|  |  |
| --- | --- |
| **数据结构与算法 综合训练报告** | |
| 图片包含 标牌  已生成极高可信度的说明 | |
|  | |
|  | |
| **姓名** |  |
| **班级** |  |
| **学号** |  |
| **电话** |  |
| **Email** |  |
| **日期** |  |

# 用树解决问题

**任务1：实现BST数据结构 1**

题目 1

数据设计 1

算法设计 2

主干代码说明 3

运行结果展示 5

总结和收获 6

**任务2：使用BST为文稿建立单词索引表 7**

题目 7

数据设计 7

算法设计 8

主干代码说明 8

运行结果展示 9

总结和收获 10

**任务3：实现Trie 11**

题目 11

数据设计 12

算法设计 12

主干代码说明 13

运行结果展示 15

总结和收获 15

**任务4：使用Trie实现T9键盘 16**

题目 16

数据设计 17

算法设计 17

主干代码说明 17

运行结果展示 19

总结和收获 20

**附录：源代码 21**

### 任务1：实现BST数据结构

### 一、题目

### 二、数据设计

BST中需要存储的数据有：BST的根结点；

BST的结点BinNode中需要存储的数据有：结点的key，结点的value，结点的左、右子节点的指针；

### 三、算法设计

依据BSTADT，BST中主要需要实现的方法有：

1. insert:为便于递归遍历BST，实现返回值和参数均含BinNode的辅助方法inserthelp，调用inserthelp传入root；
2. inserthelp:当前遍历的节点为rt，若rt为null说明遍历到底，返回一个新节点；比较rt中的key与输入值key，不相等时，分别对左、右子节点递归调用inserthelp，相等时更新value；最后返回rt。
3. remove:与insert基本同理，但为了获得被移除的节点的数据。创建一个新节点作为参数传入removehelp，移除前进行复制操作；
4. removehelp: 当前遍历的节点为rt，若rt为null说明遍历到底，返回null表明找不到；比较key，不相等时，对子节点递归调用removehelp，相等时开始执行删除操作，先将rt的key和value复制到removed中，此后判断rt的子结点数，若只有单节点，将rt更改为另一节点；若有双节点，找到右子树的最小节点，调用removehelp删除它，并将rt作为removed传入，这样就将右子树最小节点的值存储到当前rt中了。最后返回rt。
5. getMin：有左子树就是最小值，否则递归调用自己；
6. search:调用searchhelp并传入root；
7. searchhelp：当前遍历的节点为rt，若rt为null，返回null表明找不到；比较key，不相等时，对子节点递归调用searchhelp，相等时返回节点value；
8. update：调用updatehelp并传入root；
9. updatehelp：当前遍历的节点为rt，若rt为null，返回false表明找不到；比较key，不相等时，对子节点递归调用updatehelp，相等时更新value并返回true；
10. isEmpty：返回root是否为null；
11. clear：给root赋值null；
12. showStructure：调用getNodes和getHeight，给PrintWriter写入指定形式的字符串；
13. getNodes：返回此节点以后的节点数。若输入为null则返回0，否则返回两个子节点以后的节点数之和加一；
14. getHeight：返回此节点以后的高度。若输入为null返回0，否则返回左右子节点以后的高度之最大值加一；
15. printInOrder：调用printhelp并传入root；
16. printhelp：对BST进行中序周游，即先对rt的左子节点调用printhelp，之后给PrintWriter写入自己，再对rt的右子节点调用。若输入为null，直接返回。

BinNode中需要实现的方法主要有获取和修改其中的数据、将其中的key与输入的key比大小。

BSTTest中需要实现main方法，其中利用Scanner按行输入，根据输入的第一个字符对BST执行对应的操作。

### 四、主干代码说明

**public void** insert(K key, V value) {  
 **root**=inserthelp(**root**,key,value);  
}  
**private** BinNode<K,V> inserthelp(BinNode<K,V> rt,K key,V value){*//递归调用，遍历BST，找到插入位置* **if**(rt==**null**){*//遍历到底* **return new** BinNode<>(key, value);  
 }  
 **if**(rt.biggerThan(key)){*//遍历到的rt的key较大* rt.setLeft(inserthelp(rt.getLeft(),key,value));  
 }  
 **else if**(rt.smallerThan(key)){*//遍历到的rt的key较小* rt.setRight(inserthelp(rt.getRight(),key,value));  
 }  
 **else** {*//遍历到的rt的key相等* rt.setValue(value);  
 }  
 **return** rt;  
}

以上是实现插入操作的相关代码；

**else** {*//找到了* removed.setKey(rt.getKey());  
 removed.setValue(rt.getValue());*//使用这种方式将删除的内容传出* **if**(rt.getLeft()==**null**)rt=rt.getRight();*//左子树空* **else if** (rt.getRight()==**null**)rt=rt.getLeft();*//右子树空* **else** {*//双子树* BinNode<K,V> rightMin = getMin(rt.getRight());*//右子树最小值* removehelp(rt,rightMin.getKey(),rt);*//同时实现数据复制、删除最小值* }  
}

以上是实现移除操作的代码中，找到需删除的节点后的操作；

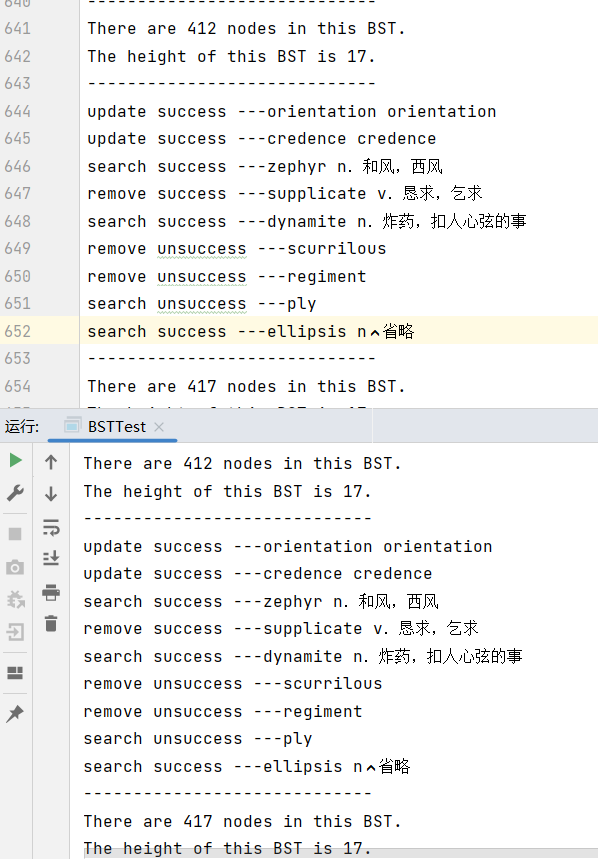
**private int** getNodes(BinNode<K,V> rt){  
 **if**(rt==**null**)**return** 0;  
 **return** getNodes(rt.getRight())+getNodes(rt.getLeft())+1;  
}  
**private int** getHeight(BinNode<K,V> rt){  
 **if**(rt==**null**)**return** 0;  
 **return** Math.*max*(getHeight(rt.getLeft()),getHeight(rt.getRight()))+1;  
}

以上是获取节点数和高度的相关代码；

**private void** printhelp(BinNode<K,V> rt,PrintWriter pw){*//中序周游* **if** (rt==**null**)**return**;  
 printhelp(rt.getLeft(),pw);  
 pw.write(**"["**+rt.getKey()+**" --- < "**+rt.getValue()+**">]\n"**);  
 printhelp(rt.getRight(),pw);  
}

以上是实现顺序输出的主干代码；

**while** (scanner.hasNextLine()){  
 String line = scanner.nextLine();  
 **switch** (line.charAt(0)){*//判断是什么操作* **case '#'**->test.showStructure(pw);  
 **case '?'**->{  
 key= line.substring(3,line.length()-2);  
 value=test.search(key);  
 **if**(value==**null**)pw.write(**"search unsuccess ---"**+key+**"\n"**);*//搜索不成功* **else** pw.write(**"search success ---"**+key+**" "**+value+**"\n"**);*//搜索成功* }  
 **case '-'**->{  
 key=line.substring(3,line.length()-2);  
 value=test.remove(key);  
 **if**(value==**null**)pw.write(**"remove unsuccess ---"**+key+**"\n"**);*//搜索不成功* **else** pw.write(**"remove success ---"**+key+**" "**+value+**"\n"**);*//搜索成功* }  
 **default**->{  
 key=line.split(**" , "**)[0];  
 value=line.split(**" , "**)[1];  
 key=key.substring(3);  
 value=value.substring(1,value.length()-3);  
 **switch** (line.charAt(0)){  
 **case '+'**->test.insert(key,value);  
 **case '='**->{  
 **if**(test.update(key,value))pw.write(**"update success ---"**+key+**" "**+value+**"\n"**);  
 **else** pw.write(**"update unsuccess ---"**+key+**" "**+value+**"\n"**);  
 }  
 }  
 }  
 }  
}

以上是实现根据输入的内容执行相关操作的代码，其中第一个switch的default意味着需要提取两个参数，提取之后再判断是添加操作还是更新操作。

### 五、运行结果展示

注：截图上部分是BST\_result.txt，下部分是输出结果。

### 六、总结和收获

本题实现了一个二叉检索树，通过实践加深了对二叉树和二叉树的应用的理解。在实现BST的插入、删除、查找和更新操作时，都需要采用一个能够递归的方法来辅助，以便实现结点的向下遍历。特别地，在实现插入和删除操作时，由于在结点变化时还需要上一级结点，必须要在每次递归时都对结点的子树进行更改，但假若没有结点变化，方法将返回参数rt本身，虽然执行了更改操作但其实子树并未发生变化。

### 任务2：使用BST为文稿建立单词索引表

### 一、题目

### 二、数据设计

该任务需要构建一个任务1中实现的BST，数据为单词和行号。先输入的行号先输出，因此可以采取此前任务中实现的队列类型存储。

### 三、算法设计

在main函数中构建一个BST<String,ResizingQueue>，利用Scanner读取每行中的每个单词，进行数据清洗后插入到BST中，并给队列插入当前行号。最后再调用BST的printInOrder方法输出。

数据清洗的规则如下：若单词长度小于等于2，或含有单引号、双引号、短横线、括号，则直接舍去该单词。若该单词的结尾为逗号、句点、感叹号、问号，则舍去单词的最后一位。getAvailable方法用来进行数据清洗，返回null代表舍去。

### 四、主干代码说明

**for** (**int** line=1;scanner.hasNextLine();line++){*//依据article建立BST* String[] words=scanner.nextLine().split(**" "**);  
 **for**(String word : words){*//遍历单词* word=*getAvailable*(word);*//处理单词，排除无效的并对有标点的进行处理* **if**(word!=**null**){*//单词有效* ResizingQueue<Integer> queue=wordBST.search(word);  
 **if**(queue==**null**){*//单词在BST中不存在* queue=**new** ResizingQueue<>();  
 queue.enqueue(line);  
 wordBST.insert(word,queue);  
 }  
 **else** {*//单词已经存在* queue.enqueue(line);  
 }  
 }  
 }  
}

以上是BST的构建过程，是任务的主要部分。

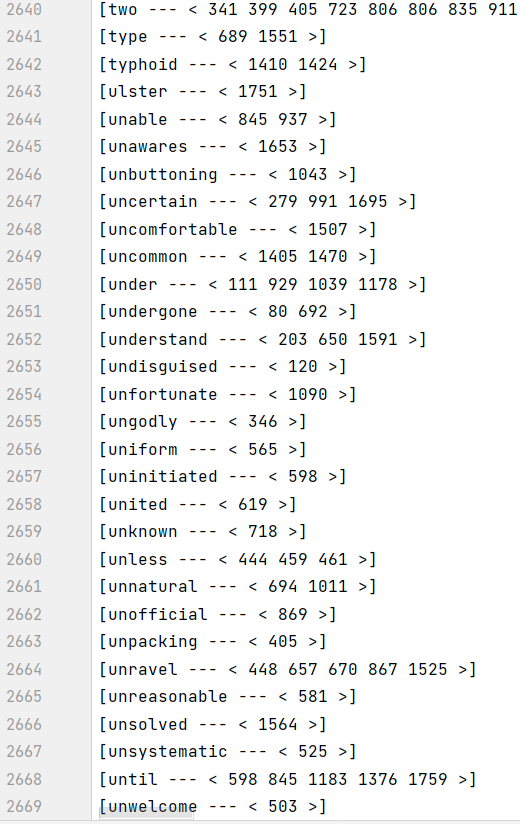
**if**(word.length()<=2)**return null**;  
**for**(**int** i=0;i<word.length();i++){  
 **char** thisChar=word.charAt(i);  
 **if**(thisChar== 39||thisChar== 34||thisChar== **'-'**||thisChar==**'('**||thisChar==**')'**)**return null**;*//单引号，双引号，短横线，括号* **if**(thisChar>=**'1'**&&thisChar<=**'9'**)**return null**;*//数字* **if**(i==word.length()-1&&(thisChar==**','**||thisChar==**'.'**||thisChar==**'?'**||thisChar==**';'**||thisChar==**'!'**)){*//末尾是句号，逗号，问号，分号，感叹号* **return** word.substring(0,word.length()-1);  
 }  
}  
**return** word;

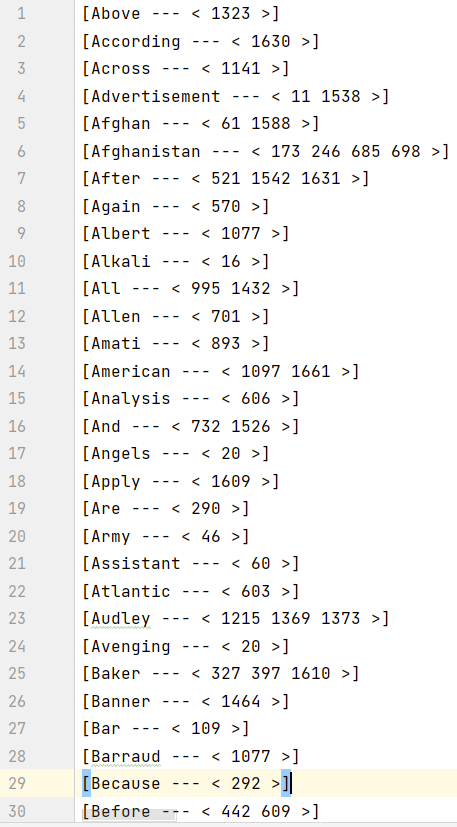
以上是数据清洗的方法的主要部分。

**public** String toString(){  
 String str = **""**;  
 **while** (size()>0){  
 str+=dequeue()+**" "**;  
 }  
 **return** str;  
}

以上是重写了ResizingQueue的toString方法，直接采用边退队边输出的方法，简便快捷。

### 五、运行结果展示

输出index\_result.txt的内容如下：

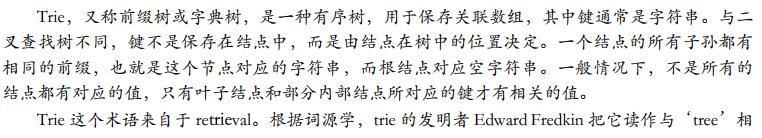
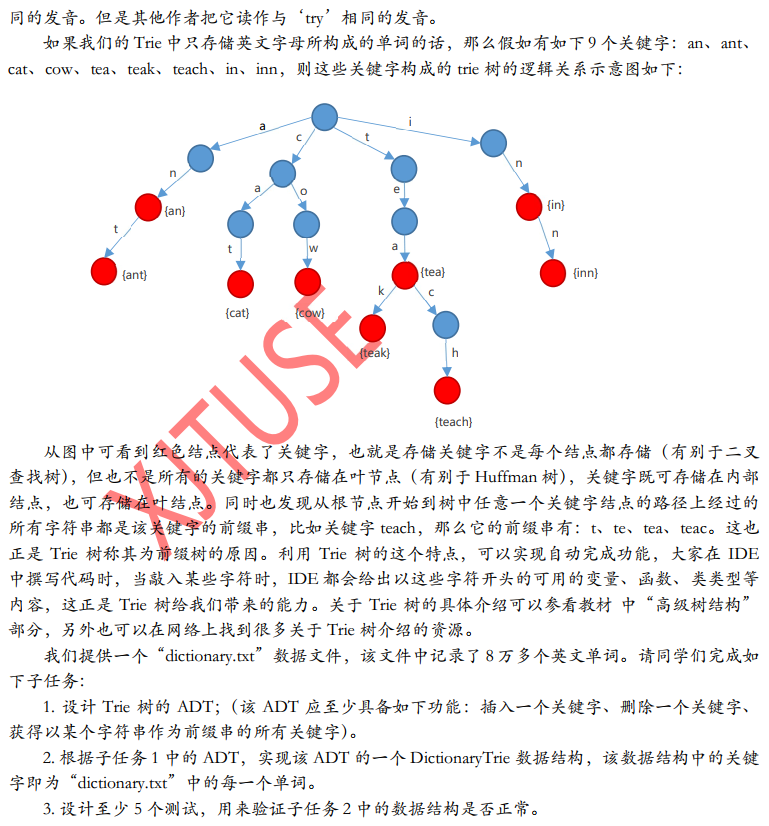
经核对与原文章一致。

### 六、总结和收获

这个任务是对任务1中实现的BST的应用，构建了一个类型为<String,ResizingQueue>的BST，并依据article.txt进行输入，形成单词索引表。利用先前实验的队列数据结构，一是可以先进先出，简单方便，二是队列的大小是根据数据量实时变化的，省去了数据溢出的烦恼，也避免了空间的浪费。

### 任务3：实现Trie

### 一、题目



### 二、数据设计

采取动态“左子节点/右兄弟节点”表示法来实现这个树。

DictionaryTrie应包含一个DictionaryTrieNode类型的根节点；

DictionaryTrieNode应包含连接该节点的线上的字母、该节点的下级节点和右节点的指针、该节点的关键字（没有关键字则为null）。

### 三、算法设计

Trie树的ADT包含insert方法、remove方法和getWords方法，对应插入、删除和获得该前缀的所有关键字。

与实现BST一样，实现DictionaryTrie中的许多方法也需要进行递归，因此引入辅助方法来提供递归。需要实现的方法有：

1. insert：调用inserthelp进行递归；
2. inserthelp：判断当前结点rt是否为关键字的一部分，若是，则判断当前关键字是否已经被遍历完，若是则只需更改当前结点，若否则继续对子节点调用本方法。若不是，则对结点的右结点调用本方法，以寻找当前关键字内的结点。若rt为null，表明需要新建结点，根据关键字是否已经被遍历完，建立新的叶节点或中间结点。
3. remove：调用removehelp进行递归；
4. removehelp：若当前结点rt为null，说明提供的关键字找不到。判断rt是否为关键字的一部分，若是，则判断当前关键字是否已经被遍历完，若是，则继续判断rt之后还有没有结点，若有则不删去结点，只删去关键字。若无则删去结点，即返回结点的右节点。若没有遍历完，递归对子节点调用本方法，若返回为null即该节点没有子节点了，则删去此节点返回右节点。最后，若rt不是关键字的一部分，对右节点递归调用本方法。
5. getWords:该方法返回以key为前缀的所有关键字。利用for循环和判断，取得前缀对应的结点的子节点rt，之后利用getWordshelp方法，返回这个结点的右侧和下侧的所有关键字。
6. getWordshelp：该方法新建一个ArrayList，若输入为null返回空ArrayList，非空时向ArrayList添加rt的关键字（若有），并对rt的子结点和右结点调用本方法，将结果加入ArrayList中。

DictionaryTrieNode中只需要获取和修改数据的方法。

DictionaryTrieTest中，与任务1类似地实现了根据符号执行对应的操作，+，-分别代表插入和移除，#代表退出，直接输入单词为获取以之为前缀的单词。

### 四、主干代码说明

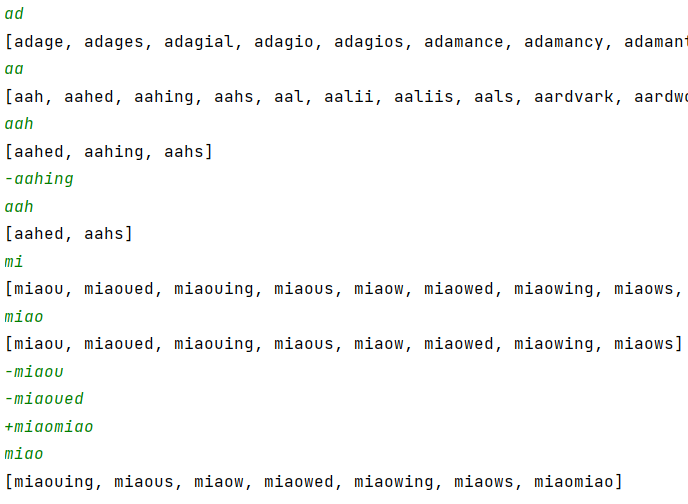
**public void** insert(String key) {  
 **root**=inserthelp(key,0,**root**);  
}  
**private** DictionaryTrieNode inserthelp(String key, **int** index, DictionaryTrieNode rt){*//从结点rt开始插入key的index之后位* **if**(rt==**null**){*//需新建结点* **if**(index==key.length()-1){*//关键字结束* **return new** DictionaryTrieNode(key.charAt(index),key);*//新建叶结点* }  
 **return new** DictionaryTrieNode(key.charAt(index),inserthelp(key,index+1,**null**));*//新建中间结点* }  
 **if**(rt.getLetter()==key.charAt(index)){*//当前结点是关键字的一部分* **if**(index==key.length()-1) {*//关键字结束* rt.setWord(key);*//在树中，更改当前结点* **return** rt;  
 }  
 rt.setNext(inserthelp(key,index+1,rt.getNext()));  
 }  
 **else** {*//当前结点不是关键字的一部分* rt.setRight(inserthelp(key,index,rt.getRight()));  
 }  
 **return** rt;  
}

以上是对于插入方法的实现。

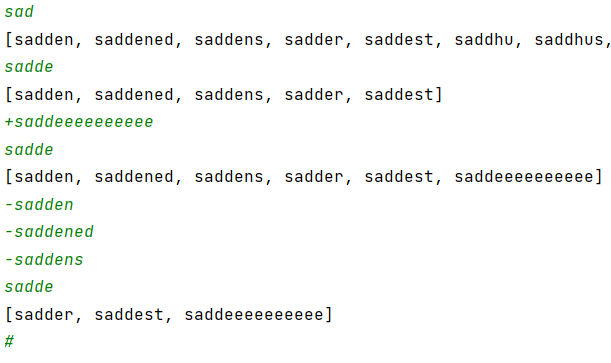
**private** DictionaryTrieNode removehelp(String key, **int** index, DictionaryTrieNode rt){  
 **if**(rt==**null**){*//关键字不在树中* **return** rt;  
 }  
 **if**(rt.getLetter()==key.charAt(index)){*//当前结点是关键字的一部分* **if**(index==key.length()-1) {*//关键字结束* **if**(rt.getNext()!=**null**){*//向后还有结点* rt.setWord(**null**);  
 **return** rt;  
 }  
 **else** {*//向后没有结点* **return** rt.getRight();  
 }  
 }  
 rt.setNext(removehelp(key,index+1,rt.getNext()));  
 **if**(rt.getNext()==**null**&&rt.getWord()==**null**)**return** rt.getRight();*//若该节点向后没有节点，且该节点没有关键字，该节点需被移除* **return** rt;  
 }  
 *//当前结点不是关键字的一部分* rt.setRight(removehelp(key,index,rt.getRight()));  
 **return** rt;  
}

以上是对移除方法的实现，与插入有异曲同工之妙。

**public** ArrayList<String> getWords(String key) {*//以key为前缀串的所有关键字* DictionaryTrieNode rt=**root**;*//根据输入的关键字找到的结点* **for**(**int** i=0;i<key.length();){  
 **if**(rt==**null**){*//在当前树中找不到当前关键字* **return null**;  
 }  
 **else if**(rt.getLetter()==key.charAt(i)){*//当前结点是关键字的一部分* rt=rt.getNext();  
 i++;  
 }  
 **else** {*//当前结点不是关键字的一部分* rt=rt.getRight();  
 }  
 }  
 **return** getWordshelp(rt);  
}  
**private** ArrayList<String> getWordshelp(DictionaryTrieNode rt){*//获得结点rt右下级的所有关键字* ArrayList<String> words = **new** ArrayList<>();  
 **if**(rt==**null**)**return** words;*//遍历到底* **if**(rt.getWord()!=**null**)words.add(rt.getWord());*//此节点有关键字* words.addAll(getWordshelp(rt.getNext()));  
 words.addAll(getWordshelp(rt.getRight()));  
 **return** words;  
}

以上是实现获得有特定前缀的单词的代码。根据任务要求中的说明，以string为前缀的单词并不包括string自身，因此直接获取了前缀结点的子结点并调用getWordshelp。

### 五、运行结果展示

注：该数据结构在构造时输入了dictionary.txt的数据，同时也可以通过+,-操作更改其中的数据。直接输入字词时，将返回以之为前缀的单词集。

### 六、总结和收获

本任务实现了一个Trie树即字典树。实践采用了动态“左子节点/右兄弟节点”方法实现树，加深了我对树数据结构的认识。实践发现，实现树的插入、删除和读取操作与二叉树可以采取相同思路，但由于右侧结点的引入，相较于二叉树更为复杂。

### 任务4：使用Trie实现T9键盘

### 一、题目

### 二、数据设计

用于实现T9键盘的Trie树与词典的Trie树区别不大。T9Trie中依旧需要一个根节点，且每个T9TrieNode中依旧需要子结点、右节点的指针，但结点中的连接结点的线上应为数字，结点存储的数据应为一个ArrayList<String>，存储在该种数字输入下所有可能的单词。

### 三、算法设计

T9Trie和T9TrieNode的算法也与DictionaryTrie基本一致。区别在于：

1. T9Trie实现一个toNum方法，用于将字母转换为T9键盘上的数字，例如a转换为2。toNum方法采取数个if实现即可。在判断当前结点是否是关键字的一部分时，必须调用toNum方法。
2. T9TrieNode实现了方法addWord和deleteWord，对结点中存储的单词数组进行添加或者删除。同时，结点的构造方法也需变化，初始化单词数组。
3. T9Trie的remove方法中，结点移除的条件变更为该节点向后没有节点，且该节点没有关键字。
4. getWords方法只需找到输入的数字串对应的结点，返回其中存储的数组即可。

### 四、主干代码说明

**private** T9TrieNode inserthelp(String key, **int** index, T9TrieNode rt){*//从结点rt开始插入key的index之后位* **if**(rt==**null**){*//需新建结点* **if**(index==key.length()-1){*//关键字结束* **return new** T9TrieNode(toNum(key.charAt(index)),key);*//新建叶结点* }  
 **return new** T9TrieNode(toNum(key.charAt(index)),inserthelp(key,index+1,**null**));*//新建中间结点* }  
 **if**(rt.getNum()==toNum(key.charAt(index))){*//当前结点是关键字的一部分* **if**(index==key.length()-1) {*//关键字结束* rt.addWord(key);*//在树中，更改当前结点* **return** rt;  
 }  
 rt.setNext(inserthelp(key,index+1,rt.getNext()));  
 }  
 **else** {*//当前结点不是关键字的一部分* rt.setRight(inserthelp(key,index,rt.getRight()));  
 }  
 **return** rt;  
}

以上是实现插入的辅助方法；

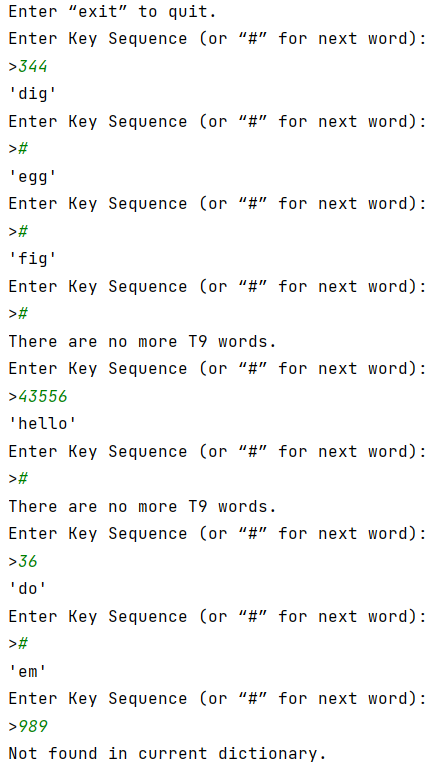
**if**(rt.getNum()==toNum(key.charAt(index))){*//当前结点是关键字的一部分* **if**(index==key.length()-1) {*//关键字结束* **if**(rt.getNext()!=**null**||rt.getWords().size()>1){*//向后还有结点，或此结点还有别的关键字* rt.deleteWord(key);  
 **return** rt;  
 }  
 **else** {*//向后没有结点，且该节点只有一个关键字，移除它* **return** rt.getRight();  
 }  
 }  
 rt.setNext(removehelp(key,index+1,rt.getNext()));  
 **if**(rt.getNext()==**null**&&rt.getWords().isEmpty())**return** rt.getRight();*//若该节点向后没有节点，且该节点没有关键字，该节点需被移除* **return** rt;  
}

以上是移除方法的主要变更部分；

**public** ArrayList<String> getWords(String key) {*//获得key对应的数个单词* T9TrieNode rt=**root**;*//根据输入的关键字找到的结点* **for**(**int** i=0;i<key.length();){  
 **if**(rt==**null**){*//在当前树中找不到当前关键字* **return null**;  
 }  
 **else if**(rt.getNum()==key.charAt(i)-48){*//当前结点是关键字的一部分* **if**(i<key.length()-1) rt=rt.getNext();*//若不在最后一层，进入下层* i++;  
 }  
 **else** {*//当前结点不是关键字的一部分* rt=rt.getRight();  
 }  
 }  
 **return** rt.getWords();  
}

这个方法获取数字序列对应的单词数组。

**int** index=0;  
**for**(ArrayList<String> words=**null**;;){  
 System.***out***.print(**"Enter Key Sequence (or “#” for next word):\n>"**);  
 String operation=scanner.next();  
 **if** (operation.equals(**"exit"**))**break**;*//退出* **else if** (operation.equals(**"#"**))index++;  
 **else** {  
 index=0;  
 words=test.getWords(operation);  
 };  
 **if**(words==**null**|| words.size()==0){  
 System.***out***.println(**"Not found in current dictionary."**);  
 }  
 **else if** (words.size()<=index) {  
 System.***out***.println(**"There are no more T9 words."**);  
 } **else** {  
 System.***out***.println(**"\'"**+words.get(index)+**"\'"**);  
 }  
}

以上是T9TrieTest里实现测试的代码。

### 五、运行结果展示

### 六、总结和收获

该任务实现了一个T9键盘的基本功能。该任务的树和结点与任务3基本类似，但由于存储数据的变化，需要在字符和数字间进行转换，并需要更改存储的类型，在进行相关操作时也需按照新类型进行。该任务让我感受到了树这种数据结构的强大作用，可以用于实现很大数据量的特定处理。

# 附录：源代码

### 任务1

BSTADT.java

**import** java.io.IOException;  
**import** java.io.PrintWriter;  
  
**public interface** BSTADT<K **extends** Comparable<K>, V> {  
 **void** insert(K key, V value);  
 V remove(K key);  
 V search(K key);  
 **boolean** update(K key, V value);  
 **boolean** isEmpty();  
 **void** clear();  
 **void** showStructure(PrintWriter pw) **throws** IOException;  
 **void** printInorder(PrintWriter pw) **throws** IOException;  
}

BST.java

**import** java.io.IOException;  
**import** java.io.PrintWriter;  
  
**public class** BST<K **extends** Comparable<K>,V> **implements** BSTADT<K, V> {  
 BinNode<K,V> **root**;  
  
 **public** BST(){**root**=**null**;}  
  
 @Override  
 **public void** insert(K key, V value) {  
 **root**=inserthelp(**root**,key,value);  
 }  
 **private** BinNode<K,V> inserthelp(BinNode<K,V> rt,K key,V value){*//递归调用，遍历BST，找到插入位置* **if**(rt==**null**){*//遍历到底* **return new** BinNode<>(key, value);  
 }  
 **if**(rt.biggerThan(key)){*//遍历到的rt的key较大* rt.setLeft(inserthelp(rt.getLeft(),key,value));  
 }  
 **else if**(rt.smallerThan(key)){*//遍历到的rt的key较小* rt.setRight(inserthelp(rt.getRight(),key,value));  
 }  
 **else** {*//遍历到的rt的key相等* rt.setValue(value);  
 }  
 **return** rt;  
 }  
  
 @Override  
 **public** V remove(K key) {  
 BinNode<K,V> removed = **new** BinNode<>(**null**, **null**);  
 **root**=removehelp(**root**,key,removed);  
 **return** removed.getValue();  
 }  
 **private** BinNode<K,V> removehelp(BinNode<K,V> rt,K key,BinNode<K,V> removed){  
 **if**(rt==**null**){*//遍历到底* **return null**;  
 }  
 **if**(rt.biggerThan(key)){*//遍历到的rt的key较大* rt.setLeft(removehelp(rt.getLeft(),key,removed));  
 }  
 **else if**(rt.smallerThan(key)) {*//遍历到的rt的key较小* rt.setRight(removehelp(rt.getRight(),key,removed));  
 }  
 **else** {*//找到了* removed.setKey(rt.getKey());  
 removed.setValue(rt.getValue());*//使用这种方式将删除的内容传出* **if**(rt.getLeft()==**null**)rt=rt.getRight();*//左子树空* **else if** (rt.getRight()==**null**)rt=rt.getLeft();*//右子树空* **else** {*//双子树* BinNode<K,V> rightMin = getMin(rt.getRight());*//右子树最小值* removehelp(rt,rightMin.getKey(),rt);*//同时实现数据复制、删除最小值* }  
 }  
 **return** rt;  
 }  
 **private** BinNode<K,V> getMin(BinNode<K,V> rt){  
 **if**(rt.getLeft()==**null**){*//最小值* **return** rt;  
 }  
 **return** getMin(rt.getLeft());  
 }  
  
 @Override  
 **public** V search(K key) {  
 **return** searchhelp(**root**,key);  
 }  
 **private** V searchhelp(BinNode<K,V> rt,K key){  
 **if**(rt==**null**){*//遍历到底* **return null**;  
 }  
 **if**(rt.biggerThan(key)){*//遍历到的rt的key较大* **return** searchhelp(rt.getLeft(),key);  
 }  
 **else if**(rt.smallerThan(key)) {*//遍历到的rt的key较小* **return** searchhelp(rt.getRight(),key);  
 }  
 **else** {  
 **return** rt.getValue();  
 }  
 }  
  
 @Override  
 **public boolean** update(K key, V value) {  
 **return** updatehelp(**root**,key,value);  
 }  
 **private boolean** updatehelp(BinNode<K,V> rt,K key,V value){  
 **if**(rt==**null**){*//遍历到底* **return false**;  
 }  
 **if**(rt.biggerThan(key)){*//遍历到的rt的key较大* **return** updatehelp(rt.getLeft(),key,value);  
 }  
 **else if**(rt.smallerThan(key)) {*//遍历到的rt的key较小* **return** updatehelp(rt.getRight(),key,value);  
 }  
 **else** {  
 rt.setValue(value);  
 **return true**;  
 }  
 }  
  
 @Override  
 **public boolean** isEmpty() {  
 **return root** == **null**;  
 }  
  
 @Override  
 **public void** clear() {  
 **root**=**null**;  
 }  
  
 @Override  
 **public void** showStructure(PrintWriter pw) **throws** IOException {  
 **if**(pw==**null**)**throw new** IOException();  
 pw.write(**"-----------------------------\n"**);  
 pw.write(String.*format*(**"There are %d nodes in this BST.\n"**,getNodes(**root**)));  
 pw.write(String.*format*(**"The height of this BST is %d.\n"**,getHeight(**root**)));  
 pw.write(**"-----------------------------\n"**);  
 }  
 **private int** getNodes(BinNode<K,V> rt){  
 **if**(rt==**null**)**return** 0;  
 **return** getNodes(rt.getRight())+getNodes(rt.getLeft())+1;  
 }  
 **private int** getHeight(BinNode<K,V> rt){  
 **if**(rt==**null**)**return** 0;  
 **return** Math.*max*(getHeight(rt.getLeft()),getHeight(rt.getRight()))+1;  
 }  
  
 @Override  
 **public void** printInorder(PrintWriter pw) **throws** IOException {  
 **if**(pw==**null**)**throw new** IOException();  
 printhelp(**root**,pw);  
 }  
 **private void** printhelp(BinNode<K,V> rt,PrintWriter pw){*//中序周游* **if** (rt==**null**)**return**;  
 printhelp(rt.getLeft(),pw);  
 pw.write(**"["**+rt.getKey()+**" --- < "**+rt.getValue()+**">]\n"**);  
 printhelp(rt.getRight(),pw);  
 }  
}

BinNode.java

**public class** BinNode<K **extends** Comparable<K>,V> {  
  
 **private** K **key**;  
 **private** V **value**;  
 **private** BinNode<K,V> **left**;  
 **private** BinNode<K,V> **right**;  
  
 **public** BinNode(K key,V value,BinNode<K,V> left,BinNode<K,V> right){  
 **this**.**key**=key;  
 **this**.**value**=value;  
 **this**.**left**=left;  
 **this**.**right**=right;  
 }  
 **public** BinNode(K key,V value){  
 **this**.**key**=key;  
 **this**.**value**=value;  
 }  
  
 **public** K getKey() {  
 **return key**;  
 }  
 **public void** setKey(K key) {  
 **this**.**key** = key;  
 }  
  
 **public** V getValue() {  
 **return value**;  
 }  
 **public void** setValue(V value) {  
 **this**.**value** = value;  
 }  
  
 **public** BinNode<K,V> getLeft() {  
 **return left**;  
 }  
 **public void** setLeft(BinNode<K,V> left) {  
 **this**.**left** = left;  
 }  
  
 **public** BinNode<K,V> getRight() {  
 **return right**;  
 }  
 **public void** setRight(BinNode<K,V> right) {  
 **this**.**right** = right;  
 }  
  
 **public boolean** biggerThan(K key){  
 **return this**.**key**.compareTo(key)>0;  
 }  
 **public boolean** smallerThan(K key){  
 **return this**.**key**.compareTo(key)<0;  
 }  
}

BSTTest.java

**import** java.io.File;  
**import** java.io.IOException;  
**import** java.io.PrintWriter;  
**import** java.util.Scanner;  
  
**public class** BSTTest {  
 **public static void** main(String[] args) **throws** IOException {  
 BST<String,String> test=**new** BST<>();  
 PrintWriter pw =**new** PrintWriter(System.***out***);  
 Scanner scanner=**new** Scanner(**new** File(**"BST\_testcases.txt"**));  
 String key,value;*//处理操作过程中用到的变量* **while** (scanner.hasNextLine()){  
 String line = scanner.nextLine();  
 **switch** (line.charAt(0)){*//判断是什么操作* **case '#'**->test.showStructure(pw);  
 **case '?'**->{  
 key= line.substring(3,line.length()-2);  
 value=test.search(key);  
 **if**(value==**null**)pw.write(**"search unsuccess ---"**+key+**"\n"**);*//搜索不成功* **else** pw.write(**"search success ---"**+key+**" "**+value+**"\n"**);*//搜索成功* }  
 **case '-'**->{  
 key=line.substring(3,line.length()-2);  
 value=test.remove(key);  
 **if**(value==**null**)pw.write(**"remove unsuccess ---"**+key+**"\n"**);*//搜索不成功* **else** pw.write(**"remove success ---"**+key+**" "**+value+**"\n"**);*//搜索成功* }  
 **default**->{  
 key=line.split(**" , "**)[0];  
 value=line.split(**" , "**)[1];  
 key=key.substring(3);  
 value=value.substring(1,value.length()-3);  
 **switch** (line.charAt(0)){  
 **case '+'**->test.insert(key,value);  
 **case '='**->{  
 **if**(test.update(key,value))pw.write(**"update success ---"**+key+**" "**+value+**"\n"**);  
 **else** pw.write(**"update unsuccess ---"**+key+**" "**+value+**"\n"**);  
 }  
 }  
 }  
 }  
 }  
 pw.flush();  
 }  
}

### 任务2

WordIndex.java

**import** java.io.File;  
**import** java.io.IOException;  
**import** java.io.PrintWriter;  
**import** java.util.Scanner;  
  
**public class** WordIndex {  
 **public static void** main(String[] args) **throws** IOException {  
 BST<String,ResizingQueue> wordBST=**new** BST<>();  
 Scanner scanner=**new** Scanner(**new** File(**"article.txt"**));  
 **for** (**int** line=1;scanner.hasNextLine();line++){*//依据article建立BST* String[] words=scanner.nextLine().split(**" "**);  
 **for**(String word : words){*//遍历单词* word=*getAvailable*(word);*//处理单词，排除无效的并对有标点的进行处理* **if**(word!=**null**){*//单词有效* ResizingQueue<Integer> queue=wordBST.search(word);  
 **if**(queue==**null**){*//单词在BST中不存在* queue=**new** ResizingQueue<>();  
 queue.enqueue(line);  
 wordBST.insert(word,queue);  
 }  
 **else** {*//单词已经存在* queue.enqueue(line);  
 }  
 }  
 }  
 }  
 PrintWriter pw=**new** PrintWriter(**"index\_result.txt"**);  
 wordBST.printInorder(pw);  
 pw.flush();  
 }  
 **private static** String getAvailable(String word){*//数据清洗* **if**(word.length()<=2)**return null**;  
 **for**(**int** i=0;i<word.length();i++){  
 **char** thisChar=word.charAt(i);  
 **if**(thisChar== 39||thisChar== 34||thisChar== **'-'**||thisChar==**'('**||thisChar==**')'**)**return null**;*//单引号，双引号，短横线，括号* **if**(thisChar>=**'1'**&&thisChar<=**'9'**)**return null**;*//数字* **if**(i==word.length()-1&&(thisChar==**','**||thisChar==**'.'**||thisChar==**'?'**||thisChar==**';'**||thisChar==**'!'**)){*//末尾是句号，逗号，问号，分号，感叹号* **return** word.substring(0,word.length()-1);  
 }  
 }  
 **return** word;  
 }  
}

ResizingQueue.java

**import** java.util.NoSuchElementException;  
  
**public class** ResizingQueue<T> {  
 **private int N**;  
 **private int length**;  
 **private int front**;  
 **private int rear**;  
 **private** T[] **arrayList**;  
 **public** ResizingQueue(){  
 **N** = 1;  
 **length** = **N**+1;  
 **front** = **rear** = 0;  
 **arrayList** = (T[])**new** Object[**length**];  
 }  
 **public void** enqueue(T element){  
 **if**(element==**null**)**throw new** IllegalArgumentException(**"Element is null."**);*//输入元素为空* **else**{  
 **if**(size()==**N**){*//若数组满，创建新数组* T[] arrayListNew = (T[])**new** Object[2\***N**+1];  
 copy(**arrayList**,arrayListNew);  
 **arrayList**=arrayListNew;  
 **front**=0;  
 **rear**=**N**;  
 **N**\*=2;  
 **length**=**N**+1;  
 }  
 **rear**=(**rear**+1)%**length**;  
 **arrayList**[**rear**]=element;*//入队* }  
 }  
 **public** T dequeue(){  
 **if**(size()==0)**throw new** NoSuchElementException(**"Queue is empty."**);*//队列空* **else**{  
 **front**=(**front**+1)%**length**;  
 T dequeueObject=**arrayList**[**front**];*//出队* **arrayList**[**front**]=**null**;  
 **if**(size()==**N**/4&&**N**>1){*//若数组内元素等于大小的四分之一，创建新数组* T[] arrayListNew = (T[])**new** Object[**N**/2+1];  
 copy(**arrayList**,arrayListNew);  
 **arrayList**=arrayListNew;  
 **front**=0;  
 **rear**=**N**/4;  
 **N**/=2;  
 **length**=**N**+1;  
 }  
 **return** dequeueObject;  
 }  
 }  
 **public int** size(){ **return** (**rear**-**front**+**length**)%**length**; }  
 **public** String toString(){  
 String str = **""**;  
 **while** (size()>0){  
 str+=dequeue()+**" "**;  
 }  
 **return** str;  
 }  
 **private void** copy(T[] arrayList,T[] arrayListNew){*//将原数组中的值复制进新数组* **for**(**int** i=1;i<size()+1;i++){  
 arrayListNew[i]=arrayList[(**front**+i)%**length**];  
 }  
 }  
}

### 任务3

TrieADT.java

**import** java.util.ArrayList;  
  
**public interface** TrieADT {  
 **void** insert(String key);  
 **void** remove(String key);  
 ArrayList<String> getWords(String key);  
}

DictionaryTrie.java

**import** java.io.File;  
**import** java.io.FileNotFoundException;  
**import** java.util.ArrayList;  
**import** java.util.Scanner;  
  
**public class** DictionaryTrie **implements** TrieADT {  
 DictionaryTrieNode **root**;  
 **public** DictionaryTrie() **throws** FileNotFoundException {  
 **root**=**null**;  
 Scanner scanner=**new** Scanner(**new** File(**"dictionary.txt"**));  
 **while** (scanner.hasNext()){  
 insert(scanner.next());  
 }  
 }  
  
 @Override  
 **public void** insert(String key) {  
 **root**=inserthelp(key,0,**root**);  
 }  
 **private** DictionaryTrieNode inserthelp(String key, **int** index, DictionaryTrieNode rt){*//从结点rt开始插入key的index之后位* **if**(rt==**null**){*//需新建结点* **if**(index==key.length()-1){*//关键字结束* **return new** DictionaryTrieNode(key.charAt(index),key);*//新建叶结点* }  
 **return new** DictionaryTrieNode(key.charAt(index),inserthelp(key,index+1,**null**));*//新建中间结点* }  
 **if**(rt.getLetter()==key.charAt(index)){*//当前结点是关键字的一部分* **if**(index==key.length()-1) {*//关键字结束* rt.setWord(key);*//在树中，更改当前结点* **return** rt;  
 }  
 rt.setNext(inserthelp(key,index+1,rt.getNext()));  
 }  
 **else** {*//当前结点不是关键字的一部分* rt.setRight(inserthelp(key,index,rt.getRight()));  
 }  
 **return** rt;  
 }  
  
 @Override  
 **public void** remove(String key) {  
 **root**=removehelp(key,0,**root**);  
 }  
 **private** DictionaryTrieNode removehelp(String key, **int** index, DictionaryTrieNode rt){  
 **if**(rt==**null**){*//关键字不在树中* **return** rt;  
 }  
 **if**(rt.getLetter()==key.charAt(index)){*//当前结点是关键字的一部分* **if**(index==key.length()-1) {*//关键字结束* **if**(rt.getNext()!=**null**){*//向后还有结点* rt.setWord(**null**);  
 **return** rt;  
 }  
 **else** {*//向后没有结点* **return** rt.getRight();  
 }  
 }  
 rt.setNext(removehelp(key,index+1,rt.getNext()));  
 **if**(rt.getNext()==**null**&&rt.getWord()==**null**)**return** rt.getRight();*//若该节点向后没有节点，且该节点没有关键字，该节点需被移除* **return** rt;  
 }  
 *//当前结点不是关键字的一部分* rt.setRight(removehelp(key,index,rt.getRight()));  
 **return** rt;  
 }  
  
 @Override  
 **public** ArrayList<String> getWords(String key) {*//以key为前缀串的所有关键字* DictionaryTrieNode rt=**root**;*//根据输入的关键字找到的结点* **for**(**int** i=0;i<key.length();){  
 **if**(rt==**null**){*//在当前树中找不到当前关键字* **return null**;  
 }  
 **else if**(rt.getLetter()==key.charAt(i)){*//当前结点是关键字的一部分* rt=rt.getNext();  
 i++;  
 }  
 **else** {*//当前结点不是关键字的一部分* rt=rt.getRight();  
 }  
 }  
 **return** getWordshelp(rt);  
 }  
 **private** ArrayList<String> getWordshelp(DictionaryTrieNode rt){*//获得结点rt右下级的所有关键字* ArrayList<String> words = **new** ArrayList<>();  
 **if**(rt==**null**)**return** words;*//遍历到底* **if**(rt.getWord()!=**null**)words.add(rt.getWord());*//此节点有关键字* words.addAll(getWordshelp(rt.getNext()));  
 words.addAll(getWordshelp(rt.getRight()));  
 **return** words;  
 }  
}

DictionaryTrieNode.java

**public class** DictionaryTrieNode {  
 **private char letter**;*//结点前的路径上的字母* **private** DictionaryTrieNode **next**;*//下级结点* **private** DictionaryTrieNode **right**;*//并列右结点* **private** String **word**;*//关键字* **public** DictionaryTrieNode(**char** letter, String word){  
 **this**.**letter**=letter;  
 **this**.**word**=word;  
 }  
 **public** DictionaryTrieNode(**char** letter, DictionaryTrieNode next){  
 **this**.**letter**=letter;  
 **this**.**next**=next;  
 }  
  
 **public** DictionaryTrieNode getNext() {  
 **return next**;  
 }  
 **public void** setNext(DictionaryTrieNode next) {  
 **this**.**next** = next;  
 }  
  
 **public** DictionaryTrieNode getRight() {  
 **return right**;  
 }  
 **public void** setRight(DictionaryTrieNode right) {  
 **this**.**right** = right;  
 }  
  
 **public char** getLetter() {  
 **return letter**;  
 }  
 **public void** setLetter(**char** letter) {  
 **this**.**letter** = letter;  
 }  
  
 **public** String getWord() {  
 **return word**;  
 }  
 **public void** setWord(String word) {  
 **this**.**word** = word;  
 }  
}

DictionaryTrieTest.java

**import** java.io.FileNotFoundException;  
**import** java.util.Scanner;  
  
**public class** DictionaryTrieTest {  
 **public static void** main(String[] args) **throws** FileNotFoundException {  
 DictionaryTrie test=**new** DictionaryTrie();  
 Scanner scanner = **new** Scanner(System.***in***);  
 **while** (**true**){  
 String operation=scanner.next();  
 **switch** (operation.charAt(0)){  
 **case '+'**-> test.insert(operation.substring(1));  
 **case '-'**-> test.remove(operation.substring(1));  
 **case '#'** -> {**return**;}  
 **default** -> System.***out***.println(test.getWords(operation));  
 }  
 }  
 }  
}

### 任务4

T9Trie.java

**import** java.io.File;  
**import** java.io.FileNotFoundException;  
**import** java.util.ArrayList;  
**import** java.util.Scanner;  
  
**public class** T9Trie **implements** TrieADT{  
 T9TrieNode **root**;  
 **public** T9Trie() **throws** FileNotFoundException {  
 **root**=**null**;  
 Scanner scanner=**new** Scanner(**new** File(**"dictionary.txt"**));  
 **while** (scanner.hasNext()){  
 insert(scanner.next());  
 }  
 }  
 @Override  
 **public void** insert(String key) {  
 **root**=inserthelp(key,0,**root**);  
 }  
 **private** T9TrieNode inserthelp(String key, **int** index, T9TrieNode rt){*//从结点rt开始插入key的index之后位* **if**(rt==**null**){*//需新建结点* **if**(index==key.length()-1){*//关键字结束* **return new** T9TrieNode(toNum(key.charAt(index)),key);*//新建叶结点* }  
 **return new** T9TrieNode(toNum(key.charAt(index)),inserthelp(key,index+1,**null**));*//新建中间结点* }  
 **if**(rt.getNum()==toNum(key.charAt(index))){*//当前结点是关键字的一部分* **if**(index==key.length()-1) {*//关键字结束* rt.addWord(key);*//在树中，更改当前结点* **return** rt;  
 }  
 rt.setNext(inserthelp(key,index+1,rt.getNext()));  
 }  
 **else** {*//当前结点不是关键字的一部分* rt.setRight(inserthelp(key,index,rt.getRight()));  
 }  
 **return** rt;  
 }  
  
  
 @Override  
 **public void** remove(String key) {  
 **root**=removehelp(key,0,**root**);  
 }  
 **private** T9TrieNode removehelp(String key, **int** index, T9TrieNode rt){  
 **if**(rt==**null**){*//关键字不在树中* **return** rt;  
 }  
 **if**(rt.getNum()==toNum(key.charAt(index))){*//当前结点是关键字的一部分* **if**(index==key.length()-1) {*//关键字结束* **if**(rt.getNext()!=**null**||rt.getWords().size()>1){*//向后还有结点，或此结点还有别的关键字* rt.deleteWord(key);  
 **return** rt;  
 }  
 **else** {*//向后没有结点，且该节点只有一个关键字，移除它* **return** rt.getRight();  
 }  
 }  
 rt.setNext(removehelp(key,index+1,rt.getNext()));  
 **if**(rt.getNext()==**null**&&rt.getWords().isEmpty())**return** rt.getRight();*//若该节点向后没有节点，且该节点没有关键字，该节点需被移除* **return** rt;  
 }  
 *//当前结点不是关键字的一部分* rt.setRight(removehelp(key,index,rt.getRight()));  
 **return** rt;  
 }  
  
 @Override  
 **public** ArrayList<String> getWords(String key) {*//获得key对应的数个单词* T9TrieNode rt=**root**;*//根据输入的关键字找到的结点* **for**(**int** i=0;i<key.length();){  
 **if**(rt==**null**){*//在当前树中找不到当前关键字* **return null**;  
 }  
 **else if**(rt.getNum()==key.charAt(i)-48){*//当前结点是关键字的一部分* **if**(i<key.length()-1) rt=rt.getNext();*//若不在最后一层，进入下层* i++;  
 }  
 **else** {*//当前结点不是关键字的一部分* rt=rt.getRight();  
 }  
 }  
 **return** rt.getWords();  
 }  
  
 **private int** toNum(**char** s){*//获得字符对应的键盘数字* **if**(s>=**'a'**&&s<=**'c'**)**return** 2;*//ABC* **if**(s>=**'d'**&&s<=**'f'**)**return** 3;*//DEF* **if**(s>=**'g'**&&s<=**'i'**)**return** 4;*//GHI* **if**(s>=**'j'**&&s<=**'l'**)**return** 5;*//JKL* **if**(s>=**'m'**&&s<=**'o'**)**return** 6;*//MNO* **if**(s>=**'p'**&&s<=**'s'**)**return** 7;*//PQRS* **if**(s>=**'t'**&&s<=**'v'**)**return** 8;*//TUV* **if**(s>=**'w'**&&s<=**'z'**)**return** 9;*//WXYZ* **return** 0;  
 }  
}

T9TrieNode.java

**import** java.util.ArrayList;  
  
**public class** T9TrieNode {  
 **private int num**;*//结点前的路径上的数字* **private** T9TrieNode **next**;*//下级结点* **private** T9TrieNode **right**;*//并列右结点* **private** ArrayList<String> **words**;*//关键字* **public** T9TrieNode(**int** num, String word){  
 **this**.**num**=num;  
 **this**.**words**=**new** ArrayList<>();  
 **words**.add(word);  
 }  
 **public** T9TrieNode(**int** num, T9TrieNode next){  
 **this**.**num**=num;  
 **this**.**words**=**new** ArrayList<>();  
 **this**.**next**=next;  
 }  
  
 **public** T9TrieNode getNext() {  
 **return next**;  
 }  
 **public void** setNext(T9TrieNode next) {  
 **this**.**next** = next;  
 }  
  
 **public** T9TrieNode getRight() {  
 **return right**;  
 }  
 **public void** setRight(T9TrieNode right) {  
 **this**.**right** = right;  
 }  
  
 **public int** getNum() {  
 **return num**;  
 }  
 **public void** setNum(**int** num) {  
 **this**.**num** = num;  
 }  
  
 **public** ArrayList<String> getWords() {  
 **return words**;  
 }  
 **public void** addWord(String word) {  
 **this**.**words**.add(word);  
 }  
 **public void** deleteWord(String word) {  
 **this**.**words**.remove(word);  
 }  
}

T9TrieTest.java

**import** java.io.FileNotFoundException;  
**import** java.util.ArrayList;  
**import** java.util.Scanner;  
  
**public class** T9TrieTest {  
 **public static void** main(String[] args) **throws** FileNotFoundException {  
 T9Trie test= **new** T9Trie();  
 Scanner scanner = **new** Scanner(System.***in***);  
 System.***out***.println(**"Enter “exit” to quit."**);  
 **for**(ArrayList<String> words=**null**;;){  
 System.***out***.print(**"Enter Key Sequence (or “#” for next word):\n>"**);  
 String operation=scanner.next();  
 **int** index=0;  
 **if** (operation.equals(**"exit"**))**break**;*//退出* **else if** (operation.equals(**"#"**))index++;  
 **else** words=test.getWords(operation);;  
 **if**(words==**null**|| words.size()==0){  
 System.***out***.println(**"Not found in current dictionary."**);  
 }  
 **else if** (words.size()<=index) {  
 System.***out***.println(**"There are no more T9 words."**);  
 } **else** {  
 System.***out***.println(**"\'"**+words.get(index)+**"\'"**);  
 }  
 }  
 }  
}