作业二：BST 排序算法的实现

张竣凯
3210300361
数学与应用数学

2022 年 10 月 19 日

二叉排序树（Binary Sort Tree），又称二叉查找树（Binary Search Tree），亦称二叉搜索树。是数据结构中的一类。在一般情况下，查询效率比链表结构要高。

——摘自百度百科《二叉排序树》

1 设计思路

1.1 阅读并理解课本所提供的头文件 BinarySearchTree.h 里头编写的所有代码

例如：class BinarySearchTree 以及其功能函数等

1.2 阅读并理解课本所提供的头文件 dsexceptions.h 里头编写的所有代码

例如：class UnderflowException、IllegalArgumentException 等

1.3 在测试程序 main.cpp 中编写 BSTSorting 函数

用于将数组 _arr 排序

1.4 在测试程序 main.cpp 中编写 test 函数

测试用的代码大同小异，故编写 test 函数以减少复制

1.5 在测试程序 main.cpp 中编写 main 函数

调用事先写好的 test 函数，且可控制数组的大小

1.6 适当地添加注释，尽可能地使用引用 & 以及充分考虑必要的 const 限制

以便于理解代码，减少内部复制和提高安全性

1.7 分别测试 _mode = 0 的最坏效率和 _mode = 1 时的平均效率

测试完毕后将结果记录下来，并进行时间效率的分析

2 额外函数

2.1 BSTSorting 函数

```
1  template <typename Comparable>
2  void BSTSorting(std::vector<Comparable> &_arr, int _mode = 0)
3  {
4      if(_mode == 1) //当数组进行乱序
5      {
6          random_shuffle(_arr.begin(),_arr.end()); //将数组乱序
7          cout << "开始进行 _mode = 1（乱序）的排序，数组的大小为" << _arr.size()
8               << "，重复进行10次后取平均值。" << endl;
9          BinarySearchTree<Comparable> bst; //构造一个二叉搜索树bst
10         double average_duration; //创建一个double类的平均时长
11         for(int j=1; j<=10; j++) //为了得到平均时长因此使用for循环
12         {
13             clock_t begin_time = clock(); //将起始时间赋值于begin_time
14             for(int i=0; i < _arr.size(); i++) //将_arr中的元素一个一个插入bst
15                 bst.insert(_arr[i]);
16             clock_t end_time = clock(); //将终止时间赋值于end_time
17             average_duration += (double)(end_time - begin_time)/CLOCKS_PER_SEC;
18             //累加每一次循环所得到的时长
19         }
20         average_duration /= 10; //上面进行了10次循环，因此这里除10
21         cout << "排序结束，平均排序时长为" << average_duration << "秒。\v" << endl;
22     }
23     else //当数组不进行乱序
24     {
25         cout << "开始进行 _mode = 0（不乱序）的排序，数组的大小为" << _arr.size()
26              << "。" << endl;
27         BinarySearchTree<Comparable> bst; //构造一个二叉搜索树bst
28         clock_t begin_time = clock(); //将起始时间赋值于begin_time
29         for(int i=0; i < _arr.size(); i++) //将_arr中的元素一个一个插入bst
30             bst.insert(_arr[i]);
31         clock_t end_time = clock(); //将终止时间赋值于end_time
32         double duration = (double)(end_time - begin_time)/CLOCKS_PER_SEC;
33         ////创建一个double类的时长，并将终止时间减去起始时间赋值于它
34         cout << "排序结束，排序时长为" << duration << "秒。\v" << endl;
35     }
36 }
```

2.2 test 函数

```
1  void mode0_test(int n) //用于测试 _mode = 0 的最坏效率
2  {
3    vector<double> _arr; //构造一个向量_arr
4
5    for(int i = 1; i <= n; i++) //通过for循环将元素1到n依次插入_arr
6      _arr.push_back(i); //数组中的元素正序时，时间效率最低
7
8    BSTSorting(_arr, 0); //调用BSTSorting函数来排序_arr
9  }
10
11 void mode1_test(int n) //用于测试 _mode = 1 的平均效率
12 {
13   vector<double> _arr; //构造一个向量_arr
14
15   for(int i = 1; i <= n; i++) //通过for循环将元素1到n依次插入_arr
16     _arr.push_back(i);
17
18   BSTSorting(_arr, 1); //调用BSTSorting函数来排序_arr
19 }
```

2.3 main 函数

```
1  int main()
2  {
3    for(int i = 10000; i <= 50000; i+=10000) //调用测试函数，进行五次不同大小的数组排序
4          mode0_test(i);
5
6    for(int i = 10000; i <= 50000; i+=10000) //调用测试函数，进行五次不同大小的数组排序
7          mode1_test(i);
8
9    return 0;
10 }
```

3 测试说明与结果

将 main.cpp、BinarySearchTree.h 与 dsexceptions.h 三个文件放入项目 bst 后，在该目录下进入终端，找到 bst 的路径后，输入 g++ -o test main.cpp 后回车，接着输入./test 后回车，将会出现以下结果：

开始进行 _mode = 0（不乱序）的排序，向量的大小为 10000。
排序结束，排序时长为 0.606883 秒。

开始进行 _mode = 0（不乱序）的排序，向量的大小为 20000。
排序结束，排序时长为 2.21933 秒。

开始进行 _mode = 0（不乱序）的排序，向量的大小为 30000。
排序结束，排序时长为 4.46622 秒。

开始进行 _mode = 0（不乱序）的排序，向量的大小为 40000。
排序结束，排序时长为 8.07494 秒。

开始进行 _mode = 0（不乱序）的排序，向量的大小为 50000。
排序结束，排序时长为 12.6013 秒。

开始进行 _mode = 1（乱序）的排序，向量的大小为 10000，重复进行 10 次后取平均值。
排序结束，平均排序时长为 0.0019587 秒。

开始进行 _mode = 1（乱序）的排序，向量的大小为 20000，重复进行 10 次后取平均值。
排序结束，平均排序时长为 0.004505 秒。

开始进行 _mode = 1（乱序）的排序，向量的大小为 30000，重复进行 10 次后取平均值。
排序结束，平均排序时长为 0.007485 秒。

开始进行 _mode = 1（乱序）的排序，向量的大小为 40000，重复进行 10 次后取平均值。
排序结束，平均排序时长为 0.0101757 秒。

开始进行 _mode = 1（乱序）的排序，向量的大小为 50000，重复进行 10 次后取平均值。
排序结束，平均排序时长为 0.0131516 秒。

不难从上面的测试结果发现，在数组不进行排序的情况下（_mode = 0），当数组的大小分别为 10000、20000、30000、40000 及 50000 时，排序所需的时间大约分别为 0.61 秒、2.22 秒、4.47 秒、8.08 秒及 12.6 秒, 因此得到 _mode = 0 的最坏效率为 $O(n^2)$。

再来，在数组进行排序的情况下（_mode = 1），当数组的大小分别为 10000、20000、30000、40000 及 50000 时，排序所需的平均时间大约分别为 $1.96 * 10^{-3}$ 秒、$4.50 * 10^{-3}$ 秒、$7.49 * 10^{-3}$ 秒、

$1.02 * 10^{-2}$ 秒及 $1.32 * 10^{-2}$ 秒，因此得到 _mode = 1 的平均效率为 $O(nlogn)$。

从结果来看，没经过乱序的数组进行二叉排序所需的时间是远大于经过乱序的数组的，因此在日后若有需要用到时，应当选择先进行乱序后再进行二叉排序，方能大幅地提升算法的效率。

4 内存泄漏检查

承接上个部分，继续在终端中输入 valgrind ./test 后回车，将会出现以下结果

==2479==
==2479== HEAP SUMMARY:
==2479== in use at exit: 0 bytes in 0 blocks
==2479== total heap usage: 300,164 allocs, 300,164 frees, 14,089,926 bytes allocated
==2479==
==2479== All heap blocks were freed – no leaks are possible
==2479==
==2479== Use –track-origin=yes to see where uninitialised values come from
==2479== For lists of detected and suppressed errors, rerun with: -s
==2479== ERROR SUMMARY: 534 errors from 62 contexts (suppressed: 0 from 0)