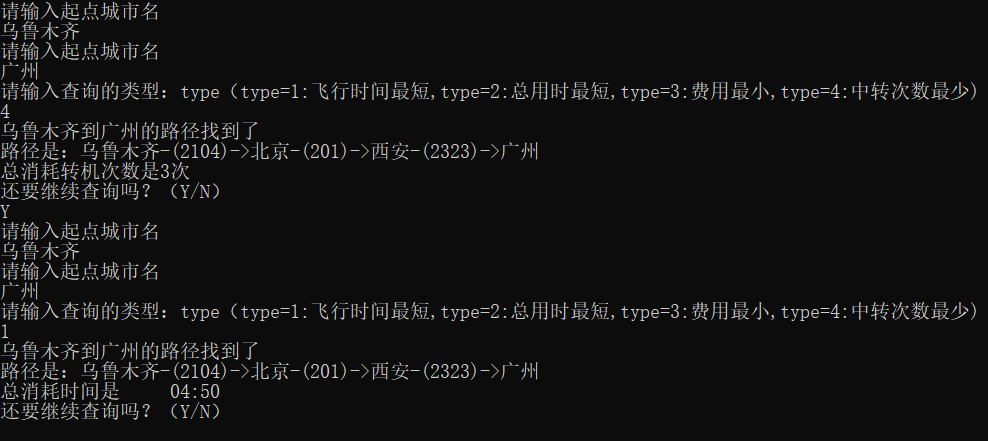
1. 运行结果（程序运行时的结果说明或运行截图等）



这是读取数据后的结果，可以清楚的看到每对城市之间的航班信息（包括花费矩阵、出发时间矩阵、到达时间矩阵）。还可以看每对城市之间的航班编号。



这是一次完整的查询，查询的是北京到上海在费用最小的情况下的最短路。



这是两次连续的查询，都是查询乌鲁木齐到广州的航班信息，不同的是第一次查询的是转机次数最少，第二次是飞行时间最短。每次查询都给出了飞行路径上的航班编号和对应type的最小消耗。

六、总结（实验中遇到的问题、取得的经验、感想等）

1、使用Dijkstra最短路算法。与ppt上的Dijkstra算法不同，这一次要计算的是4种类型下的最短路。用dis[]表示Dijkstra算法中的距离数组。其实基本的算法思路不变，改变的只是对dis[]数组的更新上。以往是在找到一个当前距离最短的结点连起来后，按照相连路径距离更新与之相连的点的距离dis[]数组。 现在就是要根据不同的查询type来用不同的数据更新dis[]数组。例如求花费最小，就要用航班花费去跟新dis[]数组，求飞行时间最小，就要用航班飞行时间去更新dis[]数组。换句话说，这里的dis[i]数组不再单纯地指原点到i结点的距离，还可以指原点到i结点的票价花费、时间花费、中转次数等等。要按照type来决定dis[]的意义。

2、每一次调用Dijkstra算法的复杂度是n^2的，n是城市的数目。主要是因为在查找最近点的时候需要遍历每个城市，而更新点的时候最多遍历n个城市（因为一个选好的城市是最多与n-1个城市相连），这两个过程平均要进行n/2次才能找到终点。所以时间复杂度是O(n^2)的。