Assignment 5 - CT2109 Object Oriented Programming: Data Structures and Algorithms

Daniel Hannon (19484286)

May 2021

1 Problem Analysis

1.1 Overview

Brief:*"For this assignment, you will use the BinaryTree implementation on Blackboard to program a guessing game. Firstly, you will manually build an initial tree in which each internal node is a yes/no question. Yes goes to left side(left child), No goes to right side(right child).Each leaf nodein the tree is a guess. If the user arrives at a leaf node and the guess is wrong, get the user to provide you with what the correct answer actually was and to provide a new yes/no question which can be added to the tree."*

1.2 Part 1: Getting the questions to work

The main part of this was implementing ***BinaryNodeInterface¡T¿*** which was quite simple. the only Alteration I made to it was I overrode the *toString()* method for my implementation but this will be explained later. *getHeight()*, *getNumberOfNodes()*, and *copy()* were all implemented using Recursion for speed.
Once I had the Binary Tree implemented, It was a matter of constructing a test tree. Logically the tree needed to be built from the results backwards so that wasn't the hardest thing to do, once that was done it was a matter of getting user input.
in order to make the input more flexible I forced it to be lowercase once input had been scanned and then it was compared to longform and shortform of yes/no.

1.3 Part 2: Adding in new questions

In order to avoid the need to perform a binary search at the end if you had to input a new question/answer, the question and answer are sanitized, I saved the position of the previous node and the direction you took when you responded. Then a new branch is created, the leaf which was formerly in that position in the tree is added to the new branch and that branch is inserted where that leaf used to be.

1.4 Part 3: Saving

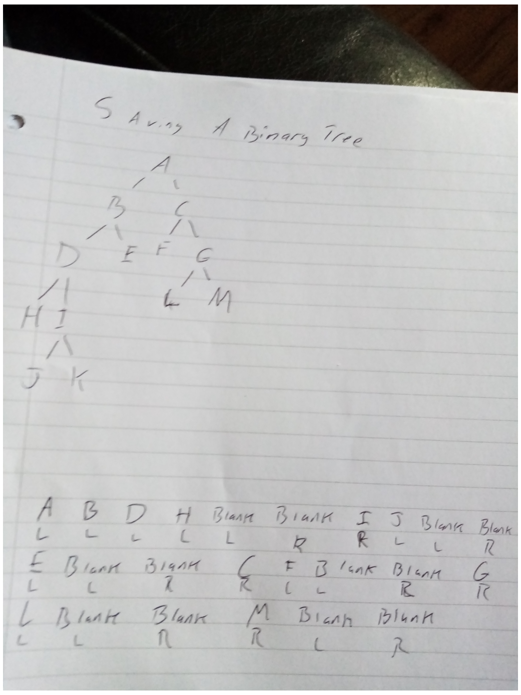

Saving A Binary Tree
A
B
C
D
E
F
G
H
I
L
M
J
K
A B D H Blank Blank I J Blank Blank
L L L L L R R L L R
E Blank Blank C F Blank Blank G
L L R R L L R R
L Blank Blank M Blank Blank
L L R R L R

Figure 1: Binary Tree and potential save schema

Saving was a matter of Finding a way to serialize the Binary Tree and then dumping this in a textfile. I figured the easiest way to serialize it would be to overwrite the *toString()* method and have it return the nodes in comma separated form, where left hand nodes are printed first until a null is found which is represented as "#" and then it prints the right hand nodes, this runs recursively of course. I then utilize a **FileWriter** and place it all in *saveFile.txt*

1.5 Part 4: Loading

Loading the file and keeping the structure was initally somewhat challenging as I had planned to use 3 Stacks and done some weird stuff but ultimately I achieved it by using a single Stack.
In order to load it I felt the easiest way would be to split savestring at every comma and work backwards, if two elements in a row were "#" it creates a leafnode using the element after the "#" in the search, this is then added to the stack. where if it lands on an element that is not "#" it pulls two elements from the Stack, creates a new branch using the current element as the data and the two recently pulled from the stack as it's branches/leaves. Once the entire array had been iterated through, the top element in the stack is returned.
As all elements either have two children or none I was able to reduce the amount of required checks for it to run to success.

2 Code

Binary Tree implementation

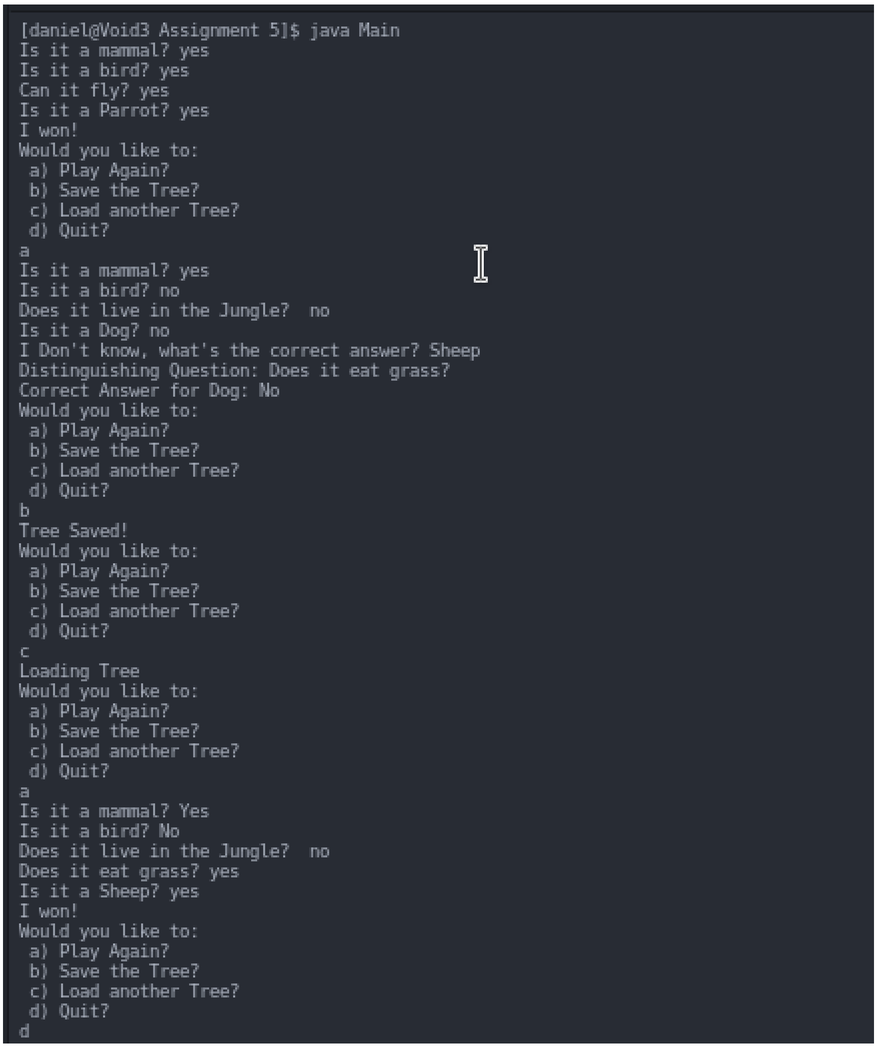
```java
public class TreeNode<T> implements BinaryNodeInterface<T> {
  private T data;
  private TreeNode<T> left;
  private TreeNode<T> right;

  public TreeNode(T inputData) {
    this.data = inputData;
    this.left = null;
    this.right = null;
  }
  public TreeNode(T inputData, TreeNode<T> leftNode, TreeNode<T> rightNode) {
    this.data = inputData;
    this.left = leftNode;
    this.right = rightNode;
  }

  public T getData() {
    return this.data;
  }

  public void setData(T newData) {
    this.data = newData;
  }

  public BinaryNodeInterface<T> getLeftChild() {
    return this.left;
  }

  public BinaryNodeInterface<T> getRightChild() {
    return this.right;
  }

  public void setLeftChild(BinaryNodeInterface<T> leftChild) {
    this.left = (TreeNode<T>)leftChild;
  }

  public void setRightChild(BinaryNodeInterface<T> rightChild) {
    this.right = (TreeNode<T>)rightChild;
  }

  public boolean hasLeftChild() {
    return (this.left == null) ? false : true;
  }

  public boolean hasRightChild() {
    return (this.right == null) ? false : true;
  }

  public boolean isLeaf() {
    return (this.left == null && this.right == null) ? true : false;
  }

  public int getNumberOfNodes() {
    if(this.left == null && this.right == null) {
      return 0;
    } else {
      int total = 0;
      if(this.left!=null) {
        total+=1;
        total += this.left.getNumberOfNodes();
      }
      if(this.right!=null) {
        total+=1;
        total += this.right.getNumberOfNodes();
      }
      return total;
    }
  }

  public int getHeight() {
    //Returns the largest height
    int templeft = 0;
    int tempright = 0;
    if(this.left!= null) {
      templeft = 1 + this.left.getHeight();
    }
    if(this.right != null) {
      tempright = 1 + this.right.getHeight();
    }
    if(templeft > tempright) {
      return templeft;
    } else {
      return tempright;
    }
  }

  public BinaryNodeInterface<T> copy() {
    TreeNode<T> leftCopy = null;
    TreeNode<T> rightCopy = null;
    if(this.left != null) {
      leftCopy = (TreeNode<T>)left.copy();
    }
    if(this.right != null) {
      rightCopy = (TreeNode<T>)right.copy();
    }
    return new TreeNode<T>(this.data,leftCopy,rightCopy);
  }

  @Override
  public String toString() {
    String leftString = "#";
    String rightString = "#";
    if(this.left != null) {
      leftString = this.left.toString();
    }
    if(this.right != null) {
      rightString = this.right.toString();
    }
    return this.data +","+leftString+","+rightString;
  }
}
```

Main file

```
import java.util.Scanner;
import java.util.Stack;
import java.io.*;
```

```java

public class Main {
  public static TreeNode<String> createTree() {
    //Create a demo question tree
    TreeNode<String> answer = new TreeNode<String>("Penguin");
    TreeNode<String> answer2 = new TreeNode<String>("Parrot");
    TreeNode<String> question1 = new TreeNode<String>("Can it fly?", answer2, answer);
    answer = new TreeNode<String>("Lion");
    answer2 = new TreeNode<String>("Dog");
    TreeNode<String> question2 = new TreeNode<String>("Does it live in the Jungle?",answer,answer2);
    TreeNode<String>question3 = new TreeNode<String>("Is it a bird?", question1, question2);

    answer = new TreeNode<String>("Shark");
    answer2 = new TreeNode<String>("Crocodile");
    question1 = new TreeNode<String>("Does it have Scales?",answer2,answer);
    question2 = new TreeNode<String>("Is it a mammal?",question3,question1);
    return question2;
  }

  public static TreeNode<String> loadTree() {
    String myBinaryTree = "";
    File myFile = new File("saveFile.txt");
    TreeNode<String> output = null;
    Stack<TreeNode<String>> myStack = new Stack<TreeNode<String>>();
    try {
      Scanner fileScanner = new Scanner(myFile);
      while(fileScanner.hasNextLine()) {
        myBinaryTree += fileScanner.nextLine();
      }
      fileScanner.close();
      if(myBinaryTree.length() == 0) {
        System.out.println("Empty file returning default file");
        return createTree();
      }
      String[] myArray = myBinaryTree.split(",");
      for(int i = myArray.length - 1; i >= 0; i--) {
        //Since each node either has two children or no children this works
        if(myArray[i].equals("#") && myArray[i-1].equals("#")) {
          //If both children are null it creates a new node and pushes it to the stack
          myStack.push(new TreeNode<String>(myArray[i-2],null,null));
          i-=2;
        } else {
          //Create a question and it's answers using two child nodes
          TreeNode<String> left  = myStack.pop();
          TreeNode<String> right = myStack.pop();
          myStack.push(new TreeNode<String>(myArray[i],left,right));
        }
      }
    } catch(IOException e) {
      System.out.println("No Save file exists! returning the default tree");
      return createTree();
    }
    //Pop top of stack as it is the first question
    output = myStack.pop();
    return output;
  }

  public static void saveTree(TreeNode<String> myTree) {
    //Create File Writer
    File myFile = new File("saveFile.txt");
    FileWriter fileOutput;
    try {
      myFile.createNewFile();
      fileOutput = new FileWriter(myFile);
      fileOutput.write(myTree.toString());
      fileOutput.close();
    } catch(IOException e) {
      System.out.print(e);
    }
  }

  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    String myGuess = "";
    boolean isRunning = true;
    TreeNode<String> questionTree = createTree();
    TreeNode<String> currentNode = questionTree;
    TreeNode<String> previous = null;
    int path = 0;
    while(isRunning) {
      while(!currentNode.isLeaf()) {
        System.out.print(currentNode.getData()+" ");
        myGuess = input.nextLine().toLowerCase();
        if(myGuess.equals("yes") || myGuess.equals("y")) {
          previous = currentNode;
          currentNode = (TreeNode<String>)currentNode.getLeftChild();
          path = 1;
        } else if(myGuess.equals("no") || myGuess.equals("n")){
          previous = currentNode;
          currentNode = (TreeNode<String>)currentNode.getRightChild();
          path = 2;
        }
      }
      while(true) {
        System.out.print("Is it a "+currentNode.getData() +"? ");
        myGuess = input.nextLine().toLowerCase();
        if(myGuess.equals("yes")||myGuess.equals("y")) {
          System.out.println("I won!");
          break;
        } else if (myGuess.equals("no") || myGuess.equals("n")) {
          System.out.print("I Don't know, what's the correct answer? ");
          String answer = input.nextLine().replace("#","").replace(",","");
          TreeNode<String> newAnswerNode = new TreeNode<String>(answer);
          System.out.print("Distinguishing Question: ");
          //Input sanitisation for saving the tree later
          String newQuestion = input.nextLine().replace("#","").replace(",","");
          while(true) {
            System.out.print("Correct Answer for " + currentNode.getData() + ": ");
            String option = input.nextLine().toLowerCase();
            if(option.equals("yes") || option.equals("y")) {
              TreeNode<String> newQuestionNode = new
               ↪  TreeNode<String>(newQuestion,currentNode,newAnswerNode);
              if(previous != null) {
                if(path == 1) {
                  //Yes
                  previous.setLeftChild(newQuestionNode);
                } else {
                  previous.setRightChild(newQuestionNode);
                }
              } else {
                questionTree = newQuestionNode;
              }
              break;
            } else if(option.equals("no") || option.equals("n")) {
              TreeNode<String> newQuestionNode = new
               ↪  TreeNode<String>(newQuestion,newAnswerNode,currentNode);
              if(previous != null) {
                if(path == 1) {
                  //Yes
                  previous.setLeftChild(newQuestionNode);
                } else {
                  //No
                  previous.setRightChild(newQuestionNode);
                }
              } else {
                //Accounting for a tree with a single leaf and nothing else
                questionTree = newQuestionNode;
              }
              break;
            } else {
              System.out.println("Please Input a valid response!");
            }
          }
          break;
        } else {
          System.out.println("Please enter a valid response!");
        }
      }
      while(true) {
        System.out.println("Would you like to:\n a) Play Again?\n b) Save the Tree?\n c)
         ↪  Load another Tree?\n d) Quit?");
        myGuess = input.nextLine().toLowerCase();
        if(myGuess.equals("a")) {
          //Play Again
          previous = null;
          currentNode = questionTree;
          path = 0;
          break;
        } else if(myGuess.equals("b")) {
          saveTree(questionTree);
          System.out.println("Tree Saved!");
        } else if(myGuess.equals("c")) {
          System.out.println("Loading Tree");
          questionTree = loadTree();

        } else if(myGuess.equals("d")) {
          isRunning = false;
          break;
        }
      }
    }
    input.close();
  }
}
```

3 Testing



```
[daniel@Void3 Assignment 5]$ java Main
Is it a mammal? yes
Is it a bird? yes
Can it fly? yes
Is it a Parrot? yes
I won!
Would you like to:
 a) Play Again?
 b) Save the Tree?
 c) Load another Tree?
 d) Quit?
a
Is it a mammal? yes
Is it a bird? no
Does it live in the Jungle?  no
Is it a Dog? no
I Don't know, what's the correct answer? Sheep
Distinguishing Question: Does it eat grass?
Correct Answer for Dog: No
Would you like to:
 a) Play Again?
 b) Save the Tree?
 c) Load another Tree?
 d) Quit?
b
Tree Saved!
Would you like to:
 a) Play Again?
 b) Save the Tree?
 c) Load another Tree?
 d) Quit?
c
Loading Tree
Would you like to:
 a) Play Again?
 b) Save the Tree?
 c) Load another Tree?
 d) Quit?
a
Is it a mammal? Yes
Is it a bird? No
Does it live in the Jungle?  no
Does it eat grass? yes
Is it a Sheep? yes
I won!
Would you like to:
 a) Play Again?
 b) Save the Tree?
 c) Load another Tree?
 d) Quit?
d
```