

lab3实验报告

学号：PB20061338 姓名：柯志伟

1 实验设备和环境

平台：windows
编程语言：C与C++

2 实验内容及要求

2.1 实验内容

实验3.1：区间树

实现区间树的基本算法，随机生成30个正整数区间，以这30个正整数区间的左端点作为关键字构建红黑树，先向一棵初始空的红黑树中依次插入30个节点，然后随机选择其中3个区间进行删除，最后对随机生成的3个区间(其中一个区间取自(25,30))进行搜索。实现区间树的插入、删除、遍历和查找算法

2.2 实验要求

1. 实验设备和环境、实验内容及要求、方法和步骤、结果与分析
2. 比较实际复杂度和理论复杂度是否相同，给出分析

2.3 方法和步骤

1. 实现红黑树并扩展成区间树

数据结构实现如下：

```
#define red 0
#define black 1

struct Interval {
    int low;
    int high;
    Interval() : low(0), high(0) {}
    Interval(int low,int high) : low(low),high(high) {}

    bool overlap(Interval interval) {
        return (this->low <= interval.high && interval.low
<= this->high);
    }
}
```

```
};
```

```
struct interval_tree_node {
    int color;
    int key;
    int max;
    Interval interval;
    interval_tree_node* par;
    interval_tree_node* lchild;
    interval_tree_node* rchild;

    interval_tree_node(int k, int high, int low, int max, int
col, interval_tree_node* p=NULLptr, interval_tree_node*
lch=NULLptr, interval_tree_node* rch=NULLptr ) :
        key(k), par(p), lchild(lch), rchild(rch),
color(col), interval(low, high), max(max){}

    static void free_node_iterate(interval_tree_node* nodeptr);
};
```

```
class INTTree {

public:
    INTTree() { root = nil;}

    ~INTTree() { rbdestroy();}

    static interval_tree_node* nil;

public:
    void rbtransplant(interval_tree_node* u,
interval_tree_node* v);
    interval_tree_node* rbsearch(int key);
    void rbleftRotate(interval_tree_node* x);
    void rbrightRotate(interval_tree_node* x);
    void rbinert(interval_tree_node* nodeptr);
    void rbinert_fixup(interval_tree_node* x);
    void rbdelete(interval_tree_node* nodeptr);
    void rbdelete_fixup(interval_tree_node* x);
    void rbdestroy();
    interval_tree_node* rbmin(interval_tree_node* nodeptr);
    interval_tree_node* get_root() { return root; }

    bool rb_tree_check();

    void interval_insert(interval_tree_node* nodeptr) {
rbinert(nodeptr); }
    void interval_delete(interval_tree_node* nodeptr) {
rbdelete(nodeptr); }
    interval_tree_node* interval_search(Interval interval);
```

```

void interval_fixup(interval_tree_node* x);

void inorder_visit(interval_tree_node* node);

void inttree_inorder_visit() {
    fout.open(fileout_path, ios::app);
    inorder_visit(this->root);
    fout.close();
}

void init_fout(string fpath) { fileout_path = fpath; }

bool inttree_check();
private:
    interval_tree_node* root;
    string fileout_path;
    ofstream fout;
};

```

具体实现见代码

2. 产生30个随机区间以及三个待搜索的区间

使用python脚本模拟随机选择的区间

```

import random

## 产生30个随机区间,所有区间取自区间[0,25]或[30,50]且各区间左端点互
## 异,不要和(25,30)有重叠

a = [i for i in range(26)]
b = [30 + i for i in range(21)]
gen_sets = ['a', 'b']

data = []
data_l = []

while (len(data) != 30):
    gen_set = random.choice(gen_sets)
    if gen_set == 'a':
        l = random.choice(a)
        r = random.choice(a)
        if l == r:
            continue
        if l > r:
            l, r = r, l

    if l in data_l:
        continue
    else:

```

```

        data_l.append(l)
        data.append((l, r))
    else:
        l = random.choice(b)
        r = random.choice(b)
        if l == r:
            continue
        if l > r:
            l, r = r, l
        if l in data_l:
            continue
        else:
            data_l.append(l)
            data.append((l, r))

with
open("E:\\Savefiles\\Labs\\Algorithm\\lab3\\ex1\\input\\input.t
xt", "w") as f:
    for (a,b) in data:
        f.write(str(a))
        f.write('\t')
        f.write(str(b))
        f.write('\n')

search_data = []
while (len(search_data) != 2):
    gen_set = random.choice(gen_sets)
    if gen_set == 'a':
        l = random.choice(a)
        r = random.choice(a)
        if l > r:
            l, r = r, l
        else:
            search_data.append((l, r))
    else:
        l = random.choice(b)
        r = random.choice(b)
        if l > r:
            l, r = r, l
        else:
            search_data.append((l, r))

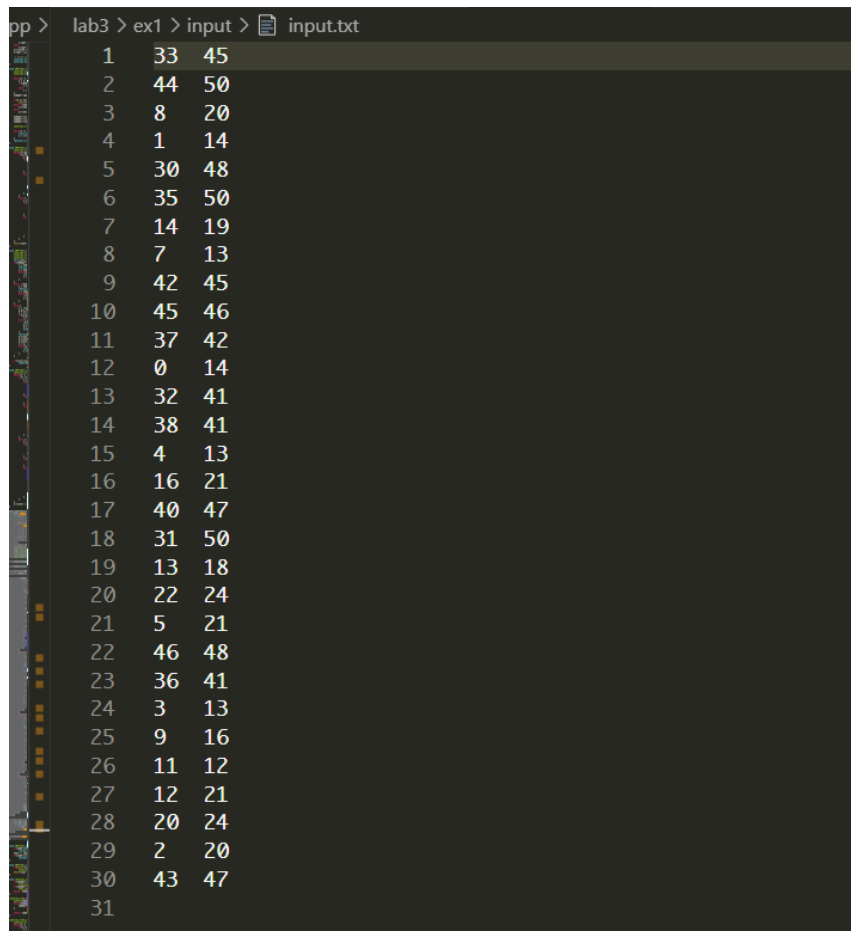
c = [26, 27, 28, 29]
l = random.choice(c)
r = random.choice(c)

if l > r:
    l, r = r, l
search_data.append((l, r))

```

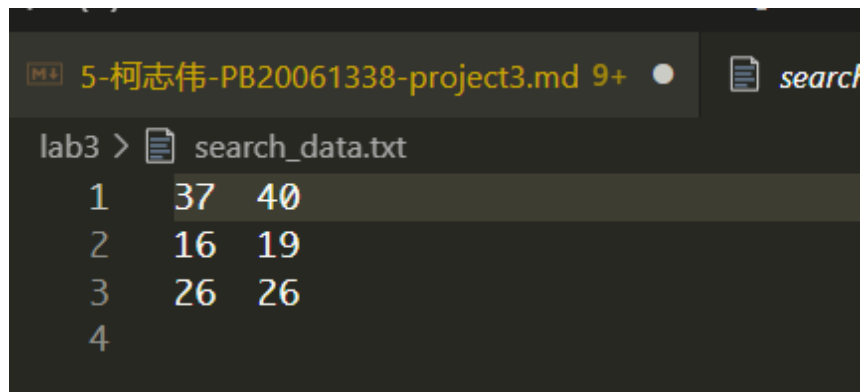
```
with
open("E:\\Savefiles\\Labs\\Algorithm\\lab3\\search_data.txt",
"w") as f:
    for (a,b) in search_data:
        f.write(str(a))
        f.write('\t')
        f.write(str(b))
        f.write('\n')
```

30个随机区间



1	33	45
2	44	50
3	8	20
4	1	14
5	30	48
6	35	50
7	14	19
8	7	13
9	42	45
10	45	46
11	37	42
12	0	14
13	32	41
14	38	41
15	4	13
16	16	21
17	40	47
18	31	50
19	13	18
20	22	24
21	5	21
22	46	48
23	36	41
24	3	13
25	9	16
26	11	12
27	12	21
28	20	24
29	2	20
30	43	47
31		

3个随机搜索区间



```
lab3 > search_data.txt
1 37 40
2 16 19
3 26 26
4
```

3. 为保证算法实现的正确性，实现区间树性质的检验代码

```
bool INTTree::inttree_check() {
    bool check = true;
    // 红黑树性质
    // 1. 每个节点或是红色,或是黑色: 通过创建节点保证
    // 2. 根节点是黑色
    if (this->root->color != black)
        check = false;
    // 3. 每个叶节点是黑色
    // 4. 如果一个节点是红色,那么两个子节点也是黑色
    // 5. 该节点到所有后代节点的黑高度相同
    // 6. 该树是一颗二叉树
    // 区间树性质
    // 7. node.max = max(node.int.high, node.lch.max,
node.rch.max)

    // 以上几条使用深度优先遍历检测
    interval_tree_node* prev;
    std::vector<interval_tree_node*> leaves;
    std::vector<interval_tree_node*> stack;
    std::map<interval_tree_node*, int> black_height;
    std::map<interval_tree_node*, bool> visited;

    black_height[nil] = 0;

    stack.push_back(this->root);

    while(! stack.empty()) {
        prev = stack.back();

        if(prev->lchird != nil && prev->rchird != nil) {
            // 检查满足二叉树性质
```

```

        if(! (prev->lchild->key <= prev->key && prev->rchild->key >= prev->key))
            check = false;
        // 检查满足区间树的max域性质
        int max = prev->interval.high;
        if(prev->lchild->max > max)
            max = prev->lchild->max;
        if(prev->rchild->max > max)
            max = prev->rchild->max;
        if(max != prev->max)
            check = false;
        // 更新黑高度
        if(prev->color == black)
            black_height[prev] = black_height[prev->par] +
1;

        else
            black_height[prev] = black_height[prev->par];
        // 如果一个节点是黑色,那么两个子节点也是黑色
        if(prev->color == red && (prev->lchild->color ==
red || prev->rchild->color == red))
            check = false;
        if(visited.find(prev->lchild) != visited.end() &&
visited[prev->lchild] == false || visited.find(prev->lchild) ==
visited.end()) {
            stack.push_back(prev->lchild);
        }
        else if(visited.find(prev->rchild) != visited.end()
&& visited[prev->rchild] == false || visited.find(prev->rchild)
== visited.end()) {
            stack.push_back(prev->rchild);
        }
        else {
            visited[prev] = true;
            stack.pop_back();
        }
    }
    else if(prev->lchild != nil) {
        // 检查满足二叉树性质
        if(! (prev->lchild->key <= prev->key))
            check = false;
        // 检查满足区间树的max域性质
        int max = prev->interval.high;
        if(prev->lchild->max > max)
            max = prev->lchild->max;
        if(max != prev->max)
            check = false;
        // 更新黑高度
        if(prev->color == black)
            black_height[prev] = black_height[prev->par] +
1;

        else

```

```

        black_height[prev] = black_height[prev->par];
// 如果一个节点是黑色,那么两个子节点也是黑色
        if(prev->color == red && prev->lchird->color ==
red)
            check = false;
            if(visited.find(prev->lchird) != visited.end() &&
visited[prev->lchird] == false || visited.find(prev->lchird) ==
visited.end()) {
                stack.push_back(prev->lchird);
            }
            else {
                visited[prev] = true;
                stack.pop_back();
            }
        }
        else if(prev->rchird != nil) {
            // 检查满足二叉树性质
            if(! (prev->rchird->key >= prev->key))
                check = false;
            // 检查满足区间树的max域性质
            int max = prev->interval.high;
            if(prev->rchird->max > max)
                max = prev->rchird->max;
            if(max != prev->max)
                check = false;
            // 更新黑高度
            if(prev->color == black)
                black_height[prev] = black_height[prev->par] +
1;

            else
                black_height[prev] = black_height[prev->par];
            // 如果一个节点是黑色,那么两个子节点也是黑色
            if(prev->color == red && prev->rchird->color ==
red)
                check = false;
                if(visited.find(prev->rchird) != visited.end() &&
visited[prev->rchird] == false || visited.find(prev->rchird) ==
visited.end()) {
                    stack.push_back(prev->rchird);
                }
                else {
                    visited[prev] = true;
                    stack.pop_back();
                }
            }
        }
        else {
            // 此时为叶子节点
            // 检查满足二叉树性质
            // 检查满足区间树的max域性质
            if(prev->max != prev->interval.high)
                check = false;

```



```

        // 更新黑高度
        if(prev->color == black)
            black_height[prev] = black_height[prev->par] +
1;

        else
            black_height[prev] = black_height[prev->par];
        // 如果一个节点是黑色,那么两个子节点也是黑色
        visited[prev] = true;
        stack.pop_back();
        leafs.push_back(prev);
    }
}

// 检查所有叶子的黑高度
int bh = black_height[*(leafs.begin())];
for(auto &node: leafs) {
    if(bh != black_height[node])
        check = false;
}

return check;
}

```

2.4 结果与分析

读入30个随机区间构建区间树,并完成中序遍历,以及第一次区间树检查

5-柯志伟-PB20061338-project3.md 9+inorder.txt

lab3 > ex1 > output > inorder.txt

1	0	14	14
2	1	14	21
3	2	20	20
4	3	13	20
5	4	13	13
6	5	21	21
7	7	13	13
8	8	20	50
9	9	16	16
10	11	12	21
11	12	21	21
12	13	18	21
13	14	19	50
14	16	21	21
15	20	24	24
16	22	24	24
17	30	48	50
18	31	50	50
19	32	41	50
20	33	45	50
21	35	50	50
22	36	41	41
23	37	42	50
24	38	41	47
25	40	47	47
26	42	45	50
27	43	47	47
28	43	47	50
29	44	50	50
30	45	46	50
31	46	48	48
32			

lab3 > ex1 > output > delete_data.txt

1	8		
2	0	14	14
3	1	14	21
4	2	20	20
5	3	13	20
6	4	13	13
7	5	21	21
8	7	13	13
9	9	16	50
10	11	12	12
11	12	21	21
12	13	18	18
13	14	19	50
14	16	21	21
15	20	24	24
16	22	24	24
17	30	48	50
18	31	50	50
19	32	41	50
20	33	45	50
21	35	50	50
22	36	41	41
23	37	42	50
24	38	41	47
25	40	47	47
26	42	45	50
27	43	47	47
28	43	47	50
29	44	50	50
30	45	46	50
31	46	48	48
32			

```
Interval Tree Check(true or false): 1
Interval Tree Check(true or false): 1
```

注: 区间树两次检查结果都在图中(true为1, false为0)

搜索三个随机区间

```
lab3 > ex1 > output > search.txt
1 33 45
2 9 16
3 Null
4
```

结果分析

由于只构建一棵区间树,而且以不同的插入方式最终构建出的区间树形态有所差异,且中间插入和删除涉及的操作执行次数也受此影响,导致时间复杂度较难分析,故不再分析