

实验报告

陆子睦

PB20051150

实验内容

实现区间树的插入、删除、遍历和查找算法。

1. 随机生成30个正整数区间，向一棵初始空的红黑树中依次插入这30个节点。中序遍历打印生成的红黑树。
2. 随机选择其中3个区间进行删除。打印删除的区间和删除后的红黑树。
3. 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。打印查找到的区间。

实验设备和环境

处理器 12th Gen Intel(R) Core(TM) i5-12500H 2.50 GHz

机带 RAM 16.0 GB

系统类型 64 位操作系统, 基于 x64 的处理器

版本 Windows 11 家庭中文版

软件 VSCode, g++

实验方法和步骤

认真阅读并理解红黑树的插入，删除，查找算法，理解红黑树的性质和结构特征，以及对红黑树操作时如何维护这些特征。

1. 编写 input_gen.cpp，用rand函数产生30个随机生成的区间，让区间位于 [0, 25], [30, 50] 之间，把结果输入到input.txt中。

其中生成某个区间内的随机数公式如下：

```
x = a + (int)(b - a) * rand() / (RAND_MAX + 1);
```

2. 编写 interval_tree.cpp，用如下结构体表示树的结点：

```
typedef struct Node {
    struct Node *p, *left, *right;
    Color color;
    int low;
    int high;
    int max_val;
} Node;
```

其中Color是枚举。

编写如下一些函数，实现插入，删除，查找的功能。

左旋和右旋：

```
void left_rotate(Node * &root, Node * x);  
void right_rotate(Node * &root, Node * x);
```

插入和插入的调整：

```
void RB_insert(Node * &root, Node * z);  
void RB_insert_fixup(Node * &root, Node * &z);
```

删除和删除的调整，以及用于辅助的嫁接，求子树最小结点函数：

```
void RB_delete(Node * &root, Node * z);  
void RB_delete_fixup(Node * &root, Node * x);  
void RB_transplant(Node * &root, Node *u, Node *v);  
Node * tree_minimum(Node * x);
```

查找函数：

```
void get_interval(int &low, int &high, int a, int b);
```

还有中序遍历打印的函数：

```
void print_tree(Node * root, ofstream & ofs);
```

其中ofs是输出文件的流。

其中代码逻辑基本上是按照书上的伪代码编写，维护max_val域的代码片段如下：

```
g->max_val = max(max(g->high, g->left->max_val), g->right->max_val);  
g = g->p;  
while(g->max_val == y->max_val && g != NIL) {  
    g->max_val = max(max(g->high, g->left->max_val), g->right->max_val);  
    g = g->p;  
}
```

其中g是删除或插入的结点的父节点，就是从改变的结点开始一层一层地维护父节点中可能改变的max_val值。

rotate时维护方法如下：

```
y->max_val = x->max_val;  
x->max_val = max(x->high, max(x->left->max_val, x->right->max_val));
```

其中y是旋转之后居上的结点，x是之前居上的结点，自然y对应的子树的max是之前x对应子树的max而x的新的max只要通过公式算一下就可以了。

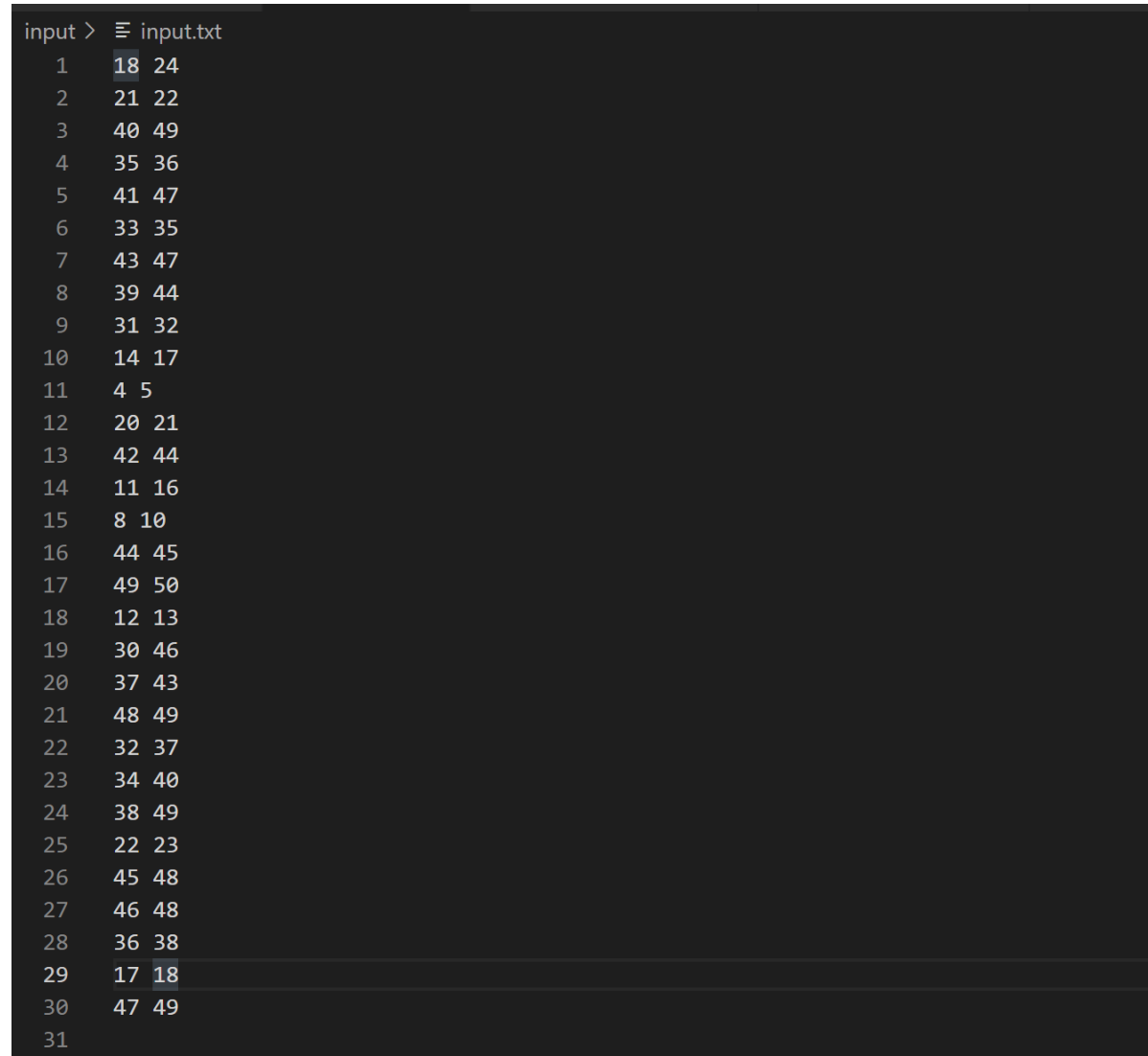
输出文件的方法是打开了三个输出文件流如下：

```
ofstream inorder_ofs, search_ofs, delete_data_ofs;
inorder_ofs.open("../output/inorder.txt", ios::out);
search_ofs.open("../output/search.txt", ios::out);
delete_data_ofs.open("../output/delete_data.txt", ios::out);
```

这样只要流式输出到这些文件流里就可以得到三个输出文件。

实验结果

输入文件input.txt



```
input > ≡ input.txt
1 18 24
2 21 22
3 40 49
4 35 36
5 41 47
6 33 35
7 43 47
8 39 44
9 31 32
10 14 17
11 4 5
12 20 21
13 42 44
14 11 16
15 8 10
16 44 45
17 49 50
18 12 13
19 30 46
20 37 43
21 48 49
22 32 37
23 34 40
24 38 49
25 22 23
26 45 48
27 46 48
28 36 38
29 17 18
30 47 49
31
```

inorder.txt, 可以看出中序输出结果low确实是有序的:

```
output > ≡ inorder.txt
```

```
1   4 5 5
2   8 10 16
3  11 16 16
4  12 13 13
5  14 17 46
6  17 18 18
7  18 24 24
8  20 21 21
9  21 22 46
10 22 23 23
11 30 46 46
12 31 32 46
13 32 37 37
14 33 35 40
15 34 40 40
16 35 36 50
17 36 38 38
18 37 43 43
19 38 49 49
20 39 44 44
21 40 49 49
22 41 47 47
23 42 44 50
24 43 47 47
25 44 45 50
26 45 48 48
27 46 48 49
28 47 49 49
29 48 49 50
30 49 50 50
```

delete_data.txt, 可以看出确实删除了那三个结点, 而且结果的中序遍历仍然有序:

```

output > ≡ delete_data.txt
1  18 24 24
2  49 50 50
3  33 35 40
4  the tree after delete:
5  4 5 5
6  8 10 16
7  11 16 16
8  12 13 13
9  14 17 46
10 17 18 18
11 20 21 21
12 21 22 46
13 22 23 23
14 30 46 46
15 31 32 46
16 32 37 37
17 34 40 40
18 35 36 49
19 36 38 38
20 37 43 43
21 38 49 49
22 39 44 44
23 40 49 49
24 41 47 47
25 42 44 49
26 43 47 47
27 44 45 49
28 45 48 48
29 46 48 49
30 47 49 49
31 48 49 49
32

```

search.txt, 可以看出搜到的结点区间确实和搜索的区间有overlap, 而且(25, 30)之间的区间没有找到 overlap:

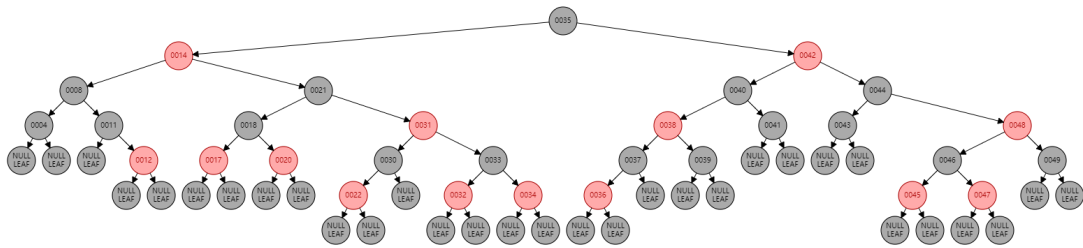
```

output > ≡ search.txt
1  the interval:
2  19 22
3  result:
4  21 22
5  the interval:
6  26 27
7  result:
8  not found
9  the interval:
10 47 49
11 result:
12 40 49
13

```

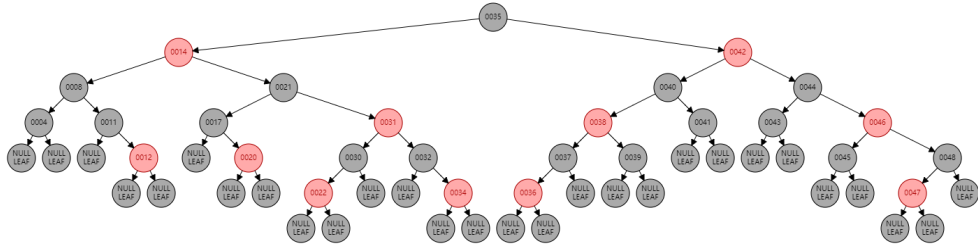
我用一个可以自动生成红黑树的网站[Red/Black Tree Visualization \(usfca.edu\)](http://usfca.edu)

绘制了这个红黑树的插入之后的图:



Animation Completed

以及删除之后的图：



Animation Completed

对结果进行了验证，发现是正确的。