**南京航空航天大学**

**数据结构课程设计报告**

|  |  |
| --- | --- |
| 学生姓名 | W4rd3n |
| 学 号 | 233 |
| 学 院 | 计算机科学与技术学院 |
| 专 业 | 11 |
| 班 级 | 10 |
| 指导教师 | Orz |

二〇一九年一月

**目 录**

[一、 购物网站信息管理 2](#_Toc16524)

[1.1题目简介 2](#_Toc24720)

[1.2数据结构 3](#_Toc28370)

[1.3算设法计思想 4](#_Toc9562)

[1.4测试数据和结果 4](#_Toc31152)

[1.5算法时间复杂度 5](#_Toc32400)

[1.6源代码 5](#_Toc27957)

[二、 公共钥匙盒 18](#_Toc16524)

[2.1题目简介 18](#_Toc24720)

[2.2数据结构 19](#_Toc28370)

[2.3算设法计思想 19](#_Toc9562)

[2.4测试数据和结果 19](#_Toc31152)

[2.5算法时间复杂度 19](#_Toc32400)

[2.6源代码](#_Toc27957)

[三、 树的应用 22](#_Toc16524)

[3.1题目简介 22](#_Toc24720)

[3.2数据结构 23](#_Toc28370)

[3.3算设法计思想 24](#_Toc9562)

[3.4测试数据和结果 24](#_Toc31152)

[3.5算法时间复杂度 24](#_Toc32400)

[3.6源代码 25](#_Toc27957)

[四、 Huffman编码与解码 30](#_Toc16524)

[4.1题目简介 30](#_Toc24720)

[4.2数据结构 30](#_Toc28370)

[4.3算设法计思想](#_Toc9562)

[4.4测试数据和结果](#_Toc31152)

[4.5算法时间复杂度](#_Toc32400)

[4.6源代码](#_Toc27957)

[五、 行车路线 18](#_Toc16524)

[5.1题目简介 18](#_Toc24720)

[5.2数据结构 19](#_Toc28370)

[5.3算设法计思想 19](#_Toc9562)

[5.4测试数据和结果 19](#_Toc31152)

[5.5算法时间复杂度 19](#_Toc32400)

[5.6源代码](#_Toc27957)

[六、 排序算法比较 18](#_Toc16524)

[6.1题目简介 18](#_Toc24720)

[6.2数据结构 19](#_Toc28370)

[6.3算设法计思想 19](#_Toc9562)

[6.4测试数据和结果 19](#_Toc31152)

[6.5算法时间复杂度 19](#_Toc32400)

[6.6源代码](#_Toc27957)

[七、 朋友圈 18](#_Toc16524)

[7.1题目简介 18](#_Toc24720)

[7.2数据结构 19](#_Toc28370)

[7.3算设法计思想 19](#_Toc9562)

[7.4测试数据和结果 19](#_Toc31152)

[7.5算法时间复杂度 19](#_Toc32400)

[7.6源代码](#_Toc27957)

[八、 Hash表应用 18](#_Toc16524)

[8.1题目简介 18](#_Toc24720)

[8.2数据结构 19](#_Toc28370)

[8.3算设法计思想 19](#_Toc9562)

[8.4测试数据和结果 19](#_Toc31152)

[8.5算法时间复杂度 19](#_Toc32400)

[8.6源代码 19](#_Toc27957)

九、 [课程及课设总结 1](#_Toc24048)18

[9.1 完成情况（代码行数） 1](#_Toc3603)18

[9.2 心得体会 1](#_Toc6584)18

# 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

1. **typedef** **struct** Commodity
2. {
3. **int** sales;
4. **double** price;
5. **char** name[20];
6. }commodity;
7. //商品
9. **typedef** **struct** Shop
10. {
11. **int** id;
12. **int** num;
13. **char** name[20];
14. **int** credibility;
15. commodity \* cmdt;
16. //指向商铺对应的商品列表
17. }shop;
18. //商铺
20. **typedef** **struct** ShopList
21. {
22. shop sp;
23. **struct** ShopList \* next;
24. //单向链表
25. }shopList;
26. //商铺列表
28. **typedef** **struct** ShopListOfCommodity
29. {
30. **int** id;
31. **int** sales;
32. **char** name[20];
33. **int** credibility;
34. **struct** ShopListOfCommodity \* next;
35. **struct** ShopListOfCommodity \* last;
36. //双向链表
37. }shopListOfCommodity;
38. //对应商品的商铺列表结点
40. **typedef** **struct** CommodityShopList
41. {
42. commodity cmdt;
43. shopListOfCommodity \* spl;
44. //单向链表
45. }commodityShopList;
46. //对应商品的商铺列表

# 1.3 算法设计思想

首先用单向链表存储商铺，同时动态分配内存保存商铺对应的商品信息

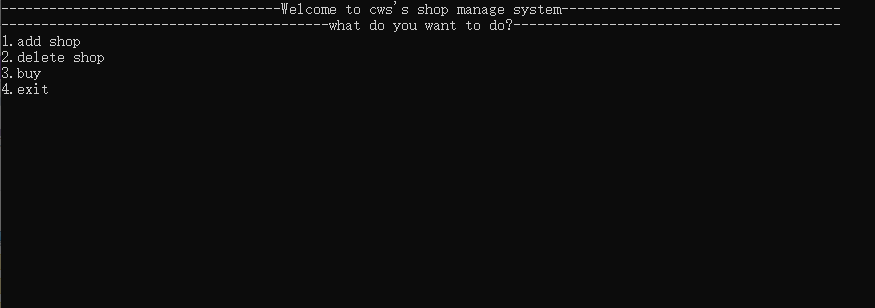
读入商品的同时初始化每个商品的商铺列表，利用插入排序得到有序的双向商铺列表

将每个商品对应的商品链表用一个单向链表连接帮助查询

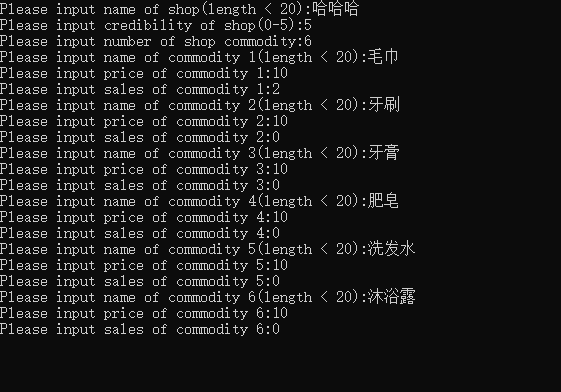
利用hash函数处理商品名索引出对应商品的商铺列表

# 1.4 测试数据和结果

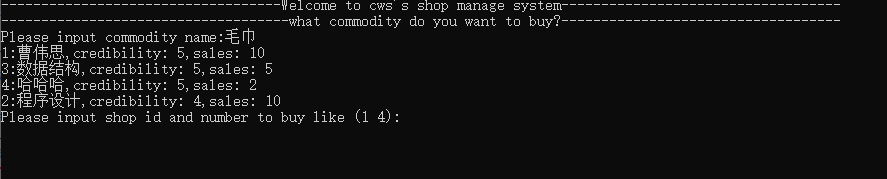
menu



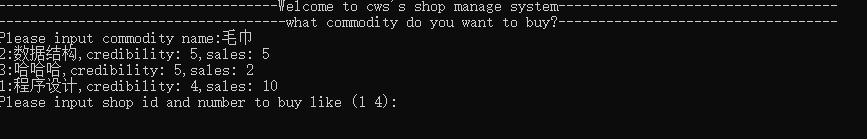
add



buy



delete之后



# 1.5 算法时间复杂度

n为商铺的数量

常数时间：clear,hash, add, menu,main.

O(n)时间：buy, del, Save.

O(n^2)时间：Init.

# 1.6 源代码

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
5. **typedef** **struct** Commodity
6. {
7. **int** sales;
8. **double** price;
9. **char** name[20];
10. }commodity;
11. //商品
13. **typedef** **struct** Shop
14. {
15. **int** id;
16. **int** num;
17. **char** name[20];
18. **int** credibility;
19. commodity \* cmdt;
20. //指向商铺对应的商品列表
21. }shop;
22. //商铺
24. **typedef** **struct** ShopList
25. {
26. shop sp;
27. **struct** ShopList \* next;
28. //单向链表
29. }shopList;
30. //商铺列表
32. **typedef** **struct** ShopListOfCommodity
33. {
34. **int** id;
35. **int** sales;
36. **char** name[20];
37. **int** credibility;
38. **struct** ShopListOfCommodity \* next;
39. **struct** ShopListOfCommodity \* last;
40. //双向链表
41. }shopListOfCommodity;
42. //对应商品的商铺列表结点
44. **typedef** **struct** CommodityShopList
45. {
46. commodity cmdt;
47. shopListOfCommodity \* spl;
48. //单向链表
49. }commodityShopList;
50. //对应商品的商铺列表
52. shopList \* head,\* tail;
53. //商铺列表的头和尾
54. commodityShopList cmdt[10000];
55. //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。
56. **int** num;
57. //当前商铺数量，用于设置id
59. **void** clear()
60. {
61. head = tail = NULL;
62. memset(cmdt,0,10000 \* **sizeof**(commodityShopList));
63. num = 0;
64. }
65. //清除当前状态。
67. **int** hash(**char** \* p)
68. {
69. **int** x = 1,y;
70. **char** \* name = p;
72. **for**(;\*p;p++)
73. {
74. x \*= \*p;
75. x -= \*p / 2;
76. x %= 10000;
77. }
78. y = x;
79. //用于发现开的数组满了
81. **while**(cmdt[x].cmdt.name[0] && strcmp(cmdt[x].cmdt.name,name))
82. //如果发送碰撞，则向后移一位
83. {
84. x++;
85. x %= 10000;
86. **if**(x == y)
87. {
88. printf("Error hash,memery is out!!!\n");
89. exit(0);
90. }
91. //防止溢出
92. }
94. **return** x;
95. }
96. //hash函数
98. **void** Init()
99. {
100. **FILE** \* file;
101. shopList \* now = NULL;
102. shopList buf;
104. file = fopen("1.dat","rb");
105. **if**(!file)
106. {
107. printf("error: fopen(\"1.dat\",\"rb\")");
108. exit(1);
109. }
110. //打开文件进行读取，带错误检测
112. **int** flag = 1;
113. //用于标识是否是第一次fread
114. **int** i = 0,j = 1;
115. //前者用于计数，后者用于判断是否新的节点排在第一个
117. **while**(1)
118. {
119. fread(&buf,**sizeof**(shopList),1,file);
120. //读取到缓冲区
122. **if**(!feof(file))
123. //判断是否到文件末尾
124. {
125. num++;
126. //每读取到一个，商铺数量加一
127. **if**(flag)
128. //第一个商铺
129. {
130. flag = 0;
131. head = now = (shopList \*)malloc(**sizeof**(shopList));
132. //初始化链表头
133. memcpy(now,&buf,**sizeof**(shopList));
134. now->sp.id = num;
135. //设置id
136. now->sp.cmdt = (commodity \*)malloc(**sizeof**(commodity) \* now->sp.num);
137. //读取商铺对应的商品列表
138. fread(now->sp.cmdt,**sizeof**(commodity),now->sp.num,file);
139. }
140. **else**
141. //后面的商铺
142. {
143. now->next = (shopList \*)malloc(**sizeof**(shopList));
144. memcpy(now->next,&buf,**sizeof**(shopList));
145. now->next->sp.id = num;
146. now = now->next;
147. now->sp.cmdt = (commodity \*)malloc(**sizeof**(commodity) \* now->sp.num);
148. fread(now->sp.cmdt,**sizeof**(commodity),now->sp.num,file);
149. //创建新的结点并将数据复制进去
150. }
151. **for**(i = 0;i < now->sp.num;i++)
152. //初始化商品对应的商铺列表
153. {
154. **if**(cmdt[hash(now->sp.cmdt[i].name)].cmdt.name[0])
155. //判断是否是第一次遇见此商品
156. {
157. //如果不是第一次遇见此商品
158. shopListOfCommodity \* now2 = cmdt[hash(now->sp.cmdt[i].name)].spl;
159. //初始化一个指针用来遍历列表进行插入排序
160. cmdt[hash(now->sp.cmdt[i].name)].cmdt.sales += now->sp.cmdt[i].sales;
161. //计算总销售量
162. **while**(now2->credibility > now->sp.credibility || (now2->credibility == now->sp.credibility && now2->sales > now->sp.cmdt[i].sales))
163. //判断新商铺是否比当前商铺更靠后
164. {
165. now2 = now2->next;
166. //如果是，向后遍历
167. **if**(now2 == cmdt[hash(now->sp.cmdt[i].name)].spl)
168. {
169. j = 0;
170. //表示次结点遍历完循环链表，应该在末尾，头节点不变
171. **break**;
172. }
173. }
174. now2 = now2->last;
175. shopListOfCommodity \* temp = now2->next;
176. now2->next = (shopListOfCommodity \*)malloc(**sizeof**(shopListOfCommodity));
177. now2->next->credibility = now->sp.credibility;
178. now2->next->sales = now->sp.cmdt[i].sales;
179. now2->next->id = now->sp.id;
180. strcpy(now2->next->name,now->sp.name);
181. //创建新的结点并将数据复制进去
182. now2->next->last = now2;
183. now2->next->next = temp;
184. temp->last = now2->next;
185. //双向循环链表的结点插入
186. **if**(now2->next->next == cmdt[hash(now->sp.cmdt[i].name)].spl && j)
187. //如果当前为头节点，且j不置0，说明其比头节点“大”
188. {
189. cmdt[hash(now->sp.cmdt[i].name)].spl = now2->next;
190. j = 1;
191. }
192. }
193. **else**
194. //第一次遇到
195. {
196. cmdt[hash(now->sp.cmdt[i].name)].cmdt = now->sp.cmdt[i];
197. cmdt[hash(now->sp.cmdt[i].name)].spl = (shopListOfCommodity \*)malloc(**sizeof**(shopListOfCommodity));
198. cmdt[hash(now->sp.cmdt[i].name)].spl->credibility = now->sp.credibility;
199. cmdt[hash(now->sp.cmdt[i].name)].spl->sales = now->sp.cmdt[i].sales;
200. cmdt[hash(now->sp.cmdt[i].name)].spl->id = now->sp.id;
201. strcpy(cmdt[hash(now->sp.cmdt[i].name)].spl->name,now->sp.name);
202. cmdt[hash(now->sp.cmdt[i].name)].spl->next = cmdt[hash(now->sp.cmdt[i].name)].spl;
203. cmdt[hash(now->sp.cmdt[i].name)].spl->last = cmdt[hash(now->sp.cmdt[i].name)].spl;
204. //初始化并建立双向循环链表
205. }
206. }
207. }
208. **else** **if**(!flag)
209. //如果读到文件尾且读入过数据
210. {
211. now->next = NULL;
212. tail = now;
213. **break**;
214. }
215. //如果读到文件尾且为读入数据
216. **else**
217. {
218. head = tail = NULL;
219. **break**;
220. }
221. }
223. fclose(file);
224. //关闭文件
225. **return**;
226. }
227. //初始化商铺列表和所有商品对应的有序的商铺列表
229. **void** Save()
230. {
231. **FILE** \* file;
232. **int** nums = 1;
233. //商铺id指示符
234. shopList \* now = head;
235. //初始化头指针用于遍历
237. file = fopen("1.dat","wb");
238. **if**(!file)
239. {
240. printf("error: fopen(\"1.dat\",\"wb\")");
241. exit(1);
242. }
244. **while**(now)
245. {
246. shopList \* temp = now;
247. now->sp.id = nums++;
248. fwrite(now,**sizeof**(shopList),1,file);
249. fwrite(now->sp.cmdt,**sizeof**(commodity),now->sp.num,file);
250. now = now->next;
251. //循环写入文件
252. free(now->sp.cmdt);
253. free(now);
254. //释放内存
255. }
257. fclose(file);
258. //关闭文件
259. **return**;
260. }
261. //将商铺信息和对应的商品信息以二进制形式保存在文件中
263. **void** add()
264. {
265. shopList now;
266. //保存用户输入
267. **int** x = 0;
268. //用于读数据和充当计数器
270. now.sp.id = ++num;
271. //设置id
272. now.next = NULL;
274. printf("Please input name of shop(length < 20):");
275. fgets(now.sp.name,20,stdin);
276. **int** l = strlen(now.sp.name);
277. **if**(l != 19)
278. {
279. now.sp.name[l - 1] = '\0';
280. }
281. //清除fgets留下的换行符
283. printf("Please input credibility of shop(0-5):");
284. scanf("%d",&x);
285. fflush(stdin);
286. //刷新缓冲区
287. **if**(x < 0||x > 5)
288. {
289. printf("Error number!!!\n");
290. system("pause");
291. **return**;
292. }
293. now.sp.credibility = x;
295. x = 0;
296. printf("Please input number of shop commodity:");
297. scanf("%d",&now.sp.num);
298. fflush(stdin);
299. now.sp.cmdt = (commodity \*)malloc(now.sp.num \* **sizeof**(commodity));
301. **while**(x++ < now.sp.num)
302. {
303. printf("Please input name of commodity %d(length < 20):",x);
304. fgets(now.sp.cmdt[x - 1].name,20,stdin);
305. l = strlen(now.sp.cmdt[x - 1].name);
306. **if**(l != 19)
307. {
308. now.sp.cmdt[x - 1].name[l - 1] = '\0';
309. }
310. printf("Please input price of commodity %d:",x);
311. scanf("%lf",&now.sp.cmdt[x - 1].price);
312. fflush(stdin);
313. printf("Please input sales of commodity %d:",x);
314. scanf("%d",&now.sp.cmdt[x - 1].sales);
315. fflush(stdin);
316. }
317. //读入对应的商品信息
319. **if**(head)
320. //如果已经有头指针
321. {
322. tail->next = (shopList \*)malloc(**sizeof**(shopList));
323. tail = tail->next;
324. memcpy(tail,&now,**sizeof**(shopList));
325. }
326. **else**
327. //没有则初始化
328. {
329. head = tail = (shopList \*)malloc(**sizeof**(shopList));
330. memcpy(tail,&now,**sizeof**(shopList));
331. }
333. system("cls");
334. //清空屏幕
335. **return**;
336. }
337. //按照用户输入增加商铺，编号自动加一，插入链表尾部。
339. **void** del()
340. {
341. printf("-----------------------------------");
342. printf("Welcome to cws's shop manage system");
343. printf("-----------------------------------\n");
344. printf("-----------------------------------");
345. printf("--what shop do you want to delete?-");
346. printf("-----------------------------------\n");
348. shopList \* now = head,\* last = NULL;
350. **while**(now)
351. {
352. printf("%d:%s\n",now->sp.id,now->sp.name);
353. now = now->next;
354. }
355. //遍历并打印商铺id和名字
357. printf("Please input the shop id:");
358. **int** x;
359. scanf("%d",&x);
361. now = head;
362. **while**(now && now->sp.id != x)
363. {
364. last = now;
365. now = now->next;
366. }
367. //找到对应id商铺
369. **if**(!now)
370. //没有找到，即now为NULL
371. {
372. printf("Error id of shop!!!\n");
373. system("pause");
374. **return**;
375. }
377. **if**(last)
378. //不是头指针
379. {
380. last->next = now->next;
381. **if**(!last->next)
382. //如果为尾指针
383. {
384. tail = last;
385. }
386. free(now->sp.cmdt);
387. free(now);
388. **while**(last->next)
389. //将后面的指针对应的商铺id逐一减一
390. {
391. last = last->next;
392. last->sp.id--;
393. }
394. }
395. **else**
396. //删除的为头指针
397. {
398. **if**(head == tail)tail = NULL;
399. head = head->next;
400. free(now->sp.cmdt);
401. free(now);
402. }
404. system("cls");
405. //清空屏幕
406. **return**;
407. }
408. //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。
410. **void** buy()
411. {
412. printf("-----------------------------------");
413. printf("Welcome to cws's shop manage system");
414. printf("-----------------------------------\n");
415. printf("-----------------------------------");
416. printf("-what commodity do you want to buy?");
417. printf("-----------------------------------\n");
419. printf("Please input commodity name:");
420. **char** name[20];
422. fgets(name,20,stdin);
423. **int** l = strlen(name),m;
424. **if**(l != 19)
425. {
426. name[l - 1] = '\0';
427. }
428. //读入商品名字
430. commodityShopList x = cmdt[hash(name)];
431. **if**(!(x.cmdt.name[0]))
432. //判断是否有商铺有此商品
433. {
434. printf("no sale of %s!!!\n",name);
435. system("pause");
436. **return**;
437. }
439. shopListOfCommodity \* y = x.spl;
441. printf("%d:%s,credibility: %d,sales: %d\n",y->id,y->name,y->credibility,y->sales);
442. y = y->next;
443. **while**(y != x.spl)
444. {
445. printf("%d:%s,credibility: %d,sales: %d\n",y->id,y->name,y->credibility,y->sales);
446. y = y->next;
447. }
448. //遍历商品对应的商铺列表并打印对应的商铺id，名字，信誉度，销量
450. printf("Please input shop id and number to buy like (1 4):");
451. scanf("%d %d",&l,&m);
453. shopList \* now = head;
454. **while**(now && now->sp.id != l)
455. {
456. now = now->next;
457. }
458. //找到对应的商铺结点
460. **if**(now)
461. {
462. **int** i = 0;
463. **for**(;i < now->sp.num;)
464. {
465. **if**(!strcmp(now->sp.cmdt[i].name,name))
466. {
467. now->sp.cmdt[i].sales += m;
468. **break**;
469. //找到对应商品并增加销量
470. }
471. }
472. }
473. **else**
474. {
475. printf("Error id!!!\n");
476. system("pause");
477. **return**;
478. }
480. system("cls");
481. //清空屏幕
482. **return**;
483. }
484. //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。
485. //购买某一商铺的商品，修改单向链表中商品的信息的销量。
487. **void** menu()
488. {
489. **while**(1)
490. {
491. printf("-----------------------------------");
492. printf("Welcome to cws's shop manage system");
493. printf("-----------------------------------\n");
494. printf("-----------------------------------");
495. printf("------what do you want to do?------");
496. printf("-----------------------------------\n");
497. printf("1.add shop\n");
498. printf("2.delete shop\n");
499. printf("3.buy\n");
500. printf("4.exit\n");
502. **int** x;
503. scanf("%d",&x);
504. fflush(stdin);
505. system("cls");
507. **switch**(x)
508. {
509. **case** 1:
510. add();
511. **break**;
512. **case** 2:
513. del();
514. **break**;
515. **case** 3:
516. buy();
517. **break**;
518. **case** 4:
519. **return**;
520. **default**:
521. printf("valid choice!!!\n");
522. **break**;
523. }
525. Save();
526. clear();
527. Init();
528. //任何的商铺信息变化，实现文件存储。
530. system("cls");
531. //清空屏幕
532. }
534. **return**;
535. }
536. //用于交互的菜单界面
538. **int** main()
539. {
540. Init();
541. //通过文件初始化
543. menu();
545. **return** 0;
546. }

# 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** Action
2. {
3. **int** num;
4. **int** time;
5. **int** flag;
6. }action;
7. //保存借钥匙和还钥匙两个动作，flag为0表示借，flag为1表示还

# 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

1. #include<iostream>
2. #include<algorithm>
3. #include<vector>
5. **using** **namespace** std;
7. **int** N,K,w,s,c;
8. **int** key[100001];
10. **typedef** **struct** Action
11. {
12. **int** num;
13. **int** time;
14. **int** flag;
15. }action;
16. //保存借钥匙和还钥匙两个动作，flag为0表示借，flag为1表示还
18. vector<action> a;
19. action temp;
21. **bool** cmp(action a,action b){
22. **if**(a.time != b.time)
23. //先按时间排
24. {
25. **return** a.time < b.time;
26. }
27. **else** **if**(a.flag != b.flag)
28. //先还再借
29. {
30. **return** a.flag > b.flag;
31. }
32. **else**
33. //编号从小到大
34. {
35. **return** a.num < b.num;
36. }
37. }
38. //用于比较的函数
40. **int** main()
41. {
42. cin >> N >> K;
44. **for**(**int** i = 1;i <= N;i++)
45. {
46. key[i]=i;
47. }
48. //初始化钥匙的位置
50. **while**(K--)
51. {
52. cin >> w >> s >> c;
54. temp.num = w;
55. temp.time = s;
56. temp.flag = 0;
57. a.push\_back(temp);
58. //借钥匙
59. temp.num = w;
60. temp.time = s + c;
61. temp.flag = 1;
62. a.push\_back(temp);
63. //还钥匙
64. }
66. sort(a.begin(),a.end(),cmp);
67. //排序
69. **for**(**int** i = 0;i < a.size();i++)
70. {
71. **if**(a[i].flag == 1)
72. //还钥匙
73. {
74. **for**(**int** k = 1;k <= N;k++)
75. {
76. **if**(key[k] == 0)
77. {
78. key[k] = a[i].num;
79. **break**;
80. }
81. }
82. }
83. **else**
84. //借钥匙
85. {
86. **for**(**int** j = 1;j <= N;j++)
87. {
88. **if**(key[j] == a[i].num)
89. {
90. key[j] = 0;
91. **break**;
92. }
93. }
94. }
95. }
97. **for**(**int** i = 1;i <= N;i++)
98. {
99. cout << key[i] <<" ";
100. }
102. **return** 0;
103. }

# 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。

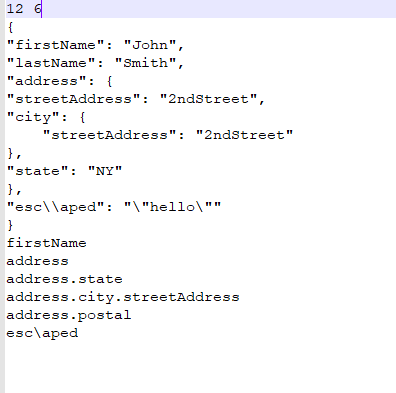
# 3.2 数据结构

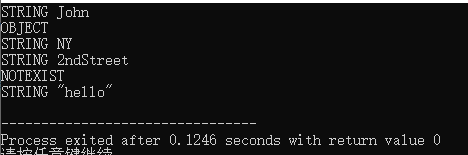
1. **typedef** **struct** Json
2. {
3. **bool** flag;
4. //标识是不是OBJECT
5. string key;
6. //键
7. string value;
8. //值
9. }json;
10. //映射键值的结构体

# 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

1. #include<iostream>
2. #include<fstream>
3. #include<cstring>
4. #include<string>
5. #include<cstdlib>
7. **using** **namespace** std;
9. **typedef** **struct** Json
10. {
11. **bool** flag;
12. //标识是不是OBJECT
13. string key;
14. //键
15. string value;
16. //值
17. }json;
18. //映射键值的结构体
20. fstream file;
21. **int** n,m;
22. json test[10000];
23. //hash映射数组
25. string nowKey;
26. string tempKey;
27. string nowValue;
28. //现在正在处理的key和value
30. **int** flag = 0;
31. //标识现在处理的是key还是value
33. **int** hash(**char** \* p)
34. {
35. **int** x = 1,y;
36. string key = p;
37. //用于处理碰撞
39. **for**(;\*p;p++)
40. {
41. x \*= \*p;
42. x -= \*p / 2;
43. x %= 10000;
44. }
45. y = x;
47. **while**(test[x].key.size() && !(test[x].key == key.substr(5,1000)))
48. //如果发生碰撞，则向后移一位
49. {
50. //cout << "hash(" << test[x].key << "):" << x << endl;
51. x++;
52. x %= 10000;
53. **if**(x == y)
54. {
55. cout << "Error hash,memery is out!!!\n";
56. exit(0);
57. }
58. //防止溢出
59. }
60. //cout << "hash(" << test[x].key << "):" << x << endl;
62. **return** x;
63. }
64. //hash函数
66. **void** toJson(**char** \* line)
67. {
68. **while**(\*line)
69. {
70. **if**(\*line == '{')
71. //对象的开始
72. {
73. **if**(nowKey.size())
74. //改变父节点的字符串
75. {
76. nowKey = tempKey;
77. nowKey += '.';
78. tempKey = nowKey;
79. flag ^= 1;
80. }
81. **else**
82. //第一个'{'字符特殊处理
83. {
84. nowKey = "json.";
85. tempKey = nowKey;
86. }
87. }
88. **else** **if**(\*line == '"')
89. //处理字符串
90. {
91. **if**(!flag)
92. //处理key
93. {
94. line++;
95. **while**(\*line != '"')
96. {
97. **if**(\*line == '\\')
98. {
99. line++;
100. }
101. tempKey += \*line;
102. line++;
103. }
104. //cout << tempKey << ":";
105. }
106. **else**
107. //处理value
108. {
109. line++;
110. **while**(\*line != '"')
111. {
112. **if**(\*line == '\\')
113. {
114. line++;
115. }
116. nowValue += \*line;
117. line++;
118. }
119. //cout << nowValue << "\n";
120. }
121. }
122. **else** **if**(\*line == ':' || \*line == ',' || \*line == '}')
123. //转变和结束
124. {
125. flag ^= 1;
126. **if**(\*line == ',' || \*line == '}')
127. //保存键值映射到json数组
128. {
129. test[hash((**char** \*)tempKey.c\_str())].key = tempKey.substr(5,1000);
130. **if**(!nowValue.size())
131. {
132. test[hash((**char** \*)tempKey.c\_str())].value = "OBJECT";
133. test[hash((**char** \*)tempKey.c\_str())].flag = 1;
134. }
135. **else**
136. {
137. test[hash((**char** \*)tempKey.c\_str())].value = nowValue;
138. test[hash((**char** \*)tempKey.c\_str())].flag = 0;
139. }
140. //cout << test[hash((char \*)tempKey.c\_str())].key << ":" << test[hash((char \*)tempKey.c\_str())].value << endl;
141. **if**(\*line == ',')
142. {
143. tempKey = nowKey;
144. }
145. nowValue = "";
146. }
147. **if**(\*line == '}')
148. //结束一个对象，修改父节点字符串
149. {
150. flag ^= 1;
151. **if**(nowKey == "json.")
152. {
153. nowKey = "";
154. tempKey = nowKey;
155. }
156. **else**
157. {
158. nowKey = nowKey.substr(0,nowKey.rfind("."));
159. tempKey = nowKey;
160. //cout << tempKey << endl;
161. nowKey = nowKey.substr(0,nowKey.rfind(".") + 1);
162. nowValue = "";
163. //cout << nowKey << endl;
164. }
165. }
166. }
168. line++;
169. }
171. **return**;
172. }
173. //将文本信息处理成json结构体
175. **void** getValue(**char** \* line)
176. {
177. string s = "json.";
179. s += line;
181. **if**(test[hash((**char** \*)s.c\_str())].value.size())
182. {
183. **if**(!test[hash((**char** \*)s.c\_str())].flag)
184. {
185. cout << "STRING ";
186. }
187. cout << test[hash((**char** \*)s.c\_str())].value << endl;
188. }
189. **else**
190. {
191. cout << "NOTEXIST" << endl;
192. }
194. **return**;
195. }
196. //通过json结构体获得当前查询的value
198. **int** main()
199. {
200. **char** line[1000];
201. file.open("3.txt",ios::in);
203. file >> n >> m;
204. file.getline(line,1000);
205. //丢弃空行
206. **while**(n--)
207. {
208. file.getline(line,1000);
209. //cout << line << endl;
211. toJson(line);
212. //处理当前行
213. }
215. **while**(m--)
216. {
217. file.getline(line,1000);
219. getValue(line);
220. //处理当前查询
221. }
223. **return** 0;
224. }

# 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** Json
15. {
16. **int** val;
17. string key;
18. }json;
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

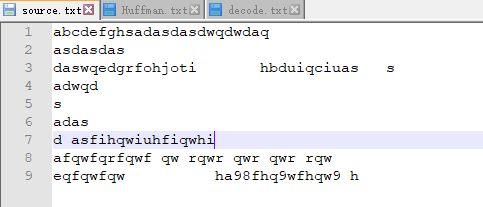
通过字符频率生成Huffman树然后进行编码

将编码后的源文件内容以二进制形式存入code.dat

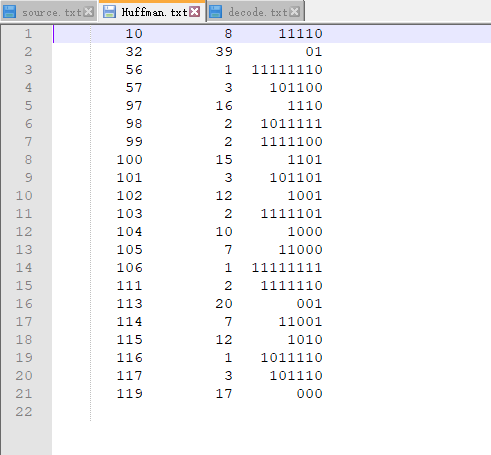
读取Huffman.txt得到编码规则然后解码code.dat获得源文件内容decode.txt

# 4.4 测试数据和结果

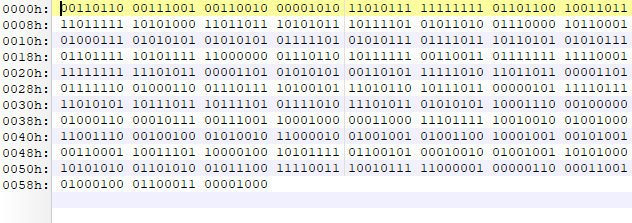
source.txt



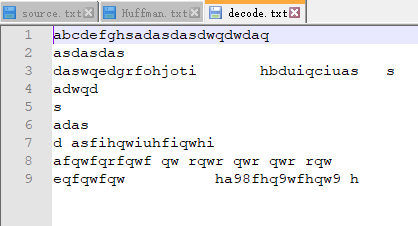
Huffman.txt



code.dat



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

1. #include<iostream>
2. #include<iomanip>
3. #include<fstream>
4. #include<vector>
5. #include<string>
6. #define INF 0x7fffffff
8. **using** **namespace** std;
10. fstream source,Huffman,code,decodes;
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** Json
27. {
28. **int** val;
29. string key;
30. }json;
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. Json test[10000];
43. //用于通过Huffman编码字符串索引到对应字符
45. **int** hash(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;
59. **while**(test[x].key.size() && !(test[x].key == key))
60. //如果发生碰撞，则向后移一位
61. {
62. x++;
63. x %= 10000;
64. **if**(x == y)
65. {
66. cout << "Error hash,memery is out!!!\n";
67. exit(0);
68. }
69. //防止溢出
70. }
72. **return** x;
73. }
74. //hash函数
76. **void** setBit(**int** \* bit,**int** i)
77. {
78. bit[i / **sizeof**(**int**) / 8] |= 1 << (i % (**sizeof**(**int**) \* 8));
80. **return**;
81. }
82. //设置指定的位为1
84. **void** clearBit(**int** \* bit,**int** i)
85. {
86. bit[i / **sizeof**(**int**) / 8] &= (1 << (i % (**sizeof**(**int**) \* 8))) ^ 0xffffffff;
88. **return**;
89. }
90. //设置指定的位为0
92. **bool** getBit(**int** \* bit,**int** i)
93. {
94. **return** bit[i / **sizeof**(**int**) / 8] & 1 << (i % (**sizeof**(**int**) \* 8));
95. }
96. //获得指定的位的值
98. **void** Init()
99. {
100. source.open("source.txt",ios::in);
101. //读取原始英文文章
102. source >> noskipws;
103. //不忽略空格和换行
105. Huffman.open("Huffman.txt",ios::out);
106. //存储各字符出现次数和编码
108. code.open("code.dat",ios::out | ios::binary);
109. //存储二进制形式的编码后的文章
111. decodes.open("decode.txt",ios::out);
112. //存储二进制形式的编码后的文章
114. **for**(**int** i = 0;i < 128;i++)
115. {
116. counts[i].key = i;
117. }
118. //初始化node中的key值
119. }
120. //初始化文件流和结构体数组
122. **void** Fini()
123. {
124. source.close();
125. Huffman.close();
126. code.close();
127. decodes.close();
129. **return**;
130. }
131. //关闭文件流
133. **void** create()
134. {
135. **for**(**int** i = 0;i < num - 1;i++)
136. {
137. **int** min1 = INF,min2 = INF;
138. **int** a = 0,b = 0,j = 0;
139. **while**(j < 256 && counts[j].count)
140. {
141. **if**(!counts[j].p && counts[j].count < min2)
142. {
143. **if**(counts[j].count < min1)
144. {
145. min2 = min1;
146. min1 = counts[j].count;
147. b = a;
148. a = j;
149. }
150. **else**
151. {
152. min2 = counts[j].count;
153. b = j;
154. }
155. }
156. j++;
157. }
158. **if**(j < 256)
159. {
160. counts[a].p = j;
161. counts[b].p = j;
162. counts[j].l = a;
163. counts[j].r = b;
164. counts[j].count = min1 + min2;
165. //cout << j << ".  " << "l:" << counts[j].l << ",r:";
166. //cout << counts[j].r << ",count:" << counts[j].count <<endl;
167. }
168. }
170. **return**;
171. }
172. //创建Huffman树
174. **void** count()
175. {
176. **char** x;
178. source >> x;
179. **while**(!source.eof())
180. {
181. counts[x].count++;
182. source >> x;
183. //cout << x;
184. }
186. **for**(**int** i = 0;i < 128;i++)
187. {
188. **if**(counts[i].count)
189. {
190. num++;
191. }
192. **else**
193. {
194. counts[i].count = INF;
195. }
196. }
198. source.close();
199. source.open("source.txt",ios::in);
200. source >> noskipws;
201. //重新打开准备编码
203. **return**;
204. }
205. //计算字符出现的次数
207. **void** encode()
208. {
209. **for**(**int** i = 0;i < 128;i++) {
210. string temp;
211. **int** j = i;
212. **int** p = counts[i].p;
213. **while**(p) {
214. **if** (counts[p].l == j)temp.insert(0,"0");
215. **else** temp.insert(0,"1");
216. j = p;
217. p = counts[p].p;
218. }
219. codes.push\_back(temp);
220. }
222. **for**(**int** i = 0;i < 128;i++) {
223. **if**(counts[i].count == INF)
224. {
225. **continue**;
226. }
227. //cout << (char)i << ".  " << "val is " << counts[i].count << ", code is " << codes[i] << endl;
228. Huffman << setw(10) << counts[i].key;
229. Huffman << setw(10) << counts[i].count;
230. Huffman << setw(10) << codes[i];
231. Huffman << endl;
232. }
233. Huffman.close();
234. Huffman.open("Huffman.txt",ios::in);
235. //重新打开准备解码成文本
237. **char** x;
239. source >> x;
240. **while**(!source.eof())
241. {
242. //cout << "string:" << codes[x] << "\t\tbit:";
243. string s = codes[x];
244. **for**(**int** i = 0;i < s.size();i++)
245. {
246. **if**(s[i] == '0')
247. {
248. clearBit(bitmap,length);
249. //cout << getBit(bitmap,length);
250. length++;
251. }
252. **else** **if**(s[i] == '1')
253. {
254. setBit(bitmap,length);
255. //cout << getBit(bitmap,length);
256. length++;
257. }
258. }
259. //cout << "\t\tchar:" << x << endl;
260. source >> x;
261. }
263. code << length << "\n";
264. code.write((**char** \*)bitmap,length / 8 + 1);
266. code.close();
267. code.open("code.dat",ios::in | ios::binary);
268. //读取二进制形式的编码后的文章准备解码

271. **return**;
272. }
273. //进行Huffman编码并以字符串形式保存，写入Huffman文件和code文件
275. **void** decode()
276. {
277. //cout << length << endl;
278. code >> length;
279. **char** pad[100];
280. **int** pad2;
281. //cout << length << endl;
282. //cout << hex << bitmap[1] << endl;
283. code.getline(pad,100);
284. code.read((**char** \*)bitmap,length / 8 + 1);
285. //cout << hex << bitmap[1] << endl;
286. code.clear();
288. string key;
289. **int** val;
290. Huffman >> val;
291. **while**(!Huffman.eof())
292. {
293. Huffman >> pad2;
294. Huffman >> key;
295. //cout << key << endl;
296. //cout << val << endl;
297. test[hash(key)].key = key;
298. test[hash(key)].val = val;
299. Huffman >> val;
300. }
302. string m;
303. **char** n;
304. **for**(**int** i = 0;i < length;i++)
305. {
306. **if**(getBit(bitmap,i))
307. {
308. m += '1';
309. }
310. **else**
311. {
312. m += '0';
313. }
314. **if**(test[hash(m)].key.size())
315. {
316. n = test[hash(m)].val;
317. //cout << "str:" << m << "\t\tval: " << (char)n << endl;
318. decodes << n;
319. m = "";
320. }
321. }
323. **return**;
324. }
325. //通过Huffman编码规则解密编码后的文件
327. **int** main()
328. {
329. Init();
331. count();
332. create();
333. encode();
334. decode();
336. Fini();
338. **return** 0;
339. }

# 五、行车路线

# 5.1 题目简介

(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+2^2=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. **typedef** **struct** Node
2. {
3. **long** **long** c;
4. //长度
5. **int** flag;
6. //标识道路的类型，1为小道，0为大道
7. }node;

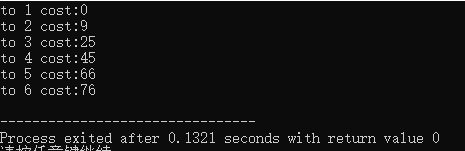
# 5.3 算法设计思想

用二维数组储存每条路的种类和花费，然后利用Dijkstra算法不断松弛花费得到单源最短路径。

# 5.4 测试数据和结果



另外增加了路径的显示



# 5.5 算法时间复杂度

设结点数位n

复杂度O(n^2):Dijkstra

# 5.6 源代码

1. #include<stdio.h>
2. #include<stdlib.h>
3. #define INF 0x7fffffff
5. **typedef** **struct** Node
6. {
7. **long** **long** c;
8. //长度
9. **int** flag;
10. //标识道路的类型，1为小道，0为大道
11. }node;
13. **FILE** \* input;
14. //文件指针
16. **int** n,m;
17. **int** t,a,b;
18. **long** **long** c;
19. //用于读取输入的变量
21. **struct** Node G[1001][1001];
22. //邻接矩阵储存道路
24. **long** **long** dd[1001];
25. **long** **long** xd[1001];
26. **long** **long** xd\_sq[1001];
27. //dd和sd分别保存每个结点到出发点的大道路径之和和小道路径之和,区分点为最后的一条路的种类
28. **long** **long** d[1001];
29. //保存花费
31. **int** p[1001];
32. //记录结点的前驱结点
33. **int** flag[1001];
34. //标记是否访问
36. **void** Dijkstra(**int** s,**int** n)
37. {
38. d[s] = 0;
39. dd[s] = 0;
40. xd[s] = 0;
41. //初始化到起点的花费为0
43. **while**(1)
44. {
45. **int** x = -1;
46. **long** **long** min = INF;
48. **for**(**int** i = 1;i <= n;i++)
49. {
50. **if**(!flag[i] && d[i] < min)
51. //遍历寻找最小花费
52. {
53. x = i;
54. //记录编号
55. min = d[i];
56. //记录当前的最小数
57. }
58. }
60. **if**(x == -1)
61. {
62. **break**;
63. }
64. flag[x] = 1;
66. **for**(**int** i = 1;i <= n;i++)
67. //遍历松弛
68. {
69. **if**(G[x][i].c != INF && !flag[i])
70. //存在道路且另外一个点没走过
71. {
72. **if**(G[x][i].flag == 1)
73. //当前道路是小道
74. {
75. **long** **long** xdnow = INF;
76. //计算前面走小道的花费
77. **long** **long** ddnow = INF;
78. //计算前面走大道的花费
80. **if**(xd[x] != INF)
81. //源结点到x有小道
82. {
83. **if**(p[x] == 0)
84. //没有前驱结点，也就是并没有交错
85. {
86. xdnow = (xd[x] + G[x][i].c) \* (xd[x] + G[x][i].c);
87. }
88. **else**
89. //有交错
90. {
91. xdnow = dd[p[x]] + (xd[x] + G[x][i].c) \* (xd[x] + G[x][i].c);
92. }
93. }
95. **if**(dd[x] != INF)
96. //源结点到x有大道
97. {
98. ddnow = (G[x][i].c) \* (G[x][i].c) + dd[x];
99. }
101. **if**(xdnow < ddnow)
102. //前面选择走小道
103. {
104. xd[i] = xd[i] < (xd[x] + G[x][i].c) ? xd[i] : (xd[x] + G[x][i].c);
105. xd\_sq[i] = xdnow < xd\_sq[i] ? xdnow : xd\_sq[i];
106. }
107. **else**
108. //前面选择走大道
109. {
110. xd[i] = xd[i] < G[x][i].c ? xd[i] : G[x][i].c;
111. xd\_sq[i] = ddnow < xd\_sq[i] ? ddnow : xd\_sq[i];
113. p[i] = x;
114. //记录前驱结点
115. }
117. d[i] = xd\_sq[i] < dd[i] ? xd\_sq[i] : dd[i];
118. //记录最小花费
119. }
120. **else**
121. //当前道路是大道
122. {
123. **long** **long** xdnow = INF;
124. **long** **long** ddnow = INF;
126. **if**(dd[x] != INF)
127. //源结点到x有大道
128. {
129. xdnow = dd[x] + G[x][i].c;
130. }
131. **if**(xd[x] != INF)
132. //源结点到x有小道
133. {
134. ddnow = xd\_sq[x] + G[x][i].c;
135. }
137. **long** **long** min2;
139. min2 = xdnow < ddnow ? xdnow : ddnow;
140. dd[i] = dd[i] < min2 ? dd[i] : min2;
141. d[i] = xd\_sq[i] < dd[i] ? xd\_sq[i] : dd[i];
142. }
143. }
144. }
145. }
147. **return**;
148. }
150. **int** main()
151. {
152. input = fopen("input.txt","r+");
154. fscanf(input,"%d %d",&n,&m);
156. **for**(**int** i = 0;i <= n;i++)
157. {
158. **for**(**int** j = 0;j <= n;j++)
159. {
160. G[i][j].c = INF;
161. G[j][i].c = INF;
162. }
163. }
164. //初始化道路的花费
166. **for**(**int** i = 0;i < 1001;i++)
167. {
168. dd[i] = INF;
169. xd[i] = INF;
170. xd\_sq[i] = INF;
171. p[i] = 0;
172. d[i] = INF;
173. }
175. **while**(m--)
176. {
177. fscanf(input,"%d %d %d %lld",&t,&a,&b,&c);
178. //printf("%d %d %d %lld\n",t,a,b,c);
180. **if**(t==1)
181. //小道
182. {
183. G[a][b].c = c;
184. G[b][a].c = c;
185. G[a][b].flag = 1;
186. G[b][a].flag = 1;
187. }
188. **else**
189. //大道
190. {
191. G[a][b].c = c;
192. G[b][a].c = c;
193. G[a][b].flag = 0;
194. G[b][a].flag = 0;
195. }
196. }
198. Dijkstra(1,n);
199. //使用Dijkstra算法求单元最短路径
201. **for**(**int** i = 1;i <= n;i++)
202. {
203. printf("to %d cost:%d\n",i,d[i]);
204. }
206. fclose(input);
207. //关闭文件指针
209. **return** 0;
210. }

# 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

用数组存储数据

# 6.3 算法设计思想

直接插入排序即不断向有序区插入元素并始终维护有序区的有序性，从而得到一个新的记录数量增加一的有序表。

希尔排序是插入排序的一种，是非稳定排序算法。希尔排序通过步长将数组分为几部分来进行插入排序可以减少由于某些数据造成的大量元素移动。

冒泡排序不断比较相邻的元素。如果第一个比第二个大，就交换他们两个。对每一对相邻元素做同样的工作，最后的元素会是最大的数。针对所有的元素重复以上的步骤，除了最后一个。

快速排序通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，按此方法对这两部分进行快速排序，整个排序过程[递归](https://baike.baidu.com/item/%E9%80%92%E5%BD%92/1740695)进行使整个数据变成有序[序列](https://baike.baidu.com/item/%E5%BA%8F%E5%88%97/1302588)。

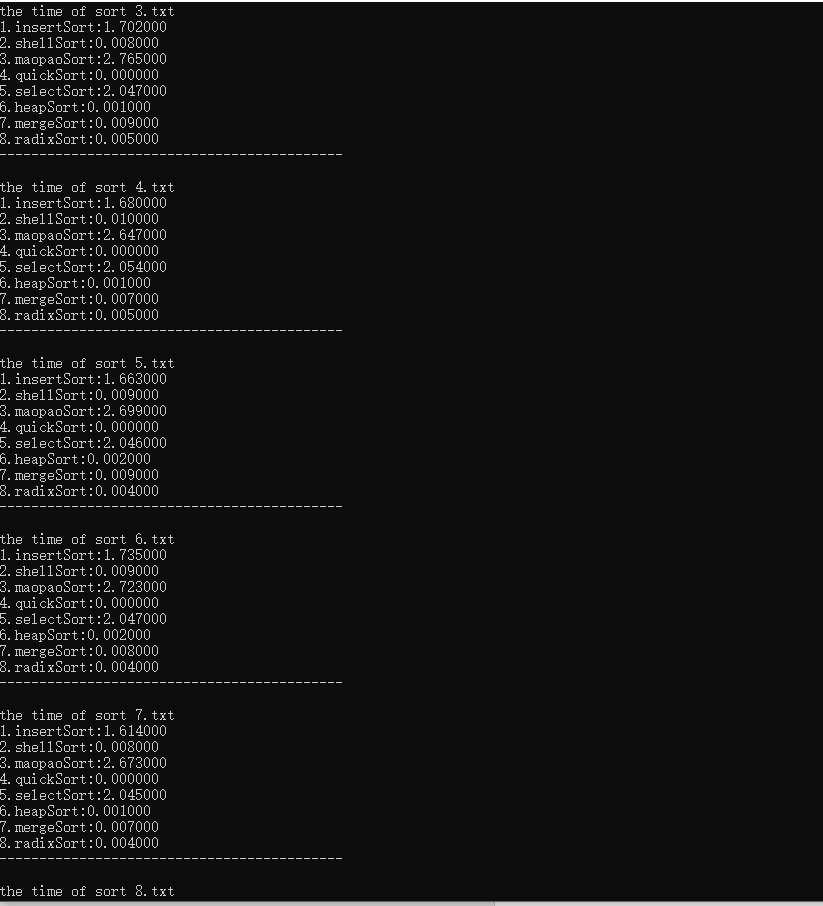
选择排序首先在未排序序列中找到最小（大）元素，存放到有序序列的末尾，然后重复这个操作。

堆排序即先使用下沉操作将数组变成二叉堆，然后即可获得最值，然后不断使用下降维护这个二叉堆。

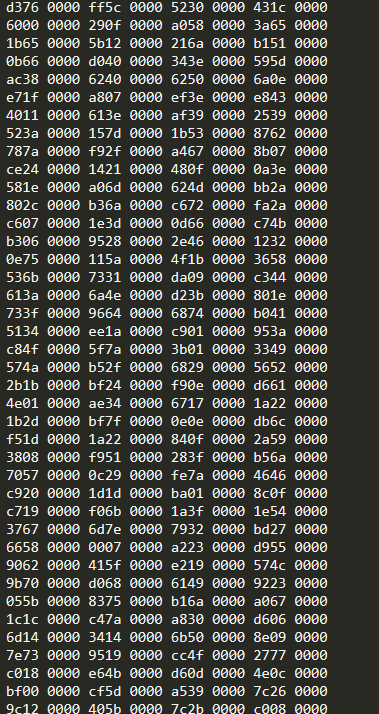
归并排序将已有序的子序列合并，得到完全有序的序列；即先使每个子序列有序，再使子序列段间有序。

基数排序先从影响小的位排序，不断向前，也可以减小大量的元素移动。

# 6.4 测试数据和结果



数据使用二进制存储



# 6.5 算法时间复杂度

设n为数据数量，基数排序中r为关键字基数，d为长度，n为关键字个数

O(n^2)：直接插入排序，选择排序，冒泡排序

O(n^1.3)：Shell排序

O(nlog2(n))：堆排序，快速排序，归并排序

O(d(r+n))：基数排序

# 6.6 源代码

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. #include<time.h>
5. #define N 40000
7. **clock\_t** begin,end;
8. //用于计算排序函数消耗的时间
9. **char** fileName[10][7]={"1.txt","2.txt","3.txt","4.txt","5.txt",
10. "6.txt","7.txt","8.txt","9.txt","10.txt"};
11. //样本名数组
12. **int** Data[N];
13. **int** reData[N];
14. //数据数组
16. **void** insertSort(**int** a[],**int** n)
17. //插入排序
18. //a为待排序数组
19. //n为数组长度
20. {
21. **for**(**int** i = 1;i < n;i++)
22. {
23. **int** j;
24. **for**(j = i - 1;j >= 0;j--)
25. //找到插入的位置
26. {
27. **if**(a[j] < a[i])
28. **break**;
29. }
31. **if**(j != i - 1)
32. //如果不在最后面
33. {
34. **int** temp = a[i];
36. **int** k;
37. **for**(k = i - 1;k > j;k--)
38. //将插入位置及后面的元素向后移一位
39. {
40. a[k + 1]=a[k];
41. }
43. a[k + 1]=temp;
44. //插入元素
45. }
46. }
48. **return**;
49. }
51. **void** shellSort(**int** a[],**int** n)
52. //希尔排序
53. //a为待排序数组
54. //n为数组长度
55. {
56. **for**(**int** gap = n / 2;gap > 0;gap /= 2)
57. //设置步长,每次减半
58. {
59. **for**(**int** i = 0;i < gap;i++)
60. //对每一组使用插入排序
61. {
62. **for**(**int** j = i + gap;j < n;j += gap)
63. {
64. **if**(a[j - gap] > a[j])
65. {
66. **int** temp = a[j];
67. **int** k = j - gap;
69. **while**(k >= 0 && a[k] > temp)
70. {
71. a[k + gap] = a[k];
72. k -= gap;
73. }
75. a[k + gap] = temp;
76. }
77. }
78. }
79. }
81. **return**;
82. }
84. **void** maopaoSort(**int** a[],**int** n)
85. //冒泡排序
86. //a为待排序数组
87. //n为数组长度
88. {
89. **for**(**int** i = 0;i < n;i++)
90. //每次循环找出一个最值放在末尾
91. {
92. **for**(**int** j = n - i - 1;j < n - 1;j++)
93. {
94. **if**(a[j] > a[j + 1])
95. {
96. **int** temp = a[j];
97. a[j] = a[j + 1];
98. a[j + 1] = temp;
99. }
100. }
101. }
103. **return**;
104. }
106. **void** quickSort(**int** a[],**int** l,**int** r)
107. //快速排序
108. //a为待排序数组
109. //n为数组长度
110. {
111. **if**(l < r)
112. {
113. **return**;
114. }
116. **int** i,j,x;
118. i = l;
119. j = r;
120. x = a[i];
121. //哨兵
123. **while**(i < j)
124. {
125. **while**(i < j && a[j] > x)
126. //找第一个小于x的数
127. {
128. j--;
129. }
131. **if**(i < j)
132. {
133. a[i] = a[j];
134. a[j] = x;
135. i++;
136. }
138. **while**(i < j && a[i] < x)
139. //找第一个大于x的数
140. {
141. i++;
142. }
144. **if**(i < j)
145. {
146. a[j] = a[i];
147. a[i] = x;
148. j--;
149. }
150. }
152. a[i] = x;
154. quickSort(a,l,i - 1);
155. quickSort(a,i + 1,r);
157. **return**;
158. }
160. **void** selectSort(**int** a[],**int** n)
161. //选择排序
162. //a为待排序数组
163. //n为数组长度
164. {
165. **int** min;
166. //无序区中最小元素位置
168. **for**(**int** i = 0; i<n; i++)
169. //逐渐增加有序区
170. {
171. min = i;
173. **for**(**int** j = i + 1;j < n;j++)
174. //找出无序区的最小元素
175. {
176. **if**(a[j] < a[min])
177. {
178. min = j;
179. }
180. }
181. **if**(min != i)
182. //交换保证有序区元素是有序的
183. {
184. **int** temp = a[i];
185. a[i] = a[min];
186. a[min] = temp;
187. }
188. }
190. **return**;
191. }
193. **void** heapDown(**int** a[],**int** start,**int** end)
194. //堆排序下沉操作
195. //a为待排序数组
196. //start为起始位置
197. //end为截至范围
198. {
199. **int** l = 2 \* start + 1;
200. //左孩子的index
201. **int** r = l + 1;
202. //右孩子的index
203. **int** temp = a[start];
205. **while**(1)
206. {
207. **if**(l < end && a[l] < a[l + 1])
208. //选择较大者
209. {
210. l++;
211. **if**(temp >= a[r])
212. //调整结束
213. {
214. **break**;
215. }
216. **else**
217. //交换值
218. {
219. a[start] = a[r];
220. a[r] = temp;
221. }
223. start = r;
224. r = 2 \* r + 1;
226. **if**(r > end)
227. {
228. **break**;
229. }
230. }
231. **else**
232. {
233. **if**(temp >= a[l])
234. //调整结束
235. {
236. **break**;
237. }
238. **else**
239. //交换值
240. {
241. a[start] = a[l];
242. a[l] = temp;
243. }
245. start = l;
246. l = 2 \* l + 1;
248. **if**(l > end)
249. {
250. **break**;
251. }
252. }
253. }
255. **return**;
256. }
258. **void** heapSort(**int** a[],**int** n)
259. //堆排序
260. //a为待排序数组
261. //n为数组长度
262. {
263. **for**(**int** i = n / 2 - 1;i >= 0; i--)
264. //逐次遍历得到一个最大二叉堆
265. {
266. heapDown(a,i,n - 1);
267. }
269. **for**(**int** i = n - 1;i > 0;i--)
270. //从最后一个元素开始对序列进行调整
271. {
272. **int** temp = a[0];
273. a[0] = a[i];
274. a[i] = temp;
275. //交换使a[i]为最大
277. heapDown(a, 0, i-1);
278. //恢复最大堆
279. }
281. **return**;
282. }
284. **void** merge(**int** a[],**int** start,**int** mid,**int** end)
285. //合并有序数组
286. //a为数组指针
287. //start到mid和mid到end为两个待排序数组
288. {
289. **int** temp[end - start + 1];
290. //存放合并后的元素
291. **int** i = start;
292. //第1个有序区的索引
293. **int** j = mid + 1;
294. //第2个有序区的索引
295. **int** k = 0;
296. //临时区域的索引
298. **while**(i <= mid && j <= end)
299. //合并数组
300. {
301. **if** (a[i] <= a[j])
302. {
303. temp[k++] = a[i++];
304. }
305. **else**
306. {
307. temp[k++] = a[j++];
308. }
309. }
311. **while**(i <= mid)
312. {
313. temp[k++] = a[i++];
314. }
316. **while**(j <= end)
317. {
318. temp[k++] = a[j++];
319. }
320. //将剩余的数组元素放入temp
322. **for**(i = 0;i < k;i++)
323. //将排序后的元素整合到数组a
324. {
325. a[start + i] = temp[i];
326. }
328. **return**;
329. }
331. **void** mergeSort(**int** a[],**int** start,**int** end)
332. //归并排序
333. //a为待排序的数组
334. //start和end为数组的起始地址和结束地址
335. {
336. **int** mid;
338. **if**(start >= end)
339. {
340. **return**;
341. }
343. mid = (start + end) / 2;
344. mergeSort(a,start,mid);
345. mergeSort(a,mid + 1,end);
347. merge(a,start,mid,end);
349. **return**;
350. }
352. **void** countSort(**int** a[],**int** n,**int** exp)
353. //对数组按照某个位进行桶排序
354. //a为待排序数组
355. //n为数组长度
356. //exp为指数
357. {
358. **int** temp[n];
359. //存储排序完元素的数组
360. **int** buckets[10] = {0};
362. **for**(**int** i = 0; i < n; i++)
363. //桶排序
364. {
365. buckets[(a[i] / exp) % 10]++;
366. }
368. **for**(**int** i = 1;i < 10;i++)
369. //更改buckets[i]为其在temp中的位置
370. {
371. buckets[i] += buckets[i - 1];
372. }
374. **for**(**int** i = n - 1;i >= 0;i--)
375. {
376. temp[buckets[(a[i] / exp) % 10] - 1] = a[i];
377. buckets[(a[i] / exp) % 10]--;
378. }
380. **for**(**int** i = 0;i < n;i++)
381. {
382. a[i] = temp[i];
383. }
385. **return**;
386. }
388. **void** radixSort(**int** a[], **int** n)
389. //基数排序
390. //a为待排序数组
391. //n为数组长度
392. {
393. **int** max;
394. // 数组a中的最大值
396. max = a[0];
397. **for**(**int** i = 1;i < n;i++)
398. {
399. **if**(a[i] > max)
400. {
401. max = a[i];
402. }
403. }
405. // 从个位开始，对数组a按"指数"进行排序
406. **for**(**int** i = 1;max / i > 0;i \*= 10)
407. {
408. countSort(a,n,i);
409. }
411. **return**;
412. }
414. **void** setData()
415. //产生10个样本每个样本有20000个随机整数
416. //并使前后两个样本分别为正序和逆序
417. //使用二进制存储
418. {
419. **FILE** \* out;
421. **for**(**int** i = 0;i < 5;i++)
422. {
423. srand(time(0));
424. **for**(**int** j = 0;j < N;j++)
425. {
426. Data[j] = rand() % 100000;
427. reData[N - j - 1] = Data[j];
428. }
430. out = fopen(fileName[i \* 2],"wb");
432. **if**(!out)
433. {
434. printf("Error!!!fopen(%s,\"wb\");\n",fileName[i \* 2]);
435. system("pause");
436. exit(1);
437. }
439. fwrite(Data,**sizeof**(**int**),N,out);
440. fclose(out);
442. out = fopen(fileName[i \* 2 + 1],"wb");
444. **if**(!out)
445. {
446. printf("Error!!!fopen(%s,\"wb\");\n",fileName[i \* 2 + 1]);
447. system("pause");
448. exit(1);
449. }
451. fwrite(reData,**sizeof**(**int**),N,out);
452. fclose(out);
453. }
455. **return**;
456. }
458. **void** getData(**char** name[],**int** a[])
459. //使用文件中的数据赋值数组a
460. {
461. **FILE** \* in;
463. in = fopen(name,"rb");
465. **if**(!in)
466. {
467. printf("Error!!!fopen(%s,\"rb\");\n",name);
468. system("pause");
469. exit(1);
470. }
472. fread(a,**sizeof**(**int**),N,in);
474. fclose(in);
476. **return**;
477. }
479. **int** main()
480. {
481. **double** Time;
483. setData();
485. **for**(**int** i = 0;i < 10;i++)
486. {
487. printf("the time of sort %s\n",fileName[i]);
489. getData(fileName[i],Data);
490. begin = clock();
491. insertSort(Data,N);
492. end = clock();
493. Time = (**double**)(end - begin) / CLOCKS\_PER\_SEC;
494. printf("1.insertSort:%lf\n",Time);
496. getData(fileName[i],Data);
497. begin = clock();
498. shellSort(Data,N);
499. end = clock();
500. Time = (**double**)(end - begin) / CLOCKS\_PER\_SEC;
501. printf("2.shellSort:%lf\n",Time);
503. getData(fileName[i],Data);
504. begin = clock();
505. maopaoSort(Data,N);
506. end = clock();
507. Time = (**double**)(end - begin) / CLOCKS\_PER\_SEC;
508. printf("3.maopaoSort:%lf\n",Time);
510. getData(fileName[i],Data);
511. begin = clock();
512. quickSort(Data,0,N - 1);
513. end = clock();
514. Time = (**double**)(end - begin) / CLOCKS\_PER\_SEC;
515. printf("4.quickSort:%lf\n",Time);
517. getData(fileName[i],Data);
518. begin = clock();
519. selectSort(Data,N);
520. end = clock();
521. Time = (**double**)(end - begin) / CLOCKS\_PER\_SEC;
522. printf("5.selectSort:%lf\n",Time);
524. getData(fileName[i],Data);
525. begin = clock();
526. heapSort(Data,N);
527. end = clock();
528. Time = (**double**)(end - begin) / CLOCKS\_PER\_SEC;
529. printf("6.heapSort:%lf\n",Time);
531. getData(fileName[i],Data);
532. begin = clock();
533. mergeSort(Data,0,N - 1);
534. end = clock();
535. Time = (**double**)(end - begin) / CLOCKS\_PER\_SEC;
536. printf("7.mergeSort:%lf\n",Time);
538. getData(fileName[i],Data);
539. begin = clock();
540. radixSort(Data,N);
541. end = clock();
542. Time = (**double**)(end - begin) / CLOCKS\_PER\_SEC;
543. printf("8.radixSort:%lf\n",Time);
545. printf("-------------------------------------------\n\n");
546. }
548. **return** 0;
549. }

# 七、朋友圈

# 7.1 题目简介

朋友圈（选做）（集合的等价关系）

[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

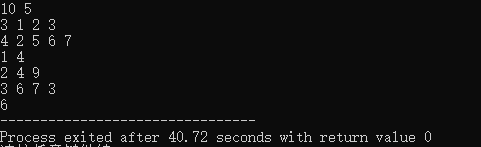
1. **int** mem[30001];
2. //mem[i]的值为i的朋友，组成一个环形链表

由于朋友的传递性，此数据结构可包含一个人所有朋友

# 7.3 算法设计思想

利用环形数组来构成并查集解决问题

# 7.4 测试数据和结果



# 7.5 算法时间复杂度

每次加入新的成员要遍历一遍朋友圈，朋友圈不断增长（按数据情况增长）

为O(M\*N/2)

# 7.6 源代码

1. #include<stdio.h>
2. #include<stdlib.h>
4. **int** mem[30001];
5. //mem[i]的值为i的朋友，组成一个环形链表
7. **void** join(**int** x[],**int** a,**int** b)
8. {
9. **int** i = a;
11. **while**(x[i] && x[i] != a)
12. {
13. i = x[i];
14. }
16. **int** j = b;
18. **while**(x[j] && x[j] != b)
19. {
20. j = x[j];
21. }
22. //找到a和b在环形链表中的前一个
24. //printf("%d\n",i);
25. //printf("%d\n",j);
27. x[i] = b;
28. x[j] = a;
29. //合并环形链表
31. **return**;
32. }
34. **int** main()
35. {
36. **int** n,m;
38. scanf("%d %d",&n,&m);
40. **while**(m--)
41. {
42. **int** num,a,b;
44. scanf("%d",&num);
46. **if**(num)
47. //防止num为0
48. {
49. scanf("%d",&a);
50. }
51. **else**
52. {
53. **continue**;
54. }
56. **for**(**int** j = 0;j < num - 1;j++)
57. {
58. b = a;
60. scanf("%d",&a);
61. join(mem,a,b);
62. }
63. }
65. **int** max = 0x80000000;
66. **for**(**int** i = 1;i <= n;i++)
67. {
68. **int** num = 1;
69. //自己算一个
71. **int** j = i;
73. **while**(mem[j] && mem[j] != i)
74. {
75. j = mem[j];
76. num++;
77. }
78. //获得朋友圈数量
80. **if**(num > max)
81. {
82. max = num;
83. }
84. }
86. printf("%d", max);
88. **return** 0;
89. }

# 八、Hash表应用

# 8.1 题目简介

Hash表应用 （选做） （查找）

[问题描述]

设计散列表实现VIP客户发掘。对身份证号进行Hash, 通过对乘客某时间段内的乘机频率、里程数统计，发掘VIP客户。

[基本要求]

（1） 设每个记录有下列数据项：身份证号码（虚构，位数和编码规则与真实一致即可）、姓名、航班号、航班日期、里程。

（2） 从文件输入各记录，以身份证号码为关键字建立散列表。

（3） 分别采用开放定址（自行选择和设计定址方案）和链地址两种方案解决冲突；显示发生冲突的次数、每次中解决冲突进行重定位的次数。

（4）记录条数至少在100条以上。

（5） 从记录中实现乘客乘机频率、里程数统计，从而发掘VIP客户。

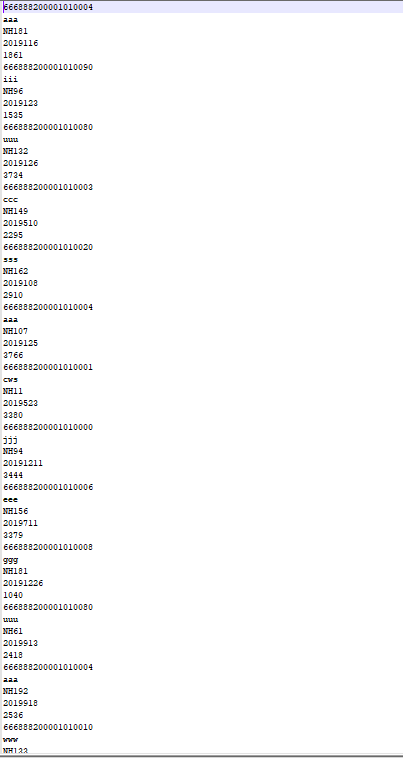
# 8.2 数据结构

1. **typedef** **struct** Record
2. {
3. string id;
4. //身份证号
5. string name;
6. //姓名
7. string flyId;
8. //航班号
9. string flyDate;
10. //航班日期
11. **int** distance;
12. //里程
13. **int** conflict;
14. //冲突数量
15. **struct** Record \* next;
16. //单向链表
17. }record;
18. //记录
20. **typedef** **struct** PassengerRecord
21. {
22. string id;
23. //身份证号
24. string name;
25. //姓名
26. **int** allDistance;
27. //里程数统计
28. **int** rate;
29. //乘机频率
30. }passengerRecord;
31. //乘客记录

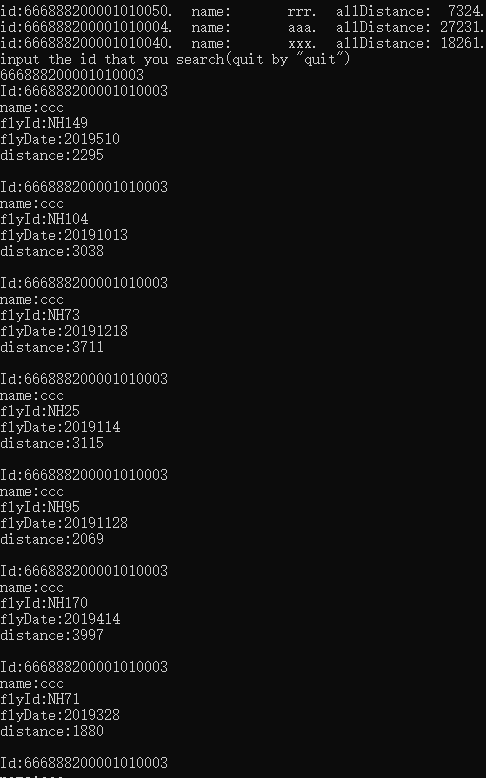
# 8.3 算法设计思想

# 8.4 测试数据和结果

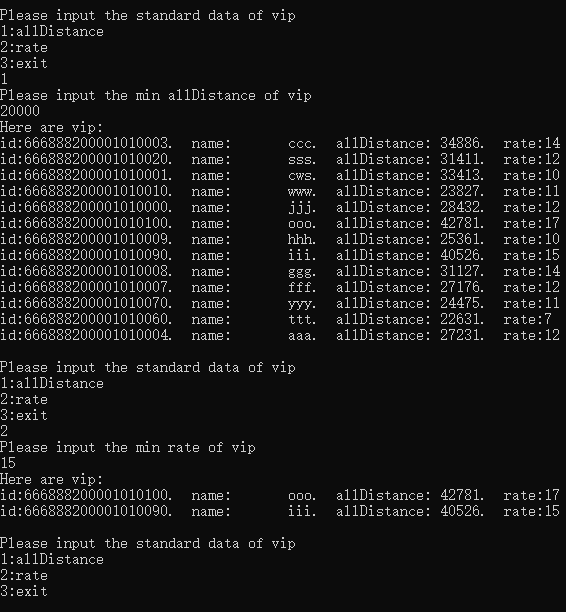
数据



按照id找寻记录



按照条件找寻vip



# 8.5 算法时间复杂度

# 8.6 源代码

1. #include<iostream>
2. #include<stdlib.h>
3. #include<fstream>
4. #include<iomanip>
5. #include<string>
6. #include<time.h>
8. **using** **namespace** std;
10. fstream file;
12. string sfz[20] = {
13. "666888200001010001","666888200001010002","666888200001010003","666888200001010004","666888200001010005",
14. "666888200001010006","666888200001010007","666888200001010008","666888200001010009","666888200001010000",
15. "666888200001010010","666888200001010020","666888200001010030","666888200001010040","666888200001010050",
16. "666888200001010060","666888200001010070","666888200001010080","666888200001010090","666888200001010100",
17. };
19. string name[20] = {
20. "cws","bbb","ccc","aaa","ddd","eee","fff","ggg","hhh","jjj",
21. "www","sss","zzz","xxx","rrr","ttt","yyy","uuu","iii","ooo",
22. };
24. **typedef** **struct** Record
25. {
26. string id;
27. //身份证号
28. string name;
29. //姓名
30. string flyId;
31. //航班号
32. string flyDate;
33. //航班日期
34. **int** distance;
35. //里程
36. **int** conflict;
37. //冲突数量
38. **struct** Record \* next;
39. //单向链表
40. }record;
41. //记录
43. record records[10000];
44. record hash1[10000];
45. record \* hash2[10000];
47. **typedef** **struct** PassengerRecord
48. {
49. string id;
50. //身份证号
51. string name;
52. //姓名
53. **int** allDistance;
54. //里程数统计
55. **int** rate;
56. //乘机频率
57. }passengerRecord;
58. //乘客记录
60. **void** setData()
61. {
62. file.open("data.txt",ios::out);
64. string id = "66688820000101";
66. srand(time(0));
68. **for**(**int** i = 0;i < 200;i++)
69. {
70. **int** j = rand() % 20;
71. file << sfz[j] << endl;
72. file << name[j] << endl;
73. file << "NH" << rand() % 200 << endl;
74. file << "2019" << rand() % 12 + 1 << rand() % 28 + 1 << endl;
75. file << rand() % 3000 + 1000 << endl;
76. }
78. file.close();
80. **return**;
81. }
83. **int** getData()
84. {
85. file.open("data.txt",ios::in);
87. **if**(!file.is\_open())
88. {
89. cout << "Error open file data.txt" << endl;
90. exit(1);
91. }
93. **int** num = 1;
95. **while**(!file.eof())
96. {
97. getline(file,records[num].id);
98. getline(file,records[num].name);
99. getline(file,records[num].flyId);
100. getline(file,records[num].flyDate);
101. file >> records[num].distance;
102. file.ignore();
103. num++;
104. }
106. file.close();
108. **return** num - 1;
109. }
111. **void** Hash1(**int** num)
112. //开发定址
113. {
114. **for**(**int** i = 1;i <= num;i++)
115. {
116. **int** x = 1,y;
117. **int** conflict = 0;
118. string id = records[i].id;
119. //用于处理碰撞
121. **for**(**int** j = 0;j < id.size();j++)
122. {
123. x \*= id[j];
124. x -= id[j] / 2;
125. x %= 10000;
126. }
127. y = x;
129. **while**(hash1[x].id.size())
130. //如果发生碰撞，则向后移一位
131. {
132. x++;
133. conflict++;
135. **if**(x > 9999)
136. {
137. x %= 10000;
138. }
140. **if**(x == y)
141. {
142. cout << "Error hash,memery is out!!!\n";
143. exit(0);
144. }
145. //防止溢出
146. }
148. hash1[x].id = records[i].id;
149. hash1[x].name = records[i].name;
150. hash1[x].flyId = records[i].flyId;
151. hash1[x].flyDate = records[i].flyDate;
152. hash1[x].distance = records[i].distance;
153. hash1[x].conflict = conflict;
154. }
156. **return**;
157. }
159. **void** Hash2(**int** num)
160. //链地址法
161. {
162. **for**(**int** i = 1;i <= num;i++)
163. {
164. **int** x = 1;
165. **int** conflict = 0;
166. string id = records[i].id;
167. //用于处理碰撞
169. **for**(**int** j = 0;j < id.size();j++)
170. {
171. x \*= id[j];
172. x -= id[j] / 2;
173. x %= 10000;
174. }
176. record \* now = hash2[x]->next;
177. record \* pre = hash2[x];
179. **while**(now && !(now->id == id))
180. {
181. pre = now;
182. now = now->next;
183. conflict++;
184. }
186. record \* temp = **new** record();
187. temp->id = records[i].id;
188. temp->name = records[i].name;
189. temp->flyId = records[i].flyId;
190. temp->flyDate = records[i].flyDate;
191. temp->distance = records[i].distance;
192. temp->conflict = conflict;
194. temp->next = now;
195. pre->next = temp;
196. //将temp插入到pre指针和now指针中间
197. }
199. **return**;
200. }
202. **void** search1(string id)
203. {
204. **int** x = 1;
206. **for**(**int** i = 0;i < id.size();i++)
207. {
208. x \*= id[i];
209. x -= id[i] / 2;
210. x %= 10000;
211. }
213. **while**(hash1[x].id.size())
214. //如果发生碰撞，则向后移一位
215. {
216. **if**(hash1[x].id == id)
217. {
218. cout << "Id:" << hash1[x].id << endl;
219. cout << "name:" <<hash1[x].name << endl;
220. cout << "flyId:" << hash1[x].flyId <<endl;
221. cout << "flyDate:" << hash1[x].flyDate << endl;
222. cout << "distance:" << hash1[x].distance << endl << endl;
223. }
225. x++;
226. }
228. **return**;
229. }
231. **void** search2(string id)
232. {
233. **int** x = 1;
235. **for**(**int** i = 0;i < id.size();i++)
236. {
237. x \*= id[i];
238. x -= id[i] / 2;
239. x %= 10000;
240. }
242. record \* now = hash2[x]->next;
243. record \* pre = hash2[x];
245. **while**(now  && now->id != id)
246. {
247. pre = now;
248. now = now->next;
249. }
251. **while**(now && now->id == id)
252. {
253. cout << "Id:" << now->id << endl;
254. cout << "name:" << now->name << endl;
255. cout << "flyId:" << now->flyId << endl;
256. cout << "flyDate:" << now->flyDate << endl;
257. cout << "distance:" << now->distance << endl << endl;
258. now = now->next;
259. }
261. **return**;
262. }
264. **int** main()
265. {
266. string aaa;
268. cout << "Do you want to setData repeat?(Y or N)";
269. cin >> aaa;
270. **if**(aaa == "Y")
271. {
272. setData();
273. }
275. **int** conflictCount = 0;
277. **for**(**int** i = 0;i < 10000;i++)
278. {
279. hash1[i].distance = -1;
280. hash1[i].conflict = 0;
281. hash2[i] = (record \*)malloc(**sizeof**(record));
282. hash2[i]->next = NULL;
283. hash2[i]->conflict = 0;
284. }
285. //初始化数据
287. **int** num;
289. num = getData();
291. Hash1(num);
292. Hash2(num);
294. cout << "------------------------------------------------------------------------------------------------------" << endl;
295. cout << "-----------------------------------------------开放定址-----------------------------------------------" << endl;
296. cout << "------------------------------------------------------------------------------------------------------" << endl;
298. **for**(**int** i = 0;i < 10000;i++)
299. {
300. **if**(hash1[i].distance != -1)
301. {
302. cout << "i:" << setw(4) << i << ".  name:" << hash1[i].name << ".  flyId:" << setw(16) << hash1[i].flyId << endl;
303. cout << "conflict:" << hash1[i].conflict <<endl;
304. **if**(hash1[i].conflict)
305. {
306. conflictCount++;
307. }
308. }
309. }
311. cout << "conflictCount:" << conflictCount << endl;
312. cout << endl;
314. **int** Personcount = 0;
316. passengerRecord Person\_record[num + 1];
318. **for**(**int** i = 1;i <= num;i++)
319. {
320. Person\_record[i].allDistance = 0;
321. Person\_record[i].rate = 0;
322. }
324. cout << "------------------------------------------------------------------------------------------------------" << endl;
325. cout << "-----------------------------------------------链地址法-----------------------------------------------" << endl;
326. cout << "------------------------------------------------------------------------------------------------------" << endl;
328. conflictCount = 0;
330. **for**(**int** i = 1;i < 10000;i++)
331. {
332. **if**(hash2[i]->next)
333. {
334. Personcount++;
335. record \* now = hash2[i]->next;
336. Person\_record[Personcount].id = now->id;
337. Person\_record[Personcount].name = now->name;
338. **while**(now)
339. {
340. **if**(now->id == Person\_record[Personcount].id)
341. {
342. Person\_record[Personcount].allDistance += now->distance;
343. Person\_record[Personcount].rate++;
344. }
345. **else**
346. {
347. Personcount++;
348. Person\_record[Personcount].id = now->id;
349. Person\_record[Personcount].name = now->name;
350. Person\_record[Personcount].allDistance += now->distance;
351. Person\_record[Personcount].rate++;
352. }
354. cout << "i:" << setw(4) << i << ".  name:" << now->name << ".  flyId:" << setw(16) << now->flyId << endl;
355. cout << "conflict:" << now->conflict <<endl;
357. **if**(now->conflict)
358. {
359. conflictCount++;
360. }
362. now = now->next;
363. }
364. }
365. }
367. cout << "conflictCount:" << conflictCount << endl;
368. cout << endl;
369. cout << "Personcount:" << Personcount << endl;
371. **for**(**int** i = 1;i <= Personcount;i++)
372. {
373. cout << "id:" << setw(4) << Person\_record[i].id << ".  name:" << setw(10) << Person\_record[i].name;
374. cout << ".  allDistance:" << setw(6) << Person\_record[i].allDistance << ".  rate:" << Person\_record[i].rate <<endl;
375. }
377. string id;
379. **while**(1)
380. {
381. cout << "input the id that you search(quit by \"quit\")" << endl;
382. cin >> id;
383. **if**(id != "quit")
384. {
385. search1(id);
386. }
387. **else**
388. {
389. **break**;
390. }
391. }
393. **int** choose;
395. **while**(1)
396. {
397. cout << endl << "Please input the standard data of vip" << endl;
398. cout << "1:allDistance" << endl;
399. cout << "2:rate" <<endl;
400. cout << "3:exit" << endl;
402. cin >> choose;
404. **if**(choose == 1)
405. {
406. **int** allDistance = 0;
407. cout << "Please input the min allDistance of vip" <<endl;
408. cin >> allDistance;
409. cout << "Here are vip:" << endl;
410. **for**(**int** i = 1;i <= Personcount;i++)
411. {
412. **if**(Person\_record[i].allDistance >= allDistance)
413. {
414. cout << "id:" << setw(4) << Person\_record[i].id << ".  name:" << setw(10) << Person\_record[i].name;
415. cout << ".  allDistance:" << setw(6) << Person\_record[i].allDistance << ".  rate:" << Person\_record[i].rate <<endl;
416. }
417. }
418. }
419. **else** **if**(choose == 2)
420. {
421. **int** rate = 0;
422. cout << "Please input the min rate of vip" <<endl;
423. cin >> rate;
424. cout << "Here are vip:" << endl;
425. **for**(**int** i = 1;i <= Personcount;i++)
426. {
427. **if**(Person\_record[i].rate >= rate)
428. {
429. cout << "id:" << setw(4) << Person\_record[i].id << ".  name:" << setw(10) << Person\_record[i].name;
430. cout << ".  allDistance:" << setw(6) << Person\_record[i].allDistance << ".  rate:" << Person\_record[i].rate <<endl;
431. }
432. }
433. }
434. **else** **if**(choose == 3)
435. {
436. **return** 0;
437. }
438. **else**
439. {
440. cout << "valid choice" << endl;
441. }
442. }
444. **for**(**int** i = 0;i < 10000;i++)
445. {
446. free(hash2[i]);
447. }
449. **return** 0;
450. }

# 九、课程设计总结

# 9.1完成情况（代码行数）

必做题1： 546 行

必做题2： 103 行

必做题3： 224 行

必做题4： 339 行

必做题5： 210 行

必做题6： 549 行

选做题8： 89 行

选做题16：450 行

共： 2510行

# [9.2 心得体会](#_Toc6584)