二叉树的应用作业报告

第一次

姓名	史佳鑫
班级	软件 2302 班
学号	2236115195
电话	17342941319
Email	3168609153@qq. com
日期	20241217

目录

实验 1

1、 题目 实现 BST 数据结构

2、 数据设计

BSTNode 类

属性:

K key：存储节点的键。

V value：存储节点的值。

BSTNode<K, V> left：指向左子节点的引用。

BSTNode<K, V> right：指向右子节点的引用。

构造方法:

BSTNode(K key, V value)：创建一个新的节点，初始化键和值，左右子节点设置为 null。

BinarySearchTree 类

属性:

private BSTNode<K, V> root：根节点的引用。

private int size：树中节点的数量。

构造方法:

BinarySearchTree()：初始化根节点为 null，节点数量为 0。

方法:

insert(K key, V value)：公开方法，插入键值对，如果键或值为 null，则不执行操作。

insertRecursive(BSTNode<K, V> node, K key, V value)：私有递归方法，实际执行插入操作。

remove(K key)：公开方法，删除键对应的节点，如果键为 null，则打印失败信息并返回 null。

removeRecursive(BSTNode<K, V> node, K key, V[] removedValue)：私有递归方法，实际执行删除操作。

getMin(BSTNode<K, V> node)：私有方法，找到子树中的最小节点。

search(K key)：公开方法，搜索键对应的值，如果键为 null 或未找到，则打印失败信息并返回 null。

searchRecursive(BSTNode<K, V> node, K key)：私有递归方法，实际执行搜索操作。

update(K key, V value)：公开方法，更新键对应的值，如果键或值为 null，则打印失败信息并返回 false。

updateRecursive(BSTNode<K, V> node, K key, V value)：私有递归方法，实际执行更新操作。

isEmpty()：公开方法，返回树是否为空。

clear()：公开方法，清空树，根节点设置为 null，节点数量设置为 0。

showStructure(PrintWriter pw)：公开方法，显示树的结构，包括节点数量和树的高度。

getHeight(BSTNode<K, V> node)：私有方法，计算节点的高度。

printInorder(PrintWriter pw)：公开方法，中序遍历树并打印节点。

printInOrderRecursive(BSTNode<K, V> node, PrintWriter pw)：私有递归方法，实际执行中序遍历和打印操作。

3、 算法设计

1. 插入（Insert）

算法描述：

公开方法 `insert(K key, V value)` 接收一个键和值，首先检查键和值是否为 null，如果是，则不执行任何操作。

调用私有递归方法 `insertRecursive(BSTNode<K, V> node, K key, V value)` 来找到正确的插入位置。

在 `insertRecursive` 方法中，比较键与当前节点的键：

如果当前节点为 null，创建一个新节点并返回。

如果键小于当前节点的键，递归地在左子树插入。

如果键大于当前节点的键，递归地在右子树插入。

如果键等于当前节点的键，更新当前节点的值（可选，根据具体需求决定是否允许键重复）。

2. 删除（Remove）

算法描述：

公开方法 `remove(K key)` 接收一个键，首先检查键是否为 null，如果是，则打印失败信息并返回 null。

调用私有递归方法 `removeRecursive(BSTNode<K, V> node, K key, V[] removedValue)` 来找到并删除节点。

在 `removeRecursive` 方法中，比较键与当前节点的键：

如果当前节点为 null，返回 null。

如果键小于当前节点的键，递归地在左子树删除。

如果键大于当前节点的键，递归地在右子树删除。

如果键等于当前节点的键，保存当前节点的值到 `removedValue` 中，并减少节点数量：

如果节点没有左子节点，返回右子节点。

如果节点没有右子节点，返回左子节点。

如果节点有两个子节点，找到右子树中的最小节点，用其键值替换当前节点的键值，然后递归地删除最小节点。

3. 搜索（Search）

算法描述：

公开方法 `search(K key)` 接收一个键，首先检查键是否为 null，如果是，则打印失败信息并返回 null。

调用私有递归方法 `searchRecursive(BSTNode<K, V> node, K key)` 来找到键对应的值。

在 `searchRecursive` 方法中，比较键与当前节点的键：

如果当前节点为 null，返回 null。

如果键小于当前节点的键，递归地在左子树搜索。

如果键大于当前节点的键，递归地在右子树搜索。

如果键等于当前节点的键，返回当前节点的值。

4. 更新（Update）

算法描述：

公开方法 `update(K key, V value)` 接收一个键和一个新值，首先检查键和值是否为 null，如果是，则打印失败信息并返回 false。

调用私有递归方法 `updateRecursive(BSTNode<K, V> node, K key, V value)` 来更新键对应的值。

在 `updateRecursive` 方法中，比较键与当前节点的键:

如果当前节点为 null，返回 false。

如果键小于当前节点的键，递归地在左子树更新。

如果键大于当前节点的键，递归地在右子树更新。

如果键等于当前节点的键，更新当前节点的值为新值并返回 true。

5. 显示树结构（Show Structure）

算法描述:

公开方法 `showStructure(PrintWriter pw)` 接收一个 `PrintWriter` 对象，用于输出树的结构。

调用私有方法 `getHeight(BSTNode<K, V> node)` 来计算树的高度。

输出树的节点数量和高度。

6. 中序遍历（Inorder Traversal）

算法描述:

公开方法 `printInorder(PrintWriter pw)` 接收一个 `PrintWriter` 对象，用于输出中序遍历的结果。

调用私有递归方法 `printInOrderRecursive(BSTNode<K, V> node, PrintWriter pw)` 来中序遍历树并打印每个节点的键值对。

在 `printInOrderRecursive` 方法中，递归地中序遍历左子树，打印当前节点，然后递归地中序遍历右子树。

这些算法设计确保了二叉搜索树的基本操作能够正确执行，并且通过递归的方式，我们可以有效地在树中定位节点，执行插入、删除、搜索和更新操作。中序遍历和显示树结构的操作则提供了对树的进一步理解和可视化。

4、 主干代码说明

```java
import java.io.IOException;
import java.io.PrintWriter;
109 个用法  1 个实现
public interface BST<K extends Comparable<K>, V> {

    1 个用法  1 个实现
    public void insert(K key, V value);
    1 个用法  1 个实现
    public V remove(K key);
    1 个用法  1 个实现
    public V search(K key);
    1 个用法  1 个实现
    public boolean update(K key, V value);
    0 个用法  1 个实现
    public boolean isEmpty();
    0 个用法  1 个实现
    public void clear();
    1 个用法  1 个实现
    public void showStructure(PrintWriter pw) throws IOException;
    0 个用法  1 个实现
    public void printInorder(PrintWriter pw) throws IOException;
}
```

```java
import java.io.IOException;
import java.io.PrintWriter;

// BST节点类
14 个用法
class BSTNode<K extends Comparable<K>, V> {
    9 个用法
    K key;
    8 个用法
    V value;
    13 个用法
    BSTNode<K, V> left, right;

    1 个用法
    BSTNode(K key, V value) {
        this.key = key;
        this.value = value;
        this.left = this.right = null;
    }
}

// BST实现类
3 个用法
public class BinarySearchTree<K extends Comparable<K>, V> implements BST<K, V> {
    10 个用法
    private BSTNode<K, V> root;
    6 个用法
    private int size;

    1 个用法
    public BinarySearchTree() {
        this.root = null;
        this.size = 0;
    }

    1 个用法
    @Override
    public void insert(K key, V value) {
        if (key == null || value == null) return;
        root = insertRecursive(root, key, value);
```

```
3 个用法
private BSTNode<K, V> insertRecursive(BSTNode<K, V> node, K key, V value) {
    if (node == null) {
        size++;
        return new BSTNode<>(key, value);
    }
    int cmp = key.compareTo(node.key);
    if (cmp < 0) node.left = insertRecursive(node.left, key, value);
    else if (cmp > 0) node.right = insertRecursive(node.right, key, value);
    else node.value = value; // 更新值
    return node;
}

1 个用法
@Override
public V remove(K key) {
    if (key == null) {
        System.out.println("remove unsuccess ---" + key);
        return null;
    }
    V[] removedValue = (V[]) new Object[1];
    root = removeRecursive(root, key, removedValue);
    if (removedValue[0] == null) {
        System.out.println("remove unsuccess ---" + key);
    } else {
        System.out.println("remove success ---" + key + " " + removedValue[0]);
    }
    return removedValue[0];
}

4 个用法
private BSTNode<K, V> removeRecursive(BSTNode<K, V> node, K key, V[] removedValue) {
    if (node == null) return null;

    int cmp = key.compareTo(node.key);
    if (cmp < 0) node.left = removeRecursive(node.left, key, removedValue);
    else if (cmp > 0) node.right = removeRecursive(node.right, key, removedValue);
    else { // 找到节点
        removedValue[0] = node.value;
        size--;
        if (node.left == null) return node.right;
```

```
            size--;
            if (node.left == null) return node.right;
            if (node.right == null) return node.left;

            BSTNode<K, V> minNode = getMin(node.right);
            node.key = minNode.key;
            node.value = minNode.value;
            node.right = removeRecursive(node.right, minNode.key, removedValue: null);
        }
        return node;
    }

    1 个用法
    private BSTNode<K, V> getMin(BSTNode<K, V> node) {
        while (node.left != null) node = node.left;
        return node;
    }

    1 个用法
    @Override
    public V search(K key) {
        V result = searchRecursive(root, key);
        if (result == null) {
            System.out.println("search unsuccess ---" + key);
        } else {
            System.out.println("search success ---" + key + " " + result);
        }
        return result;
    }
    3 个用法
    private V searchRecursive(BSTNode<K, V> node, K key) {
        if (node == null) return null;

        int cmp = key.compareTo(node.key);
        if (cmp < 0) return searchRecursive(node.left, key);
        else if (cmp > 0) return searchRecursive(node.right, key);
        else return node.value;
    }

    1 个用法
```

```
    @Override
    public boolean update(K key, V value) {
        if (key == null || value == null) {
            System.out.println("update unsuccess ---" + key);
            return false;
        }
        boolean result = updateRecursive(root, key, value);
        if (result) {
            System.out.println("update success ---" + key + " " + value);
        } else {
            System.out.println("update unsuccess ---" + key);
        }
        return result;
    }
    3 个用法
    private boolean updateRecursive(BSTNode<K, V> node, K key, V value) {
        if (node == null) return false;
        int cmp = key.compareTo(node.key);
        if (cmp < 0) return updateRecursive(node.left, key, value);
        else if (cmp > 0) return updateRecursive(node.right, key, value);
        else {
            node.value = value;
            return true;
        }
    }

    0 个用法
    @Override
    public boolean isEmpty() {
        return size == 0;
    }

    0 个用法
    @Override
    public void clear() {
        root = null;
        size = 0;
    }
```

```
    0 个用法
    @Override
    public void clear() {
        root = null;
        size = 0;
    }

    1 个用法
    @Override
    public void showStructure(PrintWriter pw) throws IOException {
        if (pw == null) throw new IOException("PrintWriter is null");
        pw.println("-----------------------------");
        pw.println("There are " + size + " nodes in this BST.");
        pw.println("The height of this BST is " + getHeight(root) + ".");
        pw.println("-----------------------------");
    }

    3 个用法
    private int getHeight(BSTNode<K, V> node) {
        if (node == null) return 0;
        return 1 + Math.max(getHeight(node.left), getHeight(node.right));
    }

    0 个用法
    @Override
    public void printInorder(PrintWriter pw) throws IOException {
        if (pw == null) throw new IOException("PrintWriter is null");
        printInOrderRecursive(root, pw);
    }

    3 个用法
    private void printInOrderRecursive(BSTNode<K, V> node, PrintWriter pw) throws IOException {
        if (node == null) return;
        printInOrderRecursive(node.left, pw);
        pw.println(node.key + " --- <" + node.value + ">");
        printInOrderRecursive(node.right, pw);
    }
}
```

```java
import java.io.*;
import java.util.Scanner;
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;
public class BSTTest {
    public static void main(String[] args) throws IOException {
        BinarySearchTree<String, String> bst = new BinarySearchTree<>();
        File inputFile = new File( pathname: "BST_testcases.txt");
        File outputFile = new File( pathname: "BST_result.txt");

        // Set up input and output
        Scanner scanner = new Scanner(inputFile);
        PrintWriter writer = new PrintWriter(outputFile);

        while (scanner.hasNextLine()) {
            String command = scanner.nextLine();
            processCommand(command, bst, writer);
        }

        writer.close();
        scanner.close();
        System.out.println("Testing complete. Results saved to BST_result.txt");
    }

    1 个用法
    private static void processCommand(String command, BinarySearchTree<String, String> bst, PrintWriter writer) {
        try {
            // 创建输出捕获工具
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            PrintStream ps = new PrintStream(baos);
            PrintStream old = System.out;

            if (command.startsWith("+")) { // Insert
                String[] parts = parseInsert(command);
                bst.insert(parts[0], parts[1]);
            } else if (command.startsWith("-")) { // Remove
                String key = parseKey(command);
                // 重定向输出
                System.setOut(ps);
                bst.remove(key);
                // 恢复输出并写入文件
```

```
                bst.remove(key);
                // 恢复输出并写入文件
                System.out.flush();
                System.setOut(old);
                writer.println(baos.toString().trim());

            } else if (command.startsWith("?")) { // Search
                String key = parseKey(command);
                // 重定向输出
                System.setOut(ps);
                bst.search(key);
                // 恢复输出并写入文件
                System.out.flush();
                System.setOut(old);
                writer.println(baos.toString().trim());

            } else if (command.startsWith("=")) { // Update
                String[] parts = parseInsert(command);
                // 重定向输出
                System.setOut(ps);
                bst.update(parts[0], parts[1]);
                // 恢复输出并写入文件
                System.out.flush();
                System.setOut(old);
                writer.println(baos.toString().trim());

            } else if (command.startsWith("#")) { // Show structure
                bst.showStructure(writer);
            } else {
                writer.println("Invalid command: " + command);
            }
        } catch (Exception e) {
        }
    }
    2 个用法
    private static String[] parseInsert(String command) {
        int start = command.indexOf('(') + 1;
        int end = command.indexOf(')');
        String[] parts = command.substring(start, end).split( regex: ",");
        parts[0] = parts[0].trim();
```

```
        parts[0] = parts[0].trim();
        parts[1] = parts[1].trim();
        return parts;
    }

    2 个用法
    private static String parseKey(String command) {
        int start = command.indexOf('(') + 1;
        int end = command.indexOf(')');
        return command.substring(start, end).trim();
    }
}
```

5、 运行结果展示（具体结果见附录）

第一张为输出结果，第二张为提供的答案，可以看出结果相同。

BST_result.txt

BST_result.txt

文件 编辑 查看

```
----------------------------
There are 1 nodes in this BST.
The height of this BST is 1.
----------------------------
remove unsuccess ---vertebrate
update success ---overlap "overlap"
remove unsuccess ---affection
search success ---persevere "v．坚忍"
----------------------------
There are 11 nodes in this BST.
The height of this BST is 5.
----------------------------
update success ---pretentiousness "pretentiousness"
----------------------------
There are 13 nodes in this BST.
The height of this BST is 5.
----------------------------
remove success ---sill "n．基石，门槛，窗台"
search unsuccess ---listless
search success ---gollop "v．n．大口吞咽"
update success ---sagacious "sagacious"
remove unsuccess ---propitiate
update success ---ideology "ideology"
search success ---puerile "adj．幼稚的，儿童的"
update success ---laggard "laggard"
search unsuccess ---penalize
search success ---deify "v．奉为神，崇拜"
search unsuccess ---midget
search success ---mutton "n．羊肉"
update success ---conflagration "conflagration"
search unsuccess ---morass
search success ---breach "v．毁坏，泄密n．缺口"
search success ---sagacious "sagacious"
----------------------------
There are 33 nodes in this BST.
The height of this BST is 8.
----------------------------
search unsuccess ---guarded
search success ---sagacious "sagacious"
remove unsuccess ---egalitarian
update success ---incendiary "incendiary"
search success ---nocturnal "adj．夜晚的，夜间发生的"
update success ---deify "deify"
remove success ---solemnity "n．庄严，肃穆"
search unsuccess ---tongs
search success ---squash "v．压碎，挤压，n．南瓜"
update success ---abate "abate"
----------------------------
There are 46 nodes in this BST.
```

BST_result.txt

文件 编辑 查看

```
----------------------------
There are 1 nodes in this BST.
The height of this BST is 1.
----------------------------
remove unsuccess ---vertebrate
update success ---overlap overlap
remove unsuccess ---affection
search success ---persevere v. 坚忍
----------------------------
There are 11 nodes in this BST.
The height of this BST is 5.
----------------------------
update success ---pretentiousness pretentiousness
----------------------------
There are 13 nodes in this BST.
The height of this BST is 5.
----------------------------
remove success ---sill n. 基石，门槛，窗台
search unsuccess ---listless
search success ---gollop v. n. 大口吞咽
update success ---sagacious sagacious
remove unsuccess ---propitiate
update success ---ideology ideology
search success ---puerile adj. 幼稚的，儿童的
update success ---laggard laggard
search unsuccess ---penalize
search success ---deify v. 奉为神，崇拜
search unsuccess ---midget
search success ---mutton n. 羊肉
update success ---conflagration conflagration
search unsuccess ---morass
search success ---breach v. 毁坏，泄密n. 缺口
search success ---sagacious sagacious
remove success ---miscreant n. 恶棍，歹徒
----------------------------
There are 33 nodes in this BST.
The height of this BST is 8.
----------------------------
search unsuccess ---guarded
search success ---sagacious sagacious
remove unsuccess ---egalitarian
update success ---incendiary incendiary
search success ---nocturnal adj. 夜晚的，夜间发生的
update success ---deify deify
remove success ---solemnity n. 庄严，肃穆
search unsuccess ---tongs
search success ---squash v. 压碎，挤压，n. 南瓜
update success ---abate abate
```

6、 总结和收获

通过实现 BST 数据结构，我们学习了二叉搜索树的基本操作和递归算法的应用。我们了解到 BST 在插入、删除和搜索操作上的时间复杂度为 O(log n)，这使得 BST 成为处理有序数据的有效数据结构。

实验 2

1、 题目 使用 BST 为文稿建立单词索引表

2、 数据设计

1. LinkedList 类

目的：用于存储每个单词在文稿中出现的所有行号。

属性：

head：指向链表的第一个节点。

方法：

add(int data)：向链表中添加一个新的行号。如果行号已存在于链表中，则不重复添加。

toString()：将链表中的所有行号转换为一个字符串，行号之间用空格分隔。

2. BSTNode 类

目的：表示二叉搜索树中的一个节点，存储单词及其相关信息。

泛型参数 K：允许节点存储任何可比较的键类型，这里特指单词（String 类型）。

属性：

key：节点存储的键，即单词。

lines：一个 LinkedList 对象，存储单词出现的行号。

left、right：指向左子节点和右子节点的引用。

构造函数：

BSTNode(K key, int line)：创建一个新的 BSTNode，初始化键、行号列表，并设置左右子节点为 null。

3. BinarySearchTree 类

目的：实现一个二叉搜索树，用于存储单词和它们出现的行号。

泛型参数 K：允许树存储任何可比较的键类型。

属性：

root：指向二叉搜索树的根节点。

方法：

insert(K key, int line)：对外提供的插入接口，用于将单词和行号插入到 BST 中。

insertRecursive(BSTNode<K> node, K key, int line)：私有递归方法，实现单词和行号的插入逻辑。

printInOrder(PrintWriter pw)：对外提供的中序遍历接口，用于将 BST 中的单词和行号写入文件。

printInOrderRecursive(BSTNode<K> node, PrintWriter pw)：私有递归方法，实现中序遍历并将结果写入文件。

3、 算法设计

1. 单词分割和预处理

输入读取：程序从文件中逐行读取文本。

单词分割：使用正则表达式 \\W+ 将每行文本分割成单词，这个正则表达式匹配任何非单词字符，从而将文本分割成单词数组。

单词预处理：将每个单词转换为小写，以确保索引的不区分大小写。

2. 插入单词和行号到 BST

BST 插入算法：

递归插入：insert 方法是对外的接口，它调用私有的递归方法 insertRecursive 来实际执行插入操作。

比较和定位：在 insertRecursive 方法中，通过比较待插入单词与当前节点单词，决定是向左子树递归还是向右子树递归。

处理重复单词：如果待插入单词与当前节点单词相同，则将当前行号添加到该单词对应的 LinkedList 中，而不是创建新节点。

创建新节点：如果待插入单词小于当前节点单词，并且左子节点为空，则在左子节点创建新节点；如果大于当前节点单词，并且右子节点为空，则在右子节点创建新节点。

3. 中序遍历 BST 并输出索引结果

中序遍历算法：

递归遍历：printInOrder 方法是对外的接口，它调用私有的递归方法 printInOrderRecursive 来实际执行中序遍历。

访问顺序：在 printInOrderRecursive 方法中，先递归遍历左子树，然后访问当前节点，最后递归遍历右子树。

输出格式：对于每个节点，按照格式 [单词---<行号列表>] 将单词和对应的行号列表写入文件。

4、 主干代码说明

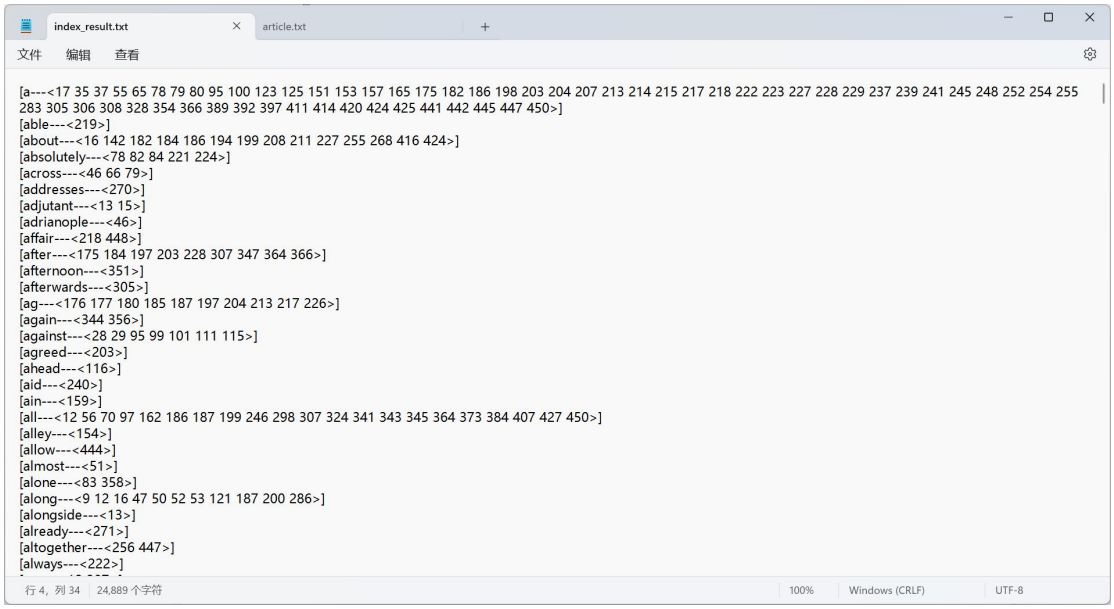
```java
import ...

// 自定义LinkedList实现，用于存储行号
2 个用法
class LinkedList {
    4 个用法
    private Node head;

    6 个用法
    private class Node {
        4 个用法
        int data;
        5 个用法
        Node next;

        2 个用法
        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }

    // 添加行号，避免重复添加同一行号
    2 个用法
    public void add(int data) {
        if (head == null) {
            head = new Node(data);
            return;
        }
        Node current = head;
        while (current.next != null) {
            if (current.data == data) return; // 避免重复添加
            current = current.next;
        }
        if (current.data != data) current.next = new Node(data);
    }

    @Override
```

```
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        Node current = head;
        while (current != null) {
            sb.append(current.data).append(" ");
            current = current.next;
        }
        return sb.toString().trim();
    }
}

// BST节点类
6 个用法
class BSTNode<K extends Comparable<K>> {
    3 个用法
    K key;
    4 个用法
    LinkedList lines; // 存储出现的行号
    4 个用法
    BSTNode<K> left, right;

    1 个用法
    BSTNode(K key, int line) {
        this.key = key;
        this.lines = new LinkedList();
        this.lines.add(line);
        this.left = this.right = null;
    }
}

// BST实现类
2 个用法
class BinarySearchTree<K extends Comparable<K>> {
    3 个用法
    private BSTNode<K> root;
```

```
    1 个用法
    public void insert(K key, int line) { root = insertRecursive(root, key, line); }

    3 个用法
    private BSTNode<K> insertRecursive(BSTNode<K> node, K key, int line) {
        if (node == null) return new BSTNode<>(key, line);

        int cmp = key.compareTo(node.key);
        if (cmp < 0) {
            node.left = insertRecursive(node.left, key, line);
        } else if (cmp > 0) {
            node.right = insertRecursive(node.right, key, line);
        } else {
            node.lines.add(line); // 单词已存在，更新行号
        }
        return node;
    }

    // 中序遍历，写入文件
    1 个用法
    public void printInOrder(PrintWriter pw) { printInOrderRecursive(root, pw); }

    3 个用法
    private void printInOrderRecursive(BSTNode<K> node, PrintWriter pw) {
        if (node == null) return;
        printInOrderRecursive(node.left, pw);
        pw.println("[" + node.key + "---<" + node.lines + ">]");
        printInOrderRecursive(node.right, pw);
    }
}

// 主类
public class IndexBuilder {
    public static void main(String[] args) {
        String inputFile = "article.txt";
        String outputFile = "index_result.txt";

        // 初始化BST
```

```java
public class IndexBuilder {
    public static void main(String[] args) {
        String inputFile = "article.txt";
        String outputFile = "index_result.txt";

        // 初始化BST
        BinarySearchTree<String> bst = new BinarySearchTree<>();

        // 读取文件并构建索引
        try (BufferedReader br = new BufferedReader(new FileReader(inputFile))) {
            String line;
            int lineNumber = 0;

            while ((line = br.readLine()) != null) {
                lineNumber++;
                String[] words = line.split( regex: "\\W+"); // 使用非字母字符分割
                for (String word : words) {
                    if (!word.isEmpty()) {
                        bst.insert(word.toLowerCase(), lineNumber); // 单词转小写
                    }
                }
            }
        } catch (IOException e) {
            System.err.println("Error reading file: " + e.getMessage());
        }

        // 输出索引结果到文件
        try (PrintWriter pw = new PrintWriter(new FileWriter(outputFile))) {
            bst.printInOrder(pw);
            System.out.println("Index successfully written to " + outputFile);
        } catch (IOException e) {
            System.err.println("Error writing file: " + e.getMessage());
        }
    }
}
```

5、 运行结构展示（具体结果见附录）



6、 总结和收获

这个程序展示了如何使用 BST 来构建一个简单的单词索引系统。学习到：BST 的基本操作，如插入和遍历。文件读写操作。字符串处理技巧，如使用正则表达式分割单词。

实验 3

1、 题目 学会使用堆

子任务一

2、 数据设计

2.1 堆的表示

数组：使用数组来存储堆中的元素。为了方便实现父节点和子节点之间的索引关系，数组的第一个元素（索引 0）不使用，从索引 1 开始存储堆的元素。这样，对于任意一个索引 i 的元素，其父节点的索引是 i/2，左子节点的索引是 2*i，右子节点的索引是 2*i + 1。

2.2 堆的基本属性

DEFAULT_CAPACITY：堆的默认容量，用于初始化堆数组的大小。这个值可以根据实际需要进行调整，以适应不同的使用场景。

currentSize：表示当前堆中元素的数量。由于数组是从索引 1 开始存储数据的，所以 currentSize 的值将始终小于数组的长度。

2.3 堆的操作

insert(int number)：插入操作，将新元素添加到堆中，并保持堆的性质。新元素被添加到数组的末尾（array[currentSize+1]），然后通过上浮操作（percolate up）调整堆。

deleteMin()：删除最小元素操作，移除堆顶元素（即数组的第一个元素），将堆的最后一个元素移动到堆顶，然后通过下沉操作（sift down）调整堆。

findMin()：获取最小元素操作，由于是最小堆，堆顶元素即为最小元素，直接返回数组的第一个元素（array[1]）。

2.4 辅助方法

isEmpty()：检查堆是否为空，即 currentSize 是否为 0。

isFull()：检查堆是否已满，即 currentSize 是否等于数组的长度减 1。

isLeaf(int i)：检查索引 i 的元素是否为叶子节点，即索引 i 大于 currentSize/2。

buildHeap()：构建最小堆，从最后一个非叶子节点开始，对每个节点进行下沉操作，以确保堆的性质。

getLeft(int i)：获取索引 i 的左子节点的索引。

getRight(int i)：获取索引 i 的右子节点的索引。

getParent(int i)：获取索引 i 的父节点的索引。

swap(int[] array, int x, int y)：交换数组中两个元素的位置。

3、 算法设计

插入操作（insert）：将新元素插入到数组的末尾，然后通过比较和交换操作，将新元素上浮到合适的位置，以保持堆的性质。

删除最小元素操作（deleteMin）：移除堆顶元素（最小元素），将数组末尾的元素移动到堆顶，然后通过下沉操作（siftDown）调整堆，以保持堆的性质。

获取最小元素操作（findMin）：直接返回堆顶元素，即数组的第一个元素。

4、 主干代码说明

```java
public class MinHeap {
    1 个用法
    private static final int DEFAULT_CAPACITY = 10; // 默认大小
    14 个用法
    private int currentSize; // 当前堆的大小
    17 个用法
    private int[] array; // 堆数组

    0 个用法
    public MinHeap() {
        this.array = new int[DEFAULT_CAPACITY + 1];
        currentSize = 0;
    }

    0 个用法
    public MinHeap(int size) {
        this.array = new int[size + 1];
        currentSize = 0;
    }

    1 个用法
    public MinHeap(int[] array) {
        this.array = new int[array.length + 1];
        for (int i = 0; i < array.length; i++) {
            this.array[i + 1] = array[i];
        } // 从1开始计算
        currentSize = array.length;
        buildHeap();
    }

    /**
     * 往堆中插入元素
     *
     * @param number 元素值
     */
    0 个用法
    public void insert(int number) {
        try {
```

```
    0 个用法
    public void insert(int number) {
        try {
            if (isFull()) {
                throw new Exception("Array is full!");
            }
            int temp = ++currentSize;
            array[temp] = number; // 将number放在数组最后
            while ((temp != 1) && (array[temp] < array[getParent(temp)])) {
                swap(array, temp, getParent(temp)); // 比父结点值小则与其交换
                temp = getParent(temp);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    2 个用法
    public int findMin() {
        return array[1];
    }

    /**
     * 删除堆顶元素
     */
    2 个用法
    public void deleteMin() {
        try {
            if (isEmpty()) {
                throw new Exception("Array is empty!");
            } else {
                swap(array, x: 1, currentSize--); // 将堆顶元素放到最后
                if (currentSize != 0) siftDown( pos: 1);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
     * 大的值下沉
     *
     * @param pos 当前位置
     */
    2 个用法
    private void siftDown(int pos) {
        try {
            if (pos < 0 || pos > currentSize) {
                throw new Exception("Illegal position!");
            }
            while (!isLeaf(pos)) {
                int j = getLeft(pos);
                if ((j < currentSize) && (array[j] > array[j + 1])) j++; //
                if (array[pos] <= array[j]) return; // 如果当前值已经比子树值小,
                swap(array, pos, j);
                pos = j;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    0 个用法
    public void print() {
        for (int i = 1; i <= currentSize; i++) {
            System.out.print(array[i] + " ");
        }
    }

    0 个用法
    public void heapSort() {
        while (currentSize != 0) {
            System.out.print(findMin() + " ");
            deleteMin();
        }
    }

    1 个用法
```

```
private boolean isEmpty() { return currentSize == 0; }

1 个用法
private boolean isFull() { return currentSize == array.length - 1; }

1 个用法
private boolean isLeaf(int i) { return i > currentSize / 2; }

1 个用法
private void buildHeap() {
    for (int i = currentSize / 2; i > 0; i--) {
        siftDown(i); // 对每个非叶子结点进行下沉操作
    }
}

1 个用法
private int getLeft(int i) {
    return 2 * i;
}

0 个用法
private int getRight(int i) {
    return 2 * i + 1;
}

3 个用法
private int getParent(int i) {
    return i / 2;
}

3 个用法
private void swap(int[] array, int x, int y) {
    int temp = array[y];
    array[y] = array[x];
    array[x] = temp;
}
```

子任务二

1、 数据设计

array：需要排序的整数数组

2、 算法设计

堆排序算法的基本思想是利用堆这种数据结构所具有的特性来进行排序。具体步骤如下：

将无序序列构建成一个堆（小顶堆）。

依次取出堆顶元素，将其与堆的最后一个元素交换，然后调整堆，使其满足堆的性质。

重复步骤 2，直到堆中只剩下一个元素，此时整个数组已经按照从小到大的顺序排列。

3、 主干代码说明

```java
public class MinHeapSort {

    1 个用法
    public static void heapSort(int[] array) {
        MinHeap minHeap = new MinHeap(array); // 建立小顶堆
        int n = array.length;
        for (int i = 0; i < n; i++) {
            array[i] = minHeap.findMin(); // 获取堆顶最小元素
            minHeap.deleteMin(); // 删除堆顶
        }
    }

    public static void main(String[] args) {
        int[] test = {7, 5, 6, 4, 2, 1, 3};
        System.out.println("Before sorting:");
        for (int num : test) {
            System.out.print(num + " ");
        }

        heapSort(test);

        System.out.println("\nAfter sorting:");
        for (int num : test) {
            System.out.print(num + " ");
        }
    }
}
```

4、 运行结果展示

```
Before sorting:
7 5 6 4 2 1 3
After sorting:
1 2 3 4 5 6 7
进程已结束，退出代码为 0
```

子任务三

1、 数据设计

2.1 堆的表示

array：使用数组来存储堆中的元素。与二叉堆不同，三叉堆的数组表示需要能够快速定位到每个节点的三个子节点和父节点。数组的第一个元素（索引 0）不使用，从索引 1 开始存储堆的元素。这样，对于任意一个索引 i 的元素，其父节点的索引是(i + 1) / 3，左子节点的索引是 3 * i - 1，中间子节点的索引是 3 * i，右子节点的索引是 3 * i + 1。

2.2 堆的基本属性

DEFAULT_CAPACITY：堆的默认容量，用于初始化堆数组的大小。这个值可以根据实际需要进行调整，以适应不同的使用场景。

currentSize：表示当前堆中元素的数量。由于数组是从索引 1 开始存储数据的，所以 currentSize 的值将始终小于数组的长度。

2.3 堆的操作

insert(int number)：插入操作，将新元素添加到堆中，并保持堆的性质。新元素被添加到数组的末尾（array[currentSize+1]），然后通过上浮操作（percolate up）调整堆。

deleteMin()：删除最小元素操作，移除堆顶元素（即数组的第一个元素），将堆的最后一个元素移动到堆顶，然后通过下沉操作（sift down）调整堆。

findMin()：获取最小元素操作，由于是最小堆，堆顶元素即为最小元素，直接返回数组的第一个元素（array[1]）。

2.4 辅助方法

isEmpty()：检查堆是否为空，即 currentSize 是否为 0。

isFull()：检查堆是否已满，即 currentSize 是否等于数组的长度减 1。

isLeaf(int i)：检查索引 i 的元素是否为叶子节点，即索引 i 大于 currentSize / 3。

buildHeap()：构建最小堆，从最后一个非叶子节点开始，对每个节点进行下沉操作，以确保堆的性质。

getLeft(int i)：获取索引 i 的左子节点的索引。

getMiddle(int i)：获取索引 i 的中间子节点的索引。

getRight(int i)：获取索引 i 的右子节点的索引。

getParent(int i)：获取索引 i 的父节点的索引。

swap(int[] array, int x, int y)：交换数组中两个元素的位置。

2、 算法设计

三叉堆是二叉堆的扩展，其中每个节点最多有三个子节点。在三叉堆中，堆序的要求是完全一致的，即对于最小堆，父节点的值必须小于或等于其所有子节点的值。

插入操作（insert）：将新元素插入到数组的末尾，然后通过比较和交换操作，将新元素上浮到合适的位置，以保持堆的性质。

删除最小元素操作（deleteMin）：移除堆顶元素（最小元素），将数组末尾的元素移动到堆顶，然后通过下沉操作（siftDown）调整堆，以保持堆的性质。

获取最小元素操作（findMin）：直接返回堆顶元素，即数组的第一个元素。

3、 主干代码说明

```
2 个用法
class TernaryMinHeap {
    1 个用法
    private static final int DEFAULT_CAPACITY = 10; // 默认大小
    13 个用法
    private int currentSize; // 当前堆的大小
    18 个用法
    private int[] array; // 堆数组

    0 个用法
    public TernaryMinHeap() {
        this.array = new int[DEFAULT_CAPACITY + 1];
        currentSize = 0;
    }

    0 个用法
    public TernaryMinHeap(int size) {
        this.array = new int[size + 1];
        currentSize = 0;
    }

    1 个用法
    public TernaryMinHeap(int[] array) {
        this.array = new int[array.length + 1];
        for (int i = 0; i < array.length; i++) {
            this.array[i + 1] = array[i];
        } // 从1开始计算
        currentSize = array.length;
        buildHeap();
    }

    /**
     * 向堆中插入元素
     *
     * @param number 元素值
     */
    0 个用法
    public void insert(int number) {
        try {
```

```
            if (isFull()) {
                throw new Exception("Array is full!");
            }
            int temp = ++currentSize;
            array[temp] = number; // 将number放在数组最后
            while ((temp != 1) && (array[temp] < array[getParent(temp)])) {
                swap(array, temp, getParent(temp)); // 比父结点值小则交换
                temp = getParent(temp);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    1 个用法
    public int findMin() {
        return array[1];
    }

    /**
     * 删除堆顶元素
     */
    1 个用法
    public void deleteMin() {
        try {
            if (isEmpty()) {
                throw new Exception("Array is empty!");
            } else {
                swap(array, x: 1, currentSize--); // 将堆顶元素放到最后
                if (currentSize != 0) siftDown( pos: 1);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * 小的值下沉
     *
```

```
     * @param pos 当前位置
     */
    2 个用法
    private void siftDown(int pos) {
        try {
            if (pos < 0 || pos > currentSize) {
                throw new Exception("Illegal position!");
            }
            while (!isLeaf(pos)) {
                int smallest = getLeft(pos); // 假设左子结点最小
                int mid = getMiddle(pos);
                int right = getRight(pos);

                // 比较三个子结点求最小值
                if (mid <= currentSize && array[mid] < array[smallest]) {
                    smallest = mid;
                }
                if (right <= currentSize && array[right] < array[smallest]) {
                    smallest = right;
                }

                if (array[pos] <= array[smallest]) return; // 已满足堆序
                swap(array, pos, smallest);
                pos = smallest;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    1 个用法
    private boolean isEmpty() {
        return currentSize == 0;
    }

    1 个用法
    private boolean isFull() {
        return currentSize == array.length - 1;
    }
```

```
}

1 个用法
private boolean isLeaf(int i) {
    return i > currentSize / 3;
}

1 个用法
private void buildHeap() {
    for (int i = currentSize / 3; i > 0; i--) {
        siftDown(i); // 对每个非叶子结点进行下沉操作
    }
}

1 个用法
private int getLeft(int i) {
    return 3 * i - 1;
}

1 个用法
private int getMiddle(int i) {
    return 3 * i;
}

1 个用法
private int getRight(int i) {
    return 3 * i + 1;
}

3 个用法
private int getParent(int i) {
    return (i + 1) / 3;
}

3 个用法
private void swap(int[] array, int x, int y) {
    int temp = array[y];
```

```
        int temp = array[y];
        array[y] = array[x];
        array[x] = temp;
    }
}

public class TernaryHeapSort {
    1 个用法
    public static void heapSort(int[] array) {
        TernaryMinHeap minHeap = new TernaryMinHeap(array); // 建立三叉
        int n = array.length;
        for (int i = 0; i < n; i++) {
            array[i] = minHeap.findMin(); // 获取堆顶最小元素
            minHeap.deleteMin(); // 删除堆顶
        }
    }

    public static void main(String[] args) {
        int[] test = {7, 5, 6, 4, 2, 1, 3};
        System.out.println("Before sorting:");
        for (int num : test) {
            System.out.print(num + " ");
        }

        heapSort(test);

        System.out.println("\nAfter sorting:");
        for (int num : test) {
            System.out.print(num + " ");
        }
    }
}
```

4、 运行结果展示

```
Before sorting:
7 5 6 4 2 1 3
After sorting:
1 2 3 4 5 6 7
进程已结束，退出代码为 0
```

5、 总结和收获

通过这三个子任务，我们不仅学习了堆的基本概念和操作，还通过实际编程实践加深了对堆排序算法的理解。我们掌握了如何使用数组来实现堆，以及如何通过上浮和下沉操作来维护堆的性质。此外，我们还探索了堆结构的扩展，比如从二叉堆到三叉堆的转变，这让我们对数据结构有了更深入的认识。

附录：每个题的源代码

任务一

BST 接口

```
import java.io.IOException;
import java.io.PrintWriter;
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;
}
```

二叉检索树

```
import java.io.IOException;
import java.io.PrintWriter;

// BST 节点类
class BSTNode<K extends Comparable<K>, V> {
```

```
        K key;
        V value;
        BSTNode<K, V> left, right;

        BSTNode(K key, V value) {
            this.key = key;
            this.value = value;
            this.left = this.right = null;
        }
}

// BST 实现类
public class BinarySearchTree<K extends Comparable<K>, V>
implements BST<K, V> {
        private BSTNode<K, V> root;
        private int size;

        public BinarySearchTree() {
            this.root = null;
            this.size = 0;
        }
```

```
    @Override
    public void insert(K key, V value) {
        if (key == null || value == null) return;
        root = insertRecursive(root, key, value);
    }

    private BSTNode<K, V> insertRecursive(BSTNode<K, V> node, K key, V value) {
        if (node == null) {
            size++;
            return new BSTNode<>(key, value);
        }
        int cmp = key.compareTo(node.key);
        if (cmp < 0) node.left = insertRecursive(node.left, key, value);
        else if (cmp > 0) node.right = insertRecursive(node.right, key, value);
        else node.value = value; // 更新值
        return node;
    }

    @Override
```

```
    public V remove(K key) {
        if (key == null) {
            System.out.println("remove unsuccess ---" + key);
            return null;
        }
        V[] removedValue = (V[]) new Object[1];
        root = removeRecursive(root, key, removedValue);
        if (removedValue[0] == null) {
            System.out.println("remove unsuccess ---" + key);
        } else {
            System.out.println("remove success ---" + key + " " +
removedValue[0]);
        }
        return removedValue[0];
    }

    private BSTNode<K, V> removeRecursive(BSTNode<K, V> node,
K key, V[] removedValue) {
        if (node == null) return null;

        int cmp = key.compareTo(node.key);
```

```
            if (cmp < 0) node.left = removeRecursive(node.left, key,
removedValue);
            else if (cmp > 0) node.right = removeRecursive(node.right,
key, removedValue);
            else { //  找到节点
                removedValue[0] = node.value;
                size--;
                if (node.left == null) return node.right;
                if (node.right == null) return node.left;

                BSTNode<K, V> minNode = getMin(node.right);
                node.key = minNode.key;
                node.value = minNode.value;
                node.right = removeRecursive(node.right,
minNode.key, null);
            }
            return node;
        }

        private BSTNode<K, V> getMin(BSTNode<K, V> node) {
            while (node.left != null) node = node.left;
            return node;
```

```
    }

    @Override
    public V search(K key) {
        V result = searchRecursive(root, key);
        if (result == null) {
            System.out.println("search unsuccess ---" + key);
        } else {
            System.out.println("search success ---" + key + " " +
result);
        }
        return result;
    }
    private V searchRecursive(BSTNode<K, V> node, K key) {
        if (node == null) return null;

        int cmp = key.compareTo(node.key);
        if (cmp < 0) return searchRecursive(node.left, key);
        else if (cmp > 0) return searchRecursive(node.right, key);
        else return node.value;
    }
```

```
    @Override
    public boolean update(K key, V value) {
        if (key == null || value == null) {
            System.out.println("update unsuccess ---" + key);
            return false;
        }
        boolean result = updateRecursive(root, key, value);
        if (result) {
            System.out.println("update success ---" + key + " " +
value);
        } else {
            System.out.println("update unsuccess ---" + key);
        }
        return result;
    }
    private boolean updateRecursive(BSTNode<K, V> node, K key, V
value) {
        if (node == null) return false;
        int cmp = key.compareTo(node.key);
        if (cmp < 0) return updateRecursive(node.left, key, value);
        else if (cmp > 0) return updateRecursive(node.right, key,
value);
```

```
        else {
            node.value = value;
            return true;
        }
    }

    @Override
    public boolean isEmpty() {
        return size == 0;
    }

    @Override
    public void clear() {
        root = null;
        size = 0;
    }

    @Override
    public void showStructure(PrintWriter pw) throws IOException {
        if (pw == null) throw new IOException("PrintWriter is null");
        pw.println("-----------------------------");
        pw.println("There are " + size + " nodes in this BST.");
```

```
            pw.println("The height of this BST is " + getHeight(root) +
"."");
            pw.println("----------------------------");
        }

        private int getHeight(BSTNode<K, V> node) {
            if (node == null) return 0;
            return 1 + Math.max(getHeight(node.left),
getHeight(node.right));
        }

        @Override
        public void printInorder(PrintWriter pw) throws IOException {
            if (pw == null) throw new IOException("PrintWriter is null");
            printInOrderRecursive(root, pw);
        }

        private void printInOrderRecursive(BSTNode<K, V> node,
PrintWriter pw) throws IOException {
            if (node == null) return;
            printInOrderRecursive(node.left, pw);
            pw.println(node.key + " --- <" + node.value + ">");
```

```
            printInOrderRecursive(node.right, pw);
        }
}
```

测试代码

```
import java.io.*;
import java.util.Scanner;
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;
public class BSTTest {
    public static void main(String[] args) throws IOException {
            BinarySearchTree<String, String> bst = new
BinarySearchTree<>();
            File inputFile = new File("BST_testcases.txt");
            File outputFile = new File("BST_result.txt");

            // Set up input and output
            Scanner scanner = new Scanner(inputFile);
            PrintWriter writer = new PrintWriter(outputFile);

            while (scanner.hasNextLine()) {
                String command = scanner.nextLine();
                processCommand(command, bst, writer);
```

```
            }

            writer.close();
            scanner.close();
            System.out.println("Testing complete. Results saved to
BST_result.txt");
        }

        private static void processCommand(String command,
BinarySearchTree<String, String> bst, PrintWriter writer) {
            try {
                // 创建输出捕获工具
                ByteArrayOutputStream baos = new
ByteArrayOutputStream();
                PrintStream ps = new PrintStream(baos);
                PrintStream old = System.out;

                if (command.startsWith("+")) { // Insert
                    String[] parts = parseInsert(command);
                    bst.insert(parts[0], parts[1]);
                } else if (command.startsWith("-")) { // Remove
                    String key = parseKey(command);
```

```
            // 重定向输出
            System.setOut(ps);
            bst.remove(key);
            // 恢复输出并写入文件
            System.out.flush();
            System.setOut(old);
            writer.println(baos.toString().trim());

        } else if (command.startsWith("?")) { // Search
            String key = parseKey(command);
            // 重定向输出
            System.setOut(ps);
            bst.search(key);
            // 恢复输出并写入文件
            System.out.flush();
            System.setOut(old);
            writer.println(baos.toString().trim());

        } else if (command.startsWith("=")) { // Update
            String[] parts = parseInsert(command);
            // 重定向输出
            System.setOut(ps);
```

```
                bst.update(parts[0], parts[1]);
                // 恢复输出并写入文件
                System.out.flush();
                System.setOut(old);
                writer.println(baos.toString().trim());

            } else if (command.startsWith("#")) { // Show structure
                bst.showStructure(writer);
            } else {
                writer.println("Invalid command: " + command);
            }
        } catch (Exception e) {
        }
    }
    private static String[] parseInsert(String command) {
        int start = command.indexOf('(') + 1;
        int end = command.indexOf(')');
        String[] parts = command.substring(start, end).split(",");
        parts[0] = parts[0].trim();
        parts[1] = parts[1].trim();
        return parts;
    }
```

```
    private static String parseKey(String command) {
        int start = command.indexOf('(') + 1;
        int end = command.indexOf(')');
        return command.substring(start, end).trim();
    }
}
```

结果

```
----------------------------
There are 1 nodes in this BST.
The height of this BST is 1.
----------------------------
remove unsuccess ---vertebrate
update success ---overlap "overlap"
remove unsuccess ---affection
search success ---persevere "v．坚忍"
----------------------------
There are 11 nodes in this BST.
The height of this BST is 5.
----------------------------
update success ---pretentiousness "pretentiousness"
```

There are 13 nodes in this BST.

The height of this BST is 5.

---

remove success ---sill "n．基石，门槛，窗台"

search unsuccess ---listless

search success ---gollop "v．n．大口吞咽"

update success ---sagacious "sagacious"

remove unsuccess ---propitiate

update success ---ideology "ideology"

search success ---puerile "adj．幼稚的，儿童的"

update success ---laggard "laggard"

search unsuccess ---penalize

search success ---deify "v．奉为神，崇拜"

search unsuccess ---midget

search success ---mutton "n．羊肉"

update success ---conflagration "conflagration"

search unsuccess ---morass

search success ---breach "v．毁坏，泄密 n．缺口"

search success ---sagacious "sagacious"

There are 33 nodes in this BST.

The height of this BST is 8.

---

search unsuccess ---guarded

search success ---sagacious "sagacious"

remove unsuccess ---egalitarian

update success ---incendiary "incendiary"

search success ---nocturnal "adj．夜晚的，夜间发生的"

update success ---deify "deify"

remove success ---solemnity "n．庄严，肃穆"

search unsuccess ---tongs

search success ---squash "v．压碎，挤压，n．南瓜"

update success ---abate "abate"

There are 46 nodes in this BST.

The height of this BST is 9.

---

search unsuccess ---vaccine

search success ---incendiary "incendiary"

search unsuccess ---unctuous

search success ---croak "n．蛙鸣声"

search success ---gravel "n．碎石，砂砾"

update success ---coy "coy"

There are 53 nodes in this BST.

The height of this BST is 9.

---

search success ---behold "v．目睹，看见"

search success ---nocturnal "adj．夜晚的，夜间发生的"

update success ---mast "mast"

There are 58 nodes in this BST.

The height of this BST is 9.

---

remove success ---peruse "v．细读，精读"

remove success ---suffice "v．足够，（食物）满足"

update success ---unwieldy "unwieldy"

update success ---auspices "auspices"

update success ---sagacious "sagacious"

search unsuccess ---infantry

search success ---hide "n．兽皮"

remove unsuccess ---recriminate

update success ---bamboozle "bamboozle"

update success ---ingress "ingress"

remove success ---profundity "n．深奥

search success ---chivalry "n.骑士制度"

update success ---hepatic "hepatic"

remove unsuccess ---reciprocity

search success ---gollop "v．n．大口吞咽"

search success ---liability "n．倾向，债务"

search success ---veal "n．小牛肉"

remove unsuccess ---throttle

remove unsuccess ---oath

search success ---impeach "v．指摘，弹劾"

remove success ---mishap "n．不幸，坏运气"

There are 81 nodes in this BST.

The height of this BST is 10.

---

remove success ---malapropism "n．字的误用"

update success ---abate "abate"

search unsuccess ---turbid

search success ---ingress "ingress"

search unsuccess ---whimsical

search success ---bamboozle "bamboozle"

search unsuccess ---filch

search success ---helmsman "n．舵手"

search success ---moratorium "n．停止偿付，禁止"

search success ---dispatch "v．派遣，一下子做完、吃完 n．迅速"

remove success ---binge "n.狂饮，无节制的行为"

There are 84 nodes in this BST.

The height of this BST is 10.

---

remove success ---trudge "v．跋涉"

search success ---conduce "tov．引起，有助于"

search success ---temperamental "adj．性情的，喜怒无常的"

update success ---leviathan "leviathan"

search unsuccess ---malcontented

search success ---usurp "v．篡夺，霸占"

search success ---inclement "adj．（天气）严酷的，严厉的"

There are 94 nodes in this BST.

The height of this BST is 11.

---

search success ---poignant "adj．伤心的

update success ---deify "deify"

search unsuccess ---fumble

search success ---jamb "n．门窗的侧柱"

search success ---poignant "adj．伤心的

search success ---marsh "n．沼泽地，湿地"

search unsuccess ---pertinent

search success ---abyss "n．深渊，深坑"

remove success ---groggy "adj．体弱的，不稳的"

update success ---colt "colt"

There are 101 nodes in this BST.

The height of this BST is 11.

---

search success ---orientation "n．定向"

update success ---dispatch "dispatch"

update success ---unwieldy "unwieldy"

search success ---exultant "adj．愉悦的"

There are 104 nodes in this BST.

The height of this BST is 11.

There are 104 nodes in this BST.

The height of this BST is 11.

---

search unsuccess ---brute

search success ---leviathan "leviathan"

remove success ---temperamental "adj．性情的，喜怒无常的"

remove success ---bane "n.祸根"

remove success ---inflamed "adj．发炎的，红肿的"

remove unsuccess ---gloaming

There are 110 nodes in this BST.

The height of this BST is 10.

---

search success ---insubordinate "adj．不服从的，反抗的"

search success ---paean "n．赞美歌，颂歌"

update success ---bamboozle "bamboozle"

There are 112 nodes in this BST.

The height of this BST is 10.

---

update success ---statute "statute"

search unsuccess ---foment

search success ---prevail "v．战胜

remove success ---sonata "n．奏鸣曲"

There are 111 nodes in this BST.

The height of this BST is 10.

---

remove unsuccess ---gloom

search success ---ingress "ingress"

search unsuccess ---complacent

search success ---psalm "n．赞美诗

search unsuccess ---scorn

search success ---puerile "adj．幼稚的，儿童的"

search success ---interlocutor "n．对话者

remove success ---whimsical "adj．古怪的，异想天开的"

update success ---fretful "fretful"

update success ---usurp "usurp"

search success ---interlace "v．编织

search success ---hepatic "hepatic"

update success ---dictum "dictum"

update success ---croak "croak"

search unsuccess ---sparkle

search success ---conceive "v．怀孕，构想"

There are 126 nodes in this BST.

The height of this BST is 11.

---

remove success ---retaliate "v．报复，反击"

search unsuccess ---banner

search success ---paean "n．赞美歌，颂歌"

There are 134 nodes in this BST.

The height of this BST is 12.

---

remove success ---conceive "v．怀孕，构想"

update success ---jamb "jamb"

There are 134 nodes in this BST.

The height of this BST is 12.

---

remove success ---meritorious "adj．值得赞赏的"

update success ---aseptic "aseptic"

remove unsuccess ---tardy

There are 134 nodes in this BST.

The height of this BST is 12.

---

remove success ---swerve "v．转向，突然改变方向"

remove success ---alteration "n．改变，变更"

update success ---torrid "torrid"

search unsuccess ---gratis

search success ---inexorable "adj．不为所动的，坚决不变的"

remove success ---supple "adj 伸屈自如的，灵活的"

There are 135 nodes in this BST.

The height of this BST is 12.

---

search success ---rampant "adj．蔓生的，猖獗的"

search success ---singe "v．（轻微地）烧焦，烫焦"

There are 137 nodes in this BST.

The height of this BST is 12.

There are 137 nodes in this BST.

The height of this BST is 12.

---

update success ---snarl "snarl"

update success ---tournament "tournament"

search success ---moor "n．旷野地，荒野 v．使（船）停泊"

remove success ---corpulence "n．肥胖，臃肿"

There are 139 nodes in this BST.

The height of this BST is 12.

---

remove success ---ingress "ingress"

remove success ---serendipity "n．善于发掘新奇事物的天赋"

search success ---gnome "n．地下宝藏的守护神，地精，格言"

There are 145 nodes in this BST.

The height of this BST is 12.

There are 145 nodes in this BST.

The height of this BST is 12.

---

update success ---terpsichorean "terpsichorean"

remove unsuccess ---adopt

search unsuccess ---ecstasy

search success ---impeach "v．指摘，弹劾"

search success ---sleek "v．（使）光滑，adj．光滑的，整洁的"

remove unsuccess ---trauma

search success ---amorphous "adj．无定形的，散漫的"

remove success ---bicker "n.v.争吵，口角"

There are 161 nodes in this BST.

The height of this BST is 12.

----

remove unsuccess ---autonomy

search success ---abysmal "adj．极深的，糟透的"

update success ---irate "irate"

remove success ---inexorable "adj．不为所动的，坚决不变的"

remove unsuccess ---promptness

update success ---kangaroo "kangaroo"

remove unsuccess ---opponent

update success ---poignant "poignant"

update success ---hood "hood"

search success ---colt "n.小雄驹"

remove unsuccess ---relevance

There are 171 nodes in this BST.

The height of this BST is 12.

----

remove unsuccess ---allurement

remove unsuccess ---pesky

search success ---lineal "adj．直系的，嫡系的"

update success ---enzyme "enzyme"

remove success ---helmsman "n．舵手"

search unsuccess ---exponent

search success ---quitter "n．遇困难即放弃者"

update success ---nebula "nebula"

search unsuccess ---prorogue

search success ---paean "n．赞美歌，颂歌"

search success ---coagulant "n.凝结剂"

There are 183 nodes in this BST.

The height of this BST is 13.

---

update success ---omission "omission"

search unsuccess ---reverence

search success ---sleek "v．（使）光滑，adj．光滑的，整洁的"

search success ---croak "croak"

search success ---moor "n．旷野地，荒野 v．使（船）停泊"

search success ---bamboozle "bamboozle"

search unsuccess ---anomalous

search success ---fern "n．羊齿植物，蕨"

update success ---snarl "snarl"

There are 190 nodes in this BST.

The height of this BST is 13.

---

search unsuccess ---oracle

search success ---soprano "n 女高音，最高者"

remove success ---nuptials "n．婚礼"

remove unsuccess ---canyon

remove success ---conduce "tov．引起，有助于"

remove unsuccess ---pinch

search success ---dictum "dictum"

remove unsuccess ---quisling

remove success ---deflect "v．偏离，转向"

update success ---mattress "mattress"

search success ---acquaintance "n．熟知，熟人"

update success ---penury "penury"

search success ---circumvent "v.用计谋战胜，规避"

remove unsuccess ---drenched

search success ---clientele "n.(医生、律师的

There are 206 nodes in this BST.

The height of this BST is 13.

---

search success ---contumacy "n．抗命，不服从"

search success ---dispatch "dispatch"

search success ---amorphous "adj．无定形的，散漫的"

search unsuccess ---notability

search success ---statute "statute"

search success ---squash "v．压碎，挤压，n．南瓜"

update success ---fusillade "fusillade"

There are 209 nodes in this BST.

The height of this BST is 13.

---

update success ---inclement "inclement"

remove success ---fugitive "adj．逃亡的，易逝的 n．逃犯，逃亡者"

update success ---leviathan "leviathan"

search success ---snarl "snarl"

There are 218 nodes in this BST.

The height of this BST is 13.

There are 218 nodes in this BST.

The height of this BST is 13.

---

remove success ---imaginary "adj．虚构的"

remove success ---diurnal "adj．白昼的，白天的"

search unsuccess ---ornate

search success ---jinx "n.不祥的人(或物

update success ---behold "behold"

search success ---abate "abate"

update success ---munificent "munificent"

search unsuccess ---altar

search success ---tranquillizer "n．镇定剂"

search unsuccess ---choppy

search success ---repellent "adj．令人厌恶的"

search unsuccess ---complaisant

search success ---protract "v．延长

update success ---banality "banality"

There are 233 nodes in this BST.

The height of this BST is 14.

---

update success ---morale "morale"

remove success ---derivative "adj．派生的，无创意的"

update success ---concussion "concussion"

There are 233 nodes in this BST.

The height of this BST is 14.

---

search success ---bawl "v.大叫

There are 234 nodes in this BST.

The height of this BST is 14.

---

search success ---null ""

remove unsuccess ---fleeting

update success ---poignant "poignant"

search success ---hood "hood"

search success ---fern "n．羊齿植物，蕨"

remove unsuccess ---gratuitous

search success ---tournament "tournament"

update success ---anode "anode"

update success ---irate "irate"

search unsuccess ---gross

search success ---iguana "n.美洲大蜥蜴"

update success ---stapler "stapler"

update success ---penury "penury"

update success ---oesophagus "oesophagus"

search success ---stapler "stapler"

search success ---mattress "mattress"

search success ---provocation "n．激怒"

update success ---exultant "exultant"

remove unsuccess ---furlough

remove unsuccess ---disdainful

update success ---oratorio "oratorio"

search success ---anode "anode"

search success ---termite "n．白蚁"

remove success ---jinx "n.不祥的人(或物

remove success ---acquaintance "n．熟知，熟人"

search success ---drawl "v．慢吞吞说，拉长腔调说"

update success ---aversion "aversion"

update success ---delectation "delectation"

There are 262 nodes in this BST.

The height of this BST is 15.

---

update success ---ideology "ideology"

update success ---circumvent "circumvent"

remove success ---interlace "v．编织

update success ---snaky "snaky"

update success ---immutable "immutable"

search success ---caustic "adj．腐蚀性的，刻薄的"

There are 266 nodes in this BST.

The height of this BST is 15.

---

search success ---solidarity "n．团结，一致"

remove unsuccess ---simulated

There are 269 nodes in this BST.

The height of this BST is 15.

---

remove success ---coy "coy"

update success ---mortify "mortify"

update success ---rotary "rotary"

remove success ---snaky "snaky"

remove success ---perception "n．洞察力，理解力"

search success ---speculate "v．沉思，思索，投机"

remove unsuccess ---renowned

search unsuccess ---oasis

search success ---nippy "adj.辛辣的，寒冷的"

search unsuccess ---salmon

search success ---abysmal "adj．极深的，糟透的"

search unsuccess ---equitable

search success ---compere "n.节目主持人，司仪，v.主持(节目

There are 283 nodes in this BST.

The height of this BST is 15.

---

update success ---jamb "jamb"

remove success ---spiral "n．螺旋，螺线"

search success ---expend "v.花费

update success ---penury "penury"

remove success ---spiny "adj．针状的，多刺的"

update success ---interlock "interlock"

remove unsuccess ---obsequious

remove unsuccess ---entwine

search success ---croak "croak"

search success ---hoop "n．（桶之）箍，铁环"

search success ---velocity "n．速度，迅速"

update success ---paean "paean"

search unsuccess ---exacerbate

search success ---inclement "inclement"

update success ---sophism "sophism"

search unsuccess ---redeem

search success ---gauge "n．标准规格，测量仪"

There are 300 nodes in this BST.

The height of this BST is 15.

---

remove success ---stipulate "v，要求以…为条件，约定"

update success ---piercing "piercing"

search success ---inaugurate "v．举行就职典礼，开创"

search success ---veal "n．小牛肉"

remove success ---baroque "n.adj.(艺术、建筑等

search success ---omission "omission"

search success ---traduce "v．中伤，诽谤"

remove success ---legislature "n．立法机关"

update success ---temerity "temerity"

search unsuccess ---relaxation

search success ---leviathan "leviathan"

remove unsuccess ---accommodating

search success ---insufferable "adj．无法忍受的"

update success ---rotary "rotary"

remove success ---gangling "adj.瘦长得难看的"

update success ---pertinent "pertinent"

There are 328 nodes in this BST.

The height of this BST is 16.

---

remove success ---inaugurate "v．举行就职典礼，开创"

search success ---deify "deify"

remove success ---raucous "adj．（声音）沙哑的，粗糙的"

update success ---sleek "sleek"

search success ---marsh "n．沼泽地，湿地"

search success ---icon "n．圣像，偶像"

search unsuccess ---interpolate

search success ---obbligato "n.伴奏"

search success ---depreciate "v．贬低，贬值"

remove unsuccess ---blase

There are 332 nodes in this BST.

The height of this BST is 16.

---

update success ---scrutinize "scrutinize"

search success ---aseptic "aseptic"

update success ---showpiece "showpiece"

search unsuccess ---permissive

search success ---expiration "n.期满

remove unsuccess ---rebarbative

search success ---plume "n．羽毛 v．整理羽毛

remove unsuccess ---outset

remove unsuccess ---assault

There are 338 nodes in this BST.

The height of this BST is 16.

---

remove success ---slumber "v．n．睡眠，安睡"

remove unsuccess ---supersede

update success ---iguana "iguana"

remove unsuccess ---dispensable

remove unsuccess ---accessible

search success ---demesne "n.领地，范围"

search success ---botanical "adj．植物（学）的"

search success ---poniard "n．匕首

remove unsuccess ---gangling

search success ---insubordinate "adj．不服从的，反抗的"

remove success ---perigee "n．近地点"

search unsuccess ---contrition

search success ---divergence "n．分歧，分开"

update success ---lunatic "lunatic"

update success ---tranquillizer "tranquillizer"

remove unsuccess ---mime

update success ---hinterland "hinterland"

update success ---chivalry "chivalry"

search unsuccess ---preservative

search success ---gravel "n．碎石，砂砾"

update success ---repugnant "repugnant"

update success ---renaissance "renaissance"

search success ---herd "n．兽群。v．聚集，放牧"

remove unsuccess ---adornment

remove unsuccess ---confirmed

search success ---texture "n．质地，结构"

search unsuccess ---chorus

There are 384 nodes in this BST.

The height of this BST is 16.

---

update success ---bubble "bubble"

search unsuccess ---exalt

search success ---poniard "n．匕首

remove success ---abolition "n．废除，革除"

remove unsuccess ---wheedle

search unsuccess ---lingual

search success ---clandestine "adj.秘密的，暗中从事的"

remove unsuccess ---fume

search success ---ulterior "adj．较晚的，较远的，不可告人的"

search unsuccess ---burrow

search success ---torrid "torrid"

remove unsuccess ---unalloyed

remove unsuccess ---onslaught

update success ---exhortation "exhortation"

remove success ---macabre "adj．骇人的，可怖的"

search unsuccess ---restorative

search success ---pawky "adj.精明的，活泼的"

search unsuccess ---incognito

search success ---dispatch "dispatch"

remove success ---ulterior "adj．较晚的，较远的，不可告人的"

remove unsuccess ---intercept

update success ---leakage "leakage"

There are 399 nodes in this BST.

The height of this BST is 17.

---

update success ---appreciable "appreciable"

remove success ---sequester "v．（使）隐退，隐居"

remove success ---tempo "n．（动作、生活的）步调，速度"

There are 399 nodes in this BST.

The height of this BST is 17.

There are 402 nodes in this BST.

The height of this BST is 17.

There are 402 nodes in this BST.

The height of this BST is 17.

---

search success ---renaissance "renaissance"

update success ---caste "caste"

update success ---behold "behold"

search success ---satiate "v．使饱足，生腻"

search success ---stapler "stapler"

search success ---tiff "v．n．吵嘴，呕气"

remove success ---antedate "（在信、文件上）写上较早日期"

search success ---zephyr "n．和风，西风"

search success ---expend "v.花费

remove success ---permeable "adj．可渗透的"

update success ---toupee "toupee"

There are 410 nodes in this BST.

The height of this BST is 17.

---

search unsuccess ---crudity

search success ---icon "n．圣像，偶像"

There are 412 nodes in this BST.

The height of this BST is 17.

---

update success ---orientation "orientation"

update success ---credence "credence"

search success ---zephyr "n．和风，西风"

search success ---dynamite "n．炸药，扣人心弦的事"

remove unsuccess ---scurrilous

remove unsuccess ---regiment

search unsuccess ---ply

search success ---ellipsis "n　省略"

There are 417 nodes in this BST.

The height of this BST is 17.

---

remove unsuccess ---arsonist

search unsuccess ---vandalism

search success ---royalty "n．皇室，版税（付给著作人的钱）"

There are 424 nodes in this BST.

The height of this BST is 17.

---

remove success ---gambol "n．雀跃，嬉戏"

remove unsuccess ---doting

任务二

```
import java.io.*;
import java.util.*;

// 自定义 LinkedList 实现，用于存储行号
```

```
class LinkedList {
    private Node head;

    private class Node {
        int data;
        Node next;

        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }

    // 添加行号，避免重复添加同一行号
    public void add(int data) {
        if (head == null) {
            head = new Node(data);
            return;
        }
        Node current = head;
        while (current.next != null) {
            if (current.data == data) return; // 避免重复添
```

```
加
                current = current.next;
            }
            if (current.data != data) current.next = new
Node(data);
        }

        @Override
        public String toString() {
            StringBuilder sb = new StringBuilder();
            Node current = head;
            while (current != null) {
                sb.append(current.data).append(" ");
                current = current.next;
            }
            return sb.toString().trim();
        }
}

// BST 节点类
class BSTNode<K extends Comparable<K>> {
    K key;
```

```
    LinkedList lines; //  存储出现的行号
    BSTNode<K> left, right;

    BSTNode(K key, int line) {
        this.key = key;
        this.lines = new LinkedList();
        this.lines.add(line);
        this.left = this.right = null;
    }
}

// BST 实现类
class BinarySearchTree<K extends Comparable<K>> {
    private BSTNode<K> root;

    //  插入单词和行号
    public void insert(K key, int line) {
        root = insertRecursive(root, key, line);
    }

    private BSTNode<K> insertRecursive(BSTNode<K>
node, K key, int line) {
```

```
            if (node == null) return new BSTNode<>(key, line);

            int cmp = key.compareTo(node.key);
            if (cmp < 0) {
                node.left = insertRecursive(node.left, key, line);
            } else if (cmp > 0) {
                node.right = insertRecursive(node.right, key,
line);
            } else {
                node.lines.add(line); // 单词已存在，更新行号
            }
            return node;
        }

        // 中序遍历，写入文件
        public void printInOrder(PrintWriter pw) {
            printInOrderRecursive(root, pw);
        }

        private void printInOrderRecursive(BSTNode<K> node,
PrintWriter pw) {
            if (node == null) return;
```

```
            printInOrderRecursive(node.left, pw);
            pw.println("[" + node.key + "---<" + node.lines +
">]");
            printInOrderRecursive(node.right, pw);
        }
}

// 主类
public class IndexBuilder {
    public static void main(String[] args) {
        String inputFile = "article.txt";
        String outputFile = "index_result.txt";

        // 初始化 BST
        BinarySearchTree<String> bst = new
BinarySearchTree<>();

        // 读取文件并构建索引
        try (BufferedReader br = new BufferedReader(new
FileReader(inputFile))) {
            String line;
            int lineNumber = 0;
```

```
                while ((line = br.readLine()) != null) {
                    lineNumber++;
                    String[] words = line.split("\\W+"); // 使用
非字母字符分割
                    for (String word : words) {
                        if (!word.isEmpty()) {
                            bst.insert(word.toLowerCase(),
lineNumber); // 单词转小写
                        }
                    }
                }
            } catch (IOException e) {
                System.err.println("Error reading file: " +
e.getMessage());
            }

            // 输出索引结果到文件
            try (PrintWriter pw = new PrintWriter(new
FileWriter(outputFile))) {
                bst.printInOrder(pw);
                System.out.println("Index successfully written
```

```
to " + outputFile);
            } catch (IOException e) {
                System.err.println("Error writing file: " +
e.getMessage());
            }
        }
}
```

输出结果

```
[a---<17 35 37 55 65 78 79 80 95 100 123 125 151 153 157
165 175 182 186 198 203 204 207 213 214 215 217 218 222
223 227 228 229 237 239 241 245 248 252 254 255 283 305
306 308 328 354 366 389 392 397 411 414 420 424 425 441
442 445 447 450>]
[able---<219>]
[about---<16 142 182 184 186 194 199 208 211 227 255
268 416 424>]
[absolutely---<78 82 84 221 224>]
[across---<46 66 79>]
[addresses---<270>]
[adjutant---<13 15>]
[adrianople---<46>]
[affair---<218 448>]
```

[after---<175 184 197 203 228 307 347 364 366>]

[afternoon---<351>]

[afterwards---<305>]

[ag---<176 177 180 185 187 197 204 213 217 226>]

[again---<344 356>]

[against---<28 29 95 99 101 111 115>]

[agreed---<203>]

[ahead---<116>]

[aid---<240>]

[ain---<159>]

[all---<12 56 70 97 162 186 187 199 246 298 307 324 341 343 345 364 373 384 407 427 450>]

[alley---<154>]

[allow---<444>]

[almost---<51>]

[alone---<83 358>]

[along---<9 12 16 47 50 52 53 121 187 200 286>]

[alongside---<13>]

[already---<271>]

[altogether---<256 447>]

[always---<222>]

[am---<12 307>]

[america---<213 451>]

[an---<78 84 117 154 183 227 318 364 448>]

[and---<11 13 14 25 26 27 28 29 30 31 33 34 35 36 37 48 49 50 53 56 67 68 80 81 82 83 86 99 113 117 121 122 134 137 139 140 141 142 152 155 173 175 176 177 182 186 190 191 193 194 198 199 200 201 205 206 207 214 217 218 219 220 221 223 239 240 241 243 246 247 248 251 253 254 266 269 281 282 284 286 287 288 298 299 300 301 302 303 304 305 306 307 317 319 320 321 326 327 329 331 332 340 341 342 343 344 349 350 356 357 358 360 361 366 371 372 382 384 385 386 389 391 393 394 395 396 406 420 428 442>]

[another---<242>]

[answer---<227>]

[any---<121 160 226 240>]

[anybody---<143>]

[anyone---<206>]

[anything---<136 184>]

[anyway---<443>]

[aosta---<267>]

[arditi_---<215>]

[are---<138>]

[arm---<34 357>]

[armistice---<198 203>]

[arms---<381>]

[around---<303 341 388 390 412>]

[as---<187 193 222 284 396>]

[asked---<258 424>]

[at---<56 65 94 122 133 141 142 151 152 201 208 209 251 266 288 306 317 320 325 331 332 385 405 441>]

[audience---<125>]

[austrian---<118>]

[avanti_---<248>]

[avenue---<152>]

[away---<31 124 305 356>]

[awfully---<68>]

[awkwardly---<113>]

[back---<86 141 187 190 213 285 318 321 354 360 361 427 430>]

[backed---<115>]

[backing---<154>]

[backward---<284>]

[bad---<255 301>]

[badly---<260>]

[balcony---<177 360>]

[ball---<428>]

[banns---<192>]

[barrera---<286 303>]

[barricade---<79>]

[barriers---<387>]

[battalion---<215 216>]

[battery---<9>]

[be---<14 112 121 157 204 207 219 220 221 263 354 361 408 409 420 448>]

[bearers---<120>]

[bearings---<428>]

[beautiful---<245>]

[became---<328>]

[because---<32 35>]

[bed---<177 186 188>]

[bedstead---<117>]

[been---<113 218 246 391 408 416 423 447>]

[before---<190 217 307 431>]

[began---<284>]

[beginning---<49>]

[behind---<243 320>]

[being---<208>]

[believe---<137 445>]

[believed---<224 256>]

[belly---<27>]

[below---<177>]

[beside---<430>]

[best---<224>]

[better---<262 424>]

[between---<289 330>]

[big---<79 114 442>]

[birth---<193>]

[blab---<183>]

[blanket---<56 412>]

[bleeding---<382>]

[blood---<288 331>]

[blue---<283>]

[boat---<213>]

[bobbing---<52>]

[bologna---<251 266>]

[bombardment---<133>]

[both---<112 156 423>]

[bottles---<176 299>]

[bought---<247>]

[box---<155>]

[boy---<218 223 254>]

[boyle---<152 154 159 165>]

[bread---<298>]

[break---<350>]

[breathing---<123>]

[bridge---<51 52 79>]

[bridle---<287>]

[brilliantly---<116>]

[bringing---<361>]

[brought---<408>]

[brown---<306>]

[buckley---<65>]

[budapest---<240>]

[buffalo---<48>]

[built---<428>]

[bull---<26 28 33 35 36 38 288 290 300 308 317 320 324 327 328 330 331 383 385 386>]

[bulls---<32 351 366 371 372>]

[bumped---<163 383>]

[bunch---<198 283>]

[bundles---<55>]

[bush---<441>]

[but---<15 31 100 162 192 194 210 220 222 258 262 270 445>]

[by---<198 385>]

[bye---<209 210 211 266>]

[cabinet---<94>]

[caf---<306>]

[called---<326>]

[came---<29 31 65 69 70 83 140 175 198 303 341 362 390 414 421>]

[camels---<52>]

[can---<32 165>]

[canter---<284>]

[cantered---<285>]

[cap---<421>]

[cape---<37 385>]

[cardinella---<405 416 422 429>]

[career---<223>]

[carefully---<121 124>]

[carried---<98 173 248>]

[carrying---<30 237 415>]

[carts---<47 48 49 51 54>]

[cattle---<48 50>]

[caught---<26 349>]

[cavalry---<53>]

[caving---<332>]

[cells---<407 409>]

[certificates---<193>]

[chair---<424 430>]

[champagne---<10>]

[chaps---<446>]

[chapter---<6 22 43 62 75 91 108 130 148 170 234 278 295 313 337 378 402 436>]

[charge---<290 321>]

[charged---<317 327>]

[cheerful---<141>]

[chicago---<227>]

[chimney---<174>]

[christ---<134 135 136 351>]

[church---<111>]

[cigar---<151>]

[cinematograph---<397>]

[clear---<112>]

[climb---<81>]

[climbed---<66>]

[clipping---<441>]

[clock---<151 405 416>]

[close---<316>]

[cogida---<367>]

[come---<204 205 208 350 354>]

[coming---<340 382>]

[commenced---<396>]

[committee---<443>]

[comrade---<239>]

[comrades---<240 270>]

[contracted---<228>]

[control---<422>]

[cool---<177>]

[copy---<248>]

[corner---<341>]

[corporal---<17>]

[corral---<390>]

[corridor---<406>]

[cot---<389 411>]

[could---<81 174 176 195 205 290 316 393>]

[couldn---<33 34>]

[counter---<243>]

[country---<245 253 263>]

[county---<406>]

[course---<447>]

[courtyard---<96>]

[crazy---<30>]

[crew---<242>]

[crooks---<159>]

[crouch---<343>]

[crouched---<54 350>]

[crowd---<25 35 37 298 303 322 331>]

[crutches---<184>]

[crying---<56>]

[cuadrilla_---<302>]

[curse---<317>]

[cursed---<320>]

[curve---<319>]

[curving---<321>]

[cushions---<299>]

[cut---<304>]

[d---<78 361>]

[damn---<354>]

[dancers---<372>]

[dancing---<341 344 357>]

[dangerous---<14>]

[dark---<10 13 175>]

[dates---<198>]

[day---<78 114 140 220>]

[dead---<96 118 119 156 397>]

[dear---<135 138>]

[delighted---<245>]

[della---<247>]

[derby---<425>]

[di---<213>]

[did---<141 162 206 226 249 268 446>]

[didn---<351>]

[different---<447>]

[difficult---<445>]

[difficulty---<124>]

[dim---<191>]

[dirty---<114>]

[disappointing---<125>]

[disgusted---<362 423>]

[do---<136 441>]

[doctor---<389 390>]

[don---<354>]

[done---<157 254>]

[door---<414>]

[down---<36 38 69 100 103 122 176 283 298 301 316 340 343 344 347 349 350 389 441>]

[downstairs---<99 362>]

[downward---<115>]

[dragged---<111>]

[drevetts---<155>]

[drevitts---<152 162>]

[drew---<325>]

[drink---<206>]

[drop---<427 431>]

[dropped---<423>]

[drove---<152>]

[drums---<340>]

[drunk---<9 11 30 307 345>]

[drunkards---<371>]

[dully---<330>]

[duomo---<190>]

[during---<184>]

[duty---<180>]

[each---<321 382>]

[eager---<271>]

[early---<253>]

[eat---<269>]

[eating---<243>]

[ed---<281>]

[either---<407>]

[end---<48>]

[enema---<182>]

[enough---<192>]

[entrails---<283>]

[equipment---<68 115>]

[evacuation---<57>]

[evening---<173>]

[everybody---<9>]

[everyone---<137 194 263>]

[everything---<49 262 264 394 396>]

[expected---<221>]

[face---<113 114 115 306 381 386>]

[fainted---<31>]

[fair---<228>]

[fall---<86>]

[fallen---<117>]

[faster---<396>]

[father---<358>]

[fed---<243>]

[feet---<321>]

[fell---<68 431>]

[felt---<193 210 381 382 384 393>]

[few---<186 447>]

[fields---<11>]

[fifes---<340>]

[fifteen---<198>]

[fifteenth---<152 153>]

[fifty---<15>]

[fighter---<308>]

[film---<397>]

[finally---<36 101 217 300>]

[fine---<85>]

[finished---<210>]

[fire---<16 112>]

[fired---<102>]

[firm---<321>]

[firmly---<318>]

[first---<25 66 102>]

[five---<32 34 100 408 409>]

[flank---<85>]

[flat---<134 411>]

[flats---<46>]

[flopped---<320>]

[flopping---<385>]

[folded---<301>]

[folds---<326>]

[following---<319>]

[fool---<354>]

[for---<31 47 180 181 192 224 269 328 343 350 351 352 389 408 450>]

[ford---<153>]

[forgive---<220>]

[forty---<83>]

[forward---<120 271 284 287 288>]

[fossalta---<133>]

[found---<156 393>]

[four---<416>]

[francesca---<247>]

[fresh---<177>]

[friend---<182>]

[friends---<206>]

[frightened---<155 410>]

[frightfully---<78 85 445>]

[from---<15 66 80 83 117 136 153 185 207 213 228 238 242 289 300 321 325 357 382 390>]

[front---<15 80 190 289 316 325>]

[full---<342>]

[funny---<16>]

[further---<69 139>]

[gallows---<414>]

[garden---<65 67 69 439 440>]

[gate---<387>]

[genoa---<213>]

[german---<66>]

[get---<33 82 134 135 185 192 200 203 207 424>]

[gets---<366>]

[getting---<120 136 394>]

[giotto---<247>]

[girl---<55 142 218 223 228>]

[glad---<180 439>]

[go---<203 252 262 266 347 384 421 444 451>]

[going---<9 10 16 120 160 258 392>]

[gone---<86 386>]

[gonorrhea---<228>]

[good---<35 102 204 209 210 211 252 266 272 308 443 445>]

[got---<25 30 67 151 155 175 184 197 227 351 395>]

[grabbed---<304 305 349 357>]

[grand---<152 388>]

[grandstand---<392>]

[grateful---<220>]

[grating---<80>]

[great---<223 392 448>]

[greek---<53>]

[greeks---<450>]

[grounds---<444>]

[guards---<419 422 424>]

[gun---<112>]

[had---<32 68 86 111 113 117 182 193 204 209 217 218 239 242 246 252 254 273 384 391 408 416 423 446>]

[half---<94>]

[halls---<187>]

[hand---<25 27 331>]

[hands---<391 411>]

[hanged---<405 408 409>]

[hanging---<408>]

[happened---<307 316>]

[hard---<97 352>]

[hardly---<34>]

[has---<307>]

[hat---<425>]

[hates---<347>]

[hating---<325>]

[hauled---<282>]

[hauling---<48>]

[have---<32 81 157 185 223 262 442 447>]

[having---<55>]

[he---<27 29 31 33 34 36 67 68 100 103 113 134 141 142 156 173 176 182 183 184 185 187 190 197 198 203 204 205 206 210 213 219 223 228 237 241 242 245 246 247 249 251 253 255 258 262 266 268 270 271 281 284 285 286 289 306 307 318 320 324 325 343 344 345 347 349 351 352 356 357 361 364 366 381 382 384 390 391 393 397 416 439 440 443 445 446 450>]

[head---<103 121 124 381 383 411 412 422>]

[headquarters---<238>]

[hear---<176>]

[heard---<85 272 340>]

[heavy---<80 428>]

[held---<27 37 272 287 304 419>]

[hell---<156 157 160>]

[her---<56 180 205 208 220 226 227>]

[herded---<53>]

[here---<135 239 262>]

[high---<406>]

[him---<11 26 28 30 35 37 67 98 99 102 111 173 181 197 200 201 222 224 240 242 243 251 255 267 269 273 285 287 302 304 306 317 320 325 342 347 349 361 383 384 385 387 419 420 421 423 444>]

[himself---<183 281>]

[his---<11 25 31 34 65 103 113 114 115 121 124 188 206 270 285 287 288 290 301 302 304 318 321 331 332 357 366 381 383 386 391 410 411 412 419 421 422>]

[hit---<113>]

[hits---<318>]

[hold---<99>]

[holding---<55 183 423>]

[hollered---<37>]

[home---<203 204 208>]

[hooted---<26>]

[hoped---<223>]

[horn---<25 27 29 382 383>]

[horse---<11 281 283 289>]

[horses---<391>]

[horthy---<254>]

[hospital---<95 97 199 214>]

[hot---<78 114 140 173 178>]

[hotel---<354 360>]

[house---<80 116 119>]

[houses---<243>]

[how---<162 200 201 258 424 441>]

[hunched---<343>]

[hung---<27 117 283>]

[hungarians---<151 153>]

[i---<11 12 17 66 136 137 165 251 252 260 267 269 272 305 307 340 342 349 354 357 360 361 364 369 445>]

[if---<136 316 361 446>]

[ignorant---<364>]

[impossible---<200>]

[in---<1 10 13 16 29 34 36 46 54 65 94 95 100 103 112 113 118 120 123 137 140 151 153 173 174 178 188 198 206 209 215 216 221 224 226 229 237 238 240 243 246 248 255 256 258 268 269 273 283 287 316 324 325 343 381 386 392 405 408 409 411 414 416 425 439 448>]

[including---<415>]

[indelible---<238>]

[infirmary---<388>]

[inside---<361>]

[instead---<241>]

[into---<11 38 69 99 151 190 251 267 283 298 384>]

[iron---<80 117>]

[is---<14 160 258 262 263 440 445 448>]

[it---<14 16 35 36 56 78 79 81 82 83 84 97 102 120 157 175 186 191 194 195 200 201 205 214 222 224 227 245 252 253 255 262 263 305 307 316 318 324 329 344 384 395 450>]

[italians---<217>]

[italy---<237 245 255 258>]

[jail---<273 406>]

[jammed---<47 52 78>]

[jerkily---<286>]

[jesus---<134 135 138 142>]

[jimmy---<156>]

[job---<203 204 207>]

[joke---<182>]

[jolly---<450>]

[jumped---<344 356>]

[just---<49 70 328 364 431>]

[karagatch---<47>]

[keep---<136>]

[keeping---<50 299>]

[kept---<10 13>]

[kerensky---<446>]

[kicked---<287>]

[kid---<31 55 305>]

[kids---<53>]

[kill---<32 324 366 372>]

[killed---<136 302>]

[kills---<371>]

[kilometers---<15>]

[kilt---<329>]

[kind---<246>]

[king---<439 442>]

[kissed---<210>]

[kitchen---<14 16 17>]

[knee---<281>]

[kneeling---<430>]

[knees---<103 301>]

[knew---<186 194 219 224>]

[knocking---<133>]

[know---<163>]

[known---<217>]

[laid---<388>]

[lance---<288>]

[larger---<394 395>]

[last---<33 272>]

[lay---<29 115 118 123 134 301 381 411>]

[leaned---<288 302>]

[leather---<299>]

[leave---<358>]

[leaves---<96>]

[left---<429>]

[leg---<67>]

[legs---<112 281 285 289 318 332 419>]

[let---<180>]

[letter---<218 227>]

[letters---<197>]

[liable---<157>]

[lieutenant---<10>]

[lift---<34 80>]

[like---<30 35 70 211 249 269 307 318 416 450>]

[liked---<187>]

[lincoln---<229>]

[line---<139>]

[listen---<352>]

[listening---<352>]

[little---<255>]

[living---<215>]

[ll---<136 137>]

[loaded---<49>]

[lonely---<214>]

[long---<450>]

[look---<174>]

[looked---<35 68 116 122>]

[looking---<56 271 324 331 360>]

[loose---<350 357>]

[lose---<195>]

[lost---<422>]

[lot---<157>]

[love---<216 223>]

[loved---<200 222>]

[luis---<351 354>]

[m---<11>]

[machine---<112>]

[machines---<54>]

[made---<36 122 216>]

[maera---<342 347 360 366 371 381 386 389 392 393>]

[magyar---<254>]

[major---<216 226>]

[make---<102 160 194 290>]

[man---<252 420 425 445>]

[mantegna---<249 269>]

[mantegnas---<268>]

[many---<197 246 247>]

[maritza---<51>]

[married---<192 194 204 207 221>]

[marry---<226>]

[masaccio---<247>]

[matador---<25 26>]

[matadors---<33>]

[matters---<138>]

[mattresses---<54>]

[maybe---<162>]

[me---<122 134 135 136 251 270 347 352 357 358 361 439 444>]

[meet---<205>]

[men---<30 50 242 254 304 386 389 407 408 409 410 447>]

[mestre---<141>]

[mexican---<364>]

[might---<204 220 447>]

[milan---<173 208>]

[milano---<266 268 269>]

[mile---<165>]

[miles---<47>]

[minarets---<46>]

[mind---<270 290>]

[ministers---<94 98>]

[mirrors---<54>]

[missing---<201>]

[moment---<328>]

[mon---<12>]

[money---<243>]

[monos_---<284 286>]

[mons---<65>]

[months---<180>]

[more---<32 69>]

[morning---<94 140 151 405 417>]

[mosta---<214>]

[moved---<139>]

[movement---<258 326>]

[moving---<51>]

[much---<68 200 239 246 270 300>]

[mud---<46 48>]

[muddy---<216>]

[muggy---<140>]

[muleta---<319 320 321 326>]

[muscle---<422>]

[music---<343 350 352 356>]

[must---<14>]

[my---<13 16 358 420>]

[nailed---<97>]

[narrow---<406>]

[near---<273>]

[necessary---<252>]

[neck---<302>]

[negroes---<410>]

[neither---<193>]

[nervously---<289>]

[never---<143 197 217 227>]

[new---<205>]

[next---<141>]

[nice---<254>]

[nick---<111 116 121 124>]

[night---<13 141 178 180 201>]

[no---<48 49 102 242 268>]

[not---<124 141 183 185 192 195 204 206 208 210 219 226 249 268 290 308 358 393 444 448>]

[now---<121 222>]

[o---<151 405 416>]

[oak---<318 428>]

[observed---<15>]

[obstacle---<84>]

[occupied---<407>]

[of---<46 80 95 96 97 98 100 111 112 116 119 134 154 155 157 174 187 193 215 216 238 241 248 255 263 264 270 273 285 286 298 301 316 326 329 342 389 406 407 409 410 415 422 424 427 428 429 447 448>]

[off---<154 165 304>]

[officer---<101>]

[officers---<83 84>]

[oh---<12 134 358 441>]

[oilcloth---<238>]

[old---<50 79>]

[on---<27 52 68 84 96 103 114 139 177 180 181 183 184 207 213 237 242 266 281 285 350 354 360 381 389 392 407 410 411 427 428>]

[once---<209 383>]

[one---<27 67 98 154 155 173 242 263 286 301 304 328 389 410 421 424>]

[oneself---<448>]

[only---<136 138 186 207 218 222 383>]

[onto---<173 414 431>]

[open---<214>]

[operated---<181>]

[operating---<181>]

[opposite---<117>]

[or---<35 182 206 226 415>]

[ordered---<442>]

[other---<28 100 119 191 226 411>]

[others---<176 390>]

[oughtn---<156>]

[our---<1>]

[out---<11 14 26 29 31 46 83 85 99 112 117 134 135 154 155 174 175 269 302 325 330 360 387 389 414>]

[outside---<444>]

[over---<37 56 66 67 69 81 82 174 267 271 272 302 303 329 421>]

[overhead---<392>]

[owned---<49>]

[padova---<207 209>]

[palace---<444>]

[park---<229>]

[party---<238>]

[pass---<267 271 272>]

[passage---<388>]

[passed---<242>]

[past---<94>]

[patients---<186>]

[patriots---<124>]

[patrol---<65>]

[paving---<96>]

[peace---<123>]

[pencil---<239>]

[people---<191 246>]

[perfect---<78 84>]

[picador---<282 287 391>]

[picked---<386>]

[pictures---<247>]

[pieces---<133 298>]

[piero---<247>]

[pigtail---<304>]

[pink---<116>]

[pipes---<341>]

[place---<28>]

[plastiras---<444>]

[pleasant---<253>]

[please---<135 138>]

[point---<263>]

[police---<153>]

[pools---<95>]

[potted---<67 82>]

[prayed---<134 190>]

[praying---<191>]

[prepared---<181>]

[priceless---<79>]

[priest---<421 430>]

[priests---<415 420 430>]

[probably---<219>]

[procession---<53>]

[puddle---<100>]

[puked---<37>]

[pulled---<282 357>]

[pumped---<288>]

[puntillo_---<303>]

[put---<14 85>]

[quarrel---<210>]

[quarrelled---<208>]

[quartered---<215>]

[queen---<440>]

[quiet---<35 141 191>]

[quietly---<101>]

[quite---<241 306>]

[railroads---<237>]

[railway---<243>]

[rain---<46 99>]

[rained---<56 97>]

[rainy---<214 216>]

[rammed---<28>]

[ran---<305>]

[re---<159 358>]

[read---<199>]

[realized---<222>]

[really---<308>]

[red---<329>]

[regularly---<289>]

[reported---<251>]

[reproductions---<248>]

[requesting---<240>]

[revolution---<256>]

[revolutionary---<443>]

[riau_---<372>]

[riding---<11 13 228>]

[right---<162 316 345 373 446>]

[rinaldi---<114 122 123 125>]

[ring---<38 299>]

[river---<66>]

[road---<9 12 17 47>]

[roaring---<322 331>]

[rods---<285>]

[romagna---<251>]

[roof---<117 173>]

[rose---<441>]

[rossa---<142>]

[rubble---<118>]

[run---<387 396>]

[running---<51 390>]

[rush---<324>]

[rushed---<83>]

[s---<157 162 254 283 289 330 364>]

[saddle---<283>]

[said---<156 159 162 165 245 260 262 266 268 307 347 350 354 358 364 366 369 371 420 425 440 441 443 445>]

[sake---<351>]

[sales---<228>]

[sam---<405 415 422 429>]

[same---<324 326>]

[sand---<29 36 381 384>]

[sat---<36 100 111 177 410 441>]

[savage---<364>]

[savages---<371>]

[saw---<66 306 342 361>]

[say---<137 209 393>]

[saying---<11 14 211 239>]

[scaffolding---<427 431>]

[scared---<56>]

[searchlights---<175>]

[seat---<154>]

[second---<26>]

[see---<206 252 316 361 439>]

[seemed---<439>]

[seen---<246>]

[senta---<122>]

[sentenced---<408>]

[separate---<123>]

[september---<253>]

[seven---<415>]

[sewing---<54 391>]

[shade---<119>]

[she---<177 181 200 217 219 221 222 223 224 440 441>]

[shelling---<139>]

[shone---<114>]

[shook---<288>]

[shoot---<81>]

[shooting---<446>]

[short---<227 306>]

[shot---<70 94 154 446 448>]

[should---<203>]

[shoulders---<330>]

[shouted---<298>]

[shouting---<392>]

[shut---<97>]

[shutters---<97>]

[shy---<241 254>]

[shyly---<268>]

[sick---<56 98 210>]

[side---<407>]

[sighted---<326>]

[silly---<184>]

[simply---<79>]

[since---<416>]

[sion---<273>]

[sitting---<103 429>]

[six---<94 405 415>]

[skipped---<430>]

[sky---<175>]

[slipped---<26>]

[slug---<30>]

[smaller---<394 395 396>]

[smiling---<125>]

[snarl---<317>]

[so---<12 33 68 183 185 195 203 300 349 352>]

[soaked---<50>]

[soda---<442>]

[soldiers---<98 101>]

[solid---<52>]

[some---<220 254 304 386>]

[someone---<384>]

[something---<393>]

[sometimes---<383>]

[son---<420>]

[sorry---<219>]

[sort---<448>]

[sorted---<198>]

[soused---<12>]

[speed---<397>]

[sphincter---<422>]

[spine---<113>]

[spite---<255>]

[spoke---<267>]

[sprawling---<115>]

[spring---<222 226>]

[spurs---<288>]

[square---<237>]

[stand---<102 388>]

[standing---<329>]

[start---<352>]

[started---<324 344 356 357 387>]

[starting---<263>]

[states---<207 218>]

[station---<153 209>]

[stayed---<180>]

[steel---<428>]

[stepped---<427>]

[sthetic---<183>]

[sticking---<301 330>]

[sticky---<382>]

[stiff---<286>]

[still---<123 381>]

[stirrups---<282>]

[stood---<100 390>]

[stop---<391>]

[stopped---<286 342>]

[store---<151>]

[straight---<116 199 282 325 329 331>]

[strapped---<429>]

[strapping---<419>]

[street---<112 118 119 152 153 340 341 343 345>]

[stretcher---<120>]

[stuck---<46 112>]

[suffered---<239>]

[sun---<114 123 140>]

[suppose---<369>]

[sure---<263>]

[surprised---<68>]

[swearing---<385>]

[sweated---<134>]

[sweatily---<125>]

[sweaty---<113>]

[swifts---<174>]

[swing---<321>]

[swiss---<273>]

[switzerland---<267>]

[sword---<25 31 33 319 325 330>]

[swung---<284 318 320 428>]

[t---<32 33 34 156 159 351 354>]

[table---<181 442>]

[tail---<385>]

[take---<185>]

[talk---<393>]

[talked---<255 450>]

[talky---<184>]

[taxicab---<229>]

[tell---<12 137 141 165>]

[temperature---<185>]

[terrible---<201>]

[than---<32>]

[thanked---<270>]

[that---<16 17 70 137 138 162 183 197 211 218 263 307 416>]

[the---<9 10 11 12 13 15 16 25 26 27 28 29 30 31 33 34 36 37 38 46 47 48 49 50 51 52 53 54 57 66 69 79 80 85 94 95 96 97 98 99 100 101 102 103 111 112 113 114 115 116 117 118 119 120 123 133 137 138 139 140 141 142 151 153 154 155 160 173 174 175 176 177 178 181 182 184 185 186 187 190 192 197 198 199 203 206 207 209 210 215 216 218 221 224 226 228 237 238 240 242 243 245 251 253 256 258 262 263 266 267 268 269 271 272 273 281 282 283 284 285 286 287 288 289 290 298 299 300 301 302 303 306 317 318 319 320 321 322 324 325 326 327 328 329 330 331 340 341 342 343 344 350 352 354 356 360 371 372 381 382 383 384 385 386 387 388 389 390 392 405 406 407 408 409 410 411 414 416 420 421 422 424 427 429 430 431 439 440 442 443 444 447>]

[their---<84 154>]

[theirs---<218>]

[them---<53 70 82 163 176 177 193 198 199 248 342 343 345 349 372 373 415>]

[then---<30 36 67 69 267 299 320 329 340 341 342 356 386 394 395 396 397>]

[there---<55 95 96 157 174 186 191 192 214 392 414 429>]

[they---<37 49 70 81 82 83 94 99 102 111 156 159 163 173 180 181 182 186 187 190 191 193 194 195 199 203 204 208 209 243 281 328 341 342 344 385 388 397 405 410 414 415 419 421 423 427>]

[thing---<138 448>]

[things---<38 119 255 447>]

[think---<446>]

[thirty---<47>]

[this---<162 241 351 440 448>]

[those---<446>]

[though---<193 446>]

[thought---<187>]

[three---<32 69 180 409>]

[threw---<38 298>]

[through---<25 27 48 50 52 57 81 199 229 384 387 414 440>]

[ticket---<241>]

[tiers---<407>]

[tight---<28 183 319 429>]

[till---<67>]

[time---<1 121 162 184 192 226 227 298 382 450>]

[times---<34>]

[tired---<33 300>]

[to---<10 11 27 30 32 51 80 81 82 86 99 102 111 133 139 157 160 180 183 185 190 192 194 200 203 205 206 207 208 209 213 214 217 218 219 220 221 227 240 242 252 255 266 267 269 271 272 284 290 324 326 350 352 354 360 387 388 391 393 396 408 409 420 421 439 444 448 451>]

[together---<253 319 419>]

[told---<101 143 443>]

[too---<80 300>]

[took---<176 251>]

[top---<174 409>]

[topping---<82>]

[torero---<303>]

[toro---<327>]

[torre---<213>]

[toward---<118 387 421>]

[town---<120 174 215 216>]

[towns---<246>]

[trailing---<319>]

[train---<207 242 266>]

[travelling---<237>]

[tree---<442>]

[trench---<133 139>]

[tried---<30 34 82 99>]

[trip---<252>]

[trouble---<157 160>]

[trying---<102>]

[turned---<121 124>]

[twisted---<118 282 356>]

[two---<98 118 151 304 415 419 420 430>]

[typhoid---<98>]

[under---<182 239 388 442>]

[understand---<220>]

[understood---<205>]

[unexpectedly---<221>]

[until---<197 204>]

[up---<13 30 46 51 66 99 102 119 139 140 152 173 185 251 282 283 290 299 331 344 349 356 386 391 397 419 423>]

[upstairs---<142>]

[us---<266>]

[used---<185 241>]

[ve---<122 351>]

[very---<85 101 114 134 239 241 254 260 268 270 271 306 410 428 439 445 450>]

[vieux---<12>]

[villa---<142>]

[villalta---<317 327 328 329 330 332>]

[volley---<103>]

[wagon---<154 155>]

[waited---<67>]

[waiting---<350>]

[walk---<267 272>]

[walked---<50 187 246 287 440>]

[walking---<271>]

[wall---<29 67 70 95 100 101 111 115 116 414>]

[want---<206>]

[wanted---<192 194 393 451>]

[warm---<382>]

[was---<9 16 17 33 35 51 55 78 79 82 84 98 102 103 113 114 120 125 133 140 177 191 192 200 201 205 214 215 219 222 224 237 239 241 245 252 253 270 271 289 300 305 306 324 329 345 349 352 360 386 392 397 406 428 429 439 440 450>]

[wash---<391>]

[water---<48 95 100 103>]

[waving---<305>]

[way---<240 384 388>]

[we---<10 12 15 65 67 70 78 82 85 86 122 139 252 369 371 372 439 441 442 450>]

[weather---<272>]

[well---<120 364>]

[went---<12 139 142 176 182 190 213 344 349 360 361 383 389>]

[were---<10 15 47 48 52 53 65 85 95 96 97 119 154 156 163 174 180 186 191 194 199 210 246 385 407 409 410 414 415 419 420 423>]

[wet---<96>]

[whack---<281>]

[whacked---<281>]

[whacking---<285>]

[wham---<28>]

[when---<17 85 102 156 163 181 209 317 318 324 342 344 361 396 421 427>]

[where---<111 252 269 390>]

[which---<427>]

[while---<37 133 175 272 349 419>]

[whiskey---<442 443>]

[whispering---<420>]

[whistling---<300>]

[white---<281 410>]

[whites---<240>]

[who---<159 239 366 423>]

[whole---<9>]

[will---<14 262 263 366 424>]

[willing---<208>]

[wind---<318>]

[wine---<299>]

[winter---<216>]

[with---<27 49 52 54 55 65 98 103 123 142 176 210 245 251 285 302 305 306 326 328 343 345 349 383 387 406 411 421>]

[without---<200>]

[wobbly---<290>]

[woman---<55>]

[women---<50 53>]

[wops---<159 163 165>]

[work---<139>]

[worked---<84>]

[working---<439>]

[world---<137 256>]

[worried---<16>]

[would---<81 121 183 185 204 205 219 223 444>]

[wrapped---<248 412>]

[written---<238>]

[wrote---<197 217 269>]

[wrought---<80>]

[yards---<83>]

[yelled---<31>]

[yelling---<300>]

[yellow---<51>]

[yes---<366 371 372 373>]

[york---<205>]

[you---<12 14 32 81 122 136 137 138 156 162 163 262 316 347 351 358 441>]

[young---<55 65 241>]

[younger---<429>]

任务三

—

```
public class MinHeap {
    private static final int DEFAULT_CAPACITY = 10; //  默认大小
```

```
    private int currentSize; //  当前堆的大小
    private int[] array; //  堆数组

    public MinHeap() {
        this.array = new int[DEFAULT_CAPACITY + 1];
        currentSize = 0;
    }

    public MinHeap(int size) {
        this.array = new int[size + 1];
        currentSize = 0;
    }

    public MinHeap(int[] array) {
        this.array = new int[array.length + 1];
        for (int i = 0; i < array.length; i++) {
            this.array[i + 1] = array[i];
        } //  从 1 开始计算
        currentSize = array.length;
        buildHeap();
    }
```

```
    /**
     * 往堆中插入元素
     *
     * @param number 元素值
     */
    public void insert(int number) {
        try {
            if (isFull()) {
                throw new Exception("Array is full!");
            }
            int temp = ++currentSize;
            array[temp] = number; // 将 number 放在数组最后
            while ((temp != 1) && (array[temp] <
array[getParent(temp)])) {
                swap(array, temp, getParent(temp)); // 比父结点
值小则与其交换
                temp = getParent(temp);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
    public int findMin() {
        return array[1];
    }

    /**
     * 删除堆顶元素
     */
    public void deleteMin() {
        try {
            if (isEmpty()) {
                throw new Exception("Array is empty!");
            } else {
                swap(array, 1, currentSize--); // 将堆顶元素放到
最后
                if (currentSize != 0) siftDown(1);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
    /**
     * 大的值下沉
     *
     * @param pos 当前位置
     */
    private void siftDown(int pos) {
        try {
            if (pos < 0 || pos > currentSize) {
                throw new Exception("Illegal position!");
            }
            while (!isLeaf(pos)) {
                int j = getLeft(pos);
                if ((j < currentSize) && (array[j] > array[j + 1])) j++;
// 找子树中的最小值
                if (array[pos] <= array[j]) return; // 如果当前值已
经比子树值小，则返回
                swap(array, pos, j);
                pos = j;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
    }

    public void print() {
        for (int i = 1; i <= currentSize; i++) {
            System.out.print(array[i] + " ");
        }
    }

    public void heapSort() {
        while (currentSize != 0) {
            System.out.print(findMin() + " ");
            deleteMin();
        }
    }

    private boolean isEmpty() {
        return currentSize == 0;
    }

    private boolean isFull() {
        return currentSize == array.length - 1;
    }
```

```
private boolean isLeaf(int i) {
        return i > currentSize / 2;
}

private void buildHeap() {
        for (int i = currentSize / 2; i > 0; i--) {
                siftDown(i); //  对每个非叶子结点进行下沉操作
        }
}

private int getLeft(int i) {
        return 2 * i;
}

private int getRight(int i) {
        return 2 * i + 1;
}

private int getParent(int i) {
        return i / 2;
}
```

```
    private void swap(int[] array, int x, int y) {
        int temp = array[y];
        array[y] = array[x];
        array[x] = temp;
    }

}
```

二

```
public class MinHeapSort {

    public static void heapSort(int[] array) {
        MinHeap minHeap = new MinHeap(array); //  建立小顶堆
        int n = array.length;
        for (int i = 0; i < n; i++) {
            array[i] = minHeap.findMin(); //  获取堆顶最小元素
            minHeap.deleteMin(); //  删除堆顶
        }
    }

    public static void main(String[] args) {
        int[] test = {7, 5, 6, 4, 2, 1, 3};
```

```
        System.out.println("Before sorting:");
        for (int num : test) {
            System.out.print(num + " ");
        }

        heapSort(test);

        System.out.println("\nAfter sorting:");
        for (int num : test) {
            System.out.print(num + " ");
        }
    }
}
```

三

```
class TernaryMinHeap {
    private static final int DEFAULT_CAPACITY = 10; // 默认大小
    private int currentSize; // 当前堆的大小
    private int[] array; // 堆数组

    public TernaryMinHeap() {
        this.array = new int[DEFAULT_CAPACITY + 1];
        currentSize = 0;
```

```
}

public TernaryMinHeap(int size) {
        this.array = new int[size + 1];
        currentSize = 0;
}

public TernaryMinHeap(int[] array) {
        this.array = new int[array.length + 1];
        for (int i = 0; i < array.length; i++) {
                this.array[i + 1] = array[i];
        } //  从 1 开始计算
        currentSize = array.length;
        buildHeap();
}

/**
 * 向堆中插入元素
 *
 * @param number  元素值
 */
public void insert(int number) {
```

```
            try {
                if (isFull()) {
                    throw new Exception("Array is full!");
                }
                int temp = ++currentSize;
                array[temp] = number; //  将 number 放在数组最后
                while ((temp != 1) && (array[temp] <
array[getParent(temp)])) {
                    swap(array, temp, getParent(temp)); //  比父结点
值小则交换
                    temp = getParent(temp);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        public int findMin() {
            return array[1];
        }

        /**
```

```
     * 删除堆顶元素
     */
    public void deleteMin() {
        try {
            if (isEmpty()) {
                throw new Exception("Array is empty!");
            } else {
                swap(array, 1, currentSize--); // 将堆顶元素放到
最后
                if (currentSize != 0) siftDown(1);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * 小的值下沉
     *
     * @param pos 当前位置
     */
    private void siftDown(int pos) {
```

```
        try {
            if (pos < 0 || pos > currentSize) {
                throw new Exception("Illegal position!");
            }
            while (!isLeaf(pos)) {
                int smallest = getLeft(pos); // 假设左子结点最小
                int mid = getMiddle(pos);
                int right = getRight(pos);

                // 比较三个子结点求最小值
                if (mid <= currentSize && array[mid] <
array[smallest]) {
                    smallest = mid;
                }
                if (right <= currentSize && array[right] <
array[smallest]) {
                    smallest = right;
                }

                if (array[pos] <= array[smallest]) return; // 已满足
堆序
                swap(array, pos, smallest);
```

```
                pos = smallest;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private boolean isEmpty() {
        return currentSize == 0;
    }

    private boolean isFull() {
        return currentSize == array.length - 1;
    }

    private boolean isLeaf(int i) {
        return i > currentSize / 3;
    }

    private void buildHeap() {
        for (int i = currentSize / 3; i > 0; i--) {
            siftDown(i); //  对每个非叶子结点进行下沉操作
```

```
        }
    }

    private int getLeft(int i) {
        return 3 * i - 1;
    }

    private int getMiddle(int i) {
        return 3 * i;
    }

    private int getRight(int i) {
        return 3 * i + 1;
    }

    private int getParent(int i) {
        return (i + 1) / 3;
    }

    private void swap(int[] array, int x, int y) {
        int temp = array[y];
        array[y] = array[x];
```

```
            array[x] = temp;
        }
    }

public class TernaryHeapSort {
    public static void heapSort(int[] array) {
        TernaryMinHeap minHeap = new TernaryMinHeap(array); //
建立三叉小顶堆
        int n = array.length;
        for (int i = 0; i < n; i++) {
            array[i] = minHeap.findMin(); //  获取堆顶最小元素
            minHeap.deleteMin(); //  删除堆顶
        }
    }

    public static void main(String[] args) {
        int[] test = {7, 5, 6, 4, 2, 1, 3};
        System.out.println("Before sorting:");
        for (int num : test) {
            System.out.print(num + " ");
        }
```

```
        heapSort(test);


        System.out.println("\nAfter sorting:");
        for (int num : test) {
            System.out.print(num + " ");
        }
    }
}
```