|  |  |
| --- | --- |
| **数据结构与算法 实验报告** | |
| 第三次 | |
| 图片包含 标牌  已生成极高可信度的说明 | |
|  | |
|  | |
|  | |
| **姓名** | 宋郅博 |
| **班级** | 强基数学2102 |
| **学号** | 221311457 |
| **电话** | 13561313919 |
| **Email** | 2105602479@qq.com |

任务一 实现 BST 数据结构

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

首先，创建了BST.java来定义操作，然后创建BSTImpl.java，来实现上述的操作，最后创建Test.java来导入BST\_testcases.txt文件，通过每一次导入一行再对这一行的首字符的区分（例如+、#）来进行不同的操作并导出结果。导出至hw\_3\work\_1\BST\_result1.txt中。

在导出文件中，与BST\_result.txt 文件相比，每一次的showStructure操作结果相同。.但有一个问题，有少量的操作不同，我发现少量的remove操作中单词的翻译出现了错误。

任务二 使用 BST 为文稿建立单词索引表

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

只需简单复用第一问的二叉树，将单词本身作为 key，将单词出现的行数储存到一个 list 数组中，作为节点的 value 即可。因此我们将work\_1中的BST和BSTImpt文件复制过来，并创建dictionary.java来导入hw\_3\work\_2\article.txt，处理每一行并用空格来分词，并计算出现的单词并记录行号，导出至hw\_3\work\_2\index\_result.txt。

dictionary.java代码如下：

package work\_2;

import java.io.\*;

import java.util.\*;

public class dictionary {

    public static void main(String[] args) {

        BST<String, List<Integer>> bst = new BSTImpl<>();

        try (BufferedReader reader = new BufferedReader(new FileReader("C:\\Users\\Thomas\\Desktop\\hw\_3\\work\_2\\article.txt"));

             PrintWriter writer = new PrintWriter("index\_result.txt")) {

            String line;

            int lineNum = 0;

            while ((line = reader.readLine()) != null) {

                lineNum++;

                String[] words = line.split("\\s+");

                for (String word : words) {

                    if (word.isEmpty()) continue;

                    word = word.replaceAll("[^a-zA-Z']", "").toLowerCase();

                    if (word.isEmpty()) continue;

                    List<Integer> lines = bst.search(word);

                    if (lines == null) {

                        lines = new ArrayList<>();

                        bst.insert(word, lines);

                    }

                    if (!lines.contains(lineNum)) {

                        lines.add(lineNum);

                    }

                }

            }

            bst.printInorder(writer);

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

任务三 学会使用堆

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

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

2）完成一个使用 1）编写的 Min\_Heap 的排序算法；

3）上课讲解的堆是二叉树的一个应用，因此也称为二叉堆。如果将堆的概念扩展到三叉树（树中每个结点的子结点数最多是 3 个），此时将形成三叉堆。除了树形和二叉堆不一致外，堆序的要求是完全一致的。使用完全三叉树形成堆，在 n 个结点下显然会降低整棵树的高度，但是在维持某个结点的堆序时需要比较的次数会增加（这个时候会是 3 个子结点之间进行互相比较求最大或最小），这也是一个权衡问题。现在要求将 1）中实现的二叉堆改造成三叉堆，并通过 2）中实现的排序验证这个三叉堆是否正确。

3.1 问题一：二叉堆

在hw\_3\work\_3\MinHeap.java创建了MinHeap数据结构，以下是此结构中的操作：

Swap：交换两个节点的数值。

Swim：不断比较当前节点与其父节点的值，如果当前节点的值小于父节点的值，则使用Swap来交换它们的位置。

Sink：接受一个值，将其插入到最小堆中并将其与子节点中的最小值进行对比，若当前节点的值小于或等于其子节点中的最小值则停止操作。

Insert：使用Swim操作来插入此节点。

Deletemin：由于堆顶代表着最小元素，只需将堆顶节点删除再替换堆顶元素即可，返回已删除的堆顶节点值。

3.2 问题二：二叉堆排序

在1）中的代码里加上以下代码即可排序，

//生成一个随机数组进行排序

    public static int[] randomArray(int size){

        int[] array = new int[size];

        for(int i = 0; i<array.length;i++){

            array[i] = (int)(Math.random()\*50);

        }

        return array;

    }

    public static void printArray(int[] arr){

        for(int i= 0 ; i<arr.length; i++){

            System.out.print(arr[i] + " ");

        }

        System.out.println();

    }

    public static void main(String arr[]) {

        int[] arr0 = randomArray(50);

        System.out.print("随机数组为：");

        printArray(arr0);

        heapSort(arr0);

        System.out.print("排序结果为：");

        printArray(arr0);

    }

生成了50个随机数，所得结果如下（hw\_3\work\_3\heapsort\_result.txt）：

随机数组为：14 37 13 16 12 10 41 43 45 4 15 31 45 25 9 49 5 36 39 31 38 1 1 33 46 40 47 8 49 17 12 25 30 27 45 40 12 4 19 21 49 7 40 34 41 20 34 14 13 2

排序结果为：1 1 2 4 4 5 7 8 9 10 12 12 12 13 13 14 14 15 16 17 19 20 21 25 25 27 30 31 31 33 34 34 36 37 38 39 40 40 40 41 41 43 45 45 45 46 47 49 49 49

3.3 问题三：三叉堆

三叉堆的数据结构与二叉堆类似，操作与二叉堆相同，但操作内有一些不同。

在操作中，求index的父节点，二叉堆的，在三叉堆中改为。而index的子节点中最左边的节点为。其余改动不大。

数据结构存储在hw\_3\work\_3\tri\_heap.java中。

3.4 问题三：三叉堆排序

与2）类似，创建了长50的数组，排序结果如下：

随机数组为：1 34 47 7 44 29 29 32 4 8 46 34 25 27 8 28 11 8 49 15 9 44 25 13 11 37 11 12 28 16 30 7 0 7 19 17 0 48 20 1 25 10 27 45 37 32 19 14 28 29

排序结果为：0 0 1 1 4 7 7 7 8 8 8 9 10 11 11 11 12 13 14 15 16 17 19 19 20 25 25 25 27 27 28 28 28 29 29 29 30 32 32 34 34 37 37 44 44 45 46 47 48 49

数据存储在（hw\_3\work\_3\tri\_heapsort\_result.txt）。