用树解决问题

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数据的数据结构。一般情况下，对信息进行检索时，都需要指定检索关键码（Key），根据该关键字找到所需要的信息（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。

下面的二叉检索树的 ADT 在描述时，依然使用了模板类的方法，并且在这次描述中，使用了两个模板参数 K 和 V（这表明 key 和 value 可以为不同类型的数据）。但是在我们后续的测试文件中，其实 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：后面任务中的 BST 中的 key 和 value 存放的数据类型可能不同，请大家注意。

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。


```
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:

方法名称	方法要求细节
	key is not null. Postcondition: Deletes the element with the same key from the binary search tree.If an element with the 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 >]

为了测试实现BST接口的数据结构是否运行正常，准备了两个文件，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

为了让大家能够更好的理解二叉检索树的命令执行和输出的内容格式，现在通过一个具体的样例进行说明，如图2所示。

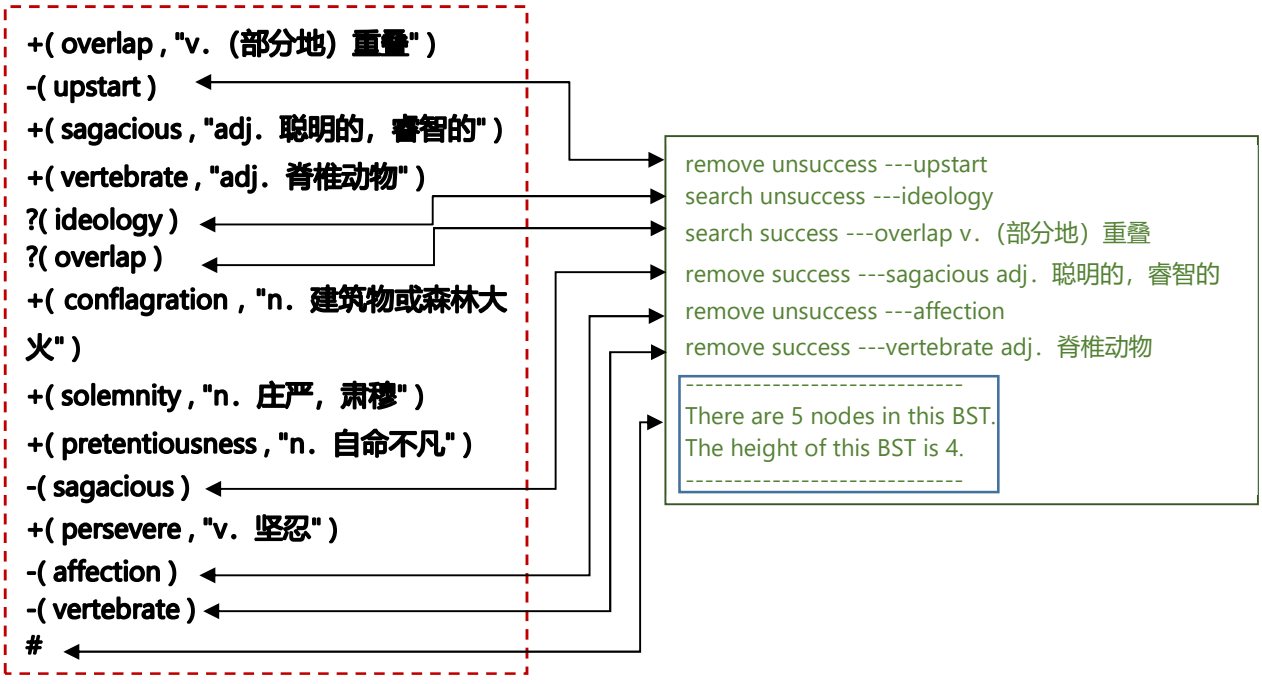
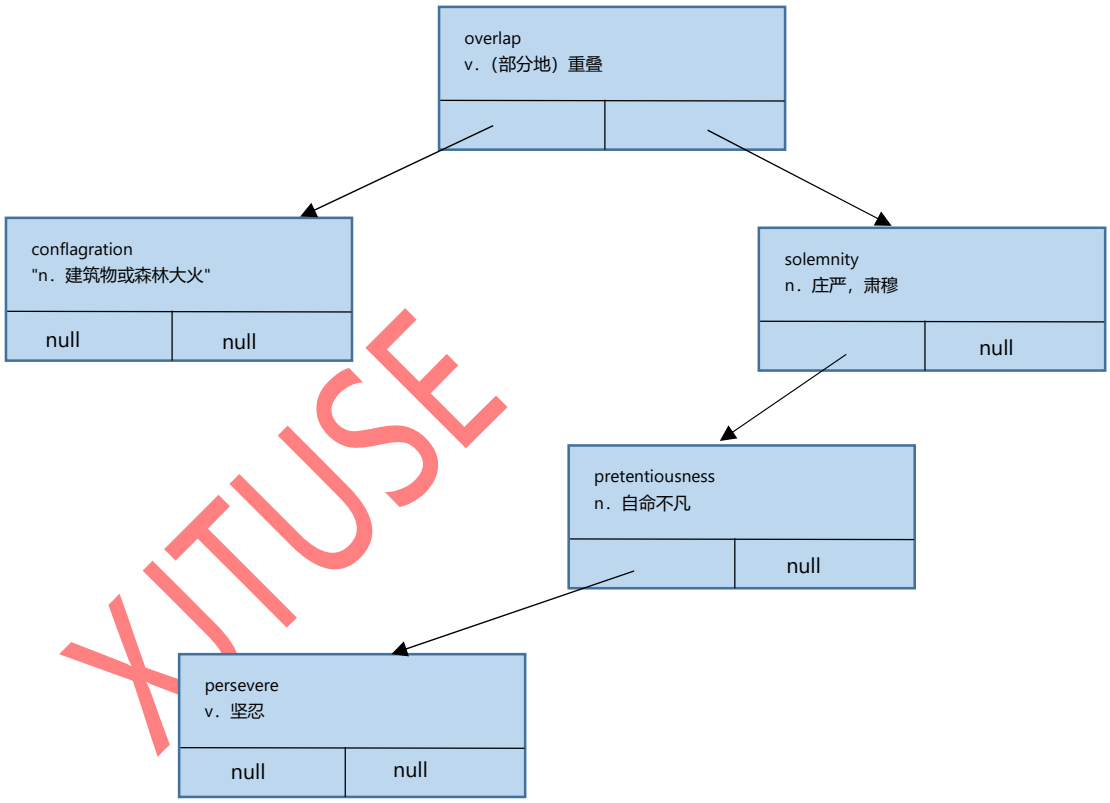


图2 命令和输出格式对应关系

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。上面样例中最后形成的二叉检索树的节点之间的关系如下：



任务 2：使用 BST 为文稿建立单词索引表

我们在使用很多文字编辑软件时，都有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，可以比较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.
```

通过我们构建的 BST，当调用了 BST 的 printInOrder 方法之后，输出的内容如下：

我为大家准备了一篇英文文稿，文件名为 article.txt[1]，请大家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。

注：单词可以不用像上面样例那样区分大小写，另外单词长度小于等于 2 的也可以忽略，单词中包含短横线、单引号、双引号都可以忽略。这就要求同学们在真正将单词插入到 BST 中时要先做“数据清洗”，数据清洗的规则制定自主权交给你们，在实验报告中要有规则的说明。

任务 3：实现 Trie

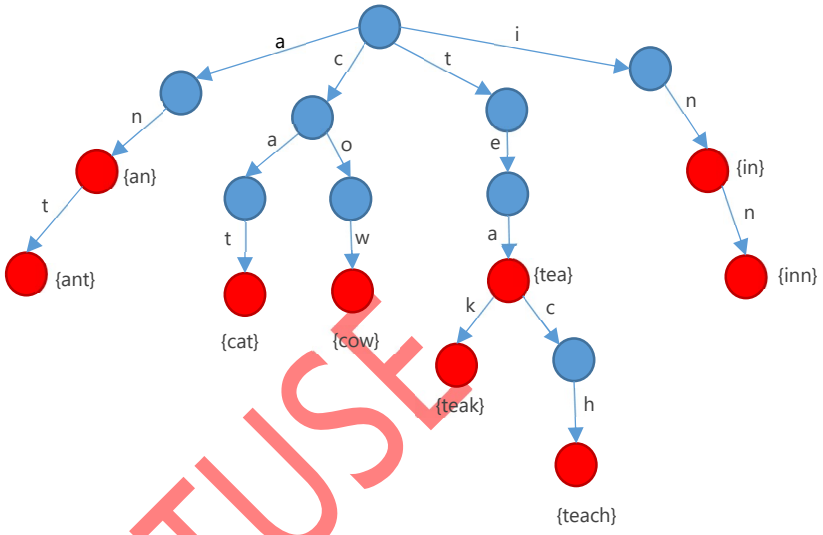
Trie，又称前缀树或字典树，是一种有序树，用于保存关联数组，其中键通常是字符串。与二叉查找树不同，键不是保存在结点中，而是由结点在树中的位置决定。一个结点的所有子孙都有相同的前缀，也就是这个节点对应的字符串，而根结点对应空字符串。一般情况下，不是所有的结点都有对应的值，只有叶子结点和部分内部结点所对应的键才有相关的值。

Trie 这个术语来自于 retrieval。根据词源学，trie 的发明者 Edward Fredkin 把它读作与‘tree’相

[1] 英文文稿的内容为福尔摩斯探案集的一部分，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。

同的发音。但是其他作者把它读作与‘try’相同的发音。

如果我们的 Trie 中只存储英文字母所构成的单词的话，那么假如有如下 9 个关键字：an、ant、cat、cow、tea、teak、teach、in、inn，则这些关键字构成的 trie 树的逻辑关系示意图如下：



从图中可看到红色结点代表了关键字，也就是存储关键字不是每个结点都存储（有别于二叉查找树），但也不是所有的关键字都只存储在叶节点（有别于 Huffman 树），关键字既可存储在内部结点，也可存储在叶结点。同时也发现从根节点开始到树中任意一个关键字结点的路径上经过的所有字符串都是该关键字的前缀串，比如关键字 teach，那么它的前缀串有：t、te、tea、teac。这也正是 Trie 树称其为前缀树的原因。利用 Trie 树的这个特点，可以实现自动完成功能，大家在 IDE 中撰写代码时，当敲入某些字符时，IDE 都会给出以这些字符开头的可用的变量、函数、类类型等内容，这正是 Trie 树给我们带来的能力。关于 Trie 树的具体介绍可以参看教材 中“高级树结构”部分，另外也可以在网络上找到很多关于 Trie 树介绍的资源。

我们提供一个“dictionary.txt”数据文件，该文件中记录了 8 万多个英文单词。请同学们完成如下子任务：

1. 设计 Trie 树的 ADT；（该 ADT 应至少具备如下功能：插入一个关键字、删除一个关键字、获得以某个字符串作为前缀串的所有关键字）。

2. 根据子任务 1 中的 ADT，实现该 ADT 的一个 DictionaryTrie 数据结构，该数据结构中的关键字即为“dictionary.txt”中的每一个单词。

3. 设计至少 5 个测试，用来验证子任务 2 中的数据结构是否正常。

任务 4：使用 Trie 实现 T9 键盘

T9 键盘又称 9 键文字输入法或 9 键输入法，是一种含有 3×3 数字键盘的功能手机上的所使用的预测性文本技术（Predictive Text），如下图所示就是我们在手机上常见的 T9 键盘模式：

T9 键盘上从 2-9 的 8 个数字键，每个键都代表 3~4 个字符，数字键 1 有时代表一些特殊字符的输入，有时代表空格键，我们重点关注 2-9 这 8 个数字键，这 8 个数字键映射的字符如上图所示。因为多个字符映射到一个单独的数字键，当我们敲击一连串的数字键时，可能会产生多个候选单词，假如敲击“2665”，那么可能代表输入“book”，也可能代表输入“cool”，甚至于还有可能有更多的候选单词。

为了能够将数字键序列翻译成候选的单词，Trie 数据结构完全可以胜任。在任务 3 中我们实现了一种 Trie 数据结构，现在需要同学们根据当前的任务重新实现一种适配于 T9 键盘输入的 Trie 数据结构，该数据结构的数据基础依然是“dictionary.txt”。

当实现了 T9 键盘之后，同学们可以按照如下命令行的交互方式验证自己的 T9 键盘：

```
Enter “exit” to quit.
Enter Key Sequence (or “#” for next word):
>76257
‘rocks’
Enter Key Sequence (or “#” for next word):
>#
‘socks’
Enter Key Sequence (or “#” for next word):
>53556
‘jello’
Enter Key Sequence (or “#” for next word):
>#
There are no more T9 words.
Enter Key Sequence (or “#” for next word):
 >76257#
‘socks’
Enter Key Sequence (or “#” for next word):
>76257##
There are no more T9 words.
>4423
Not found in current dictionary.
>exit
```