数据结构大作业-黄河源-201800820087

作业写了两种版本，Java和Python。运行环境：IntelliJ——JDK1.8，PyCharm——python3.7。

为了搞懂数据结构的基本原理并掌握其底层实现与应用，本篇作业TreeSet和Map数据结构第一次编写时均使用Java编写底层原理，而非选择简单不易报错的Python调用各种封装好的库。Java成功实现之后，使用Python**不调用任何封装好的库函数**，从底层逻辑开始实现了TreeSet和Map自定义类。

代码分为三个文件夹：

TreeSet文件夹内，com文件夹用于打印树形结构，Set.java是interface接口文件，TreeSet.java是实现接口功能的TreeSet类代码。Main.java是测试TreeSet接口功能的文件，可以直接运行。**Person.java是自定义类，用于测试TreeSet元素类型泛型<E>为自定义类时的排序功能，可以直接运行。**

Map文件夹内的Map.java是interface接口文件，BST_Map.java是实现接口功能的Map类代码，Main.java是测试Map类所有接口功能的文件，可以直接运行。

PythonSetMap文件夹内是用Python实现的TreeSet和Map自定义类，TreeSet.py和Map.py都可以直接运行。

用二叉搜索树实现TreeSet数据结构

TreeSet数据结构即在树结构的基础上实现集合set的去重功能，由于二叉搜索树在添加相同元素时本身就自带去重效果（遇到相同元素值时，一般用新的值覆盖旧的值，不会新增相同的节点，故实现去重效果），所以TreeSet的实现绝大多数可以**直接继承BST的方法。**其方法的实现思路上课已经讲过，此处不赘述，已经写在文档最后的附录中。

因为TreeSet兼具去重和排序功能，故对于节点类型为**自定义类**的排序，需要重写Compare方法，自定义比较器进行排序。代码中以Person为例，person有两种需要排序的属性，age和height，排序时现根据age，age相同时再根据height排序。person的id属性不参与排序，只是表示其独有身份。

1.当节点元素element泛型为常见变量类型的代码演示：Main.java

<E>泛型在演示中取Integer：

```
生成TreeSet的二叉搜索树形状为
4┐
 │
┌8┐
│ │
5  9┐
    │
    12
TreeSet的size为5
TreeSet是否为空：false
TreeSet是否包含元素4：true
TreeSet是否包含元素999：false
这是中序遍历-按元素从小到大的顺序遍历
4 5 8 9 12
这是中序遍历-遍历部分元素-到8终止遍历
4 5 8
删除节点4，8之后的二叉搜索树TreeSet形状为
5┐
 │
 9┐
  │
  12
清空TreeSet
TreeSet是否为空：true
TreeSet的size为0

Process finished with exit code 0
```

2.当节点元素element泛型为自定义类时：运行Person.java

```
public static void main(String[] args) {
    Person p1=new Person( age: 15, height: 165, id: 1);
    //p1和p6的排序元素相同，所以会自动去重，覆盖掉p1的id，不会添加新节点
    Person p6=new Person( age: 15, height: 165, id: 6);
    Person p2=new Person( age: 14, height: 170, id: 2);
    Person p3=new Person( age: 16, height: 168, id: 3);
    Person p4=new Person( age: 13, height: 165, id: 4);
    Person p5=new Person( age: 14, height: 172, id: 5);
```

效果如图：

```
Person
D:\software_download\JDK\bin\java.exe ...
Person_TreeSet的size为5
TreeSet是否为空：false
这是中序遍历-按自定义排序元素从小到大的顺序遍历：
Person4 Person2 Person5 Person6 Person3
中序遍历-遍历部分元素-到Person5终止遍历：
Person4 Person2 Person5
是否包含Person5：true
删除Person4之后的TreeSet：
Person2 Person5 Person6 Person3
清空TreeSet
TreeSet是否为空：true
TreeSet的size为0

Process finished with exit code 0
```

逻辑捉虫

课堂上写的node(element)方法是do...while结构

```
// 查找父节点
do {
    int cmp = element.compareTo(node.element);
    if (cmp > 0) {
        node = node.right;
    } else if (cmp < 0) {
        node = node.left;
    } else { // 相等
        // 找到了对应的节点，直接返回
        return node;
    }
} while (node != null);
```

实际运行测试时发现可能会报错，因为node非null的判断放在后面，就会先执行do的内容。然而第一行compareTo里面，如果node此时为null，则会因为node.element不存在而报错。

所以需要修改成while...do的形式，先判断node非null，再执行后续方法。

```
// 查找父节点
while (node != null)
{
    int cmp = element.compareTo(node.element);
    if (cmp > 0) {
        node = node.right;
    } else if (cmp < 0) {
        node = node.left;
    } else { // 相等
        // 找到了对应的节点，直接返回
        return node;
    }
}
```

实现Map数据结构

Map数据结构存储的内容是键值对（key-value），其中key可以为任意类型，而不仅仅是字符串。常见的具体类型有TreeMap、HashMap、WeakHashMap和Hashtable。Map的底层通过哈希表实现，哈希表的存储常采用**数组+链表+红黑树**，当链表长度大于8时，哈希表就会将链表转化为红黑树，提高查找效率。

根据所学的知识：动态数组、链表、栈与队列、二叉树、二叉搜索树，也可以实现Map数据结构的基础接口功能。设计思路如下：

用二叉搜索树的节点作为存储键值对的单元：

Map结构的所有接口均以二叉搜索树为基础，衍生改编而来。

Node<K,V>存储key，value，左右子节点和父节点：left,right,parent。

1. size（），isEmpty（），clear（）与二叉搜索树的基础方法相同
2. V put(K key, V value)：

   先检查输入key不能为空。如果根节点为空则先添加根节点，如果根节点不为空，则先查找key是否已经存在，已经存在则覆盖原来的value，若不存在，则创建并添加新的节点。返回put的value值。

3. V get(K key)：

   先声明private方法 Node<K,V> getNode（key）：如果key为null，输出报错信息；key不为null，从根节点开始查找（**通过二叉搜索树的性质，小的在左边，大的在右边，可以快速查找**），直到查找到该节点，返回该node，或者找不到该节点，返回null。

   V get(K key)调用getNode后，输出其值为node.value或者null。

4. V remove(K key)：

   先调用getNode（key）方法获取要删除的节点，找不到该节点则返回null，找得到则进行删除操作。

   删除操作与二叉搜索树的删除流程一样，先计算节点的度（0，1，2），度为0和1的删除与二叉搜索树相同，度为2的删除则要先找到前驱/后继节点，（代码中使用的是后继节点），使用后继节点的key，value值覆盖被删除节点的key，value值，再删除后继节点。

5. boolean containsKey(K key)：

   通过getNode（key）方法查找有无该节点存在，如果方法返回null则表示不存在该node，输出false，否则为true。

6. boolean containsValue(V value)：

   通过层序遍历LevelOrderTraversal，使用**队列**来获取每个节点的值value，一一匹配，有相等则返回true，没有则返回false。

7. void traversal(Visitor<K, V> visitor)：

   实现的功能是打印遍历每个节点的key-value键值对，代码逻辑为采用中序遍历，使打印根据key的从小到大排序。同理也可以改变traversal里访问左右子树、根节点的顺序，采用前序或者后序遍历打印每个节点的键值对。**还可以改return true的条件，使遍历到某个点后停止遍历（return key==n或者value==n）。**

综上所述，不使用红黑树，利用学过的**二叉搜索树与队列**结构，即线性表+BST，就能实现Map的基础接口功能。

此外还重写了toString（）方法，便于直接打印map，看其中存储的内容。

运行Main.java测试所有接口功能，控制台打印如下：演示中traversal按key遍历到等于10停止。

```
是否包含key：5=true
是否包含key：9249=false
是否包含value：13434=false
是否包含value：vvv=true
按照key中序遍历
key:2, value:aaa
key:4, value:vvv
key:5, value:gfgs
key:9, value:aaa
key:10, value:ddd
key:42, value:d72
key:77, value:gdfvgr
按照key中序遍历到key=10为止
key:2, value:aaa
key:4, value:vvv
key:5, value:gfgs
key:9, value:aaa
key:10, value:ddd
删除key为2的节点值aaa
删除key为42的节点值d72
map={key：4——value：vvv,key：5——value：gfgs,key：9——value：aaa,key：10——value：ddd,key：77——value：gdfvg}
清空map
clear后的map：{}
map的size是0
map是否为空true

Process finished with exit code 0
```

附加：额外用Python实现了自定义Map数据结构类和TreeSet类

不调用任何封装好的python库，从底层逻辑代码开始编写，思路与Java代码一致。

运行TreeSet.py和Map.py演示效果如下：

```
TreeSet的size是5
TreeSet是否为空False
traversal选择中序遍历TreeSet为：
4
5
8
9
12
TreeSet是否包含元素4:True
TreeSet是否包含元素999:False
删除节点4, 8之后,traversal中序遍历TreeSet为：
5
9
12
清空TreeSet
TreeSet的size是0
TreeSet是否为空True

Process finished with exit code 0
```

```
map的size是7
map是否为空False
key为5的value值是gfgs
key为1241431的value值是None
traversal选择中序遍历map为：
key:2  value:aaa
key:4  value:kkk
key:5  value:gfgs
key:9  value:aaa
key:10  value:ddd
key:42  value:d72
key:77  value:gdfvgr
是否包含key：5=True
是否包含key：9249=False
是否包含value：13434=False
是否包含value：kkk=True
删除key为2的节点值aaa
删除key为42的节点值d72
traversal选择中序遍历map为：
key:4  value:kkk
key:5  value:gfgs
key:9  value:aaa
key:10  value:ddd
key:77  value:gdfvgr
清空map
map的size是0
map是否为空True

Process finished with exit code 0
```

附录——TreeSet继承的BST方法实现思路

Node<E>存储element元素内容，左右子节点和父节点：left,right,parent。

1. int size()；boolean isEmpty(); void clear();

   直接调用private变量作判断赋值即可。

2. void add(E element);

   先检查输入element不能为null。如果根节点为空则先添加根节点，如果根节点不为空，则先查找存储了element的节点是否已经存在，如果已经存在，则用新值覆盖原来的值，若不存在，则创建并添加新的节点。size++并返回add的element值。

3. void remove(E element);

   先声明private方法Node<E> node(element)：如果element为null，输出报错信息；element不为null，从根节点开始查找（**通过二叉搜索树的性质，小的在左边，大的在右边，可以快速查找**），直到查找到该节点，返回该node，或者找不到该节点，返回null。

   在remove方法中先调用node方法获取要删除的节点，找不到该节点则返回null，找得到则进行删除操作。先计算节点的度（0，1，2），度为0和1的删除只需更改其父节点和其子节点之间的指向关系，度为2的删除则要先找到前驱/后继节点，（代码中使用的是前驱节点），用前驱节点的值覆盖被删除节点的值，再删除前驱节点。

4. boolean constains(E element);

调用node方法获取查找到的节点，为null则表示没找到该节点，不含该元素，返回false，否则返回true。

5. void traversal(Visitor<E> visitor);

   因为要求按照元素从小到大的顺序遍历，故这里选用**中序遍历**。还可以改visit函数return true的条件，使遍历到某个节点元素n后停止遍历（return element == n）。

参考博客

Visitor学习参考了该博客：

http://www.360doc.com/content/12/1226/12/8189294_256339069.shtml

Map架构学习参考了博客：

https://blog.csdn.net/eson_15/article/details/51150033

https://www.cnblogs.com/szrs/p/12164982.html

Python面向对象编程学习参考博客：

https://www.cnblogs.com/cudy/p/9269864.html

https://blog.csdn.net/weixin_42134789/article/details/80194788