Cathal McSweeney : 19731485

**OOP ASSIGNMENT FIVE – EXPANDABLE BINARY TREE**

**Problem Statement:**

For this assignment we are faced with creating a tree of objects that tries to guess what a user is thinking of. It is to be expandable so that if the tree is unable to guess what a user is thinking of then through a series of prompts a user is able to add to the tree. The tree is expandable in that there is no limit to the number of additions that a user can make to the tree. Once a correct guess is found the user is then prompted to make a selection choosing to, play again, save the tree, load a tree, or exit. Incorporating the various Binary Tree classes as provided from blackboard, we altered the “BinaryTreeDemo” class in order to achieve this.

**Analysis and Design Notes**

Pseudocode :

* Main Method

Initialize binary tree object

Create starting binary tree by calling createTree method

Prompt user to load saved tree or not

Display stats on tree being used

Display nodes in tree

Call questions method to start yes no questions for user

* Method One
* Method Two

Public Boolean elementByElementCheck(input){

Palindrome boolean

String forward

Int length of input

For(length of input){

If(forwardCharat(i)==forwardCharAt(length)){

Boolean = true

Length--

}

Else{

Boolean = false

break

}

}

Return boolean

}

* Method Three

Public Boolean stackAndQueue(input){

Palindrome boolean

String input

For(length of input){

Push(input per char)

Enqueue(input per char)

}

While(stack is not empty){

If(pop == dequeue){

continue

}

Else{

Boolean is false

Empty queue and stack

}

}

Return boolean

}

* Method Four

Public Boolean elementByElementCheck(input){

Palindrome boolean

String forward = input

String reverse = reverse(input)

If(forward = reverse){

Boolean = true

}

Else{

Boolean = false

}

Return boolean

}

* Recursive Method Reverse

Public string reverse(input){

If(input is empty){

Return input

}

Return reverse(input.substring(1)) + first char

}

* Decimal to Binary Method

Public string stringToBinary(number){

String binary

Int array[]

Int index

While(num>0){

Array[index++] = num%2

Num = num/2

}

For(length of index, decrement each iteration){

string += array[i]

}

Return string

}

Flow of control

The order in which the code runs is the main class instantiates the class by calling the palindromes method.

This method begins with creating 1 million objects from the DecimalBinaryPalindromes class which contains only 2 Boolean values. Next it gets the binary representation of each number between 0 and 1 million, storing them in an array of strings. Next a variable gets the start time, followed by a loops that iterates through the decimal numbers 0 to 1 million which calls the reversedString method. In this method the string is reversed and using the .equals method identifies if it is a palindrome. Each time a palindrome is found a Boolean value of true is returned and a counter is iterated and the array of palindrome objects decimal Boolean is changed to true. Upon completion of the loop an end time is taken and the total time calculated. Next we have again a timer getting the start time followed by a loop of 1 million iterations starting at 0. The string being passed to the reversedString method is the binary representation of the decimal number. When a palindrome is found then a Boolean value is returned. The binary Boolean value of the palindrome is then changed to true. Upon completion of the loop a end time is taken and the total time calculated.

We then move onto testing method two which has the same layout as the testing for method one. Takes the time before commencing the loop to a million where each number is passed to the method called elementByElement. In this method the first char of a string is compared to the last char of the same string. If they match it continues to check the second char to the second last char and so on. If a non-match is found the method Returns a false Boolean value. If all are matches for the string are the same a true Boolean value is returned. The End time is taken and then the total time calculated. The binary version follows the same loop this time passing the binary representation as a string into the method with the return Boolean values being returned if it is a palindrome or not.

The third method is tested using the timing method and loop as the previous two methods. Both decimal and binary are passed to the method in separate loops with the return values being either a true or false boolean value depending if the string is a palindrome or not. The third method called stackAndQueue utilizes the ArrayStack and ArrayQueue classes by pushing and enqueueing the input string character by character to both the stack and queue. The program then uses the pop and dequeue methods of these classes to remove the first and last char of the same string and compare them. If they match the code continues to check the length of the string if a set doesn’t match then it exits the code and returns a false boolean value.

The Fourth and final method also follows the same format with the time being taken before and after the loop to calculate the total time taken. The loop the iterates one million times for both the binary and decimal representations of all numbers between 0 and one million. The method take in a string as an input, stores it and uses a recursive function called reverse to store a reversed version of the input. The recursive function works by taking the input, calling reverse again with the input being a substringing from the second char on of the current input string until there is an empty string being sent in which case it then returns the chars with each being appended to the end of the return string. Once method four gets the reversed string it compares it to the original using .equals. If a palindrome it returns a true Boolean value, if not it returns a false.

Finally the results are printed to screen using a JOptionPane.

**Testing:**

To ensure that the testing of the methods was correct and that it was correctly identifying which decimal and binary numbers were palindromes we used a counter that would be iterated each time a palindrome was found in each of the methods. The reasoning for this was that palindrome decimal numbers between 0 and 1 million consist of a total of 1999 palindromes with the binary equivalent having 200 palindromes between 0 and 1 million. The time taken to calculate each of the method 1 million times was also calculated identifying which of the methods was the fastest at carrying out identification of which number were palindromes. Using the utility class that stored the Boolean values of decimal numbers found to be palindromes and binary numbers that are palindromes it is possible to find the 20 palindromes between 0 and 1 million that have both a decimal number as a palindrome and a binary representation which too is a palindrome.

Text

Description automatically generated

Text

Description automatically generated

In counting the primitive operations of each method four global variables are created to be populated with the number of primitive operations carried out. We count the number of primitive operations in each method using a standard procedure throughout all four methods including any utility methods that are utilized. Throughout the running of each method the current count of a counter is printed to the console every 50,000 iterations throughout the loop to be used in graphing the methods and comparing later.

Method One has a counter called “n1b” this is incremented by one each time a line is called. The for loop that is present in this method contains 2 counters which account for the single line of code in the loop and for the line read each time the loop parameters are checked. The remainder of the code also increments the counter by one per line checked.

Method Two follows the same rules as method one, each line carries a counter weight of 1. This method contains a loop which increments the counter everytime its conditions are checked with the if statement within the loop also incrementing the counter once per check, with its contents carrying the weight of one incrementation per line.

Method Three for the most part follows the same rules for incrementating the counter as the previous two methods, however due to the implementation of the ArrayStack and ArrayQueue classes and the calling of their isEmpty, push, pop, enqueue and dequeue methods a different approach was required. The push and enqueue methods when called both carry a weight of 4 due to the number of lines that these methods consisted of. The while loop due to utilizing the isEmpty method is given the incrementation weight of 2 as a result of only containing a single line. The if statement that is contained within the while loop has a changing weight that is dependent on the length of the string that has been passed into the method, this is due to the dequeue method requiring re-shuffling each time it is called to the length of the string. The else statement also utilizes this incrementation in the case a non-palindrome is found in order to empty the remaining contents of the stack and queue.

Method Four increments the counter similar to methods one and two for the most part. However when it comes to the recursive method utilized for method four. The return line increments the counter by 3 due to the use of the substring method, the charAt method and the calling of the recursive method.

Using the console data we developed an excel spreadsheet that helped in developing the graphs to demonstrate the difference in primitive operations for each of the methods used.

Text

Description automatically generated with medium confidenceText

Description automatically generated with medium confidenceGraphical user interface, text

Description automatically generatedGraphical user interface, text

Description automatically generated with medium confidence

Graphical user interface, text

Description automatically generatedText

Description automatically generated with low confidenceGraphical user interface, text

Description automatically generatedGraphical user interface, text

Description automatically generated

Chart, line chart

Description automatically generated**Chart, line chart

Description automatically generated**

From this data we can see that method three is the most demanding in the number of primitive operations which is backed up from the results returned for the timing it takes to carry out method threes functions both in decimal and in binary. This is also true for all the methods tested as the number of primitive operations and the time taken match for which method takes the longest has the highest number of operations with the method taking the shortest amount of time containing the least number of primitive operations.

**CODE**

import java.awt.\*;

import javax.swing.JOptionPane;

public class Palindromes {

private boolean isPalindrome = false;

private ArrayStack stack = new ArrayStack();

private ArrayQueue queue = new ArrayQueue();

private DecimalBinaryPalindromes[] palindromes = new DecimalBinaryPalindromes[1000000];

//

private long test1Decimal;

private long test1Binary;

private long test2Decimal;

private long test2Binary;

private long test3Decimal;

private long test3Binary;

private long test4Decimal;

private long test4Binary;

//used to get num of operations at intervals

private int gNo = 50000;

//counter for binary operations

private int n1b=0;

private int n2b=0;

private int n3b=0;

private int n4b=0;

public static void main(String []args) {

Palindromes test = new Palindromes();

}

public Palindromes() {

long start, end;

String decimal;

String output="";

//initialise array of objects with 2 booleans to identify which numbers have both palindromes

for(int i = 0; i < palindromes.length; i++) {

palindromes[i] = new DecimalBinaryPalindromes();

}

//Initialise array of binary numbers

String[] binaryNums = new String[1000000];

for(int i=0;i<1000000;i++) {

binaryNums[i]=stringToBinary(i);

}

System.out.println("METHOD ONE DECIMAL");

//decimal test for first method

start = System.currentTimeMillis();//start time for method

output+="----------Method One----------\n";

//counts number of decimal palindroms returned

int dp1 =0;

for(int i=0;i<1000000;i++) {

if(reversedString(decimal =""+ i)) {

palindromes[i].decimal = true;

dp1++;

}

//printing to console numbers for graphs

if(i%gNo == 0 && i >0) {

System.out.println(n1b);

}

}

end = System.currentTimeMillis();

test1Decimal = end - start;//finds total time taken for the method

output += "decimal time Taken ="+test1Decimal+"\ndecimal palindromes found = "+dp1;

System.out.println("Method total= "+n1b);

n1b=0;

System.out.println("METHOD ONE BINARY");

//binary test for first method

start = System.currentTimeMillis();

int bp1 = 0;

for(int i=0;i<1000000;i++) {

if(reversedString(binaryNums[i])) {

palindromes[i].binary = true;

bp1++;

}

if(i%gNo == 0 && i >0) {

System.out.println(n1b);

}

}

end = System.currentTimeMillis();

test1Binary = end - start;

output += "\nbinary palindromes found = "+bp1+"\nbinary time taken = "+test1Binary;

System.out.println("Method total= "+n1b);

//------------------------------------------------------------------------------------------

//Decimal test for Second method

start = System.currentTimeMillis();

output+="\n----------Method Two----------\n";

int test=0;

System.out.println("METHOD TWO DECIMAL");

int dp2 = 0;

for(int i=0;i<1000000;i++) {

if(elementByElementCheck(decimal =""+ i)) {

dp2++;

}

if(i%gNo == 0 && i >0) {

System.out.println(n2b);

}

}

end = System.currentTimeMillis();

test2Decimal = end - start;

output += "decimal time Taken ="+test2Decimal+"\ndecimal palindromes found = "+dp2;

System.out.println("Method total= "+n2b);

n2b=0;

//binary test for Second method

start = System.currentTimeMillis();

System.out.println("METHOD TWO BINARY");

int bp2 =0;

for(int i=0;i<1000000;i++) {

if(elementByElementCheck(binaryNums[i])) {

bp2++;

}

if(i%gNo == 0 && i >0) {

System.out.println(n2b);

}

}

end = System.currentTimeMillis();

test2Binary = end - start;

output += "\nbinary palindromes found = "+bp2+"\nbinary time taken = "+test2Binary;

System.out.println("Method total= "+n2b);

//------------------------------------------------------------------------------------------

//Decimal test for Third method

start = System.currentTimeMillis();

output+="\n----------Method Three----------\n";

System.out.println("METHOD THREE DECIMAL");

int dp3 = 0;

for(int i=0;i<1000000;i++) {

if(stackAndQueue(decimal =""+ i)) {

dp3++;

}

if(i%gNo == 0 && i >0) {

System.out.println(n3b);

}

}

end = System.currentTimeMillis();

test3Decimal = end - start;

output += "decimal time Taken ="+test3Decimal+"\ndecimal palindromes found = "+dp3;

System.out.println("Method total= "+n3b);

n3b = 0;

//binary test for Third method

start = System.currentTimeMillis();

System.out.println("METHOD THREE BINARY");

int bp3 =0;

for(int i=0;i<1000000;i++) {

if(stackAndQueue(binaryNums[i])) {

bp3++;

}

if(i%gNo == 0 && i >0) {

System.out.println(n3b);

}

}

end = System.currentTimeMillis();

test3Binary = end - start;

output += "\nbinary palindromes found = "+bp3+"\nbinary time taken = "+test3Binary;

System.out.println("Method total= "+n3b);

//------------------------------------------------------------------------------------------

//Decimal test for Fourth method

start = System.currentTimeMillis();

output+="\n----------Method Four----------\n";

System.out.println("METHOD FOUR DECIMAL");

int dp4 =0;

for(int i=0;i<1000000;i++) {

if(reverseUsingRecursion(decimal =""+ i)) {

dp4++;

}

if(i%gNo == 0 && i >0) {

System.out.println(n4b);

}

}

end = System.currentTimeMillis();

test4Decimal = end - start;

output += "decimal time Taken ="+test4Decimal+"\ndecimal palindromes found = "+dp4;

System.out.println("Method total= "+n4b);

n4b = 0;

//binary test for Fourth method

start = System.currentTimeMillis();

System.out.println("METHOD FOUR BINARY");

int bp4 =0;

for(int i=0;i<1000000;i++) {

if(reverseUsingRecursion(binaryNums[i])) {

bp4++;

}

if(i%gNo == 0 && i >0) {

System.out.println(n4b);

}

}

end = System.currentTimeMillis();

test4Binary = end - start;

output += "\nbinary palindromes found = "+bp4+"\nbinary time taken = "+test4Binary;

System.out.println("Method total= "+n4b);

String bothPalindromes="Decimal & Binary eqivalent which are palindromes\n";

int i =0;

//iterates through the array of objects and adds those with both boolean values

//which are true to the string to display which have both numbers as palindromes

for(DecimalBinaryPalindromes curObj: palindromes) {

if(curObj.binary==true && curObj.decimal == true) {

bothPalindromes+= i+" and "+binaryNums[i]+"\n";

}

i++;

}

//shows the total time taken and number of palindromes found for each of the methods

JOptionPane.showMessageDialog(null,""+output);

//shows the numbers that are palindromes both in binary and decimal

JOptionPane.showMessageDialog(null,""+bothPalindromes);

}

public boolean reversedString(String s) {

isPalindrome = false; n1b++;

String normal = s; n1b++;

String reversed =""; n1b++;

//iterates through the length of the input and reverses it

for(int i = normal.length()-1 ; i >= 0 ; i--,n1b++) {

reversed += normal.charAt(i);

n1b++;

}

n1b++;

//compares the two different strings sets return boolean to true

if(normal.equals(reversed)) {

isPalindrome = true;n1b++;

}

n1b++;

return isPalindrome;

}

public boolean elementByElementCheck(String s) {

isPalindrome = false; n2b++;

String forward = s; n2b++;

int length = forward.length()-1; n2b++;

//checks the first digit against the last digit needing to only iterate through half the digits

for(int i = 0 ; i < forward.length() ; i++, n2b++) {

n2b++;

if(forward.charAt(i)==forward.charAt(length)) {

length--; n2b++;

isPalindrome = true;n2b++;

}

//if a non-palindrome is found returns false

else {

n2b++;

isPalindrome = false;n2b++;

n2b++;

break;

}

}

n2b++;

return isPalindrome;

}

public boolean stackAndQueue(String s) {

isPalindrome = true; n3b++;

String input = s; n3b++;

//pushes the numbers both onto the stack and into the queue

for(int i =0 ; i < input.length(); i++,n3b++) {

stack.push(input.charAt(i)); n3b+=4;

queue.enqueue(input.charAt(i)); n3b+=4;

}

n3b++;

//both being the same length we only need to check one

while(stack.isEmpty()!=true) {

n3b+=2;

//popping from the top of the stack (first digit) and comparing against the dequeue

//(the last digit) to see if they are the same

if(stack.pop()==queue.dequeue()) {

n3b+=(input.length()+9);

}

else {

n3b++;

isPalindrome = false;n3b++;

while(stack.isEmpty()!=true) {n3b++;

stack.pop();

queue.dequeue();

n3b+=(input.length()+9);

}

return isPalindrome;

}

}

n3b++;

return isPalindrome;

}

public boolean reverseUsingRecursion(String s) {

isPalindrome = false; n4b++;

String forward = s; n4b++;

//returns the reversed version of the number using the recursion method

String reversed = reverse(forward); n4b++;

n4b++;

//checks

if(forward.equals(reversed)) {

n4b++;

isPalindrome = true;

}

else {

n4b++;

isPalindrome = false;

}

n4b++;

return isPalindrome;

}

public String reverse(String s) {

n4b++;

if(s.isEmpty()) {

n4b++;

return s;

}

n4b+=3;

//creates a recursive method until the length of the number has been iterated through

//returning the reversed version of the number

return reverse(s.substring(1)) +s.charAt(0);

}

//creates the binary version of a decimal number using modulus

public String stringToBinary(int num) {

String binaryReturn = "";

if(num==0) {binaryReturn = "0";}

int binary[] = new int[40];

int index = 0;

while(num > 0){

binary[index++] = num%2;

num = num/2;

}

for(int i = index-1;i >= 0;i--){

binaryReturn += "" + binary[i];

}

return binaryReturn;

}

}

**Utility Class DecimalBinaryPalindromes**

public class DecimalBinaryPalindromes {

protected boolean decimal=false;

protected boolean binary=false;

}