二叉树的应用

任务 1：实现 BST 数据结构

二叉检索树即 BST，是利用二叉树的非线性关系，结合数据之间的大小关系进行存储的一种用于检索目的的数据结构。一般情况下，对信息进行检索时，需要指定检索关键码（Key），根据该关键码找到所需要的信息（比如学生信息里关键码是学号，而姓名等信息就是该关键码对应的信息），所以当提到检索时，都会有“键值对”这个概念，用(key,value)表示键值和对应信息的关系。

下面的二叉检索树 ADT 在描述时，使用了模板类的方法，并且使用了两个模板参数 K 和 V（这表明 key 和 value 可以是不同类型的数据）。在后续的测试文件中，让 key 和 value 都是 String 类型，所以对模板参数运用有困难的同学，可以直接设定 key 和 value 的类型为 String。

注意：任务 2 中 BST 的 key 和 value 存放的数据类型可能不同。

图 1 为 BST 接口的定义（仅作参考），表 1 为对接口函数的详细说明。

XJTUSE

```
public interface BST<K extends Comparable<K>, V> {
    public void insert(K key , V value);
    public V remove(K key);
    public V search(K key);
    public boolean update(K key , V value);
    public boolean isEmpty();
    public void clear();
    public void showStructure(PrintWriter pw) throws IOException;
    public void printInorder(PrintWriter pw) throws IOException;
}
```

图 1 BST 接口的定义

表 1 BST 接口的详细说明

方法名称	方法要求细节
public void insert(K key, V value)	Precondition: Binary Search Tree is not full. key and value are not null. Postcondition: Inserts a newElement(includes key and value) into a binary search tree.If an element with the same key already exists in the tree,then updates that element's value fields with the newElement's value field.
public V remove(K key)	Precondition: key is not null. Postcondition: Deletes the element with the same key from the binary search tree.If an element with the

方法名称	方法要求细节
	same key dosen't exsit in the tree, then returns null. Otherwise ,returns the element's value field.
public V search(K key)	Precondition: key is not null. Postcondition: Searches a binary search tree for the element with the same key. If this element is found,then returns the element's value field.Otherwise, returns null.
public boolean update(K key,V value)	Precondition: key and value are not null. Postcondition: Searches a binary search tree for the element with the same key. If this element is not found,then returns false.Otherwise, updates that element's value field with the argument value and returns true.
public boolean isEmpty()	Precondition: none Postcondition: If there is not any nodes in this binary search tree ,returns true. Otherwise, returns false.
public void clear()	Precondition: none Postcondition: Deletes all elements in the binary search tree.
public void showStructure(PrintWriter pw)	Precondition: PrintWriter is not null. Postcondition: Outputs the information about the binary search tree.The information includes the number of nodes and the height of the binary search tree. Outputed contents should be formatted as shown in Figure 2 (in the next page).
public void printInorder(PrintWriter pw)	Precondition: PrintWriter is not null. Postcondition: Outputs all the nodes in the binary search tree by asending order. An element is outputted in each line. The output of each element is in the following format:[key --- < value >]

XJTUSE

为了测试实现 BST 接口的数据结构是否运行正常，实验中提供两个文件，一个是 BST_testcases.txt，一个是 BST_result.txt 文件。其中，前一个包含了所有的对二叉检索树进行操作的命令内容，后一个则是执行命令文件之后的输出。具体的命令格式如表 2 所示。

表 2 命令的具体意义说明

命令符号	命令格式	命令举例
+	+(key , value)	+(auspices , "n.资助,赞助") 向二叉检索树中插入一个 key 为 auspices,value 为"n.资助,赞助"的元素，插入的要求遵守 BST 接口中的 insert 方法的定义
-	-(key)	-(morass) 从二叉检索树中删除一个 key 为 morass 的元素，删除的要求遵守 BST 接口中的 remove 方法的定义
?	?(key)	?(hide) 在二叉检索树中查找一个 key 为 hide 的元素，查找的要求遵守 BST 接口中的 search 方法的定义
=	=(key, value)	=(laggard , "laggard") 将二叉检索中对 key 为 laggard 的元素的 value 值更新为“laggard”，更新的要求遵守 BST 接口中的 update 方法的定义
#	#	# 调用二叉检索树的 showStructure 方法

VJTUSE

图 2 给出了一个样例，用来说明命令的执行和对应输出的内容格式，供参考。

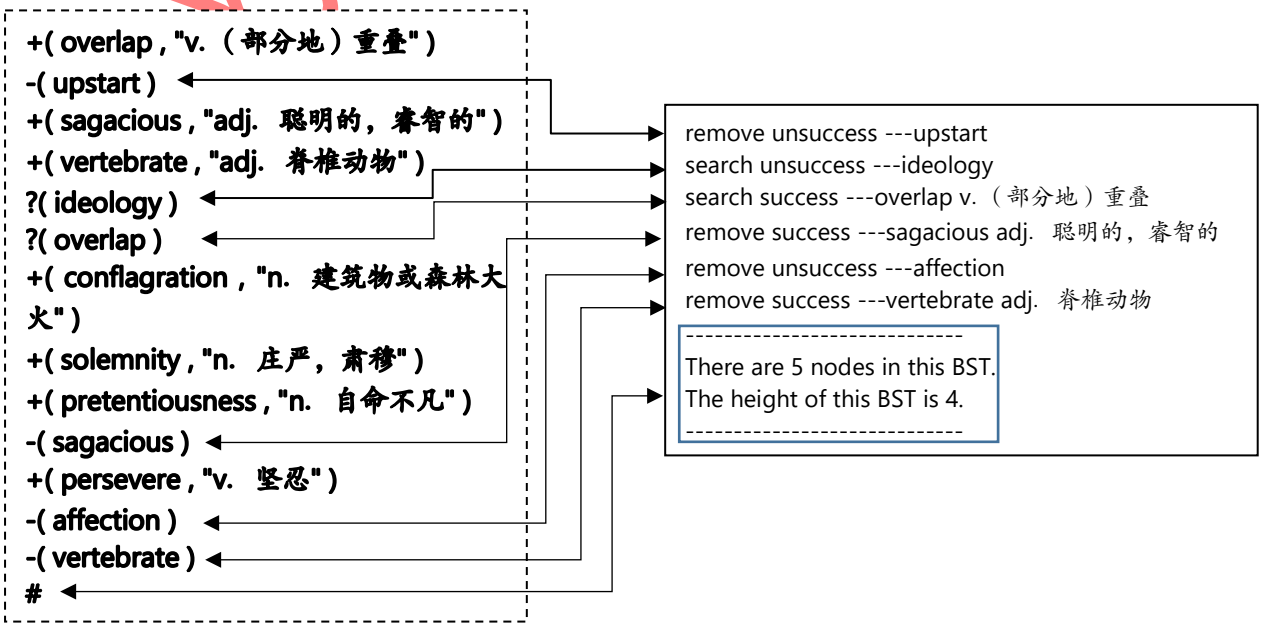

图 2 命令和输出格式对应关系

请大家要留意，本次输出的内容对每个命令的执行都有输出内容的要求（除了‘+’命令，因为该命令对应的 insert 方法没有返回值），而每个命令的输出格式很规律，结合每个命令对应的相应接口函数的返回值考虑即可。

图 2 样例中最后形成的二叉检索树的结点之间的关系如图 3 所示。

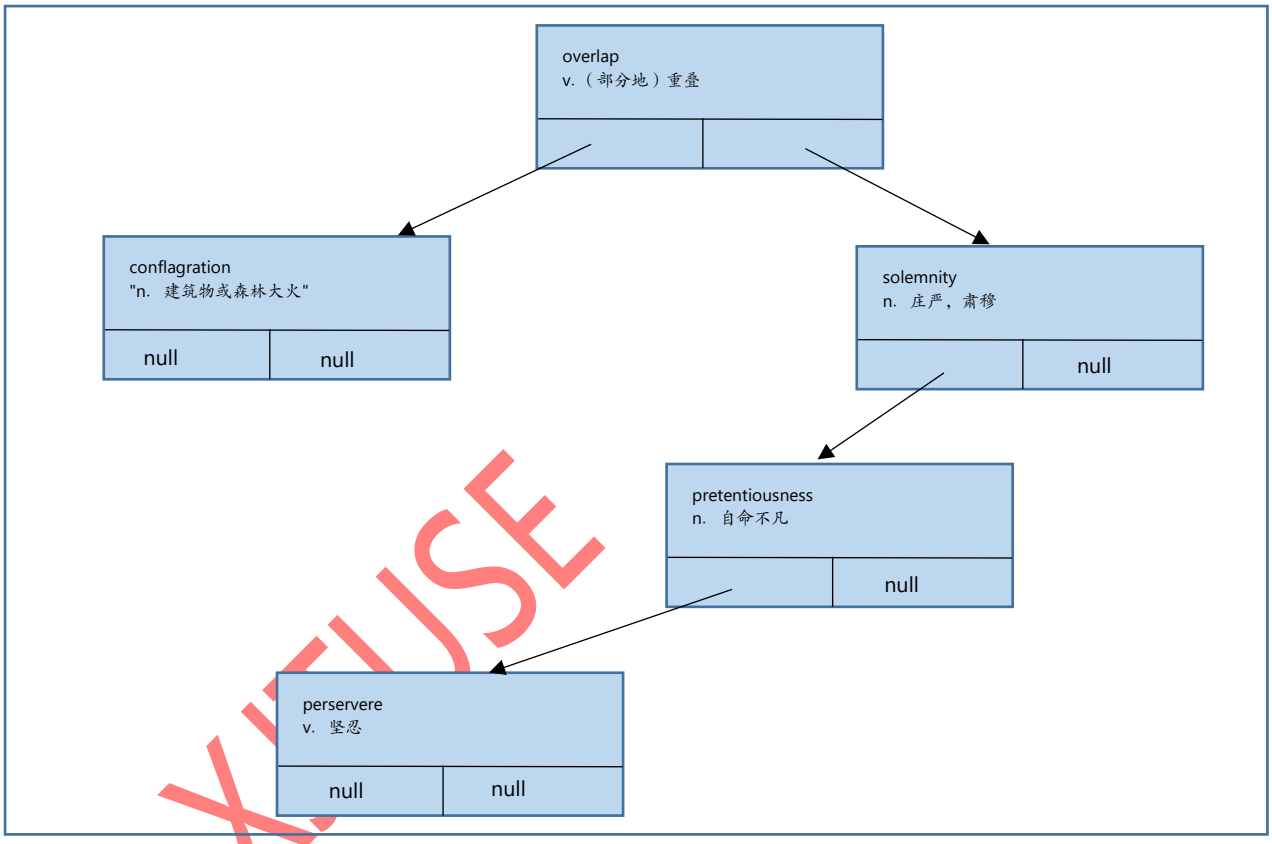

图 3 针对图 2 样例，执行所有命令之后生成的 BST

任务 2：使用 BST 为文稿建立单词索引表

在使用很多文字编辑软件时，都提供了对所编辑的文稿进行搜索的功能。当文稿内容的数据量比较大时，检索的速度就必须要进行考虑了。利用任务 1 中构建的 BST 数据结构，可以有效地解决这个问题。将文稿中出现的每个单词都插入到用 BST 构建的搜索表中，记录每个单词在文章中出现的行号。

例如，对于如下的文本段：（左侧列表为行号）。

```
1  Dudley's birthday -- how could he have forgotten? Harry got slowly
2  out of bed and started looking for socks. He found a pair under
3  his bed and, after pulling a spider off one of them, put them on.
4  Harry was used to spiders, because the cupboard under the stairs
5  was full of them, and that was where he slept.
6  When he was dressed he went down the hall into the kitchen. The
7  table was almost hidden beneath all Dudley's birthday presents. It
8  looked as though Dudley had gotten the new computer he wanted, not
9  to mention the second television and the racing bike. Exactly why
10 Dudley wanted a racing bike was a mystery to Harry, as Dudley was
11 very fat and hated exercise -- unless of course it involved
12 punching somebody. Dudley's favorite punching bag was Harry, but
13 he couldn't often catch him.
```

通过使用 BST，文稿中的单词被有效的存储到了 BST 中，当调用 BST 的 printInOrder 方法之后，输出的内容如下所示，类似于生成了一个索引表。

XJTUSE

实验材料里有一篇英文文稿，文件名为 article.txt[1]，请按照上面的样例根据该 article.txt 中的内容构建索引表，并将索引表的内容按照单词的字典序列输出到 index_result.txt 文件中。

在构建 BST 中的结点数据时，请认真思考记录行号的数据类型，如果可能，尽可能使用在之前实验中自己构建的数据结构，也是对之前数据结构使用的一种验证。

任务 3：学会使用堆

堆是一棵完全二叉树，是二叉树众多应用中一个最适合用数组方式存储的树形，所以必须要掌握。堆的主要用途是构建优先队列 ADT，除此之外，高效的从若干个数据中连续找到剩余元素中最小（或最大）都是堆的应用场景，比如构建 Huffman 树时，我们需要从候选频率中选择最小的两个；比如最短路径 Dijkstra 算法中要从最短路径估计值数组中选取当前最小值等，所有的这些应用都因为使用了堆而如虎添翼。下面从所列的几个方面对堆进行实践：

1）参看书中的代码，撰写一个具有 insert、delete、getMin 等方法的 Min_Heap

2）完成一个使用 1）编写的 Min_Heap 的排序算法；

3）上课讲解的堆是二叉树的一个应用，因此也称为二叉堆。如果将堆的概念扩展到三叉树

[1] 英文文稿的内容为海明威的《In Our Times》，该内容我们没有版权，请同学们不要在网络上传播，该文稿仅限于教学使用。

（树中每个结点的子结点数最多是 3 个），此时将形成三叉堆。除了树形和二叉堆不一致外，堆序的要求是完全一致的。使用完全三叉树形成堆，在 n 个结点下显然会降低整棵树的高度，但是在维持某个结点的堆序时需要比较的次数会增加（这个时候会是 3 个子结点之间进行互相比较求最大或最小），这也是一个权衡问题。现在要求将 1）中实现的二叉堆改造成三叉堆，并通过 2）中实现的排序验证这个三叉堆是否正确。

XJTUSE