

**Department of Computer Science**

**2019-20**

**M.Sc. – IT (Part - I)**

Course: Data Structure and Algorithms

Course Code: MIT 11

Name: Akshay Dhargalkar

Roll No: SP192209

**Index**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Date** | **Title** | **Page No.** |
|  | 01-07-2019 | Number of groups based on height. | 3 |
|  | 03-07-2019 | Maximum Length of Subsequence. | 5 |
|  | 08-07-2019 | Appearance of “saba” in n\*m grid. | 7 |
|  | 04-07-2019 | Instagram Dimension Restrictions. | 9 |
|  | 04-07-2019 | Pattern in 2D Array. | 12 |
|  | 04-07-2019 | ASCII Magic Numbers. | 14 |
|  | 10-07-2019 | Tower of Hanoi. | 17 |
|  | 10-07-2019 | Staircase. | 19 |
|  | 15-07-2019 | Maximum Element in the Stack. | 21 |
|  | 15-07-2019 | Balanced Parenthesis. | 23 |
|  | 17-07-2019 | Stack of Books. | 25 |
|  | 24-07-2019 | Linked List Loop Detection and Breaking. | 28 |
|  | 29-07-2019 | Rotating the Linked List. | 31 |
|  | 31-07-2019 | Binary Search Tree. | 34 |
|  | 11-09-2019 | AVL Tree. | 37 |
|  | 25-09-2019 | Huffman Encoding. | 42 |
|  | 25-09-2019 | Open Addressing. | 45 |

**1) Lab Session 1 - Problem 1.**

Date: 01-07-2019

Problem Statement:

Due to the demonetization move, there is a long queue of people in front of ATMs. Due to withdrawal limit per person per day, people come in groups to withdraw money. Groups come one by one and line up behind the already present queue. The groups have a strange way of arranging themselves. In a particular group, the group members arrange themselves in increasing order of their height (not necessarily strictly increasing).

Swapy observes a long queue standing in front of curious kid, he wants to count the total number of groups present in the queue waiting to withdraw money. Since groups are standing behind each other, one cannot differentiate between different groups and the exact count cannot be given. Can you tell him the minimum number of groups that can be observed in the queue?

Source Code:

package test;

import java.util.Scanner;

/\*\*

\*

\* @author Akshay

\*/

public class GroupHeight {

public static void main(String[] args) {

Scanner h = new Scanner(System.in);

int i, j, group=1, numOfPeople;

System.out.println("Enter the number of people :");

numOfPeople=h.nextInt();

int height[] = new int[numOfPeople];

System.out.println("Enter height");

for(i=0;i<numOfPeople;i++) {

height[i]=h.nextInt();

}

for(i=0;i<numOfPeople;i++) {

j=i+1;

if(j==numOfPeople)

break;

if(height[i]>height[j]) {

group=group+1;

}

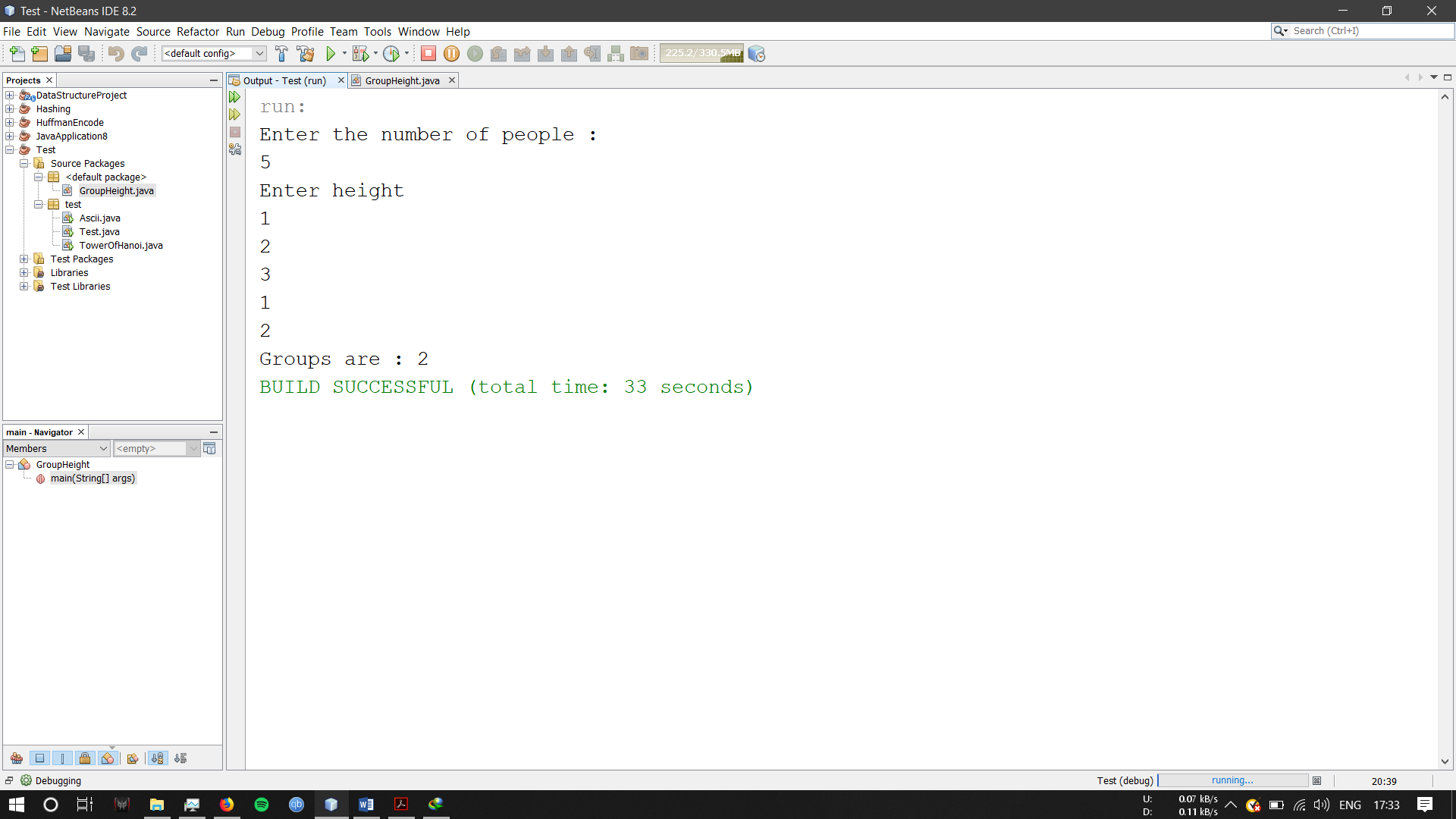
}

System.out.println("Groups are : "+group);

}

}

Screenshot of Output:



References:

<https://www.geeksforgeeks.org/arrays-in-java/>

**2) Lab Session 1 - Problem 2.**

Date: 03-07-2019

Problem Statement:

You are given an array A of length N. You need to find maximum length of subsequence of A which contains anyone digit common in all the elements of that subsequence. You are not allowed to count leading zeroes.

Source Code:

import java.util.\*;

public class SubsequenceOfArray {

public static void main(String[] args) {

Scanner h=new Scanner(System.in);

int i,n,t1,ans=0,st,t2;

System.out.println("Number of elements:");

n=h.nextInt();

int num1[] = new int[n];

int num2[][] = new int[10][n];

System.out.println("Enter elements");

for(i=0;i<n;i++) {

num1[i] = h.nextInt();

}

for(i=0;i<10;i++) {

for(int j=0;j<n;j++){

t1=num1[j];

while(t1>0) {

t2=t1%10;

num2[t2][j]=1;

t1=t1/10;

}

}

}

for(int k=0;k<10;k++) {

for(int p=0;p<n;p++) {

//System.out.println(num2[k][p]);

}

}

for(int u=0;u<10;u++) {

st=0;

for(int v=0;v<n;v++) {

if(num2[u][v]==1) {

st=st+1;

}

}

ans=Math.max(st, ans);

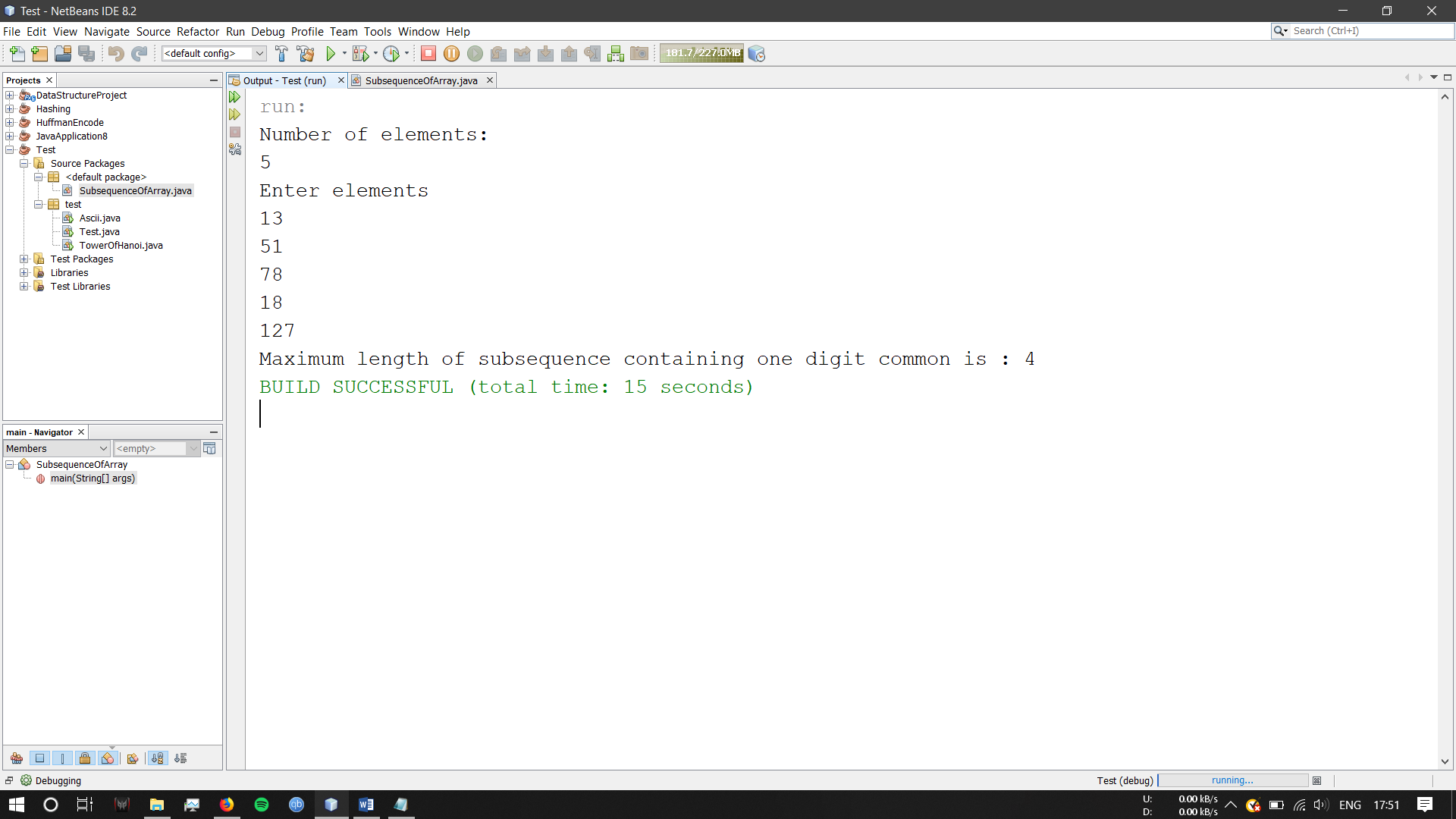
}

System.out.println("Maximum length of subsequence containing one digit common is : " + ans);

}

}

Screenshot of Output:



References:

<https://www.geeksforgeeks.org/java-math-max-method-examples/>

<https://www.geeksforgeeks.org/collections-max-method-in-java-with-examples/>

<https://www.youtube.com/watch?v=etyrkipdKvc>

**3) Lab Session 1 - Problem 3.**

Date: 08-07-2019

Problem Statement:

You are given an n\*m grid which contains lower case English letters. How many times does the phrase & “saba” appear horizontally, vertically, and diagonally in the grid?

Source Code:

import java.util.\*;

public class Saba {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

System.out.println("Enter Number of Rows : ");

int rows=sc.nextInt();

System.out.println("Enter Number of Columns : ");

int columns=sc.nextInt();

char a[][]=new char[rows][columns];

String x;

for(int i=0;i<rows;i++)

{

System.out.println("Enter the string elements :");

x=sc.next();

char s[]=x.toCharArray();

for(int j=0;j<columns;j++)

{

a[i][j]=s[j];

}

}

int c=0;

for(int i=0;i<rows;i++)

{

for(int j=0;j<columns-3;j++)

{

if(a[i][j]=='s' && a[i][j+1]=='a' && a[i][j+2]=='b' && a[i][j+3]=='a')

c++;

}

}

for(int i=0;i<rows-3;i++)

{

for(int j=0;j<columns;j++)

{

if(a[i][j]=='s' && a[i+1][j]=='a' && a[i+2][j]=='b' && a[i+3][j]=='a')

c++;

}

}

for(int i=0;i<rows-3;i++)

{

for(int j=0;j<columns-3;j++)

{

if(a[i][j]=='s' && a[i+1][j+1]=='a' && a[i+2][j+2]=='b' && a[i+3][j+3]=='a')

c++;

}

}

for(int i=rows-1;i>2;i--)

{

for(int j=0;j<columns-3;j++)

{

if(a[i][j]=='s' && a[i-1][j+1]=='a' && a[i-2][j+2]=='b' && a[i-3][j+3]=='a')

c++;

}

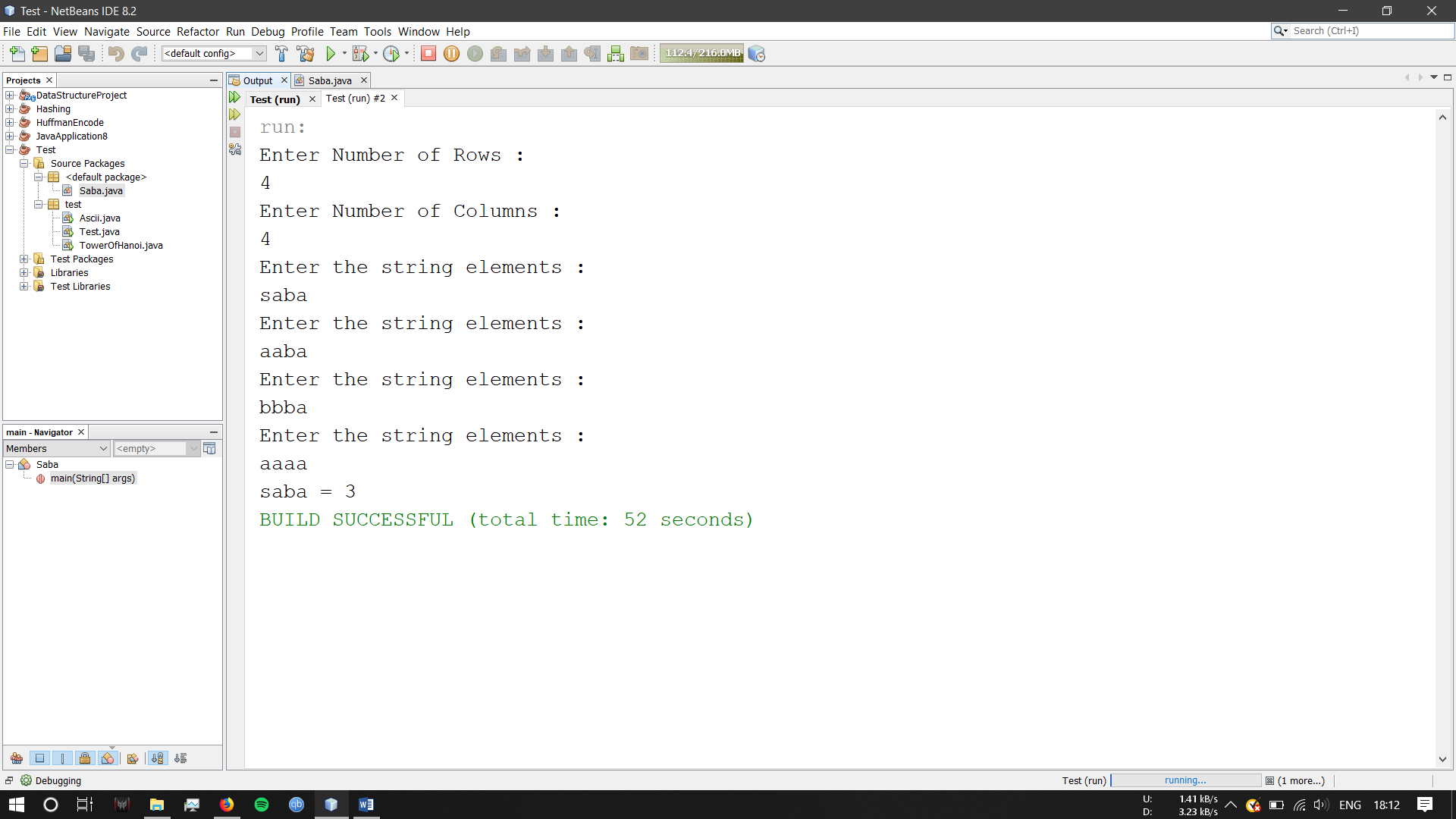
}

System.out.println(c);

}

}

Screenshot of Output:



References:

<https://www.geeksforgeeks.org/system-arraycopy-in-java/>

<https://www.hackerearth.com/submission/31359294/>

<https://www.programiz.com/java-programming/multidimensional-array>

**4) Lab Session 2, Problem 1.**

Date: 04-07-2019

Problem Statement:

Mr.X wants to change his profile picture on Instagram. Now Instagram has some restriction over the dimension of picture that we can upload.

Minimum dimension of the picture can be L x L, where L is the length of the side of square.

Now Roy has N photos of various dimensions.

Dimension of a photo is denoted as W x H

where W - width of the photo and H - Height of the photo

When any photo is uploaded following events may occur:

[1] If any of the width or height is less than L, user is prompted to upload another one. Print ‘UPLOAD ANOTHER’ in this case.

[2] If width and height, both are large enough and

(a) if the photo is already square then it is accepted. Print ‘ACCEPTED’ in this case.

(b) else user is prompted to crop it. Print &quot;CROP IT&quot; in this case.

Given L, N, W and H as input, print appropriate text as output.

Source Code:

import java.util.Scanner;

/\*\*

\*

\* @author Akshay

\*/

public class InstagramImageDimensions {

public static void main(String[] args){

Scanner in = new Scanner(System.in);

int l, n, i, w, h;

do {

System.out.println("Enter the dimension of the image : ");

while (!in.hasNextInt()) {

System.out.println("Please Enter an Integer Value!");

in.next();

}

l = in.nextInt();

} while (l <= 0);

do {

System.out.println("Enter the number of pictures you want to upload :");

while (!in.hasNextInt()) {

System.out.println("Please Enter an Integer Value!");

in.next();

}

n = in.nextInt();

} while (l <= 0);

for(i=0;i<n;i++){

do {

System.out.println("Enter the Width");

while (!in.hasNextInt()) {

System.out.println("Please Enter Integer Values!");

System.out.println("Enter the Width");

in.next();

}

w=in.nextInt();

} while (l <= 0);

do {

System.out.println("Enter the Height");

while (!in.hasNextInt()) {

System.out.println("Please Enter Integer Values!");

System.out.println("Enter the Height");

in.next();

}

h=in.nextInt();

} while (l <= 0);

if(w<l && h<l){

System.out.println("Upload Another");

}

if(w>l && h>l){

System.out.println("Crop it");

}

if(w==l && h==l){

System.out.println("Accepted");

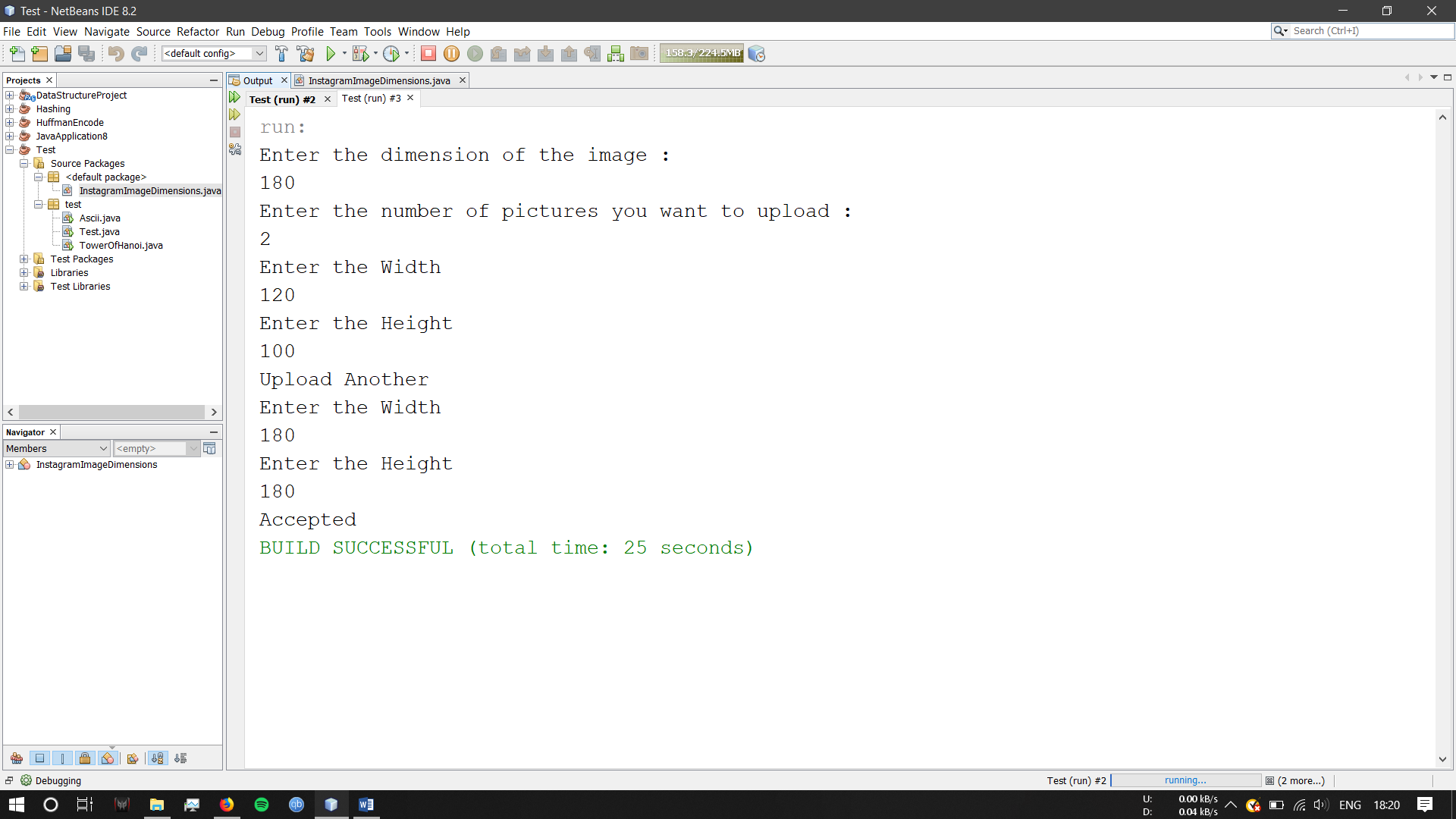
}

}

}

}

Screenshot of Output:



References:

<https://stackoverflow.com/questions/3059333/validating-input-using-java-util-scanner>

**5) Lab Session 2, Problem 2.**

Date: 04-07-2019

Problem Statement:

Given a 6\*6 2D Array

1 1 1 0 00

0 1 0 0 00

1 1 1 0 00

0 0 0 0 00

0 0 0 0 00

0 0 0 0 00

We define a pattern in 2D Array to be a subset of values with indices falling in this pattern in graphical representation:

a b c

d

e f g

There are 16 patterns in 2D Array . Calculate the sum for every pattern in the 2D Array , then print the maximum patterns sum.

Source Code:

import java.util.\*;

public class TwoDimenasionalArrayIPattern {

public static void main(String[] args){

System.out.print("Enter 2D array size : ");

Scanner sc=new Scanner(System.in);

int sum;

int previous\_sum=Integer.MIN\_VALUE;

int r=sc.nextInt();

int c=sc.nextInt();

System.out.println("Enter array elements : ");

int a[][]=new int[r][c];

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

for(int j=0; j<c;j++) {

a[i][j]=sc.nextInt();

}

}

System.out.print("\nThe Array that you entered : \n");

for(int []x:a){

for(int y:x){

System.out.print(y+" ");

}

System.out.println();

}

for(int i=0; i<4;i++){

for(int j=0; j<4;j++) {

sum=a[i][j]+a[i][j+1]+a[i][j+2]+a[i+1][j+1]+a[i+2][j]+a[i+2][j+1]+a[i+2][j+2];

previous\_sum=Math.max(previous\_sum, sum);

}

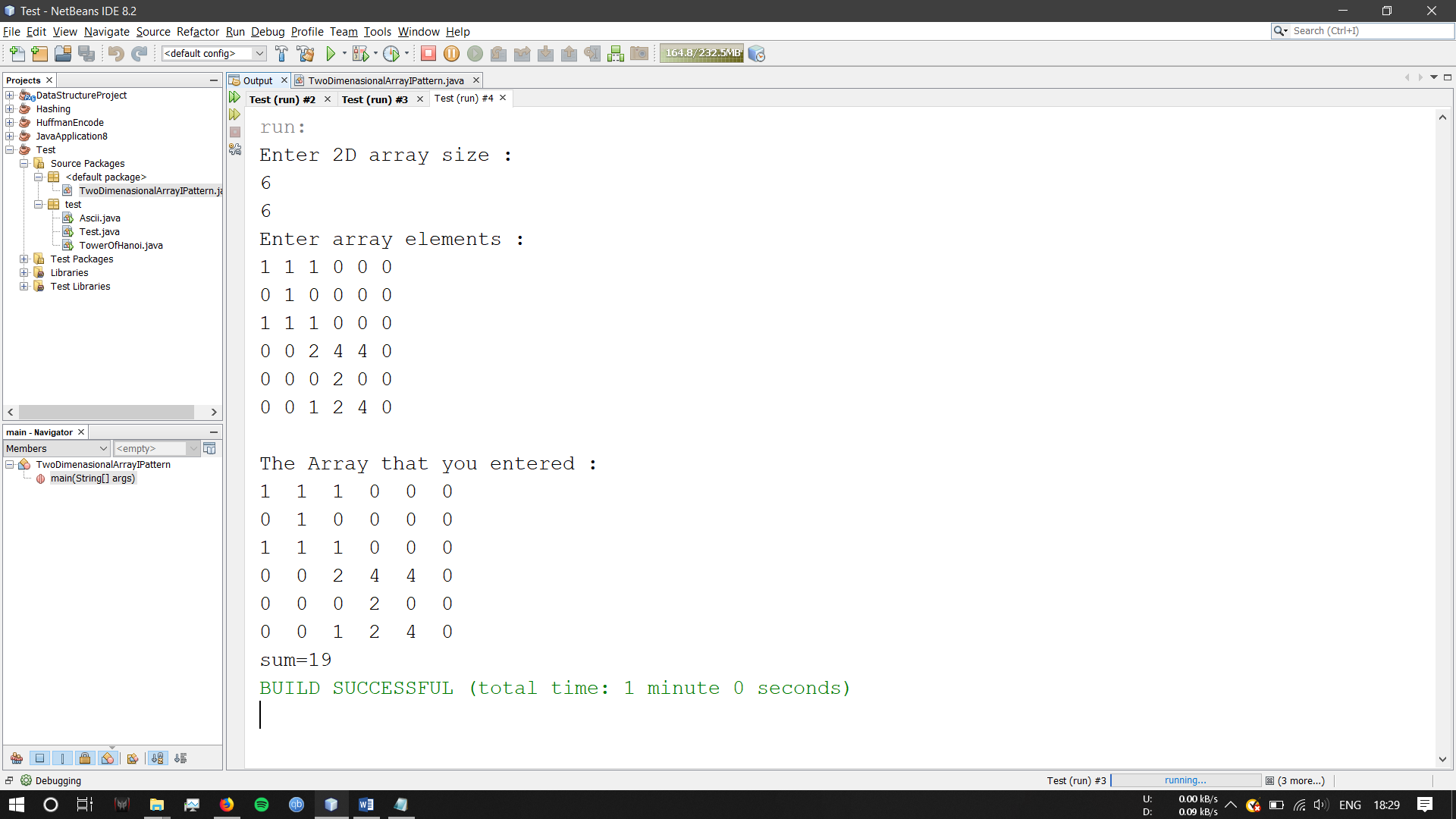
}

System.out.println("sum="+previous\_sum);

}

}

Screenshot of Output:



References:

<https://gist.github.com/eMahtab/297419cb805fe93346c0>

<https://stackoverflow.com/questions/16050222/put-a-range-on-the-input-of-an-array-in-java>

**6) Lab Session 2, Problem 3.**

Date: 04-07-2019

Problem Statement:

Dhananjay has recently learned about ASCII values.He is very fond of experimenting. With his knowledge of ASCII values and character he has developed a special word and named it Dhananjays Magical word.

A word which consist of alphabets whose ASCII values is a prime number is an Dhananjay&#39;s Magical word. An alphabet is Dhananjays Magical alphabet if its ASCII value is prime.

Dhananjays nature is to boast about the things he know or have learnt about. So just to defame his friends he gives few string to his friends and ask them to convert it to Dhananjays Magical word. None of his friends would like to get insulted. Help them to convert the given strings to Dhananjays Magical Word.

Rules for converting:

1. Each character should be replaced by the nearest Dhananjays Magical alphabet.

2. If the character is equidistant with 2 Magical alphabets. The one with lower ASCII value will be considered as its replacement.

Source Code:

import java.math.BigInteger;

import java.util.\*;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class MagicNumber {

static long nextprime(long n) {

BigInteger b=new BigInteger(String.valueOf(n));

return Long.parseLong(b.nextProbablePrime().toString());

}

public static void main(String[] args) {

int n,c,i,j,t=0;

Scanner s=new Scanner(System.in);

System.out.println("Enter the number of test cases: ");

n=s.nextInt();

System.out.println("Enter number of characters: ");

c=s.nextInt();

String[] a=new String[n];

String b;

char[][] x=new char[n][c];

int[][] q=new int[n][c];

char [] m=new char[c\*n];

long [] e=new long[c\*n];

String[][] s1= new String[n][c];

System.out.println("Enter the name: ");

for(i=0;i<n;i++) {

b=s.next();

a[i]=b.toUpperCase();

t=b.length();

Pattern p = Pattern.compile("[^A-Za-z0-9]");

Matcher o = p.matcher(b);

boolean tr = o.find();

if(b.length()>c || tr==true) {

System.out.println("More than"+c );

i--;

}

if(tr==true) {

System.out.println("Special character added");

i--;

}

}

if(t==c) {

for(i=0;i<n;i++)

for(j=0;j<c;j++) {

x[i][j]=a[i].charAt(j);

}

for(i=0;i<n;i++)

for(j=0;j<c;j++) {

q[i][j]=(int)x[i][j];

}

long[] p=new long[c\*n];

int k=0;

for(i=0;i<n;i++)

for(j=0;j<c;j++) {

p[k]=q[i][j];

if(p[k]==89 || p[k]==90) {

e[k]=67;

}

else {

e[k]=nextprime(p[k]);

}

k++;

}

for(i=0;i<c\*n;i++) {

m[i]=(char)e[i];

}

int l=0;

for(j=0;j<n;j++)

for(i=0;i<c;i++) {

s1[j][i]=Character.toString(m[l]);

l++;

}

for(j=0;j<n;j++) {

for(i=0;i<c;i++) {

System.out.print(s1[j][i]);

}

System.out.println(" ");

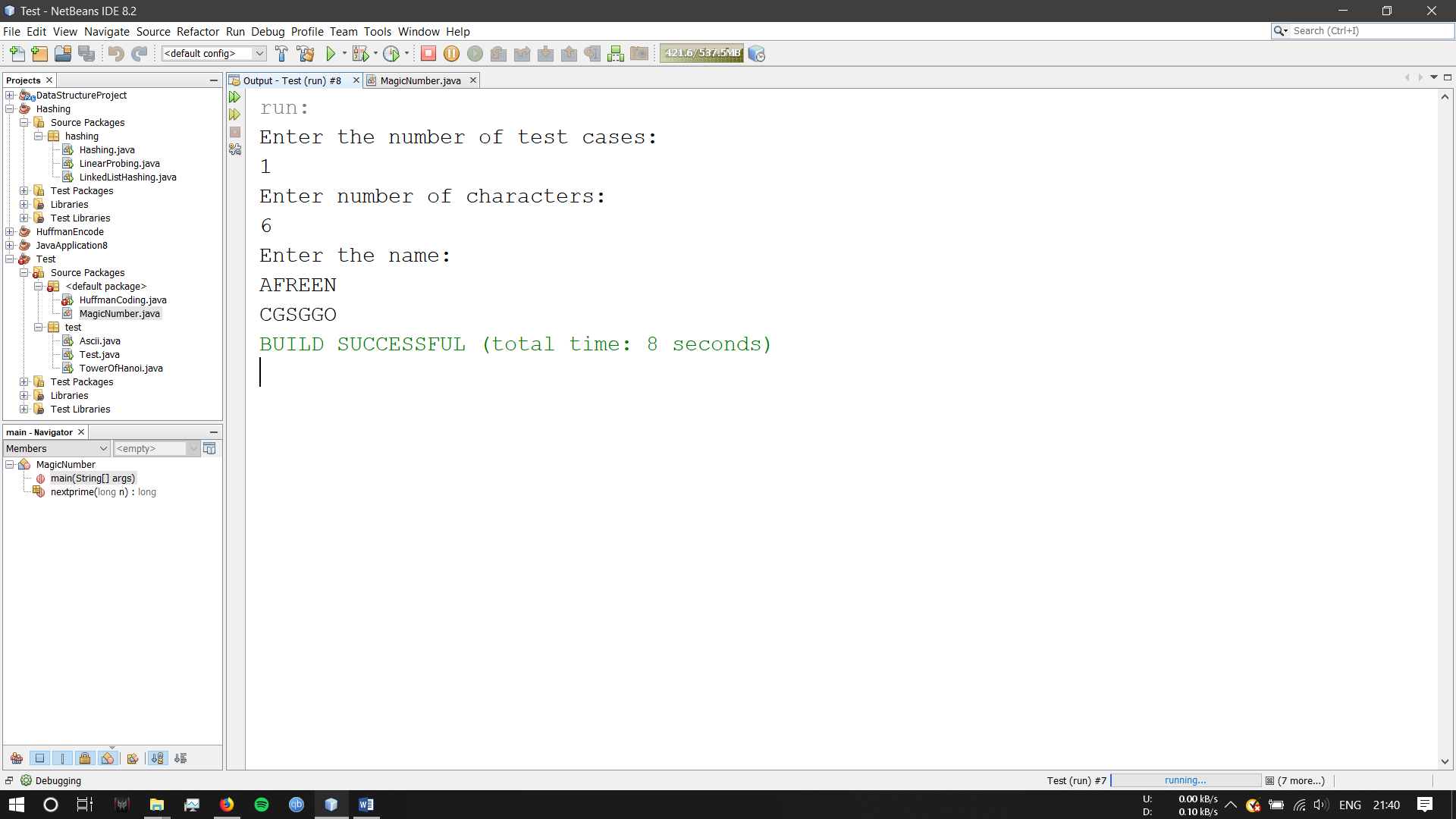
}

}

}

}

Screenshot of Output:



References:

<https://www.geeksforgeeks.org/biginteger-class-in-java/>

<https://www.geeksforgeeks.org/program-print-ascii-value-character/> **7) Lab Session 3, Problem 1.**

Date: 10-07-2019

Problem Statement:

Tower of Hanoi consists of three pegs or towers with n disks placed one over the other.

The objective of the puzzle is to move the stack to another peg following these simple rules.

1: Only one disk can be moved at a time.

2: No disk can be placed on top of the smaller disk.

Source code:

import java.util.Scanner;

/\*\*

\*

\* @author Akshay

\*/

public class TowerOfHanoi {

static void towerOfHanoi(int n, char from, char to, char aux)

{

if (n == 1)

{

System.out.println("Move disk 1 from " + from + " to " + to);

return;

}

towerOfHanoi(n-1, from, aux, to);

System.out.println("Move disk " + n + " from " + from + " to " + to);

towerOfHanoi(n-1, aux, to, from);

}

public static void main(String args[])

{

Scanner sc = new Scanner(System.in);

System.out.println("Enter number of Disks");

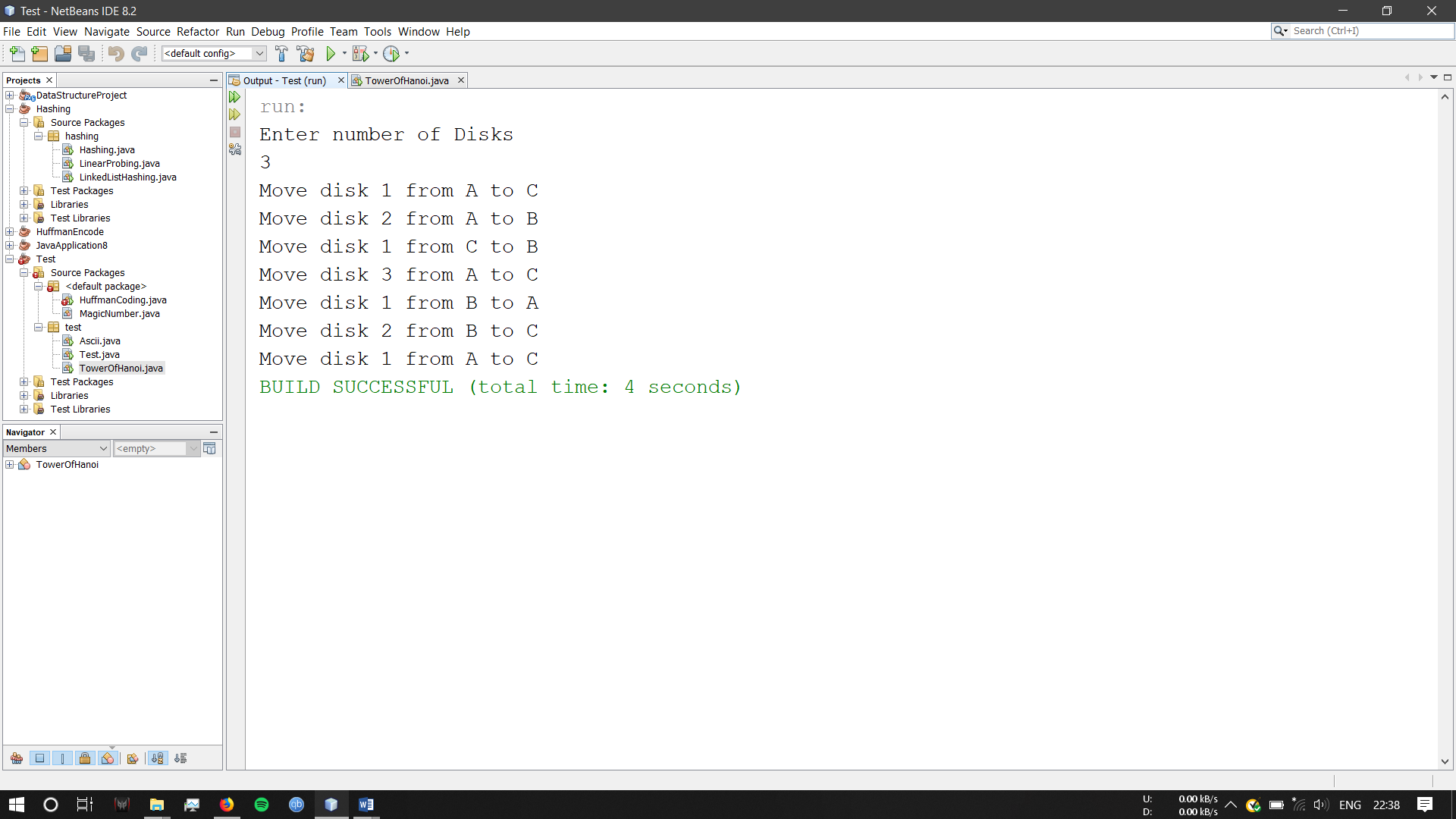
int n = sc.nextInt();

towerOfHanoi(n, 'A', 'C', 'B');

}

}

Screenshot of Output:



References:

<https://www.geeksforgeeks.org/biginteger-class-in-java/>

<https://www.geeksforgeeks.org/program-print-ascii-value-character/>

**8) Lab Session 3, Problem 2.**

**Date:** 10-07-2019

**Problem Statement:**

Simran is running up a staircase with N steps, and can hop(jump) either 1 step, 2 steps or 3

steps at a time.You have to count, how many possible ways Simran can run up to the stairs.

**Source Code:**

import java.util.Scanner;

public class Stairs {

static int count;

public static int count(int n) {

if(n<=1) {

return 1;

}

else {

if(n==2) {

return count(n-1)+count(n-2);

}

else {

return count(n-1)+count(n-2)+count(n-3);

}

}

}

public static void main(String[] args) {

int n,steps=0;

Stairs st=new Stairs();

System.out.println("Enter number of stairs:");

Scanner s=new Scanner(System.in);

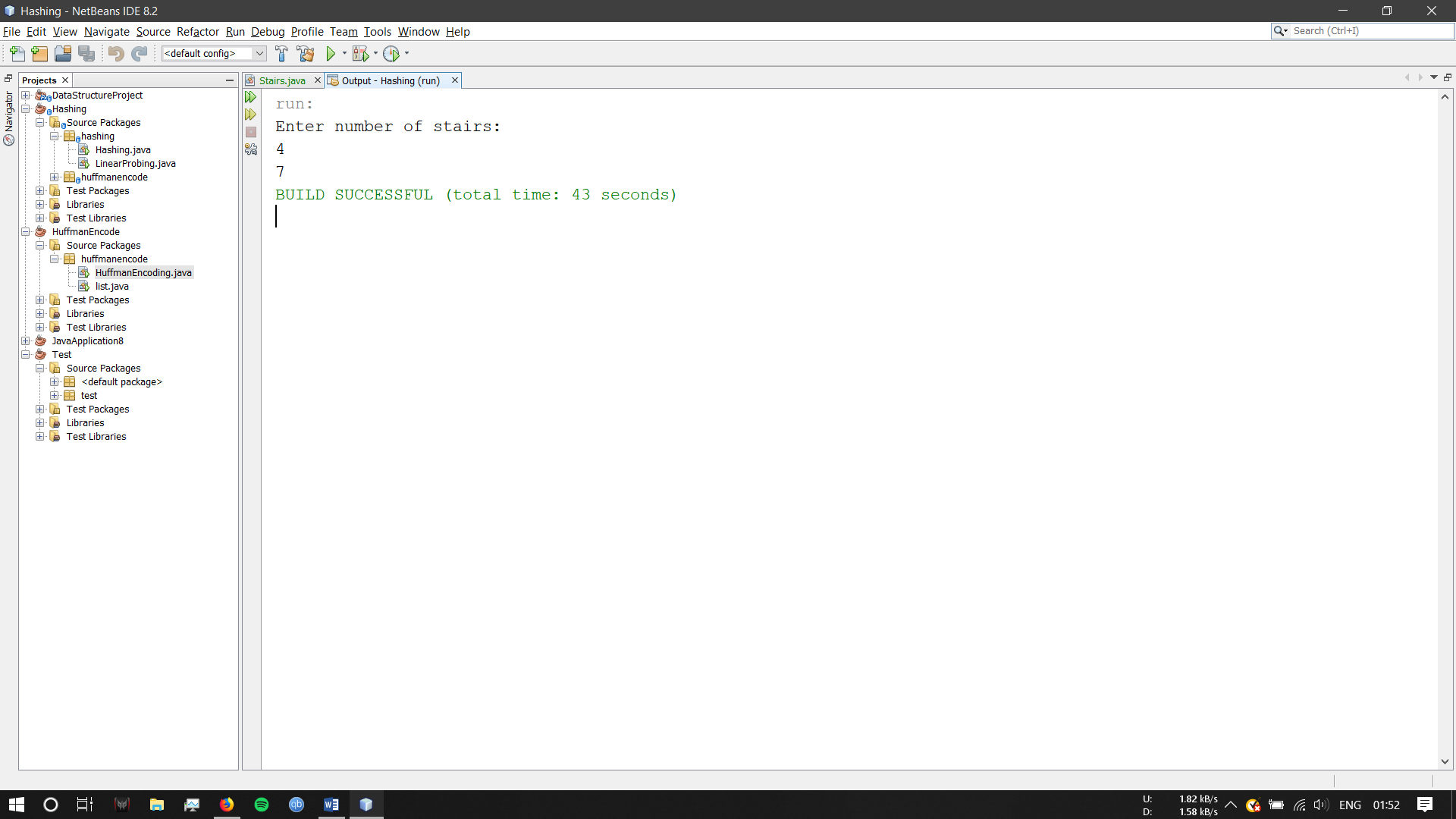
n=s.nextInt();

System.out.println(count(n));

}

}

Output Screenshot:



**9) Lab Session 4 - Problem 1.**

Date: 15-07-2019

Problem Statement:

You have an empty sequence, and you will be given N queries. Each query is one of these

three types:

1 x - Push the element x into the stack.

2 -Delete the element present at the top of the stack.

3 -Print the maximum element in the stack.

Source Code:

import java.util.Stack;

import java.util.\*;

public class MaxElementInStack {

/\*\*

\*

\* @author Akshay

\* @param args

\*/

public static void main(String[] args) {

Stack<Integer> main=new Stack<>();

Stack<Integer> track = new Stack<>();

Scanner sc = new Scanner(System.in);

System.out.println("Enter the number of queries you want to perform.");

int number = sc.nextInt();

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

System.out.println("1. To push an element 'x'.");

System.out.println("2. To delete the element at the top of the stack.");

System.out.println("3. To print the maximum element in the stack.");

System.out.println("");

System.out.println("Enter the Query you want to perform.");

int qu = sc.nextInt();

switch (qu) {

case 1:

System.out.println("Enter the element 'x' to be pushed.");

int a = sc.nextInt();

main.push(a);

if(track.isEmpty()) {

track.push(a);

} else {

if(a>track.peek()) {

track.push(a);

}

}

break;

case 2:

int b = main.pop();

if(b==track.peek()) {

track.pop();

}

System.out.println("The poped value is" +b);

break;

case 3:

System.out.println("The maximun element in the stack is : " +track.peek());

break;

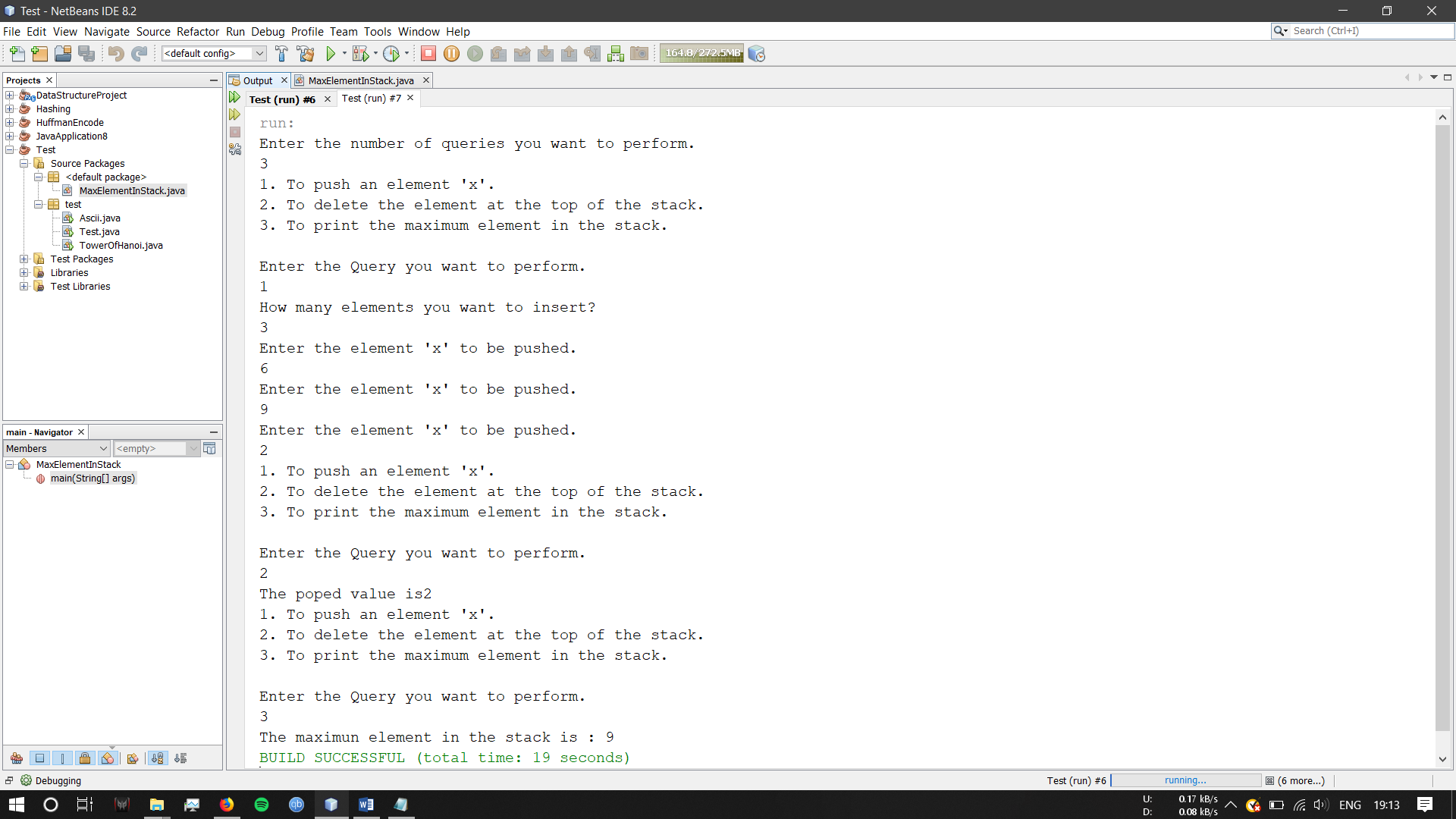
}

}

}

}

Screenshot of Output:



References:

<http://www.iitk.ac.in/esc101/08Jan/Solutions/lab8/thursday/Teststack.java>

<https://www.geeksforgeeks.org/tracking-current-maximum-element-in-a-stack/>

<https://algorithms.tutorialhorizon.com/track-the-maximum-element-in-a-stack/>

**10) Lab Session 4 - Problem 2.**

Date: 15-07-2019

Problem Statement:

A bracket is considered to be any one of the following characters: (, ), {, }, [, or ].

Two brackets are considered to be a matched pair if the an opening bracket (i.e., (, [, or {) occurs to the left of a closing bracket (i.e., ), ], or }) of the exact same type. There are three types of matched pairs of brackets: [], {},and ().

A matching pair of brackets is not balanced if the set of brackets it encloses are not matched. For example, {[(])} is not balanced because the contents in between { and } are not balanced. The pair of square brackets encloses a single, unbalanced opening bracket, (, and the pair of parentheses encloses a single, unbalanced closing square bracket,].

By this logic, we say a sequence of brackets is balanced if the following conditions are met:

It contains no unmatched brackets.

The subset of brackets enclosed within the confines of a matched pair of brackets is also a matched pair of brackets.

Given n strings of brackets, determine whether each sequence of brackets is balanced. If a string is balanced, return YES. Otherwise, return NO.

Source Code:

import java.util.Scanner;

import java.util.Stack;

public class BalancedParenthesisStack {

public static void main(String[] args) {

int i;

Stack<Character> stack = new Stack();

Scanner input = new Scanner(System.in);

System.out.println("Enter the number of brackets (n) : ");

int n = input.nextInt();

int m=n+1;

String a[]=new String[m];

System.out.println("Enter the brackets");

for(i=0; i<m;i++) {

a[i] = input.nextLine();

}

char ch;

for(i=0; i<m;i++) {

String g=a[i];

for(int j=0;j<g.length();j++) {

ch=g.charAt(j);

System.out.println(""+ch);

if(ch=='{' || ch=='(' || ch=='[') {

stack.push(ch);

} else {

if(ch==']' && stack.peek()=='[' || ch=='}' && stack.peek()=='{' || ch==')' && stack.peek()=='(') {

stack.pop();

if(stack.isEmpty()) {

System.out.println("YES");

}

} else {

System.out.println("NO");

}

}

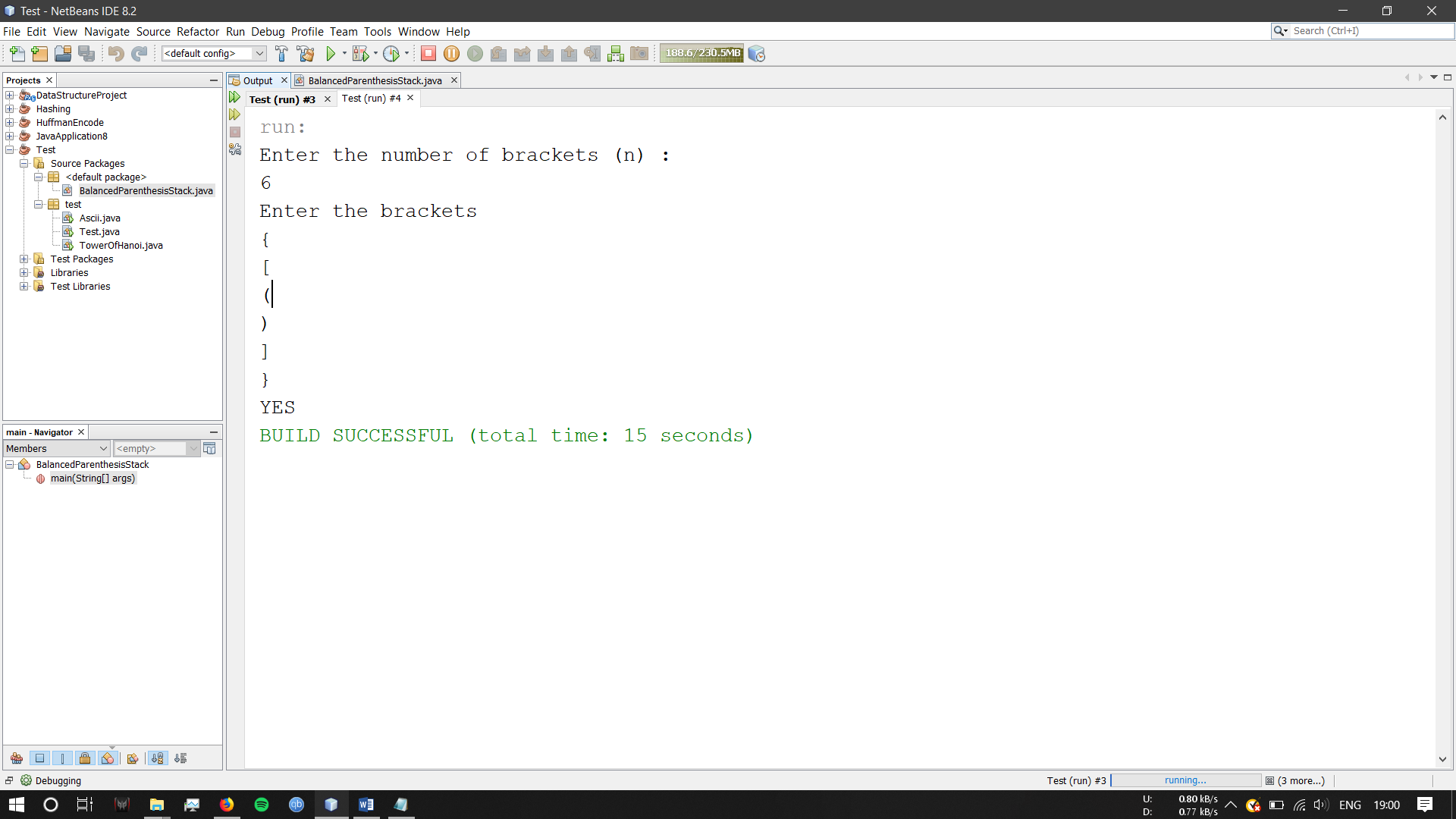
}

}

}

}

Screenshot of Output:



References:

<https://www.geeksforgeeks.org/check-for-balanced-parentheses-in-an-expression/>

<https://www.javatpoint.com/java-string-charat>

**11) Lab Session 4 - Problem 3.**

Date: 17-07-2019

Problem Statement:

You have three stacks of books where each book has the same length, but they may vary in height. You can change the height of a stack by removing and discarding its topmost book any number of times.

Find the maximum possible height of the stacks such that all of the stacks are exactly the same height. This means you must remove zero or more books from the top of zero or more of the three stacks until they&#39;re all the same height, then print the height. The removals must be performed in such a way as to maximize the height.

Source Code:

import java.util.Scanner;

import java.util.Stack;

/\*\*

\*

\* @author Akshay

\*/

public class BookStack {

public static void main(String[] args){

int firststk, secondstk, thirdstk;

int i, j, k;

Stack<Integer> stack1 = new Stack();

Stack<Integer> stack2 = new Stack();

Stack<Integer> stack3 = new Stack();

Scanner sc = new Scanner(System.in);

System.out.println("How many elements you want top enter in the first Stack? :");

firststk=sc.nextInt();

System.out.println("Enter "+firststk+" elements in first stack!");

for(i=0;i<firststk;i++){

int x = sc.nextInt();

stack1.push(x);

}

System.out.println(""+stack1+"");

System.out.println("How many elements you want top enter in the second Stack? :");

secondstk=sc.nextInt();

System.out.println("Enter "+secondstk+" elements in second stack!");

for(i=0;i<secondstk;i++){

int y = sc.nextInt();

stack2.push(y);

}

System.out.println(""+stack2+"");

System.out.println("How many elements you want top enter in the third Stack? :");

thirdstk=sc.nextInt();

System.out.println("Enter "+thirdstk+" elements in third stack!");

for(i=0;i<thirdstk;i++){

int z = sc.nextInt();

stack3.push(z);

}

System.out.println(""+stack3+"");

int sum1 =0;

for(i=0;i<stack1.size();i++){

sum1 = sum1 + stack1.pop();

}

System.out.println(""+sum1+"");

int sum2 =0;

for(i=0;i<stack2.size();i++){

sum2 = sum2 + stack2.pop();

}

System.out.println(""+sum2+"");

int sum3 =0;

for(i=0;i<stack3.size();i++){

sum3 = sum3 + stack3.pop();

}

System.out.println(""+sum3+"");

if (sum1==sum2 & sum2==sum3){

System.out.println("The Max Sum is : "+sum1);

}

if(sum1>sum2 && sum1>sum3) {

sum1=sum1-stack1.pop();

}

else if(sum2>sum1 && sum2>sum3) {

sum2=sum2-stack2.pop();

}

else if(sum3>sum1 && sum3>sum2) {

sum3=sum3-stack3.pop();

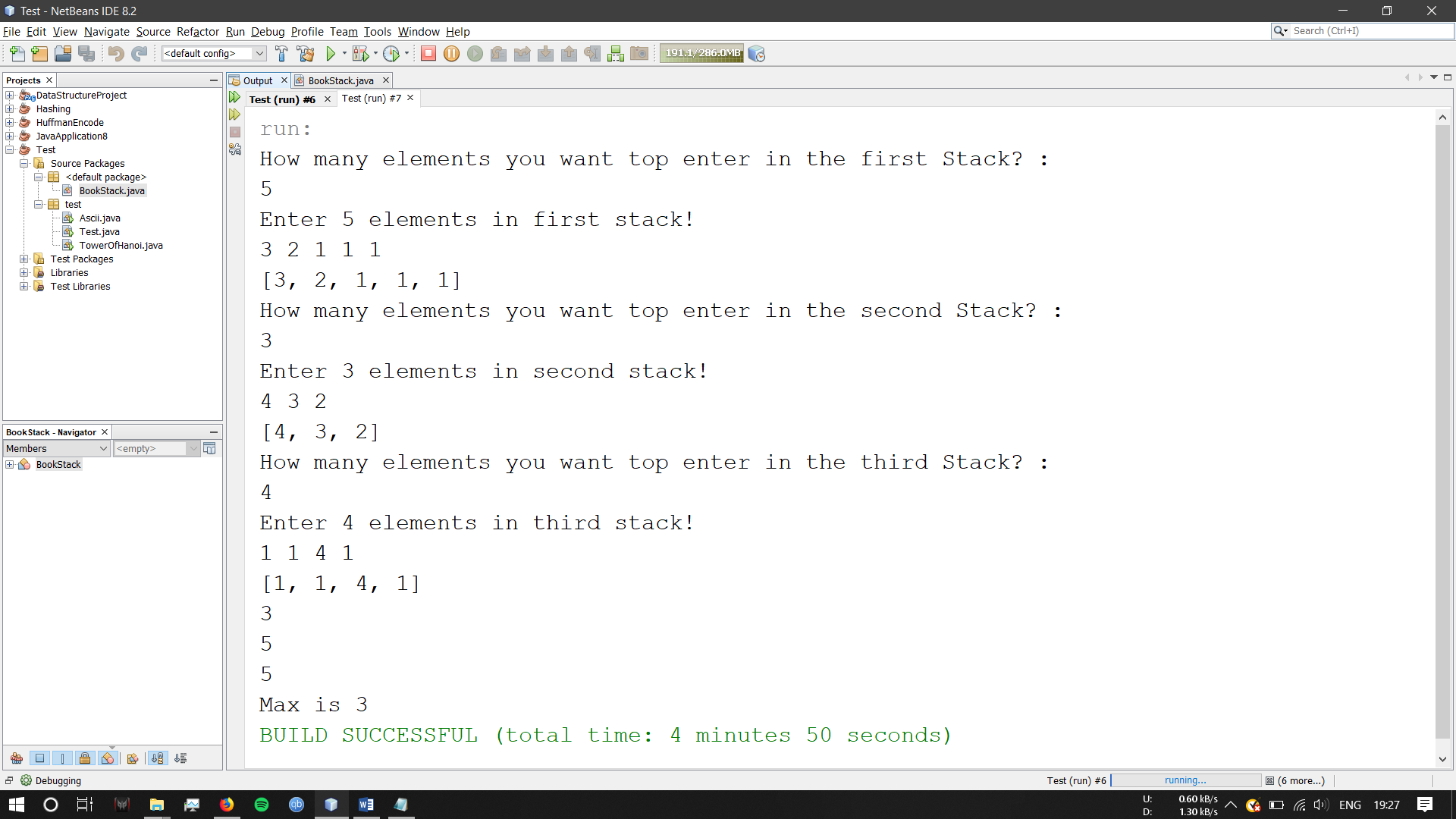
}

System.out.println("Max is "+sum1);

}

}

Screenshot of Output:



References:

<https://www.geeksforgeeks.org/find-maximum-sum-possible-equal-sum-three-stacks/>

**12) Lab Session 5 - Problem 1.**

Date: 24-07-2019

Problem Statement:

A linked list is said to contain a cycle if any node is visited more than once while traversing the list. Your function must return a boolean denoting whether or not there is a cycle in the list. If there is a cycle, return true; otherwise, return false.

Note: If the list is empty, head will be null.

Source Code:

package labsession5;

import java.util.Scanner;

class Node {

int data;

Node nextLink;

Node(int d)

{

data = d;

nextLink = null;

}

}

public class LinkedListBreakLoop {

static Node head = null;

static Node lastLink = null;

static void addNode(int data){

Node newNode = new Node(data);

if(head==null){

head = newNode;

lastLink = newNode;

}

else{

lastLink.nextLink = newNode;

//lastLink will become the newNode

lastLink = newNode;

}

}

public void display(){

Node currentptr = head;

if(head == null) {

System.out.println("Linked List is Empty");

return;

}

while(currentptr != null) {

//Prints each node

System.out.print(currentptr.data + " ");

currentptr = currentptr.nextLink;

}

System.out.println();

}

public void loop(){

Node lastLink=head;

while(lastLink.nextLink != null) {

lastLink=lastLink.nextLink;

}

lastLink.nextLink=head.nextLink;

}

public void detect() {

Node tort = head;

Node hare = head;

while(hare.nextLink.nextLink != null) {

tort=tort.nextLink;

hare=hare.nextLink.nextLink;

if(tort==hare){

System.out.println("Loop Detected in the LinkedList.");

break;

}

}

}

public void breakloop(){

Node tort = head;

Node hare = head;

while(hare.nextLink.nextLink != null) {

tort=tort.nextLink;

hare=hare.nextLink.nextLink;

if(tort==hare){

System.out.println("Loop Detected in the LinkedList.");

break;

}

}

int c=1;

hare = head.nextLink;

while(hare!=tort){

hare=hare.nextLink;

c++;

}

hare=head.nextLink;

while(hare.nextLink!=tort){

hare=hare.nextLink;

}

hare.nextLink=null;

System.out.println("Loop is broken.");

}

public static void main(String[] args){

LinkedListBreakLoop linklist = new LinkedListBreakLoop();

int n, i;

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number of nodes you want: ");

n=sc.nextInt();

System.out.println("Enter "+n+" nodes");

for(i=0;i<n;i++){

int da=sc.nextInt();

addNode(da);

}

System.out.print(" The LinkList is ");

linklist.display();

//System.out.print(" The Looped LinkList is ");

linklist.loop();

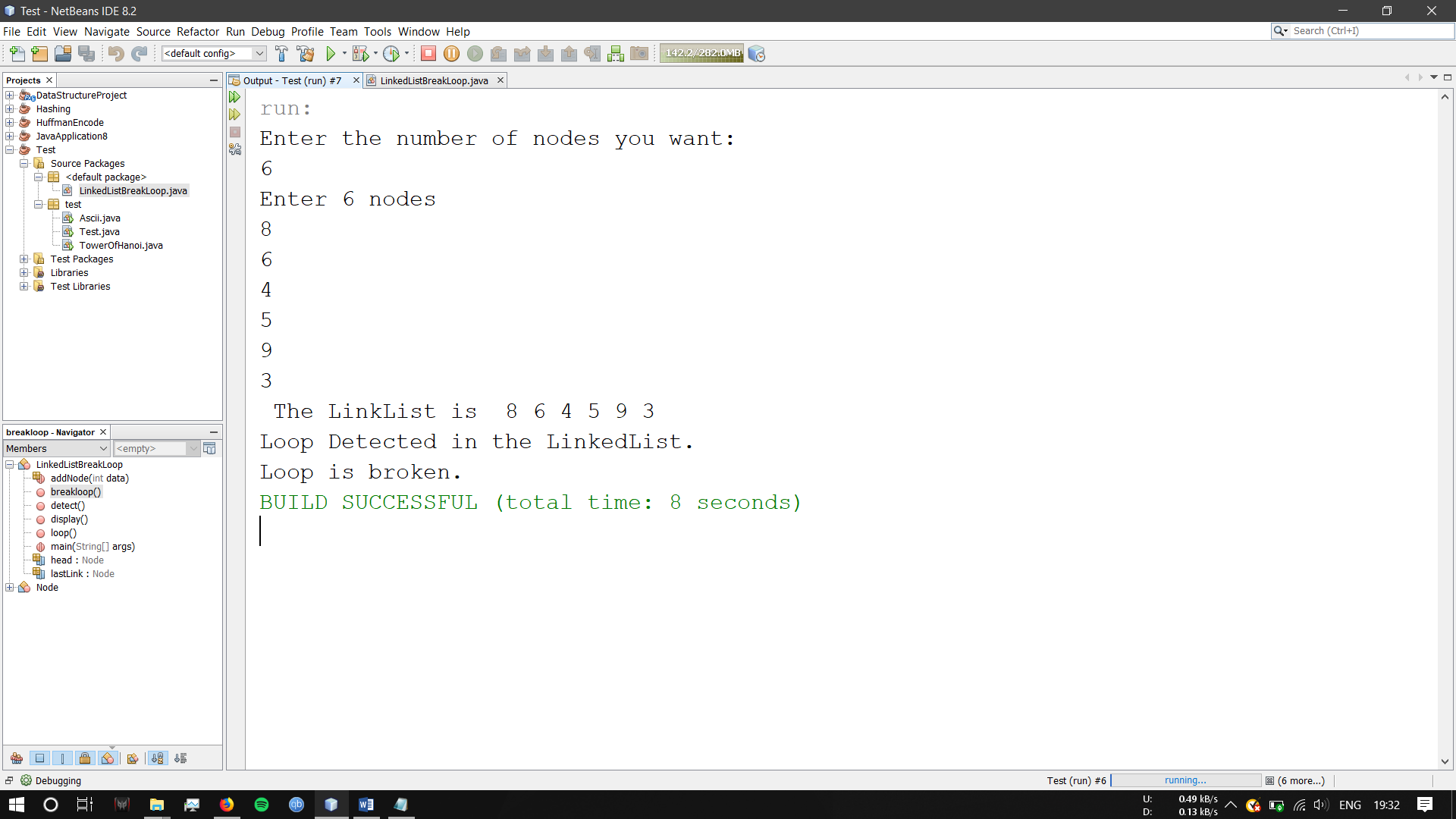
linklist.detect();

linklist.breakloop();

}

}

Screenshot of Output:



References:

<https://www.geeksforgeeks.org/detect-and-remove-loop-in-a-linked-list/>

<https://www.geeksforgeeks.org/make-loop-k-th-position-linked-list/>

<https://www.javatpoint.com/java-program-to-create-and-display-a-singly-linked-list>

<https://www.geeksforgeeks.org/implementing-a-linked-list-in-java-using-class/>

**13) Lab Session 5 - Problem 2.**

Date: 29-07-2019

Problem Statement:

Given a linked list, rotate the list to the right by k places, where k is non-negative.

Source Code:

import java.util.Scanner;

class Node {

int data;

Node next;

Node(int d)

{

data = d;

next = null;

}

}

public class LinkListRotate {

static Node head = null;

static Node lastLink = null;

static void addNode(int data){

Node newNode = new Node(data);

if(head==null){

head = newNode;

lastLink = newNode;

}

else{

lastLink.next = newNode;

//lastLink will become the newNode

lastLink = newNode;

}

}

public void display(){

Node currentptr = head;

if(head == null) {

System.out.println("Linked List is Empty");

return;

}

while(currentptr != null) {

//Prints each node

System.out.print(currentptr.data + " ");

currentptr = currentptr.next;

}

System.out.println();

}

public void rotate(int k){

if(k==0) return;

Node cpt=head;

int c=1, l;

l=k%n;

//Debug.println(n + "", null);

while( c < l && cpt!=null){

cpt=cpt.next;

c++;

}

if(cpt==null) return;

Node kn = cpt;

while(cpt.next!=null)

cpt=cpt.next;

cpt.next=head;

head=kn.next;

kn.next=null;

}

public static int n;

public static void main(String[] args){

LinkListRotate linklist = new LinkListRotate();

int i, k;

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number of nodes you want: ");

n=sc.nextInt();

System.out.println("Enter "+n+" nodes");

for(i=0;i<n;i++){

int da=sc.nextInt();

addNode(da);

}

System.out.print("The LinkList is ");

linklist.display();

System.out.println("By how many places you want to rotate the LINKLIST?");

k=sc.nextInt();

linklist.rotate(k);

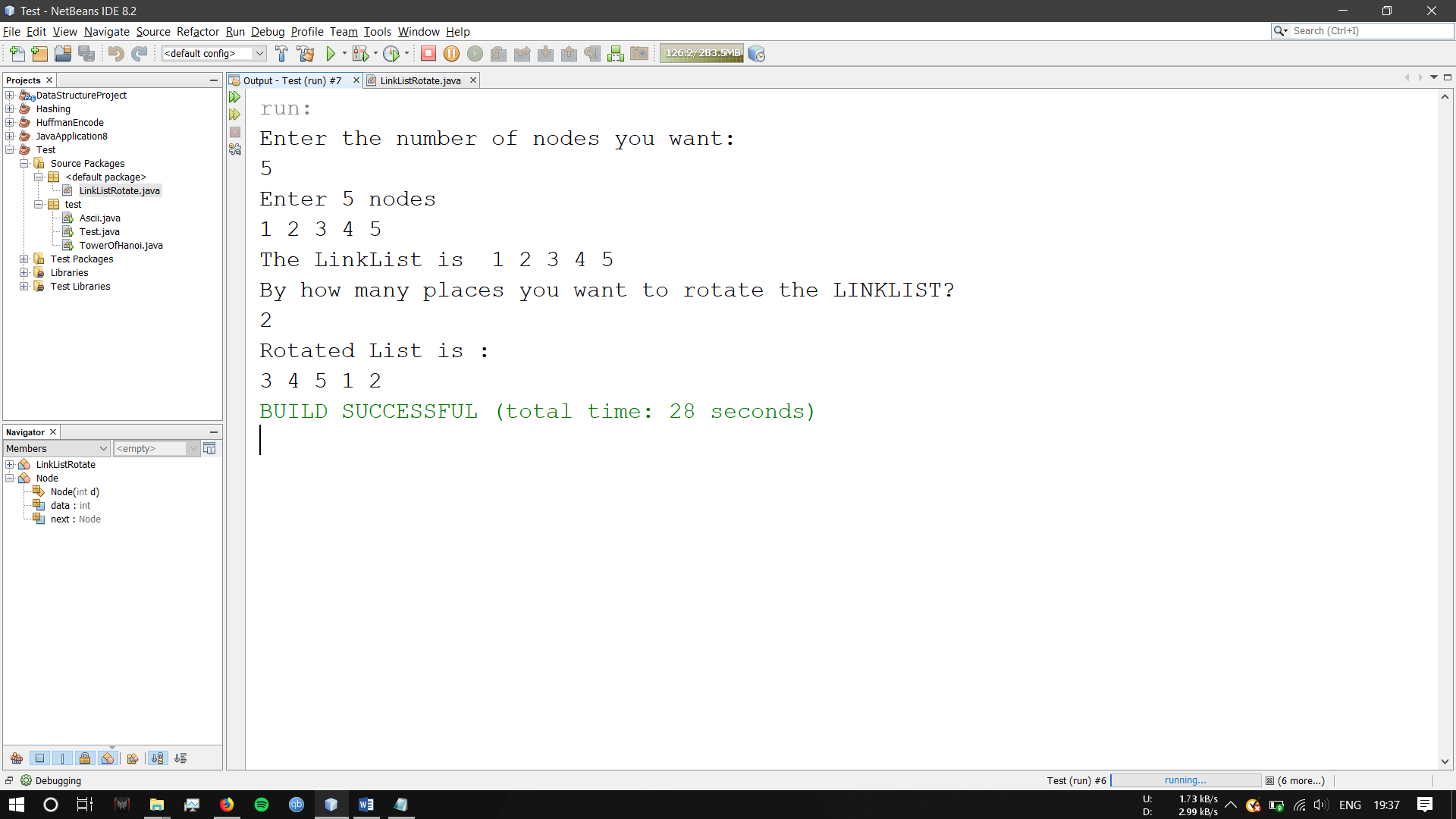
System.out.println("Rotated List is : ");

linklist.display();

}

}

Screenshot of Output:



References:

<https://www.youtube.com/watch?v=rllxESAWeP8&feature=youtu.be>

<https://www.geeksforgeeks.org/rotate-a-linked-list/>

**14) Lab Session 6 - Problem 1.**

Date: 31-07-2019

Problem Statement:

You are given a pointer to the root of a binary search tree and values to be inserted into the tree. Insert the values into their appropriate position in the binary search tree. The program should be able to insert, delete nodes and display all the elements of the tree.

Source Code:

package labsession5;

import java.util.Scanner;

public class BinarySearchTree {

class Node {

int key;

Node left, right;

public Node(int d){

key = d;

left = right = null;

}

}

Node root;

BinarySearchTree(){

root = null;

}

void insert(int key) {

root = insertRec(root, key);

}

Node insertRec(Node root, int key) {

if (root == null) {

root = new Node(key);

return root;

}

if (key < root.key) {

root.left = insertRec(root.left, key);

}

else if (key > root.key) {

root.right = insertRec(root.right, key);

}

return root;

}

void deleteKey(int key)

{

root = del(root, key);

}

Node del(Node root, int key)

{

if (root == null) return root;

if (key < root.key)

root.left = del(root.left, key);

else if (key > root.key)

root.right = del(root.right, key);

else

{

if (root.left == null)

return root.right;

else if (root.right == null)

return root.left;

root.key = minV(root.right);

root.right = del(root.right, root.key);

}

return root;

}

int minV(Node root)

{

int minv = root.key;

while (root.left != null)

{

minv = root.left.key;

root = root.left;

}

return minv;

}

void inorder() {

inorderTraversal(root);

}

void inorderTraversal(Node root){

if(root!=null){

inorderTraversal(root.left);

System.out.println(root.key);

inorderTraversal(root.right);

}

}

public static void main(String[] args){

BinarySearchTree tree = new BinarySearchTree();

Scanner sc = new Scanner(System.in);

int n, dat, e, j;

System.out.println("How many nodes do you want to enter?");

n=sc.nextInt();

System.out.println("Enter the values : ");

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

dat=sc.nextInt();

tree.insert(dat);

}

System.out.println("The INORDER traversal of the tree is : ");

tree.inorder();

System.out.println("How many elements you want to delete?");

e=sc.nextInt();

for(j=0;j<e;j++){

System.out.println("Delete a node");

int kj = sc.nextInt();

tree.deleteKey(kj);

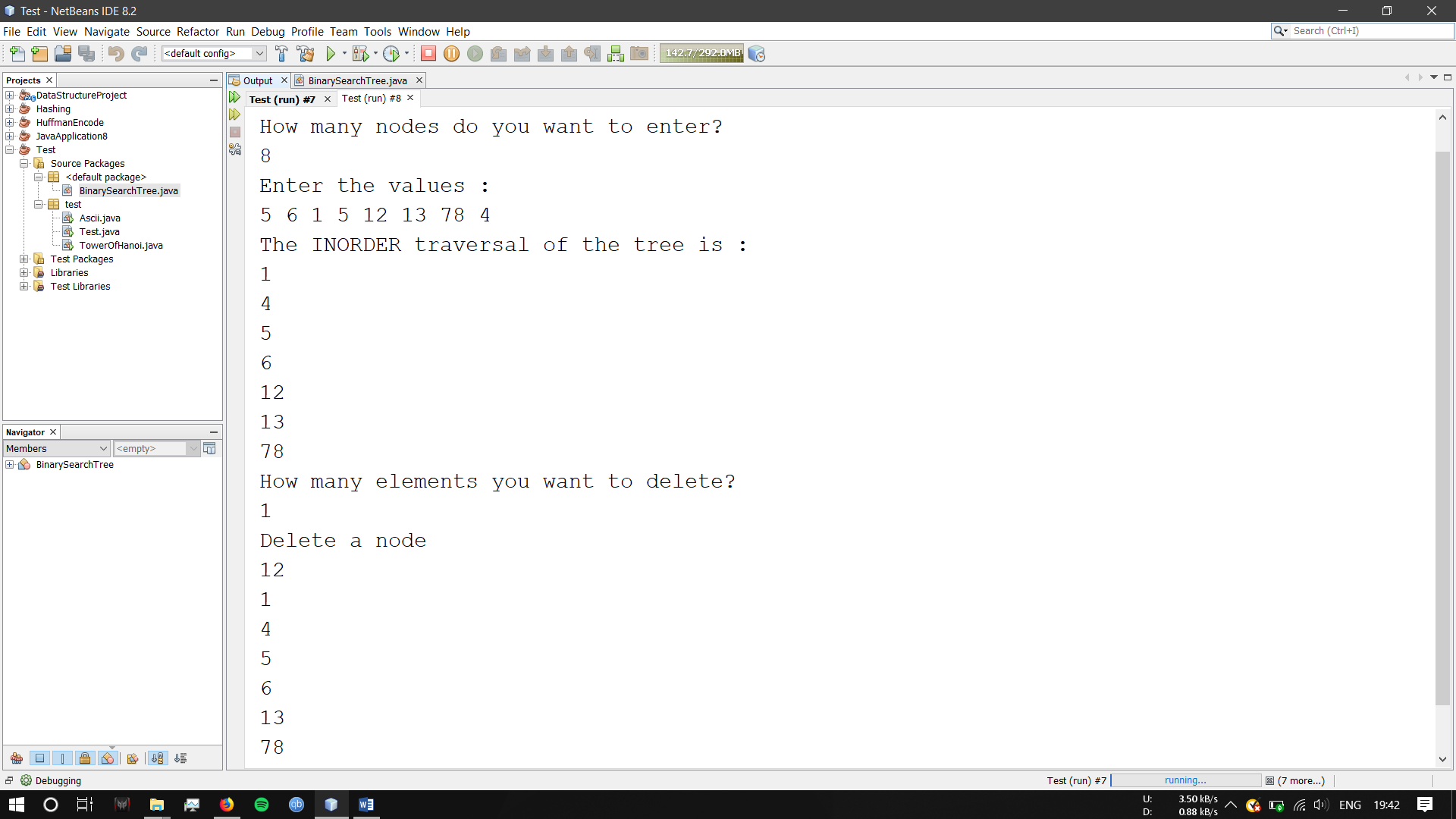
tree.inorder();

}

}

}

Screenshot of Output:



References:

<https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/>

<https://www.geeksforgeeks.org/binary-search-tree-set-2-delete/>

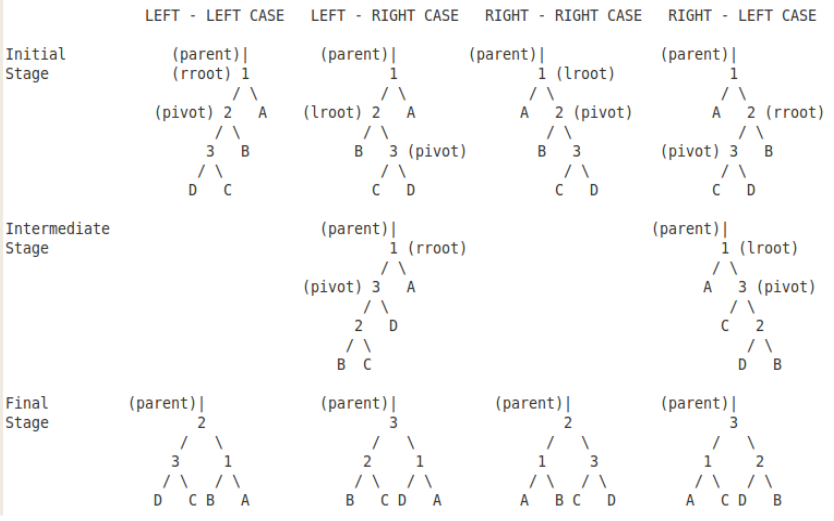
**15) Lab Session 7 - Problem 1.**

Date: 11-09-2019

Problem Statement:

An AVL tree is a binary search tree in which the balance factor of every node, which is defined as the difference between the heights of the node’s left &amp; right sub trees.

Construction of AVL tree is same as Binary search tree, but the difference is after inserting the new node into the tree or deleting a node from the tree, to check each node for the balance. If the tree is not balanced, we need to restructure or rebuild the tree for balancing using rotation.



Source Code:

import java.util.Scanner;

class Node

{

int key, height;

Node left, right;

Node(int item)

{

key = item;

left = right = null;

height = 1;

}

}

class AVLTree

{

Node root;

int height(Node node) {

if (node == null)

return 0;

return node.height;

}

Node rightRotateAVLTree(Node y){

Node x = y.left;

Node b = x.right;

x.right = y;

y.left = b;

int rhy = Math.max(height(y.left), height(y.right));

int rhx = Math.max(height(x.left), height(x.right));

y.height = rhy + 1;

x.height = rhx + 1;

return x;

}

Node leftRotateAVLTree(Node x){

Node y = x.right;

Node b = y.left;

y.left = x;

x.right = b;

int lhy = Math.max(height(y.left), height(y.right));

int lhx = Math.max(height(x.left), height(x.right));

y.height = lhy + 1;

x.height = lhx + 1;

return y;

}

int getHeight(Node node)

{

if (node == null)

return 0;

else

{

int left = getHeight(node.left);

int right = getHeight(node.right);

int height = Math.max(left,right)+1;

return height;

}

}

int getBalanceFactor(Node node){

if (node == null)

return 0;

int balanceFactor = height(node.left) - height(node.right);

return balanceFactor;

}

void insert(int key) {

root = insertRec(root, key);

}

Node insertRec(Node node, int key) {

if (node == null)

return (new Node(key));

if (key < node.key)

node.left = insertRec(node.left, key);

else if (key > node.key)

node.right = insertRec(node.right, key);

else

return node;

node.height = 1 + Math.max(height(node.left),

height(node.right));

int balance = getBalanceFactor(node);

if (balance > 1 && balance < -1 && key < node.left.key) {

return rightRotateAVLTree(node);

}

// Right Right Case

if (balance < -1 && balance > 1 && key > node.right.key) {

return leftRotateAVLTree(node);

}

// Left Right Case

if (balance > 1 && balance < -1 && key > node.left.key) {

node.left = leftRotateAVLTree(node.left);

return rightRotateAVLTree(node);

}

// Right Left Case

if (balance < -1 && balance > 1 && key < node.right.key) {

node.right = rightRotateAVLTree(node.right);

return leftRotateAVLTree(node);

}

return node;

}

void inorder() {

inorderTraversal(root);

}

void inorderTraversal(Node root){

if(root!=null){

inorderTraversal(root.left);

System.out.println(root.key);

inorderTraversal(root.right);

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

AVLTree tree = new AVLTree();

int n, data;

int number = 100;

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

System.out.println("||||| AVLTree Operations |||||");

System.out.println(" 1. Insert Data.");

System.out.println(" 2. Delete Data.");

System.out.println(" 3. Tree Height.");

System.out.println(" 4. Balance Factor.");

System.out.println(" 5. Print Inorder.");

System.out.println(" 6. EXIT");

int choice = sc.nextInt();

switch (choice) {

case 1:

System.out.println("How many elements do you want to enter?");

n=sc.nextInt();

System.out.println("Enter the values : ");

for(int j=0;j<n;j++){

data=sc.nextInt();

tree.insert(data);

}

break;

case 2:

case 3:

System.out.println("Height of tree is : " + tree.getHeight(tree.root));

System.out.println("");

break;

case 4:

System.out.println("Balance factor of root is : " + tree.getBalanceFactor(tree.root));

System.out.println("");

break;

case 5:

tree.inorder();

break;

case 6:

return;

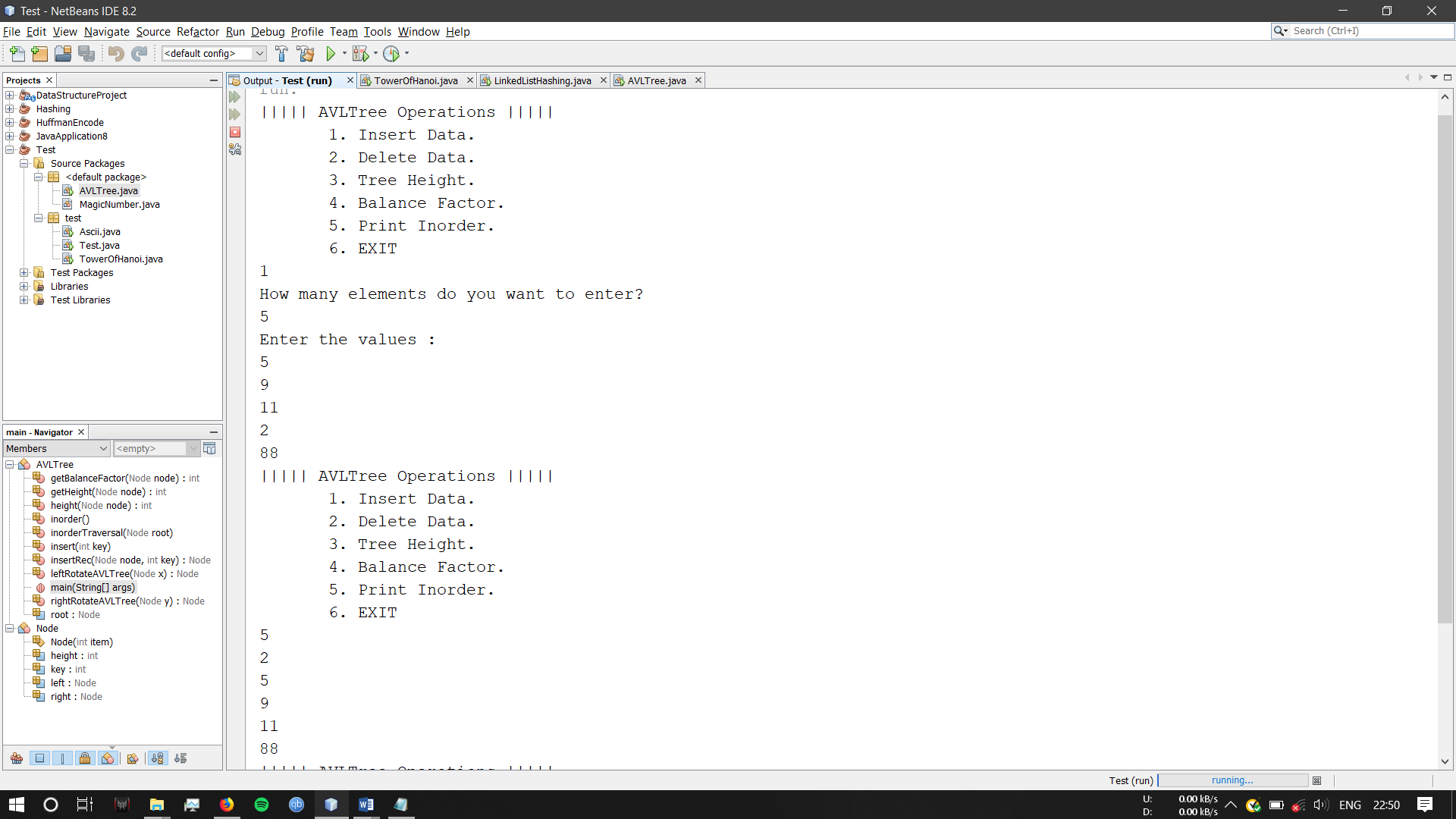
}

}

}

}

Screenshot of Output:



References:

https://www.geeksforgeeks.org › avl-tree-set-1-insertion/

**16) Lab Session 8 - Problem 1.**

Date: 25-09-2019

Problem Statement:

Huffman encoding is a way to assign binary codes to symbols that reduces the overall number of bits used to encode a typical string of those symbols.

For example, if you use letters as symbols and have details of the frequency of occurrence of those letters in typical strings, then you could just encode each letter with a fixed number of bits, such as in ASCII codes. You can do better than this by encoding more frequently occurring letters such as e and a, with smaller bit strings; and less frequently occurring letters such as q and x with longer bit strings.

Any string of letters will be encoded as a string of bits that are no-longer of the same length per letter. The Huffman coding scheme takes each symbol and its weight (or frequency of occurrence), and generates proper encodings for each symbol taking account of the weights of each symbol, so that higher weighted symbols have fewer bits in their encoding.

A Huffman encoding can be computed by first creating a tree of nodes.

Huffman coding:

1. Create a leaf node for each symbol and add it to the priority queue.
2. While there is more than one node in the queue:
3. Remove the node of highest priority (lowest probability) twice to get two nodes.
4. Create a new internal node with these two nodes as children and with probability equal to the sum of the two nodes probabilities.
5. Add the new node to the queue.
6. The remaining node is the root node and the tree is complete.

Source Code:

import java.util.Collection;

import java.util.PriorityQueue;

import java.util.Scanner;

import java.util.Comparator;

import java.util.HashMap;

/\*\*

\*

\* @author Akshay

\*/

class HuffmanNode {

int data;

char c;

HuffmanNode left;

HuffmanNode right;

}

class HuffmanComparator implements Comparator<HuffmanNode> {

public int compare(HuffmanNode l, HuffmanNode r)

{

return l.data - r.data;

}

}

public class HuffmanCoding {

public static void printTreeTraversal(HuffmanNode root, String s)

{

if (root.left == null && root.right == null && Character.isLetter(root.c)) {

System.out.println(root.c + ":" + s);

return;

}

printTreeTraversal(root.left, s + "0");

printTreeTraversal(root.right, s + "1");

}

public static void main(String[] args)

{

char t;

Scanner sc = new Scanner (System.in);

String str;

System.out.println("Enter the String");

str=sc.nextLine();

String toLowerCase = str.toLowerCase();

HashMap<Character, Integer> map = new HashMap<>();

int p;

for(p=0;p<toLowerCase.length();p++){

t=toLowerCase.charAt(p);

if(map.containsKey(t)){

int val = map.get(t);

val = val + 1;

map.put(t,val);

}

else{

map.put(t, 1);

}

}

System.out.println(map);

Collection<Integer> values=map.values();

Character [] keyArray=map.keySet().toArray(new Character[map.size()]);

Integer [] freqArray=values.toArray(new Integer[map.size()]);

int n = keyArray.length;

PriorityQueue<HuffmanNode> pQueue = new PriorityQueue<HuffmanNode>(n, new HuffmanComparator());

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

HuffmanNode hn = new HuffmanNode();

hn.c = keyArray[i];

hn.data = freqArray[i];

hn.left = null;

hn.right = null;

pQueue.add(hn);

}

HuffmanNode root = null;

while (pQueue.size() > 1) {

HuffmanNode x = pQueue.peek();

pQueue.poll();

HuffmanNode y = pQueue.peek();

pQueue.poll();

HuffmanNode hNode = new HuffmanNode();

hNode.data = x.data + y.data;

hNode.c = '-';

hNode.left = x;

hNode.right = y;

root = hNode;

pQueue.add(hNode);

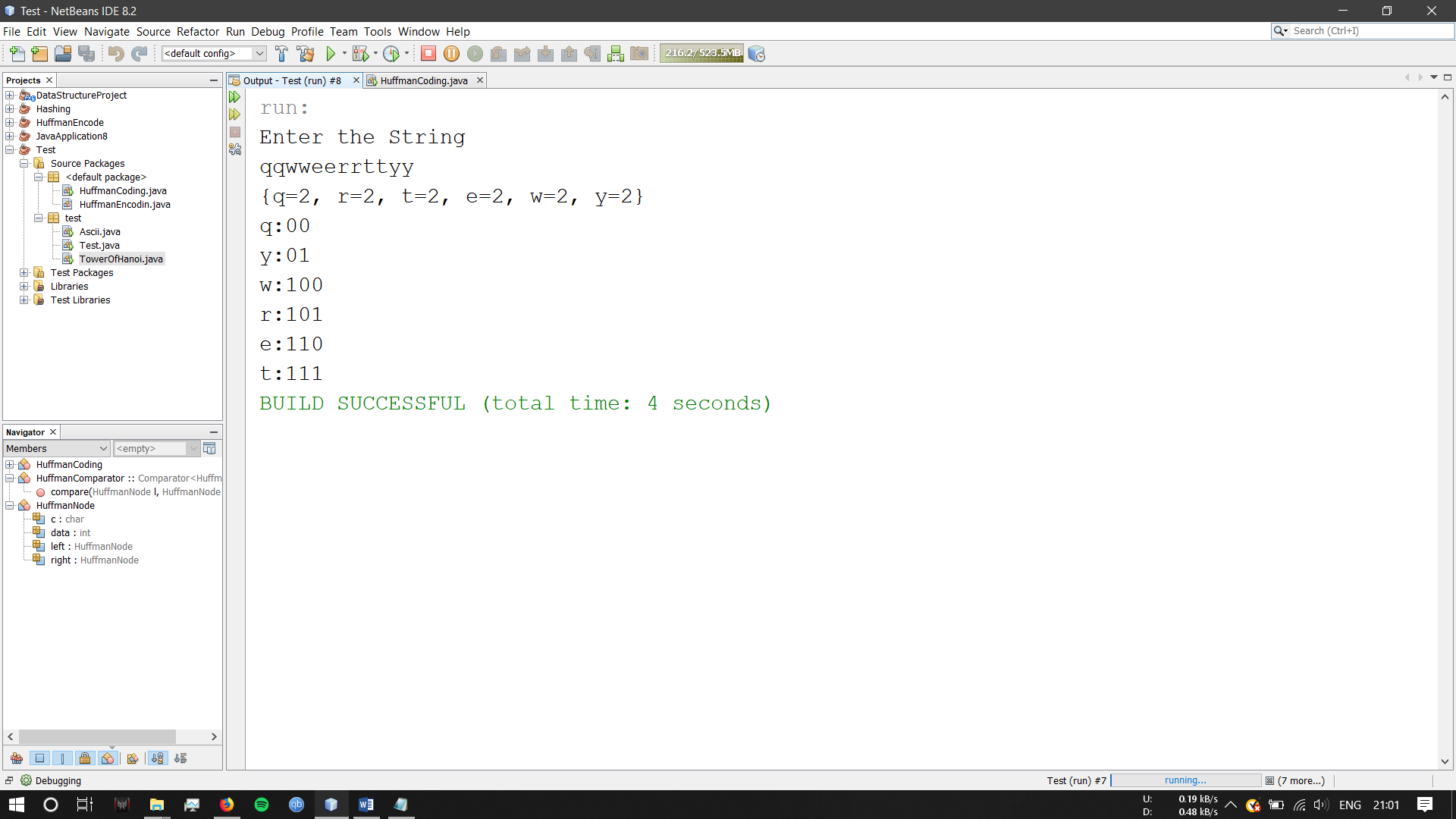
}

printTreeTraversal(root, "");

}

}

Screenshot of Output:



References:

<https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/>

<https://www.tutorialspoint.com/java/character_isletter.htm>

<https://www.journaldev.com/23246/huffman-coding-algorithm>

**17) Lab Session 9 - Problem 1.**

Date: 25-09-2019

Problem Statement:

Source Code:

package hashing;

import java.util.Scanner;

/\*\*

\*

\* @author Akshay

\*/

public class LinearProbing {

public static int length;

public static int compression(int d){

return d%length;

}

public static void main(String []args){

int i, n;

int index;

Scanner sc = new Scanner(System.in);

System.out.println("Number of elements:");

length=sc.nextInt();

int num[] = new int[length];

System.out.println("Enter elements");

for(i = 0; i < length; i++)

{

int no = sc.nextInt();

index = compression(no);

{

boolean isRunning = true;

while(isRunning){

if(index >= length)

index = 0;

if(num[index] == 0)

{

num[index] = no;

isRunning = false;

}

index++;

}

}

}

int j;

for(j=0;j<length;j++){

System.out.println(num[j]);

}

}

}

Screenshot of Output:

