数据结构上机实验题报告

题目：集合的并、交和差运算

姓名：

班级：

学号：

提交日期2021.11.08

**一、题目**

描述题目所要求的功能，说明输入和输出。

1. 编写一个能演示执行集合的并、交和差运算的程序
2. 集合的元素判定和子集判定运算
3. 集合的补集

**二、程序设计**

描述数据类型定义，或抽象数据类型定义。

单链表，其val为字母

节点用struct结构。

struct node

{

char count;

struct node \*next;

};

描述算法执行步骤，用类C语言描述算法。

输入第一个集合个数

输入第一个集合

输入第二个集合个数

输入第二个集合

排序第一个集合

排序第二个集合

去重第一个集合

去重第二个集合

输出并集

输出交集

输出差集

输出第一个集合补集

输出第二个集合补集

释放两个集合的空间

分析算法的时间复杂度和空间复杂度。

时间复杂度 O(n2)，瓶颈是链表的冒泡排序

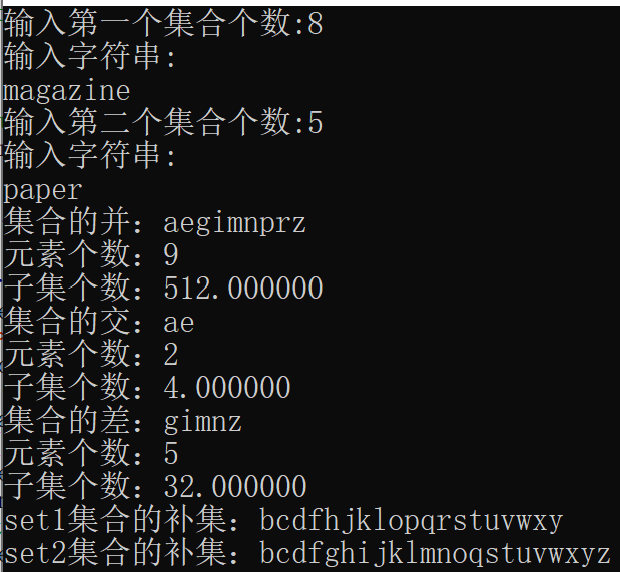
空间复杂度 O(m+n)，是链表储存的花费。此外，有建立字符数组的固定花费

**三、程序测试运行**

说明编译环境和运行环境

codeblocks

说明测试数据，输入的数据与输出的测试结果。



**四、编程工作总结**

说明编程工作内容、心得，讨论哪些可以改进。

**内容：**对于要求的多个需求，制作了多个函数，分别实现功能。详情见源代码

**心得：**

1. 各个功能要分别写函数，彼此独立。
2. 函数的接口一定要设计好，比如传送的应该是指针的指针，而不是指针。
3. 去重操作意义重大，去重后就好做很多了

**改进：**排序可以改成归并排序，时间复杂度可以降到O(logn)

**五、程序源代码**

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. #include<math.h>
5. struct node
6. {
7. char count;
8. struct node \*next;
9. };
10. char map[]="abcdefghijklmnopqrstuvwxyz";
11. typedef struct node LIST;
12. void release(struct node \*\*headptrptr);
13. struct node\* add(int n);
14. void printfdata(struct node \*\*headptrptr);
15. void bubble(struct node \*\*headptrptr); *//冒泡排序升序*
16. void duplicateremove(struct node \*\*headptrptr);*//去重*
17. void merge(struct node \*\*headptrptr1,struct node \*\*headptrptr2);
18. void intersection(struct node \*\*headptrptr1,struct node \*\*headptrptr2);
19. void deletenode(struct node \*\*headptrptr2,int num);
20. void insert(struct node \*\*headptrptr1,int num);
21. void difference(struct node \*\*headptrptr1,struct node \*\*headptrptr2);
22. void Complement(struct node \*\*headptrptr1);
23. int main(void)
24. {
25. int n,m;
26. printf("输入第一个集合个数:");
27. scanf("%d",&n);
28. LIST \*set1=add(n);
29. printf("输入第二个集合个数:");
30. scanf("%d",&m);
31. LIST \*set2=add(m);
32. bubble(&set1);
33. bubble(&set2);
34. duplicateremove(&set1);
35. duplicateremove(&set2);
36. merge(&set1,&set2);
37. intersection(&set1,&set2);
38. difference(&set1,&set2);
39. printf("set1集合的补集：");
40. Complement(&set1);
41. printf("set2集合的补集：");
42. Complement(&set2);
43. release(&set1);
44. release(&set2);
45. return 0;
46. }
47. struct node\* add(int n)
48. {
49. struct node \*last=NULL;
50. struct node \*current=NULL;
51. struct node \*head=NULL;
52. char a[150];
53. char \*sptr;
54. sptr=a;
55. printf("输入字符串:\n");
56. getchar();
57. gets(a);
58. while(n!=0)
59. {
60. current=(struct node\*)malloc(sizeof(struct node));
61. if(current!=NULL)
62. {
63. current->count=\*sptr;
64. if(head==NULL)
65. {
66. head=current;
67. last=current;
68. }
69. else
70. {
71. last->next=current;
72. last=current;*//修正*
73. }
74. }
75. n=n-1;
76. sptr++;
77. }
78. if(last!=NULL)
79. {
80. last->next=NULL;
81. }
82. return head;
83. }
84. void release(struct node \*\*headptrptr)
85. {
86. struct node \*temp;
87. while(\*headptrptr!=NULL)
88. {
89. temp=\*headptrptr;
90. \*headptrptr=(\*headptrptr)->next;
91. free(temp);
92. }
93. }
94. void intersection(struct node \*\*headptrptr1,struct node \*\*headptrptr2)
95. {
96. struct node \*temp1,\*temp2;
97. temp1=\*headptrptr1;
98. temp2=\*headptrptr2;
99. char a[100];
100. int count=0;
101. printf("集合的交：");
102. while(temp1!=NULL&&temp2!=NULL)
103. {
104. if(temp2->count<temp1->count)
105. {
106. temp2=temp2->next;
107. }
108. else if(temp2->count==temp1->count)
109. {
110. a[count]=temp1->count;
111. count++;
112. temp2=temp2->next;
113. }
114. else
115. {
116. temp1=temp1->next;
117. }
118. }
119. for(int i=0;i<strlen(a);i++)
120. printf("%c",a[i]);
121. printf("\n元素个数：%d\n",count);
122. printf("子集个数：%lf\n",pow(2,count));
123. }
124. void printfdata(struct node \*\*headptrptr)
125. {
126. struct node \*temp;
127. temp=\*headptrptr;
128. while(temp!=NULL)
129. {
130. printf("%c ",temp->count);
131. temp=temp->next;
132. }
133. printf("\n");
134. }
135. void duplicateremove(struct node \*\*headptrptr)
136. {
137. struct node \*temp;
138. temp=\*headptrptr;
139. if(temp->next != NULL)
140. {
141. while(temp->next->next!=NULL)
142. {
143. if(temp->next->count == temp->count)
144. {
145. struct node \*te = temp->next;
146. temp->next = temp->next->next;
147. free(te);
148. }
149. else
150. temp = temp->next;
151. }
152. if(temp->next!=NULL)
153. {
154. if(temp->next->count == temp->count)
155. {
156. free(temp->next);
157. temp->next = NULL;
158. }
159. }
160. }
161. }
162. void bubble(struct node \*\*headptrptr)
163. {
164. struct node \*previous, \*current;
165. int tempValue;
166. int flag=1; *//保证主体循环的标记*
167. if(headptrptr==NULL)
168. {
169. return ;
170. }
171. while(flag>0)
172. {
173. flag=0;
174. previous = \*headptrptr;
175. */\*当前节点指向有效数据节点\*/*
176. if(\*headptrptr != NULL)
177. {
178. current = (\*headptrptr)->next;
179. }
180. else
181. {
182. current = NULL;
183. }
184. while(current!=NULL)
185. {
186. if(current->count < previous->count)
187. {
188. tempValue = current->count;
189. current->count = previous->count;
190. previous->count = tempValue;
191. flag++;
192. }
193. previous = current;
194. current = current->next;
195. }
196. }
197. }
198. void merge(struct node \*\*headptrptr1,struct node \*\*headptrptr2)
199. {
200. struct node \*temp1,\*temp2,\*temp4;
201. char a[100];
202. int flag=0;
203. int count=0;
204. temp1=\*headptrptr1;
205. temp2=\*headptrptr2;
206. while(temp1!=NULL&&temp2!=NULL)
207. {
208. if(temp2->count<temp1->count)
209. {
210. for(int i=0;i<strlen(a);i++)
211. {
212. if(a[i]==temp2->count)
213. {
214. flag=1;
215. break;
216. }
217. }
218. if(flag==0)
219. {
220. a[count]=temp2->count;
221. count++;
222. }
223. temp2=temp2->next;
224. }
225. else if(temp2->count==temp1->count)
226. {
227. a[count]=temp2->count;
228. count++;
229. temp1=temp1->next;
230. temp2=temp2->next;
231. }
232. else
233. {
234. for(int i=0;i<strlen(a);i++)
235. {
236. if(a[i]==temp1->count)
237. {
238. flag=1;
239. break;
240. }
241. }
242. if(flag==0)
243. {
244. a[count]=temp1->count;
245. count++;
246. }
247. temp1=temp1->next;
248. }
249. flag=0;
250. }
251. while(temp1)
252. {
253. a[count]=temp1->count;
254. count++;
255. temp1=temp1->next;
256. }
257. while(temp2)
258. {
259. a[count]=temp2->count;
260. count++;
261. temp2=temp2->next;
262. }
263. printf("集合的并：");
264. for(int i=0;i<count;i++)
265. printf("%c",a[i]);
266. printf("\n元素个数：%d\n",count);
267. printf("子集个数：%lf\n",pow(2,count));
268. }
269. void deletenode(struct node \*\*headptrptr2,int num)
270. {
271. struct node \*previous;
272. struct node \*current;
273. previous=NULL;
274. current=\*headptrptr2;
275. while(current!=NULL&&current->count!=num)
276. {
277. previous=current;
278. current=current->next;
279. }
280. if(current!=NULL)*//条件*
281. {
282. if(previous==NULL)
283. {
284. \*headptrptr2=current->next;
285. }
286. else
287. {
288. previous->next=current->next;
289. }
290. }
291. }
292. void insert(struct node \*\*headptrptr1,int num)
293. {
294. struct node \*previous;*//先前的*
295. struct node \*current;*//现在的*
296. struct node \*news;*//要插入的*
297. previous = NULL;*//防止要插入的元素在第一位上*
298. current = \*headptrptr1;
299. while(current!=NULL&&current->count<num)
300. {
301. previous=current;
302. current=current->next;
303. }
304. news=(struct node\*)malloc(sizeof(struct node));
305. if(news==NULL)
306. {
307. return ;
308. }
309. news->count=num;
310. news->next=current;
311. if(previous==NULL)
312. {
313. \*headptrptr1=news;
314. }
315. else
316. {
317. previous->next=news;
318. }
319. }
320. void difference(struct node \*\*headptrptr1,struct node \*\*headptrptr2)
321. {
322. struct node \*temp1,\*temp2;
323. temp1=\*headptrptr1;
324. temp2=\*headptrptr2;
325. char a[100];
326. int count=0,flag=0;
327. printf("集合的差：");
328. while(temp1!=NULL&&temp2!=NULL)
329. {
330. if(temp2->count<temp1->count)
331. {
332. temp2=temp2->next;
333. }
334. else if(temp2->count==temp1->count)
335. {
336. temp1=temp1->next;
337. temp2=temp2->next;
338. }
339. else
340. {
341. for(int i=0;i<strlen(a);i++)
342. {
343. if(a[i]==temp2->count)
344. {
345. flag=1;
346. break;
347. }
348. }
349. if(flag==0)
350. {
351. a[count]=temp1->count;
352. count++;
353. }
354. temp1=temp1->next;
355. }
356. flag=0;
357. }
358. while(temp1)
359. {
360. a[count]=temp1->count;
361. count++;
362. temp1=temp1->next;
363. }
364. for(int i=0;i<strlen(a);i++)
365. printf("%c",a[i]);
366. printf("\n元素个数：%d\n",count);
367. printf("子集个数：%lf\n",pow(2,count));
368. }
369. void Complement(struct node \*\*headptrptr1)
370. {
371. LIST \*temp1;
372. temp1=\*headptrptr1;
373. int count=0;
374. for(int i=0;i<26;i++)
375. {
376. if(((temp1->count)>map[i])&&temp1)
377. {
378. printf("%c",map[i]);
379. count++;
380. }
381. else if(((temp1->count)==map[i])&&temp1)
382. {
383. temp1=temp1->next;
384. if(temp1==NULL)
385. {
386. for(int j=i+1;j<26;j++)
387. {
388. printf("%c",map[j]);
389. count++;
390. }
391. i=25;
392. }
393. }
394. }
395. printf("\n");
396. }