Lab 1

WAP to guess the animal name

Background:

The game is played between the user and the computer. The computer generates the result according to the input the computer enters. It matches the condition user enters and finally generate the conclusion.

The program is implemented using binary search tree. The binary search tree is used to search the correct answer in the leaf of the tree. The non-leaf nodes are the question and the leaf gives the answer of the question. The questions that are added to train the program are stored in the leaf.

Methodology:

This program is written in Java and is implemented using different classes and methods. Various techniques of data structure and algorithm is also implemented in this program. The questions are stored in this program. If the question asked are answered correctly other different questions are asked as well to make sure the guess is right and other question are asked to be added if the guesses are wrong till the program is terminated. The questions are in the form of yes/no questions.

For example

I have made a program to guess the bird. User enters the name of the bird and if the condition matches the answer user is seeking then, it gives the output

Program:

import java.util.Scanner;

public class BIRD {

private static Scanner stdin = new Scanner(System.in);

public static void main(String[ ] args)

{

BTNode<String> root;

instruct( );

root = beginningTree( );

do

play(root);

while (query("Lets try again playing,shall we?"));

System.out.println("Thank you so much.");

}

public static void instruct( )

{

System.out.println("Please think of a bird.");

System.out.println("Let me help you with some guessing");

System.out.println("to find your intelligence in AI");

}

public static void play(BTNode<String> current)

{

while (!current.isLeaf( ))

{

if (query(current.getData( )))

current = current.getLeft( );

else

current = current.getRight( );

}

System.out.print("Let me guess " + current.getData( ) + ". ");

if (!query("Does it match?"))

learn(current);

else

System.out.println("I am genius!");

}

public static BTNode<String> beginningTree( )

{

BTNode<String> root;

BTNode<String> child;

final String ROOT\_QUESTION = "Are you a bird?";

final String LEFT\_QUESTION = "Can you fly?";

final String RIGHT\_QUESTION = "Do you prey?";

final String BIRD1 = "Eagle";

final String BIRD2 = "Vulture";

final String BIRD3 = "Duck";

final String BIRD4 = "Crow";

root = new BTNode<String>(ROOT\_QUESTION, null, null);

child = new BTNode<String>(LEFT\_QUESTION, null, null);

child.setLeft(new BTNode<String>(BIRD1, null, null));

child.setRight(new BTNode<String>(BIRD2, null, null));

root.setLeft(child);

child = new BTNode<String>(RIGHT\_QUESTION, null, null);

child.setLeft(new BTNode<String>(BIRD3, null, null));

child.setRight(new BTNode<String>(BIRD4, null, null));

root.setRight(child);

return root;

}

public static void learn(BTNode<String> current)

{

String guessBIRD;

String correctBIRD;

String newQuestion;

guessBIRD = current.getData( );

System.out.println("I QUIT!! ");

correctBIRD = stdin.nextLine( );

System.out.println("YES||NO");

System.out.println("Please type a distinct answer to distinctly find");

System.out.println(correctBIRD + " from " + guessBIRD + ".");

newQuestion = stdin.nextLine( );

current.setData(newQuestion);

System.out.println(" " + correctBIRD + ", " + newQuestion);

if (query("Please answer"))

{

current.setLeft(new BTNode<String>(correctBIRD, null, null));

current.setRight(new BTNode<String>(guessBIRD, null, null));

}

else

{

current.setLeft(new BTNode<String>(guessBIRD, null, null));

current.setRight(new BTNode<String>(correctBIRD, null, null));

}

}

public static boolean query(String prompt)

{

String answer;

System.out.print(prompt + " YES||NO: ");

answer = stdin.nextLine( ).toUpperCase( );

while (!answer.startsWith("YES") && !answer.startsWith("NO"))

{

System.out.print("ERROR!!!ANSWER WITH YES OR NO: ");

answer = stdin.nextLine( ).toUpperCase( );

}

return answer.startsWith("YES");

}

public static class BTNode<E>

{

private E data;

private BTNode<E> left, right;

public BTNode(E initialData, BTNode<E> initialLeft, BTNode<E> initialRight)

{

data = initialData;

left = initialLeft;

right = initialRight;

}

public E getData( )

{

return data;

}

public BTNode<E> getLeft( )

{

return left;

}

public E getLeftmostData( )

{

if (left == null)

return data;

else

return left.getLeftmostData( );

}

public BTNode<E> getRight( )

{

return right;

}

public E getRightmostData( )

{

if (left == null)

return data;

else

return left.getRightmostData( );

}

public void inorderPrint( )

{

if (left != null)

left.inorderPrint( );

System.out.println(data);

if (right != null)

right.inorderPrint( );

}

public boolean isLeaf( )

{

return (left == null) && (right == null);

}

public void preorderPrint( )

{

System.out.println(data);

if (left != null)

left.preorderPrint( );

if (right != null)

right.preorderPrint( );

}

public void postorderPrint( )

{

if (left != null)

left.postorderPrint( );

if (right != null)

right.postorderPrint( );

System.out.println(data);

}

public void print(int depth)

{

int i;

for (i = 1; i <= depth; i++)

System.out.print(" ");

System.out.println(data);

if (left != null)

left.print(depth+1);

else if (right != null)

{

for (i = 1; i <= depth+1; i++)

System.out.print(" ");

System.out.println("--");

}

if (right != null)

right.print(depth+1);

else if (left != null)

{

for (i = 1; i <= depth+1; i++)

System.out.print(" ");

System.out.println("--");

}

}

public BTNode<E> removeLeftmost( )

{

if (left == null)

return right;

else

{

left = left.removeLeftmost( );

return this;

}

}

public BTNode<E> removeRightmost( )

{

if (right == null)

return left;

else

{

right = right.removeRightmost( );

return this;

}

}

public void setData(E newData)

{

data = newData;

}

public void setLeft(BTNode<E> newLeft)

{

left = newLeft;

}

public void setRight(BTNode<E> newRight)

{

right = newRight;

}

public static <E> BTNode<E> treeCopy(BTNode<E> source)

{

BTNode<E> leftCopy, rightCopy;

if (source == null)

return null;

else

{

leftCopy = treeCopy(source.left);

rightCopy = treeCopy(source.right);

return new BTNode<E>(source.data, leftCopy, rightCopy);

}

}

public static <E> long treeSize(BTNode<E> root)

{

if (root == null)

return 0;

else

return 1 + treeSize(root.left) + treeSize(root.right);

}

}

}

Output:

