**南京航空航天大学**

**数据结构课程设计报告**

|  |  |
| --- | --- |
| 学生姓名 | 刘志赞 |
| 学 号 | 161730123 |
| 学 院 | 计算机科学与技术学院 |
| 专 业 | 软件工程 |
| 班 级 | 1617304 |
| 指导教师 | 高航 |

二〇一九年一月

**目 录**

[一、 购物网站信息管理 4](#_Toc16524)

[1.1题目简介 4](#_Toc24720)

[1.2数据结构 4](#_Toc28370)

[1.3算设法计思想 4](#_Toc9562)

[1.4测试数据和结果 8](#_Toc31152)

[1.5算法时间复杂度 8](#_Toc32400)

[1.6源代码 8](#_Toc27957)

[二、 公共钥匙盒 2](#_Toc16524)4

[2.1题目简介 2](#_Toc24720)5

[2.2数据结构 2](#_Toc28370)5

[2.3算设法计思想 2](#_Toc9562)5

[2.4测试数据和结果 2](#_Toc31152)5

[2.5算法时间复杂度 2](#_Toc32400)6

[2.6源代码](#_Toc27957) 26

[三、 树Json的应用 2](#_Toc16524)9

[3.1题目简介 3](#_Toc24720)1

[3.2数据结构 3](#_Toc28370)1

[3.3算设法计思想 3](#_Toc9562)1

[3.4测试数据和结果 3](#_Toc31152)2

[3.5算法时间复杂度 3](#_Toc32400)3

[3.6源代码 3](#_Toc27957)3

[四、 Huffman编码与解码 3](#_Toc16524)7

[4.1题目简介 3](#_Toc24720)7

[4.2数据结构 3](#_Toc28370)8

[4.3算设法计思想](#_Toc9562) 38

[4.4测试数据和结果](#_Toc31152) 39

[4.5算法时间复杂度](#_Toc32400) 40

[4.6源代码](#_Toc27957) 40

[五、 行车路线 4](#_Toc16524)7

[5.1题目简介 4](#_Toc24720)8

[5.2数据结构 4](#_Toc28370)8

[5.3算设法计思想 4](#_Toc9562)8

[5.4测试数据和结果 4](#_Toc31152)9

[5.5算法时间复杂度 4](#_Toc32400)9

[5.6源代码 4](#_Toc27957)9

[六、 排序算法比较 5](#_Toc16524)3

[6.1题目简介 5](#_Toc24720)3

[6.2数据结构 5](#_Toc28370)3

[6.3算设法计思想 5](#_Toc9562)4

[6.4测试数据和结果 5](#_Toc31152)5

[6.5算法时间复杂度 5](#_Toc32400)6

[6.6源代码 5](#_Toc27957)7

[七、 朋友圈问题 7](#_Toc16524)0

[7.1题目简介 7](#_Toc24720)0

[7.2数据结构 7](#_Toc28370)0

[7.3算设法计思想 7](#_Toc9562)0

[7.4测试数据和结果 7](#_Toc31152)1

[7.5算法时间复杂度 7](#_Toc32400)1

[7.6源代码 7](#_Toc27957)1

[八、 迷宫比较 7](#_Toc16524)2

[8.1题目简介 7](#_Toc24720)2

[8.2数据结构 7](#_Toc28370)3

[8.3算设法计思想 7](#_Toc9562)3

[8.4测试数据和结果 7](#_Toc31152)4

[8.5算法时间复杂度 7](#_Toc32400)5

[8.6源代码 7](#_Toc27957)5

[九、算术表达式 8](#_Toc16524)2

[9.1题目简介 8](#_Toc24720)2

[9.2数据结构 8](#_Toc28370)2

[9.3算设法计思想 8](#_Toc9562)3

[9.4测试数据和结果 8](#_Toc31152)3

[9.5算法时间复杂度 8](#_Toc32400)6

9[.6源代码 8](#_Toc27957)7

# 一、购物网站信息管理

# 1.1 题目简介

购物网站信息管理（必做）（线性表）

[问题描述]

设计一个程序，对商铺信息管理，商铺信息包括：商铺编号，商铺名，信誉度（0-5），（商品名称1，价格1，销量1），（商品名称2，价格2，销量2），（商品名称3，价格3，销量3）…。商品名称包括(毛巾，牙刷，牙膏，肥皂，洗发水，沐浴露等6种以上商品)，每个商铺具有其中事先确定若干商品及价格，由文件输入，销量初始为0。

[基本要求]

（1）建立一个单向链表存储所有商铺信息（至少30个），以编号为序，编号从1开始递增，从文件中读取数据，并能将数据存储在文件。商铺信息结点的数据结构自行设计。

（2）可以增、删商铺。增加商铺，编号自动加一，插入链表尾部；删除商铺，以编号为准，并修改后续结点的编号，保持编号连续性。可增、删商品。

（3）查询某一种商品名称，建立一个双向循环链表，结点信息是包含该商品的所有商铺编号、商铺名、信誉度、商品名称、价格、销量，以信誉度从高至低，并按销量排序，并逐一显示。

（4）购买某一商铺的商品，修改单向链表中商品的信息的销量。

（5）建立一个顺序结构，按商品名Hash分配地址，存储当前每种商品总销量并输出。

（6）任何的商铺信息变化，实现文件存储。

# 1.2 数据结构

单向链表，双向循环链表

# 1.3 算法设计思想

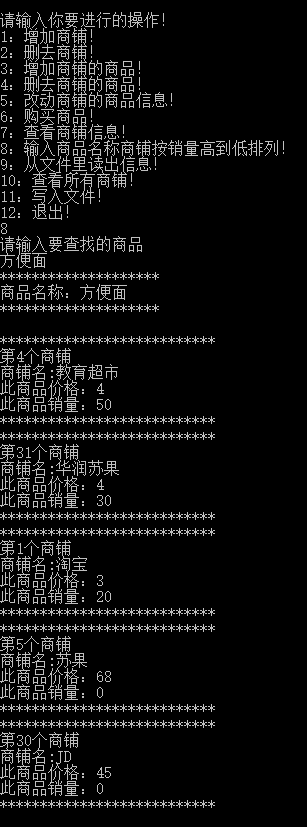
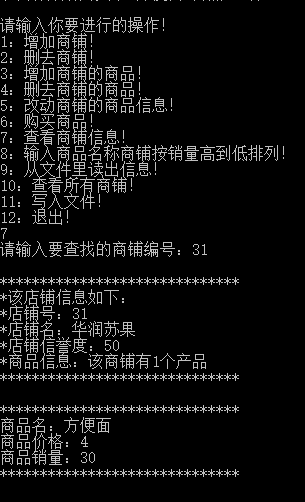
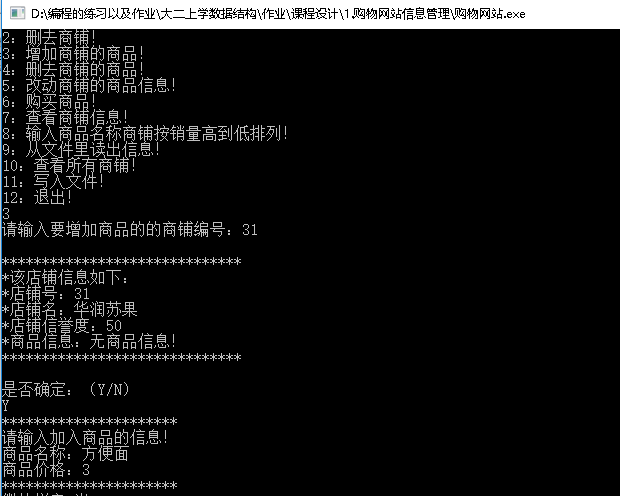
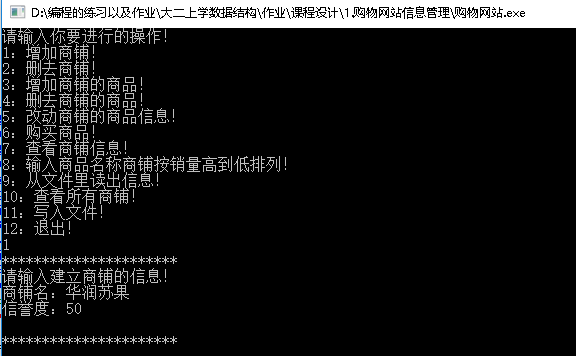
（1）建立单向链表存储商铺的信息。

（2）对链表节点进行操作增删商铺

（3）建立双向循环链表存储包含某一商品的所有商铺，按销量从高到低进行排序。在建立时，先遍历所有商铺的所有产品，若商品名称和给定商品一致，则插入到有序的双向链表中。在插入时进行销量的比较，找到位置插入

# 1.4 测试数据和结果

测试结果：



# 1.5 算法时间复杂度

（1）查找商铺的时间复杂度为o(n)

（2）增加和删除商铺的时间复杂度为o(1)

（3）查找商品的时间复杂度为o(n)

（4）增加、删除、修改和购买商品的时间复杂度为o(1)

（5）查询商品名称建立双线循环链表分两步，查询所有商品名称的时间复杂

1.6 源代码(700行左右

Message.cpp

1. #include"Message.h"
2. #define ERROR 0
3. #define OK 1
5. **bool** CreateSort(Store \*&S,string pdtName,SortMessage \*&SortM)
6. {
7. SortM = **new** SortMessage;
8. SortMessage \*SortM1= **new** SortMessage;
9. SortM1->next = SortM;
10. SortM1->last = SortM;
11. SortM->next = SortM1;
12. SortM->last = SortM1;
13. SortM->id = 0;
14. SortM->pdtName = pdtName;//定义头节点SortM和尾节点SortM1
15. Store \*now = S->next;
16. **while**(now != NULL)
17. {
18. Product \*pdt = now->pdt;
19. **while**(pdt != NULL)
20. {
21. **if**(pdt->pdtName == SortM->pdtName)
22. {
23. SortMessage \*newm = **new** SortMessage;
24. SortM->id++;
25. newm->id = now->storeId;
26. newm->last = SortM1->last;
27. newm->pdtName = pdtName;
28. newm->pdtPrice = pdt->pdtprice;
29. newm->salesVolume = pdt->salesVolume;
30. newm->StoreName = now->storeName;
31. SortMessage \*SortM2 = SortM1;
32. SortMessage \*SortM3 = SortM->next;//循环变量
33. **while**(SortM3 != SortM1)
34. {
35. **if**(SortM3->salesVolume < newm->salesVolume)
36. {
37. SortM2 = SortM3;
38. **break**;
39. }//寻找插入位置，按照销量降序
40. SortM3 = SortM3->next;
41. }
42. SortM2->last->next = newm;
43. newm->next = SortM2;
44. SortM2->last = newm;
45. **break**;
46. }
47. **else**
48. pdt = pdt->next;
49. }
50. now = now->next;
51. }
52. **return** OK;
53. }
55. **bool** ShowMessage(SortMessage \*&SortM)
56. {
57. **if**(SortM->id == 0)
58. {
59. cout << "没有商铺销售此商品" << endl;
60. **return** ERROR;
61. }
62. SortMessage \*now = SortM->next;
63. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
64. cout << "商品名称："<<SortM->pdtName << endl;
65. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;
66. **while**(now != SortM->last)
67. {
68. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
69. cout << "第" << now->id << "个商铺" << endl;
70. cout << "商铺名:" << now->StoreName << endl;
71. cout << "此商品价格：" << now->pdtPrice << endl;
72. cout << "此商品销量：" << now->salesVolume << endl;
73. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
74. now = now->next;
75. }
76. **return** OK;
77. }

Message.h

1. #ifndef MESSAGE\_H\_INCLUDED
2. #define MESSAGE\_H\_INCLUDED
3. #include<iostream>
4. #include<cstdio>
5. #include<cstdlib>
6. #include"Store.h"
7. **using** **namespace** std;
8. **typedef** **struct** SortMessage
9. {
10. **int** id;//存放拥有该商品的商铺的个数
11. string StoreName;
12. string pdtName;
13. **double** pdtPrice;
14. **int** salesVolume;
15. **struct** SortMessage \*next;
16. **struct** SortMessage \*last;
17. }SortMessage;
18. **bool** CreateSort(Store \*&S,string pdtName,SortMessage \*&SortM);
19. **bool** ShowMessage(SortMessage \*&SortM);
21. #endif // MESSAGE\_H\_INCLUDED

Product.h

1. #ifndef PRODUCT\_H\_INCLUDED
2. #define PRODUCT\_H\_INCLUDED
3. #include<iostream>
4. #include<cstdio>
5. **using** **namespace** std;
6. **typedef** **struct** Product
7. {
8. string pdtName;
9. **double** pdtprice;
10. **int** salesVolume;//头结点存商品数目
11. **struct** Product \*next;
12. }Product;

15. #endif // PRODUCT\_H\_INCLUDED

Store.h

1. #ifndef STORE\_H\_INCLUDED
2. #define STORE\_H\_INCLUDED
3. #include<iostream>
4. #include<cstdio>
5. #include"Product.h"
6. **using** **namespace** std;
7. **typedef** **struct** Store
8. {
9. **int** storeId;//头结点存商铺数目
10. string storeName;
11. **int** storeCredit;
12. Product \*pdt;
13. **struct** Store \*next;
14. }Store;
16. **bool** CreateStore(Store \*&S);//S是Store链表的头结点
17. **bool** InsertStore(Store \*&S);
18. **bool** checkMessage(Store \*&S,**int** id);
19. **bool** DeleteStore(Store \*&S,**int** id);
20. **bool** InsertProduct(Store \*S,**int** id);
21. **bool** DeleteProduct(Store \*&S,**int** id,string pdtName);
22. **bool** ChangeProduct(Store \*S,**int** id,string pdtName);
23. **bool** Purchase(Store \*S,**int** id,string pdtName,**int** num);
24. **void** readTxT(Store \*&S);
25. **void** writeTXT(Store \*&S);
26. **bool** TraverseStore(Store \*&S);
27. #endif // STORE\_H\_INCLUDED

Store.cpp

1. #include"Store.h"
2. #include<cstdlib>
3. #include<fstream>
4. #include<cassert>
5. #include<sstream>
6. #include<string>
7. #include<cstring>
8. #define OK 1
9. #define ERROR 0
10. #define OVERFLOW 0
11. #define TRUE 1
12. #define FALSE 0
13. **using** **namespace** std;
14. **bool** CreateStore(Store \*&S)//S是Store链表的头结点
15. {
16. S = **new** Store;
17. **if**(!S)
18. exit(OVERFLOW);
19. S -> storeId = 0;
20. S -> pdt = NULL;
21. S -> storeCredit = 0;
22. S -> next = NULL;
23. **return** OK;
24. }
25. **bool** InsertStore(Store \*&S)
26. {
27. **if**(!S)
28. **return** ERROR;
29. Store \*newStore = **new** Store;
30. **if**(!newStore)
31. exit(OVERFLOW);
32. newStore -> pdt = **new** Product;
33. Product \*pdt = newStore->pdt;
34. pdt->next = NULL;
35. pdt->salesVolume = 0;//初始化商铺的商品数目为0
36. string name;
37. **int** num = 0;
38. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* "<< endl;
39. cout << "请输入建立商铺的信息！"<< endl;
40. cout << "商铺名：";
41. cin >> name;
42. cout << "信誉度：";
43. cin >> num;
44. newStore->storeCredit = num;
45. cout << newStore->storeName<<endl;
46. newStore->storeName = name;
47. newStore ->next = NULL;
48. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
49. S->storeId++;//商铺总数+1
50. newStore->storeId = S->storeId;
51. **if**(S->next == NULL)
52. S->next = newStore;
53. **else**
54. {
55. Store \*now = S->next;
56. **while**(now->storeId != S->storeId-1)
57. now = now->next;
58. now->next = newStore;
59. }
60. **return** OK;
61. }
62. **bool** DeleteStore(Store \*&S,**int** id)
63. {
64. **if**(id > S->storeId || id <= 0)
65. {
66. cout << "不存在该店铺！" << endl;
67. **return** ERROR;
68. }
69. Store \*now = S->next;
70. Store \*last = S;
71. **while**(now->storeId != id && now != NULL )
72. {
73. last = now;
74. now = now->next;
75. }
77. last->next = now->next;
78. free(now->pdt);
79. free(now);
80. now = last->next;
81. **while**(now != NULL)
82. {
83. now->storeId--;
84. now = now->next;
85. }
86. S->storeId--;
87. **return** OK;
88. }
90. **bool** checkMessage(Store \*&S,**int** id)
91. {
92. **if**(id > S->storeId || id <= 0)
93. {
94. cout << "不存在该店铺！" << endl;
95. **return** ERROR;
96. }
97. Store \*snow = S->next;
98. **while**(snow->storeId != id)
99. snow = snow->next;
100. Product \*pnow = snow->pdt;
101. cout << endl << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
102. cout << "\*该店铺信息如下：" << endl;
103. cout << "\*店铺号：" << snow->storeId << endl;
104. cout << "\*店铺名：" << snow->storeName << endl;
105. cout << "\*店铺信誉度：" << snow->storeCredit << endl;
106. cout << "\*商品信息：";
107. **if**(pnow->next == NULL)
108. cout << "无商品信息！" << endl;
109. **else**
110. cout << "该商铺有" << pnow->salesVolume << "个产品" << endl;
111. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
112. cout<<endl;
113. pnow = pnow->next;
114. **while**(pnow != NULL)
115. {
116. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
117. cout << "商品名：" << pnow->pdtName << endl;
118. cout << "商品价格：" << pnow->pdtprice << endl;
119. cout << "商品销量：" << pnow->salesVolume << endl;
120. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" <<endl<<endl;
121. pnow = pnow->next;
122. }
123. **return** OK;
124. }
126. **bool** InsertProduct(Store \*S,**int** id)
127. {
128. **if**(id > S->storeId || id <= 0)
129. {
130. cout << "不存在该店铺！" << endl;
131. **return** ERROR;
132. }
133. Product \*newPdt = **new** Product;
134. string Name;
135. **int** num;
136. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
137. cout << "请输入加入商品的信息！" << endl;
138. cout << "商品名称：";
139. cin >> Name;
140. cout << "商品价格：";
141. cin >> num;
142. newPdt->pdtprice = num;
143. newPdt->pdtName = Name;
144. newPdt->salesVolume = 0;
145. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
146. Store \*snow = S->next;
147. **while**(snow->storeId != id)
148. snow = snow->next;
149. Product \*pnow = snow->pdt;
150. pnow->salesVolume++;
151. **while**(pnow->next != NULL)
152. pnow = pnow->next;
153. pnow->next = newPdt;
154. newPdt->next = NULL;
155. **return** OK;
156. }
157. **bool** DeleteProduct(Store \*&S,**int** id,string pdtName)
158. {
159. **if**(id > S->storeId || id <= 0)
160. {
161. cout << "不存在该店铺！" << endl;
162. **return** ERROR;
163. }
164. Store \*snow = S->next;
165. **while**(snow->storeId != id)
166. snow = snow->next;
167. Product \*p = snow->pdt;
168. p->salesVolume--;
169. Product \*pnow = (snow->pdt)->next;
170. Product \*plast = snow->pdt;
171. **while**(pnow != NULL && pnow->pdtName != pdtName)
172. {
173. plast = pnow;
174. pnow = pnow->next;
175. }
176. **if**(pnow == NULL)
177. {
178. cout << "没有找到该商品！" << endl;
179. **return** ERROR;
180. }
181. cout << "该商品信息为：" << endl;
182. cout << "Name：" << pnow->pdtName << endl;
183. cout << "price：" << pnow->pdtprice << endl;
184. cout << "volume：" << pnow->salesVolume << endl;
185. plast->next = pnow->next;
186. plast = plast->next;
187. **delete**(pnow);
188. **return** OK;
189. }
190. **bool** ChangeProduct(Store \*S,**int** id,string pdtName)
191. {
192. **if**(id > S->storeId || id <= 0)
193. {
194. cout << "不存在该店铺！" << endl;
195. **return** ERROR;
196. }
197. Store \*snow = S->next;
198. **while**(snow->storeId != id)
199. snow = snow->next;
200. Product \*pnow = (snow->pdt)->next;
201. **while**(pnow != NULL && pnow->pdtName != pdtName)
202. pnow = pnow->next;
203. **if**(pnow == NULL)
204. {
205. cout << "没有找到该商品！" << endl;
206. **return** ERROR;
207. }
208. cout << "你选择的商品信息如下" << endl;
209. cout << "商品名称：" << pnow->pdtName << endl;
210. cout << "商品价格：" << pnow->pdtprice << endl;
211. cout << "商品销量：" << pnow->salesVolume << endl << endl;
212. **int** choose;
213. cout << "请选择要修改的信息！" <<endl;
214. cout << "1：商品名称" << endl;
215. cout << "2：商品价格" << endl;
216. cout << "3: 商品名称及价格" << endl;
217. cin >> choose;
218. **if**(choose == 1)
219. {
220. cout << "请输入新的商品名称：";
221. cin >> pnow->pdtName;
222. **return** OK;
223. }
224. **else** **if**(choose == 2)
225. {
226. cout << "请输入新的商品价格：";
227. cin >> pnow->pdtprice;
228. **return** OK;
229. }
230. **else** **if**(choose == 3)
231. {
232. cout << "请输入新的商品名称：";
233. cin >> pnow->pdtName;
234. cout << "请输入新的商品价格：";
235. cin >> pnow->pdtprice;
236. **return** OK;
237. }
238. **else**
239. cout << "输入错误，请重新输入！"<<endl;
240. }
241. **bool** Purchase(Store \*S,**int** id,string pdtName,**int** num)
242. {
243. **if**(id > S->storeId || id <= 0)
244. {
245. cout << "不存在该店铺！" << endl;
246. **return** ERROR;
247. }
248. Store \*snow = S->next;
249. **while**(snow->storeId != id)
250. snow = snow->next;
251. Product \*pnow = (snow->pdt)->next;
252. **while**(pnow != NULL && pnow->pdtName != pdtName)
253. pnow = pnow->next;
254. **if**(pnow == NULL)
255. {
256. cout << "没有找到该商品！" << endl;
257. **return** ERROR;
258. }
259. cout << "你选择的商品信息如下" << endl;
260. cout << "商品名称：" << pnow->pdtName << endl;
261. cout << "商品价格：" << pnow->pdtprice << endl;
262. cout << "商品销量：" << pnow->salesVolume << endl << endl;
263. cout << "确定购买？(1为确定)" << endl;
264. **int** choose = 0;
265. cin >> choose;
266. **if**(choose == 1)
267. pnow->salesVolume += num;
268. cout << "恭喜你成功购买了 " << pnow->pdtName << " 商品 " << num << "件" << endl;
269. **return** OK;
270. }
272. **void** readTxT(Store \*&S)
273. {
274. S = **new** Store;
275. ifstream infile;
276. infile.open("data.txt",ios::in);
277. **if**(!infile.is\_open())
278. {
279. cout << "文件打开失败！" << endl;
280. exit(ERROR);
281. }
282. assert(infile.is\_open());
283. string str;
284. stringstream ss;
285. getline(infile,str);
286. **int** Storenum;
287. ss << str;
288. ss >> Storenum;
289. S->storeId = Storenum;//读入商铺个数
290. **for**(**int** i = 0;i < S->storeId;i++)
291. {
292. **int** m;
293. Store \*newS = **new** Store;
294. newS->pdt = **new** Product;
295. Product \*pdt = newS->pdt;
296. newS->next = NULL;
297. pdt->next = NULL;
299. getline(infile,str);
300. m = atoi(str.c\_str());
301. newS->storeId = m;//读入商铺ID
303. getline(infile,str);
304. newS->storeName = str;//读入商铺名
306. getline(infile,str);
307. m = atoi(str.c\_str());
308. newS->storeCredit = m;//读入商铺信息
310. getline(infile,str);
311. m = atoi(str.c\_str());
312. pdt->salesVolume = m;//读入商品个数
313. cout << "店铺编号：" << newS->storeId << endl;
314. cout << "店铺名：" << newS->storeName << endl;
315. cout << "店铺信誉："<<newS->storeCredit << endl;
316. cout << "店铺商品数：" << pdt->salesVolume << endl;
317. cout << endl;
318. **for**(**int** j = 0;j < pdt->salesVolume;j++)
319. {
320. Product \*newP  = **new** Product;
321. newP->next = NULL;
323. getline(infile,str);
324. newP->pdtName = str;//读入商品名
326. getline(infile,str);
327. **double** price = (**double**)atof(str.c\_str());//string 转 double
328. newP->pdtprice = price;//读入商品价格
330. getline(infile,str);
331. **int** volume = atoi(str.c\_str());//string 转 int
332. newP->salesVolume = volume;//读入商品销量
334. /\*cout << "Name：" << newP->pdtName << endl;
335. cout << "price：" << price << "  " << newP->pdtprice << endl;
336. cout << "volume：" << newP->salesVolume << endl;
337. cout << endl;\*/
338. Product \*pnow = newS->pdt;
339. **for**(**int** k = 0;k < j;k++)
340. pnow = pnow ->next;
341. pnow->next = newP;//连上pdt链表
342. }
343. Store \*snow = S;
344. **for**(**int** k = 0;k < i;k++)
345. snow = snow->next;
346. snow->next = newS;
347. }
348. infile.close();
349. }
351. **void** writeTXT(Store \*&S)
352. {
353. ofstream outfile;
354. outfile.open("data.txt",ios::out);
355. **if**(!outfile.is\_open())
356. {
357. cout << "文件打开失败！" << endl;
358. exit(ERROR);
359. }
360. Store \*snow = S->next;
361. outfile << S->storeId << endl;
362. **for**(**int** i = 0;i < S->storeId;i++)
363. {
365. outfile << snow->storeId << endl;
366. outfile << snow->storeName << endl;
367. outfile << snow->storeCredit << endl;
368. Product \*pnow = snow->pdt;
369. **int** num  = pnow->salesVolume;
370. outfile << pnow->salesVolume <<endl;
371. **for**(**int** j = 0;j < num;j++)
372. {
373. pnow = pnow->next;
374. outfile << pnow->pdtName << endl;
375. outfile << pnow->pdtprice <<endl;
376. outfile << pnow->salesVolume <<endl;
377. }
378. snow = snow->next;
379. }
380. outfile.close();
381. }
382. **bool** TraverseStore(Store \*&S)
383. {
384. Store \*snow = S->next;
386. **for**(**int** i = 0 ;i < S->storeId;i++)
387. {
388. Product \*pnow = snow->pdt;
389. **int** num = pnow->salesVolume;
390. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
391. cout << "店铺编号：" << snow->storeId << endl;
392. cout << "店铺名：" << snow->storeName << endl;
393. cout << "店铺信誉："<<snow->storeCredit << endl;
394. cout << "店铺商品数：" << pnow->salesVolume << endl;
395. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
396. cout << endl;
397. **for**(**int** j = 0;j < num;j++)
398. {
399. pnow = pnow->next;
400. cout << "Name：" << pnow->pdtName << endl;
401. cout << "price：" << pnow->pdtprice << endl;
402. cout << "volume：" << pnow->salesVolume << endl;
403. cout << endl;
404. }
405. snow = snow->next;
406. }
407. }

Shop.cpp

1. #include"Product.h"
2. #include"Store.h" //函数声明
3. #include"Message.h"//储存
4. #include"Store.cpp"//函数实现
5. #include"Message.cpp"
6. #define OK 1
7. #define ERROR 0
8. **int** main()
9. {
10. Store \*S;
11. CreateStore(S);
12. **int** count = 0;
13. **while**(1)
14. {
15. cout << "请输入你要进行的操作！" << endl;
16. cout << "1：增加商铺！" << endl;
17. cout << "2：删去商铺！" << endl;
18. cout << "3：增加商铺的商品！" << endl;
19. cout << "4：删去商铺的商品！" << endl;
20. cout << "5：改动商铺的商品信息！" << endl;
21. cout << "6：购买商品！" << endl;
22. cout << "7：查看商铺信息！" <<endl;
23. cout << "8：输入商品名称商铺按销量高到低排列！" << endl;
24. cout << "9：从文件里读出信息！" << endl;
25. cout << "10：查看所有商铺！" << endl;
26. cout << "11：写入文件！" <<endl;
27. cout << "12：退出！" << endl;
28. scanf("%d",&count);
29. **if**(count == 1)
30. {
31. InsertStore(S);
32. }
33. **if**(count == 2)
34. {
35. **int** Id;
36. cout << "请输入要删除的的商铺编号：";
37. cin >> Id;
38. checkMessage(S,Id);
39. **char** ch;
40. cout<<"是否确定：（Y/N）" << endl;
41. cin >> ch;
42. **if**(ch == 'Y')
43. DeleteStore(S,Id);
44. }
45. **if**(count == 3)
46. {
47. **int** Id;
48. cout << "请输入要增加商品的的商铺编号：";
49. cin >> Id;
50. checkMessage(S,Id);
51. **char** ch;
52. cout<<"是否确定：（Y/N）" << endl;
53. cin >> ch;
54. **if**(ch == 'Y')
55. InsertProduct(S,Id);
56. }
57. **if**(count == 4)
58. {
59. **int** Id;
60. string Name;
61. cout << "请输入要删去商品的的商铺编号：";
62. cin >> Id;
63. checkMessage(S,Id);
64. **char** ch;
65. cout<<"是否确定：（Y/N）" << endl;
66. cin >> ch;
67. **if**(ch == 'Y')
68. {
69. cout << "请输入要删去的商品名称：";
70. cin >> Name;
71. DeleteProduct(S,Id,Name);
72. }
73. }
74. **if**(count == 5)
75. {
76. **int** Id;
77. string Name;
78. cout << "请输入要改动信息的商品所在的商铺编号：";
79. cin >> Id;
80. checkMessage(S,Id);
81. **char** ch;
82. cout<<"是否确定：（Y/N）" << endl;
83. cin >> ch;
84. **if**(ch == 'Y')
85. {
86. cout << "请输入要改动信息的商品名称：";
87. cin >> Name;
88. ChangeProduct(S,Id,Name);
89. }
90. }
91. **if**(count == 6)
92. {
93. **int** Id;
94. string Name;
95. **int** num;
96. cout << "请输入要购买的商品所在的商铺编号：";
97. cin >> Id;
98. checkMessage(S,Id);
99. cout << "请输入要购买的商品名称：";
100. cin >> Name;
101. cout << "请输入要购买的商品数量：";
102. cin >> num;
103. Purchase(S,Id,Name,num);
104. }
105. **if**(count == 7)
106. {
107. **int** Id;
108. cout << "请输入要查找的商铺编号：";
109. cin >> Id;
110. checkMessage(S,Id);
111. }
112. **if**(count == 8)
113. {
114. string name;
115. cout << "请输入要查找的商品" << endl;
116. cin >> name;
117. SortMessage \*SortM;
118. CreateSort(S,name,SortM);
119. ShowMessage(SortM);
120. }
121. **if**(count == 9)
122. {
123. readTxT(S);
124. }
125. **if**(count == 10)
126. {
127. TraverseStore(S);
128. }
129. **if**(count == 11)
130. {
131. writeTXT(S);
132. }
133. **if**(count == 12)
134. {
135. **return** OK;
136. }
137. cout << endl;
138. }
139. **return** OK;
140. }

# 二、 公共钥匙盒

# 2.1 题目简介

公共钥匙盒（必做）（线性表，栈，队列）

[问题描述]

　　有一个学校的老师共用N个教室，按照规定，所有的钥匙都必须放在公共钥匙盒里，老师不能带钥匙回家。每次老师上课前，都从公共钥匙盒里找到自己上课的教室的钥匙去开门，上完课后，再将钥匙放回到钥匙盒中。

　　钥匙盒一共有N个挂钩，从左到右排成一排，用来挂N个教室的钥匙。一串钥匙没有固定的悬挂位置，但钥匙上有标识，所以老师们不会弄混钥匙。

　　每次取钥匙的时候，老师们都会找到自己所需要的钥匙将其取走，而不会移动其他钥匙。每次还钥匙的时候，还钥匙的老师会找到最左边的空的挂钩，将钥匙挂在这个挂钩上。如果有多位老师还钥匙，则他们按钥匙编号从小到大的顺序还。如果同一时刻既有老师还钥匙又有老师取钥匙，则老师们会先将钥匙全还回去再取出。

　　今天开始的时候钥匙是按编号从小到大的顺序放在钥匙盒里的。有K位老师要上课，给出每位老师所需要的钥匙、开始上课的时间和上课的时长，假设下课时间就是还钥匙时间，请问最终钥匙盒里面钥匙的顺序是怎样的？

[基本要求]

输入格式

　　输入的第一行包含两个整数N, K。

　　接下来K行，每行三个整数w, s, c，分别表示一位老师要使用的钥匙编号、开始上课的时间和上课的时长。可能有多位老师使用同一把钥匙，但是老师使用钥匙的时间不会重叠。

　　保证输入数据满足输入格式，你不用检查数据合法性。

输出格式

　　输出一行，包含N个整数，相邻整数间用一个空格分隔，依次表示每个挂钩上挂的钥匙编号。

样例输入

5 2

4 3 3

2 2 7

样例输出

1 4 3 2 5

样例说明

　　第一位老师从时刻3开始使用4号教室的钥匙，使用3单位时间，所以在时刻6还钥匙。第二位老师从时刻2开始使用钥匙，使用7单位时间，所以在时刻9还钥匙。

　　每个关键时刻后的钥匙状态如下（X表示空）：

　　时刻2后为1X345；

　　时刻3后为1X3X5；

　　时刻6后为143X5；

　　时刻9后为14325。

课程设计要求：

（1）要求从文本文件中输入；

（2）根据时间进程，将取走钥匙和归还钥匙分别视为事件，放入队列中，然后通过每个事件的先后发生对钥匙盒的状态进行变更；

（3）严格按照要求的输入输出格式进行数据的输入、输出（训练CSP考试中的格式化输入输出的正确性）；

（4）选做：通过图形界面来显示钥匙盒的即时状态，以及事件队列的状态。

# 2.2 数据结构

1. **typedef** **struct** {
2. **int** keyId;
3. **int** time;
4. }Key;

# 2.3 算法设计思想

用数据结构模拟动作的发生，按照时间，还借，编号的顺序将动作排序，一一执行

# 2.4 测试数据和结果



# 2.5 算法时间复杂度

O(N\*K)：main

# 2.6 源代码（163

1. #include<cstdio>
2. #include<cstdlib>
3. #include<windows.h>
4. #include<iostream>
5. #include<queue>
7. **using** **namespace** std;
9. **typedef** **struct** {
10. **int** keyId;
11. **int** time;
12. }Key;
14. // 全局变量
15. **FILE** \*fp;
16. **char** fileName[] = "input.txt";

19. // 打开文件
20. **void** fileOpen(**char** \*method) {
21. **if**((fp=fopen(fileName,method)) == NULL) {
22. printf("file cannot be opened\n");
23. **return**;
24. }
25. //else printf("file has been opened\n");
26. }
28. // 关闭文件
29. **void** fileClose() {
30. **if**(fclose(fp)!=0) printf("file cannot be closed\n");
31. //else printf("file has been closed\n");
32. **return**;
33. }
35. // 显示图形界面
36. **void** ShowBox(**int** n, **int** \*box, **int** time, **int** \*key) {
37. system("cls");
38. cout<<"正在盒子里的钥匙"<<endl;
39. cout<<"                          ";
40. **for** (**int** i=1; i<=n; i++) {
41. printf("-----");
42. }
43. printf("\n");
44. cout<<"                          ";
45. **for** (**int** i=1; i<=n; i++) {
46. **if** (box[i]) printf("| %2d ",box[i]);
47. **else** printf("|    ");
48. }
49. printf("|\n");
50. cout<<"                          ";
51. **for** (**int** i=1; i<=n; i++) {
52. printf("-----");
53. }
54. printf("\n\n");
55. cout<<"在老师手上的钥匙"<<endl;
56. cout<<"                          ";
57. **for** (**int** i=1; i<=n; i++) {
58. **if** (key[i] == 0) printf(" %d ", i);
59. }
60. printf("\n");
62. Sleep(800 \* time);
63. }
65. **int** main() {
66. fileOpen("r");
68. **int** n = 0, k = 0;
69. fscanf(fp, "%d%d", &n, &k);
71. **int** w = 0, s = 0, c = 0;
72. Key \*borrow = (Key\*)malloc(**sizeof**(Key) \* k), \*ret = (Key\*)malloc(**sizeof**(Key) \* k);//借出和返回的储存
73. **for** (**int** i=0; i<k; i++) {
74. fscanf(fp, "%d%d%d", &w, &s, &c);
75. // 借出的钥匙号和时间
76. borrow[i].keyId = w;
77. borrow[i].time = s;
78. // 归还的钥匙号和时间
79. ret[i].keyId = w;
80. ret[i].time = s + c;
81. }
83. // 对借出和归还的钥匙，根据时间从小到大排序
84. **for** (**int** i=0; i<k; i++) {
85. **for** (**int** j=i; j<k; j++) {
86. **if** (borrow[j].time < borrow[i].time || (borrow[j].time == borrow[i].time && borrow[j].keyId < borrow[i].keyId)) {
87. Key temp = borrow[j];
88. borrow[j] = borrow[i];
89. borrow[i] = temp;
90. }
91. **if** (ret[j].time < ret[i].time || (ret[j].time == ret[i].time && ret[j].keyId < ret[i].keyId)) {
92. Key temp = ret[j];
93. ret[j] = ret[i];
94. ret[i] = temp;
95. }
96. }
97. }
99. // 将事件放进队列中
100. queue<Key> q;
101. **int** i = 0, j = 0;
102. **while** (i<k && j<k) {
103. **if** (borrow[i].time < ret[j].time) {
104. q.push(borrow[i]);
105. i++;
106. }
107. **else** **if** (borrow[i].time > ret[j].time) {
108. q.push(ret[j]);
109. j++;
110. }
111. **else** {
112. q.push(ret[j]);
113. j++;
114. }
115. }
116. **while** (i<k) {
117. q.push(borrow[i]);
118. i++;
119. }
120. **while** (j<k) {
121. q.push(ret[j]);
122. j++;
123. }
125. // 对key钥匙和box钥匙盒数组初始化。key数组：下标为编号，值为在box中的位置
126. **int** key[n+1], box[n+1];
127. **for** (**int** i=1; i<=n; i++) {
128. box[i] = i;
129. key[i] = i;
130. }
131. // 根据队列进行事件安排
132. **int** time = 0;
133. **while** (!q.empty()) {
134. Key cur = q.front();
135. q.pop();
137. ShowBox(n, box, cur.time - time, key);
138. time = cur.time;
140. // 如果该钥匙已经发在一个位置了，就取出
141. **if** (key[cur.keyId]) {
142. box[key[cur.keyId]] = 0;
143. key[cur.keyId] = 0;
144. }
145. **else** {
146. **for** (**int** i=1; i<=n; i++) {
147. **if** (box[i] == 0) {
148. key[cur.keyId] = i;
149. box[i] = cur.keyId;
150. **break**;
151. }
152. }
153. }
154. }
155. ShowBox(n, box, 1, key);

158. printf("\n所有老师已经下课\n");
160. free(borrow);
161. free(ret);
162. fileClose();
163. }

# 三、树的应用

# 3.1 题目简介

树的应用 (必做)（树）

[问题描述]

　　JSON (JavaScript Object Notation) 是一种轻量级的数据交换格式，可以用来描述半结构化的数据。JSON 格式中的基本单元是值 (value)，出于简化的目的本题只涉及 2 种类型的值：

　　\* 字符串 (string)：字符串是由双引号 " 括起来的一组字符（可以为空）。如果字符串的内容中出现双引号 "，在双引号前面加反斜杠，也就是用 \" 表示；如果出现反斜杠 \，则用两个反斜杠 \\ 表示。反斜杠后面不能出现 " 和 \ 以外的字符。例如：""、"hello"、"\"\\"。

　　\* 对象 (object)：对象是一组键值对的无序集合（可以为空）。键值对表示对象的属性，键是属性名，值是属性的内容。对象以左花括号 { 开始，右花括号 } 结束，键值对之间以逗号 , 分隔。一个键值对的键和值之间以冒号 : 分隔。键必须是字符串，同一个对象所有键值对的键必须两两都不相同；值可以是字符串，也可以是另一个对象。例如：{}、{"foo": "bar"}、{"Mon": "weekday", "Tue": "weekday", "Sun": "weekend"}。

　　除了字符串内部的位置，其他位置都可以插入一个或多个空格使得 JSON 的呈现更加美观，也可以在一些地方换行，不会影响所表示的数据内容。例如，上面举例的最后一个 JSON 数据也可以写成如下形式。

　　{

　　"Mon": "weekday",

　　"Tue": "weekday",

　　"Sun": "weekend"

　　}

　　给出一个 JSON 格式描述的数据，以及若干查询，编程返回这些查询的结果。

输入格式

　　第一行是两个正整数 n 和 m，分别表示 JSON 数据的行数和查询的个数。

　　接下来 n 行，描述一个 JSON 数据，保证输入是一个合法的 JSON 对象。

　　接下来 m 行，每行描述一个查询。给出要查询的属性名，要求返回对应属性的内容。需要支持多层查询，各层的属性名之间用小数点 . 连接。保证查询的格式都是合法的。

[基本要求]

输出格式

　　对于输入的每一个查询，按顺序输出查询结果，每个结果占一行。

　　如果查询结果是一个字符串，则输出 STRING <string>，其中 <string> 是字符串的值，中间用一个空格分隔。

　　如果查询结果是一个对象，则输出 OBJECT，不需要输出对象的内容。

　　如果查询结果不存在，则输出 NOTEXIST。

样例输入

10 5

{

"firstName": "John",

"lastName": "Smith",

"address": {

"streetAddress": "2ndStreet",

"city": "NewYork",

"state": "NY"

},

"esc\\aped": "\"hello\""

}

firstName

address

address.city

address.postal

esc\aped

样例输出

STRING John

OBJECT

STRING NewYork

NOTEXIST

STRING "hello"

[基本要求]

（1）要求从文本文件中输入；

（2）本题目其实就是一棵普通的树（即每个结点的孩子数不固定，不能单纯采用n叉树来解决），可以考虑使用孩子兄弟表示法等进行表示和存储；

（3）严格按照要求的输入输出格式进行数据的输入、输出（训练CSP考试中的格式化输入输出的正确性）；

（4）选做：使用图形界面（或字符格式化摆成的树形结构，参考Linux下的tree命令），以树状形式显示输入的JSON格式数据。

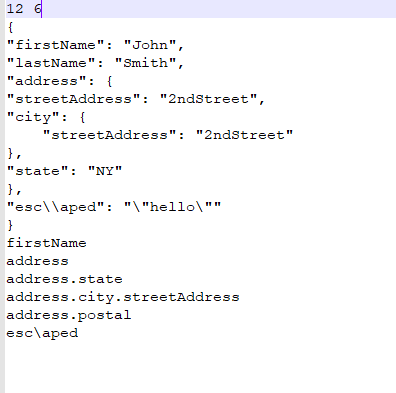
# 3.2 数据结构

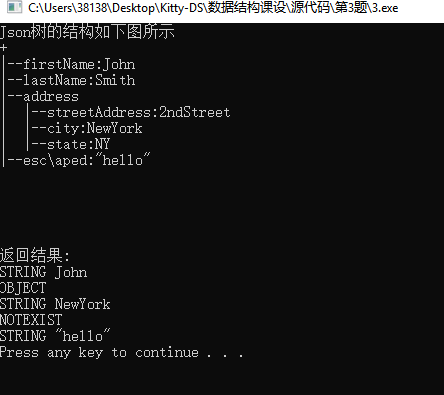
1. **typedef** **struct** Json
2. {
3. **bool** flag;
4. //标识是不是OBJECT
5. string key;
6. //键
7. string value;
8. //值
9. }json;
10. //映射键值的结构体

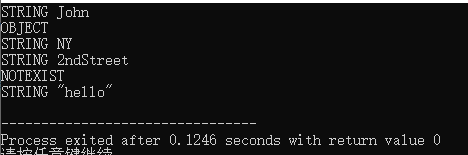
# 3.3 算法设计思想

对闭合状态的转化，设计为有限状态机，利用映射结构体保存映射并用hash函数索引

# 3.4 测试数据和结果







# 3.5 算法时间复杂度

文本长度为n

O(n):toJson

# 3.6 源代码（200

1. #include<cstdio>
2. #include<cstdlib>
3. #include<cstring>
4. #include<iostream>
5. #include<windows.h>
7. **using** **namespace** std;
9. **typedef** **struct** TNode {
10. **char** key[100];
11. **char** value[100];
12. TNode \*child;
13. TNode \*brother;
14. }\*Tree;
16. // 全局变量
17. **FILE** \*fp;
18. **char** fileName[] = "input.txt";

21. // 打开文件
22. **void** fileOpen(**char** \*method) {
23. **if**((fp=fopen(fileName,method)) == NULL) {
24. printf("file cannot be opened\n");
25. **return**;
26. }
27. //else printf("file has been opened\n");
28. }
30. // 关闭文件
31. **void** fileClose() {
32. **if**(fclose(fp)!=0) printf("file cannot be closed\n");
33. //else printf("file has been closed\n");
34. **return**;
35. }
37. // 先序遍历
38. **void** TraverseTree(Tree tree) {
39. **if** (tree) {
40. printf("%s:%s\n", tree->key, tree->value);
41. TraverseTree(tree->brother);
42. TraverseTree(tree->child);
43. }
44. }
46. // 销毁
47. **void** DestroyTree(Tree &tree) {
48. **if** (tree) {
49. DestroyTree(tree->brother);
50. DestroyTree(tree->child);
51. free(tree);
52. }
53. }
55. // 字符串处理函数
56. **void** HandleStr(**char** \*source, **int** start, **int** end, **char** \*dist) {
58. **int** i = start+1;
59. **int** j = 0;
60. **while** (i != end) {
61. **if** (source[i] == '\\') i++;
62. dist[j] = source[i];
63. i++;
64. j++;
65. }
66. dist[j] = '\0';
67. }


71. // 建树函数，递归
72. **void** CreateTree(Tree &tree) {
73. **char** str[1000];
74. fscanf(fp, "%s", str);      // 读取'{'的下一行的字符串
75. **int** i = 0;
77. TNode \*preNode = tree, \*node;
78. **bool** isFirstNode = **true**;    // 判断是否为根节点
80. **while** (str[i] != '}') {
81. node = (TNode\*)malloc(**sizeof**(TNode));
83. printf("前一半======%s\n", str);
84. **if** (str[i] == '"') {
85. **int** j = i;
86. **while** (str[j] != ':') j++;
87. HandleStr(str, i, j-1, node->key);
88. i = j;  // i位置为':'
89. }
90. i++;// 跳过':'到value部分
92. printf("后一半=====%s\n", str);
93. **if** (str[i] == '{') {
94. strcpy(node->value, "object");
95. CreateTree(node->child); // 值是一个对象，则递归操作建子树
96. }
97. **else** **if** (str[i] == '"') {
98. node->child = NULL;
99. **int** j = i;
100. // j<str字符串长度是为了结束字符'}'前没有','的情况
101. **while** (str[j] != ',' && str[j] != '}' && j < strlen(str)) j++;
102. HandleStr(str, i, j-1, node->value);
103. i = j;      // i位置为','或'}'
104. }
106. // 为第一个结点时将node结点指针赋给tree指针
107. **if** (isFirstNode) {
108. tree = node;
109. isFirstNode = **false**;
110. }
111. // 不是第一个结点，则将上个结点的兄弟指针指向当前结点
112. **else** preNode->brother = node;
113. preNode = node;
115. fscanf(fp, "%s", str);
116. i = 0;
117. }
118. node->brother = NULL;
119. }
121. // 找value值
122. **char**\* FindValue(Tree tree, **char** \*str) {
123. **int** i = 0;
124. **char** key[1000];
125. **while** (str[i]!='.' && i<strlen(str)) {
126. key[i] = str[i];
127. i++;
128. }
129. key[i] = '\0';
131. TNode \*node = tree;
132. **while** (node) {
133. **if** (strcmp(node->key, key) == 0) {
134. **if** (str[i] == '.') {
135. **int** j = 0;
136. **char** nextStr[1000];
137. i++;
138. **while** (i<strlen(str)) {
139. nextStr[j] = str[i];
140. i++;
141. j++;
142. }
143. nextStr[j] = '\0';
144. **return** FindValue(node->child, nextStr);
145. }
146. **else** **return** node->value;
147. }
148. node = node->brother;
149. }
150. **return** NULL;
151. }
152. // 显示树
153. **void** ShowTree(Tree tree, **int** tab) {
154. TNode \*node = tree;
155. **while** (node) {
156. **for** (**int** i=0; i<tab; i++) printf("|  ");
157. printf("|--%s", node->key);
158. **if** (node->child) {
159. printf("\n");
160. ShowTree(node->child, tab+1);
161. }
162. **else** printf(":%s\n", node->value);
163. node = node->brother;
164. }
165. }
167. **int** main() {
168. fileOpen("r");
169. **int** n = 0, m = 0;
170. fscanf(fp, "%d%d", &n, &m);
172. Tree tree = NULL;
173. **char** strLine[10000];
174. fscanf(fp, "%s", strLine);// 读取开始的'{'
175. CreateTree(tree);
177. system("cls");  // 清屏
178. cout<<"Json树的结构如下图所示"<<endl;
179. printf("+\n");
180. ShowTree(tree, 0);
182. cout<<"\n\n\n\n\n返回结果:"<<endl;
183. // 读取输入数据
184. **for** (**int** i=0; i<m; i++) {
185. fscanf(fp, "%s", strLine);
186. **char** \*ret = FindValue(tree, strLine);
187. **char** value[1000];
188. **if** (!ret) sprintf(value, "NOTEXIST");
189. **else** **if** (strcmp(ret, "object") == 0) sprintf(value, "OBJECT");
190. **else** sprintf(value, "STRING %s", ret);
191. printf("%s\n", value);
192. }
194. DestroyTree(tree);
195. system("pause");
196. fileClose();
197. }

# 四、Huffman编码与解码

# 4.1 题目简介

Huffman编码与解码(必做)（Huffman编码、二叉树）

[问题描述]

对一篇不少于2000字符的英文文章（source.txt），统计各字符出现的次数，实现Huffman编码(code.dat)，以及对编码结果的解码(recode.txt)。

[基本要求]

（1） 输出每个字符出现的次数和编码,并存储文件(Huffman.txt)。

（2） 在Huffman编码后，英文文章编码结果保存到文件中(code.dat)，编码结果必须是二进制形式，即0 1的信息用比特位表示，不能用字符’0’和’1’表示。

（3） 实现解码功能。

# 4.2 数据结构

1. **typedef** **struct** Node
2. {
3. **int** key;
4. //对应字符
5. **int** count;
6. //出现次数
7. **int** p;
8. //父结点
9. **int** l;
10. **int** r;
11. //左右子结点
12. }node;
14. **typedef** **struct** HaffmanMap
15. {
16. **int** val; //字符
17. string key; //编码
18. }HaffmanMap;
19. //映射编码和字符

# 4.3 算法设计思想

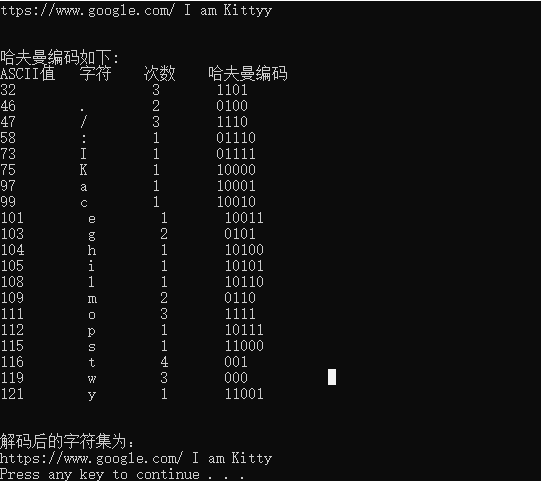
先通过计数统计字符频率

通过字符频率生成Huffman树然后进行编码

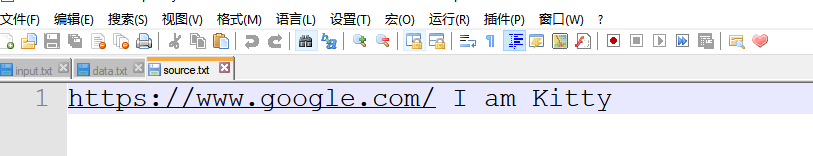
将编码后的源文件内容以二进制形式存入code.dat

读取Huffman.txt得到编码规则然后解码code.dat获得源文件内容decode.txt

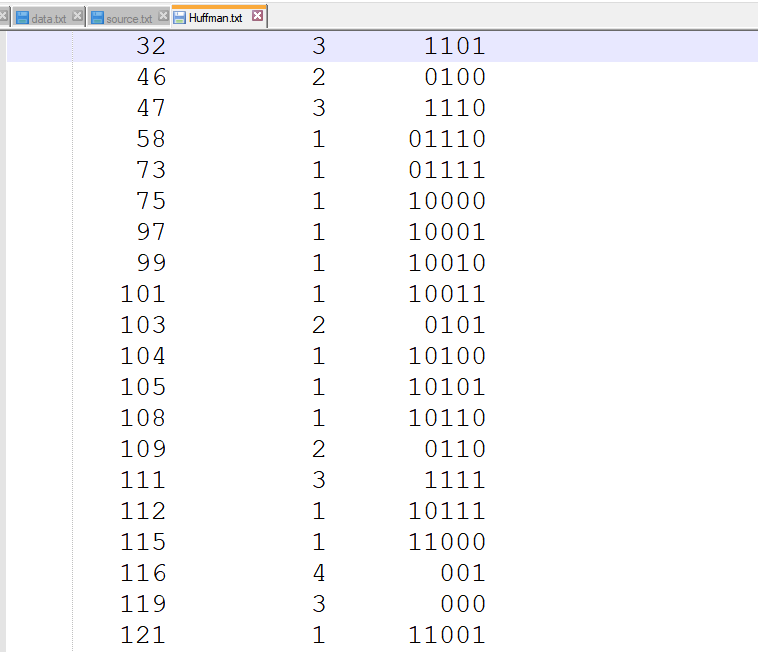
# 4.4 测试数据和结果



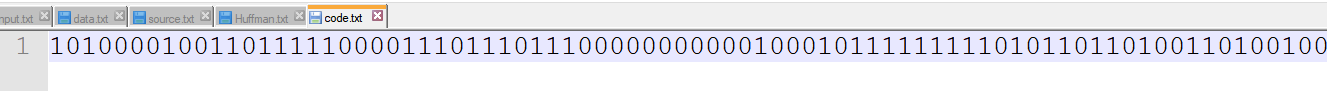
source.txt



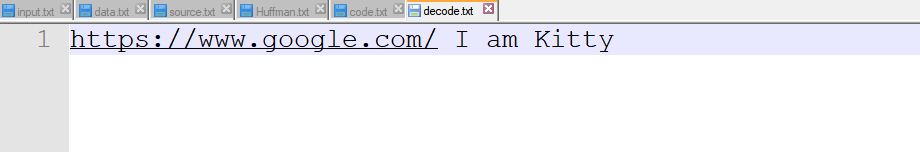
Huffman.txt



code.txt



decode.txt



# 4.5 算法时间复杂度

文本长度为n

main函数时间复杂度为O(n)

# 4.6 源代码(300

1. #include<iostream>
2. #include<iomanip>
3. #include<fstream>
4. #include<vector>
5. #include<string>
6. #define INF 9999999
8. **using** **namespace** std;
10. fstream sourceInput,HuffmanMap,Huffmancode,decodeOutput;
11. //文件流对象
13. **typedef** **struct** Node
14. {
15. **int** key;
16. //对应字符
17. **int** count;
18. //出现次数
19. **int** p;
20. //父结点
21. **int** l;
22. **int** r;
23. //左右子结点
24. }node;
26. **typedef** **struct** HaffmanMap
27. {
28. **int** val; //字符
29. string key; //编码
30. }HaffmanMap;
31. //映射编码和字符
33. node counts[256];
34. //统计字符出现次数
35. **int** num;
36. //统计出现的字符数
37. **int** bitmap[40000];
38. **int** length;
39. //用于存储二进制Huffman编码，和使用位的长度
40. vector<string> codes;
41. //用于存储字符串形式的编码
42. HaffmanMap test[10000];
43. //用于通过Huffman编码字符串索引到对应字符
45. **int** Sethash(string p)
46. {
47. **int** x = 1,y;
48. string key = p;
49. //用于处理碰撞
51. **for**(**int** i = 0;i < p.size();i++)
52. {
53. x \*= p[i];
54. x -= p[i] / 2;
55. x %= 10000;
56. }
57. y = x;
58. **while**(test[x].key.size() && !(test[x].key == key))
59. //如果发生碰撞，则向后移一位
60. {
61. x++;
62. x %= 10000;
63. **if**(x == y)
64. {
65. cout << "Error hash,memery is out!!!\n";
66. exit(0);
67. }
68. //防止溢出
69. }
71. **return** x;
72. }
73. //hash函数


77. **void** InitHaffman()
78. {
79. sourceInput.open("source.txt",ios::in);
80. //读取原始英文文章
81. sourceInput >> noskipws;
82. //不忽略空格和换行
84. HuffmanMap.open("Huffman.txt",ios::out);
85. //存储各字符出现次数和编码
87. Huffmancode.open("code.txt",ios::out );
88. //存储编码后的文章
89. Huffmancode << noskipws;
91. decodeOutput.open("decode.txt",ios::out);
92. //存储二进制形式的编码后的文章
94. **for**(**int** i = 0;i < 128;i++)
95. {
96. counts[i].key = i;
97. }
98. //初始化node中的key值
99. }
100. //初始化文件流和结构体数组
101. **void** FiniHaffman()
102. {
103. sourceInput.close();
104. HuffmanMap.close();
105. Huffmancode.close();
106. decodeOutput.close();
107. **return**;
108. }
109. //关闭文件流
110. **void** CreateHaffman()
111. {
112. **for**(**int** i = 0;i < num - 1;i++)
113. {
114. **int** min1 = INF,min2 = INF;
115. **int** a = 0,b = 0,j = 0;
116. **while**(j < 256 && counts[j].count)
117. {
118. **if**(!counts[j].p && counts[j].count < min2)
119. {
120. **if**(counts[j].count < min1)
121. {
122. min2 = min1;
123. min1 = counts[j].count;
124. b = a;
125. a = j;
126. }
127. **else**
128. {
129. min2 = counts[j].count;
130. b = j;
131. }
132. }
133. j++;
134. }
135. **if**(j < 256)
136. {
137. counts[a].p = j;
138. counts[b].p = j;
139. counts[j].l = a;
140. counts[j].r = b;
141. counts[j].count = min1 + min2;
142. //cout << j << ".  " << "l:" << counts[j].l << ",r:";
143. //cout << counts[j].r << ",count:" << counts[j].count <<endl;
144. }
145. }
147. **return**;
148. }
149. //创建Huffman树
150. **void** CountTimes()
151. {
152. **char** x;
154. sourceInput >> x;
155. cout<<"原文为："<<endl;
156. **while**(!sourceInput.eof())
157. {
158. counts[x].count++;
159. sourceInput >> x;
160. cout << x;
161. }
162. cout<<"\n\n"<<endl;
163. **for**(**int** i = 0;i < 128;i++)
164. {
165. **if**(counts[i].count)
166. {
167. num++;
168. }
169. **else**
170. {
171. counts[i].count = INF;
172. }
173. }
175. sourceInput.close();
176. sourceInput.open("source.txt",ios::in);
177. sourceInput >> noskipws;
178. //重新打开准备编码
180. **return**;
181. }
182. //计算字符出现的次数
183. **void** HaffmanEncode()
184. {
185. **for**(**int** i = 0;i < 128;i++) {
186. string temp;
187. **int** j = i;
188. **int** p = counts[i].p;
189. **while**(p) {
190. **if** (counts[p].l == j)temp.insert(0,"0");
191. **else** temp.insert(0,"1");
192. j = p;
193. p = counts[p].p;
194. }
195. codes.push\_back(temp);
196. }
197. **for**(**int** i = 0;i < 128;i++) {
198. **if**(counts[i].count == INF)
199. {
200. **continue**;
201. }
202. //cout << (char)i << ".  " << "val is " << counts[i].count << ", code is " << codes[i] << endl;
203. HuffmanMap << setw(10) << counts[i].key;
204. HuffmanMap << setw(10) << counts[i].count;
205. HuffmanMap << setw(10) << codes[i];
206. HuffmanMap << endl;
207. }
208. HuffmanMap.close();
209. HuffmanMap.open("Huffman.txt",ios::in);
210. //重新打开准备解码成文本
211. **char** x;
212. sourceInput >> x;
213. **while**(!sourceInput.eof())
214. {
215. //cout << "string:" << codes[x] << "\t\tbit:";
216. string s = codes[x];//字符串-》S
217. Huffmancode<<s;
218. //cout << "\t\tchar:" << x << endl;
219. sourceInput >> x;
220. }
222. //读取二进制形式的编码后的文章准备解码
223. Huffmancode.close();
224. **return**;
225. }
226. //进行Huffman编码并以字符串形式保存，写入Huffman文件和code文件
228. **void** HaffmanDecode()
229. {
230. Huffmancode.open("code.txt",ios::in );
231. //存储编码后的文章
232. Huffmancode << noskipws; //noskipws在输入文件时，不忽略换行符，如
233. **int** pad;
234. Huffmancode.clear();
235. string key;
236. **int** val;
237. HuffmanMap >> val;
238. cout<<"哈夫曼编码如下:  "<<endl;
239. cout<<"ASCII值   字符    次数    哈夫曼编码"<<endl;
240. **while**(!HuffmanMap.eof())
241. {
242. HuffmanMap >> pad; //次数
243. HuffmanMap >> key;
244. cout << val <<"        ";
245. cout << (**char**)val <<"        ";
246. cout<< pad<<"       ";
247. cout << key <<endl;
249. test[Sethash(key)].key = key;
250. test[Sethash(key)].val = val;
251. HuffmanMap >> val;
252. }
253. string m;
254. **char** n,x;
255. cout<<"\n\n解码后的字符集为："<<endl;
256. **while**(!Huffmancode.eof())
257. {
258. Huffmancode>>x;
259. m += x;
260. **if**(test[Sethash(m)].key.size())
261. {
263. n = test[Sethash(m)].val; //n为解码后的文章
264. decodeOutput << n;
265. cout<<n;
266. m = "";
267. }
268. }
269. cout<<endl;
270. **return**;
271. }
272. //通过Huffman编码规则解密编码后的文件
274. **int** main()
275. {
276. InitHaffman(); //打开文件
277. CountTimes(); //计算字符次数
278. CreateHaffman();
279. HaffmanEncode();
280. HaffmanDecode();
281. FiniHaffman(); //关闭文件
282. system("pause");
283. **return** 0;
284. }

# 五、行车路线

# 5.1 题目简介

行车路线 (必做)（图）

[问题描述]

　　小明和小芳出去乡村玩，小明负责开车，小芳来导航。

　　小芳将可能的道路分为大道和小道。大道比较好走，每走1公里小明会增加1的疲劳度。小道不好走，如果连续走小道，小明的疲劳值会快速增加，连续走s公里小明会增加s2的疲劳度。

　　例如：有5个路口，1号路口到2号路口为小道，2号路口到3号路口为小道，3号路口到4号路口为大道，4号路口到5号路口为小道，相邻路口之间的距离都是2公里。如果小明从1号路口到5号路口，则总疲劳值为(2+2)2+2+22=16+2+4=22。

现在小芳拿到了地图，请帮助她规划一个开车的路线，使得按这个路线开车小明的疲劳度最小。

[基本要求]

输入格式：

　　输入的第一行包含两个整数n, m，分别表示路口的数量和道路的数量。路口由1至n编号，小明需要开车从1号路口到n号路口。

接下来m行描述道路，每行包含四个整数t, a, b, c，表示一条类型为t，连接a与b两个路口，长度为c公里的双向道路。其中t为0表示大道，t为1表示小道。保证1号路口和n号路口是连通的。

输出格式

输出一个整数，表示最优路线下小明的疲劳度。

样例输入

6 7

1 1 2 3

1 2 3 2

0 1 3 30

0 3 4 20

0 4 5 30

1 3 5 6

1 5 6 1

样例输出

76

样例说明

　　从1走小道到2，再走小道到3，疲劳度为52=25；然后从3走大道经过4到达5，疲劳度为20+30=50；最后从5走小道到6，疲劳度为1。总共为76。

课程设计要求：

（1）要求从文本文件中输入；

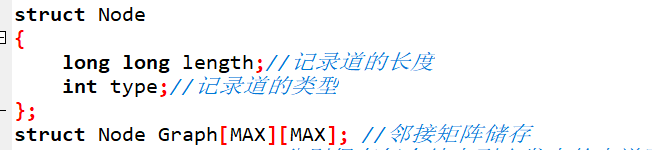
（2）采用适当的数据结构存储由输入数据中的道路所形成的图结构；

（3）编写尽可能优的算法，处理好连续走小道造成的疲劳值的指数增长（提示：基于迪杰斯特拉算法进行改进即可完成本题）；

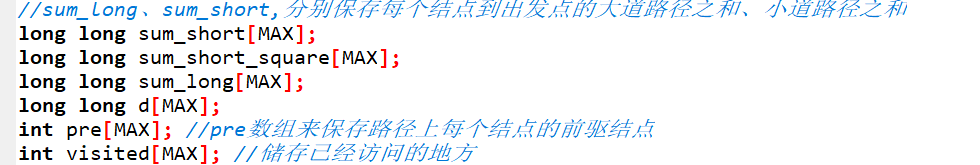
（4）除严格按题目要求进行输出以外，还要求输出最优路线的路径，以及从出发点到各个点的最小疲劳值。

# 5.2 数据结构

1、邻接矩阵储存图



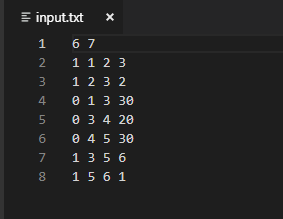
1. 另外有部分辅助数组

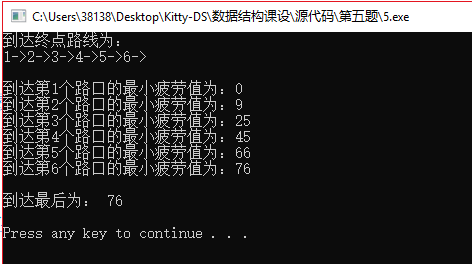


# 5.3 算法设计思想

直接用二维数组来构建图设置了两个数组d\_long、d\_short,分别保存每个结点到出发点的大道路径之和、小道路径之和。其次，更新点权的时候，需要考虑两个来源，一个来源就是上一个结点的d\_short值，一个来源就是上一个结点的d\_long值。在更新过程中要特别注意，会有多条连续的小道相连的情况，还有可能就是大道、小道相互交错的情况，所以要设置pre数组来保存路径上每个结点的前驱结点。

# 5.4 测试数据和结果





# 5.5 算法时间复杂度

由于dijsktra算法采用推优化后时间复杂度为O((m+n)logn)(用堆优化后)。本题未设置优化，故时间复杂度为O(n^2)

# 5.6 源代码(150

1. #include <stdio.h>
2. #include <string.h>
3. #include <stdlib.h>
4. #include <algorithm>
5. #include <iostream>
6. #include <fstream>
7. **using** **namespace** std;
8. #define MAX 999 //假设最多这么多站点
9. #define INF 9999999  //初始最初距离，表现为无穷大
10. **struct** Node
11. {
12. **long** **long** length;//记录道的长度
13. **int** type;//记录道的类型
14. };
15. **struct** Node Graph[MAX][MAX]; //邻接矩阵储存
16. //sum\_long、sum\_short,分别保存每个结点到出发点的大道路径之和、小道路径之和
17. **long** **long** sum\_short[MAX];
18. **long** **long** sum\_short\_square[MAX];
19. **long** **long** sum\_long[MAX];
20. **long** **long** d[MAX];
21. **int** pre[MAX]; //pre数组来保存路径上每个结点的前驱结点
22. **int** visited[MAX]; //储存已经访问的地方
23. **int** ans=INF;
24. **long** **long** square(**long** **long** x)
25. {
26. **return** x\*x;
27. }
28. **void** Dijkstra(**int** s,**int** n)
29. {
30. memset(visited,0,**sizeof**(visited));//初始化数据
31. fill(sum\_long,sum\_long+MAX,INF);
32. fill(sum\_short,sum\_short+MAX,INF);
33. fill(sum\_short\_square,sum\_short\_square+MAX,INF);
34. fill(pre,pre+MAX,0);
35. fill(d,d+MAX,INF);
36. d[s]=0;
37. sum\_long[s]=0;
38. sum\_short[s]=0;
39. **while**(**true**)
40. {
41. **int** u=-1;
42. **long** **long** mini=INF;//走到当前节点最小值
43. **for**(**int** i=1;i<=n;i++)
44. {
45. **if**(visited[i]==0&&d[i]<mini) //d[i]走到了i 就能赋值 否者break 相当于更新了mini,找到下一个点u
46. {
47. u=i;//遍历点 如果没走过就是u为（1，2，3，4，5，6）
48. mini=d[i];//记录当前的最小数 （0，9，25，45，66，76）
49. }
50. }
51. **if**(u==-1) **break**;
52. visited[u]=1; //此点走过
53. cout<<u<<"->";
54. **for**(**int** v=1;v<=n;v++) //到达u寻找v
55. {
56. **if**(Graph[u][v].length!=INF&&visited[v]==0) //有路 并且另外一个点没走过
57. {
58. **if**(Graph[u][v].type==1)//小路
59. {
60. **long** **long** d1=INF;
61. **long** **long** d2=INF;
62. **if**(sum\_short[u]!=INF)//如果有小道与源结点相连,即前面有小路
63. {
64. //需要处理一种特殊情况，就是大道、小道交错出现的情况
65. **if**(pre[u]==0) d1=square(sum\_short[u]+Graph[u][v].length);
66. **else** d1=sum\_long[pre[u]]+square(sum\_short[u]+Graph[u][v].length);//可能出现a通过大道与b相连，b通过小道与c相连
67. //c通过小道与d相连的情况
68. }
69. **if**(sum\_long[u]!=INF)//如果前面是大路
70. d2=square(Graph[u][v].length)+sum\_long[u];
71. **if**(d1<d2)
72. {
73. sum\_short[v]=min(sum\_short[v],sum\_short[u]+Graph[u][v].length);//更新结点对应的小道点权
74. sum\_short\_square[v]=min(d1,sum\_short\_square[v]);
75. }
76. **else**
77. {
78. sum\_short[v]=min(sum\_short[v],Graph[u][v].length);//由于上一条道路是大道，所以这里需要从头开始考虑小道的点权
79. sum\_short\_square[v]=min(d2,sum\_short\_square[v]);
80. pre[v]=u;//记录当前结点的前驱结点
81. }
82. d[v]=min(sum\_short\_square[v],sum\_long[v]);
83. }
84. **else**//当前道路是大道
85. {
87. **long** **long** d1=INF;
88. **long** **long** d2=INF;
89. **if**(sum\_long[u]!=INF)  d1=sum\_long[u]+Graph[u][v].length;
90. **if**(sum\_short[u]!=INF) d2=sum\_short\_square[u]+Graph[u][v].length;
91. **long** **long** mini=min(d1,d2);
92. sum\_long[v]=min(sum\_long[v],mini);
93. d[v]=min(sum\_short\_square[v],sum\_long[v]);
94. }
95. }
96. }
97. }
98. **return**;
99. }
101. **int** main()
102. {
103. **int** n,m;
104. **int** t,a,b;
105. **long** **long** c;
106. ifstream infile("input.txt");
107. //scanf("%d %d",&n,&m);
108. infile>>n>>m;
109. **for**(**int** i=0;i<=n;i++)
110. {
111. **for**(**int** j=0;j<=n;j++)
112. {
113. Graph[i][j].length=INF;
114. Graph[j][i].length=INF;
115. }
116. }
117. **for**(**int** i=0;i<m;i++)
118. {
119. infile>>t>>a>>b>>c;
120. //scanf("%d%d%d%lld",&t,&a,&b,&c);
121. **if**(t==1)//小道
122. {
123. Graph[a][b].length=c;
124. Graph[b][a].length=c;
125. Graph[a][b].type=1;
126. Graph[b][a].type=1;
127. }
128. **else**
129. {
130. Graph[a][b].length=c;
131. Graph[b][a].length=c;
132. Graph[a][b].type=0;
133. Graph[b][a].type=0;
134. }
135. }
136. infile.close();
137. cout<<"到达终点路线为：\n";
138. Dijkstra(1,n);
139. cout<<"\n"<<endl;
140. **for**(**int** i=1;i<=n;i++)
141. cout<<"到达第"<<i<<"个路口的最小疲劳值为："<<d[i]<<endl;
142. cout<<endl;
143. printf("到达最后为： %lld\n",d[n]);
144. cout<<endl;
145. system("pause");
146. **return** 0;
147. }

# 六、排序算法比较

# 6.1 题目简介

排序算法比较 （必做）（排序）

[问题描述]

利用随机函数产生10个样本，每个样本有20000个随机整数（并使第一个样本是正序，第二个样本是逆序），利用直接插入排序、希尔排序，冒泡排序、快速排序、选择排序、堆排序，归并排序、基数排序8种排序方法进行排序（结果为由小到大的顺序），并统计每一种排序算法对不同样本所耗费的时间。

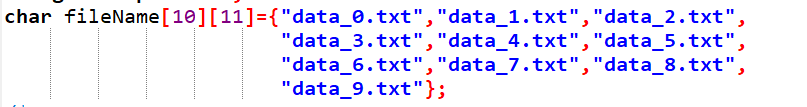
[基本要求]

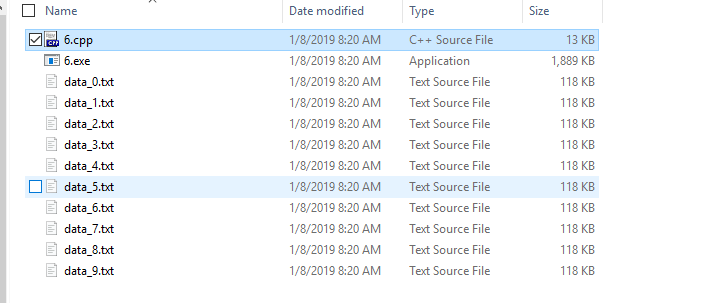
（1） 原始数据存在文件中，用相同样本对不同算法进行测试；

（2） 屏幕显示每种排序算法对不同样本所花的时间；

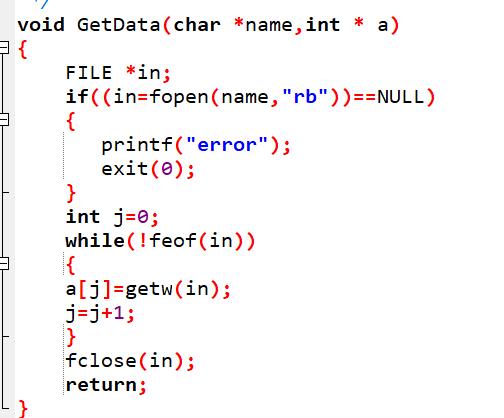
# 6.2 数据结构

1、采用十个文件二进制存储10个样本





1. 通过GetData获取内部数据

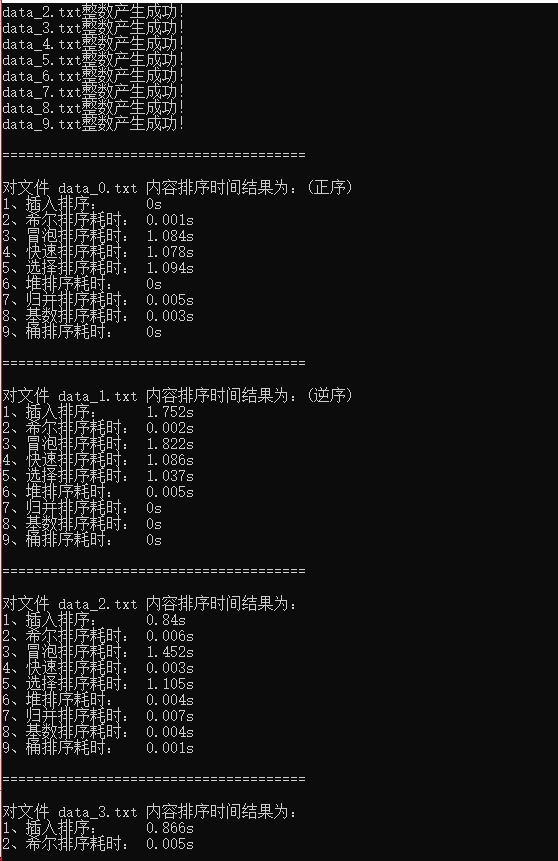


# 6.3 算法设计思想

先编写生成数据和获取数据样本，然后分别编写九大排序，在每个排序内部使用clock\_t start

和finish分别计算开始和结束时间，最后相减获取程序跑的时间，注意就是每次排序后要重新拿到数据。

# 6.4 测试数据和结果，



# 6.5 算法时间复杂度



# 6.6 源代码(600

1. /\*\*
2. \* 九大排序：C++
3. \*
4. \* @author Kitty
5. \* @date 2019/1/8
6. \*/
7. #include<iostream>
8. #include<cstring>
9. #include<cstdlib>
10. #include<fstream>
11. #include<time.h>
12. #define N 30000
13. **clock\_t** start,finish;
14. **using** **namespace** std;
15. **char** fileName[10][11]={"data\_0.txt","data\_1.txt","data\_2.txt",
16. "data\_3.txt","data\_4.txt","data\_5.txt",
17. "data\_6.txt","data\_7.txt","data\_8.txt",
18. "data\_9.txt"};
19. /\*
20. \* 1、插入排序
21. \*
22. \* 参数说明：
23. \*     a -- 待排序的数组
24. \*     n -- 数组的长度
25. \*/
26. **void** insertSort(**int** \*a,**int** n)
27. {
28. start=clock();
29. **int** i,j,k;
30. **for**(i=1;i<n;i++)
31. {
32. //为a[i]在前面的a[0...i-1]有序区间中找一个合适的位置
33. **for**(j=i-1;j>=0;j--)
34. {
35. **if**(a[j]<a[i])
36. **break**;
37. }
38. **if**(j!=i-1)
39. {
40. //比a[i]大的后移
41. **int** temp=a[i];
42. **for**(k=i-1;k>j;k--)
43. a[k+1]=a[k];
44. //将a[i]放到正确位置上
45. a[k+1]=temp;
46. }
47. }
48. finish=clock();
49. **double** Time=(**double**)((finish-start))/CLOCKS\_PER\_SEC;//计算时间
50. cout<<"1、插入排序：     "<<Time<<"s\n";
51. }
52. /\*
53. \* 2、希尔排序
54. \*
55. \* 参数说明：
56. \*     a -- 待排序的数组
57. \*     n -- 数组的长度
58. \*/
59. **void** shell\_sort(**int** \*a, **int** n)
60. {
61. start=clock();
62. **int** i,j,gap,temp;
63. // gap为步长，每次减为原来的一半。
64. **for**(gap=n/2;gap>0;gap/=2)
65. {
66. // 共gap个组，对每一组都执行直接插入排序
67. **for**(**int** i=0;i<gap;i++)
68. {
69. **for**(j=i+gap;j<n;j+=gap)
70. {
71. // 如果a[j] < a[j-gap]，则寻找a[j]位置，并将后面数据的位置都后移。
72. **if**(a[j-gap]>a[j])
73. {
74. temp=a[j];
75. **int** k=j-gap;
76. **while**(k>=0&&a[k]>temp)
77. {
78. a[k+gap]=a[k];
79. k-=gap;
80. }
81. a[k+gap]=temp;
82. }
83. }
84. }
85. }
86. finish=clock();
87. **double** Time=(**double**)((finish-start))/CLOCKS\_PER\_SEC;//计算时间
88. cout<<"2、希尔排序耗时： "<<Time<<"s\n";
89. }
90. /\*
91. \* 3、冒泡排序
92. \*
93. \* 参数说明：
94. \*     a -- 待排序的数组
95. \*     n -- 数组的长度
96. \*/
97. **void** bubbleSort(**int** \*a,**int** n)
98. {
99. start=clock();
100. **int** i,j,temp=0;
101. **for**(i=0;i<n;i++)
102. {
103. // 将最大的数据放在末尾
104. **for**(j=n-i-1;j<n-1;j++)
105. {
106. //n-i-1可以直接用0代替，本代码更优化
107. **if**(a[j]>a[j+1])
108. {
109. // 交换a[j]和a[j+1]
110. temp=a[j];
111. a[j]=a[j+1];
112. a[j+1]=temp;
113. }
114. }
115. }
116. finish=clock();
117. **double** Time=(**double**)((finish-start))/CLOCKS\_PER\_SEC;//计算时间
118. cout<<"3、冒泡排序耗时： "<<Time<<"s\n";
119. }
120. /\*
121. \* 4、快速排序
122. \*
123. \* 参数说明：
124. \*     a -- 待排序的数组
125. \*     l -- 数组的左边界(例如，从起始位置开始排序，则l=0)
126. \*     r -- 数组的右边界(例如，排序截至到数组末尾，则r=a.length-1)
127. \*/
128. **void** quickSort(**int** \*a, **int** l, **int** r)
129. {
130. **if** (l < r)
131. {
132. **int** i,j,x;
134. i = l;
135. j = r;
136. x = a[i];
137. **while** (i < j)
138. {
139. **while**(i < j && a[j] > x)
140. j--; // 从右向左找第一个小于x的数
141. **if**(i < j)
142. {
143. a[i]=a[j];
144. a[j]=x;
145. i++;
146. }
147. **while**(i < j && a[i] < x)
148. i++; // 从左向右找第一个大于x的数
149. **if**(i < j)
150. {
151. a[j]=a[i];
152. a[i]=x;
153. j--;
154. }
155. }
156. a[i] = x;
157. quickSort(a, l, i-1); /\* 递归调用 \*/
158. quickSort(a, i+1, r); /\* 递归调用 \*/
159. }
160. }
161. /\*
162. \* 5、选择排序
163. \*
164. \* 参数说明：
165. \*     a -- 待排序的数组
166. \*     n -- 数组的长度
167. \*/
168. **void** selectSort(**int**\* a, **int** n)
169. {
170. start=clock();
171. **int** i;      // 有序区的末尾位置
172. **int** j;      // 无序区的起始位置
173. **int** min;    // 无序区中最小元素位置
175. **for**(i=0; i<n; i++)
176. {
177. min=i;
179. // 找出"a[i+1] ... a[n]"之间的最小元素，并赋值给min。
180. **for**(j=i+1; j<n; j++)
181. {
182. **if**(a[j] < a[min])
183. min=j;
184. }
186. // 若min!=i，则交换 a[i] 和 a[min]。
187. // 交换之后，保证了a[0] ... a[i] 之间的元素是有序的。
188. **if**(min != i)
189. {
190. **int** tmp = a[i];
191. a[i] = a[min];
192. a[min] = tmp;
193. }
194. }
195. finish=clock();
196. **double** Time=(**double**)((finish-start))/CLOCKS\_PER\_SEC;//计算时间
197. cout<<"5、选择排序耗时： "<<Time<<"s\n";
198. }
199. /\*
200. \* 6、堆排序
201. \* 6.1、(最大)堆的向下调整算法
202. \*
203. \* 注：数组实现的堆中，第N个节点的左孩子的索引值是(2N+1)，右孩子的索引是(2N+2)。
204. \*     其中，N为数组下标索引值，如数组中第1个数对应的N为0。
205. \*
206. \* 参数说明：
207. \*     a -- 待排序的数组
208. \*     start -- 被下调节点的起始位置(一般为0，表示从第1个开始)
209. \*     end   -- 截至范围(一般为数组中最后一个元素的索引)
210. \*/
211. **void** maxHeapDown(**int**\* a, **int** start, **int** end)
212. {
213. **int** c = start;          // 当前(current)节点的位置
214. **int** l = 2\*c + 1;        // 左(left)孩子的位置
215. **int** tmp = a[c];         // 当前(current)节点的大小
216. **for** (; l <= end; c=l,l=2\*l+1)
217. {
218. // "l"是左孩子，"l+1"是右孩子
219. **if** ( l < end && a[l] < a[l+1])
220. l++;        // 左右两孩子中选择较大者，即m\_heap[l+1]
221. **if** (tmp >= a[l])
222. **break**;      // 调整结束
223. **else**            // 交换值
224. {
225. a[c] = a[l];
226. a[l]= tmp;
227. }
228. }
229. }
230. /\*
231. \* 6.2、堆排序(mian)
232. \*
233. \* 参数说明：
234. \*     a -- 待排序的数组
235. \*     n -- 数组的长度
236. \*/
237. **void** heapSort(**int**\* a, **int** n)
238. {
239. **int** i,tmp;
241. // 从(n/2-1) --> 0逐次遍历。遍历之后，得到的数组实际上是一个(最大)二叉堆。
242. **for** (i = n / 2 - 1; i >= 0; i--)
243. maxHeapDown(a, i, n-1);
245. // 从最后一个元素开始对序列进行调整，不断的缩小调整的范围直到第一个元素
246. **for** (i = n - 1; i > 0; i--)
247. {
248. // 交换a[0]和a[i]。交换后，a[i]是a[0...i]中最大的。
249. tmp = a[0];
250. a[0] = a[i];
251. a[i] = tmp;
252. // 调整a[0...i-1]，使得a[0...i-1]仍然是一个最大堆。
253. // 即，保证a[i-1]是a[0...i-1]中的最大值。
254. maxHeapDown(a, 0, i-1);
255. }
256. }
258. /\*
259. \* 7、归并排序
260. \* 7.1、将一个数组中的两个相邻有序区间合并成一个
261. \*
262. \* 参数说明：
263. \*     a -- 包含两个有序区间的数组
264. \*     start -- 第1个有序区间的起始地址。
265. \*     mid   -- 第1个有序区间的结束地址。也是第2个有序区间的起始地址。
266. \*     end   -- 第2个有序区间的结束地址。
267. \*/
268. **void** merge(**int**\* a, **int** start, **int** mid, **int** end)
269. {
270. **int** \*tmp = **new** **int**[end-start+1];    // tmp是汇总2个有序区的临时区域
271. **int** i = start;          // 第1个有序区的索引
272. **int** j = mid + 1;        // 第2个有序区的索引
273. **int** k = 0;              // 临时区域的索引
275. **while**(i <= mid && j <= end)
276. {
277. **if** (a[i] <= a[j])
278. tmp[k++] = a[i++];
279. **else**
280. tmp[k++] = a[j++];
281. }
283. **while**(i <= mid)
284. tmp[k++] = a[i++];
286. **while**(j <= end)
287. tmp[k++] = a[j++];
289. // 将排序后的元素，全部都整合到数组a中。
290. **for** (i = 0; i < k; i++)
291. a[start + i] = tmp[i];
293. **delete**[] tmp;
294. }
295. /\*
296. \* 7.2、归并排序(main)
297. \*
298. \* 参数说明：
299. \*     a -- 待排序的数组
300. \*     start -- 数组的起始地址
301. \*     end -- 数组的结束地址,初始为n-1
302. \*/
303. **void** mergeSort(**int**\* a, **int** start, **int** end)
304. {
305. **if**(a==NULL || start >= end)
306. **return** ;
308. **int** mid = (end + start)/2;
309. mergeSort(a, start, mid); // 递归排序a[start...mid]
310. mergeSort(a, mid+1, end); // 递归排序a[mid+1...end]
312. // a[start...mid] 和 a[mid...end]是两个有序空间，
313. // 将它们排序成一个有序空间a[start...end]
314. merge(a, start, mid, end);
315. }
316. /\*
317. \* 8、基数排序
318. \*/
319. /\*
320. \* 8.1、获取数组a中最大值
321. \*
322. \* 参数说明：
323. \*     a -- 数组
324. \*     n -- 数组长度
325. \*/
326. **int** getMax(**int** a[], **int** n)
327. {
328. **int** i, max;
330. max = a[0];
331. **for** (i = 1; i < n; i++)
332. **if** (a[i] > max)
333. max = a[i];
334. **return** max;
335. }
336. /\*
337. \* 8.2、对数组按照"某个位数"进行排序(桶排序)
338. \*
339. \* 参数说明：
340. \*     a -- 数组
341. \*     n -- 数组长度
342. \*     exp -- 指数。对数组a按照该指数进行排序。
343. \*
344. \* 例如，对于数组a={50, 3, 542, 745, 2014, 154, 63, 616}；
345. \*    (01) 当exp=1表示按照"个位"对数组a进行排序
346. \*    (02) 当exp=10表示按照"十位"对数组a进行排序
347. \*    (03) 当exp=100表示按照"百位"对数组a进行排序
348. \*    ...
349. \*/
350. **void** countSort(**int** \*a, **int** n, **int** exp)
351. {
352. **int** output[n];          // 存储"被排序数据"的临时数组
353. **int** i, buckets[10] = {0};
355. // 将数据出现的次数存储在buckets[]中
356. **for** (i = 0; i < n; i++)
357. buckets[ (a[i]/exp)%10 ]++;
359. // 更改buckets[i]。目的是让更改后的buckets[i]的值，是该数据在output[]中的位置。
360. **for** (i = 1; i < 10; i++)
361. buckets[i] += buckets[i - 1];
363. // 将数据存储到临时数组output[]中
364. **for** (i = n - 1; i >= 0; i--)
365. {
366. output[buckets[ (a[i]/exp)%10 ] - 1] = a[i];
367. buckets[ (a[i]/exp)%10 ]--;
368. }
370. // 将排序好的数据赋值给a[]
371. **for** (i = 0; i < n; i++)
372. a[i] = output[i];
373. }
374. /\*
375. \* 8.3、基数排序
376. \*
377. \* 参数说明：
378. \*     a -- 数组
379. \*     n -- 数组长度
380. \*/
381. **void** radixSort(**int** \*a, **int** n)
382. {
383. start=clock();
384. **int** exp;    // 指数。当对数组按各位进行排序时，exp=1；按十位进行排序时，exp=10；...
385. **int** max = getMax(a, n); // 数组a中的最大值
387. // 从个位开始，对数组a按"指数"进行排序
388. **for** (exp = 1; max/exp > 0; exp \*= 10)
389. countSort(a, n, exp);
390. finish=clock();
391. **double** Time=(**double**)((finish-start))/CLOCKS\_PER\_SEC;//计算时间
392. cout<<"8、基数排序耗时： "<<Time<<"s\n";
393. }
394. /\*
395. \* 9、(附加)桶排序排序
396. \*/
397. /\*
398. \* 9.1、获取数组a中最大值
399. \*
400. \* 参数说明：
401. \*     a -- 数组
402. \*     n -- 数组长度
403. \*/
404. **int** getMaxforbucket(**int** \*a, **int** n)
405. {
406. **int** i, max;
408. max = a[0];
409. **for** (i = 1; i < n; i++)
410. **if** (a[i] > max)
411. max = a[i];
412. **return** max;
413. }
414. /\*
415. \* 9.2、桶排序
416. \*
417. \* 参数说明：
418. \*     a -- 待排序数组
419. \*     n -- 数组a的长度
420. \*/
421. **void** bucketSort(**int**\* a, **int** n)
422. {
423. **int** max=getMaxforbucket(a,n)+1;//获取最大值
424. **int** i, j;
425. **int** \*buckets;
427. **if** (a==NULL || n<1 || max<1) //异常检验
428. **return** ;
430. // 创建一个容量为max的数组buckets，并且将buckets中的所有数据都初始化为0。
431. **if** ((buckets = **new** **int**[max])==NULL)
432. **return** ;
433. memset(buckets, 0, max\***sizeof**(**int**));
435. // 1. 计数
436. **for**(i = 0; i < n; i++)
437. buckets[a[i]]++;
439. // 2. 排序
440. **for** (i = 0, j = 0; i < max; i++)
441. **while**( (buckets[i]--) >0 )
442. a[j++] = i;
443. **delete**[] buckets;
444. }
445. /\*
446. \*产生样本
447. \*二进制储存数
448. \*随机函数产生10个样本每个样本有20000个随机整数?
449. \*并使第一个样本是正序，第二个样本是逆序
450. \*
451. \*/
452. **void** PutData()
453. {
455. **for**(**int** i=0;i<10;i++)
456. {
457. **if**(i==0) //正序
458. {
459. **FILE** \*out;
460. **if**((out=fopen(fileName[i],"wb"))==NULL)
461. {
462. printf("Open error.\n");
463. **return**;
464. }
465. **for**(**int** j=0;j<N;j++)
466. {
467. putw(j,out);
468. }
469. fclose(out);
470. cout<<fileName[i]<<"整数产生成功！"<<endl;
471. }
472. **else** **if**(i==1) //逆序
473. {
474. **FILE** \*out;
475. **if**((out=fopen(fileName[i],"wb"))==NULL)
476. {
477. printf("Open error.\n");
478. **return**;
479. }
480. **for**(**int** j=0;j<N;j++)
481. {
482. **int** num=N-j;
483. putw(num,out);
484. }
485. fclose(out);
486. cout<<fileName[i]<<"整数产生成功！"<<endl;
487. }
488. **else**{
489. **FILE** \*out;
490. **if**((out=fopen(fileName[i],"wb"))==NULL)
491. {
492. printf("Open error.\n");
493. **return**;
494. }
495. srand((unsigned)time(0)); //产生随机时间种子
496. **for**(**int** j=0;j<N;j++)
497. {
498. **int** ran\_num=rand()%N;
499. putw(ran\_num,out);
500. }
501. fclose(out);
502. cout<<fileName[i]<<"整数产生成功！"<<endl;
504. }
505. }
506. cout<<"\n======================================\n"<<endl;
507. }
508. /\*
509. \* 帮助数组从文件中读取获得数据
510. \*参数说明：
511. \*     name -- 待读取文件名
512. \*     a --  读取赋值的目标数组
513. \*/
514. **void** GetData(**char** \*name,**int** \* a)
515. {
516. **FILE** \*in;
517. **if**((in=fopen(name,"rb"))==NULL)
518. {
519. printf("error");
520. exit(0);
521. }
522. **int** j=0;
523. **while**(!feof(in))
524. {
525. a[j]=getw(in);
526. j=j+1;
527. }
528. fclose(in);
529. **return**;
530. }
531. **int** main()
532. {
533. **double** Time;
534. **int** a[N+1];
535. PutData();
536. **for**(**int** i=0;i<10;i++)
537. {
538. cout<<"对文件 "<<fileName[i]<<" 内容排序时间结果为：";
539. **if**(i==0) cout<<"(正序)"<<endl;
540. **else** **if**(i==1) cout<<"(逆序)"<<endl;
541. **else** cout<<endl;
543. GetData(fileName[i],a);     //每次排序后从新拿到数据
544. insertSort(a,N);
546. GetData(fileName[i],a);
547. shell\_sort(a,N);
549. GetData(fileName[i],a);
550. bubbleSort(a,N);
552. GetData(fileName[i],a);
553. start=clock();
554. quickSort(a,0,N-1);
555. finish=clock();
556. Time=(**double**)((finish-start))/CLOCKS\_PER\_SEC;//计算时间
557. cout<<"4、快速排序耗时： "<<Time<<"s\n";
559. GetData(fileName[i],a);
560. selectSort(a,N);
562. GetData(fileName[i],a);
563. start=clock();
564. heapSort(a,N);
565. finish=clock();
566. Time=(**double**)((finish-start))/CLOCKS\_PER\_SEC;//计算时间
567. cout<<"6、堆排序耗时：   "<<Time<<"s\n";
569. GetData(fileName[i],a);
570. start=clock();
571. mergeSort(a,0,N-1);
572. finish=clock();
573. Time=(**double**)((finish-start))/CLOCKS\_PER\_SEC;//计算时间
574. cout<<"7、归并排序耗时： "<<Time<<"s\n";
576. GetData(fileName[i],a);
577. radixSort(a,N);
579. GetData(fileName[i],a);
580. start=clock();
581. bucketSort(a,N);
582. finish=clock();
583. **double** Time=(**double**)((finish-start))/CLOCKS\_PER\_SEC;//计算时间
584. cout<<"9、桶排序耗时：   "<<Time<<"s\n";
585. cout<<"\n======================================\n"<<endl;
586. }
587. system("pause");
588. }

# 七、朋友圈

# 7.1 题目简介

[问题描述]

某学校有N个学生，形成M个俱乐部。每个俱乐部里的学生有着相似的兴趣爱好，

形成一个朋友圈。一个学生可以同时属于若干个不同的俱乐部。

根据“我的朋友的朋友也是我的朋友”这个推论可以得出，如果A和B是朋友，且B和C是朋友，则A和C也是朋友。

请编写程序计算最大朋友圈中有多少人。

[基本要求]

（1）输入说明：输入的第一行包含两个正整数N (N<=30 000)和M (M<=1000)，

分别代表学校的学生总数和俱乐部的个数。随后的M行每行按以下格式给出一个俱乐部的信息，其中学生从1-N编号：

第i个俱乐部的人数Mi（空格）学生1（空格）学生2… 学生Mi

（2）输出说明：输出一个整数，表示在最大朋友圈中有多少人。

（3）测试用例：

输入

7 4

3 1 2 3

2 1 4

3 5 6 7

1 6

输出

4

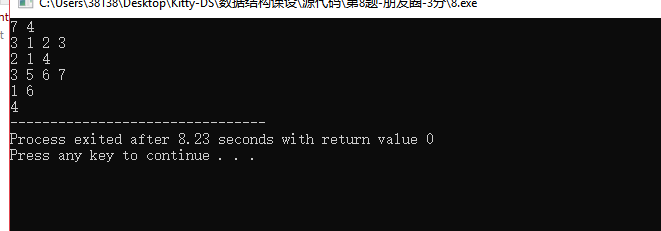
# 7.2 数据结构

采用并查集快速求解

# 7.3 算法设计思想

通过并查集，寻找最终每个圈中人数，找出最大的那个

7.4 测试数据和结果，



# 7.5 算法时间复杂度

（m\*n)/2 不是很确定

# 7.6 源代码(51

1. #include<cstdio>
2. #include<cstdlib>
4. **void** Connect(**int** \*arr, **int** a, **int** b) {
5. **int** i=a;
6. **while** (arr[i] && arr[i] != a) {
7. i = arr[i];
8. }
10. **int** j=b;
11. **while** (arr[j] && arr[j] != b) {
12. j = arr[j];
13. }
15. arr[i] = b;
16. arr[j] = a;
17. }
19. **int** main() {
21. **int** n=0, m=0;
22. scanf("%d%d", &n, &m);
23. **int** \*stu = (**int**\*)malloc(**sizeof**(**int**) \* (n+1));
24. **for** (**int** i=0; i<=n; i++) stu[i] = 0;
25. **for** (**int** i=0; i<m; i++) {
26. **int** count=0;
27. scanf("%d", &count);
28. **int** \*club = (**int**\*)malloc(**sizeof**(**int**) \* count);
29. **for** (**int** j=0; j<count; j++) {
30. scanf("%d", &club[j]);
31. }
32. **for** (**int** j=1; j<count; j++) {
33. Connect(stu, club[j-1], club[j]);
34. }
35. free(club);
36. }
38. **int** max=0;
39. **for** (**int** i=1; i<=n; i++) {
40. **int** count=1;
41. **int** j=i;
42. **while** (stu[j] && stu[j] != i) {
43. j = stu[j];
44. count++;
45. }
46. **if** (count > max) max = count;
47. }
48. printf("%d", max);
50. free(stu);
51. }

# 八、迷宫

# 8.1 题目简介

迷宫问题（选做）（栈与队列），（深度搜索，广度搜索）

[问题描述]

利用栈操作实现迷宫问题求解。

[基本要求]

（1）从文件中读取数据，生成模拟迷宫地图，不少于10行10列。

（2）给出任意入口和出口，显示输出迷宫路线。

# 8.2 数据结构

1. **typedef** **struct**
2. {
3. **int** m,n;
4. **int** row;
5. **int** col;
6. **char** arr[RANGE][RANGE];
7. }Maze;
8. **typedef** **struct**
9. {
10. **int** x,y;
11. **int** direction
12. }Position;

# 8.3 算法设计思想

本题有2种方法解决，栈与递归

栈：

将迷宫转化为一个int型矩阵，若有障碍则设为1，没有障碍设为0

在搜索过程中的某一时刻所在图中某个方块位置”，则求迷宫中一条路径的算法的基本思想是：

若当前位置“可通”，则纳入“当前路径”，并继续朝“下一位置”探索，即切换“下一位置”为“当前位置”，如此重复直至到达出口；

若当前位置“不可通”，则应顺着“来向”退回到“前一通道块”，然后朝着除“来向”之外的其他方向继续探索；

若该通道块的四周4个方块均“不可通”，则应从“当前路径”上删除该通道块。谓“下一位置”指的是当前位置四周4个方向（东、南、西、北）上相邻的方块。假设以栈S记录“当前路径”，则栈顶中存放的是“当前路径上最后一个通道块”。

由此，“纳入路径”的操作即为“当前位置入栈”；“从当前路径上删除前一通道块”的操作即为“出栈”。

递归：

将迷宫转化为一个int型矩阵，若有障碍则设为1，没有障碍设为0

在搜索过程中的某一时刻所在图中某个方块位置”，则求迷宫中一条路径的算法的基本思想是：

若当前位置“可通”，则纳入“当前路径”，并继续朝“下一位置”探索，即切换“下一位置”为“当前位置”，如此重复直至到达出口；

若当前位置“不可通”，则应顺着“来向”退回到“前一通道块”，然后朝着除“来向”之外的其他方向继续探索；

若该通道块的四周4个方块均“不可通”，则应从“当前路径”上删除该通道块。

所谓“下一位置”指的是当前位置四周4个方向（东、南、西、北）上相邻的方块。若该位置可通返回true，若不可通，返回false；

8.4 测试数据和结果，



# 8.5 算法时间复杂度

寻找路径的函数时间复杂度最差情况（未找到路径），将全图的节点都访问完，所以时间复杂度为o(n)。n为节点数。

# 8.6 源代码(300

Main.cpp

1. #include"Maze.h" //函数声明
2. #include"Maze.cpp" //函数实现
3. #include"Position.h" //点的坐标
4. #include"Stack.h" //自己实现一个栈
5. #include<iostream>
6. **using** **namespace** std;
7. **int** main()
8. {
9. Maze maze;
10. Stack S;
11. Position start;
12. Position end;
13. start.x = 1;
14. start.y = 1;
15. end.x = 10;
16. end.y = 10;
17. InitMaze(maze);
18. cout << "原迷宫：" << endl << endl;
19. PrintMaze(maze);
20. cout  << endl << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
21. cout << "\*   请输入起点：   \*" << endl;
22. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;
23. cout << "x:" ;
24. cin >> start.x;
25. cout << "y:" ;
26. cin >> start.y;
27. cout  << endl << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
28. cout << "\*   请输入终点：   \*" << endl;
29. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;
30. cout << "x:" ;
31. cin >> end.x;
32. cout << "y:" ;
33. cin >> end.y;
34. **if**(MazePath(maze,start,end,S))
35. {
36. cout << endl << endl << "迷宫的解：(\*代表路线)" << endl << endl;
37. PrintMaze(maze);
38. }
39. **else**
40. {
41. cout << endl << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
42. cout << "\*   此迷宫无解！   \*" << endl;
43. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;
44. }
45. **return** 0;
46. }

Maze.cpp

1. #include"Maze.h"
2. #include"Stack.h"
3. #include"Stack.cpp"
4. #include<iostream>
5. #include<cstdio>
6. #include<cstdlib>
7. #include<fstream>
8. #include<iomanip>
9. #define ERROR 0
10. #define OK 1
11. #define FALSE 0
12. #define TRUE 1
14. **bool** mark[RANGE][RANGE];
15. **using** **namespace** std;
16. **bool** InitMaze(Maze &maze)
17. {
18. ifstream infile("data.txt",ios::in);
19. **if**(!infile.is\_open())
20. {
21. cout << "文件不能打开！" <<endl;
22. exit(ERROR);
23. }
24. **int** x = 0;
25. **int** row = 0;
26. **int** col = 0;
27. infile >> row;
28. infile >> col;
29. maze.row = row;
30. maze.col = col;
31. **for**(**int** i = 1;i <= row;i++)
32. **for**(**int** j = 1;j <= col;j++)
33. maze.arr[i][j] = ' ';
34. **for**(**int** i = 0;i <= row+1;i++)
35. {
36. maze.arr[i][0] = 30;
37. maze.arr[i][col+1] = 30;
38. }
39. **for**(**int** i = 0;i <= col+1;i++)
40. {
41. maze.arr[0][i] = 30;
42. maze.arr[row+1][i] = 30;
43. }
44. **for**(**int** i = 1;i <= row;i++)
45. {
46. **for**(**int** j = 1;j <= col;j++)
47. {
48. infile >> x;
49. **if**(x == 1)
50. {
51. maze.arr[i][j] = 30;
52. }
53. }
54. }
55. infile.close();
56. }
58. **bool** nextpos(Position &a,Position &b,**int** row,**int** col)
59. {
60. a.x = -1;
61. **if**(b.y+1 <= col && mark[b.x][b.y+1] == FALSE)//右
62. {
63. a.x = b.x;
64. a.y = b.y+1;
65. **return** OK;
66. }
67. **if**(b.x+1 <= row && mark[b.x+1][b.y] == FALSE)//下
68. {
69. a.x = b.x+1;
70. a.y = b.y;
71. **return** OK;
72. }

75. **if**(b.y-1 > 0 && mark[b.x][b.y-1] == FALSE)//左
76. {
77. a.x = b.x;
78. a.y = b.y-1;
79. **return** OK;
80. }
81. **if**(b.x-1 > 0 && mark[b.x-1][b.y] == FALSE)//上
82. {
83. a.x = b.x-1;
84. a.y = b.y;
85. **return** OK;
86. }
88. }
90. **bool** MazePath(Maze &maze,Position start,Position end,Stack S)
91. {
92. InitStack(S);
93. Position now = start;
94. Position e;
95. **bool** found = FALSE;
96. **for**(**int** i = 0;i <= maze.row+1;i++)
97. **for**(**int** j = 0;j <= maze.col+1;j++)
98. {
99. **if**(maze.arr[i][j] == 30)
100. mark[i][j] = TRUE;
101. **else**
102. mark[i][j] = FALSE;
103. }//初始化mark数组，0表示未访问过，1表示是墙，2表示留下了足迹
104. PushStack(S,now);
105. mark[now.x][now.y] = 2;//留下足迹
106. **while**(S.nodenum != 0 && !(now.x == end.x && now.y == end.y))
107. {
108. PopStack(S,e);
109. PushStack(S,e);//取得栈顶元素
110. nextpos(now,e,maze.row,maze.col);//找下一个位置，若都被访问过，将now.x赋值为-1
111. **if**(now.x == -1)
112. {
113. PopStack(S,e);//回溯到上一个节点
114. **continue**;
115. }
116. mark[now.x][now.y] = 2;//留下足迹
117. PushStack(S,now);
118. }
119. **if**(now.x == end.x && now.y == end.y)
120. found = TRUE;
121. **else**
122. found = FALSE;
123. **while**(S.nodenum != 0)
124. {
125. PopStack(S,e);
126. maze.arr[e.x][e.y] = '\*';
127. }//栈中的元素即为路径输入到maze中
128. StackTraverse(S);
129. **return** found;
130. }
132. **void** PrintMaze(Maze &maze)
133. {
134. **for**(**int** i = 0;i <= maze.row+1;i++)
135. {
136. **for**(**int** j = 0;j <= maze.col+1;j++)
137. cout << maze.arr[i][j] << " ";
138. cout << endl;
139. }
140. }

Maze.h

1. #ifndef MAZE\_H\_INCLUDED
2. #define MAZE\_H\_INCLUDED
3. #define RANGE 1002
4. #include"Position.h"
5. **typedef** **struct**
6. {
7. **int** m,n;
8. **int** row;
9. **int** col;
10. **char** arr[RANGE][RANGE];
11. }Maze;
13. **bool** InitMaze(Maze &maze,**int** row,**int** col);
14. **bool** MazePath(Maze &maze,Position start,Position end);//入口start到出口end的一条路径
15. **void** PrintMaze(Maze &maze);

18. #endif // MAZE\_H\_INCLUDED

Position.h

1. #ifndef POSITION\_H\_INCLUDED
2. #define POSITION\_H\_INCLUDED
3. **typedef** **struct**
4. {
5. **int** x,y;
6. **int** direction;//
7. }Position;

10. #endif // POSITION\_H\_INCLUDED

# Stack.cpp

1. #include"Stack.h"
2. #include"Position.h" //存放方向
3. #include<iostream>
4. #include<cstdio>
5. #include<cstdlib>
6. #define OK 1
7. #define ERROR 0
8. #define OVERFLOW 0
9. #define etype Position
10. **using** **namespace** std;
12. **bool** InitStack(Stack &S)
13. {
14. S.top = (SNode \*)malloc(**sizeof**(SNode));
15. S.base = S.top;
16. **if**(!S.top)
17. {
18. cout << "OVERFLOW!" << endl;
19. exit(OVERFLOW);
20. }
21. S.top->next = NULL;
22. S.base->next = NULL;
23. S.nodenum = 0;
24. **return** OK;
25. }
26. **bool** PushStack(Stack &S,etype &e)
27. {
28. **if**(!S.top || !S.base)
29. **return** ERROR;
30. SNode \*NewNode = (SNode \*)malloc(**sizeof**(SNode));
31. NewNode->seat.x = e.x;
32. NewNode->seat.y = e.y;
33. NewNode->seat.direction = e.direction;
34. NewNode->next = S.top;
35. S.top = NewNode;
36. S.nodenum++;
37. **return** OK;
38. }
39. **bool** PopStack(Stack &S,etype &e)
40. {
41. **if**(!S.top || !S.base || S.base == S.top)
42. **return** ERROR;
43. e.x = S.top->seat.x;
44. e.y = S.top->seat.y;
45. e.direction = S.top->seat.direction;
46. SNode \*node = S.top;
47. S.top = S.top->next;
48. free(node);
49. S.nodenum--;
50. **return** OK;
51. }
52. **bool** visits(etype &e)
53. {
54. cout << "此节点的坐标为:" << "x:"<< e.x << "  y:" << e.y << endl;
55. **return** OK;
56. }
57. **bool** StackTraverse(Stack &S)
58. {
59. **if**(!S.top || !S.base || S.nodenum == 0)
60. **return** ERROR;
61. SNode \*top = S.top;
62. **while**(top != S.base)
63. {
64. visits(top->seat);
65. top = top->next;
66. }
67. **return** OK;
68. }

Stack.h

1. #ifndef STACK\_H\_INCLUDED
2. #define STACK\_H\_INCLUDED
3. #include"Position.h"
4. #define etype Position
5. **typedef** **struct** SNode
6. {
7. Position seat;
8. **struct** SNode \*next;
9. }SNode;
11. **typedef** **struct**
12. {
13. SNode \*base;
14. SNode \*top;
15. **int** nodenum;
16. }Stack;
17. **bool** InitStack(Stack &S);
18. **bool** PushStack(Stack &S,etype &e);
19. **bool** PopStack(Stack &S,etype &e);
20. **bool** visits(etype &e);
21. **bool** StackTraverse(Stack &S);

24. #endif // STACK\_H\_INCLUDED

# 九、算术表达式求值

# 9.1 题目简介

[问题描述]

　　一个算术表达式是由操作数(operand)、运算符(operator)和界限符(delimiter)组成的。假设操作数是正实数，运算符只含加减乘除等四种运算符，界限符有左右括号和表达式起始、结束符“#”，如：#6+15\*（21-8/4）#。引入表达式起始、结束符是为了方便。编程利用“运算符优先法”求算术表达式的值。

[基本要求]

（1） 从键盘或文件读入一个合法的算术表达式，输出正确的结果。

（2） 显示输入序列和栈的变化过程。

（3） 考虑算法的健壮性，当表达式错误时，要给出错误原因的提示。

（4） 实现非整数的处理。

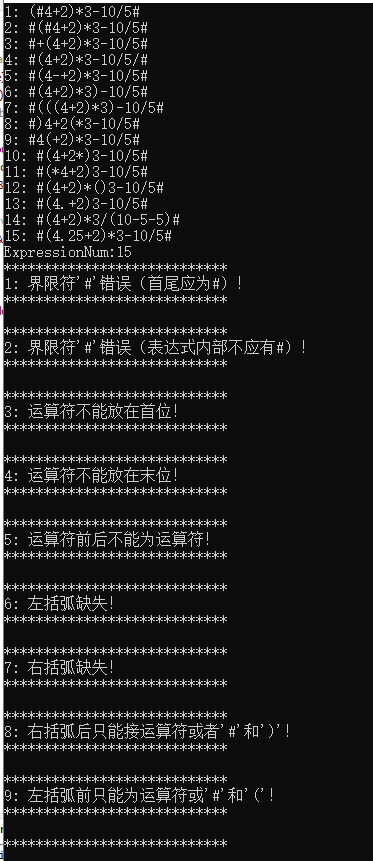
# 9.2 数据结构

用一个运算符栈存放运算符，用一个操作数栈存放操作数。

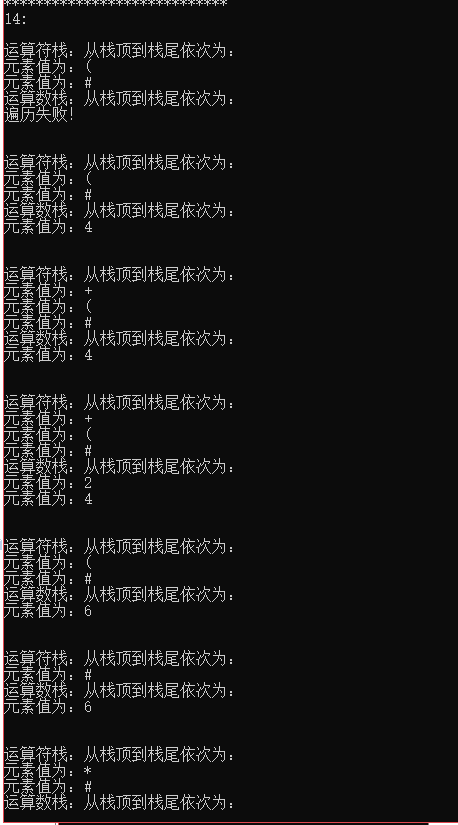
# 9.3 算法设计思想

先扫描整个式子，判断它是否合法。若合法，建立两个栈：一个运算符栈，一个操作数栈。进入循环，若当前的运算符优先级没有运算符栈顶运算符优先级高，则从运算符栈中弹出一个运算符，从操作数栈中弹出两个操作数进行运算，结果依然存放到操作数栈中。若遇到右括号，则在运算之后弹出左括号拖括号。值得注意的是，除数不能够为0，所以在做除法运算时需要判断。

9.4 测试数据和结果，



# 6.5 算法时间复杂度



# 

# 

# 9.6 源代码(416

1. #include<iostream>
2. #include<cstdio>
3. #include<cstdlib>
4. #include<string>
5. #include<fstream>
6. #include<cmath>
7. #include<cstring>
8. #include<string.h>
9. #include<iomanip>
10. #define OK 1
11. #define ERROR 0
12. #define TRUE 1
13. #define FALSE 0
14. #define STACK\_INIT\_SIZE 100
15. #define STACKINCREMENT 10
16. #define MAX\_N INT\_MAX
17. **using** **namespace** std;
18. **typedef** **struct**
19. {
20. **double** \*base;
21. **double** \*top;
22. **int** stacksize;
23. }doubleSqStack;
25. **typedef** **struct**
26. {
27. **char** \*base;
28. **char** \*top;
29. **int** stacksize;
30. }charSqStack;
32. **void** InitdoubleStack(doubleSqStack& S)
33. {
34. S.base=(**double**\*)malloc(STACK\_INIT\_SIZE\***sizeof**(**double**));
35. **if**(!S.base)
36. exit(ERROR);
37. S.top=S.base;
38. S.stacksize=STACK\_INIT\_SIZE;
39. }
41. **void** doublePush(doubleSqStack &S,**double** &e)
42. {
43. **if**(S.top-S.base>=S.stacksize)
44. {
45. S.base=(**double**\*)realloc(S.base,(S.stacksize+STACKINCREMENT)\***sizeof**(**double**));
46. **if**(!S.base)
47. exit(0);
48. S.top=S.base+S.stacksize;
49. S.stacksize+=STACKINCREMENT;
50. }
51. \*S.top = e;
52. ++S.top;
53. }
55. **bool** doublePop(doubleSqStack &S,**double** &e)
56. {
57. **if**(S.top==S.base)
58. **return** **false**;
59. S.top--;
60. e=\*S.top;
61. **return** **true**;
62. }
63. **double** doubleGetTop(doubleSqStack S)
64. {
65. **if**(S.top==S.base)
66. {
67. cout<<"error";
68. exit(0);
69. }
70. **return** \*(S.top-1);
71. }
73. **bool** doubleTraverse(doubleSqStack &S)
74. {
75. **if**(!S.base || !S.top || S.base == S.top)
76. {
77. cout << "遍历失败！" << endl;
78. **return** ERROR;
79. }
80. **double** \*base = S.base;
81. **double** \*top = S.top;
82. **while**(top != base)
83. {
84. cout << "元素值为：" << \*(top-1)<<endl;
85. top--;
86. }
87. **return** OK;
88. }
90. **void** InitcharStack(charSqStack& S)
91. {
92. S.base=(**char**\*)malloc(STACK\_INIT\_SIZE\***sizeof**(**char**));
93. **if**(!S.base)
94. exit(0);
95. S.top=S.base;
96. S.stacksize=STACK\_INIT\_SIZE;
97. }
99. **void** charPush(charSqStack& S,**char** e)
100. {
101. **if**(S.top-S.base>=S.stacksize)
102. {
103. S.base=(**char**\*)realloc(S.base,(S.stacksize+STACKINCREMENT)\***sizeof**(**char**));
104. **if**(!S.base)
105. exit(0);
106. S.top=S.base+S.stacksize;
107. S.stacksize+=STACKINCREMENT;
108. }
109. \*S.top=e;
110. ++S.top;
111. }
113. **char** charGetTop(charSqStack S)
114. {
115. **if**(S.top==S.base)
116. {
117. cout<<"error";
118. exit(0);
119. }
120. **return** \*(S.top-1);
121. }
123. **bool** charPop(charSqStack &S,**char** &e)
124. {
125. **if**(S.top==S.base)
126. **return** **false**;
127. S.top--;
128. e=\*S.top;
129. **return** **true**;
130. }
131. **bool** charTraverse(charSqStack &S)
132. {
133. **if**(!S.base || !S.top || S.base == S.top)
134. {
135. cout << "遍历失败！" << endl;
136. **return** ERROR;
137. }
138. **char** \*base = S.base;
139. **char** \*top = S.top;
140. **while**(top != base)
141. {
142. cout << "元素值为：" << \*(top-1)<<endl;
143. top--;
144. }
145. **return** OK;
146. }
148. **char** Precede(**char** a,**char** b)
149. {
150. **char** ch[7] = {'+','-','\*','/','(',')','#'};
151. **int** ch1,ch2;
152. **char** pre[7][10];
153. strcpy(pre[0],">><<<>>");
154. strcpy(pre[1],">><<<>>");
155. strcpy(pre[2],">>>><>>");
156. strcpy(pre[3],">>>><>>");
157. strcpy(pre[4],"<<<<<= ");
158. strcpy(pre[5],">>>> >>");
159. strcpy(pre[6],"<<<<< =");
160. **for**(**int** i = 0;i < 7;i++)
161. {
162. **if**(ch[i] == a)
163. ch1 = i;
164. **if**(ch[i] == b)
165. ch2 = i;
166. }
167. **return** pre[ch1][ch2];
168. }
169. **bool** isOPTR(**char** ch)
170. {
171. **if**(ch == '+' || ch == '-' || ch == '\*' || ch == '/')
172. **return** OK;
173. **else**
174. **return** ERROR;
175. }
177. **bool** isMARK(**char** ch)
178. {
179. **if**(ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '(' || ch == ')' || ch == '#' || ch == '.')
180. **return** OK;
181. **else**
182. **return** ERROR;
183. }
184. **double** Operate(**double** &a,**char** theta,**double** &b)
185. {
186. **if**(theta == '+')
187. **return** a+b;
188. **else** **if**(theta == '-')
189. **return** a-b;
190. **else** **if**(theta == '\*')
191. **return** a\*b;
192. **else** **if**(theta == '/')
193. {
194. **if**(b == 0)
195. {
196. cout << "ERROR :无法除0！" << endl;
197. **return** MAX\_N;
198. }
199. **return** a/b;
200. }
201. }
202. **double** Pow(**double** x,**int** y)
203. {
204. **double** result = 1;
205. **for**(**int** i = y;y <0;y++)
206. {
207. result = result/x;
208. }
209. **return** result;
210. }
211. **bool** GetValue(string s)
212. {
213. doubleSqStack OPND;
214. charSqStack OPTR;
215. InitcharStack(OPTR);
216. InitdoubleStack(OPND);
217. charPush(OPTR,'#');
218. **int** count1 = 0;
219. **int** count2 = 0;
220. **if**(s[0] != '#' || s[s.length()-1] != '#')
221. {
222. cout << "界限符'#'错误（首尾应为#）！" << endl;
223. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl<<endl;
224. **return** ERROR;
225. }
226. **if**(isOPTR(s[1]))
227. {
228. cout << "运算符不能放在首位！" << endl;
229. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl<<endl;
230. **return** ERROR;
231. }
232. **if**(isOPTR(s[s.length()-2]))
233. {
234. cout << "运算符不能放在末位！" << endl;
235. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl<<endl;
236. **return** ERROR;
237. }
238. **for**(**int** i = 1;i <= s.length()-2;i++)
239. {
240. **if**(s[i] == '#')
241. {
242. cout << "界限符'#'错误（表达式内部不应有#）！" << endl;
243. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl<<endl;
244. **return** ERROR;
245. }
246. **if**(isOPTR(s[i]) && (isOPTR(s[i-1]) || isOPTR(s[i-1])))//运算符前后是运算符
247. {
248. cout << "运算符前后不能为运算符！" << endl;
249. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl<<endl;
250. **return** ERROR;
251. }
252. **if**(s[i] == '(')
253. {
254. count1++;
255. **if**(!isOPTR(s[i-1]) && s[i-1] != '#' && s[i-1] != '(')
256. {
257. cout << "左括弧前只能为运算符或'#'和'('！ " << endl;
258. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl<<endl;
259. **return** ERROR;
260. }
261. **if**(isOPTR(s[i+1]))
262. {
263. cout << "左括弧后不能直接接运算符！" << endl;
264. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl<<endl;
265. **return** ERROR;
266. }
267. **else** **if**(s[i+1] == ')')
268. {
269. cout << "括弧内不能为空！" << endl;
270. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl<<endl;
271. **return** ERROR;
272. }
274. }
275. **else** **if**(s[i] == ')')
276. {
277. count2++;
278. **if**(isOPTR(s[i-1]))
279. {
280. cout << "右括弧前不能直接接运算符！" << endl;
281. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl<<endl;
282. **return** ERROR;
283. }
284. **if**(!isOPTR(s[i+1]) && s[i+1] != '#' && s[i+1] != ')')
285. {
286. cout << "右括弧后只能接运算符或者'#'和')'！" << endl;
287. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl<<endl;
288. **return** ERROR;
289. }
290. }
291. **if**(s[i] == '.' && (isMARK(s[i+1]) || isMARK(s[i-1])))
292. {
293. cout << "小数点前后应为数字！" << endl;
294. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl<<endl;
295. **return** ERROR;
296. }
297. }
298. **if**(count1 != count2)
299. {
300. **if**(count1 > count2)
301. {
302. cout << "右括弧缺失！" << endl;
303. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl<<endl;
304. **return** ERROR;
305. }
306. **else**
307. {
308. cout << "左括弧缺失！" << endl;
309. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl<<endl;
310. **return** ERROR;
311. }
312. }
313. **int** count = 1;
314. **char** theta;
315. **double** a,b;
316. **int** flag = 0;
317. **while**((s[count] != '#' || charGetTop(OPTR)!='#') && count <= s.length()-1)
318. {
319. cout << endl;
320. **if**(!isMARK(s[count]))
321. {
322. **double** num1 = 0;
323. **double** num2 = 0;
324. **while**(!isMARK(s[count]))
325. {
326. num1 = num1\*10 + (s[count]-'0');
327. count++;
328. //cout << s[count];
329. }//当count位置上的元素为字符时停止
330. **if**(s[count] == '.')
331. {
332. count ++;
333. **int** i = -1;
334. **while**(!isMARK(s[count]))
335. {
336. num2 = num2 + (**double**)(s[count]-'0')\*Pow(10.0,i);
337. i--;
338. count++;
339. //cout << s[count];
340. }
341. }
342. **double** num = num1 + num2;
343. //cout <<endl<< "num:" <<num<<endl;
344. //cout << "num1:" << num1<<"  num2:" << num2<<endl;
345. doublePush(OPND,num);
346. }
347. **else**
348. {
349. **switch**(Precede(charGetTop(OPTR),s[count]))
350. {
351. **case** '<':
352. {
353. charPush(OPTR,s[count]);
354. count++;
355. **break**;
356. }
357. **case** '=':
358. {
359. charPop(OPTR,theta);
360. count++;
361. **break**;
362. }
363. **case** '>':
364. {
365. charPop(OPTR,theta);
366. doublePop(OPND,b);
367. doublePop(OPND,a);
368. **double** result = Operate(a,theta,b);
369. **if**(result == MAX\_N)
370. **return** ERROR;
371. **else**
372. doublePush(OPND,result);
373. **break**;
374. }
375. }
376. }
378. cout  << endl << "运算符栈：";
379. cout << "从栈顶到栈尾依次为：" << endl;
380. charTraverse(OPTR);
381. cout << "运算数栈：";
382. cout << "从栈顶到栈尾依次为：" << endl;
383. doubleTraverse(OPND);
384. }
385. **double** result = doubleGetTop(OPND);
386. cout << "运算结果为：" << result << endl;
387. **return** OK;
388. }
390. **int** main()
391. {
392. ifstream infile("data.txt",ios::in);
393. **if**(!infile.is\_open())
394. {
395. cout << "文件无法打开！" << endl;
396. exit(ERROR);
397. }
398. string expression[20];
399. **int** ExpressionNum = 1;
400. **while**(!infile.eof())
401. {
402. getline(infile,expression[ExpressionNum]);
403. cout << ExpressionNum << ": " << expression[ExpressionNum] << endl;
404. ExpressionNum++;
405. }
406. ExpressionNum--;
407. cout << "ExpressionNum:"<<ExpressionNum<<endl;
408. **for**(**int** i = 1;i <= ExpressionNum;i++)
409. {
410. cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;
411. cout << i << ": ";
412. GetValue(expression[i]);
413. }
414. system("pause");
415. infile.close();
416. }