**数据结构第一次上机实验报告**

学号：161730126 姓名：李双玖

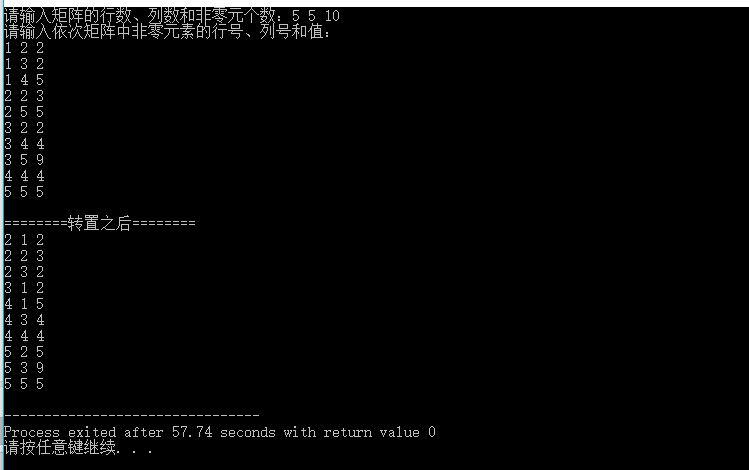
1. 调试成功程序及说明

**1、**

**题目：输入稀疏矩阵，建立稀疏矩阵三元组顺序结构，实现转置（方法一）。**

算法思想：将i从1加到矩阵的列数，每次遍历三元组，若行号相等就将其放到转置之后三元组的末尾。时间复杂度：O(T.col\*T.num)

运行结果：



结果分析：成功。

源程序：

1. #include<stdio.h>
3. **typedef** **struct**{
4. **int** data[100][3];
5. **int** row, col, num;  // 行数，列数，非零个数
6. }TSMatrix;
8. **int** main() {
9. TSMatrix T;
10. printf("请输入矩阵的行数、列数和非零元个数：");
11. scanf("%d%d%d", &T.row, &T.col, &T.num);
13. printf("请输入依次矩阵中非零元素的行号、列号和值：\n");
14. **for**(**int** i=0; i<T.num; i++) {
15. scanf("%d%d%d", &T.data[i][0], &T.data[i][1], &T.data[i][2]);
16. }
18. TSMatrix M;
19. M.col = T.row;
20. M.row = T.col;
21. M.num = T.num;
22. **for**(**int** col=1, j=0; col<=T.col; col++) {
23. **for**(**int** i=0; i<T.num; i++) {
24. **if**(T.data[i][1] == col) {
25. M.data[j][0] = T.data[i][1];
26. M.data[j][1] = T.data[i][0];
27. M.data[j][2] = T.data[i][2];
28. j++;
29. }
30. }
31. }
33. printf("\n========转置之后========\n");
34. **for**(**int** i=0; i<M.num; i++) {
35. printf("%d %d %d\n", M.data[i][0], M.data[i][1], M.data[i][2]);
36. }
37. }

**2、**

**题目：输入稀疏矩阵，建立稀疏矩阵三元组顺序结构，实现转置（方法二）。**

算法思想：

第一，定义一个长度为矩阵列数+1的数组temp，然后遍历三元组T，令temp[T.col]++；第二，i从1到temp的长度，令temp[i]+=temp[i-1]；第三，将temp数组中的每个元素向后移一位，最后得到的temp数组代表第i行的第一个非零元素的在置换后三元组M中的位置；第四，遍历三元组T，将每个元素置换到置换后三元组M的temp[T.col]位置，再让temp[T.col]++。时间复杂度：O(T.num\*2+T.num\*2)。

运行结果：



结果分析：成功。

源程序：

1. #include<stdio.h>
2. #include<stdlib.h>
4. **typedef** **struct**{
5. **int** data[100][3];
6. **int** row, col, num;  // 行数，列数，非零个数
7. }TSMatrix;
9. **int** main() {
10. TSMatrix T;
11. printf("请输入矩阵的行数、列数和非零元个数：");
12. scanf("%d%d%d", &T.row, &T.col, &T.num);
14. printf("请输入依次矩阵中非零元素的行号、列号和值：\n");
15. **for**(**int** i=0; i<T.num; i++) {
16. scanf("%d%d%d", &T.data[i][0], &T.data[i][1], &T.data[i][2]);
17. }
19. TSMatrix M;
20. M.col = T.row;
21. M.row = T.col;
22. M.num = T.num;
24. **int** \*temp = (**int**\*)malloc(**sizeof**(**int**)\*(M.row+1));
25. **for**(**int** i=0; i<=M.row; i++) temp[i] = 0;
27. **for**(**int** i=0; i<T.num; i++) {
28. temp[T.data[i][1]]++;
29. }
30. **for**(**int** i=2; i<=M.row; i++) {
31. temp[i] += temp[i-1];
32. }
33. **for**(**int** i=M.row; i>=1; i--) {
34. temp[i] = temp[i-1];
35. }
37. // 开始将T转置成M
38. **for**(**int** i=0; i<T.num; i++) {
39. **int** pos = temp[T.data[i][1]];
40. temp[T.data[i][1]]++;
41. M.data[pos][0] = T.data[i][1];
42. M.data[pos][1] = T.data[i][0];
43. M.data[pos][2] = T.data[i][2];
44. }
46. printf("\n========转置之后========\n");
47. **for**(**int** i=0; i<M.num; i++) {
48. printf("%d %d %d\n", M.data[i][0], M.data[i][1], M.data[i][2]);
49. }
50. free(temp);
51. }

**3、**

**题目：问题描述：首先输入正整数n（n<10000），接着输入n个正整数（最大值为10000），对于这n个数，统计输出其中的相邻数对（差值为1的数对），相同数据只被统计一次。**

**问题分析：这个看似是一个ｎ个数与n个数进行比较（O(n2)）的问题,能否用高效的方法解决？**

**样例输入**

**6  
1 3 8 2 5 2**

**样例输出**

**2  
样例输入**

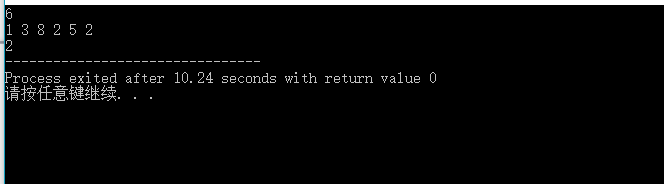
**5  
4 3 6 3 5 2**

**样例输出**

**4**

算法思想：用链表结构进行存储。输入一个就顺序插入到链表的相应位置。插入结束后，遍历链表，前后对比即能得到相邻数对个数。

运行结果：



结果分析：成功。

源程序：

1. #include<stdio.h>
2. #include<stdlib.h>
4. **typedef** **struct** LNode{
5. **int** data;
6. LNode \*next;
7. }\*LinkList;
9. **int** main() {
11. LinkList list = (LNode\*)malloc(**sizeof**(LNode));
12. list->next = NULL;
14. **int** n = 0;
15. //printf("请输入n：");
16. scanf("%d", &n);
18. LNode \*temp, \*p, \*pre;
19. **int** i;
20. **for**(i=0; i<n; i++) {
21. temp = (LNode\*)malloc(**sizeof**(LNode));
22. scanf("%d", &temp->data);
24. p = list->next;
25. pre = list;
26. **while**(p && p->data<temp->data) {
27. pre = p;
28. p = pre->next;
29. }
30. temp->next = p;
31. pre->next = temp;
32. }
34. LNode \*next;
35. p = list->next;
36. **int** account = 0;
37. **while**(p) {
38. next = p->next;
39. **if**(next && (next->data - p->data)==1) account++;
40. p = p->next;
41. }
42. printf("%d", account);
44. p = list;
45. **while**(p) {
46. temp = p;
47. p = p->next;
48. free(temp);
49. }
50. }

**4、**

**题目：问题描述：请实现一个铁路购票系统的简单座位分配算法，来处理一节车厢的座位分配。  
　　假设一节车厢有20排、每一排5个座位。为方便起见，我们用1到100来给所有的座位编号，第一排是1到5号，第二排是6到10号，依次类推，第20排是96到100号。  
　　购票时，一个人可能购一张或多张票，最多不超过5张。如果这几张票可以安排在同一排编号相邻的座位，则应该安排在编号最小的相邻座位。否则应该安排在编号最小的几个空座位中（不考虑是否相邻）。  
　　假设初始时车票全部未被购买，现在给了一些购票指令，请你处理这些指令。**

**输入格式：对于所有评测用例，1 ≤ n ≤ 100，所有购票数量之和不超过100。**

**输入的第一行包含一个整数n，表示购票指令的数量。  
　　第二行包含n个整数，每个整数p在1到5之间，表示要购入的票数，相邻的两个数之间使用一个空格分隔。**

**输出格式**

**输出n行，每行对应一条指令的处理结果。  
　　对于购票指令p，输出p张车票的编号，按从小到大排序。**

**问题分析：这个问题可以用顺序结构或链式结构实现。**

**样例输入**

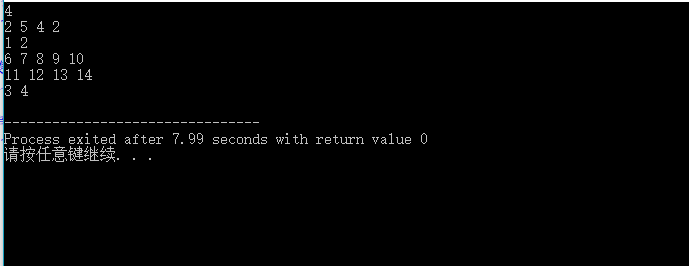
**4  
2 5 4 2**

**样例输出**

**1 2  
6 7 8 9 10  
11 12 13 14  
3 4**

算法思想：建一个二维数组a[20][6]，第一列用来存当前行还剩多少个位置，数组中其余位置赋0、1，0代表未被购票，1代表已有人购票。输入购买票数p，即将i从第一行到第二十行,遍历a[i][0]。若p<=a[i][0]，则将该行前面几个值为0的座位输出并置为1。若遍历结束，p都>a[i][0]，则将数组中值为0的座位都输出并置为1。时间复杂度为O(n\*(0-20))。

运行结果：



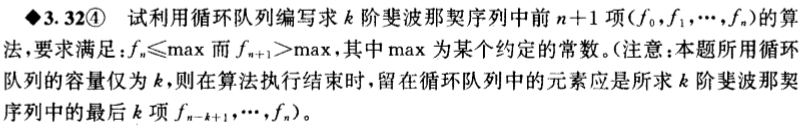
结果分析：成功。

源程序：

1. #include<stdio.h>
3. **int** main() {
4. **int** seat[20][5] = {0};
5. **int** surplus[20];    // 代表每排剩余的座位数
6. **for**(**int** i=0; i<20; i++) surplus[i] = 5;
8. **int** n = 0;
9. scanf("%d", &n);
10. **int** p = 0;
11. **for**(**int** i=0; i<n; i++) {
12. scanf("%d", &p);
13. **int** j = 0;  // j代表第j排
14. **while**(j<20) {
15. // 如果要购买的票数小于第j排的剩余票数
16. **if**(p <= surplus[j]) {
17. surplus[j] -= p;
18. **for**(**int** k=0; k<5 && p>0; k++) {
19. **if**(seat[j][k] == 0) {
20. seat[j][k] = 1;
21. p--;
22. printf("%d ",j\*5+k+1);
23. }
24. }
25. }
26. j++;
27. }
28. // 如果不存在相邻位置
29. **if**(j == 20) {
30. **for**(j=0; j<20; j++) {
31. **for**(**int** k=0; k<5 && p>0; k++) {
32. **if**(seat[j][k] == 0) {
33. seat[j][k] = 1;
34. p--;
35. printf("%d ",j\*5+k+1);
36. }
37. }
38. }
39. }
40. printf("\n");
41. }
42. }

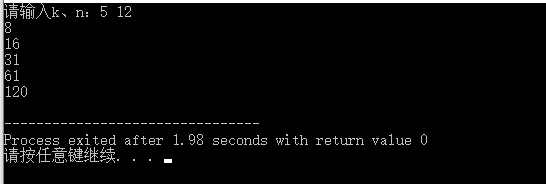
**5、**

**题目：**



算法思想：建一个长度为k的数组作为循环链表。将链表中前k-1位数赋值为0，第k个数赋值为1。定义一个数sum用来记前k项的和，初始为1。然后，每次将sum加到队尾，同时将sum减去队头数的值、加上队尾数的值，最后队尾++、队头++。执行结束即得到结果。时间复杂度O(n+1)。

运行结果：



结果分析：成功。

源程序：

1. #include<stdio.h>
2. #include<stdlib.h>
4. **typedef** **struct** {
5. **int** font, rear;
6. **int** length;
7. }Queue;
9. **int** main() {
10. printf("请输入k、n：");
11. **int** k = 0, n = 0;
12. scanf("%d%d", &k, &n);
13. **int** \*q = (**int**\*)malloc(4\*k);
15. Queue queue;
16. queue.font = queue.rear = queue.length = 0;
17. **int** sum = 0;
18. **for**(**int** i=0; i<=n; i++) {
19. **if**(i < k-1) {
20. q[queue.rear] = 0;
21. queue.rear++;
22. }
23. **if**(i == k-1) {
24. q[queue.rear] = 1;
25. queue.rear++;
26. sum = 1;
27. }
28. **if**(i > k-1) {
29. **int** temp = sum - q[queue.font%k];
30. queue.font++;
31. q[queue.rear%k] = sum;
32. //printf("-----------%d\n", sum);
33. queue.rear++;
34. sum += temp;
35. }
36. }
38. **for**(**int** i=queue.font; i<queue.rear; i++) {
39. //printf("================%d\n", i);
40. printf("%d\n", q[i%k]);
41. }
43. free(q);
44. }