实验三: 区间树的实现

问题提出

本次实验欲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。

其中，输入和输出分别满足条件如下，

- 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

并且目录为

- 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

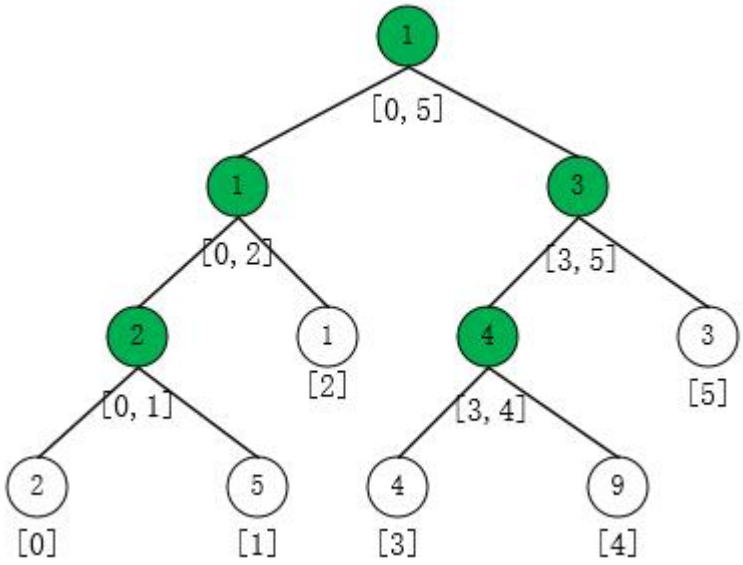
算法原理和实现

区间树 (Interval tree), 是一种二叉搜索树。它将一个区间划分成一些单元区间(即单个数据)，每个单元区间对应一个叶节点，非叶节点表示其所代表的子树对应的子区间,支持动修改和查询的数据结构. 区间树中，每个节点的子节点分别表示它的左右半区间。对于每一个非叶节点 ，设其对应区间为[a, b]，它的左子树表示的区间为[a, (a+b)/2], 右子树表示的区间为[(a+b)/2+1, b]。因此区间树是平衡二叉树, 其存储的空间复杂度是O(N)(确切说是4N-2), 操作的时间复杂度是O(logN).

区间树中，叶节点的value是数据，非叶节点的value是其对应区间的最小值。

下图给出一个以数组array构建的区间树的示例。

```
const int N = 6;
int array[N] = {2, 5, 1, 4, 9, 3};
```



构建区间树

区间树的构建采用递归方式，当前区间是单元区间时直接赋值给节点，否则，首先递归构建左右子树，然后将左右子树的最小值赋值给当前结点。其复杂度是O(N).

查询区间树

区间查询是指，用户输入一个区间，获取该区间的相关信息，比如区间的最大值，最小值，第k大的值等。对于本文的示例程序，是获取区间的最小值。

若查询区间与当前结点对应区间没有交叠，返回-1. 若查询区间包含在当前区间内，返回当前结点的value。否则，对子区间递归查询，并将查询结果合并返回。查询复杂度为O(logN).

递归中，将查询区间不加改变的传下去，每次查询只返回包含在查询区间内的结果即可。

节点更新

更新一个节点的value时，除了需要更新节点值，还要递归的更新其父节点，时间复杂度为O(logN).

区间更新

区间更新是指更新某个区间内的叶子节点的值，同时涉及到的非叶节点也需要同时更新。但是因为需要更新的节点很多，其复杂度将大于O(logN)。因此引入延迟标记的概念，这也是区间树的精髓。

延迟标记：每个节点增加一个标记，用于记录该节点是否进行了某种修改。当进行区间更新时，按照区间查询的方式，将更新区间划分为区间树中的节点，对这些修改这些。

实验结果

实验环境

```
OS: arm64-apple-darwin22.1.0, Compiler: clang-1400.0.29.202
```

结果输出如下，其中输入请参照输入文件。

1. 获取输入后的结果

```
After initialization, all intervals:
Interval(1, 12)  (No.1).
Interval(3, 19)  (No.2).
Interval(4, 4)   (No.3).
Interval(4, 25)  (No.4).
Interval(5, 6)   (No.5).
Interval(6, 7)   (No.6).
Interval(7, 12)  (No.7).
Interval(7, 17)  (No.8).
Interval(8, 24)  (No.9).
Interval(10, 17)         (No.10).
Interval(10, 18)         (No.11).
Interval(10, 24)         (No.12).
Interval(11, 21)         (No.13).
Interval(13, 14)         (No.14).
Interval(14, 15)         (No.15).
Interval(14, 16)         (No.16).
Interval(15, 22)         (No.17).
Interval(16, 17)         (No.18).
Interval(16, 20)         (No.19).
Interval(18, 19)         (No.20).
Interval(20, 21)         (No.21).
Interval(30, 30)         (No.22).
Interval(30, 41)         (No.23).
Interval(35, 40)         (No.24).
Interval(36, 40)         (No.25).
Interval(36, 46)         (No.26).
Interval(37, 37)         (No.27).
Interval(37, 41)         (No.28).
Interval(41, 41)         (No.29).
Interval(45, 47)         (No.30).
```

2. 在删除三个区间后的结果

```
Selected Interval(7, 17)
Selected Interval(18, 19)
Selected Interval(36, 46)


After delete 3 nodes, all intervals:
Interval(1, 12)  (No.1).
Interval(3, 19)  (No.2).
Interval(4, 4)   (No.3).
Interval(4, 25)  (No.4).
Interval(5, 6)   (No.5).
Interval(6, 7)   (No.6).
Interval(7, 12)  (No.7).
Interval(8, 24)  (No.8).
Interval(10, 17)         (No.9).
Interval(10, 18)         (No.10).
Interval(10, 24)         (No.11).
Interval(11, 21)         (No.12).
Interval(13, 14)         (No.13).
Interval(14, 15)         (No.14).
Interval(14, 16)         (No.15).
Interval(15, 22)         (No.16).
Interval(16, 17)         (No.17).
Interval(16, 20)         (No.18).
Interval(20, 21)         (No.19).
Interval(30, 30)         (No.20).
Interval(30, 41)         (No.21).
Interval(35, 40)         (No.22).
Interval(36, 40)         (No.23).
Interval(37, 37)         (No.24).
Interval(37, 41)         (No.25).
Interval(41, 41)         (No.26).
Interval(45, 47)         (No.27).
```

3. 查询的结果

```
Search and print of Interval(8, 30)
Overlapping of Interval(8, 30):
Interval(1, 12)
Interval(3, 19)
Interval(4, 25)
Interval(7, 12)
Interval(8, 24)
Interval(10, 24)
Interval(10, 18)
Interval(10, 17)
Interval(11, 21)
Interval(13, 14)
Interval(14, 16)
Interval(14, 15)
Interval(15, 22)
Interval(16, 20)
Interval(16, 17)
Interval(20, 21)
Interval(30, 41)
Interval(30, 30)
Search and print of Interval(22, 44)
Overlapping of Interval(22, 44):
Interval(4, 25)
Interval(8, 24)
Interval(10, 24)
Interval(15, 22)
Interval(30, 41)
Interval(30, 30)
Interval(35, 40)
Interval(36, 40)
Interval(37, 41)
Interval(37, 37)
Interval(41, 41)
Search and print of Interval(26, 28)
Overlapping of Interval(26, 28):
```

总结

1. 树状数组逻辑上是一棵普通的树，而区间树逻辑上是一颗红黑树；
2. 插入和删除的复杂度都为$O(lg\ n)$
3. 在一些具体的应用中，需要对输入的数据进行离散化，以减小树的大小，此时需要注意离散前和离散后的数据对应。此外，在某些应用中，需要使用lazy思想，在一些操作中先不对线段树进行更新，而是推迟到查找的过程中，在查找的过程中进行更新。