

《数据结构》上机报告

2018 年 12 月 21 日

姓名： 赵得泽 学号： 1753642 班级： 电子 2 班 得分： _____

实验题目	查找（二分法、二叉排序树）	
问题描述	<p>二分法将所有元素所在区间分成两个子区间，根据计算要求决定下一步计算是在左区间还是右区间进行；重复该过程，直到找到解为止。</p> <p>二叉排序树（二叉查找树）或者是一棵空树，或者是具有下列性质的二叉树：（1）每个结点都有一个作为查找依据的关键字(key)，所有结点的关键字互不相同。</p> <p>（2）左子树(若非空)上所有结点的关键字都小于根结点的关键字。</p> <p>（3）右子树(若非空)上所有结点的关键字都大于根结点的关键字。</p> <p>（4）左子树和右子树也是二叉排序树。</p>	
基本要求	<p>1. 本题给定已排序的一组整数，包含重复元素，请改写折半查找算法，找出关键字 key 在有序表中出现的第一个位置（下标最小的位置），保证时间代价是 $O(\log n)$。若查找不到，返回-1。</p> <p>2. 本题通过输入一个数据序列，创建查找表，完成基本操作，并计算等概率情况下，查找成功的平均查找长度。</p>	
	已完成基本内容（序号）：	1, 2
选做要求		
	已完成选做内容（序号）	
数据结构设计	<pre>typedef struct { int *elem; int length; } SSTable; typedef struct { int key; } ElemType; typedef struct BiTNode { ElemType data; BiTNode *lchild, *rchild; int depth; } BiTNode, *BiTree;</pre> <p>本次实验用到的数据结构主要是二叉树和顺序表；</p> <p>二叉树是 n ($n \geq 0$) 个结点的集合，它或为空树，或是由一个称之为根的结点加上两棵分别称为左子树和右子树的互不相交的二叉树组成。它的指针域包括*lchild 和 *rchild, 这两个指针域分别指向一个结点的左右孩子结点，还有结点存储的数据 data, 这是结点要存储的数据。这样其实也就构成了一个二叉链表。</p> <p>二分查找主要用到的是顺序表*elem 用来存储待查找的序列。其中还包括表长 length;</p>	

功能(函数)说明

```

/*****
函数功能：二分查找
返回值：返回待查找元素的第一次出现的位置
函数说明：见下代码注释区域。
*****/
int Search_Bin(SSTable &ST, int key, int n)
{
    int mid, high, low;
    ST.length = n;
    low = 0; //数组头
    high = ST.length - 1; //数组尾
    int flag = 0; //是否查找到的标志
    while (low <= high)
    {
        mid = (high + low) / 2; //二分位置
        if (ST.elem[mid] == key) //二分位置元素待查找关键字相等
        {
            while (1)
            {
                /*如果前一个始终和关键字相等，一直向前循环遍历，并且保证mid>0*/
                if (ST.elem[mid - 1] == key && mid > 0)
                    mid--;
                else
                    break;
            }
            flag = 1; //关键字存在
            return mid; //返回关键字第一次出现的位置
        }
        else if (ST.elem[mid] > key) //二分位置元素比待查找关键字大
            high = mid - 1; //在二分位置的左侧序列中查找
        else
            low = mid + 1; //在二分位的右侧序列中查找
    }
    if (flag == 0)
        return -1; //关键字不存在，返回-1;
}

/*****
函数功能：在二叉排序树查找关键字
函数说明：利用二叉排序树左孩子关键值<根节点<右孩子进行查找，
具体过程见代码注释区域
*****/
int SearchBST(BiTree &T, int key)
{
    BiTree p = NULL;

```

```

if (!T)
{
    p = T;
    while (p)
    {
        if (p->data.key == key)//根节点的关键字和待查找关键字相等
            break;//查找到了，退出循环
        else if (p->data.key > key)//根节点的关键字>待查找关键字
            p = p->lchild;//在右子树上面查找
        else//根节点的关键字<待查找关键字
            p = p->rchild;//在左子树上查找
    }
}
if (p == NULL)
    return 0;//没找到
else
    return 1;//找到了
}
/*****
函数功能：在二叉排序树上插入关键字节点
函数说明：具体过程见下代码注释区域
*****/
void InsertBST(BiTree &T, BiTree S)
{
    BiTree p, q = NULL;
    if (!T) {
        T = S;
    }
    else
    {
        p = T;//p指向根节点
        while (p)
        {
            q = p;
            if (p->data.key > S->data.key)
                p = p->lchild;//如果待插入结点比根节点的关键值小，在左子树上插入
            else
                p = p->rchild;//如果待插入节点比根节点的关键值大，在右子树上插入
        }
        if (q->data.key > S->data.key)//当p空的时候，如果其双亲节点q的关键值大于待插入
        关键值
            q->lchild = S;//则插入到双亲q的左孩子
        else
            q->rchild = S;//否则，插入到双亲q的右孩子
    }
}

```

```

    }
}

/*****
函数功能：创建二叉排序树
函数说明：将输入的数字创建一个节点，关键值等于输入的数字，然后
左右孩子置空，最后再将创建的结点S插入到二叉树T中；
*****/

void CreateBST(BiTree &T)
{
    BiTree S;
    int x, num = 0;
    while (num < n)
    {
        cin >> x;
        S = new BiTNode;
        S->data.key = x;
        S->lchild = NULL;
        S->rchild = NULL;
        InsertBST(T, S);
        num++;
    }
}

/*****
函数功能：在二叉排序树上删除节点
函数说明：核心：它的左子树中寻找中序下的最后一个结点(关键字最大)，
用它的值填补到被删结点中；具体步骤见下代码注释。
*****/

int DeleteNode(BiTree &p)
{
    BiTree s, q;
    int flag = 0; //是否删除成功标志
    if (!p->lchild)
    { /*如果待删除结点的左孩子为空, 将结点p赋给辅助变量q,
      p指向它的右孩子, 即将右子树挂在双亲的节点p上, 再删除q即可*/
        q = p;
        p = p->rchild;
        flag = 1;
        delete q;
    }
    else if (!p->rchild)
    { /*如果待删除结点的右孩子为空, 将结点p赋给q, p指向它的
      左孩子, 即将左子树挂在双亲的节点p上, 再删除q即可*/
        q = p;
        p = p->lchild;
        flag = 1;
    }
}

```

```

        delete q;
    }
    else
    {
        /*待删除节点的左右孩子都不为空，
        可以在它的左子树中寻找中序下的最后一个结点(关键字最大)，
        用它的值填补到被删结点中，再来处理这个结点的删除问题。*/

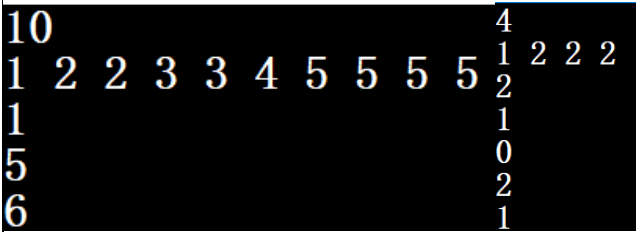
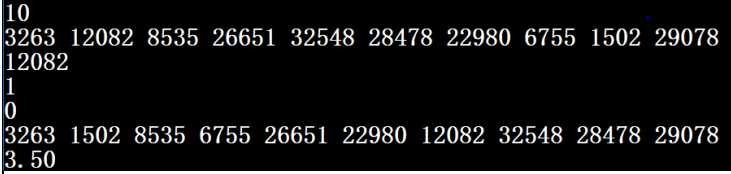
        q = p;
        s = p->lchild;
        while (s->rchild)
        {
            q = s;
            s = s->rchild;
        }
        p->data = s->data; //中序下的最后一个结点的值赋给待删除节点
        if (p != q) //若待删除节点的左子树根节点有右子树
            q->rchild = s->lchild; //则将中序下最后一个节点的左子树挂到其双亲节点右孩子
        else //若待删除节点的左子树根节点没有右子树
            q->lchild = s->lchild; //则将该根节点的左子树挂到待删除节点的左子树;
        flag = 1;
        delete s; //删除找到的中序下最后一个节点
    }
    if (flag == 1)
        return 1;
    else
        return 0; //没删除成功
}

/*****
函数功能：在二叉排序树上查找要删除的关键值并删除
函数说明：递归查找，若是根节点值<待查找的关键值，在根节点右子树查找
（递归），若是根节点值>待查找的关键值，在左子树查找（递归），若相等，则
解决删除该节点的问题。
*****/

int DeleteBST(BiTree &T, int key) {
    if (!T) return 0;
    else {
        if (key == T->data.key) { return DeleteNode(T); }
        else if (key < T->data.key) return DeleteBST(T->lchild, key);
        else return DeleteBST(T->rchild, key);
    }
}

/*****
函数功能：计算每个节点高度
函数说明：利用逆中序递归，每次递归Depth+1；递归结束即可计算出每个
结点的高度；
*****/

```

	<pre>void NodeDepth(BiTree &T, int Depth) { if (T) { NodeDepth(T->rchild, Depth + 1); T->depth = Depth; NodeDepth(T->lchild, Depth + 1); } } /***** 函数功能：计算总查找长度 函数说明：利用先序递归遍历整棵树，每次都访问结点的高度，再相加， 最后计算出的高度之和就是总查找长度。 *****/ int SumSearchLength(BiTree &T) { if (!T) return sum; else { sum += T->depth; SumSearchLength(T->lchild); SumSearchLength(T->rchild); } }</pre>
开发环境	Win10, vs2017, C++高级程序设计
调试分析	<p>Problem1:</p>  <p>Problem2:</p> 

	<pre> 5 12 23 343 65 123 10 0 0 12 10 23 343 65 123 3. 40 </pre>
心得体会	<p>本次实验主要是查找的应用——二分查找和二叉排序树的查找；</p> <p>在本次运用中，我体会到了二分查找的便捷性，以及在它变形运用的基础上，真正掌握了它的内在意义。</p> <p>主要则是二叉排序树，二叉排序树是以二叉树为载体进行排序的——将比根节点大的放到右子树，小的放在左子树，这样一来查找就有了目的性，从而也大大提高了查找的效率。其中包括各种基本操作，删除节点、插入结点、查找关键值、计算平均查找长度等等。在我看来比较复杂的就是如何删除某个特定节点，在这个过程中，我们要考虑待删除节点是否有左右子树，以及在有左子树/右子树，或者左右子树都有的情况下如何删除，前两种比较简单，难点出现在第三种情况上——左右子树都有，在此提出的解决办法是：找到待删除节点的左子树在中序遍历下的最后一个节点（关键值最大）；或找出右子树在中序遍历下的第一个节点（关键值最小）。这样又回到了中序遍历。接着将找到的节点的关键值与待删除节点的关键值替换，再删除找到的节点，这样就避免了大费周章的去删除待删除节点。</p> <p>总之，两种查找都有着同样的核心意义——二分，让查找都有了目的性，这样便大大提高了查找的效率。</p>