数据结构与算法
作业报告

第三次

姓名

班级

学号

电话

Email

日期

目录

任务 1：实现 BST 数据结构

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

下面的二叉检索树的 ADT 在描述时，依然使用了模板类的方法，并且在这次描述中，使用了两个模板参数 K 和 V（这表明 key 和 value 可以为不同类型的数据）。在本次任务的测试文件中，key 和 value 都是 String 类型，对模板参数运用有困难的同学，可以直接设定 key 和 value 的类型为 String。

但是请注意：任务 2 的 BST 的 key 和 value 存放的数据类型不同，因此如果在本任务中没有使用模板类构建 BST，那么在任务 2 中则需要将任务 1 中的代码进行重新撰写，这个重写过程应该也能让不使用模板类的同学发现代码复用率的下降。

图 1 为 BST 接口的定义（仅作参考），表 1 为对接口函数的详细说明。

```
public interface BST<K extends Comparable<K>, V> {
    public void insert(K key , V value);
    public V remove(K key);
    public V search(K key);
    public boolean update(K key , V value);
    public boolean isEmpty();
    public void clear();
    public void showStructure(PrintWriter pw) throws IOException;
    public void printInorder(PrintWriter pw) throws IOException;
}
```

图 1 BST 接口的定义

表 2 命令的具体意义说明

命令符号	命令格式	命令举例
+	+(key , value)	+(auspices , "n.资助,赞助") 向二叉检索树中插入一个 key 为 auspices,value 为"n.资助,赞助"的元素，插入的要求遵守 BST 接口中的 insert 方法的定义
-	-(key)	-(morass) 从二叉检索树中删除一个 key 为 morass 的元素，删除的要求遵守 BST 接口中的 remove 方法的定义
?	?(key)	?(hide) 在二叉检索树中查找一个 key 为 hide 的元素，查找的要求遵守 BST 接口中的 search 方法的定义
=	=(key, value)	=(laggard , "laggard") 将二叉检索中对 key 为 laggard 的元素的 value 值更新为“laggard”，更新的要求遵守 BST 接口中的 update 方法的定义
#	#	# 调用二叉检索树的 showStructure 方法

为了测试实现 BST 接口的数据结构是否运行正常，准备了两个文件，一个是 homework3_testcases.txt，一个是 homework3_result.txt 文件。其中，前一个包含了所有的对二叉检索树进行操作的命令内容，后一个则是执行命令文件之后的输出。具体的命令格式如表 2 所示。

图 2 命令和输出格式对应关系

本次输出的内容对每个命令的执行都有输出内容的要求（除了‘+’命令，因为该命令对应的 insert 方法没有返回值），而每个命令的输出格式很规律，结合每个命令对应的相应接口函数的返回值考虑即可。

设计：

一．编写 BST 接口：

按照题目要求的格式输入 BST 接口（下列格式为题目所给）：

```
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.编写 BinarySearchTree 类：

1.对 BinarySearchTree 实现 BST<K,V>接口。

2.编写内部类 Node<K,V>类:

（1）创建 K 类成员变量 key，作为 Node 类对象的键;

（2）创建 V 类成员变量 value，作为 Node 类对象的值;

（3）创建 Node 类对象 left，作为当前对象的左子结点;

（4）创建 Node 类对象 right，作为当前对象的右子结点;

（5）调用有参构造器，给当前 Node 对象的键与值赋值。

3.创建 Node 类对象 root，作为当前二叉检索树的根节点。

4.重写 BST 接口的 insert（K key，V value）方法：

（1）调用 if 语句判断当 key 和 value 均为空（null）时，抛出错误并提示“key and value cannot be null.”

（2）给 root 赋值为 insert（root，key，value）（该方法为上述 insert 方法的重载方法，在下一步讲解）；

（3）重载 insert 方法，接收 Node<K,V>，接收参数为（Node<K,V>,K key，V value）：

①调用 if 语句，判断当 node 为空时，创建一个新节点并将之返回；

②创建 int 类对象 cmp，其值为 key 调用 compareTo（node.key）后的结果；

③调用 if 语句判断 cmp 的大小，若 cmp 大于 0 则对 node 结点的右子节点调用 insert（node.right，key，value）方法；若 cmp 小于 0 则对 node 结点的左子结点调用 insert（node.left，key，value）方法；若 cmp 等于 0，则将 node 的值更新为 value；

④返回 node。

5.重写 BST 接口的 remove（K key）方法：

（1）调用 if 语句判断当 key 为空时，抛出错误并提示“key cannot be null.”

（2）给 root 赋值为 remove（root，key）（该方法为上述 remove 方法的重载方法，在下一步讲解）；

（3）重载 remove 方法，接收 Node<K,V>，接收参数为（Node<K,V> node,K key）：

①调用 if 语句，判断当 node 为空时，返回 null；

②创建 int 类对象 cmp，其值为 key 调用 compareTo（node.key）后的结果；

③调用 if 语句判断 cmp 的大小，若 cmp 大于 0 则对 node 结点的右子节点调用 remove（node.right，key）方法；若 cmp 小于 0 则对 node 结点的左子结点调用 remove（node.left，key）方法；

若 cmp 等于 0，先调用 if 语句判断 node 的左子树是否为空，若是则直接返回 node 的右子树；同样判断 node 的右子树是否为空，是则直接返回 node 的左子树；若左右子树均不为空，创建 Node 类对象 minNode，其值为对 node.right 调用 findMin 方法（后续步骤书写）的结果，并将 node 的 key 及 value 的值均以 minNode 的键和值代替，并对 node 的右子树调用 remove（node.right，minNode.key）方法，通过递归删除右子树中的最小结点；

④返回 node。

6.编写 findMin（Node<K,V> node）方法：

（1）调用 while 循环，判断当 node 结点的左子结点不为空时，使 node=node.left 以递归寻找最小的节点；

（2）while 循环结束后，返回 node。

7.重写 BST 接口的 search（K key）方法：

（1）调用 if 语句判断当 key 为空时，抛出错误并提示“key cannot be null.”

（2）返回 search（root，key）（该方法为上述 search 方法的重载方法，在下一步讲解）；

（3）重载 search 方法，接收 V，接收参数为（Node<K,V> node,K key）：

①调用 if 语句，判断当 node 为空时，返回 null；

②创建 int 类对象 cmp，其值为 key 调用 compareTo（node.key）后的结果；

③调用 if 语句判断 cmp 的大小，若 cmp 大于 0 则对 node 结点的右子节点调用 search（node.right，key）方法；若 cmp 小于 0 则对 node 结点的左子结点调用 search（node.left，key）方法；若 cmp 等于 0，则直接返回 node。

8.重写 BST 接口的 update（K key，V value）方法：

（1）调用 if 语句判断当 key 与 value 均为空时，抛出错误并提示“key and value cannot be null.”

（2）返回 update（root，key，value）（该方法为上述 update 方法的重载方法，在下一步讲解）；

（3）重载 search 方法，接收 boolean 类型变量，接收参数为（Node<K,V> node,K key,V value）：

①调用 if 语句，判断当 node 为空时，返回 false；

②创建 int 类对象 cmp，其值为 key 调用 compareTo（node.key）后的结果；

③调用 if 语句判断 cmp 的大小，若 cmp 大于 0 则对 node 结点的右子节点调用 update（node.right，key，value）方法；若 cmp 小于 0 则对 node 结点的左子结点调用 update（node.left，key，value）方法；若 cmp 等于 0，则更新 node 的值为 value，并返回 true。

9.重写 BST 接口的 isEmpty（）方法：

当 root 为空时返回 true，否则为 false。

10.重写 BST 接口的 clear（）方法：

令 root 等于 null，以清空二叉检索树。

11.重写 BST 接口的 showStructure（PringWriter pw）方法：

（1）在方法名后加上后缀 throws IOException，作为异常声明；

（2）创建 int 类数组 result，其中储存两个值，第一个值为当前二叉检索树的节点个数，第二个值为当前二叉检索树的高度；

（3）调用 showStructure（root,result）方法（showStructure 方法的重载方法），开始遍历二叉检索树；

（4）按照格式对 pw 进行输出，具体如下：

```
pw.println("-----------------------------");
pw.println("There are " + result[0] + " nodes in this BST.");
pw.println("The height of this BST is " + result[1] + ".");
pw.println("-----------------------------");
```

（5）重载 showStructure 方法，接收参数为（Node<K,V> node,K key,int[] result）：

①调用 if 语句判断 node 不为空时，使 result[0]的值加一，并分别对 node 的左右子节点调用 showStructure 方法以遍历其左右子树；

②给 result[1]赋值，其值为 result[1]和 height（node）中的较大值。

12. 编写 height（Node<K,V> node）方法：

（1）node 为空时返回-1；

（2）node 不为空时，创建两个 int 类型变量 leftHeight 和 rightHeight，其值分别为对 node 的左右子节点调用 height 方法后的结果；

（3）返回 leftHeight 和 rightHeight 中的较大值再加一的结果。

13. 重写 BST 接口的 printInorder（PrintWriter pw）方法：

（1）在方法名后加上后缀 throws IOException，作异常声明；

（2）调用 printInorder（root，pw）方法（printInorder 方法的重载方法）；

（3）重载 printInorder 方法，接收参数（Node<K,V> node，PrintWriter pw）：

①对二叉检索树进行中序遍历，输出结果按照以下格式：

```
pw.println("[" + node.key + " ---- < " + node.value + " >]");
```

三．编写 BSTTest 类，用于读取文件并输出理想结果：

编写主方法：

1.创建 BinarySearchTree<String，String>对象 bst；

2.调用 try 方法，创建 BufferedReader 类对象 br，其值为接收了 FileReader 类参数的对象 new FileReader（“testcases.txt”），“testcases.txt”作为读取文本；创建 PrintWriter 类对象 pw，接收参数“result2.txt”，作为输出文本；

3.创建 String 类型对象 line，以行为单位读取文本；

4.调用 while 循环，判断当 line=br.readLine 且不为空时进行以下操作：

（1）调用 if 语句读取该行第一个字符；

（2）第一个字符为“+”时，执行插入操作，读取该行的单词及翻译，并通过对 bst 调用 insert 方法将单词作为键、翻译作为值存入二叉检索树中；

（3）第一个字符为“-”时，执行删除操作，读取该行的单词，通过对 bst 调用 remove 方法将单词作为键传入，成功删除或未找到该单词均有相应输出；

（4）第一个字符为“?”时，执行查找操作，读取该行的单词，通过对 bst 调用 search 方法将单词作为键传入，是否查找到该单词均有相应输出；

（5）第一个字符为“=”时，执行更新操作，读取该行的单词，通过对 bst 调用 update 方法将单词作为键传入，成功更新该单词会有相应输出；

（6）该行为“#”时，对 bst 调用 showStructure（pw）方法，显示当前二叉检索树中的结点个数及树的高度。

5.调用 catch 方法捕捉异常。

运行结果展示：

下面是部分测试代码，第一张图为老师给的测试结果（部分），第二张图为本人代码运行结果（部分），可看出此部分代码完全相同，代码成功运行且未出错：

• 开头

result（所给的文件）：

result.txt × result2.txt

```
----------------------------
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
```

result2（输出的结果）：

result.txt result2.txt

```
----------------------------
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
```

- 结尾

result（所给的文件）：

result.txt × result2.txt

```
644  update success ---orientation orientation
645  update success ---credence credence
646  search success ---zephyr n. 和风，西风
647  remove success ---supplicate v. 恳求，乞求
648  search success ---dynamite n. 炸药，扣人心弦的事
649  remove unsuccess ---scurrilous
650  remove unsuccess ---regiment
651  search unsuccess ---ply
652  search success ---ellipsis n. 省略
653  ----------------------------
654  There are 417 nodes in this BST.
655  The height of this BST is 17.
656  ----------------------------
657  remove unsuccess ---arsonist
658  search unsuccess ---vandalism
659  search success ---royalty n. 皇室，版税（付给著作人的钱）
660  ----------------------------
661  There are 424 nodes in this BST.
662  The height of this BST is 17.
663  ----------------------------
664  remove success ---gambol n. 雀跃，嬉戏
665  remove unsuccess ---doting
```

result2（输出的结果）：

result.txt result2.txt ×

```
update success ---credence credence
search success ---zephyr n. 和风，西风
remove success ---supplicate v. 恳求，乞求
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
```

总结与收获：

一．总结

1. 实现了对基础的二叉检索树的编写；

2. 通过读取并解析文件的方式将二叉检索树作为“字典”存储单词；

二．收获

1. 通过对二叉检索树的编写，巩固加深了与二叉树相关知识的印象；

2. 巩固加深了读取文件、输出文件相关的知识的印象。

任务 2：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，都有对所编辑的文稿进行搜索的功能。当文稿内容的数据量比较大时，检索的效率就必须要进行考虑了。利用任务 1 中构建的 BST 数据结构，可以比较有效地解决这个问题。将文稿中出现的每个单词都插入到用 BST 构建的搜索表中，记录每个单词在文章中出现的行号。

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

```
[Dudley ---- <  8 10 10 >]
[Dudley's ---- <1  7 12 >]
[Exactly ---- < 9 >]
[Harry ---- < 1 4 10 12 >]
[He ---- < 2 >]
[It ---- < 7 >]
[The ---- < 6 >]
[When ---- < 6 >]
[a ---- < 2 3 10 10 >]
[after ---- < 3 >]
[all ---- < 7 >]
[almost ---- < 7 >]
[and ---- < 2 3 5 9 11 >]
[as ---- < 8 10 >]
[bag ---- < 12 >]
[because ---- < 4 >]
[bed ---- < 2 3 >]
[beneath ---- < 7 >]
[bike ---- < 9 10 >]
[birthday ---- < 1 7 >]
[but ---- < 12 >]
[catch ---- < 13 >]
[computer ---- < 8 >]
[could ---- < 1 >]
[couldn't ---- < 13 >]
[course ---- < 11 >]
[cupboard ---- < 4 >]
[down ---- < 6 >]
[dressed ---- < 6 >]
[exercise ---- < 11 >]
[fat ---- < 11 >]
[favorite ---- < 12 >]
[for ---- < 2 >]
[forgotten ---- < 1 >]
[found ---- < 2 >]
[full ---- < 5 >]
[got ---- < 1 >]
[gotten ---- < 8 >]
[had ---- < 8 >]
[hall ---- < 6 >]
[hated ---- < 11 >]
[have ---- < 1 >]
[he ---- < 1 5 6 6 8 13 >]
[hidden ---- < 7 >]
[him ---- < 13 >]
[his ---- < 3 >]
[how ---- < 1 >]
[into ---- < 6 >]
[involved ---- < 11 >]
[it ---- < 11 >]
[kitchen ---- < 6 >]
[looked ---- < 8 >]
[looking ---- < 2 >]
[mention ---- < 9 >]
[mystery ---- < 10 >]
[new ---- < 8 >]
[not ---- < 8 >]
[of ---- < 2 3 5 11 >]
[off ---- < 3 >]
[often ---- < 13 >]
[on ---- < 3 >]
[one ---- < 3 >]
[out ---- < 2 >]
[pair ---- < 2 >]
[presents ---- < 7 >]
[pulling ---- < 3 >]
[punching ---- < 12 12 >]
[put ---- < 3 >]
[racing ---- < 9 10 >]
[s ---- < 1 >]
[second ---- < 9 >]
[slept ---- < 5 >]
[slowly ---- < 1 >]
[socks ---- < 2 >]
[somebody ---- < 12 >]
[spider ---- < 3 >]
[spiders ---- < 4 >]
[stairs ---- < 4 >]
[started ---- < 2 >]
[table ---- < 7 >]
[television ---- < 9 >]
[that ---- < 5 >]
[the ---- < 4 4 6 6 8 9 9 >]
[them ---- < 3 3 5 >]
[though ---- < 8 >]
[to ---- < 4 9 10 >]
[under ---- < 2 4 >]
[unless ---- < 11 >]
[used ---- < 4 >]
[very ---- < 11 >]
[wanted ---- < 8 10 >]
[was ---- < 4 5 5 6 7 10 10 12 >]
[went ---- < 6 >]
[where ---- < 5 >]
[why ---- < 9 >]
```

本次实验准备了一篇英文文稿，文件名为 article.txt[1]，请按照上面的样例根据该 article.txt 中的内容构建索引表，并将索引表的内容按照单词的字典序列输出到 index_result.txt 文件中。

在构建 BST 中的结点数据时，请认真思考记录行号的数据类型，如果可能，尽可能使用在之前实验中自己构建的数据结构，也是对之前数据结构使用的一种验证。

设计：

编写 BSTSort 类：

一．编写 sort（BinarySearchTree<String,String> index,FileReader readFile,FileWriter outPutFile）方法，用于读取 readFile 文件的信息，通过 index 二叉树进行排序，最后输出至 outPutFile 文件中：

1.调用 try 方法：

（1）创建 BufferedReader 类对象 br，其为 BufferedReader 的有参构造器接收了 readFile 的结果；创建 PrintWriter 类对象 pw, 其为 PrintWriter 的有参构造器接收了 outPutFile 的结果；

（2）创建临时 String 类型变量 line，用于存储读取到的行的内容；创建 int 类型变量 lineNumber，以记录当前读取的行数；

（3）调用 while 循环，判定当 line 读取到的行不为空时进行循环：

①使 lineNumber 的值加一，即行数加一；

②创建 String 类型数组 words，等于 line 调用 split（“ ”）方法的结果，用以将当前行按空格分割为一个个单词；

③调用 for 循环，遍历上述创建的 words，并将当前遍历到的单词赋值给临时变量 word，调用 if 语句判断 word 不为空时：对 index 调用 search 方法查找 word 的 value，并将其

值赋给临时变量 lineNumbers，此时若 lineNumbers 为空则令其为“”，并将 lineNumber 添加到 lineNumbers 中，最后对 index 调用 insert 方法以 word 为键 lineNumbers 为值插入到二叉树中。

（4）对 index 调用 printInorder（pw）方法，将二叉树中的元素输出到文件中。

2.调用 catch（IOException e）方法捕捉异常。

二．编写主函数:

1.加上后缀 throws IOException，作异常声明;

2.创建二叉树对象 index;

3.创建 FileReader 类对象 readFile，作为待读取的文件;

4.创建 FileWriter 类对象 outPutFile，作为待输出的文件;

5.调用 sort（index，readFile，outPutFile）方法。

运行结果展示:

下列为输出文件“index_result.txt”中的部分内容，可看出单词的输出顺序符合字典的顺序，且行号的输出是以从小到大的形式，该文件的完整内容将会与代码一同压缩并提交:

index_result.txt

```
[a ---- < 1 10 18 20 43 71 75 79 82 85 86 88 89 93 94 97 97 101 109 110 111 111 112
[aback ---- < 25193 28912 >]
[abandon ---- < 2859 3313 3753 32801 33360 33366 >]
[abandoned ---- < 3151 3641 4165 9605 16581 17590 >]
[abandons ---- < 15316 >]
[abbots ---- < 11101 >]
[abductor ---- < 21772 >]
[abdullah ---- < 5522 8795 8836 8921 8959 9049 9310 9323 >]
[abelwhite ---- < 8684 8694 8710 >]
[abelwhites ---- < 8726 >]
[aberdeen ---- < 14711 19063 >]
[abhor ---- < 4956 >]
[abhorrent ---- < 9579 >]
[abide ---- < 23261 >]
[abilities ---- < 29509 31794 >]
[ability ---- < 22538 22643 23830 23873 24616 25038 29658 >]
[abjure ---- < 14196 >]
[able ---- < 81 265 653 799 1744 1802 2018 2917 3220 3294 3537 3852 4129 4182 4649 4
[abnormal ---- < 20969 28119 >]
[abnormally ---- < 21499 >]
[aboard ---- < 1713 8305 9279 25076 25082 25219 >]
[abode ---- < 17636 >]
[abominable ---- < 14761 25350 32600 >]
[abomination ---- < 12861 >]
[aborigines ---- < 7504 >]
[abound ---- < 13896 >]
```

总结与收获：

一．总结

通过对该部分作业的探究与思考，通过读取文件内容的方式，将文件中独立的单词存入二叉检索树中，对于重复出现的单词标注其每一次出现的行数，并对其每一次出现的行数进行排序，以从小到大的形式输出，以此建立所要求的单词索引表。

二．收获

通过编写该部分代码，更加熟悉了读取文件、排序等操作，并完成了该部分要求，成功建立出了“article.txt”文本对应的单词表。

任务 3：学会使用堆

堆是一棵完全二叉树，是二叉树众多应用中一个最适合用数组方式存储的树形，所以必须要完全掌握。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，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。本任务从几个方面对堆进行实践：

1) 参看书中的代码，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) 堆是二叉树的一个应用，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。

设计：

一．编写 Min_Heap 类（子任务“1)”的相关代码）：

1.创建 ArrayList<Interger>类对象 heap；

2.编写无参构造器，令 heap=new ArrayList<>（）；

3.编写 insert（int value）方法，作为插入操作：

（1）对 heap 调用 add 操作，将 value 添加到堆的末尾；

（2）对新添加的元素调用 heapUp 方法，进行上浮操作。

4.编写 delete（int value）方法，作为删除操作：

（1）调用 if 语句判断 heap 中是否存在 value 这一元素，若存在则：

①对 heap 调用 indexOf（value）方法，找到要删除的元素在堆中的索引，并将其值赋给整数类型变量 index；

②调用 swap（index，heap.size（）-1）方法将要删除的元素与最后一个元素进行交换；

③调用 remove 方法移除堆中的最后一个元素；

④对交换后在 index 索引处的值调用 heapDown 方法进行下沉操作。

5.编写 getMin（）方法：

判断堆不为空时，返回 heap 的第一个元素，否则返回-1。

6.编写 heapUp（int index）方法，作为上浮操作：

（1）由于堆的特性，父节点的索引应是子节点索引的一半，故创建变量 parent 作为父节点的索引值，其值为（index-1）/2 的结果；

（2）调用 while 循环，当 index>0 且当前索引处的值小于 parent 索引处的值时：

①调用 swap 方法交换 index 和 parent 处的值；

②令 index=parent；

③令 parent=（index-1）/2，以进行递归操作。

7.编写 heapDown（int index）方法：

（1）创建三个整数类型变量，分别为 left、right、min，其值分别为当前索引的左子结点的索引、右子节点的索引、index 的值；

（2）判断 left<heap.size 且左子结点的值小于 min 索引处的值时，令 min=left；

（3）同理若右子节点的值小于 min 索引处的值时令 min=right;

（4）判断 min 的值发生了改变时(即!=index 时)，调用 swap 方法交换 min 处索引和 index 处的值，并对 min 调用 heapDown 方法做下沉操作。

8.编写 swap（int i，int j）方法，用于交换对在 i 索引处和在 j 索引处的值：创建临时变量 temp，令其等于 i 处的值，令 i 处的值等于 j 处的值，再令 j 处的值等于 temp，即可完成交换。

二．编写 HeapSort 类（子任务“2)”的相关代码）：

1.编写 heapSort_1（int[] array）方法，该方法使用编写的二叉堆进行排序:

（1）创建 Min_Heap 类对象 minHeap;

（2）调用 for 循环将 array 中的所有元素依次通过 insert 方法插入到 minHeap 中;

（3）调用 for 循环，对 minHeap 调用 getMin（）方法，并在每一次调用 getMin（）方法后将该元素在堆中删除，以便循环获取当前堆中的最小元素，并将之存入 array 数组中。

2.编写 heapSort_2（）（int[] array）方法，该方法使用编写的三叉堆进行排序: 与步骤 1 类似，仅需将创建的 Min_Heap 类对象改为 TernaryHeap 类对象即可。

3.调用 printArray（int[] array）方法：调用 for 循环，依次将 array 数组中的元素输出至控制台即可。

三．编写 TernaryHeap 类（子任务“3)”的相关代码）：

1.由于代码重复率比较高，故 insert、delete、getMin、swap 方法的代码延用步骤一中编写二叉堆的代码;

2.下面是三叉堆中 heapUp（int index）方法的编写:

（1）与二叉堆不同，父节点的索引 parent 应为（index-1）/3 的结果;

（2）调用 while 循环，判断当前结点不是根节点且当前节点的值小于父节点的值时:

①调用 swap 方法交换 index 处和 parent 处的结点的值;

②更新当前节点为父节点;

③更新父节点索引为（index-1）/3。

3.下面是三叉堆中 heapDown（int index）方法的编写:

（1）创建四个临时变量 child1、child2、child3、min，前三者分别作为当前结点从左至右的三个子节点的索引值，min 作为 index 处的索引值;

（2）判断找出三个子节点的最小值并将其值赋给 min;

（3）判断 min 是否等于 index，若非的话调用 swap 方法交换 index 索引处和 min 索引处的值，并对 min 索引处的值调用 heapDown 方法进行下沉操作。

运行结果展示：

下列是对二叉堆及三叉堆的测试，其中 array1 与 array2 相同，均为待排序数组，heapSort_1 是使用二叉堆进行排序，而 heapSort_2 是使用三叉堆进行排序：

```
public class Test {
    public static void main(String[] args) {
        int[] array1 = {4, 10, 3, 5, 1,11,2,8,6,5,9};
        int[] array2 = {4, 10, 3, 5, 1,11,2,8,6,5,9};
        System.out.println("排序前的数组:");
        printArray(array1);

        heapSort_1(array1);

        System.out.println("二叉堆排序后的数组:");
        printArray(array1);

        System.out.println("排序前的数组:");
        printArray(array2);

        heapSort_2(array2);

        System.out.println("三叉堆排序后的数组:");
        printArray(array2);
    }
}
```

下列是运行结果：

```
排序前的数组:
4 10 3 5 1 11 2 8 6 5 9
二叉堆排序后的数组:
1 2 3 4 5 5 6 8 9 10 11
排序前的数组:
4 10 3 5 1 11 2 8 6 5 9
三叉堆排序后的数组:
1 2 3 4 5 5 6 8 9 10 11
```

总结与收获：

一．总结

1. 通过对书上代码的翻看并结合网上的信息，编写出了一个具有 insert、delete、getMin 等方法的最小堆 Min_Heap；

2. 结合二叉堆的相关知识与三叉堆的概念，编写出了最小三叉堆 TernaryHeap；

3. 通过反复调用二叉堆及三叉堆的 getMin（）和 delete（）方法，完成对数组从小到大的排序。

二．收获

通过构思与编写此部分代码，巩固了与堆、数组相关的知识和概念，并通过独立思考，实现了二叉堆的改良——三叉堆，且通过自主编写的 Test 类的测试，证明了所编写的二叉堆和三叉堆的正确性。