二叉排序树

描述

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。 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。

输入

第 1 行一个整数 n，表示输入的关键字个数

第 2 行 n 个数据，表示插入的关键字

第 3 行 1 个数据，表示先删除，后查找的关键字，删除按照课本用其前驱替代。

输出

第 1 行输出删除关键字后的结果，若删除成功，返回 1，否则返回 0；

第 2 行输出查找关键字后的结果，查找成功，返回 1，否则返回 0，并插入；

第 3 行输出先序遍历的结果

第 4 行输出平均查找长度，保留 2 位小数，double 类型

样例输入 1

10

3263 12082 8535 26651 32548 28478 22980 6755 1502 29078

12082

样例输出 1

1

0

3263 1502 8535 6755 26651 22980 12082 32548 28478 29078

3.50

数据范围

n<=10000

```
#include<iostream>
#include<iomanip>
using namespace std;
typedef struct {
    int key;
}ElemType;
typedef struct BiTNode {
    ElemType data;
    BiTNode *lchild, *rchild;
    int depth;
}BiTNode, *BiTree;
double n;
BiTree T;
double sum = 0.0;
void InsertBST(BiTree &T, BiTree S)
{
    BiTree p, q = NULL;
    if (!T) {
        T = S;
    }
    else
    {
        p = T;
        while (p)
        {
            q = p;
            if (p->data.key > S->data.key)
                p = p->lchild;
            else
                p = p->rchild;
        }
        if (q->data.key > S->data.key)
            q->lchild = S;
        else
            q->rchild = S;
    }
```

```
}
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++;
    }
}
int SearchBST(BiTree &T, int key)
{
    BiTree p = NULL;
    if (!T)
    {
        p = T;
        while (p)
        {
            if (p->data.key == key)
                break;
            else if (p->data.key > key)
                p = p->lchild;
            else
                p = p->rchild;
        }
    }
    if (p == NULL)
        return 0;
    else
        return 1;
}
int DeleteNode(BiTree &p)
{
    BiTree s, q;
    int flag = 0;
    if (!p->lchild)
    {
        q = p;
```

```
        p = p->rchild;
        flag = 1;
        delete q;
    }
    else if (!p->rchild)
    {
        q = p;
        p = p->lchild;
        flag = 1;
        delete q;
    }
    else
    {
        q = p;
        s = p->lchild;
        while (s->rchild)
        {
            q = s;
            s = s->rchild;
        }
        p->data = s->data;
        if (p != q)
            q->rchild = s->lchild;
        else
            q->lchild = s->lchild;
        flag = 1;
        delete s;
    }
    if (flag == 1)
        return 1;
    else
        return 0;
}
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);
    }
}
void PreOrderTraverse(BiTree &T)
{
    if (!T)
```

```
        return;
    else
    {
        cout << T->data.key << " ";
        PreOrderTraverse(T->lchild);
        PreOrderTraverse(T->rchild);
    }
}
void NodeDepth(BiTree &T, int Depth)
{
    if (T)
    {
        NodeDepth(T->rchild, Depth + 1);
        T->depth = Depth;
        NodeDepth(T->lchild, Depth + 1);
    }
}
int SumSearchLength(BiTree &T)
{
    if (!T)
        return sum;
    else
    {
        sum += T->depth;
        SumSearchLength(T->lchild);
        SumSearchLength(T->rchild);
    }
}
int main()
{
    int key;
    cin >> n;
    CreateBST(T);
    cin >> key;
    cout << DeleteBST(T, key);
    cout << endl;
    cout << SearchBST(T, key);
    cout << endl;
    BiTree S;
    S = new BiTNode;
    S->data.key = key;
    S->lchild = NULL;
    S->rchild = NULL;
    InsertBST(T, S);
```

```
    PreOrderTraverse(T);
    cout << endl;
    double Depth = 1;
    NodeDepth(T, Depth);
    cout << setiosflags(ios::fixed) << setprecision(2) <<
SumSearchLength(T) / n << endl;
    return 0;
}
```