lab3红黑树和区间树

实验内容及要求

■实验3.1：红黑树

□实现红黑树的基本算法， 分别对整数 n =20、40、60、80、100，随机生成n 个互异的正整数（K1, K2, K3, ……, Kn），以这 n 个正整数作为结点的关键字，向一棵初始空的红黑树中依次插入n 个节点，统计算法运行所需时间 ，画出时间曲线。

□随机删除红黑树中n/4个结点，统计删除操作所需时间，画出时间曲线图。

■实验3.2：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，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。实现区间树的插入、删除、遍历和查找算法。

实验设备和环境

- 编译运行环境
  - Windows10-mingw-w64
  - vscode
- 电脑配置

```
计算机名:  DESKTOP-NAIGI2Q
操作系统:  Windows 10 家庭中文版 64 位 (10.0，版本 18363)
语言:  中文(简体) (区域设置: 中文(简体))
系统制造商:  Dell Inc.
系统型号:  G7 7588
BIOS:  1.9.0
处理器:  Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz (12 CPUs), ~2.2GHz
内存:  16384MB RAM
页面文件:  9058MB 已用，10319MB 可用
```

实验方法和步骤

计时函数

- 和lab2相同，采用了`<windows.h>`中的`QueryPerformance`来计时，精度高，避免了之前数据量较小时时间为0的情况
- 开始计时

```
LARGE_INTEGER t1, t2, tc;
QueryPerformanceFrequency(&tc);
QueryPerformanceCounter(&t1);
```

- 运行需要计时的部分
- 结束计时并计算用时，以微秒为单位

```
QueryPerformanceCounter(&t2);
double time = (double)(t2.QuadPart - t1.QuadPart) / (double)tc.QuadPart;
```

红黑树-主要文件

my_header.h

- 声明所需要的头文件和命名空间
- 给出当前目录的基础路径`base_path`，后续的文件读写都是以该路径为基础
- 定义一些文件输出流

generate_random_nums.h

- `make_generate`函数用来生成n个互异的随机整数（n =20、 40、 60、 80、 100 ），并写入文件`input/input.txt`

RB_Tree.h

- 用枚举体`RB_Color`表示红色和黑色
- 定义红黑树的结点类`RB_Node`，包含
  - 颜色`color`
  - 类型为 T 的关键字`key`
  - 双亲结点`parent`
  - 左孩子`left`
  - 右孩子`right`
- 通过结点类定义红黑树类`RB_Tree`，其中包含多个方法
  - 红黑树的根结点`root`
  - 红黑树的`NIL`结点
  - 中序遍历`In_Order_Walk`，用于输出红黑树的中序序列
  - 左旋`Left_Rotate`，用于修改指针结构并保持二叉搜索树性质
  - 右旋`Right_Rotate`，和左旋对称
  - 插入`RB_Insert`，用于向红黑树中插入一个新结点
    - 辅助过程`RB_Insert_Fixup`，用于维持插入结点后的红黑树的红黑性质
  - 删除`RB_Delete`，用于从红黑树中删除一个结点
    - 辅助过程`RB_Delete_Fixup`，用于通过改变颜色和执行旋转来恢复红黑性质
    - 辅助过程`RB_Transplant`，用于将另一棵子树替换一棵子树并成为其双亲的孩子结点
    - 辅助过程`get_min`，用于获得以传入参数作为根结点的子树中关键字最小的结点
  - 搜索`RB_Search_Node`，用于根据传入的关键字搜索和关键字匹配的结点
  - 清空`RB_Empty`(后面没用上)
- 上述的结点类和红黑树类都是模板类(`template <class T>`)，且结点的关键字的类型为 T ，这样可以通过不同类型的模板构建不同类型的关键字的结点，这使得代码易于扩展，后续的区间树只需重新定义一个模板 T 即可

- 在实现的过程中发现课本的`RB_Insert_Fixup`和`RB_Delete_Fixup`的伪代码书写有一定的问题，在`RB_Insert_Fixup`的case2和`RB_Delete_Fixup`的case3部分的`else if() other`应改成`else {if() other}`,因为按照书中的理论，对于`RB_Insert_Fixup`，如果按照书中的写法，则每次循环都会执行一次case3，这是不正确的；同理对于`RB_Delete_Fixup`，按照书中的写法每次循环都会执行一次case4。

main.cpp

- 调用随机数生成函数生成要求的随机数
- 读取生成的随机数，如果发现同一个n出现了相同的随机数（这是一个小概率事件）则报告错误并退出程序（错误码为1），需要重新运行
    - 通过`c++`的set库来判断去重后的容器和原容器长度是否相同，以此来判断是否有重复的元素
- 以红黑树为模板建立包含5棵红黑树的容器，并根据读入的数据逐个插入，以此来建立红黑树，并分别计时，写入`output/time1.txt`
- 对构建好的红黑树分别进行中序遍历，并将结果写入文件`output/inorder.txt`
- 从之前读入的数据中随机挑出n/4个，将对应的结点删除，并分别计时，写入`output/time2.txt`
    - 这里通过动态生成存放n个数据的容器索引的随机数来进行随机挑选
- 对删除结点后的红黑树分别进行中序遍历，并将删除的结点关键字和中序遍历的结果写入文件`output/delete_data.txt`

区间树-主要文件

由于区间树是基于红黑树的数据结构的扩张，故二者的源码大部分都是相同的，下面只说明扩张的部分

my_header.h

- 添加区间类

```
class section{
public:
    int low;
    int high;
    int max = 0;
};
```

  以该类作为模板构建红黑树，其中max初始化为0便于自底向上维护max域。这个类相当于课本中的int域以及max域

- 对第一部分的思路作了一点改进，声明了一个全局的`vector<section>`用来存放生成的随机数据，免去了重新从文件读入的麻烦

generate_random_section.h

- 首先生成[0, 24], [30, 49]的30个互异的随机整数作为左端点（以25,50作为左端点无法满足右端点值大于左端点值的要求）
- 接着对于上面的每个左端点，随机生成对应的整数右端点
    - 若左端点low属于[0, 24], 则右端点high属于[low + 1, 25]
    - 若左端点low属于[30, 49], 则右端点high属于[high + 1, 50]
- 用类似的方式生成要搜索的3个区间
    - 前两个区间的左端点low属于[0, 49],右端点high属于[low + 1, 50]
    - 第三个区间的左端点low属于[26, 28],右端点high属于[low + 1, 29]

RB_Tree.h

- `get_max`，用来维护max域，从左孩子的max、右孩子的max和当前结点的high中挑选出最大值作为当前结点的max
- `RB_Interal_Search`,遍历区间树，找出区间树中与所寻找的区间重叠的一个结点，若不存在则返回`NIL`
- 旋转操作的末尾添加

```
x->key.max=get_max(x->key.high, x->left->key.max, x->right->key.max);
y->key.max=get_max(y->key.high, y->left->key.max, y->right->key.max);
```

  即在旋转完成后自下而上更新max域

- `RB_Max_Fixup`,用来维护区间树的max域。从当前结点向上一直到根结点更新max域
- `RB_Insert`
  - 插入前将结点的max域设为对应的high值
  - 插入完成后调用`RB_Max_Fixup`更新max域
- `RB_Delete`
  - 删除完成后调用`RB_Max_Fixup`更新max域
- 在各个方法内部进行关键字的比较时，应比较`key.low`，即比较左端点的值

main.cpp

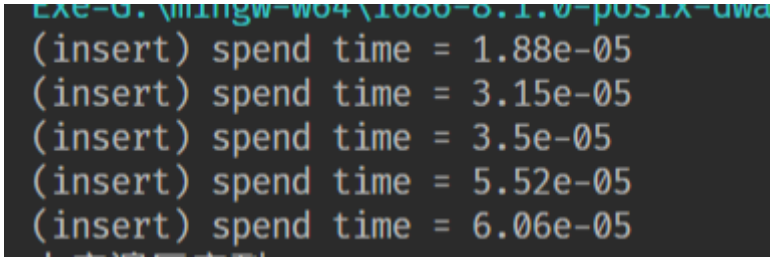
- 生成30个随机区间，并根据这些区间建立区间树
- 对构建好的区间树进行中序遍历，并将区间树的中序遍历序列写入`output/inorder.txt`
- 随机生成3个索引，用以删除三个区间，将删除的数据和删除后区间树的中序遍历序列写入`output/delete_data.txt`
- 从删除后的区间树中搜索3个随机区间，并将搜索的区间和搜索的结果写入`output/search.txt`

实验结果和分析

结果截图

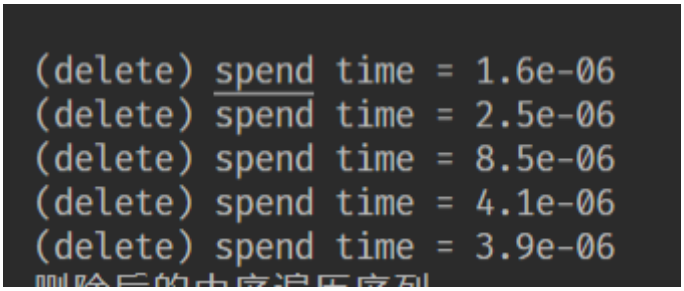
红黑树

- 插入花费的时间



- 构建好的红黑树的中序遍历序列（n=20）

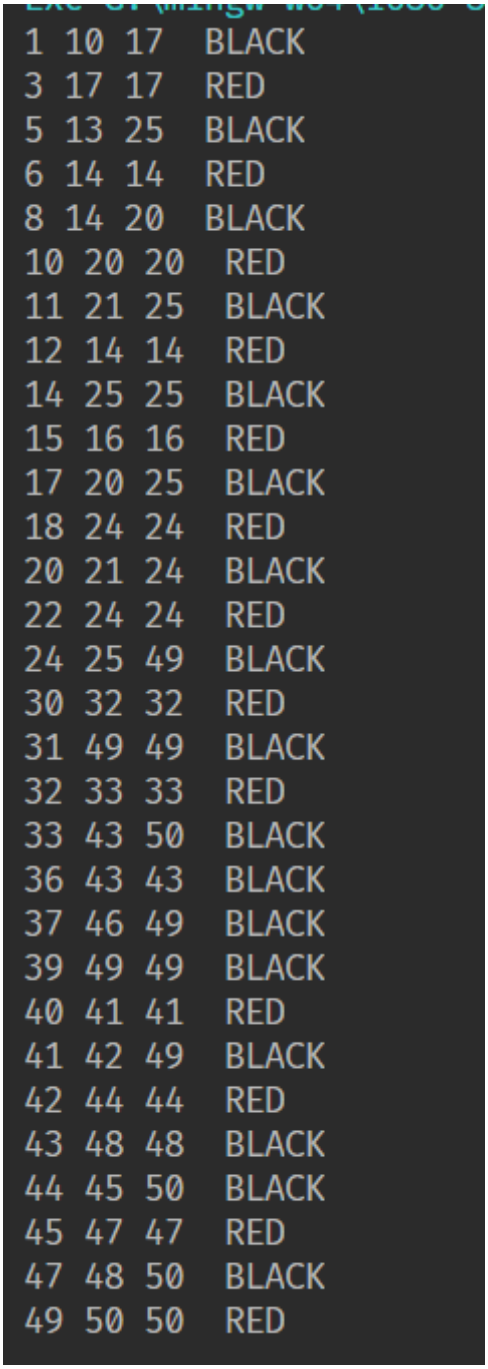
- 删除花费的时间



- 删除的结点以及删除后的中序遍历序列（n=20）

区间树

- 构建好的区间树的中序遍历序列



- 删除的数据以及删除完成后的区间树的中序遍历序列

```
删除的数据
44 45
12 14
41 42
删除后的中序遍历序列
1 10 17  BLACK
3 17 17  RED
5 13 25  BLACK
6 14 14  RED
8 14 20  BLACK
10 20 20  RED
11 21 25  BLACK
14 25 25  BLACK
15 16 16  RED
17 20 25  BLACK
18 24 24  RED
20 21 24  BLACK
22 24 24  RED
24 25 49  BLACK
30 32 32  RED
31 49 49  BLACK
32 33 33  RED
33 43 50  BLACK
36 43 43  BLACK
37 46 49  BLACK
39 49 49  BLACK
40 41 41  RED
42 44 44  BLACK
43 48 48  BLACK
45 47 50  BLACK
47 48 50  BLACK
49 50 50  RED
```

- 搜索

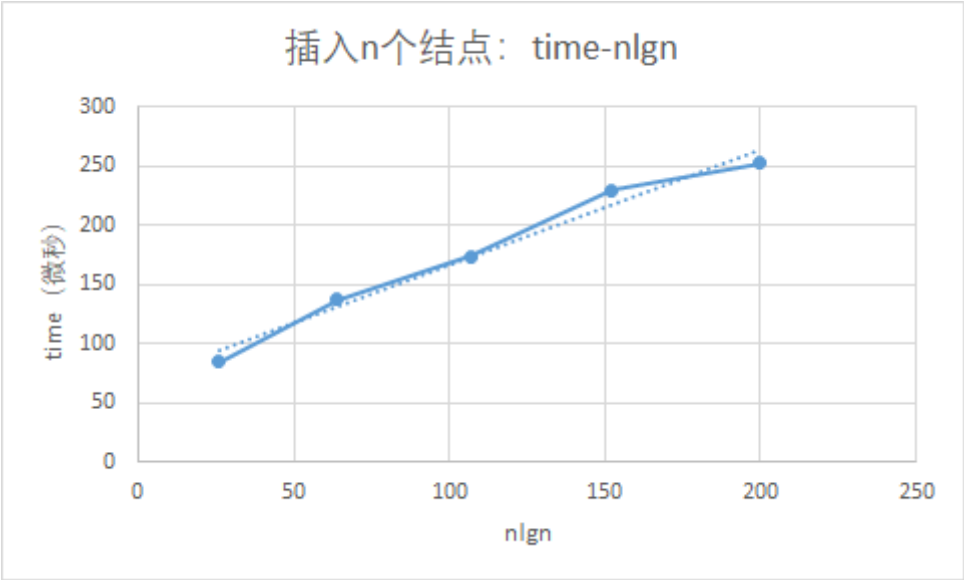
```
search section: 43 45
find result: 33 43
search section: 33 34
find result: 33 43
search section: 28 29
find NULL
```

时间复杂度分析

取某次具有代表性的数据作图

插入

```
8.47e-05
0.0001374
0.0001723
0.00023
0.0002523
```

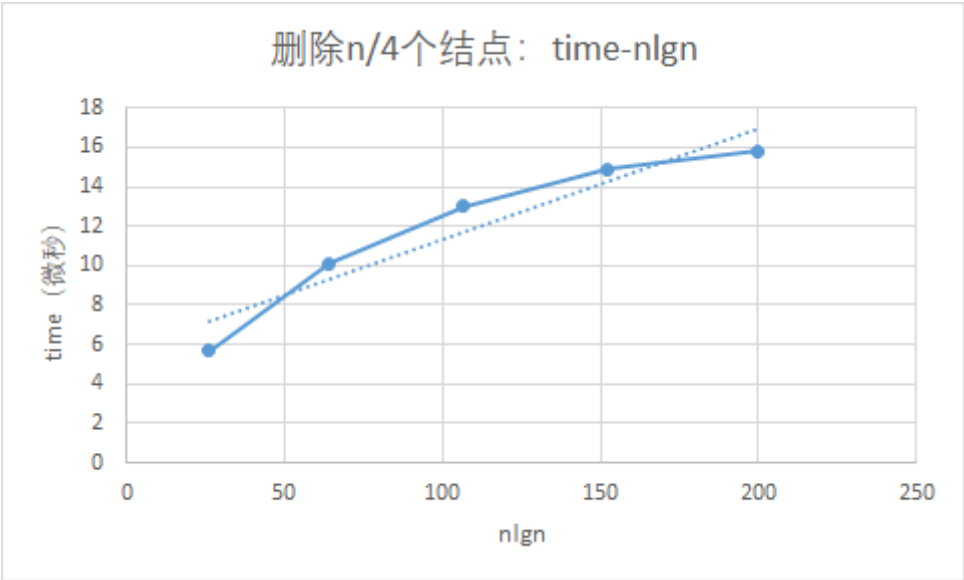


向一棵含n个结点的红黑树中插入一个新结点需要O（lgn）的时间，而构建红黑树的过程就是依次向含有0、1、2。。。n - 1个结点的红黑树插入一个新结点，故理论时间复杂度为O（lg1 + lg2 + lg3 + lg(n-1) = O(nlgn)

由图像可知，实际的时间复杂度和理论较为吻合

删除

```
5.7e-06
1.01e-05
1.3e-05
1.49e-05
1.58e-05
```



从一个含有n个结点的红黑树中删除一个结点所需的时间为O（lgn），实验中逐个删除了n/4个结点，故理论时间复杂度为O（lgn + lg（n-1）+ lg（n-2）+。。。+lg（n/4 + 1））=O（nlgn）

由图像可知，实际的时间复杂度和理论较为吻合

实验总结

- 通过本次实验，对二叉搜索树、红黑树、区间树的实现和有关操作有了更加深入的理解
- 对数据结构扩张的方法有了更加深入的体会
- 对于伪代码到具体实现的转换有了更加深入的体会，第一次尝试利用C++ 的类、模板类来构建结点和树，体会到了模板编程的强大之处
- 对于随机数的生成有了更深入的认识，这次实验要求生成各种各样限制的随机数，不同的部分需要不同的随机数范围以及随机数生成方法