一 作业题目

这个学期需要完成以下的全部作业习题。
作业习题列表:

第 2 章

1.在一个有序的线性表中插入一个元素，但要保持线性表的有序性。

（1）设线性表存储在数组 A[0..arrsize-1]的前 elenum 个单元中，且递增有序。试编写一个算法：在线性表中插入元素 x，需保持线性表的有序性，并且分析算法的时间复杂度。
输入:
存入数组的元素个数（以-1 结束整个程序的运行):
存入数组的数据（有序):
待插入的元素:
输出:
插入前的数组数据:
插入后的数组数据:

2.在一个线性表中找出一个最小值并删除之。
（1）利用顺序存储的方法来实现，可以直接采用数组来完成。
输入:
存入数组的元素个数（以-1 结束整个程序的运行):
存入数组的数据:
输出:
删除前的数组数据:
删除后的数组数据:

3.在一个有序的线性表中插入一个元素，但要保持线性表的有序性。
（1）线性表存储在单链表 L 中，且该单链表的结点是按值非递减有序排列的，试编写一算法在链表 L 中插入值为 X 的结点，使得 L 仍然有序。
输入:
存入链表的元素个数（以-1 结束整个程序的运行):
存入链表的数据（有序):

待插入的元素:
输出:
插入前的链表数据:
插入后的链表数据:

4.在一个线性表中找出一个最小值并删除之。
（1）利用循环链表的方法来实现。
输入:
存入链表的元素个数（以-1 结束整个程序的运行):
存入链表的数据:
输出:
删除前的链表数据:
删除后的链表数据:

5.设计递归函数，以实现单链表的逆置。

输入:
输入存放到链表的数据(以-1 来结束数据的输入，另外，作为整个程序运行的结束，也是通过输入-1 来实现)
输出:
逆置前的链表数据:
逆置后的链表数据:

第 5 章

1.一棵具有 n 个结点的完全二叉树存放在二叉树的顺序存储结构中，试编写非递归算法对该树进行中序遍历。
输入:
输入完全二叉树的元素个数（以-1 结束整个程序的运行):
输入二叉树的结点数据:
输出:
中序遍历序列

2.编写一个将二叉树的所有叶子结点从左向右链接成单链表的算法。
输入
（1）输入有多少棵二叉树要进行测试（以-1 结束整个程序的运行):
（2）分别输入扩充的二叉树前序遍历序列（整数值序列)，0 表示空子树。
输出:
单链表从头到尾的输出结果。

3.设计一个程序：输入后序遍历序列和中序遍历序列以生成二叉树二叉链式结构，并采用前序遍历序列输出该二叉树。

输入：

（1）输入有多少棵二叉树要进行测试（以-1 结束整个程序的运行）：

（2）输入二叉树的后序遍历序列

（3）输入二叉树中序遍历序列

输出：

二叉树的前序遍历序列

4.编写一个输出二叉树中所有左右子树高度差的绝对值为 2 的结点。二叉树的存储结构为二叉链式结构。

输入

（1）输入有多少棵二叉树要进行测试（以-1 结束整个程序的运行）：

（2）分别输入扩充的二叉树前序遍历序列（整数值序列），0 表示空子树。

输出：

输出左右子树高度差的绝对值为 2 的结点。

第 6、7 章

1.设计一个程序：实现判断该一棵二叉树是否为二叉查找树。假设二叉树采用二叉链式结构存储。

输入：

（1）输入有多少棵二叉树要进行测试（以-1 结束整个程序的运行）：

（2）输入扩充的二叉树前序遍历序列（整数值序列），0 表示空子树。

输出：是二叉查找树或不是二叉查找树

2.设计一个程序，实现二叉查找树的建立、查找、插入和删除操作。假设二叉树采用二叉链式结构存储。

输入：

（1）输入要建立的二叉查找树的结点个数（以-1 结束整个程序的运行）：

（2）输入建立二叉查找树的数据（整数值序列）

（3）输入一个待插入的元素

输出：

（1）查找上述输入（3）中所插入的元素的结果(如果存在则输出查找成功，否则输出查找失败)

（2）删除上述输入（3）中所插入的元素，并输出再次查找该元素的结果。

第 8 章

1.写出以邻接矩阵为图（不带权的有向图或无向图）的存储结构，分别实现广度优先遍历和深度优先遍历的程序。
输入：
（1）图的结点总数，以输入 -1 来结束整个程序的运行。
（2）邻接矩阵的每行数据（整数）
（3）输入遍历的出发点序号
输出：
（1）广度优先遍历序列
（2）深度优先遍历序列

2.写出以邻接表为图（不带权的有向图或无向图）的存储结构，分别实现广度优先遍历和深度优先遍历的程序。
输入：
（1）图的结点总数，以输入 -1 来结束整个程序的运行。
（2）分别输入每个结点的邻接点序号，以-1 作结束
（3）输入遍历的出发点序号
输出：
（1）广度优先遍历序列
（2）深度优先遍历序列

3.写出以邻接矩阵为无向带权图的存储结构，分别实现卡鲁斯卡尔和普里姆求最小生成树的程序。
输入：
（1）图的结点总数，以输入 -1 来结束整个程序的运行。
（2）邻接矩阵的每行数据（整数）
输出：
（1）卡鲁斯卡尔求最小生成树的结果序列
（2）普里姆求最小生成树的结果序列

4.写出以邻接矩阵为有向带权图的存储结构，实现求从一源点到其他各个结点的最短路径。
输入：
（1）图的结点总数，以输入 -1 来结束整个程序的运行。
（2）邻接矩阵的每行数据（非负整数）
（3）输入源点对应的结点序号
输出：
（1）从源点到其他各个结点的最短路径值

5.写出以邻接表为有向不带权图的存储结构，实现求拓扑序列的程序。
输入：
（1）图的结点总数，以输入 -1 来结束整个程序的运行。
（2）分别输入每个结点所邻接到的邻接结点序号，以-1 作结束
输出：
（1）如果图不存在回路，则输出拓扑序列结果
（2）如果图存在回路，则输出“该图存在回路”

第 9 章

1.设计一个程序：用冒泡排序的递归方法实现单链表的排序功能
输入：
（1）以 0 作为输入结束的若干个整数数据。
（2）以输入 -1 来结束整个程序的运行。
输出：
（1）分别在不同行输出排序前后的的序列。

2.设计一个程序：分别用快速排序递归算法和非递归算法进行排序功能。

输入：
（1）以 0 作为输入结束的若干个整数数据。
（2）以输入 -1 来结束整个程序的运行。

输出：
（1）分别在不同行输出快速排序递归算法排序前后的的序列。
（2）分别在不同行输出快速排序非递归算法排序前后的的序列。

二 作业评分标准参考与范例

例题：数制转换问题
设计一个算法将一个十进制数转换为 D 进制数。(可以只考虑整数的情形)
十进制数 N 和其他 D 进制数转换的简单算法基于原理：
N = (N / D)×D + N MOD D，其中：MOD 为求余运算（取模），D 为数制。
1、描述算法的基本设计思想；(描述算法的详细实现步骤)；
2、依据设计思想，用 C 或 C++语言描述算法，关键之处给出解释；

3、说明你所设计算法的时间和空间复杂度；
4、测试你的算法，如果发现错误请说明原因并给出解决方法或想法。
解答

1 设计思想：20 分

计算过程是从低位到高位的顺序产生 D 进制数的各个数位；
打印输出应该从高位到低位进行，恰好和计算过程相反。
如果将计算中得到的 D 进制数各位顺序进栈，则按出栈序列打印输出的即为与输入对应的 D 进制数。这是利用栈的“后进先出”操作特性的最简单例子。

2 用 C 或 C++语言描述算法如下：50 分

```
#define MaxSize 100
int stack[MaxSize];//顺序栈
int top = -1;//栈顶，构造空顺序栈 stack
void conversionD(void) {// 对于输入的任意非负十进制整数，打印输出与其等值的其他进制数
        int n,D,k,e;
        cout<<"Input a number to convert:\n";
        cin>>n;// 输入任意非负十进制数 n
        cout<<"Input a base number to convert:\n"
        cin>>D; // 输入任意基数 D
        if (n < 0) {
            cout<<"\nThe number must be over 0.";
            return;
        }

        if (n == 0) stack[++top]=0;
        //while (n != 0) {// 从低到高位产生 D 进制数各个数位
        while (n) {
            k = n % D;
            //stack[++top]=n%D;
                                                    if(top<MaxSize){
            stack[++top] = k;// 将 N 取模 D 后入栈 stack
            //cout << k;
            n = n/D;// 将 N 整除 D }
            //if (n == 0) break;
            else{
              cout<<"stack is full!"<<endl;
            break;
            }
        }
```

```
        cout<<"the result is: ";
        while (top != -1) {//栈不空，// 从高到低位输出 D 进制数各个数位
                                        //while( !(top = -1)) {
            e = stack[top--];// 将转换后的 D 进制数出栈 stack
            cout<<e;// 输出
                                                    //cout<<stack[top--];
        }
    }// conversionD
```

3 时间和空间复杂度：10 分

①问题规模：待转换的非负“十进制数”N 和“基”D；
② 基本操作：入栈、出栈操作；
③ 时间分析：算法执行时间主要花费在两个 while 循环基本操作上，两个 while 循环的执行次数均为 $\log_D N$，因此算法的时间复杂度为 $O(\log_D N)$。
空间复杂度：采用了临时变量 k 和 e（可以不采用），O（1）；另外利用了一个栈，实际栈的大小为 $\log_D N$。

4 测试 10 分

使用自定义的数据结构或本例的直接编写方法。本例可以直接进行测试。
测试范例
例 1 Input a number to convert:
35
Input a base number to convert:
2
the result is: 100011

例 2Input a number to convert:
4
Input a base number to convert:
2
the result is: 100
测试的两例整数正确
例 3 测试负数
Input a number to convert:
-4
Input a base number to convert:
2

The number must be over 0.
例 4 测试 0

```
Input a number to convert:
0
Input a base number to convert:
2
the result is: 0
```

5 总结与改进措施 10 分