第二次研讨课报告

一、 二叉排序树

a) 请给出一颗二叉排序树

b) 利用二叉排序树得到一个递升数列

c) 利用二叉排序树得到一个递减数列

1、 研讨

二叉查找树，也称为二叉搜索树、有序二叉树（ordered binary tree）或排序二叉树（sorted binary tree），是指一棵空树或者具有下列性质的二叉树：

i. 若任意节点的左子树不空，则左子树上所有节点的值均小于它的根节点的值；

ii. 若任意节点的右子树不空，则右子树上所有节点的值均大于它的根节点的值；

iii. 任意节点的左、右子树也分别为二叉查找树；

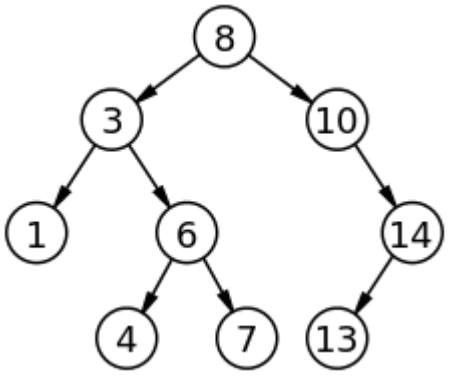
iv. **没有键值相等的节点。**

二叉查找树相比于其他数据结构的优势在于查找、插入的时间复杂度较低。为 O(log n)。二叉查找树是基础性数据结构，用于构建更为抽象的数据结构，如集合、多重集、关联数组等。

二叉查找树的查找过程和次优二叉树类似，通常采取二叉链表作为二叉查找树的存储结构。**中序遍历二叉查找树可得到一个关键字的有序序列**，一个无序序列可以通过构造一棵二叉查找树变成一个有序序列，构造树的过程即为对无序序列进行查找的过程。每次插入的新

的结点都是二叉查找树上新的叶子结点，**在进行插入操作时，不必移动其它结点，只需改动某个结点的指针，由空变为非空即可。搜索、插入、删除的复杂度等于树高，期望 O(log n)，最坏 O(n)（数列有序，树退化成线性表）。**

对于问题 a，给出二叉排序树如下：



对于问题 b，通过中序遍历就可以得到一个递增数列。由于二叉排序树的左子节点永远小于根节点永远小于右子节点，可以根据这个顺序进行遍历。对于上图得到的遍历顺序为：1，3，4，6，7，8，10，13，14；

对于问题 c，通过右子树-根节点-左子树的顺序遍历就可以得到一个递减数列。对于上图得到的遍历顺序为：14，13，10，8，7，6，4，3，1。

2、 课后思考

二叉排序树，二叉树的一个变种，主要的特点在于，该树的值在分布的时候具有非常明显的特征，左子树的值小于根节点的值，而根节点的值小于右子树的值，这在进行搜索，查找的时候是非常有利的，

因为它的平均操作时间接近 O(h)，h 为树的高度，而且，二叉排序树本身是具有动态性的，可以动态地进行节点的删除，插入等的操作。

下面对二叉排序树的各种操作进行总结：

i. 数据结构

```
/* 二叉树的二叉链表结点结构定义 */
typedef  struct BiTNode    /* 结点结构 */
{
    int data;    /* 结点数据 */
    struct BiTNode *lchild, *rchild; /* 左右孩子指针 */
} BiTNode, *BiTree;
```

ii. 查找算法

在二元排序树 b 中查找 x 的过程为：

1.若 b 是空树，则搜索失败，否则：

2.若 x 等于 b 的根节点的数据域之值，则查找成功；否则：

3.若 x 小于 b 的根节点的数据域之值，则搜索左子树；否则：

4.查找右子树。

```
Status SearchBST(BiTree T, int key, BiTree f, BiTree *p)
{
    if (!T)    /*  查找不成功 */
    {
        *p = f;
        return FALSE;
    }
    else if (key==T->data) /*  查找成功 */
    {
        *p = T;
        return TRUE;
    }
    else if (key<T->data)
        return SearchBST(T->lchild, key, T, p);  /*  在左子树中继续查找 */
    else
        return SearchBST(T->rchild, key, T, p);  /*  在右子树中继续查找 */
}
```

iii. 插入算法

利用查找函数，将关键字放到树中的合适位置。

```
Status InsertBST(BiTree *T, int key)
{
    BiTree p,s;
    if (!SearchBST(*T, key, NULL, &p)) /* 查找不成功 */
    {
        s = (BiTree)malloc(sizeof(BiTNode));
        s->data = key;
        s->lchild = s->rchild = NULL;
        if (!p)
            *T = s;           /*  插入 s 为新的根结点 */
        else if (key<p->data)
            p->lchild = s;    /*  插入 s 为左孩子 */
        else
            p->rchild = s;  /*  插入 s 为右孩子 */
        return TRUE;
    }
    else
        return FALSE;  /*  树中已有关键字相同的结点，不再插入 */
}
```

iv. 删除算法

在二叉排序树中删去一个结点，分三种情况讨论：

1.若*p 结点为叶子结点，即 PL(左子树)和 PR(右子树)均为空树。由于删去叶子结点不破坏整棵树的结构，则只需修改其双亲结点的指针即可。

2.若*p 结点只有左子树 PL 或右子树 PR，此时只要令 PL 或 PR 直接成为其双亲结点*f 的左子树（当*p 是左子树）或右子树（当*p 是右子树）即可，作此修改也不破坏二叉排序树的特性。

3.若*p 结点的左子树和右子树均不空。在删去*p 之后，为保持其它元素之间的相对位置不变，可按中序遍历保持有序进行调整。比较好的做法是，找到*p 的直接前驱（或直接后继）*s，用*s 来替换结点*p，然后再删除结点*s。

```
Status DeleteBST(BiTree *T,int key)
{
    if(!*T) /* 不存在关键字等于key 的数据元素 */
        return FALSE;
    else
    {
        if (key==(*T)->data) /* 找到关键字等于key 的数据元素 */
            return Delete(T);
        else if (key<(*T)->data)
            return DeleteBST(&(*T)->lchild,key);
        else
            return DeleteBST(&(*T)->rchild,key);

    }
}

/* 从二叉排序树中删除结点p，并重接它的左或右子树。 */
Status Delete(BiTree *p)
{
    BiTree q,s;
    if((*p)->rchild==NULL) /* 右子树空则只需重接它的左子树（待删结点是叶子也走此分支)
*/
    {
        q=*p; *p=(*p)->lchild; free(q);
    }
    else if((*p)->lchild==NULL) /* 只需重接它的右子树 */
    {
        q=*p; *p=(*p)->rchild; free(q);
    }
    else /* 左右子树均不空 */
    {
        q=*p; s=(*p)->lchild;
        while(s->rchild) /* 转左，然后向右到尽头（找待删结点的前驱） */
        {
            q=s;
            s=s->rchild;
```

```
            }
            (*p)->data=s->data; /*  s 指向被删结点的直接前驱（将被删结点前驱的值取代被删结点的值） */
            if(q!=*p)
                q->rchild=s->lchild; /*  重接 q 的右子树 */
            else
                q->lchild=s->lchild; /*  重接 q 的左子树 */
            free(s);
        }
        return TRUE;
    }
```

v. 性能分析:

每个结点的 Ci 为该结点的层次数。最好的情况是二叉排序树的形态和折半查找的判定树相同，其**平均查找长度和 logn 成正比（O(log2(n)))**。最坏情况下，当先后插入的关键字有序时，构成的二叉排序树为一棵斜树，树的深度为 n，其平均查找长度为(n + 1) / 2。也就是时间复杂度为 O(n)，等同于顺序查找。**因此，如果希望对一个集合按二叉排序树查找，最好是把它构建成一棵平衡的二叉排序树（平衡二叉树）。**

二、 树形结构在文件管理中的应用

a) **怎样利用树形结构来管理文件目录，并能够将文件和文件夹加以区分**

b) **如何统计一个节点下文件夹和文件的数目**

c) **从目录树的管理上看，要实现文件夹和文件的删除、复制、移动。请描述算法实现的思路。**

d) **地址路径和目录树结构怎么映射?**

1、 研讨

a) 对于问题 1，可以将每一个文件夹和文件储存为树的结点，为了区分每个结点是文件夹还是文件，**使用一个标志位进行区分**。由此可以得到一个普遍意义的树，即森林。但这样存在的问题是，森林并不是进行文件管理的最优方案，二叉树无论在各项操作上都体现出更强的管理能力。

所以我们将森林转换为二叉树的形式进行储存和管理。使用孩子兄弟表示法，每个节点的左子树连接到实际存在的子节点，右子树连接到该节点的兄弟。

除了上述的增加标志位的方法，**我们同时提出一种增加虚拟节点的方法继续管理**。因为每一个叶节点可以是文件，也可以是空的文件夹。我们可以效仿链表头结点的方式，在每个文件夹结点的左子树上都先增加一个用于管理的“头结点”，该头结点是无意义的。然后在这个节点下进行管理。这样一来，每个文件夹必定不是叶节点，而每个叶结点的有意义项均为文件。

b) 对于问题 2.如果我们使用了标志位的储存方法，就对树进行遍历，统计不同标志位的个数即可。

如果我们采用了头结点的方法，统计每个叶节点的有意义项的数量即是文件的数量，统计结点的数量减去叶结点的数量即是文件夹的数量。

c) 对于问题 3

i. 文件夹的删除

寻址到目标文件夹，将该结点的右子节点连接到该节点的前

驱结点，然后清空所有左子树的空间。

ii. 文件的删除

寻址到目标文件，**将该结点的右子节点连接到该节点的前驱结点**，然后清空该结点的空间。

iii. 文件夹的复制

寻址到目标文件夹，申请足够大的空间，将该结点的左子树全部赋值到新的空间中去。然后寻址到目标位置，**若该结点的左子树为空，直接将左子节点连接到新空间，否则，一直遍历右子树，直到右子树为空，将右子节点连接到新空间。**

iv. 文件的复制

寻址到目标文件，申请足够大的空间，将该结点赋值给新空间。将该结点的右子树全部链接到前一个结点。然后寻址到目标位置，若该结点的左子树为空，直接将左子节点连接到新空间，否则，一直遍历左子树的右子树，直到某节点左子树为空，将左子节点连接到新空间。

v. 文件夹的移动

移动可以理解为两个步骤：复制和删除。操作步骤如上。先复制，再删除原有的文件夹。

vi. 文件的移动

移动可以理解为两个步骤：复制和删除。操作步骤如上。先复制，再删除原有的文件。

d) 对于问题 4，给出地址直接顺序读取，每一个分隔符之间的即为

结点存储的值，依次遍历即可得到位置。如果给定了位置，从该结点回溯至根节点，逆序输出各节点的值即可，可以使用栈的方式储存输出。

2、 课后思考

目前树形结构的文件储存方式十分常见，为了检验设想的正确性，本文进入了本机的 cmd ，使用了 tree >list.txt 命令打印了 E:/demo 下的树形目录。(部分如下)：

```
卷 Code 的文件夹 PATH 列表
卷序列号为 7AF0-C066
E:.
├─Data-structure
|  ├─Chapter_1
|  ├─Chapter_2
|  |  ├─2-1
|  |  |  └─Debug
|  |  |      └─2-1.tlog
|  |  ├─2-10
|  |  |  └─Debug
|  |  |      └─2-10.tlog
|  |  ├─2-2
|  |  |  └─Debug
|  |  |      └─2-2.tlog
|  |  ├─2-3
|  |  |  └─Debug
|  |  |      └─2-3.tlog
|  |  ├─2-4
|  |  |  └─Debug
|  |  |      └─2-4.tlog
|  |  ├─2-5
|  |  |  └─Debug
|  |  |      └─2-5.tlog
```

三、 有一千万条短信，有重复，以文本文件（ASCII）的形式保存，一行一条，请找出重复最多的前 10 条。

1、 研讨

a) 建立一个字典树。不必使用二叉树的方法进行储存，储存每条短信的公共前缀。每个叶节点的后面增加一个储存出现次数的结点。每读到一次这个节点，该次数就加 1。最后遍历整个树，找到最大的前 10 个。

b) 可以先排序，再遍历一次，记载重复次数最多的 10 个，但这个算法最快也就是 O（nlgn）。

2、 课后思考

a) 可以使用哈希表对一千万条分成若干组边扫描边建散列表。第一次扫描取首字节尾字节，中间随便取两个作为 Hash Code,插入到哈希表中。记录其地址和重复次数，和 hash code 等长的就疑似是相同。相同记录只加一次进入到哈希表，但将重复次数加 1.再进行第二次哈希处理。在 O(n)内即可完成。

b) 使用内存映射。以为你短信的长度不会太大。对每条短信的第 i 个字母用 ASCII 码进行分组，就是创建树的深度进行遍历。

四、 感想与建议

1、 此次研讨课学习到了不少新的知识，比如二叉排序树等，同时新颖的应用题也让我有了新的尝试，对一些更偏实际的算法有了了解。

2、 自由讨论的氛围很棒，只是可能这一次的时间不是很充足。