**Lab Report**

|  |  |  |  |
| --- | --- | --- | --- |
| **Lab Number:** | *4* | **Date:** | *2018-11-21* |
| **Participant:** | *Ke ZHANG* | | |
| **Participant:** | *Audrone Ravdaite* | | |

**Task 1 Binary Trees**

d.

The total number of keys in this BST is 104 and depth of this BST is 14.

The best-case tree’s depth is 7, because it is a totally balanced binary tree, every part is partially sorted. The worst-case tree’s depth’s is 104.

e.

|  |
| --- |
| public class Lab4 {  public static void main(String[] args) {    String path = "//Users//apple//Desktop//AD//labs//speech.txt";  String delimiter = " ";  String[] speech = TextReader.readFromFile(path, delimiter);    BST<String, List<Integer>> bstSpeech = new BST<String, List<Integer>>();  for (int i = 0; i < speech.length; i++) {    //Create a list of occurrences in the text as values  List<Integer> occurrences = new ArrayList<Integer>();  //Check if there exists a same key  if (bstSpeech.get(speech[i]) == null) {  occurrences.add(i);  } else {  bstSpeech.get(speech[i]).add(i);  occurrences = bstSpeech.get(speech[i]);  }  bstSpeech.put(speech[i], occurrences);  }    //task1.c  System.out.println("Cicero's first speech against Catiline - Index");    String key;  List<Integer> vaule;  Iterator iterator = bstSpeech.iterator();  Stack<String> keys = new Stack<String>();    while (iterator.hasNext()) {  key = iterator.next().toString();  keys.push(key);  vaule = bstSpeech.get(key);    System.out.println(key + ":" + vaule);  }    //task1.d  System.out.println("\nThe total number of keys in this BST: " + keys.size());  System.out.println("The depth of this BST: " + bstSpeech.depth());    }    } |
| **Listing 1:** main method of task 1 |

|  |
| --- |
| run:  Cicero's first speech against Catiline - Index  about:[36]  abusing:[8]  adopted:[168]  alarm:[62]  all:[70]  already:[120]  an:[27]  and:[66, 90, 122]  any:[99, 176]  are:[110]  arrested:[121]  as:[37]  assembling:[78]  audacity:[32]  be:[26]  before:[146]  body:[95]  by:[125, 169]  catiline:[2]  cease:[7]  city:[58]  conspiracy:[118]  countenances:[91]  defensible:[84]  design:[163]  detected:[111]  did:[140]  do:[3, 41, 51, 86, 103, 112]  does:[39, 59]  effect:[100]  end:[28]  every:[129]  feel:[106]  good:[71]  guards:[45]  have:[98]  here:[96, 131]  hill:[50]  how:[11]  in:[81]  is:[13, 23, 119, 136, 148, 180]  it:[38, 134, 149]  knowledge:[127]  last:[141]  long:[12]  looks:[89]  madness:[15]  mean:[5]  meet:[160]  mendoes:[72]  mock:[20]  most:[83]  night:[142, 145]  nightly:[44]  not:[42, 52, 60, 73, 87, 105, 114]  now:[40]  o:[1]  of:[16, 29, 33, 63, 69, 77, 92, 133, 178]  on:[47]  one:[130, 177]  our:[9]  palatine:[49]  patience:[10]  people:[65]  place:[85]  placed:[46]  plans:[109]  possesses:[132]  posted:[55]  powerless:[124]  precaution:[75]  present:[97]  rendered:[123]  see:[115]  senate:[80]  still:[18]  summoned:[158]  swaggering:[35]  taken:[76]  that:[14, 30, 107, 116, 138, 150, 156, 175]  the:[43, 48, 53, 57, 61, 64, 67, 74, 79, 88, 126, 144]  there:[24, 137, 155, 165]  think:[174]  this:[82, 93]  throughout:[56]  to:[6, 19, 25, 159]  unacquainted:[181]  unbridled:[31]  union:[68]  upon:[101]  us:[21, 179]  venerable:[94]  was:[154, 164, 167]  watches:[54]  were:[152]  what:[135, 143, 162]  when:[0, 22]  where:[147]  which:[128, 166, 172]  who:[153]  with:[171]  you:[4, 102, 104, 113, 139, 151, 157, 161, 170, 173]  your:[108, 117]  yours:[17, 34]    The total number of keys in this BST: 104  The depth of this BST: 14  BUILD SUCCESSFUL (total time: 0 seconds) |
| **Listing 2:** the results of task1 |

**Task 2 Priority Queue**

|  |
| --- |
| public class Lab4 {  public static void main(String[] args) {    //task 1.a put the speech of Cicero in a String array  String path ="//Users//apple//Desktop//AD//labs//speech.txt";  String delimiter = " ";  String[] speech = TextReader.readFromFile(path, delimiter);    //task 1.b put the keys and list of occurrences in the BST  BST<String, List<Integer>> bstSpeech = new BST<String, List<Integer>>();  for (int i = 0; i < speech.length; i++) {  //Create a list of occurrences in the text as values  List<Integer> occurrences = new ArrayList<Integer>();  //Check if there exists a same key  if (bstSpeech.get(speech[i]) == null) {  occurrences.add(i);  } else {  bstSpeech.get(speech[i]).add(i);  occurrences = bstSpeech.get(speech[i]);  }  bstSpeech.put(speech[i], occurrences);  }    //task 1.c put out the word index in an alphabetical order  System.out.println("Cicero's first speech against Catiline - Index");    String key;  List<Integer> value;  int occurrences = 0;  Iterator iterator = bstSpeech.iterator();  Stack<String> keys = new Stack<String>();  Heap<Node<String, Integer>> priorityQueue = new Heap<>(105);    while (iterator.hasNext()) {  key = iterator.next().toString();  keys.push(key);  value = bstSpeech.get(key);  occurrences = value.size();    System.out.println(key + ":" + value);    //task 2.b map words and the number of occurrences in a heap  priorityQueue.insert(new Node<String, Integer>(key, occurrences));  }    //print out the number of keys and the depth of BST  System.out.println("\nThe total number of keys in this BST: " + keys.size());  System.out.println("The depth of this BST: " + bstSpeech.depth() + "\n");    //task 2.c print out three most occuring words in order  for (int i = 1; i < 4; i++) {  Node max = priorityQueue.getMax();  System.out.println("No " + i + ". '" + max.getKey() + "' Occurrences: " + max.getOccrrences());  }  }  } |
| **Listing 3:** main method of task 2 |

|  |
| --- |
| public class Node<Word, Occurrences extends Comparable<Occurrences>> implements Comparable<Node> {  Word word;  Occurrences occrrences;  public Node( Word word, Occurrences occrrences) {  this.word = word;  this.occrrences = occrrences;    }  @Override  public int compareTo(Node compareNode) {  if (this.occrrences.getClass().equals(compareNode.occrrences.getClass())) {  return this.occrrences.compareTo((Occurrences) compareNode.occrrences);  }  return -255;  }  public Word getKey() {  return word;  }    public Occurrences getOccrrences() {  return occrrences;  }  } |
| **Listing 4:** Node class |

|  |
| --- |
|  |
| **Listing 5:** screenshot of the output |