**数据结构实验报告—图的基本操作实现**

设计者姓名：张帆

设计者班级：2班

设计者学号：20192131077

上机环境：DEV-C++

设计日期：2020-12-21

1. **实验题目**

1） 选择邻接表作为无向图的存储结构，设计一个程序实现图的基本操作（包括

输出、广度遍历、深度遍历）

1. 选择邻接矩阵作为有向图图的存储结构，设计一个程序实现求最短路径的算法
2. 选择邻接表作为有向图图的存储结构，设计一个程序实现图的拓扑排序算法

4） 选择邻接矩阵作为无向图图的存储结构，分别设计用 prim 求最小生成树和 用 kruskal 求最小生成树的算法

程序的菜单功能项如下：

初始化（图的邻接表存储为空）//可以不要该选项

图邻接表的建立 //针对的图为不带权图

深度优先遍历 //针对的存储结构为 2 所建的邻接表

广度优先遍历 //针对的存储结构是 2 所建的邻接表

最小生成树（prim/kruskal）//先建立带权图的邻接矩阵，然后实用 prim 和

kruskal 算法

拓扑排序 //先建立带权图的邻接表，然后用拓扑排序

单源点到其余各个顶点的最短路径 //先建立带权图的邻接矩阵，然后用最

短路径的求法

退出 //输出放在文件中

1. **实验项目目的**

使学生熟练掌握图的各种基本操作

1. **实验项目的程序结构**



**四、实验项目包含的各个文件中的函数的功能描述**

1. #include <iostream>
2. #include <list>
3. #include <cmath>
4. #include <stack>
5. #include <queue>
6. #include <vector>
7. #include <cstdio>
8. #include <cstdlib>
9. #include <cstring>
10. #include <fstream>
11. #include <algorithm>
12. **using** **namespace** std;
13. /\* run this program using the console pauser or add your own getch, system("pause") or input loop \*/
14. **static** **const** **int** MAX = 100;
15. **static** **const** **int** INFTY = (1 << 21);
16. **static** **const** **int** WHITE = 0;
17. **static** **const** **int** GRAY = 1;
18. **static** **const** **int** BLACK = 2;

21. **int** n;
23. /\*\*\*邻接表\*\*\*/
24. **bool** visit[MAX];  //标定顶点是否被访问过
25. vector<**int**> G[MAX];  //邻接表存储图
26. **int** nt[MAX], color[MAX];
27. list<**int**> out;
28. **int** indeg[MAX];
29. **bool** flag[MAX];
31. /\*\*\*邻接矩阵\*\*\*/
32. **int** M[MAX][MAX];  //邻接矩阵存储图

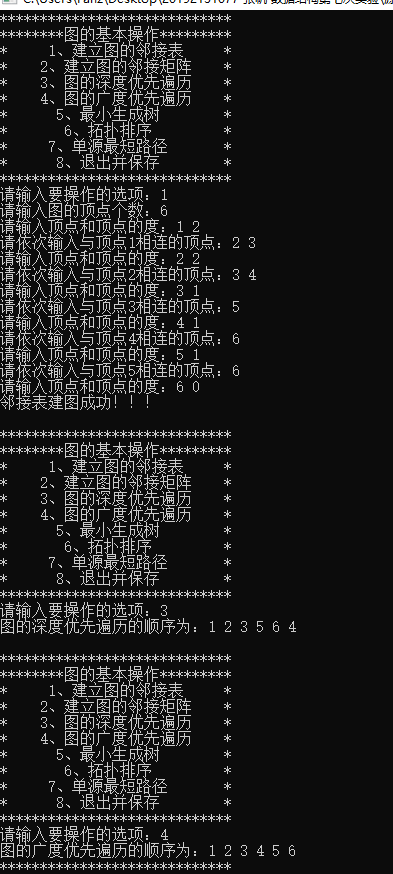
35. **int** next(**int** u)
36. {
37. **if** (nt[u] < G[u].size())
38. {
39. nt[u]++;
40. **return** G[u][nt[u] - 1];
41. }
42. **return** -1;
43. }
45. **void** dfs\_visit(**int** r)
46. {
47. **for** (**int** i = 1; i <= n; i++)
48. {
49. nt[i] = 0;
50. visit[i] = **false**;
51. }
53. stack<**int**> S;
54. S.push(r);
55. color[r] = GRAY;
57. **while**(!S.empty())  //当栈不为空的时候
58. {
59. **int** u = S.top();
60. **int** v = next(u);  //下一个与顶点u相邻的点
61. **if** (visit[u] == **false**)
62. {
63. cout << u << " ";
64. visit[u] = **true**;
65. }
66. **if** (v != -1)  //如果没有与顶点u相邻的点的情况
67. {
68. **if** (color[v] == WHITE)
69. {
70. color[v] = GRAY;
71. S.push(v);
72. }
73. }
74. **else**
75. {
76. S.pop();
77. color[u] = BLACK;
78. }
79. }
80. }
82. **void** dfs()
83. {
84. **for** (**int** i = 1; i <= n; i++)  //初始化
85. {
86. color[i] = WHITE;
87. }
89. cout << "图的深度优先遍历的顺序为：";
90. **for** (**int** u = 1; u <= n; u++)  //如果有尚未到达的顶点则深度遍历此点
91. {
92. **if** (color[u] == WHITE)  dfs\_visit(u);
93. }
94. cout << endl;
95. }
97. **void** bfs()
98. {
99. **for** (**int** i = 1; i <= n; i++)  //初始化
100. {
101. nt[i] = 0;
102. visit[i] = **false**;
103. color[i] == WHITE;
104. }
106. queue<**int**> q;
107. q.push(1);
108. visit[1] = **true**;
110. cout << "图的广度优先遍历的顺序为：";
112. **while**(!q.empty())
113. {
114. **int** u = q.front();  //取出队头的顶点
115. cout << u << " ";
116. q.pop();  //删除队头
117. **for** (**int** i = 0; i < G[u].size(); i++)
118. {
119. **if** (!visit[G[u][i]])  //如果与此点相邻的点有没被访问过的则加入队列
120. {
121. visit[G[u][i]] = **true**;
122. q.push(G[u][i]);
123. }
124. }
125. }
126. }
128. **int** prim()  //prim最小生成树
129. {
130. **int** u, minv;
131. **int** d[MAX], p[MAX];
132. **for** (**int** i = 1; i <= n; i++)
133. {
134. d[i] = INFTY;
135. p[i] = -1;
136. color[i] = WHITE;
137. }
138. d[1] = 0;
139. **while** (**true**)
140. {
141. minv = INFTY;
142. u = -1;
143. **for** (**int** i = 1; i <= n; i++)
144. {
145. **if** (minv > d[i] && color[i] != BLACK)
146. {
147. u = i;
148. minv = d[i];
149. }
150. }
151. **if** (u == -1)  **break**;
152. color[u] = BLACK;
153. **for** (**int** v = 1; v <= n; v++)
154. {
155. **if** (color[v] != BLACK && M[u][v] != INFTY)
156. {
157. **if** (d[v] > M[u][v])
158. {
159. d[v] = M[u][v];
160. p[v] = u;
161. color[v] = GRAY;
162. }
163. }
164. }
165. }
166. **int** sum = 0;
167. **for** (**int** i = 1; i <= n; i++)
168. {
169. **if** (p[i] != -1)
170. {
171. sum += M[i][p[i]];
172. }
173. }
174. **return** sum;
175. }
177. **void** bfs2(**int** s)
178. {
179. queue<**int**> q;
180. q.push(s);
182. **while** (!q.empty())
183. {
184. **int** u = q.front(); q.pop();
185. out.push\_back(u);
186. **for** (**int** i = 0; i < G[u].size(); i++)
187. {
188. **int** v = G[u][i];
189. indeg[v]--;
190. **if** (indeg[v] == 0 && flag[v] == 0)
191. {
192. flag[v] = **true**;
193. q.push(v);
194. }
195. }
196. }
197. }
199. **void** tsort()
200. {
201. **for** (**int** i = 1; i <= n; i++)
202. {
203. indeg[i] = 0;
204. flag[i] = **false**;
205. }
207. **for** (**int** u = 1; u <= n; u++)
208. {
209. **for** (**int** i = 0; i < G[u].size(); i++)
210. {
211. **int** v = G[u][i];
212. indeg[v]++;
213. }
214. }
216. **for** (**int** u = 1; u <= n; u++)
217. {
218. **if** (indeg[u] == 0 && flag[u] == 0)
219. bfs2(u);
220. }
222. cout << "拓扑排序的顺序为：";
224. **for** (list<**int**>::iterator it = out.begin(); it != out.end(); it++)
225. {
226. cout << \*it << " ";
227. }
228. cout << endl;
229. }
231. **void** dijkstra(**int** s)
232. {
233. **int** minv;
234. **int** d[MAX];
235. **for** (**int** i = 1; i <= n; i++)
236. {
237. d[i] = INFTY;
238. color[i] = WHITE;
239. }
240. d[s] = 0;
241. color[s] = GRAY;
242. **while** (1)
243. {
244. minv = INFTY;
245. **int** u = -1;
246. **for** (**int** i = 1; i <= n; i++)
247. {
248. **if** (color[i] != BLACK && minv > d[i])
249. {
250. u = i;
251. minv = d[i];
252. }
253. }
254. **if** (u == -1)  **break**;
255. color[u] = BLACK;
256. **for** (**int** v = 1; v <= n; v++)
257. {
258. **if** (color[v] != BLACK && M[u][v] != INFTY)
259. {
260. **if** (d[v] > d[u] + M[u][v])
261. {
262. d[v] = d[u] + M[u][v];
263. color[v] = GRAY;
264. }
265. }
266. }
267. }
268. **for** (**int** i = 1; i <= n; i++)
269. {
270. cout << "点" << s << "到点" << i << "的单源最短路径为：" << (d[i] == INFTY ? -1 : d[i])  << endl;
271. }
272. }
274. **void** Menu()  //菜单函数
275. {
276. **while**(1)
277. {
278. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
279. cout << "\*\*\*\*\*\*\*\*图的基本操作\*\*\*\*\*\*\*\*\*" << endl;
280. cout << "\*     1、建立图的邻接表     \*" << endl;
281. cout << "\*    2、建立图的邻接矩阵    \*" << endl;
282. cout << "\*    3、图的深度优先遍历    \*" << endl;
283. cout << "\*    4、图的广度优先遍历    \*" << endl;
284. cout << "\*      5、最小生成树        \*" << endl;
285. cout << "\*       6、拓扑排序         \*" << endl;
286. cout << "\*     7、单源最短路径       \*" << endl;
287. cout << "\*      8、退出并保存        \*" << endl;
288. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
289. cout << "请输入要操作的选项：";
290. **int** choice;
291. cin >> choice;
292. **if** (choice == 1)
293. {
294. **int** u, k, v;
295. cout << "请输入图的顶点个数：";
296. cin >> n;
297. **for** (**int** i = 0; i < n; i++)
298. {
299. cout << "请输入顶点和顶点的度：";
300. cin >> u >> k;
301. **if** (k != 0)
302. {
303. cout << "请依次输入与顶点" << u << "相连的顶点：";
304. **for** (**int** j = 0; j < k; j++)
305. {
306. cin >> v;
307. G[u].push\_back(v);
308. }
309. }
310. }
311. cout << "邻接表建图成功！！！" << endl;
312. }
313. **else** **if** (choice == 2)
314. {
315. cout << "请输入图的顶点个数：";
316. cin >> n;
317. **for** (**int** i = 1; i <= n; i++)
318. {
319. cout << "请输入点" << i << "到各顶点的距离(边不存在记为-1)：";
320. **for** (**int** j = 1; j <= n; j++)
321. {
322. **int** e;
323. cin >> e;
324. M[i][j] = (e == -1) ? INFTY : e;
325. }
326. }
327. cout << "邻接矩阵建图成功！！！" << endl;
328. }
329. **else** **if** (choice == 3)
330. {
331. dfs();
332. }
333. **else** **if** (choice == 4)
334. {
335. bfs();
336. }
337. **else** **if** (choice == 5)
338. {
339. cout << "使用prim算法求的的最小生成树为：" << prim() << endl;
340. }
341. **else** **if** (choice == 6)
342. {
343. tsort();
344. }
345. **else** **if** (choice == 7)
346. {
347. cout << "请输入图的顶点个数：";
348. cin >> n;
349. **for** (**int** i = 1; i <= n; i++)
350. {
351. **for** (**int** j = 1; j <= n; j++)
352. {
353. M[i][j] = INFTY;
354. }
355. }
356. **int** k, c, u, v;
357. **for** (**int** i = 0; i < n; i++)
358. {
359. cout << "请输入顶点和顶点的度：";
360. cin >> u >> k;
361. **for** (**int** j = 0; j < k; j++)
362. {
364. cout << "请依次输入与顶点" << u << "相连的顶点和边的权值：";
365. cin >> v >> c;
366. M[u][v] = c;
367. }
368. }
369. dijkstra(1);
370. cout << endl;
371. }
372. **else** **if** (choice == 8)
373. {
374. ofstream fop("TextFile.txt",ios::out|ios::binary|ios::trunc);  //保存
375. fop << "Graph!";
376. fop.close();
377. cout << "已存到文件TextFile.txt中！" << endl;
378. **break**;
379. }
380. cout << endl;
381. }
382. }

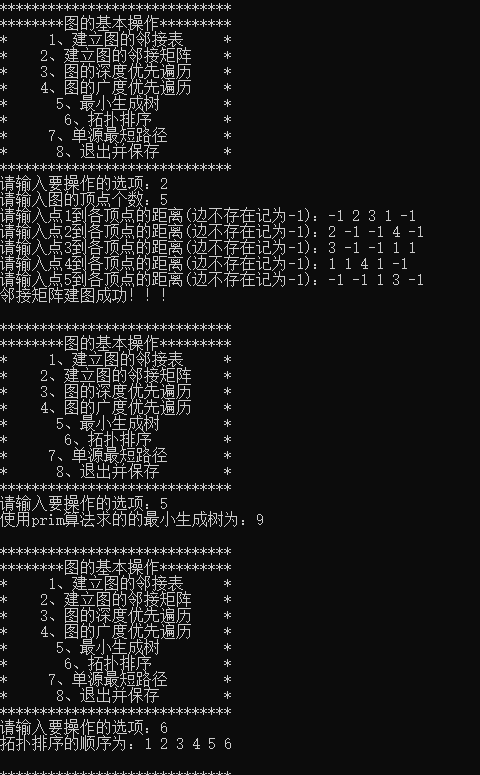
385. **int** main(**int** argc, **char**\*\* argv)
386. {
387. Menu();  //菜单函数
388. **return** 0;
389. }
391. //邻接表测试样例  6 1 2 2 3 2 2 3 4 3 1 5 4 1 6 5 1 6 6 0
392. //邻接矩阵测试样例 5 -1 2 3 1 -1 2 -1 -1 4 -1 3 -1 -1 1 1 1 4 1 -1 3 -1 -1 1 3 -1
393. //单源最短路径测试样例 5 1 3 3 3 4 1 2 2 2 2 1 2 4 4 3 3 1 3 4 1 5 1 4 4 3 1 1 1 2 4 5 3 5 2 3 1 4 3

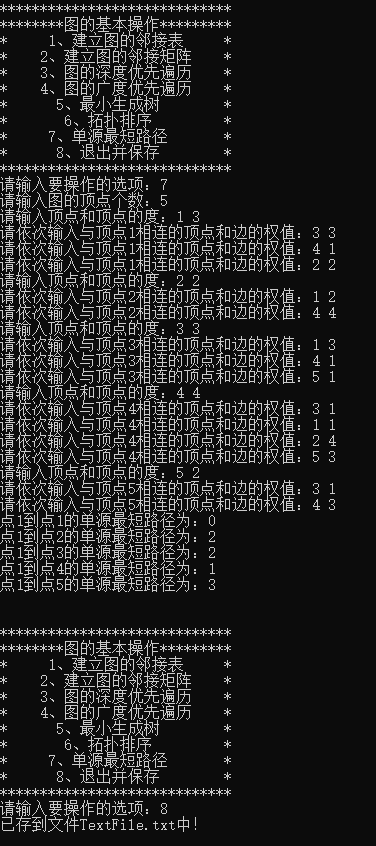
**五、算法描述或流程图**

Prim算法和Dijkstra算法非常类似，他们的伪码几乎相近，只是他们优先队列所排序的键值不同而已。Prim算法的键值为节点与集合S中顶点间的最轻边的权重，而在Dijkstra算法中，键值为由起始点到某节点的完整路径长度。Dijstra算法主要特点是从起始点开始，采用贪心算法的策略，每次遍历到始点距离最近且未访问过的顶点的邻接节点，直到扩展到终点为止。

**六、实验数据和实验结果分析**







1. **实验体会**

这次实验，让我更加理解了图的存储操作，对图的prim算法、kruskal算法和dijstra算法有了一个更深入的了解。