Lab 10

Binary Search Trees (BSTs), Search, and IOStreams

1. 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**    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. 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?

**- The subtree to the left of the root node contains values greater than 8, and the subtree to the right of the root node contains values less than 8.**

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

**- No, Nodes 3 and 5 are on the left of their root node but are greater than that root node.**

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

**- Yes, tree D is a binary search tree.**

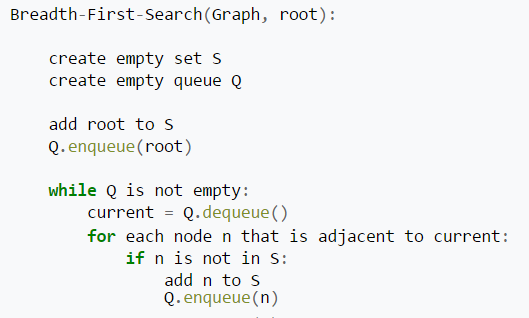
1.5. If we were searching tree A for the value 15, which node would get checked first?

**-The root node, 27.**

1.6. If we were searching tree D for the value 13, what path of nodes would we follow? (Five nodes, 4 points)

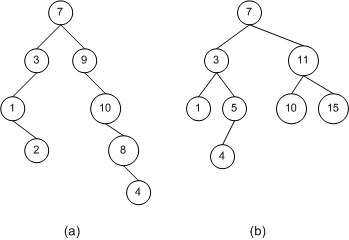
**25 -> 20 -> 10 -> 12 -> 15.**

2. Breadth-first search (10 points)



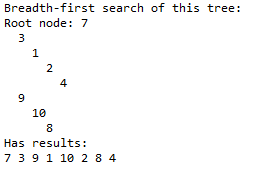
Thanks to <https://en.wikipedia.org/wiki/Breadth-first_search>

Write Java code construct the following trees and call a breadth-first search. You are not searching for a value, you are creating a list S that contains all the nodes of the tree. Use an ArrayList to store the node values in the order they are enumerated.



Thanks to: <https://msdn.microsoft.com/en-us/library/ms379572(v=vs.80).aspx>

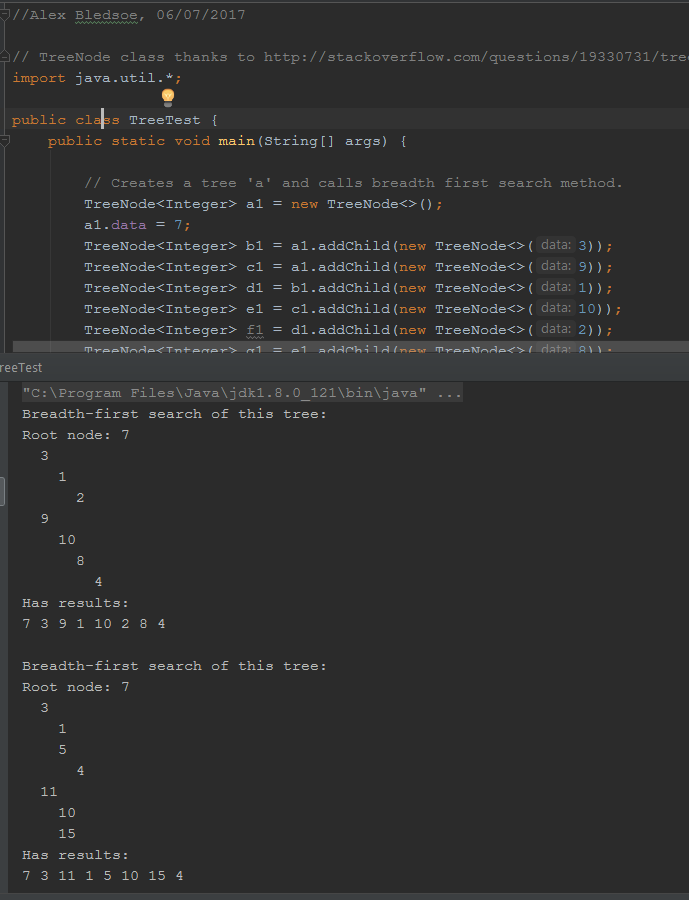
Java code and tree-construction examples are included. Sample output follows.



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  
Tree construction and breadth-first search: 8 points  
Q&A: 2 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.



**TreeTest.java:**

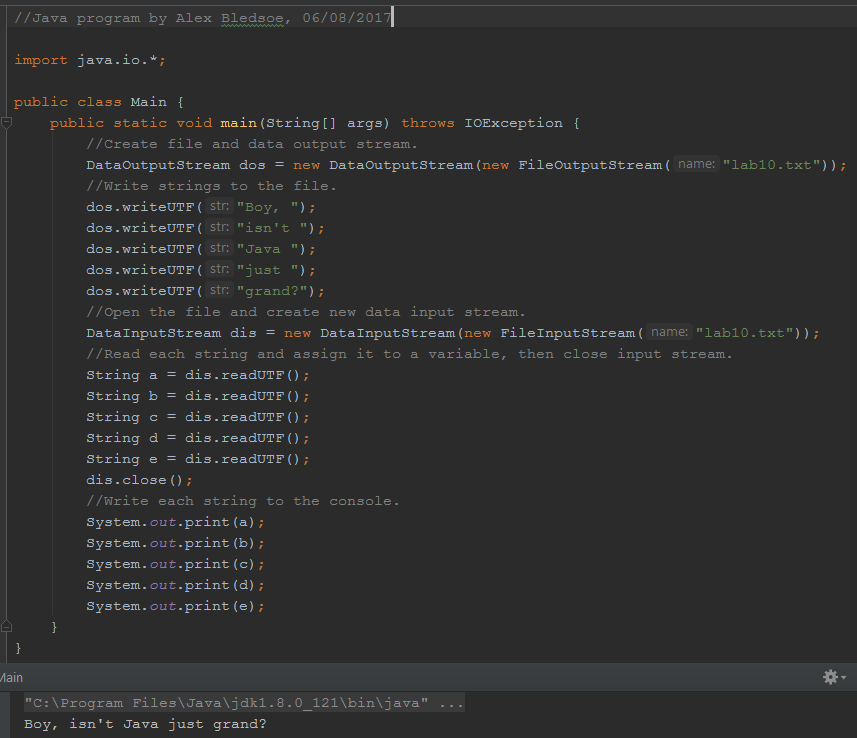
//Alex Bledsoe, 06/07/2017  
  
// TreeNode class thanks to http://stackoverflow.com/questions/19330731/tree-implementation-in-java-root-parents-and-children  
import java.util.\*;  
  
public class TreeTest {  
 public static void main(String[] args) {  
   
 // Creates a tree 'a' and calls breadth first search method.  
 TreeNode<Integer> a1 = new TreeNode<>();  
 a1.data = 7;  
 TreeNode<Integer> b1 = a1.addChild(new TreeNode<>(3));  
 TreeNode<Integer> c1 = a1.addChild(new TreeNode<>(9));  
 TreeNode<Integer> d1 = b1.addChild(new TreeNode<>(1));  
 TreeNode<Integer> e1 = c1.addChild(new TreeNode<>(10));  
 TreeNode<Integer> f1 = d1.addChild(new TreeNode<>(2));  
 TreeNode<Integer> g1 = e1.addChild(new TreeNode<>(8));  
 TreeNode<Integer> h1 = g1.addChild(new TreeNode<>(4));  
  
 *breadthFirstSearch* (a1);  
  
 // Creates a tree 'b' and calls breadth first search method.  
 TreeNode<Integer> a2 = new TreeNode<>();  
 a2.data = 7;  
 TreeNode<Integer> b2 = a2.addChild(new TreeNode<>(3));  
 TreeNode<Integer> c2 = a2.addChild(new TreeNode<>(11));  
 TreeNode<Integer> d2 = b2.addChild(new TreeNode<>(1));  
 TreeNode<Integer> e2 = b2.addChild(new TreeNode<>(5));  
 TreeNode<Integer> f2 = e2.addChild(new TreeNode<>(4));  
 TreeNode<Integer> g2 = c2.addChild(new TreeNode<>(10));  
 TreeNode<Integer> h2 = c2.addChild(new TreeNode<>(15));  
  
 *breadthFirstSearch* (a2);  
 }  
   
  
 // 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  
 System.*out*.println("Breadth-first search of this tree:");  
 *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);  
 }  
}

3. 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.

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.



**Main.java:**

//Java program by Alex Bledsoe, 06/08/2017  
  
import java.io.\*;  
  
public class Main {  
 public static void main(String[] args) throws IOException {  
 //Create file and data output stream.  
 DataOutputStream dos = new DataOutputStream(new FileOutputStream("lab10.txt"));  
 //Write strings to the file.  
 dos.writeUTF("Boy, ");  
 dos.writeUTF("isn't ");  
 dos.writeUTF("Java ");  
 dos.writeUTF("just ");  
 dos.writeUTF("grand?");  
 //Open the file and create new data input stream.  
 DataInputStream dis = new DataInputStream(new FileInputStream("lab10.txt"));  
 //Read each string and assign it to a variable, then close input stream.  
 String a = dis.readUTF();  
 String b = dis.readUTF();  
 String c = dis.readUTF();  
 String d = dis.readUTF();  
 String e = dis.readUTF();  
 dis.close();  
 //Write each string to the console.  
 System.*out*.print(a);  
 System.*out*.print(b);  
 System.*out*.print(c);  
 System.*out*.print(d);  
 System.*out*.print(e);  
 }  
}

4. Comprehensive (20 points) Based on Goodrich Programming Projects 12.1.

Write a program to solve the “Vegetarians and Meat Eaters” problem. Three vegetarians and three hungry meat-eaters need to cross a river. Unfortunately, the boat only holds two people. If the meat-eaters outnumber the vegetarians on either bank, the vegetarians will be eaten!

Your challenge is to find a series of moves that gets all three vegetarians **and** all three meat-eaters across the river safely.

Write a program that solves the vegetarians and meat-eaters problem. You are given broad leeway on how to represent this problem. Recursion and backtracking (depth-first search) is **one** way to solve this.

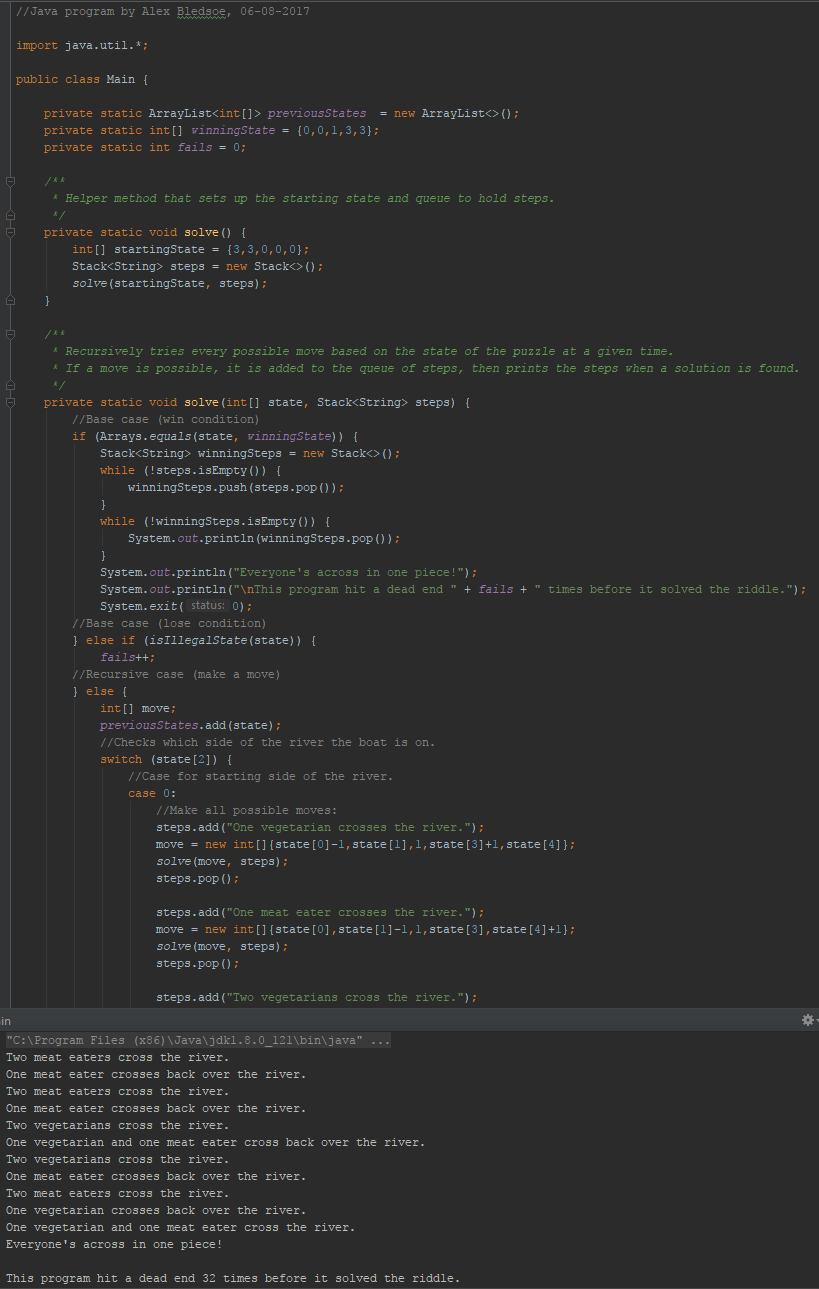
There is a hint below in invisible text. If you get stuck, change the color of the text below to read the hint.

Hint: The base cases are:

4.1. 3 vegetarians and 3 meat eaters on far side of bridge = success  
4.2. 3 vegetarians and 3 meat eaters on near side of bridge = we’re back at the start and have looped around, stop following this path  
4.3. More meat eaters than vegetarians on near side of bridge = vegetarians get eaten  
4.4. More meat eaters than vegetarians on far side of bridge = vegetarians get eaten

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  
Program works: 5 points  
Program logic: 15 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.



**Main.java:**

//Java program by Alex Bledsoe, 06-08-2017  
  
import java.util.\*;  
  
public class Main {  
  
 private static ArrayList<int[]> *previousStates* = new ArrayList<>();  
 private static int[] *winningState* = {0,0,1,3,3};  
 private static int *fails* = 0;  
  
 */\*\*  
 \* Helper method that sets up the starting state and queue to hold steps.  
 \*/* private static void solve() {  
 int[] startingState = {3,3,0,0,0};  
 Stack<String> steps = new Stack<>();  
 *solve*(startingState, steps);  
 }  
  
 */\*\*  
 \* Recursively tries every possible move based on the state of the puzzle at a given time.  
 \* If a move is possible, it is added to the queue of steps, then prints the steps when a solution is found.  
 \*/* private static void solve(int[] state, Stack<String> steps) {  
 //Base case (win condition)  
 if (Arrays.*equals*(state, *winningState*)) {  
 Stack<String> winningSteps = new Stack<>();  
 while (!steps.isEmpty()) {  
 winningSteps.push(steps.pop());  
 }  
 while (!winningSteps.isEmpty()) {  
 System.*out*.println(winningSteps.pop());  
 }  
 System.*out*.println("Everyone's across in one piece!");  
 System.*out*.println("\nThis program hit a dead end " + *fails* + " times before it solved the riddle.");  
 System.*exit*(0);  
 //Base case (lose condition)  
 } else if (*isIllegalState*(state)) {  
 *fails*++;  
 //Recursive case (make a move)  
 } else {  
 int[] move;  
 *previousStates*.add(state);  
 //Checks which side of the river the boat is on.  
 switch (state[2]) {  
 //Case for starting side of the river.  
 case 0:  
 //Make all possible moves:  
 steps.add("One vegetarian crosses the river.");  
 move = new int[]{state[0]-1,state[1],1,state[3]+1,state[4]};  
 *solve*(move, steps);  
 steps.pop();  
  
 steps.add("One meat eater crosses the river.");  
 move = new int[]{state[0],state[1]-1,1,state[3],state[4]+1};  
 *solve*(move, steps);  
 steps.pop();  
  
 steps.add("Two vegetarians cross the river.");  
 move = new int[]{state[0]-2,state[1],1,state[3]+2,state[4]};  
 *solve*(move, steps);  
 steps.pop();  
  
 steps.add("Two meat eaters cross the river.");  
 move = new int[]{state[0],state[1]-2,1,state[3],state[4]+2};  
 *solve*(move, steps);  
 steps.pop();  
  
 steps.add("One vegetarian and one meat eater cross the river.");  
 move = new int[]{state[0]-1,state[1]-1,1,state[3]+1,state[4]+1};  
 *solve*(move, steps);  
 steps.pop();  
 break;  
 //Case for goal side of the river.  
 case 1:  
 //Make all possible moves:  
 steps.add("One vegetarian crosses back over the river.");  
 move = new int[]{state[0]+1,state[1],0,state[3]-1,state[4]};  
 *solve*(move, steps);  
 steps.pop();  
  
 steps.add("One meat eater crosses back over the river.");  
 move = new int[]{state[0],state[1]+1,0,state[3],state[4]-1};  
 *solve*(move, steps);  
 steps.pop();  
  
 steps.add("Two vegetarians cross back over the river.");  
 move = new int[]{state[0]+2,state[1],0,state[3]-2,state[4]};  
 *solve*(move, steps);  
 steps.pop();  
  
 steps.add("Two meat eaters cross back over the river.");  
 move = new int[]{state[0],state[1]+2,0,state[3],state[4]-2};  
 *solve*(move, steps);  
 steps.pop();  
  
 steps.add("One vegetarian and one meat eater cross back over the river.");  
 move = new int[]{state[0]+1,state[1]+1,0,state[3]-1,state[4]-1};  
 *solve*(move, steps);  
 steps.pop();  
 break;  
 }  
 }  
 }  
  
 */\*\*  
 \* Checks current state of puzzle for lose conditions, e.g.:  
 \* - Meat eaters outnumber vegetarians on either side where vegetarians are present.  
 \* - Everybody is in the same positions they were in at a (the puzzle has looped).  
 \* - There are a negative number of vegetarians or meat eaters on either side.  
 \* - There are more than 3 vegetarians or meat eaters on either side.  
 \*/* private static boolean isIllegalState(int[] state) {  
 return *isPreviousState*(state) ||  
 state[1] > state[0] && state[0] > 0 ||  
 state[4] > state[3] && state[3] > 0 ||  
 state[0] > 3 ||  
 state[0] < 0 ||  
 state[1] > 3 ||  
 state[1] < 0;  
 }  
  
 */\*\*  
 \* Checks whether the current state of the puzzle is identical to  
 \* any of the previous moves made, indicating a loop.  
 \*/* private static boolean isPreviousState(int[] currentState) {  
 for (int[] previousState : *previousStates*) {  
 if (Arrays.*equals*(currentState, previousState)) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 public static void main(String[] args) {  
 *solve*();  
 }  
}