实验三 区间树

袁雨 PB20151804

一、实验设备和环境

1. 实验设备

- 设备：HUAWEI MateBook X Pro
- 处理器：Intel(R) Core(TM) i5-10210U CPU @1.60GHz 2.11 GHz

2. 实验环境

- vscode, gcc

二、实验内容和要求

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、遍历和算法。

2.实验要求

（1）编程要求

- C/C++

（2）目录格式

实验需建立根文件夹，文件夹名称为：编号-姓名-学号-project3，在根文 件夹下需包括实验报告和ex1实验文件夹，每个实验文件夹包含3个子文件夹：

- input文件夹：存放输入数据
- src文件夹：源程序
- output文件夹：输出数据

实验3.1 区间树

- ex1/input/
    - input.txt：
        - 输入文件中每行两个随机数据，表示区间的左右端点，其右端点值大于左端点值，总行数大于等于30。
        - 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。

        - 读取每行数据作为区间树的x.int域，并以其左端点构建红黑树，实现插入、删除、查找操作。
- ex1/output/
    - inorder.txt:
        - 输出构建好的区间树的中序遍历序列，每行三个非负整数，分别为各节点int域左右端点和max域的值。
    - delete_data.txt：
        - 输出删除的数据，以及删除完成后区间树的中序遍历序列。
    - search.txt:
        - 对随机生成的3个区间(其中一个区间取自(25,30))进行搜索得到的结果，搜索成功则返回一个与搜索区间重叠的区间，搜索失败返回Null。
- 同行数据间用空格隔开

（3）实验报告

- 实验设备和环境、实验内容及要求、方法和步骤、结果与分析。

三、实验方法和步骤

1. 区间树的实现

扩张红黑树来支持由区间构成的动态集合上的一些操作。把一个区间 $[t_1, t_2]$ 表示成一个结构体 `interval`，其中属性 `interval->low` = t_1 为低端点 (low endpoint)，属性 `interval. high` =t_2 为高端点 (high endpoint) 。

```
typedef struct Interval
{
    int low;
    int high;
} Interval;
```

步骤 1：基础数据结构

我们选择这样一棵红黑树, 其每个结点 x 包含一个区间属性 `x. interval`，且 x 的关键字为区间的低端点 `x.int->low`。因此，该数据结构按中序遍历列出的就是按低端点的次序排列的各区间。

步骤 2: 附加信息

每个结点 x 中除了自身区间信息之外，还包含一个值 `x->max`, 它是以 x 为根的子树中所有区间的端点的最大值。

步骤 3: 对信息的维护

通过给定区间 $x.interval$ 和结点 x 的子结点的 max 值，可以确定 $x.max$ 值：

```
x->max = MAX(x->interval.high, x->left->max, x->right->max);
```

步骤 4: 设计新的操作

这里我们仅需要唯一的一个新操作 `INTERVAL-SEARCH(T, i)`, 它是用来找出树 T 中与区间 i 重叠的那个结点。若树中与 i 重叠的结点不存在，则返回指向哨兵 `T->NIL` 的 指针。

具体红黑树相关信息与操作参考教材13章，区间树相关信息与操作参考教材14章。

实现的函数如下：

```
int MAX(int high, int leftMax, int rightMax);//求三者中的最大值
```

```
Node *IntervalMin(RBTree *T, Node *x);//求子树中具有最小关键字的结点
RBTree *IntervalCreateTree();//创建区间树
Node *IntervalCreateNode(int low, int high);//创建区间为[low,high]的结点
Node *IntervalFindNode(RBTree *T, int num);//寻找关键字为num的结点
int OverLap(Node *x, Node *y);//判断x与y的区间是否重叠
Node *IntervalSearch(RBTree *T, Node *i);//查找与区间i重叠的结点
void LeftRotate(RBTree *T, Node *x);//左旋
void RightRotate(RBTree *T, Node *x);//右旋
void IntervalInsertFixUp(RBTree *T, Node *z);//插入结点后保持红黑性质
void IntervalInsert(RBTree *T, Node *z);//插入结点
void IntervalTransplant(RBTree *T, Node *u, Node *v);//删除结点的子过程
void IntervalDeleteFixUp(RBTree *T, Node *x);//删除结点
void IntervalDelete(RBTree *T, Node *z);//删除结点后保持红黑性质
void InOrder(FILE *fp, RBTree *T, Node *x);//中序遍历
```

具体代码见src。

2. 主函数

（1）打开文件，定义文件指针、变量等。

```
FILE *fp_in = fopen("..\\input\\input.txt", "w+");
FILE *fp_out1 = fopen("..\\output\\inorder.txt", "w");
FILE *fp_out2 = fopen("..\\output\\delete_data.txt", "w");
FILE *fp_out3 = fopen("..\\output\\search.txt", "w");

int i = 0, choice = 0, temp = 0, templow = 0, temphigh = 0, delIndex = 0;
int num[51] = {0}, low[30] = {0}, high[30] = {0}, searchlow[3] = {0},
searchhigh[3] = {0};
Node *x, *delNode, *searchNode;
```

（2）随机生成30个左端点互异的整数区间，所有区间取自[0,25]或[30,50]。定义数组num[51]，初始化为全0，用于左端点的互异性判断。例：随机生成的一个区间的左端点为6，则令num[6]=1。下次生成时，若生成的左端点为x，则首先查看num[x]是否为1，若是，则继续随机生成，直到生成x使得num[x]=0。右端点在左端点的基础上生成。将生成的数据存入input.txt。

```
// 随机生成30个左端点互异的自然数区间
    srand((unsigned)time(NULL));
    for (i = 0; i < 30; i++)
    {
        choice = rand() % 2;
        if (choice == 0)
        { //[0,25]
            low[i] = rand() % 26;
            while (num[low[i]] == 1)
            {
                low[i] = rand() % 26;
                // printf("num[%d]:%d ", low[i], num[low[i]]);
            }
            high[i] = low[i] + rand() % (25 - low[i] + 1);
        }
        else
        { //[30,50]
            low[i] = 30 + rand() % 21;
```

```
            while (num[low[i]] == 1)
            {
                low[i] = 30 + rand() % 21;
                // printf("num[%d]:%d ", low[i], num[low[i]]);
            }
            high[i] = low[i] + rand() % (50 - low[i] + 1);
        }

        fprintf(fp_in, "%d %d\n", low[i], high[i]);
        num[low[i]] = 1;
    }
```

（3）创建、插入操作。从input.txt中读取数据，创建区间树，依次插入30个节点。并以其左端点构建区间树。中序遍历构建好的区间树，并将结果存入inorder.txt。

```
rewind(fp_in);

// 构建红黑树
RBTree *T = IntervalCreateTree();
// 依次插入30个节点
for (i = 0; i < 30; i++)
{
    fscanf(fp_in, "%d %d\n", &templow, &temphigh);
    x = IntervalCreateNode(templow, temphigh);
    IntervalInsert(T, x);
}
fclose(fp_in);

// 中序遍历
InOrder(fp_out1, T, T->root);
fclose(fp_out1);
```

（5）删除操作。随机选择3个区间进行删除，将删除的数据输出到delete_data.txt。中序遍历删除完成后的区间树，并将结果存入delete_data.txt。

```
// 随机选择其中三个区间进行删除
for (i = 0; i < 3; i++)
{
    delIndex = i * 10 + rand() % 11;
    delNode = IntervalFindNode(T, low[delIndex]);
    // 输出删除的数据
    fprintf(fp_out2, "%d %d %d\n", delNode->interval.low, delNode-
>interval.high, delNode->max);
    IntervalDelete(T, delNode);
}
// 删除完成后的中序遍历序列
InOrder(fp_out2, T, T->root);
fclose(fp_out2);
```

（6）查找操作。随机生成3个区间，其中一个区间取自(25,30)。对这3个区间进行搜索，搜索成功则返回一个与搜索区间重叠的区间，搜索失败返回空。将对应的搜索区间与搜索结果存入search.txt。

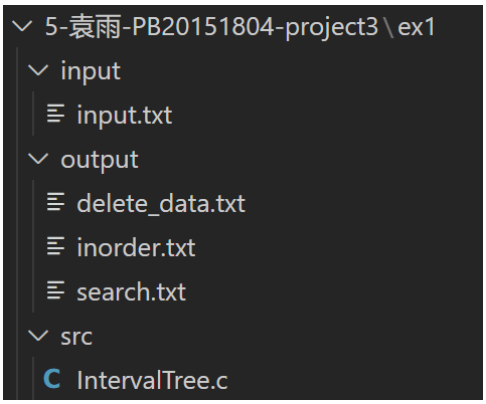
```
// 随机生成三个区间进行搜索
//(25,30)
```

```
searchlow[0] = 26 + rand() % (29 - 26 + 1);
searchhigh[0] = searchlow[0] + rand() % (29 - searchlow[0] + 1);
//[0,25]
searchlow[1] = rand() % (25 + 1);
searchhigh[1] = searchlow[1] + rand() % (25 - searchlow[1] + 1);
//[30,50]
searchlow[2] = 30 + rand() % (50 - 30 + 1);
searchhigh[2] = searchlow[2] + rand() % (50 - searchlow[2] + 1);
for (i = 0; i < 3; i++)
{
    x = IntervalCreateNode(searchlow[i], searchhigh[i]);
    searchNode = IntervalSearch(T, x);
    //存入搜索区间
    fprintf(fp_out3, "%d %d ", searchlow[i], searchhigh[i]);
    //存入搜索结果
    if (searchNode == T->NIL)
        fprintf(fp_out3, "");
    else
        fprintf(fp_out3, "%d %d", searchNode->interval.low, searchNode-
>interval.high);
    fprintf(fp_out3, "\n");
}
fclose(fp_out3);
```

四、实验结果与分析

1.实验结果

- 目录结构



- input.txt

Algorithm > 5-袁雨-PB20151804-project3 > ex1 > input > input.txt

```
16 25
40 42
20 23
42 50
0 11
39 47
23 25
24 25
46 50
47 48
15 23
3 21
1 3
14 22
17 22
22 25
18 22
25 25
31 48
8 22
10 14
19 20
38 40
6 22
5 25
13 15
36 50
4 24
12 17
32 33
```

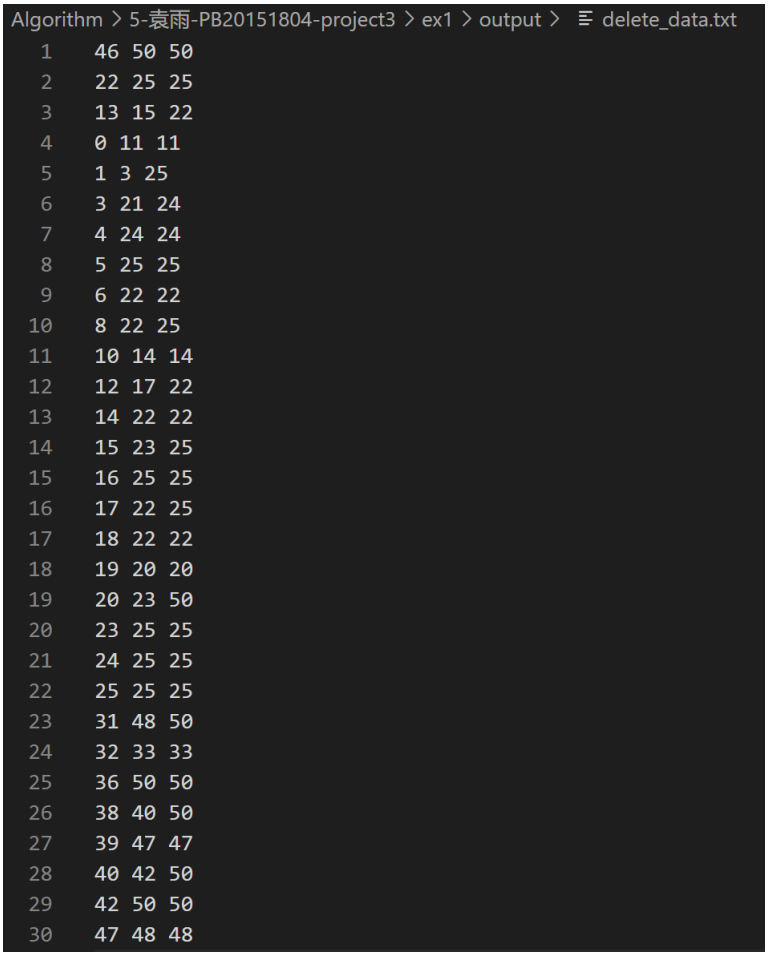
- inorder.txt

Algorithm > 5-袁雨-PB20151804-project3 > ex1 > output > inorder.txt

```
0 11 11
1 3 25
3 21 24
4 24 24
5 25 25
6 22 22
8 22 25
10 14 17
12 17 17
13 15 22
14 22 22
15 23 25
16 25 25
17 22 25
18 22 22
19 20 20
20 23 50
22 25 25
23 25 25
24 25 25
25 25 25
31 48 50
32 33 33
36 50 50
38 40 50
39 47 47
40 42 50
42 50 50
46 50 50
47 48 48
```

中序遍历序列，输出的是按低端点的次序排列的各区间，以及对应的max域。

- delete_data.txt



前三行为随机选出的要删除的3个结点，后面为删除完成后区间树的中序遍历序列。

- search.txt

每行的前两个数字为随机生成的区间的左端点与右端点。后两个数字为搜索结果。

对区间[27,29]，区间树中不存在与之重叠的区间，结果输出为空。

对区间[21,23]，与之重叠的区间为[20,23]。

对区间[32,45]，与之重叠的区间为[31,48]。