

《数据结构》上机报告

2018 年 12 月 10 日

姓名：刘思源 学号：1651390 班级：电子三班 得分：_____

实验题目	查找
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。二分法的计算效率是 $O(\log n)$，在已知的很多算法中都采用了二分法，例如：折半查找，快速排序，归并排序等。</p> <p>(二叉排序树) 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。（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。（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。（4）左子树和右子树也是二叉排序树。</p>
基本要求	<p>1. (p1) 折半查找要求查找表是有序排列的，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. (p2) 二叉排序树的基本操作集包括：创建、查找，插入，删除，查找最大值，查找最小值等。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>
	<div>已完成基本内容（序号）：</div> <div>1, 2</div>
选做要求	<div>已完成选做内容（序号）</div>
数据结构设计	<pre> class OrderTNode{ public: ElemType data; OrderTNode *lchild; OrderTNode *rchild; OrderTNode() { lchild = NULL; rchild = NULL; } }; </pre>

功能(函数)说明	<p>1、折半查找</p> <p>折半查找即二分法查找，时间复杂度为 $O(\log n)$。前提要求给出的数字序列必须为有序数列。主要思想是将比较的数和当前数列的中值进行比较，如果小于该值，就将当前的比较数列减半，只对较小的一部分加入到后续比较；如果偏大，就将偏大的一部分加入到后续比较。</p> <p>一般的二分查找不能返回到第一次出现的查找位置，本题应该额外加入一个循环，当找到某个值符合条件时，将其向前回溯，直到回溯的值不是我们想要寻找的值，或者回溯到比较数列的最低处。</p> <pre> int halfIntervalSearch (ElemType arr[], ElemType tar, int n) { int low = 0, high = n - 1; int mid, i; while (high - low > 1) { mid = (high + low) / 2; if (arr[mid] == tar) { for (i = mid; i >= low; i--) { if (arr[i] == arr[mid]) { mid = i; } } return mid; } else if (arr[mid] < tar) { low = mid; continue; } else { high = mid; continue; } } return -1; } </pre> <p>2、寻找二叉排序树元素</p> <p>二叉排序树的特点在于左子树的所有节点均小于根节点的值，右子树的所有节点均大于根节点的值。可以根据这一特性进行查找。当寻找的节点不是空时，如果要查找的值小于当前节点的值，就将我们的寻址节点赋给左子节点；反之赋给右子节点。若没找到，返回 NULL。</p>
----------	--

```

OrderTNode *OrderT::Search(ElemType value, OrderTNode *&p)
{
    OrderTNode *q = NULL;
    q = root;
    while (q) {
        p = q;
        if (q->data == value) {
            return p;
        }
        else if (q->data > value) {
            q = q->lchild;
        }
        else {
            q = q->rchild;
        }
    }
    return NULL;
}

```

3、插入二叉排序树

首先寻找该二叉树中是否存在该结点，如果没找到，那么返回的指针的值一定是最与当前需要插入的值接近的。比较这个节点的值和插入节点的值，直接插入为左孩子或者右孩子。

```

void OrderT::Insert(ElemType value)
{
    OrderTNode *p = NULL, *q;
    q = root;
    if (Search(value, p) == NULL) {
        OrderTNode *ins = new OrderTNode;
        ins->data = value;
        if (q == NULL) {
            root = ins;
            return;
        }
        if (p->data < value) {
            p->lchild = ins;
        }
        else if (p->data > value) {
            p->rchild = ins;
        }
    }
}

```

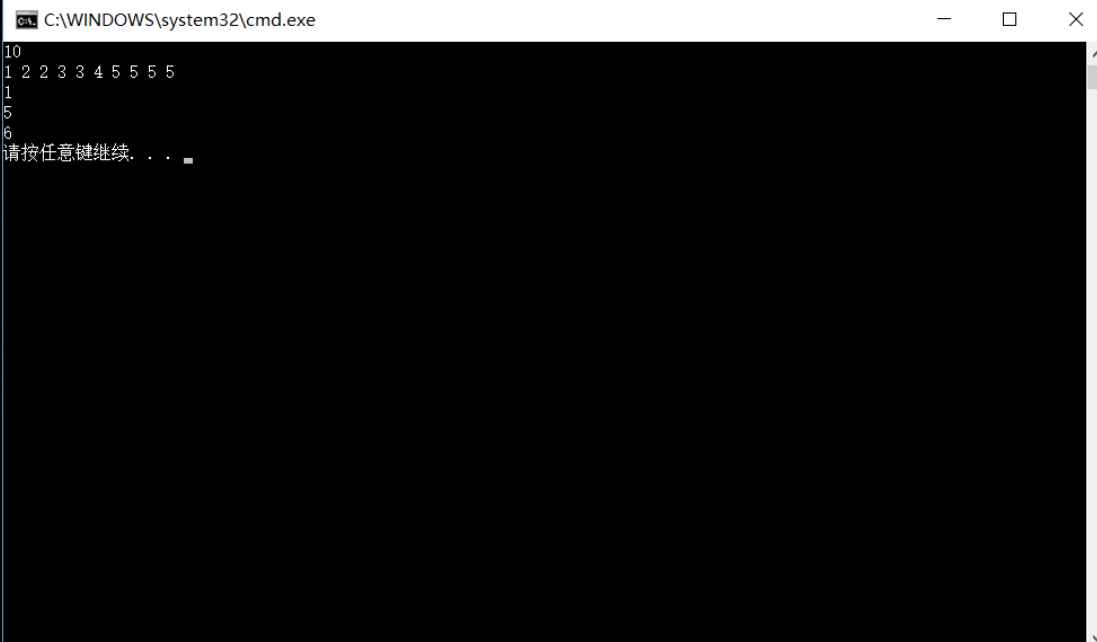
4、删除二叉排序树节点

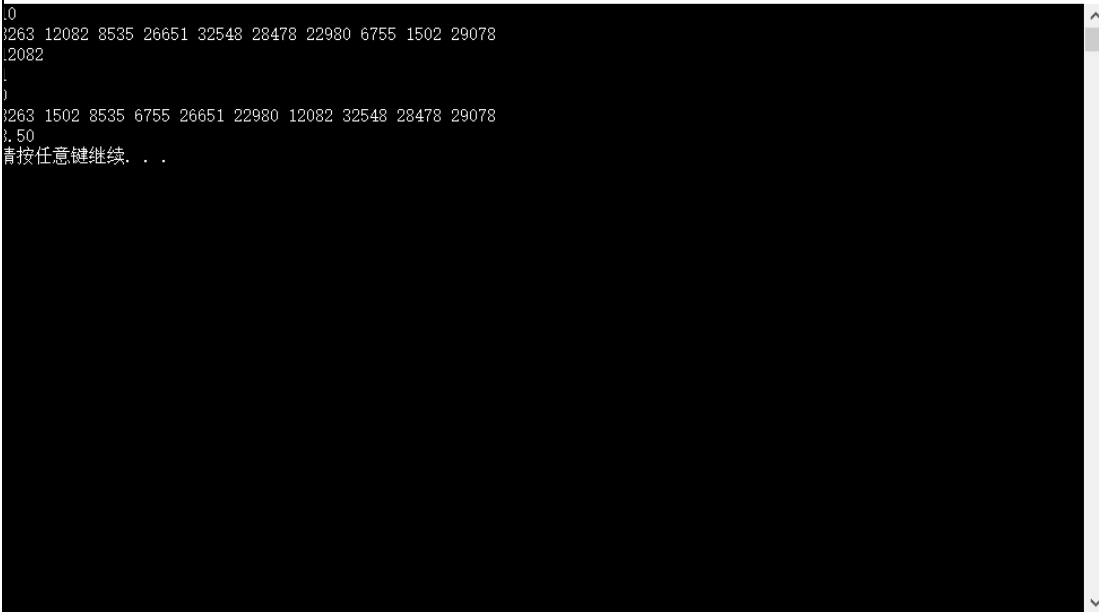
删除二叉排序树节点数一定要注意当前节点还存在后继的左子树和右子树。首先寻址到当前结点处，如果是我们要找的元素，判断它是否存在左子树，如果有的话，将前驱结点的左子树连接到新的节点，接着判断该结点的右节

点是否存在，若存在，该右子树必定小于前驱结点的右子树，将该子树直接链到前驱结点右子树的最后即可。最后将前驱结点的右子树连接到该节点的右子树。

```
int OrderT::Delete(ElemType value)
{
    OrderTNode *pre, *lcur, *rpre, *rcur;
    lcur = root;
    pre = NULL;
    while (lcur && lcur->data != value) {
        pre = lcur;
        if (lcur->data > value)
            lcur = lcur->lchild;
        else
            lcur = lcur->rchild;
    }
    if (!lcur)
        return 0;
    if (lcur->lchild == NULL) {
        if (pre == NULL)
            root = lcur->rchild;
        else if (pre->lchild == lcur)
            pre->lchild = lcur->rchild;
        else
            pre->rchild = lcur->rchild;
        delete lcur;
    }

    else {
        rpre = lcur;
        rcur = lcur->lchild;
        while (rcur->rchild) {
            rpre = rcur;
            rcur = rcur->rchild;
        }
        if (rpre == lcur)
            rpre->lchild = rcur->lchild;
        else
            rpre->rchild = rcur->lchild;
        lcur->data = rcur->data;
        delete rcur;
    }
    return 1;
}
```

	<p>5、计算可查找长度</p> <p>首先在先序遍历的时候加入一个 level 的整形变量作为层数的记载。使用一个数组进行存储。每次进入一次递归，相当于进入了下一层，将对应层数下标位置的数组元素++。可统计出所有的层的节点数目。最后将所有的节点数目乘以节点数相加，除以总的节点数，即可得到。</p> <pre>void OrderT::PreOrderTra (OrderTNode *p, int level) { if (p){ a[level]++; cout << p->data << " "; PreOrderTra(p->lchild, level + 1); PreOrderTra(p->rchild, level + 1); } } int main() { OrderTree.PreOrderTra (OrderTree.root, 0); cout << endl; for (int i = 0; a[i] != 0; i++) { sum += (a[i] * (i + 1)); } cout << setiosflags(ios::fixed) << setprecision(2) << sum / num << endl; return 0; }</pre>
开发环境	Visual studio 2017
调试分析	

	
心得体会	<p>1、折半查找时要返回第一次出现的值，这就要求在当前找到的基础上进行再一次的回溯。注意还要保持 $O(\log n)$ 的时间复杂度，所以不能回溯太久远，否则就会变成 $O(n)$ 的复杂度。因为之前的操作已经确定了下界 <code>low</code>，所以回溯到 <code>low</code> 就行。</p> <p>2、题目二同样可以使用数组计算。用一个数组模拟二叉排序树，可以避免多次申请空间带来的时间损耗。虽然在大数下空间复杂度高。</p> <p>代码如下：</p> <pre> #include<iostream> #include<iomanip> using namespace std; #define MAXSIZE 10000 #define DATA 0 #define LEFT 1 #define RIGHT 2 typedef int ElemType; struct Node { ElemType node[3]; }; Node *pre; void insert(Node *OrderT, int value) { int i = 0, pos = 0, flag=0; while (OrderT[i].node[DATA]) { </pre>

```

        if (value < OrderT[i].node[DATA]) {
            if (OrderT[i].node[LEFT] != -1) {
                i = OrderT[i].node[LEFT];
                continue;
            }
            else {
                flag = 1;
                break;
            }
        }
        else {
            if (OrderT[i].node[RIGHT] != -1) {
                i = OrderT[i].node[RIGHT];
                continue;
            }
            else {
                flag = 2;
                break;
            }
        }
    }

    pos = i;
    for (i=0; OrderT[i].node[DATA] != 0; i++);
    OrderT[i].node[DATA] = value;
    if (flag == 1) {
        OrderT[pos].node[LEFT] = i;
    }
    if (flag == 2) {
        OrderT[pos].node[RIGHT] = i;
    }
}

void delOrderNode(Node OrderT[], int value)
{
    int i=0, pre=0, flag=0;
    while (OrderT[i].node[DATA]) {
        if (value == OrderT[i].node[DATA]) {
            break;
        }
        else if (value < OrderT[i].node[DATA]) {
            pre = i;

```

```

        flag = 1;
        if (OrderT[i].node[LEFT] != -1) {
            i = OrderT[i].node[LEFT];
        }
        else {
            cout << "0" << endl;
            return;
        }
    }
    else {
        pre = i;
        flag = 2;
        if (OrderT[i].node[RIGHT] != -1) {
            i = OrderT[i].node[RIGHT];
        }
        else {
            cout << "0" << endl;
            return;
        }
    }
}

int pos = OrderT[i].node[LEFT];
int delpos = i;
if (flag == 1) {
    OrderT[pre].node[LEFT] = pos;
}
else {
    OrderT[pre].node[RIGHT] = pos;
}

if (OrderT[pos].node[RIGHT] != -1) {
    while (OrderT[i].node[RIGHT]) {
        i = OrderT[i].node[RIGHT];
    }
    OrderT[i].node[RIGHT] = OrderT[pos].node[RIGHT];
}

OrderT[pos].node[RIGHT] = OrderT[delpos].node[RIGHT];

OrderT[delpos].node[RIGHT] = -1;
OrderT[delpos].node[LEFT] = -1;
OrderT[delpos].node[DATA] = 0;
cout << "1" << endl;

```



```
}
```

```
void preOrderTra(Node *OrderT, int cur, int pos, int arr[], int level)
{
    arr[0]++;
    arr[level+1]++;
    cout << OrderT[cur].node[DATA] << " ";
    if (OrderT[cur].node[LEFT] != -1) {
        preOrderTra(OrderT, OrderT[cur].node[LEFT], cur, arr, level+1);
    }
    if (OrderT[cur].node[RIGHT] != -1) {
        preOrderTra(OrderT, OrderT[cur].node[RIGHT], cur, arr, level + 1);
    }
}
```

```
void searchIns(Node *OrderT, int value)
{
    int i = 0, pre = 0, flag = 0;
    while (OrderT[i].node[DATA]) {
        if (value == OrderT[i].node[DATA]) {
            cout << "1" << endl;
            break;
        }
        else if (value < OrderT[i].node[DATA]) {
            pre = i;
            flag = 1;
            if (OrderT[i].node[LEFT] != -1) {
                i = OrderT[i].node[LEFT];
            }
            else {
                cout << "0" << endl;
                insert(OrderT, value);
                return;
            }
        }
        else {
            pre = i;
            flag = 2;
            if (OrderT[i].node[RIGHT] != -1) {
                i = OrderT[i].node[RIGHT];
            }
            else {
                cout << "0" << endl;
            }
        }
    }
}
```

```

        insert(OrderT, value);
        return;
    }
}

}

}

int main()
{
    int i, n;
    int value;
    Node OrderT[MAXSIZE];
    cin >> n;
    for (i = 0; i < n; i++) {
        OrderT[i].node[DATA] = 0;
        OrderT[i].node[LEFT] = -1;
        OrderT[i].node[RIGHT] = -1;
    }
    for (i = 0; i < n; i++) {
        cin >> value;
        insert(OrderT, value);
    }

    cin >> value;
    delOrderNode(OrderT, value);

    searchIns(OrderT, value);

    int arr[MAXSIZE];
    for (i = 0; i <= n; i++) {
        arr[i] = 0;
    }
    preOrderTra(OrderT, 0, 0, arr, 0);
    cout << endl;

    double sum = 0;
    for (i = 0; arr[i+1] != 0; i++) {
        sum += (i + 1)*arr[i+1];
    }
    if (n == 0) {
        cout << fixed << setprecision(2) << 1.00 << endl;
    }
}

```

	<pre>else { cout << fixed << setprecision(2) << (sum / arr[0]) << endl; } return 0; }</pre>
--	---