山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

学号：202200130048	姓名： 陈静雯	班级： 6
实验题目：搜索树		
实验学时：2	实验日期： 12.7	

实验目的：
创建带索引的二叉搜索树类。存储结构使用链表，提供操作:插入、删除、按名次删除、查找、按名次查找、升序输出所有元素。

软件开发工具：
Vscode

1. 实验内容
输入第一行一个数字 m（m<=1000000），表示有 m 个操作。
接下来 m 行，每一行有两个数字 a，b：
当输入的第一个数字 a 为 0 时，输入的第二个数字 b 表示向搜索树中插入 b
当输入的第一个数字 a 为 1 时，输入的第二个数字 b 表示向搜索树中查找 b
当输入的第一个数字 a 为 2 时，输入的第二个数字 b 表示向搜索树中删除 b
当输入的第一个数字 a 为 3 时，输入的第二个数字 b 表示查找搜索树中名次为 b 的元素
当输入的第一个数字 a 为 4 时，输入的第二个数字 b 表示删除搜索树中名次为 b 的元素

2. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）
插入：从根节点开始，如果待插入元素比节点小，往左孩子走，如果比节点大，往右孩子走，直到子节点为空。最后更新索引值，如果在插入的时候往左子树走，节点的索引值+1

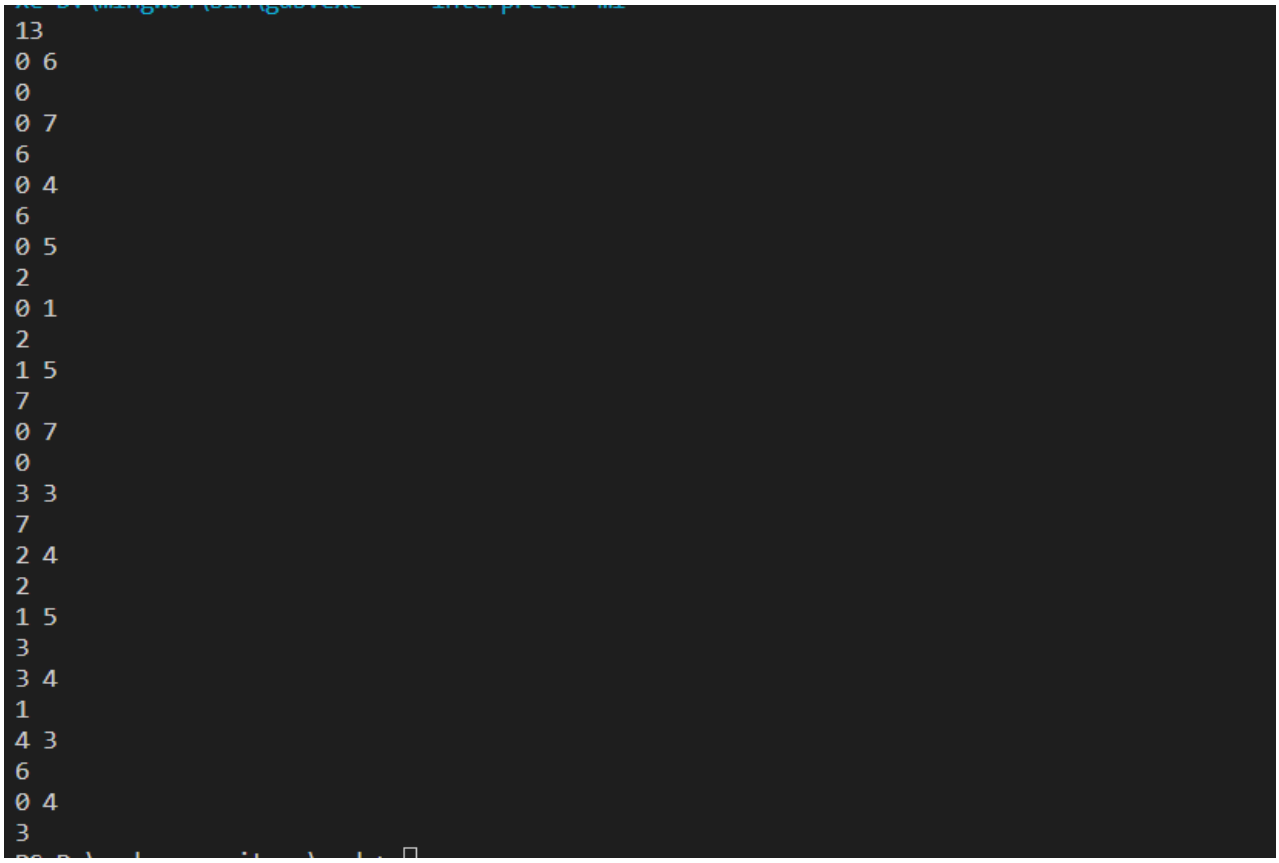
查找：从根节点开始，如果待查找元素比节点小，往左孩子走，如果比节点大，往右孩子走，直到找到该元素，或子节点为空，即找不到元素

删除：先查找该元素，再进行删除。删除的时候，若删除的节点左右孩子都不为空，则把右子树的最小节点放到删除的元素的位置，若删除的节点左右孩子至少有一个为空，则直接把不为空的子树连接到删除的位置即可。最后更新索引值。

按名次查找：根据索引值，定义一个 pre，若往右孩子走，则 pre+=父节点的索引值+1，即该节点前面的元素个数，pre+节点索引值+1 为该节点的名次，待查找名次跟它比较，小往左，大往右，直到找到待查找名次的元素为止

按名次删除：先按名次查找，再进行删除，最后更新索引值。

3. 测试结果（测试输入，测试输出）



4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）

Scanf 和 printf 比 cin、cout 快

索引值要在最后更新，查找的同时更新会扰乱本来该查找的名次

5. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）

```
#include <iostream>
#include <utility>
using namespace std;

template <class K,class E>
struct binarytreenode{
    pair<K,E> element;
    binarytreenode<K,E>* leftchild,
                       * rightchild;
    binarytreenode(){
        element=make_pair(0,0);
        leftchild=rightchild=NULL;
    }
    binarytreenode(const pair<K,E> thelement){
        element=thelement;
        leftchild=NULL;
        rightchild=NULL;
```

```
    }
    binarytreenode(const pair<K,E> thelement, binarytreenode<K,E>* theleftchild,
binarytreenode<K,E>* therightchild){
        element=thelement;
        leftchild=theleftchild;
        rightchild=therightchild;
    }
};

template <class K,class E>
class binarysearchtree{
public:
    binarysearchtree(){ root = NULL; treesize=0;}
    bool empty(){ return treesize==0;}
    void insert(const E& thekey);
    void find(const E& thekey);
    void erase(const E& thekey);
    void find_rank(int t);
    void erase_rank(int t);
private:
    binarytreenode<K,E>* root;
    int treesize;
};

template<class K,class E>
void binarysearchtree<K,E>::insert(const E& thekey){
    binarytreenode<K,E> *p = root;
    binarytreenode<K,E> *pp = NULL;
    int num = 0;
    while(p!=NULL){
        pp = p;
        if(thekey<p->element.second){                //插入的元素比节点小，往左孩子
走；大，往右孩子走
            num^=p->element.second;
            p=p->leftchild;
        }
        else if(thekey>p->element.second){
            num^=p->element.second;
            p=p->rightchild;
        }
        else{
            printf("0\n");
            return ;
        }
    }
    binarytreenode<K,E>* temp = new binarytreenode<K,E>(make_pair(0,thekey));
```

```
    if(root!=NULL){
        if(thekey<pp->element.second) pp->leftchild = temp;
        else pp->rightchild = temp;
    }
    else{
        root = temp;
    }
    printf("%d\n",num);
    treesize++;
    p = root;
    while(p->element.second!=thekey){           //跟新索引值，如果插入到左子树，
索引值++
        if(thekey<p->element.second){
            p->element.first++;
            p=p->leftchild;
        }
        else if(thekey>p->element.second){
            p=p->rightchild;
        }
    }
}

template<class K,class E>
void binarysearchtree<K,E>::find(const E& thekey){
    binarytreenode<K,E> *p = root;
    int num=thekey;
    while(p!=NULL){
        if(thekey < p->element.second){
            num^=p->element.second;
            p=p->leftchild;
        }
        else if(thekey > p->element.second){
            num^=p->element.second;
            p=p->rightchild;
        }
        else{
            printf("%d\n",num);
            return ;
        }
    }
    printf("0\n");
}

template<class K,class E>
void binarysearchtree<K,E>::erase(const E& thekey){
    binarytreenode<K,E> *p = root;
```

```
    binarytreenode<K,E> *pp = NULL;
    int num=thekey;
    while(p!=NULL && p->element.second!=thekey){
        if(thekey<p->element.second){
            num^=p->element.second;
            p=p->leftchild;
        }
        else if(thekey>p->element.second){
            num^=p->element.second;
            p=p->rightchild;
        }
    }
    if(p==NULL) {
        printf("0\n");
        return ;
    }
    printf("%d\n",num);
    p = root;
    while(p!=NULL && p->element.second!=thekey){
        pp=p;
        if(thekey<p->element.second){
            p->element.first--;
            p=p->leftchild;
        }
        else if(thekey>p->element.second){
            p=p->rightchild;
        }
    }

    if(p->leftchild!=NULL && p->rightchild!=NULL){
        binarytreenode<K,E> *s = p->rightchild;
        binarytreenode<K,E> *ps = p;
        while(s->leftchild!=NULL){
            ps=s;
            s->element.first--;
            s=s->leftchild;
        }
        binarytreenode<K,E>       *q       =       new       binarytreenode<K,E>
(make_pair(p->element.first,s->element.second),p->leftchild,p->rightchild);
        if(pp==NULL) root = q;
        else if(p==pp->leftchild){
            pp->leftchild = q;
        }
        else pp->rightchild = q;
        if(ps == p) pp=q;
        else pp=ps;        //pp 为 p 的父节点
```

```
        delete p;
        p=s;        //p 为要删去的节点，即右子树的最小元素
    }

    binarytreenode<K,E> *c;
    if(p->leftchild != NULL) c=p->leftchild;
    else c=p->rightchild;
                                                        //p 为右子树的最小元素，
即 c=NULL
    if(p==root) root=c;
    else{
        if(p==pp->leftchild){
            pp->leftchild = c;
        }
        else pp->rightchild = c;
    }
    delete p;
    treesize--;
}

template<class K,class E>
void binarysearchtree<K,E>::find_rank(int t){
    if(t>treesize || t<=0){
        printf("0\n");
        return ;
    }
    binarytreenode<K,E> *p = root;
    int num=0;
    int pre=0;
    while(p!=NULL){
        if(t < pre + p->element.first +1){          //根据索引值找到名次为 t 的节
点
            num^=p->element.second;
            p=p->leftchild;
        }
        else if(t > pre + p->element.first +1){
            num^=p->element.second;
            pre+=p->element.first+1;            //往右子树走要更新 pre 值
            p=p->rightchild;
        }
        else{
            num^=p->element.second;
            printf("%d\n",num);
            return ;
        }
    }
```

```
}

template<class K,class E>
void binarysearchtree<K,E>::erase_rank(int t){
    if(t>treesize || t<=0){
        printf("0\n");
        return ;
    }
    binarytreenode<K,E> *p = root;
    binarytreenode<K,E> *pp = NULL;
    int num=0;
    int pre=0;
    while(p!=NULL && t != pre + p->element.first +1){   //索引是左子树元素个数
        pp=p;
        if(t < pre + p->element.first +1){
            num^=p->element.second;
            p=p->leftchild;
        }
        else if(t > pre + p->element.first +1){
            num^=p->element.second;
            pre+=p->element.first+1;
            p=p->rightchild;
        }
    }
    num^=p->element.second;
    printf("%d\n",num);
    E thekey = p->element.second;
    p = root;
    while(p!=NULL && p->element.second!=thekey){
        pp=p;
        if(thekey<p->element.second){
            p->element.first--;
            p=p->leftchild;
        }
        else if(thekey>p->element.second){
            p=p->rightchild;
        }
    }
    if(p->leftchild!=NULL && p->rightchild!=NULL){
        binarytreenode<K,E> *s = p->rightchild;
        binarytreenode<K,E> *ps = p;
        while(s->leftchild!=NULL){
            ps=s;
            s->element.first--;
            s=s->leftchild;
        }
```

```
        binarytreenode<K,E>       *q       =       new       binarytreenode<K,E>
(make_pair(p->element.first,s->element.second),p->leftchild,p->rightchild);
        if(pp==NULL) root = q;
        else if(p==pp->leftchild){
            pp->leftchild = q;
        }
        else pp->rightchild = q;
        if(ps == p) pp=q;
        else pp=ps;         //pp 为 p 的父节点
        delete p;
        p=s;        //p 为要删去的节点，即右子树的最小元素
    }

    binarytreenode<K,E> *c;
    c=NULL;
    if(p->leftchild != NULL) c=p->leftchild;
    else if(p->rightchild!=NULL) c=p->rightchild;
                                                          //p 为右子树的最小元素，
即 c=NULL
    if(p==root) root=c;
    else{
        if(p==pp->leftchild){
            pp->leftchild = c;
        }
        else pp->rightchild = c;
    }
    delete p;
    treesize--;
}

int main(){
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int n;
    scanf("%d",&n);
    binarysearchtree<int,int> tree;
    for(int i=0;i<n;i++){
        int temp,key;
        scanf("%d%d",&temp,&key);
        if(temp==0) tree.insert(key);
        if(temp==1) tree.find(key);
        if(temp==2) tree.erase(key);
        if(temp==3) tree.find_rank(key);
        if(temp==4) tree.erase_rank(key);
    }
}
```