**苏州大学 数据结构 课程试卷7卷（共3**页）

考试形式：闭卷 年 月

院系 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 年级 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 专业 \_\_\_\_\_\_\_\_\_\_\_\_\_\_

学号 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 姓名 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 成绩 \_\_\_\_\_\_\_\_\_\_\_\_\_\_

一、填空题（每题3分，共30分）

1、数据结构常用的存储结构分成两类： ， 。

2、在顺序表中插入或删除一个元素，平均需要移动 元素，具体移动元素个数与 有关。

3、以顺序实现的共享栈类含有私有数据成员：数组S[1..n]是存放两个栈的共享空间，top1和top2分别指向两个栈的栈顶位置，top1由小到大，top2由大到小，则判断栈空的条件是 ；判断栈满的条件是 。

4、哈希查找的装载因子是指： 。用开放定址法解决冲突，装载因子必须 。

5、已知一棵二叉树的中序遍历结果为DBHEAFICG，前序遍历结果为ABDEHCFIG，则该二叉树后序遍历的结果为 。

6、 二维数组A[10..20][5..10]采用行序为主方式存储，每个元素占4个存储单元，并且A[10][5]的存储地址是1000，则A[18][9]的地址是 。

7、对二叉排序树进行中序遍历，可以得到一个 　。

8、写出图的两种存储结构： 、 。写出图的两种遍历方法： 、 。

9、顺序查找n 个元素的顺序表，如查找成功，则比较关键字的次数最多为

　 　次；当使用监视哨时，若查找失败，则比较关键字的次数为

　 　 次。

10、当采用第一个记录作为枢轴，当初始关键字 时，快速排序的效率最差。当初始关键字 时，插入排序的效率最差。

二、应用题（每题10分，共40分）

1、对于一个栈，给出输入项为A，B，C。如果输入序列为A，B，C所组成，试给出全部可能的输出序列。

2、上三角矩阵（aij）n×n，将其上三角元素逐行存于数组B(1:m)中，（m充分大），使得B[k]= aij且k=f1(i)+f2(j)+c,。试推导出f1,f2和常数c（要求f1,f2中不含有参数项）。

3、已知有向图G=(V,E),其中

V={a,b,c,d,e,f,g}, E={<a,b>,<a,g>,<b,g>,<c,b>,<d,c>,<d,f>,<e,d>,<f,a>,<f,e>,<g,c>,<g,d>,<g,f>},

请画出图G并写出其邻接矩阵和邻接表表示。

4、散列表的地址空间为0..16，开始时散列表为空，用线性探测法处理冲突，对数据元素 Jan，Feb，Mar，Jun，Aug，Sep，Oct，Nov，Dec， 试构造相应的散列表，并求在等概率情况下成功查找的平均查找长度。H(key)=i mod 2，其中i为关键字的第一个字母在字母表中的序号。

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| A | B | C | D | E | F | G | H | I | J | K | L | M |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| N | O | P | Q | R | S | T | U | V | W | X | Y | Z |

三、算法设计题（每题10分，共30分）

1、指针la, lb分别指向两个不带头结点的单链表a 、b的第一个结点，试设计一个从表a中删除自第i个元素起共len个元素，并将这len个元素插入到表b中的i个元素之前的算法。

template <class List\_entry>

Error\_code move(List <List\_entry > la , List <List\_entry> lb)

2、假设用带头结点的单链表(采用第一种不带current指针的单链表类型)作为线性表的存储结构，编写删除线性表的所有值为x的元素的算法。请按以下函数原型进行设计。（12分）

Error\_code remove\_all\_x(const List\_entry &x, List a\_list)

3、二叉树采用二叉链表存储结构，试设计一个算法计算一棵给定二叉树的所有结点数。

template <class Entry>

int Binary\_tree<Entry> :: recursive\_nodecount(Binary\_node<Entry> \*sub\_root) const