红黑树和区间树实验报告

实验设备和环境

- 硬件条件:一台 PC 机，
- 软件条件: mac 操作系统，VS code 编辑器，gcc 编译器 。

实验目的

- 熟练掌握红黑树数据结构，能够使用C语言实现它的基本操作
- 了解数据结构的扩张，在红黑树的基础上实现区间树的基本操作。

实验内容与要求

内容

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
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。

要求

1. 文件目录要求

    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、ex2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

2. 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。 比较实际复杂度和理论复杂度是否相同，给出分析。

实验方法与步骤

1. 构建文件目录

   建立一个根文件夹实验需建立根文件夹，文件夹名称为:编号-姓 名-学号-project2，在根文 件夹下需包括实验报告和 ex1 子文件夹， 和 ex2 文件夹。实验报告命名为编号-姓名-学号-project2.pdf， ex1

   子文件夹和 ex2 子文件夹又包含 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

2. 红黑树
   - 先写一个按照 制定范围，指定个数，产生随机数的算法，将结果输出到`input.txt` 文件中。
   - 然后开始写创建红黑树的 算法。这里参照书上的算法，使用将新结点插入的方式产生新的红黑树，插入位置是较好确定的，关键是`RB_insert_fixup()` 函数 学要对红黑树结点的颜色进行维护。
   - `RB_insert_fixup()` 的实现 。这里一共分为三种情况，在代码里有详细注释。以父亲为左儿子为例。第一种，父亲的兄弟结点为红，此时需要将问题上移到祖父结点。第二种 父亲兄弟结点为黑色，uncle为右儿子，此时进行一下左旋，已到左儿子。第三种uncle直接是左儿子，此时直接右旋
   - 然后就是删除算法，此算法的关键是找到真实的删除结点。这于删除的Z的位置有关。可以根据不同情况找到。
   - 然后就是`RB_del_fixup()` 的实现 。这有四种情况。在代码中有详细注释，类似插入的fixup函数，对每种情况做相应的颜色调整和旋转 。
   - 然后后实现一个中序遍历的函数，进行输出。
   - 最后在主函数中进行时间记录，以及写输出结果。
3. 区间树

   这个的实现在红黑树的基础上，其实容易多了，只需要维护 `Node.max` 的值就行。

   下面详细介绍 ，在各个操作中对 `Node.max` 的维护。

   - 在insert函数，和delete 函数中，专门写了一个 `void INT_max_fixup(INT_tree* T , Node* Curr)` 函数。

```
void INT_max_fixup(INT_tree* T , Node* Curr){
     if(Curr == T->nil) return;
     while(Curr->parent != T->nil){
         Curr->parent->max = getmax(Curr->max,Curr->parent->max);
         Curr = Curr->parent;
     }
     return ;
}
```

- 而在左旋右旋中，则在函数内直接维护，因为左旋 右旋操作，维护max 的代价为O(1) .较容易实现。

  只需比较一下最大值即可

```
 //调整max
     Curr->max = getmax(Curr->interval.right,getmax(Curr->l_child->max,Curr->r_child->max));
     tmp->max = getmax(tmp->interval.right,getmax(tmp->l_child->max,tmp->r_child->max));
```

实验结果与分析

1. 下面先展示红黑树运行结果。（可以只产看规模为20的）其他的规模太大，一屏幕 ，难以展示完全。

```
1 > output > inorder.txt
1    140  141  366  732  1419  1440  1726  2017  2080  2129  2235  2307  2470  3064  3149  3299  3801  4027  4770  4993
2    165  331  468  568  656  775  938  1293  1343  1431  1866  1981  2048  2147  2247  2456  2640  2662  2861  2940  3032
3    66  70  87  284  301  309  374  412  435  490  586  745  975  1049  1097  1230  1252  1313  1341  1568  1577  1793  190
4    6  107  222  273  285  330  378  572  682  703  833  835  862  869  1006  1015  1090  1199  1211  1232  1281  1411  151
5    22  44  45  53  65  81  95  101  154  176  189  302  333  355  406  446  482  532  667  852  853  962  987  991  996  1
6
```

中序遍历结果

```
1 > output > delete_data.txt
1    规模为 20 的问题 ,随机删除的数据为: 2080 4027 141 2129 3801
2    删除后中序遍历的结果为: 140  366  732  1419  1440  1726  2017  2235  2307  2470  3064  3149  3299  4770  4993
3    规模为 40 的问题 ,随机删除的数据为: 4678 4482 568 4283 3386 4671 1981 2640 4517 331
4    删除后中序遍历的结果为: 165  468  656  775  938  1293  1343  1431  1866  2048  2147  2247  2456  2662  2861  2940  3032  30
5    规模为 60 的问题 ,随机删除的数据为: 1049 3910 2903 1230 2792 66 2736 4792 435 2724 2611 284 4282 3529 2701
6    删除后中序遍历的结果为: 70  87  301  309  374  412  490  586  745  975  1097  1252  1313  1341  1568  1577  1793  1907  193
7    规模为 80 的问题 ,随机删除的数据为: 1911 4465 572 3595 1199 4122 3894 4632 1090 1211 682 1015 4000 107 1232 2621 1825 3854 1
8    删除后中序遍历的结果为: 6  222  273  285  330  378  703  833  835  862  869  1006  1281  1411  1513  1558  1563  1619  1659
9    规模为 100 的问题 ,随机删除的数据为: 3054 3686 406 3312 1780 2285 2595 95 189 1502 3066 2477 4011 4297 853 2873 1662 4539 38
10   删除后中序遍历的结果为: 22  44  45  53  65  81  101  154  176  302  333  355  446  482  532  667  987  991  996  1078  1133
```

随机删除 N/4的结果

```
x1 > output > time1.txt
1    The 20 scale problem creat RB_tree costs 0.000015 s ...
2    The 40 scale problem creat RB_tree costs 0.000007 s ...
3    The 60 scale problem creat RB_tree costs 0.000012 s ...
4    The 80 scale problem creat RB_tree costs 0.000015 s ...
5    The 100 scale problem creat RB_tree costs 0.000019 s ...
```

建树运行时间结果

```
x1 > output > ≡ time2.txt
1   The 20 scale problem delete N/4 RB_tree randomly costs 0.000001 s ...
2   The 40 scale problem delete N/4 RB_tree randomly costs 0.000003 s ...
3   The 60 scale problem delete N/4 RB_tree randomly costs 0.000002 s ...
4   The 80 scale problem delete N/4 RB_tree randomly costs 0.000003 s ...
5   The 100 scale problem delete N/4 RB_tree randomly costs 0.000004 s ...
```

删除运行时间结果

2.下面先展示区间树运行结果。

```
ex2 > output > ≡ inorder.txt
 1    1  12   12
 2    2  12   12
 3    3  19   19
 4    4  16   16
 5    5  17   17
 6    6  9   9
 7    8  12   19
 8    10  12   12
 9    11  19   19
10    12  17   25
11    13  22   22
12    16  22   22
13    17  22   23
14    19  20   20
15    20  23   23
16    21  25   25
17    22  25   25
18    23  24   25
19    24  25   25
20    30  40   50
21    31  34   34
22    32  45   50
23    33  45   45
24    37  50   50
25    39  46   46
26    40  43   50
27    42  50   50
28    45  46   50
29    46  47   49
30    47  49   49
```

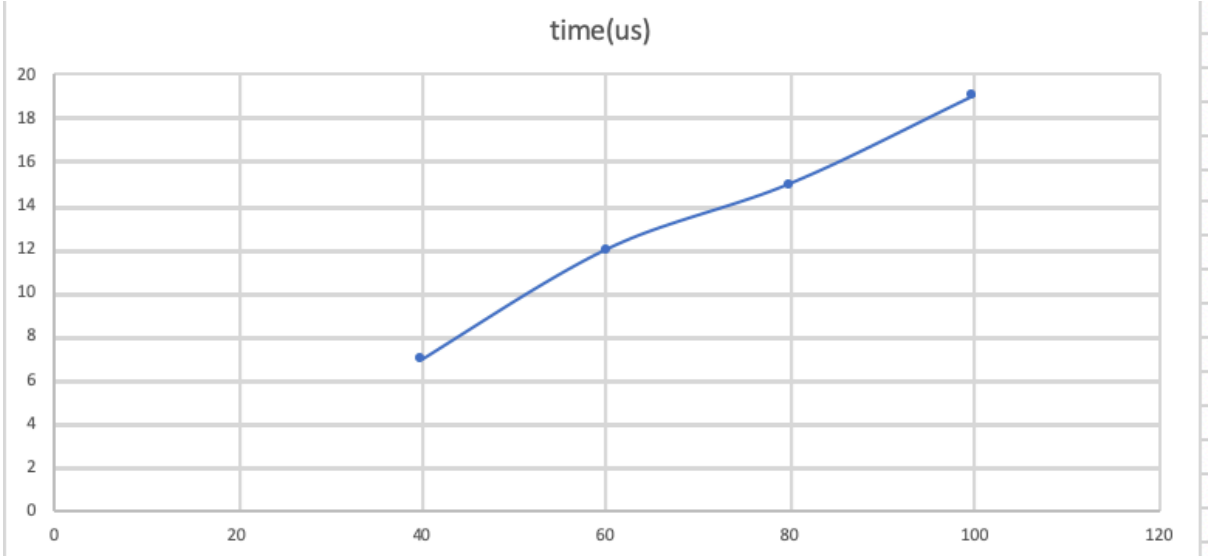
```
x2 > output > ≡ delete_data.txt
1    随机删除的数据为:
2    39  46
3    37  50
4    46  47
5    删除后中序遍历的结果为:
6    1  12   12
7    2  12   12
8    3  19   19
9    4  16   16
10   5  17   17
11   6  9   9
12   8  12   19
13   10  12   12
14   11  19   19
15   12  17   25
16   13  22   22
17   16  22   22
18   17  22   23
19   19  20   20
20   20  23   23
21   21  25   25
22   22  25   25
23   23  24   25
24   24  25   25
25   30  40   50
26   31  34   34
27   32  45   50
28   33  45   45
29   40  43   50
30   42  50   50
31   45  46   50
32   47  49   49
```

```
x2 > output > ≡ search.txt
1    搜索的区间为 :
2    26  27
3    14  37
4    79  93
5    搜索的结果为 :
6    Null
7    30  40
8    Null
```

中序遍历　　删除三个区间　　搜索区间

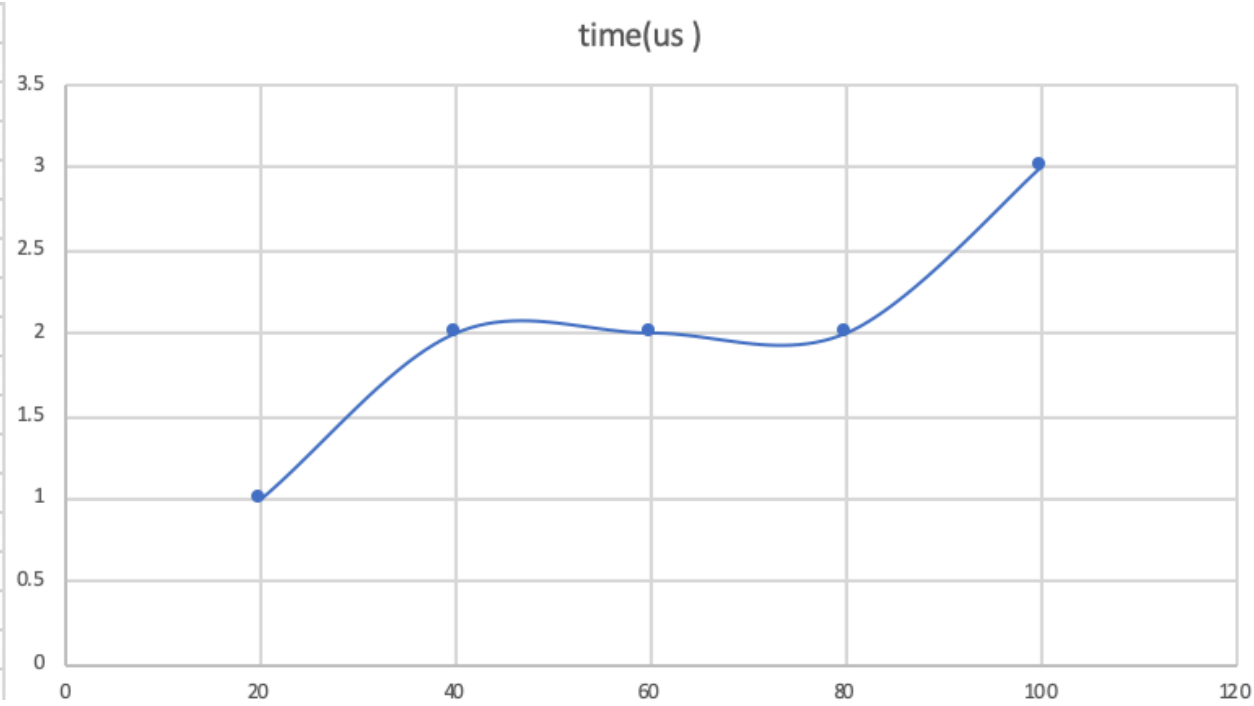
3. 时间曲线，与时间复杂度分析 。

   创建红黑树所需时间曲线。



分析： 由于 创建第一颗树，可能由于 在内存中miss数据，所以第一课创建的树的时间与后面的规模没有可比性。所以，为了控制变量，在这里20规模问题的时间不画入图中。再上图中 可见图形近似线性，稍有下凸。而插入n个结点的算法时间复杂度应为 O(nlogn)，图形符合的还是比较好的。

随机删除N/4结点所用时间 曲线。



时间复杂度分析 。

随机删除n/4个结点的算法时间复杂度为，O(nlogn) 。而得到的图形 随着n增大，下凸性变的较明显。但由于在规模小时，时间较短，得到的误差也就大，所以得到的图形前期不太符合 O(nlogn) 。

4. 实验总结 。

   该实验很好的帮助我了解了红黑树数据结构，也进一步熟悉了树结构的使用 。该实验共将近1000行代码。包含了红黑树的很多操作。插入，删除，遍历，查找，左旋，右旋等。

实验中遇到的问题。实验中经常会有`segmentation fault` 。这往往是指针指错造成的，每当遇到这种问题，往往是边界条件没有考虑周全。人肉debug的能力也有所锻炼。