# CIS 143 Lab 9: Work with binary search trees (BSTs), search, and IOStreams (45 points)

Please leave the lab questions/instructions/rubrics/etc. in place. Just paste your screenshots and code below the rubric.

Purpose/knowledge/skills: Search is a common task in computer science. Binary search is a very efficient way to search. IOStreams help you get information into and out of your Java programs.

## Task 1: Questions about binary search trees (10 points, about 1 point per question)

|  |  |
| --- | --- |
| **Tree A**  Binary Search Tree  Thank tos [https://www.tutorialspoint.com/ data\_structures\_algorithms/binary\_search\_tree.htm](https://www.tutorialspoint.com/data_structures_algorithms/binary_search_tree.htm) . | **Tree B** Image result for binary tree  Thanks to <https://www.cs.cmu.edu/~adamchik/15-121/lectures/Trees/trees.html> |
| **Tree C**  Diagram, schematic  Description automatically generated  Thanks to <https://www.cs.cmu.edu/~adamchik/15-121/lectures/Trees/trees.html> | **Tree D**  http://btechsmartclass.com/DS/images/BST%20Example.pngThanks to: <http://btechsmartclass.com/DS/U5_T1.html> |

1.1. Is tree A binary search tree? If not, which property of BSTs does it fail?

|  |
| --- |
| Yes, tree A is a binary search tree. |

1.2. Why is tree B not a binary search tree? Which property of BSTs does it fail?

|  |
| --- |
| Tree B not a binary search tree because Node 12 is in the left tree of Node 8. All nodes in the left subtree of a given node should have values less than the given node. |

1.3. Is tree C binary search tree? If not, which property of BSTs does it fail?

|  |
| --- |
| Tree C not a binary search tree because Node 2 is in the left tree of Node 1. All nodes in the left subtree of a given node should have values less than the given node. |

1.4. Is tree D binary search tree? If not, which property of BSTs does it fail?

|  |
| --- |
| Tree D is a binary search tree. |

1.5. If we were searching tree A for the value 15, which node would get checked first?

|  |
| --- |
| The root node, 27. |

1.6. If we were searching tree D for the value 12, what path of nodes would we follow? (Four nodes, 4 points)

|  |
| --- |
| 1. The root node, 25. 2. The left child, 20. 3. The left child, 10. 4. The right child, 12. |

The lab continues on the following page.

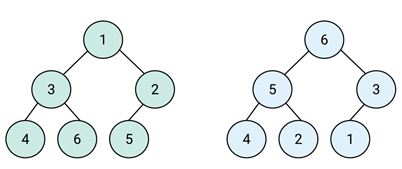
## Task 2: Work with breadth-first search (BFS) in Java (10 points)

Text

Description automatically generated

Thanks to <https://en.wikipedia.org/wiki/Breadth-first_search>

Write Java code to construct the following trees and call a breadth-first search. You are not searching for a value, you are creating an ArrayList S that contains all the nodes of the tree.



Thanks to: zenalc.com

**Java code and tree-construction examples are included in the lab files for this week.** Sample output (of a different tree) follows.

Chart, text, scatter chart

Description automatically generated

**However, you are responsible for constructing and enumerating two trees in your Java code.**

Rubric:

* Student name and today’s date is a comment in the first line of the programs: -5 points if fails
* Screenshot and program code: -5 points if fails
* Construct two trees and run breadth-first search: 10 points

Please paste a screenshot of a successful program run, and copy-and-paste the source code from your main program's .java file, here.

|  |
| --- |
|  |
| // Kai Gillespie 20240310  **import** java.util.\*;  **public** **class** Lab9\_Task2 {  **public** **static** **void** main(String[] args) {    // Create the first tree structure  TreeNode<Integer> root1 = **new** TreeNode<Integer>();  root1.data = 1;  TreeNode<Integer> three = root1.addChild(**new** TreeNode<Integer>(3));  TreeNode<Integer> two = root1.addChild(**new** TreeNode<Integer>(2));  TreeNode<Integer> four = three.addChild(**new** TreeNode<Integer>(4));  TreeNode<Integer> six = three.addChild(**new** TreeNode<Integer>(6));  TreeNode<Integer> five = two.addChild(**new** TreeNode<Integer>(5));  // Perform breadth-first search on the first tree  System.***out***.println("Breadth-first search of tree 1:");  *breadthFirstSearch*(root1);  // Create the second tree structure  TreeNode<Integer> root2 = **new** TreeNode<Integer>(6);  TreeNode<Integer> five2 = root2.addChild(**new** TreeNode<Integer>(5));  TreeNode<Integer> three2 = root2.addChild(**new** TreeNode<Integer>(3));  TreeNode<Integer> four2 = five2.addChild(**new** TreeNode<Integer>(4));  TreeNode<Integer> two2 = three2.addChild(**new** TreeNode<Integer>(2));  TreeNode<Integer> one = three2.addChild(**new** TreeNode<Integer>(1));  // Perform breadth-first search on the second tree  System.***out***.println("Breadth-first search of tree 2:");  *breadthFirstSearch*(root2);  }  // breadthFirstSearch  // Enumerates all nodes in a tree  // Displays the tree and enumerated list of nodes to the console  **public** **static** <T> **void** breadthFirstSearch (TreeNode<T> root)  {  // A class that implements the Set interface  ArrayList<TreeNode<T>> s = **new** ArrayList<>();    // A class that implements the Queue interface  ArrayDeque<TreeNode<T>> q = **new** ArrayDeque<>();    // Run a breadth-first search  s.add(root);  q.offer(root);    **while** (!q.isEmpty() ){  TreeNode<T> current = q.remove();  **for** (TreeNode<T> i: current.getChildren()) {  **if** (! s.contains(i)) {  s.add(i);  q.offer(i);  }  }  }    // Output results  *outputTree* (root);    System.***out***.println("Has results:");  **for** (TreeNode<T> i: s) {  System.***out***.print(i.data + " ");  }  System.***out***.println();  System.***out***.println();  }    // ouputTree  // Prints a tree to the console  // Students do not need to modify this code.  **public** **static** <T> **void** outputTree (TreeNode<T> t) {    **if** (t.getParent() == **null**)  System.***out***.print("Root node: " );    // This recurses repeatedly through the parents of each branch of the tree  // This is O(n^2) and could be dramatically improved.  TreeNode<T> parent = t.getParent();  **while** (parent != **null**) {  System.***out***.print(" ");  parent = parent.getParent();  }  System.***out***.println(t.data);    **for** (TreeNode<T> a : t.getChildren())  *outputTree*(a);  }  } |

## Task 3: Work with IOStreams (5 points)

Write a Java program that writes fives strings to the file “lab10.txt”, and then reads those five strings back into different variables.

The newer versions of Java have changed how they work with file output streams! As a student noted in class: FileOutputStream throws an exception of type FileNotFoundException, and DataOutputStream.close() throws an IOException.

In practice, you won't run into these in the lab. But you do need to call them out when declaring static void main:

 public static void main(String[] args) throws IOException

handles both cases.

Rubric:

* Student name and today’s date is a comment in the first line of the programs: -5 points if fails
* Screenshot and program code: -5 points if fails
* Program works: 5 points

Please paste a screenshot of a successful program run, and copy-and-paste the source code from your main program's .java file, here.

|  |
| --- |
|  |
| // Kai Gillespie 20240310  **import** java.io.\*;  **public** **class** Lap9\_Task3 {  **public** **static** **void** main(String[] args) **throws** IOException {  String[] stringsToWrite = {"Hello", "World", "Java", "Programming", "Lab10"};  String filePath = "lab10.txt";  // Write strings to the file  **try** (FileWriter writer = **new** FileWriter(filePath)) {  **for** (String string : stringsToWrite) {  writer.write(string + System.*lineSeparator*());  }  } **catch** (IOException e) {  System.***err***.println("An error occurred while writing to the file: " + e.getMessage());  **return**;  }  // Read strings back into different variables  String string1, string2, string3, string4, string5;  **try** (BufferedReader reader = **new** BufferedReader(**new** FileReader(filePath))) {  string1 = reader.readLine();  string2 = reader.readLine();  string3 = reader.readLine();  string4 = reader.readLine();  string5 = reader.readLine();  } **catch** (IOException e) {  System.***err***.println("An error occurred while reading from the file: " + e.getMessage());  **return**;  }  // Output the read strings to verify the operation  System.***out***.println("String 1: " + string1);  System.***out***.println("String 2: " + string2);  System.***out***.println("String 3: " + string3);  System.***out***.println("String 4: " + string4);  System.***out***.println("String 5: " + string5);  }  } |

## Task 4: Work with recursion in Java (20 points)

Follow the **lecture algorithm example** to create a recursive program that lists all the permutations of the letters of a string. This also happens to be the textbook algorithm for PERMUTE.

Test your code with the string “POEM”.

Rubric:

* Student name and today’s date is a comment in the first line of the programs: -10 points if fails
* Screenshot and program code: -10 points if fails
* Permute() calculations work correctly: 10 points
* Follows lecture slide algorithm: available = available.substring(0, i) + available.substring(i + 1); and permute(chosen, available); : 10 points

Please paste a screenshot of a successful program run, and copy-and-paste the source code from your main program's .java file, here.

|  |
| --- |
|  |
| // Kai Gillespie 20240310  **public** **class** Lab9\_Task4 {  // Outputs all permutations of the given string  **public** **static** **void** main(String[] args) {  *permute*("", "POEM"); // Initial call with an empty 'chosen' string and 'POEM' as the available characters  }  // Overloaded private recursive method that generates permutations  **private** **static** **void** permute(String chosen, String available) {  // base case: no choices left to be made (all characters have been chosen)  **if** (available.length() == 0) {  System.out.println(chosen); // Print the permutation result  } **else** {  // recursive case: choose each possible next letter  **for** (**int** i = 0; i < available.length(); i++) {  // Choose the ith character from the remaining available characters  **char** c = available.charAt(i);  // Construct the new 'available' string without the chosen character  available = available.substring(0, i) + available.substring(i + 1);  // Add the chosen character to the 'chosen' string  chosen += c;  // Recursively call permute with the new chosen and available strings  permute(chosen, available);  // Backtrack: remove the last chosen character to try a new character in the next iteration  // This step is necessary to return to the state before the recursive call  chosen = chosen.substring(0, chosen.length() - 1);  // Restore the original 'available' string by inserting the character back to its position  available = available.substring(0, i) + c + available.substring(i);  }  }  } |