

目录

任务 1: 实现 BST 数据结构	1
1.1 题目及任务	1
1.2 题目分析	3
1.2.1 BST 树数据结构设计	3
1.2.2 基本方法编写	3
1.2.3 输入输出部分	4
1.3 主干代码说明	4
1.3.1 BST 基本接口	4
1.3.2 输入输出辅助	7
1.4 运行结果展示	8
1.5 总结和收获	8
任务 2: 使用 BST 为文稿建立	
单词索引表	9
2.1 题目及任务	9
2.2 题目分析	9
2.2.1 数据设计	9
2.2.2 数据清洗	10
2.2.2 方法调整	10
2.3 主干代码说明	10
2.3.1 数据清洗	10
2.3.2 中序遍历	11
2.4 运行结果展示	12
2.5 总结和收获	12
任务 3: 实现 Trie	13
3.1 题目及任务	13
3.2 题目分析	14
3.2.1 TrieADT	14
3.2.2 DictionaryTrie	14
3.2.3 测试方法	15
3.3 主干代码说明	16

3.3.1 insert	16
3.3.2 delete	16
3.3.1 PrefixAs	17
3.4 运行结果展示	19
3.5 总结和收获	19
任务 4: 使用 Trie 实现 T9 键盘	20
4.1 题目及任务	20
4.2 题目分析	21
4.2.1 数据设计	21
4.2.2 T9Find 函数	21
4.2.3 T9Shell	22
4.3 主干代码说明	22
4.3.1 T9Find	22
4.3.2 T9Shell	22
4.4 运行结果展示	24
4.5 总结和收获	25
附录: 源代码汇总	26
任务一	26
BSTNode.java	26
testBST.java	30
任务二	33
wordBST.java	33
任务三	35
DictionaryTrie.java	35
testTrie.java	39
任务四	41
T9Trie.java	41

任务 1：实现 BST 数据结构

1.1 题目及任务

下面的二叉检索树的 ADT 在描述时，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）。下面给出 BST 接口的定义和详细说明（仅作参考）。

```
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;  
}
```

- **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 same key doesn't exist 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. Outputted 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 ascending 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。具体的命令格式如下所示

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

1.2 题目分析

本题目很简单, 即依要求实现 BST 树, 并按照其要求编写代码通过测试。

在 BSTNode.java 文件中依要求编写 BSTNode 类, 在 testBST.java 文件中编写输入输出相关代码

1.2.1 BST 树数据结构设计

首先需要简单设计一下 BST 结点:

- K key. BST 结点的唯一标识符, 结点也是用这个互相比较。
- V value. BST 结点的值
- BSTNode left, right. BST 结点的两个子树
- BSTNode parent. BST 结点的父树

构造方法略

1.2.2 基本方法编写

依据课堂和教科书内容编写即可, 为了增强代码可读性, 我放弃了 method_helper 的写法, 这需要加一些额外的条件判断, 并且以自己的 key 是否为 null 来判断。

1. void insert(K k, V v) - 和当前结点比较后, 有对应子树则递归, 无则新建一个结点作为子树。

2. `BSTNode searchNode(K k)` - 这是一个辅助函数，返回 `key=k` 的结点
3. `V remove(K k)` - 找到对应的结点后：
 - 若无子树：
 - 若只有一个子树：用子树代替自己即可
 - 若有两个子树，则选取右子树的最小结点，并和自己交换 `key` 和 `value`，再在右子树中删除自己。
4. `V search(K k)` - 找到对应结点并返回其 `value`
5. `V update(K k, V v)` - 找到对应结点并更新
6. `void clear()` - 如果有父节点通过父节点删除，否则直接将自己的 `key` 设为 `null` 即可
7. `Boolean isEmpty()` - 如果自己的 `key` 为 `null` 则为空
8. `int getCons()` - 递归计算节点数
9. `int getHeight()` - 递归计算树的高度
10. `void showStructure()` - 依照题目要求打印信息

1.2.3 输入输出部分

本部分不是实验重点，故简要描述。

根据样例文件不难确定，测试用的 `testBST` 类的 `root` 及结点应该采用 `<String,String>` 的方式定义，输入时通过 `Scanner` 变量逐行读取，并以空格为间隔字符分割字符串，对分割的结果依照不同命令的格式读取并执行。

题目要求从 `BST_testcases.txt` 文件输入，为了方便和标准结果 `BST_result.txt` 对比，将结果输出到 `BST_my_res.txt`，在 `testBST` 类中用 `java` 的 `File` 和流相关函数编写一个比较文件差异的函数。

1.3 主干代码说明

1.3.1 *BST* 基本接口

Listing 1: `BSTNode`

```

1 public void insert(K k, V v) {
2     if (this.value == null && this.key == null) {

```

```
3         this.key = k;
4         this.value = v;
5     } else if (this.key.compareTo(k) == 0) this.value = v;
6     else if (this.key.compareTo(k) < 0) {
7         if (this.right == null) this.right = new BSTNode<K, V>(k, v, this);
8         else this.right.insert(k, v);
9     } else if (this.key.compareTo(k) > 0){
10        if (this.left == null) this.left = new BSTNode<K, V>(k, v, this);
11        else this.left.insert(k, v);
12    }
13 }
14
15 public V remove(K k) {
16     BSTNode<K, V> n = this.searchNode(k);
17     if (n == null) return null;
18     V v = n.value;
19     if (n.left != null && n.right != null) {
20         // get the smallest Node of right subtree.
21         BSTNode<K, V> tmp = n.right;
22         while (tmp.left != null) tmp = tmp.left;
23         n.key = tmp.key;
24         n.value = tmp.value;
25         tmp.key = k;
26         tmp.value = v;
27         n.right.remove(k);
28     } else if (n.left != null) {
29         if (n.parent == null) {
30             n.key = n.left.key;
31             n.value = n.left.value;
32             n.left = null;
33         } else {
34             n.left.parent = n.parent;
35             if (n.parent.left == n) n.parent.left = n.left;
36             else n.parent.right = n.left;
37         }
38     } else if (n.right != null) {
```

```
39         if (n.parent == null) {
40             n.key = n.right.key;
41             n.value = n.right.value;
42             n.right = null;
43         } else {
44             n.right.parent = n.parent;
45             if (n.parent.left == n) n.parent.left = n.right;
46             else n.parent.right = n.right;
47         }
48     } else {
49         if (n.parent.left == n) n.parent.left = null;
50         else n.parent.right = null;
51     }
52     return v;
53 }
54
55
56 public V search(K k) {
57     if (this.searchNode(k) != null) return this.searchNode(k).value;
58     return null;
59 }
60
61 public BSTNode<K, V> searchNode(K k) {
62     if (this.key == null) return null;
63     else if (k.compareTo(this.key) == 0) return this;
64     else if (k.compareTo(this.key) > 0 && this.right != null) return this.
        right.searchNode(k);
65     else if (k.compareTo(this.key) < 0 && this.left != null) return this.left
        .searchNode(k);
66     else return null;
67 }
68
69 public boolean update(K k, V newV) {
70     BSTNode<K, V> tmp = searchNode(k);
71     if (tmp == null) return false;
72     tmp.value = newV;
```



```
73     return true;
74 }
75
76 public boolean isEmpty() {
77     return this.key == null;
78 }
79
80 public void clear() {
81     // considering java has perfect gc, just assign objects as null
82     this.left = null;
83     this.right = null;
84     this.remove(this.key);
85 }
86
87 public int getHeight() {
88     if (this.key == null) return 0;
89     if (this.left == null && this.right == null) return 1;
90     else if (this.left == null) return this.right.getHeight() + 1;
91     else if (this.right == null) return this.left.getHeight() + 1;
92     else return Math.max(this.left.getHeight(), this.right.getHeight()) +
93         1;
94 }
95
96 public int getNodeCons() {
97     if (this.left == null && this.right == null) return 1;
98     else if (this.left == null) return this.right.getNodeCons() + 1;
99     else if (this.right == null) return this.left.getNodeCons() + 1;
100    else return this.left.getNodeCons() + this.right.getNodeCons() + 1;
101 }
```

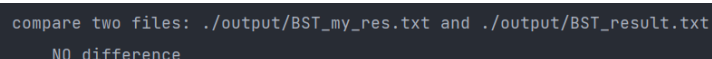
1.3.2 输入输出辅助

Listing 2: CompareFile

```
1 public static void compareTwoFile(String oldFile, String newFile) throws
   IOException {
2     List<String> list1 = Files.readAllLines(Paths.get(oldFile));
```

```
3 List<String> list2 = Files.readAllLines(Paths.get(newFile));
4 System.out.printf("compare two files: %s and %s \n", oldFile, newFile);
5 List<String> finalList = list2.stream().filter(line ->
6     list1.stream().noneMatch(line2 -> line2.equals(line))
7 ).toList();
8 if (finalList.size() == 0) {
9     System.out.println("\tNO difference");
10 }else{
11     System.out.println("diff:");
12     finalList.forEach(System.out::println);
13 }
14 }
```

1.4 运行结果展示



```
compare two files: ./output/BST_my_res.txt and ./output/BST_result.txt
NO difference
```

1.5 总结和收获

- 本次实验中，不使用 helper 让我对 BST 和 java 语言有了更深入的理解，对输入命令不同的解析方法考验了我对代码的整洁要求，自己编写一个比较文件的函数让我认识了 Java 对于文件的强大支持和流的巧妙。
- 着手前，在编写输入命令的解析时应该先在输入文件中仔细观察输入命令而不是只通过 pdf 的图片，这样才能找到唯一的最简单解析方法。
- 因为第一次尝试不使用 helper 函数的写法，出现了较多 bug，所幸经过不断的调试都修复了。每次调用实例函数都需要思考该实例是否为 null，同时出现大面积 bug 时不要急于修复，看看代码是否会有另一种同样正确的写法比如我就因为在删除结点的方法中一开始用的是左子树最大值而和标准输出的高度在后半部分总是差一点，第一次出现 bug 的时候已经有一百多个结点了，调了几个小时后才想起还可以使用右子树最小值。

任务 2：使用 BST 为文稿建立 单词索引表

2.1 题目及任务

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。

[Dudley ---- < 8 10 10 >]	[gotten ---- < 8 >]	[somebody ---- < 12 >]
[Dudley's ---- < 1 7 12 >]	[had ---- < 8 >]	[spider ---- < 3 >]
[Exactly ---- < 9 >]	[hall ---- < 6 >]	[spiders ---- < 4 >]
[Harry ---- < 1 4 10 12 >]	[hated ---- < 11 >]	[stairs ---- < 4 >]
[He ---- < 2 >]	[have ---- < 1 >]	[started ---- < 2 >]
[It ---- < 7 >]	[he ---- < 1 5 6 6 8 13 >]	[table ---- < 7 >]
[The ---- < 6 >]	[hidden ---- < 7 >]	[television ---- < 9 >]
[When ---- < 6 >]	[him ---- < 13 >]	[that ---- < 5 >]
[a ---- < 2 3 10 10 >]	[his ---- < 3 >]	[the ---- < 4 4 6 6 8 9 9 >]

请大家按照样例和任务 1 中描述过的打印函数根据该 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。

2.2 题目分析

基于已经实现好的模板类，新建 wordBST.java 作为任务二的主体

2.2.1 数据设计

root - 根结点，根据题目要求定义为 BSTNode< String, Vector<Integer> >

String 为数据清洗中得到的单词，Vector<Integer> 代表其所在的行号，所以还需要在读入文件时记录行号。

2.2.2 数据清洗

根据题目要求, 我决定采用 Java 自带的 StringTokenizer 类, 该类可以用来对给定的字符串以给定的符号逐个分割并读出, 故在 Scanner 变量每读入新的一行后, 就以构造方法的形式输入到新建的 StringTokenizer 变量中。经过反复实验后, 我采用了 `[,;!?"-');&*(` (和空格作为表示分割的字符)。

根据题目要求, 需要从 txt 文件中提取出长度小于等于 2 的单词, 我在尝试打印所有符合条件的单词后也发现该类型一般为极其常用或无意义的字母组合, 故在每次字符串分割出后根据其长度再清洗一次。反复实验后, 我发现大小写的区分在绝大多数情况下是影响统计结果的易读性的, 所以我使每次字符串分割后都调用 `toLowerCase()`。

实验中我注意到这种清洗后还会有一些数字存留下来, 在尝试过 String 自带的 `remove` 等方法后, 我还是认为 String 自带的 `match` 方法更好用且可以使用正则表达式, 用正则表达式 `[0-9]+[a-z]*` 可以直接剔除无意义的数字和序数词。

2.2.2 方法调整

本题目只需要用到任务 1 中定义好的 `insert` 和 `printInOrder` 方法

- `insert` - 依照之前的定义, 只要在测试方法中插入前先调用 `search` 方法寻找是否已经插入即可若已经插入, 则对 `search` 所得的 Vector 调用 `add` 方法插入新行号; 若未找到则插入新的 String 和只含有当前行号的 Vector
- `printInorder` - 依照题目格式要求打印。依照二叉树的性质采用中序遍历。在打印 Vector 时直接调用 `toString` 函数并取去掉第一个和最后一个字符的子串并去除 `','` 即可。

2.3 主干代码说明

2.3.1 数据清洗

Listing 3: ReadIn

```
1 private void ReadIn(String filename) throws FileNotFoundException {
2     Scanner sc = new Scanner(new FileReader(filename));
3     int linenums = 0;
4     String tmp;
5     Vector<Integer> v;
6     Vector<String> res = new Vector<>();
7     while (sc.hasNextLine()) {
```

```
8      ++linenums;
9      tmp = sc.nextLine();
10     StringTokenizer st = new StringTokenizer(tmp, "[, . ! ? \n \n - ' ) ; & * ( "
        );
11     while (st.hasMoreTokens()) {
12         tmp = st.nextToken().toLowerCase();
13         if (tmp.length() <= 2 || tmp.matches("[0-9]+[a-z]*")) continue;
14         if ((v = root.search(tmp)) != null) v.add(linenums);
15         else {
16             v = new Vector<>();
17             root.insert(tmp, v);
18             v.add(linenums);
19         }
20     }
21 }
22 }
```

2.3.2 中序遍历

Listing 4: printInorder

```
1 private void printHelper(BSTNode<String, Vector<Integer>> b) {
2     if (b == null) return;
3     printHelper(b.left);
4     System.out.println "[" + b.key + " --- < " + b.value.toString().
        substring(1, b.value.toString().length() - 1).replace(",", "") + " > ]"
        ;
5     printHelper(b.right);
6 }
7
8 public void printInorder(){
9     printHelper(this.root);
10 }
```

2.4 运行结果展示

运行结果过长，附于实验报告上传。

2.5 总结和收获

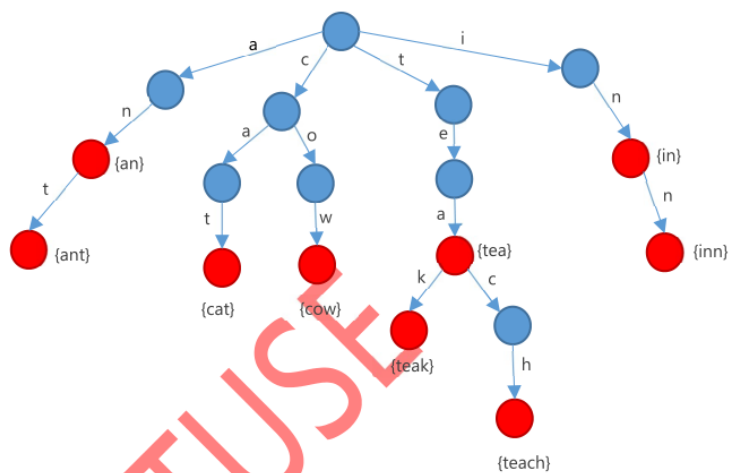
经历过第一次实验中大量的 debug 和大量的样例测试，本次实验较为简单，在数据清洗的过程中学习到 `StringTokenizer` 类的基本方法，愈发感受到 java 对字符串处理的支持之强，也让我复习了一下正则表达式的用法

任务 3：实现 Trie

3.1 题目及任务

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。

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：



从图中可以看出红色节点，即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。

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。

3.2 题目分析

用 DictionaryTrie.java 文件存储 Trie 类，testTrie.java 文件存放测试方法

3.2.1 TrieADT

因为字典树的键值由位置决定，所以每个结点只需要存储一个字符即可，接口如下：

- public void insert(String keyword) - 插入关键字 keyword
- public void delete(String keyword) - 删除关键字 keyword，若未找到对应关键词打印错误值；
- public Vector<String> PrefixAs(String prefix) - 获得以 prefix 作为前缀串的所有关键字并以 Vector<String> 的形式输出
- public DictionaryTrie match(String target) - 从当前节点开始匹配对应字符串，返回匹配到的最后一个结点，若未完整匹配到，返回 null
- public Vector<String> printSons() - 打印出从当前节点开始的所有关键字
- public String printDad() - 打印出从根节点到当前结点的字符串
- public Vector<String> printDad_Sons() - 打印出当前结点的所有子树的完整关键字
- public void ReadIn(String Filename) - 从 Filename 读入并插入到 this

3.2.2 DictionaryTrie

数据设计

- Trie dad - 父结点
- Vector<Trie> sons - 子结点集合
- char ch - 当前结点代表的字符

- `int dup_times` - 当前结点作为关键词重复的次数, 可用于统计词频; 大于等于 1 即可视为关键词有效

接口实现

构造方法较简单, 略。

- `insert` - 对于给定的字符串, 从根节点开始对其子结点依次匹配
 - 若未匹配到, 则用当前字符串的首字符新建一个子结点, 并对该子结点递归依次插入字符串剩余的字符
 - 若匹配到, 则对这个子结点调用函数递归插入剩余的字符串
- `match` - 辅助函数。对于给定的字符串, 在当前结点的子节点中匹配首字符, 若匹配到则递归调用匹配子字符串直到字符串长度为 1 并返回该结点, 否则返回 `null`
- `delete` - 对于一个字符串, 调用 `match` 方法匹配, 若返回 `null` 打印错误信息, 否则将该结点的 `dup_times` 置为 0 即可 (考虑到作为一个字典树可能会有大量插入删除, 为了减少内存分配的速度, 保留结点)
- `printSons` - 辅助函数。定义一个 `Vector<String> res`, 对每个子节点, 若关键字有效则将其字符加入 `res`, 随后用子节点调用, 最后将调用的返回值利用 `replaceAll` 函数在其前加上当前子节点的字符作为前缀, 并整体加入 `res`, 递归实现。
- `PrefixAs` - 利用 `match` 找到对应的结点, 打印并将返回值利用 `replaceAll` 加上前缀即可, 如果当前结点关键词有效, 也要加入当前结点。

3.2.3 测试方法

测试方法中的 `root` 用 `'\0'` 初始化

五个测试方法思路如下:

- `InsertNumTest` - 从文件读入后和文件行数对比, 测试插入是否成功
- `aarBasicTest` - 打印所有以 `aar` 开头的单词
- `aarDeleteTest` - 删除 `aargh` 后用 `PrefixAs` 方法打印所有以 `aar` 开头的单词, 和输入文件对比;
- `aarMultiInsertTest` - 多次插入同一个值后用 `PrefixAs` 方法打印所有以 `aar` 开头的单词, 和输入文件对比;
- `zyzMultiTest` - 中间三条测试统一在 `zyz` 前缀上再测试一次

3.3 主干代码说明

3.3.1 insert

Listing 5: void insert(String keyword)

```
1 // insert a "keyword" to this trie.
2 public void insert(String keyword) {
3     // search for same prefix string.
4     for (DictionaryTrie son : this.sons) {
5         if (son.ch == keyword.charAt(0)) {
6             if (keyword.length() == 1) son.dup_times++;
7             else son.insert(keyword.substring(1));
8             return;
9         }
10    }
11    // if this has no son which have same prefix as keyword.
12    DictionaryTrie tmp = new DictionaryTrie(this, keyword.charAt(0));
13    if (keyword.length() != 1) tmp.insert(keyword.substring(1));
14    else tmp.dup_times++;
15 }
```

3.3.2 delete

Listing 6: DictionaryTrie match(String target)

```
1 /**
2  * begin with "this", iterate to get the bottom Trie whose prefix match the
3  * given target.
4  * if found, return Trie; else return null
5  */
6 public DictionaryTrie match(String target) {
7     if (this.sons == null) return null;
8     for (DictionaryTrie son : this.sons) {
9         if (son.ch == target.charAt(0)) {
10             if (target.length() == 1) return son;
11             else return son.match(target.substring(1));
12         }
13     }
14 }
```

```

12     }
13     return null;
14 }

```

Listing 7: void delete(String keyword)

```

1  /**
2  * delete a "keyword" to this Trie.
3  * if found no matter it is keyword
4  * if not found, printSons ERROR message on the terminal.
5  */
6  public void delete(String keyword) {
7      DictionaryTrie res = this.match(keyword);
8      if (res == null) System.out.println("ERROR--[Trie.delete]: no such
          keyword as \"" + keyword + "\"");
9      else res.dup_times = 0;
10 }

```

3.3.1 PrefixAs

Listing 8: Vector<String> printSons()

```

1  /**
2  return all valid keyword begin with this Trie use DFS.
3  */
4  public Vector<String> printSons() {
5      Vector<String> res = new Vector<>();
6      if (this.sons == null) return null;
7      for (DictionaryTrie son : this.sons) {
8          if (son.dup_times > 0) res.add(String.valueOf(son.ch));
9          Vector<String> tmp = son.printSons();
10         if (tmp != null) tmp.replaceAll(s -> son.ch + s);
11         res.addAll(tmp);
12     }
13     return res;
14 }

```

Listing 9: Vector<String> PrefixAs(String prefix)

```
1 public Vector<String> PrefixAs(String prefix) {  
2     DictionaryTrie res = this.match(prefix);  
3     if (res == null) {  
4         System.out.println("ERROR--[Trie.PrefixAs]: no such keyword to  
5             match as \"" + prefix + "\"");  
6         return null;  
7     }  
8     Vector<String> sen = res.printSons();  
9     sen.replaceAll(s -> prefix + s);  
10    if (res.dup_times > 0) sen.add(prefix);  
11    sen.sort(String::compareTo);  
12    return sen;  
13 }
```

3.4 运行结果展示

```
---- InsertNumTest begins ----
expect:
    80368
run:
    80368

---- aarBasicTest begins ----
expect:
    5   aardvark   aardwolf   aargh   aarrgh   aarrghh
run:
    5   aardvark   aardwolf   aargh   aarrgh   aarrghh

---- aarDeleteTest begins ----
expect:
    4   aardvark   aardwolf   aarrgh   aarrghh
run:
    4   aardvark   aardwolf   aarrgh   aarrghh

---- aarMultiInsertTest begins ----
expect:
    4   aardvark   aardwolf   aarrgh   aarrghh
run:
    4   aardvark   aardwolf   aarrgh   aarrghh

---- aarMultiInsertTest begins ----
expect:
    1   zyzzyvas
run:
    1   zyzzyvas

Process finished with exit code 0
```

3.5 总结和收获

- 本次实验的主题字典树恰巧也是我寒假学习一门数据库公开课时面对的第一个 project，先后用不同的语言以不同的思路实现同一种数据结构真是一种奇妙的体验；
- 在 Trie 的编写过程中我采用了大量递归的写法，递归的思想一次又一次地让我感受到递归思想的自然、巧妙和深奥之所在；
- 第三个任务中，设计测试的思想用到了上学期在“软件质量保证”课程中学到的内容，计算机的各个部分是紧密连接的，这也正是我喜欢这个学科的原因。

任务 4：使用 Trie 实现 T9 键盘

4.1 题目及任务

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
```

4.2 题目分析

在 T9Trie.java 这个文件里实现该任务相关

本次任务的主体可以看作两部分，实现一个输入数字返回一个包含对应查询结果的 Vector 和模拟一个交互的 shell

4.2.1 数据设计

- root - 根节点
- KeyValues - 一个字符串数组以存储每个按键对应的字符串，实现其映射关系

4.2.2 T9Find 函数

该函数的目的是输入一个字典树和全为数字的字符串，依照 T9 输入法以 Vector<String> 的形式输出匹配的字符串结果

由题目要求不难发现这里适合层序遍历字典树并依照 T9 输入法不断剪枝

1. 首先创建一个 Vector<DictionaryTrie> 变量 res，对给定字符串的首字符，转换为数字并通过两次遍历在找到当前的结点的所有匹配子节点，并将其插入 res。
2. 创建一个 Vector<DictionaryTrie> 变量 tmp 以模拟队列在广度优先搜索中的作用，只要当前字符串的长度不为 1，则对 res 中的每一个子节点，取当前字符串从 1 开始的子串，调用 T9Find，返回值只要不为 null 则加入全部加入 tmp 中，遍历结束后令 res = tmp，此时 res 即为所求值；边界：若字符串长度为 1，说明到了最后一层，此时需要对 res 中的所有结果审核关键词有效性。

3. 最后返回 res

4.2.3 T9Shell

该函数模拟一个 shell 环境，可以接受题目中给的命令格式，函数功能很简单，简单判断一下逻辑即可。

4.3 主干代码说明

4.3.1 T9Find

Listing 10: Vector<DictionaryTrie> T9Find(DictionaryTrie trie, String digital)

```

1 static Vector<DictionaryTrie> T9Find(DictionaryTrie trie, String digital) {
2     Vector<DictionaryTrie> res = new Vector<>();
3     int indexOfT9 = Integer.parseInt(String.valueOf(digital.charAt(0)));
4     if (trie == null) return null;
5     for (int i = 0; i < KeyValues[indexOfT9].length(); i++)
6         for (DictionaryTrie son : trie.sons)
7             if (son.ch == KeyValues[indexOfT9].charAt(i)) res.add(son);
8     Vector<DictionaryTrie> tmp = new Vector<>();
9     if (digital.length() != 1) {
10         for (DictionaryTrie son : res) {
11             Vector<DictionaryTrie> x = T9Find(son, digital.substring(1));
12             if (x != null) tmp.addAll(x);
13         }
14         res = tmp;
15     } else for (int i=0;i< res.size();i++) if (res.get(i).dup_times <= 0)
16         res.removeElement(res.remove(i));
17     return res;
18 }

```

4.3.2 T9Shell

Listing 11: void T9Shell()

```

1 public static void T9Shell() {
2     System.out.println("Enter \"exit\" to quit.");

```



```
3      Scanner sc = new Scanner(System.in);
4      String command = "", options = "#";
5      Vector<String> res = new Vector<>();
6      int resIndex = 0;
7      boolean isLookUP = false;
8      while (true) {
9          System.out.println("Enter Key Sequence (or “#” for next word):");
10         System.out.print("> ");
11         if (sc.hasNextLine()) command = sc.next();
12         if (command.equals("exit")) return;
13         else if (command.contains("#")) {
14             options = command.substring(command.indexOf("#"));
15             command = command.substring(0, command.indexOf("#"));
16         } else options = "";
17         // judge if options are all #
18         if (!options.equals("")) {
19             if (options.replace("#", "").length() != 0) {
20                 System.out.println("Please input as tips: some numbers
21                                     followed by some '#'s.");
22                 continue;
23             }
24             int optionCount = options.length();
25             if (command.equals("")) {
26                 if (isLookUP) {
27                     resIndex += optionCount;
28                     if (resIndex < res.size()) System.out.println("'" + res.get(
29                         resIndex) + "'");
30                     else System.out.println("There are no more T9 words.");
31                 } else System.out.println("You're not in searching!");
32             } else {
33                 boolean flag = true; // is command valid
34                 for (int i = 0; i < command.length(); i++) {
35                     if (command.charAt(i) < '0' || command.charAt(i) > '9') {
36                         System.out.println("Please input as tips: some numbers
37                                             followed by some '#'s.");
38                     }
39                 }
40             }
41         }
42     }
```

```
36         flag = false;
37     }
38 }
39 if (flag) {
40     res.clear();
41     Vector<DictionaryTrie> tmp = T9Find(root, command);
42     if(tmp == null || tmp.size() == 0){
43         System.out.println("Not found in current dictionary.");
44         continue;
45     }
46     for (DictionaryTrie t : tmp) res.add(t.printDad());
47     res.sort(String::compareTo);
48     isLookUP = true;
49     System.out.println("'" + res.get(optionCount) + "'");
50     resIndex = optionCount;
51 }
52 }
53 }
54 }
```

4.4 运行结果展示

(结果似乎和题目展示不符, 但经过直接在输入文件中比对没有发现问题)

```
Enter "exit" to quit.
Enter Key Sequence (or "#" for next word):
> 76257
'pocks'
Enter Key Sequence (or "#" for next word):
> #
'rocks'
Enter Key Sequence (or "#" for next word):
> 53556
Not found in current dictionary.
Enter Key Sequence (or "#" for next word):
> #
There are no more T9 words.
Enter Key Sequence (or "#" for next word):
> 76257#
'rocks'
Enter Key Sequence (or "#" for next word):
> 76257##
'soaks'
Enter Key Sequence (or "#" for next word):
> 4423
'gibe'
Enter Key Sequence (or "#" for next word):
> exit
```

4.5 总结和收获

本题目在上一次任务的基础上依然采用了递归的方式实现，没有太大难度。

在写代码/实现一个算法的过程中需要抓住算法的思想内核和本质，而非局限在传统常见的实现方式上。在一开始我总是把这个题目往广度优先的角度上靠，舍本逐末地思考要不要先实现一个队列出来，实际上并无此必要。

附录：源代码汇总

任务一

BSTNode.java

Listing 12: BSTNode.java

```
1 package tree.BST;
2
3 import java.io.IOException;
4 import java.io.PrintWriter;
5
6 //二叉检索树的结点类
7 public class BSTNode<K extends Comparable<K>,V> {
8
9     K key;
10    V value;
11    BSTNode<K, V> parent = null;
12    BSTNode<K, V> left = null;
13    BSTNode<K, V> right = null;
14
15    public BSTNode() {
16        key = null;
17        value = null;
18    }
19
20    public BSTNode(K k, V v) {
21        key = k;
22        value = v;
23    }
24
25    public BSTNode(K k, V v, BSTNode<K, V> p) {
26        key = k;
27        value = v;
```

```
28         parent = p;
29     }
30
31     public BSTNode(K k, V v, BSTNode<K, V> l, BSTNode<K, V> r) {
32         key = k;
33         value = v;
34         left = l;
35         right = r;
36     }
37
38     public BSTNode(K k, V v, BSTNode<K, V> l, BSTNode<K, V> r, BSTNode<
39         K, V> p) {
40         key = k;
41         value = v;
42         left = l;
43         right = r;
44         parent = p;
45     }
46
47     public void insert(K k, V v) {
48         if (this.value == null && this.key == null) {
49             this.key = k;
50             this.value = v;
51         } else if (this.key.compareTo(k) == 0) this.value = v;
52         else if (this.key.compareTo(k) < 0) {
53             if (this.right == null) this.right = new BSTNode<K,
54                 V>(k, v, this);
55             else this.right.insert(k, v);
56         } else if (this.key.compareTo(k) > 0) {
57             if (this.left == null) this.left = new BSTNode<K, V
58                 >(k, v, this);
59             else this.left.insert(k, v);
60         }
61     }
62
63     public V remove(K k) {
```

```
61         BSTNode<K, V> n = this.searchNode(k);
62         if (n == null) return null;
63         V v = n.value;
64         if (n.left != null && n.right != null) {
65             // get the smallest Node of right subtree.
66             BSTNode<K, V> tmp = n.right;
67             while (tmp.left != null) tmp = tmp.left;
68             n.key = tmp.key;
69             n.value = tmp.value;
70             tmp.key = k;
71             tmp.value = v;
72             n.right.remove(k);
73         } else if (n.left != null) {
74             if (n.parent == null) {
75                 n.key = n.left.key;
76                 n.value = n.left.value;
77                 n.left = null;
78             } else {
79                 n.left.parent = n.parent;
80                 if (n.parent.left == n) n.parent.left = n.
                    left;
81                 else n.parent.right = n.left;
82             }
83         } else if (n.right != null) {
84             if (n.parent == null) {
85                 n.key = n.right.key;
86                 n.value = n.right.value;
87                 n.right = null;
88             } else {
89                 n.right.parent = n.parent;
90                 if (n.parent.left == n) n.parent.left = n.
                    right;
91                 else n.parent.right = n.right;
92             }
93         } else {
94             if (n.parent.left == n) n.parent.left = null;
```

```
95         else n.parent.right = null;
96     }
97     return v;
98 }
99
100
101 public V search(K k) {
102     if (this.searchNode(k) != null) return this.searchNode(k).
        value;
103     return null;
104 }
105
106 public BSTNode<K, V> searchNode(K k) {
107     if (this.key == null) return null;
108     else if (k.compareTo(this.key) == 0) return this;
109     else if (k.compareTo(this.key) > 0 && this.right != null)
        return this.right.searchNode(k);
110     else if (k.compareTo(this.key) < 0 && this.left != null)
        return this.left.searchNode(k);
111     else return null;
112 }
113
114 public boolean update(K k, V newV) {
115     BSTNode<K, V> tmp = searchNode(k);
116     if (tmp == null) return false;
117     tmp.value = newV;
118     return true;
119 }
120
121 public boolean isEmpty() {
122     return this.key == null;
123 }
124
125 public void clear() {
126     // considering java has perfect gc, just assign objects as
        null
```

```
127         this.left = null;
128         this.right = null;
129         this.remove(this.key);
130     }
131
132     public int getHeight() {
133         if (this.key == null) return 0;
134         if (this.left == null && this.right == null) return 1;
135         else if (this.left == null) return this.right.getHeight() +
136             1;
137         else if (this.right == null) return this.left.getHeight() +
138             1;
139         else return Math.max(this.left.getHeight(), this.right.
140             getHeight()) + 1;
141     }
142
143     public int getNodeCons() {
144         if (this.left == null && this.right == null) return 1;
145         else if (this.left == null) return this.right.getNodeCons()
146             + 1;
147         else if (this.right == null) return this.left.getNodeCons()
148             + 1;
149         else return this.left.getNodeCons() + this.right.
150             getNodeCons() + 1;
151     }
152
153     public int getDepth() {
154         if (this.parent != null) return 1 + this.parent.getDepth();
155         else return 0;
156     }
157 }
```

testBST.java

Listing 13: testBST.java

```
1 package tree.BST;
```



```
2
3 import java.io.FileReader;
4 import java.io.IOException;
5 import java.io.PrintStream;
6 import java.nio.file.Files;
7 import java.nio.file.Paths;
8 import java.util.List;
9 import java.util.Scanner;
10
11 public class testBST {
12     BSTNode<String, String> root = new BSTNode<>();
13
14
15     public static void main(String[] args) throws IOException {
16         testBST testBST = new testBST();
17         testBST.BSTShell("./input/BST_testcases.txt", "./output/BST_my_res.
18             txt");
19     }
20
21     public void BSTShell(String input, String output) throws IOException {
22         Scanner sc = new Scanner(new FileReader(input));
23         PrintStream out = System.out, fout = new PrintStream(output);
24         System.setOut(fout);
25         String s;
26         String[] tmp;
27         while (sc.hasNextLine()) {
28             tmp = sc.nextLine().split(" ");
29             switch (tmp[0].charAt(0)) {
30                 case '+':
31                     root.insert(tmp[1], tmp[3].substring(1, tmp[3].length()
32                         - 1));
33                     break;
34                 case '-':
35                     s = root.remove(tmp[1]);
36                     if (s == null) System.out.println("remove unsuccess ---
```

```

        " + tmp[1]);
36     else System.out.println("remove success ---" + tmp[1] +
        " " + s);
37     break;
38     case '?':
39         s = root.search(tmp[1]);
40         if (s == null) System.out.println("search unsucess ---"
        " + tmp[1]);
41         else System.out.println("search success ---" + tmp[1] +
        " " + s);
42         break;
43     case '=':
44         if (root.update(tmp[1], tmp[3].substring(1, tmp[3].
        length() - 1)))
45             System.out.println("update success ---" + tmp[1] +
46             " " + tmp[3].substring(1, tmp[3].length() -
        1));
47         else System.out.println("update unsucess ---" + tmp[1]
        +
48             " " + tmp[3].substring(1, tmp[3].length() - 1))
        ;
49         break;
50     case '#':
51         PrintSplitLine();
52         System.out.println("There are " + root.getNodeCons() +
        " nodes in this BST.");
53         System.out.println("The height of this BST is " + root.
        getHeight() + ".");
54         PrintSplitLine();
55     }
56 }
57 System.setOut(out);
58 compareTwoFile("./output/BST_my_res.txt", "./output/BST_result.txt")
    ;
59 }
60

```

```

61
62     private void PrintSplitLine() {
63         for (int i = 0; i < 29; i++) System.out.print('-');
64         System.out.println();
65     }
66
67     public static void compareTwoFile(String oldFile, String newFile)
68         throws IOException {
69         List<String> list1 = Files.readAllLines(Paths.get(oldFile));
70         List<String> list2 = Files.readAllLines(Paths.get(newFile));
71         System.out.printf("compare two files: %s and %s \n", oldFile,
72             newFile);
73         List<String> finalList = list2.stream().filter(line ->
74             list1.stream().noneMatch(line2 -> line2.equals(line))
75         ).toList();
76         if (finalList.size() == 0) {
77             System.out.println("\tNO difference");
78         }else{
79             System.out.println("diff:");
80             finalList.forEach(System.out::println);
81         }
82     }
83 }

```

任务二

wordBST.java

Listing 14: wordBST.java

```

1 package tree.BST;
2
3 import java.io.*;
4 import java.util.*;
5
6 public class wordBST {
7     BSTNode<String, Vector<Integer>> root = new BSTNode<>();

```

```
8
9     public static void main(String[] args) throws FileNotFoundException {
10         System.setOut(new PrintStream("./output/index_result.txt"));
11         wordBST w = new wordBST();
12         w.ReadIn("./input/article.txt");
13         w.printInorder();
14     }
15
16
17     /**
18      * read words from filename
19      * @param filename
20      */
21     private void ReadIn(String filename) throws FileNotFoundException {
22         Scanner sc = new Scanner(new FileReader(filename));
23         int linenums = 0;
24         String tmp;
25         Vector<Integer> v;
26         Vector<String> res = new Vector<>();
27         //可能会出现异常，直接throws就行了
28         while (sc.hasNextLine()) {
29             ++linenums;
30             tmp = sc.nextLine();
31             StringTokenizer st = new StringTokenizer(tmp, "[ ,.! ? \\n\\\"'-)
32                 :;&*(");
33             while (st.hasMoreTokens()) {
34                 tmp = st.nextToken().toLowerCase();
35                 if (tmp.length() <= 2 || tmp.matches("[0-9]+[a-z]*"))
36                     continue;
37                 if ((v = root.search(tmp)) != null) v.add(linenums);
38                 else {
39                     v = new Vector<>();
40                     root.insert(tmp, v);
41                     v.add(linenums);
42                 }
43             }
44         }
45     }
```

```

42     }
43 }
44
45 private void printHelper(BSTNode<String,Vector<Integer>> b) {
46     if (b == null) return;
47     printHelper(b.left);
48     System.out.println "[" + b.key + " --- < " +
49         b.value.toString().substring(1,b.value.toString().length()
50             -1).replace(",",",") + " >]");
51     printHelper(b.right);
52 }
53
54 public void printInorder(){
55     printHelper(this.root);
56 }

```

任务三

DictionaryTrie.java

Listing 15: DictionaryTrie.java

```

1 package tree.Trie;
2
3 import java.io.FileNotFoundException;
4 import java.io.FileReader;
5 import java.util.Scanner;
6 import java.util.Vector;
7
8 public class DictionaryTrie {
9     DictionaryTrie dad = null;
10    Vector<DictionaryTrie> sons = new Vector<DictionaryTrie>();
11    char ch;
12    int dup_times = 0;
13
14    public DictionaryTrie(char c) {

```

```
15         ch = c;
16     }
17
18     public DictionaryTrie(DictionaryTrie father, char c) {
19         dad = father;
20         father.sons.add(this);
21         ch = c;
22     }
23
24     // insert a "keyword" to this trie.
25     public void insert(String keyword) {
26         // search for same prefix string.
27         for (DictionaryTrie son : this.sons) {
28             if (son.ch == keyword.charAt(0)) {
29                 if (keyword.length() == 1) son.dup_times++;
30                 else son.insert(keyword.substring(1));
31                 return;
32             }
33         }
34         // if this has no son which have same prefix as keyword.
35         DictionaryTrie tmp = new DictionaryTrie(this, keyword.charAt(0));
36         if (keyword.length() != 1) tmp.insert(keyword.substring(1));
37         else tmp.dup_times++;
38     }
39
40     /**
41      * delete a "keyword" to this Trie.
42      * if found no matter it is keyword
43      * if not found, printSons ERROR message on the terminal.
44      */
45     public void delete(String keyword) {
46         DictionaryTrie res = this.match(keyword);
47         if (res == null) System.out.println("ERROR--[Trie.delete]: no such
            keyword as \"" + keyword + "\"");
48         else res.dup_times = 0;
49     }
```

```
50
51 // return String
52 public Vector<String> PrefixAs(String prefix) {
53     DictionaryTrie res = this.match(prefix);
54     if (res == null) {
55         System.out.println("ERROR--[Trie.PrefixAs]: no such keyword to
56             match as \"" + prefix + "\"");
57         return null;
58     }
59     Vector<String> sen = res.printSons();
60     sen.replaceAll(s -> prefix + s);
61     if (res.dup_times > 0) sen.add(prefix);
62     sen.sort(String::compareTo);
63     return sen;
64 }
65
66 /**
67  * begin with "this", iterate to get the bottom Trie whose prefix match
68  * the given target.
69  * if found, return Trie; else return null
70  */
71 public DictionaryTrie match(String target) {
72     if (this.sons == null) return null;
73     for (DictionaryTrie son : this.sons) {
74         if (son.ch == target.charAt(0)) {
75             if (target.length() == 1) return son;
76             else return son.match(target.substring(1));
77         }
78     }
79     return null;
80 }
81
82 /**
83  * return all valid keyword begin with this Trie use DFS.
84  */
85 public Vector<String> printSons() {
```

```
84         Vector<String> res = new Vector<>();
85         if (this.sons == null) return null;
86         for (DictionaryTrie son : this.sons) {
87             if (son.dup_times > 0) res.add(String.valueOf(son.ch));
88             Vector<String> tmp = son.printSons();
89             if (tmp != null) tmp.replaceAll(s -> son.ch + s);
90             res.addAll(tmp);
91         }
92         return res;
93     }
94
95     /**
96      * read in by lines
97      *
98      * @param Filename
99      */
100    public void ReadIn(String Filename) {
101        try {
102            Scanner sc = new Scanner(new FileReader(Filename));
103            while (sc.hasNextLine()) { //按行读取字符串
104                String line = sc.nextLine();
105                this.insert(line);
106            }
107        } catch (FileNotFoundException e) {
108            System.out.println("文件未找到~! ");
109        }
110    }
111
112    public String printDad() {
113        if (this.dad == null) return "";
114        else return this.dad.printDad() + this.ch;
115    }
116
117    public Vector<String> printDad_Sons() {
118        Vector<String> res = this.printSons();
119        res.replaceAll(s -> this.printDad() + s);
```



```
120         return res;
121     }
122 }
```

testTrie.java

Listing 16: testTrie.java

```
1 package tree.Trie;
2
3 import java.util.Vector;
4
5 public class testTrie {
6     private static DictionaryTrie root = new DictionaryTrie('\0');
7
8     public static void main(String[] args) {
9         root.ReadIn("./input/dictionary.txt");
10        InsertNumTest();
11        System.out.println();
12        aarBasicTest();
13        System.out.println();
14        aarDeleteTest();
15        System.out.println();
16        aarMultiInsertTest();
17        System.out.println();
18        zyzMultiTest();
19    }
20
21    public static void InsertNumTest() {
22        System.out.println("---- InsertNumTest begins ----");
23        System.out.println("expect:\n\t80368\nrun:");
24        System.out.println("\t" + root.printSons().size());
25    }
26
27    public static void aarBasicTest() {
28        System.out.println("---- aarBasicTest begins ----");
29        System.out.println("expect:\n\t5\taardvark\taardwolf\taargh\taarrgh
```

```
        \taarrghh\nrun:");
30     Vector<String> res = root.PrefixAs("aar");
31     if (res != null) {
32         System.out.print("\t" + res.size());
33         for (String s : res) System.out.print("\t" + s);
34     }
35     System.out.println();
36 }
37
38 public static void aarDeleteTest() {
39     System.out.println("---- aarDeleteTest begins ----");
40     System.out.println("expect:\n\t4\taardvark\taardwolf\taarrgh\
        taarrghh\nrun:");
41     root.delete("aargh");
42     Vector<String> res = root.PrefixAs("aar");
43     if (res != null) {
44         System.out.print("\t" + res.size());
45         for (String s : res) System.out.print("\t" + s);
46     }
47     System.out.println();
48 }
49
50 public static void aarMultiInsertTest() {
51     System.out.println("---- aarMultiInsertTest begins ----");
52     System.out.println("expect:\n\t4\taardvark\taardwolf\taarrgh\
        taarrghh\nrun:");
53     root.insert("aardwolf");
54     Vector<String> res = root.PrefixAs("aar");
55     if (res != null) {
56         System.out.print("\t" + res.size());
57         for (String s : res) System.out.print("\t" + s);
58     }
59     System.out.println();
60 }
61
62 public static void zyzMultiTest() {
```

```

63     System.out.println("---- aarMultiInsertTest begins ----");
64     System.out.println("expect:\n\t1\tzyzzyvas\nrun:");
65     root.delete("zyzzyva");
66     root.insert("zyzzyvas");
67     Vector<String> res = root.PrefixAs("zyz");
68     if (res != null) {
69         System.out.print("\t" + res.size());
70         for (String s : res) System.out.print("\t" + s);
71     }
72     System.out.println();
73 }
74 }

```

任务四

T9Trie.java

Listing 17: T9Trie.java

```

1  package tree.Trie;
2
3  import java.util.Scanner;
4  import java.util.Vector;
5
6  public class T9Trie {
7      private static DictionaryTrie root = new DictionaryTrie('\0');
8      final static String[] KeyValues = {"", "", "abc", "def", "ghi", "jkl",
9          "mno", "pqrs", "tuv", "wxyz"};
10
11     public static void main(String[] args) {
12         root.ReadIn("./input/dictionary.txt");
13         T9Shell();
14     }
15
16     /**
17      * for any-length given digital numbers, return matched prefixes;
18      *

```

```

18      * @param digital given digital numbers
19      * @param trie     given Trie tree
20      * @return matched T9 words
21      */
22      static Vector<DictionaryTrie> T9Find(DictionaryTrie trie, String
        digital) {
23          Vector<DictionaryTrie> res = new Vector<>();
24          int indexOfT9 = Integer.parseInt(String.valueOf(digital.charAt(0)))
            ;
25          if (trie == null) return null;
26          for (int i = 0; i < KeyValues[indexOfT9].length(); i++)
27              for (DictionaryTrie son : trie.sons)
28                  if (son.ch == KeyValues[indexOfT9].charAt(i)) res.add(son);
29          Vector<DictionaryTrie> tmp = new Vector<>();
30          if (digital.length() != 1) {
31              for (DictionaryTrie son : res) {
32                  Vector<DictionaryTrie> x = T9Find(son, digital.substring(1)
                    );
33                  if (x != null) tmp.addAll(x);
34              }
35              res = tmp;
36          } else for (int i=0;i< res.size();i++) if (res.get(i).dup_times <=
            0) res.removeElement(res.remove(i));
37          return res;
38      }
39
40      public static void T9Shell() {
41          System.out.println("Enter \"exit\" to quit.");
42          Scanner sc = new Scanner(System.in);
43          String command = "", options = "#";
44          Vector<String> res = new Vector<>();
45          int resIndex = 0;
46          boolean isLookUP = false;
47          while (true) {
48              System.out.println("Enter Key Sequence (or “#” for next word)
                :");

```

```
49         System.out.print("> ");
50         if (sc.hasNextLine()) command = sc.next();
51         if (command.equals("exit")) return;
52         else if (command.contains("#")) {
53             options = command.substring(command.indexOf("#"));
54             command = command.substring(0, command.indexOf("#"));
55         } else options = "";
56         // judge if options are all #
57         if (!options.equals("")) {
58             if (options.replace("#", "").length() != 0) {
59                 System.out.println("Please input as tips: some numbers
60                                     followed by some '#'s.");
61                 continue;
62             }
63             int optionCount = options.length();
64             if (command.equals("")) {
65                 if (isLookUP) {
66                     resIndex += optionCount;
67                     if (resIndex < res.size()) System.out.println("'" + res.
68                                     get(resIndex) + "'");
69                     else System.out.println("There are no more T9 words.");
70                 } else System.out.println("You're not in searching!");
71             } else {
72                 boolean flag = true; // is command valid
73                 for (int i = 0; i < command.length(); i++) {
74                     if (command.charAt(i) < '0' || command.charAt(i) > '9')
75                     {
76                         System.out.println("Please input as tips: some
77                                     numbers followed by some '#'s.");
78                         flag = false;
79                     }
80                 }
81                 if (flag) {
82                     res.clear();
83                     Vector<DictionaryTrie> tmp = T9Find(root, command);
```

```
81         if(tmp == null || tmp.size() == 0){
82             System.out.println("Not found in current dictionary
83                 .");
84             continue;
85         }
86         for (DictionaryTrie t : tmp) res.add(t.printDad());
87         res.sort(String::compareTo);
88         isLookUP = true;
89         System.out.println("'" + res.get(optionCount) + "'");
90         resIndex = optionCount;
91     }
92 }
93 }
94 }
```