**Data Structures and Concurrency**

**Continuous Assessment 1 October 2022**

**Due Friday 11th November**

**N.B. Please keep your report brief and to the point.**

Submit your completed document to Canvas->Data Structures and Concurrency. (Ouriginal will be used)

Student Name Jack Patterson

Submission Date 02/11/2022

Read the document “DS & C CA 1 October 2022” before you complete this report.

To insert code in this document:

**N.B. Please save all the code you develop as you may be requested to demonstrate it.**

**1.** (a) Code for keySet method of BinarySearchTreeMap

public Set<K> keySet()  
{  
 if (root == null) return null;  
 return keySetSub(root, new HashSet<K>()); *// used a HashSet as Set is abstract and can't be instantiated*}  
  
private Set<K> keySetSub(Node node, Set<K> key){  
 key.add(node.key);  
 if (node.left != null) key.addAll(keySetSub(node.left, key));  
 if (node.right != null) key.addAll(keySetSub(node.right, key));  
  
 return key;  
}

(b) JUnit test code for this method.

@Test  
void keySet()  
{  
 BinarySearchTreeMap<Integer, String> bstm1 = new BinarySearchTreeMap<>();  
 ArrayList<Integer> testData = new ArrayList<>();  
  
 for (int i = 0; i < 10; i++)  
 {  
 bstm1.put(i, "test " + i);  
 testData.add(i);  
 }  
  
 keySetSub((Set<E>) bstm1.keySet(), (ArrayList<E>) testData); *// Have to cast it to make it work* BinarySearchTreeMap<String, String> bstm2 = new BinarySearchTreeMap<>();  
 ArrayList<String> testData2 = new ArrayList<>();  
  
 bstm2.put("fizz", "buzz");  
 bstm2.put("buzz", "fizz");  
 bstm2.put("buzzy", "fizz");  
 bstm2.put("fizzy", "fizz");  
 bstm2.put("buzziest", "fizz");  
 bstm2.put("fizzyest", "fizz");  
  
 testData2.add("buzziest"); *// Flipped due to the tree* testData2.add("fizzyest");  
 testData2.add("fizzy");  
 testData2.add("buzzy");  
 testData2.add("fizz");  
 testData2.add("buzz");  
  
 keySetSub((Set<E>) bstm2.keySet(), (ArrayList<E>) testData2);  
}

**2.** Name of the Gutenberg files you used to generate inverted index:

A Room with a view: <https://www.gutenberg.org/ebooks/2641>

Little Women: https://www.gutenberg.org/ebooks/37106

Middlemarch: https://www.gutenberg.org/ebooks/145

**3.** Use BinarySearchTreeMap implementation for index and build it using the Gutenberg files. Call height() method on the index that is generated and output it’s value.

(a) What is the height of the tree generated for the inverted index?

47

(b) How many words are in the index? (All collections classes in java api have a size() method)

23823

(c) What is the minimum height for a tree with this number of words (1 + log2n) ?

7

(d) Explain why the actual height differs from the minimum height

Not all nodes will be filled in and there may be more than the minimum height

**4.** Give code for Occurrence and updated InvertedIndex. Highlight in red the changes you have made to InvertedIndex.

public class Occurance  
{  
 private File file;  
 private List<Integer> lineNumbers; *// What implementation would you use for this?* public Occurance(File file, List<Integer> lineNumbers)  
 {  
 this.file = file;  
 this.lineNumbers = lineNumbers;  
 }  
  
 public File getFile()  
 {  
 return file;  
 }  
  
 public void setFile(File file)  
 {  
 this.file = file;  
 }  
  
 public List<Integer> getLineNumbers()  
 {  
 return lineNumbers;  
 }  
  
 public void setLineNumbers(List<Integer> lineNumbers)  
 {  
 this.lineNumbers = lineNumbers;  
 }  
  
 @Override  
 public String toString()  
 {  
 return "File: " + file + " Line Numbers: " + lineNumbers.toString();  
 }  
  
  
}

public class InvertedIndex {  
 *// Properties* private BinarySearchTreeMap<String, List<Occurance>> index;  
 *//private Map<String, List<Occurance>> index;* private Set<String> allWords; *//set of all unique words in the index  
 //used in profiling the search method* int lineNo = -1;  
  
 *// Methods* public InvertedIndex() {  
 index = new BinarySearchTreeMap<>();  
 *//index = new TreeMap<>(); // Use HashMap in a separate run* allWords = new HashSet<>();  
 }  
  
 public void buildIndex(List<File> files) {  
 for (File file : files) {  
 try {  
  
 List<Integer> lineNoList = new ArrayList<>();  
  
 Scanner in = new Scanner(file);  
  
 String line;  
 String[] words;  
  
 while (in.hasNextLine()) {  
 lineNo++;  
 *//read a line* line = in.nextLine();  
 *// parse line into words  
 // the character pattern that separates words is any sequence of  
 // characters other than letters, numbers and apostrophe  
 // This strips off punctuation marks* words = line.split("[^A-Za-z0-9']+");  
  
 *//add words and filename to index* for (String word : words) {  
 if (!word.equals("")) {  
 if (!index.containsKey(word)) {  
 List<Occurance> list = new ArrayList<>();  
 lineNoList.add(lineNo);  
 list.add(new Occurance(file, lineNoList));  
 index.put(word, list);  
 allWords.add(word);  
 } else {  
 *// word already in index - check if fileName there* List<Occurance> list = index.get(word);  
  
 ArrayList<File> filesList = new ArrayList<>();  
 for (Occurance o: list){  
 filesList.add(o.getFile());  
 }  
 if (!filesList.contains(file)) {  
 lineNoList.add(lineNo);  
 list.add(new Occurance(file, lineNoList));  
 }  
 }  
 }  
 }  
 }  
 lineNo = -1;  
 } catch (IOException exc) {  
 System.*out*.println("File does not exist");  
 exc.printStackTrace();  
 System.*exit*(1);  
 }  
 }  
 }  
  
 public void print() {  
 *// Print all keys and values in the map* Set<String> keySet = index.keySet();  
 for (String key : keySet) {  
 List<Occurance> value = index.get(key);  
 System.*out*.println(key + " : " + value);  
 }  
 System.*out*.println("\nMap Height: " + index.height());  
 *//System.out.println("\nSize: " + index.size());* }  
  
 public List<Occurance> search(String s) {  
 List<Occurance> keyOccur = new ArrayList<>();  
 for (Occurance o: index.get(s)){  
 keyOccur.add(o);  
 }  
 return keyOccur;  
 }  
  
 *//method to profile search: call search method with all words in inverted index* public void searchAllWords() {  
 int reps = 1000; *// increase this if necessary* for (int i = 1; i < reps; i++)  
  
 for (String s : allWords) {  
 search(s);  
 }  
 }  
}

**5.** JUnit test code for search() method of InvertedIndex :

@Test  
void search()  
{  
 *// Reading Files* JOptionPane.*showMessageDialog*(null, "Hit return to continue");  
 InvertedIndex index = new InvertedIndex();  
 String projPath = System.*getProperty*("user.dir") + "\\Files"; *// gets the project directory and adds on the files folder* File f = new File(projPath);  
 List<File> files = new ArrayList<>(Arrays.*asList*(f.listFiles()));  
 index.buildIndex(files);  
  
 *// Actual Test* String fileEnd = "";  
 List<Integer> intCount = new ArrayList<>();  
  
 *// Test for 1 element* intCount.add(0);  
 List<Occurance> listOccur = new ArrayList<>();  
 fileEnd = "\\File1.txt";  
 listOccur.add(new Occurance(new File(projPath + fileEnd), intCount));  
 *assertEquals*(index.search("Ireland").toString(), listOccur.toString());  
  
 *// Test for 2/multiple elements* listOccur = new ArrayList<>();  
 intCount = new ArrayList<>();  
 intCount.add(0);  
 intCount.add(1);  
 fileEnd = "\\File1.txt";  
 listOccur.add(new Occurance(new File(projPath + fileEnd), intCount));  
  
 intCount = new ArrayList<>();  
 intCount.add(0);  
 fileEnd = "\\File2.txt";  
 listOccur.add(new Occurance(new File(projPath + fileEnd), intCount));  
  
 *assertEquals*(index.search("oak").toString(), listOccur.toString());  
  
 *// Test for a value that doesn't exist  
 assertThrows*(NullPointerException.class, () ->  
 {  
 index.search("Irish");  
 });  
  
}

**6.** (a) Results of profiling the search method with searchAllWords:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Time 1 –see note below | Time from profiler | Big Oh |
| BinarySearchTreeMap | 4 | 4.182 |  |
| TreeMap | 6 | 5.292 | O(log n) |
| HashMap | 3 | 2.089 | O(1) |

Obtained with Intel Core i5 processor, 2.7 GHz, Java Version 17, Windows 10 (or specify if not…)

**Note: Time 1** is the time got by getting the time before and after the relevant code executes and computing the difference between them.

Like we did in Labs – use System.*currentTimeMillis*();

To find the Big Oh for TreeMap and HashMap, look up these classes in the java api and see the efficiency for the get() method.

(b) Specify what the java api says about the efficiency of get method for

(1) TreeMap:

O(log n)

(2) HashMap:

O(1)

(c) Insert here the screenshot showing the values you used to populate “Time from profiler” column of the table above:

Binary Tree

Graphical user interface, text, application, email

Description automatically generated

TreeMap

Graphical user interface, text, application, email

Description automatically generated

HashMap

Graphical user interface, application, Word

Description automatically generated

(d) Are the timings consistent with the Big Oh values? Yes

Explain your answer.

The speed is consistent with what is expected from the big O values, with hasmap being the fastest and the treemap being much slower.

**7.** References/Sources of information. Specify any sources you used.

Big O Values (Couldn’t find on API) <https://stackoverflow.com/questions/559839/big-o-summary-for-java-collections-framework-implementations>

Found addall method: <https://www.geeksforgeeks.org/merge-two-sets-in-java/>

***Important Note: Plagiarism and Academic Dishonesty***

Please familiarize yourself with IT Tralee Anti-plagiarism Policy and Procedures document. There is a copy of the document at <http://www.ittralee.ie/en/InformationAbout/QualityAssurance/>

At this link you will see: A5 Assessment of Learners -> A5.2 Anti-Plagiarism Policy and Procedures

The Computing Dept document “How To Cite Source Code” is available on Canvas at

<https://mtukerry.instructure.com/courses/2038/files/102042?module_item_id=23484>

Please note that if the work you submit is not your own, a mark of 0 will be awarded.

**Appendix A**

**‘Declaration of Originality Form’**- MTU-Kerry.

|  |  |
| --- | --- |
| This form **must** be completed and signed and submitted with all assignments. | |
| Please complete the information below (using BLOCK CAPITALS). | |
| Name Jack Patterson  T Number t00217640  Class Group KCPGD\_B\_Y3  Assignment Title Data Structures and Concurrency CA 1 | |
| **Students are advised to inform themselves of the Institute Anti-Plagiarism Policy.** | |
| **I confirm that this assignment is my own work and that I have:** | |
| Familiarised myself with the Institute’s Anti-Plagiarism Policy | ✔ |
| Used the Institute’s approved referencing style throughout | ✔ |
| Clearly referenced, in both the text and the bibliography or references, all sources used in the work  Not made use of the work of any other student(s) past or present without acknowledgement. This includes any of my own work, that has been previously, or concurrently, submitted for assessment, either at this or any other educational institution | ✔  ✔ |
| Not sought or used the services of any professional agencies to produce this work | ✔ |
| In addition, I understand that any false claim in respect of this work will result in disciplinary action in accordance with Institute regulations | ✔ |
|  |  |
| DECLARATION:I am aware of and understand the Institutes’ policy on plagiarism and I certify that this assignment is my own work, except where indicated by referencing, and that I have followed the good academic practices noted aboveSigned *Jack Patterson* | |