**Assignment Stack**

Q1. Given a rows x cols binary matrix filled with 0's and 1's, find the largest rectangle containing

only 1's and return its area.

matrix =

[["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]

6

Explanation: The maximal rectangle is shown in the above picture.

Example 2:

matrix = [["0"]]

0

Example 3:

matrix = [["1"]]

1

import java.util.Stack;

public class MaxRectangle {

// Method to compute the largest rectangle area in a histogram

private static int maxRectangleInHistogram(int[] heights) {

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

int maxArea = 0;

int index = 0;

while (index < heights.length) {

if (stack.isEmpty() || heights[index] >= heights[stack.peek()]) {

stack.push(index++);

} else {

int topOfStack = stack.pop();

int area = heights[topOfStack] \* (stack.isEmpty() ? index : index - stack.peek() - 1);

maxArea = Math.max(maxArea, area);

}

}

while (!stack.isEmpty()) {

int topOfStack = stack.pop();

int area = heights[topOfStack] \* (stack.isEmpty() ? index : index - stack.peek() - 1);

maxArea = Math.max(maxArea, area);

}

return maxArea;

}

// Method to find the largest rectangle containing only 1s in a binary matrix

public static int maximalRectangle(char[][] matrix) {

if (matrix.length == 0) return 0;

int rows = matrix.length;

int cols = matrix[0].length;

int[] heights = new int[cols];

int maxArea = 0;

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

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

// Update the heights array

heights[j] = matrix[i][j] == '1' ? heights[j] + 1 : 0;

}

// Calculate the maximum area for the current histogram

maxArea = Math.max(maxArea, maxRectangleInHistogram(heights));

}

return maxArea;

}

public static void main(String[] args) {

char[][] matrix1 = {

{'1', '0', '1', '0', '0'},

{'1', '0', '1', '1', '1'},

{'1', '1', '1', '1', '1'},

{'1', '0', '0', '1', '0'}

};

System.out.println("Maximal Rectangle Area: " + maximalRectangle(matrix1)); // Output: 6

}

}

Q2. Given an encoded string, return its decoded string. The encoding rule is: k[encoded\_string],

where the encoded\_string inside the square brackets is being repeated exactly k times. Note

that k is guaranteed to be a positive integer.

You may assume that the input string is always valid; there are no extra white spaces, square

brackets are well-formed, etc. Furthermore, you may assume that the original data does not

contain any digits and that digits are only for those repeat numbers, k. For example, there will

not be input like 3a or 2[4].

1 0 1 0 0

1 0

1 1

1 0 0 1 0

matrix =

[["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]

6

Explanation: The maximal rectangle is shown in the above picture.

Example 2:

matrix = [["0"]]

0

Example 3:

matrix = [["1"]]

1

Example 1:

s = "3[a]2[bc]"

"aaabcbc"

Example 2:

s = "3[a2[c]]"

"accaccacc"

Example 3:

s = "2[abc]3[cd]ef"

"abcabccdcdcdef"

import java.util.Stack;

public class DecodeString {

// Method to decode the encoded string

public static String decodeString(String s) {

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

Stack<String> strStack = new Stack<>();

StringBuilder currentString = new StringBuilder();

int currentNum = 0;

for (char ch : s.toCharArray()) {

if (Character.isDigit(ch)) {

currentNum = currentNum \* 10 + (ch - '0');

} else if (ch == '[') {

numStack.push(currentNum);

strStack.push(currentString.toString());

currentString = new StringBuilder();

currentNum = 0;

} else if (ch == ']') {

StringBuilder tempString = new StringBuilder(strStack.pop());

int repeatTimes = numStack.pop();

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

tempString.append(currentString);

}

currentString = tempString;

} else {

currentString.append(ch);

}

}

return currentString.toString();

}

public static void main(String[] args) {

String s1 = "3[a]2[bc]";

String s2 = "3[a2[c]]";

String s3 = "2[abc]3[cd]ef";

System.out.println("Decoded String 1: " + decodeString(s1)); // Output: "aaabcbc"

System.out.println("Decoded String 2: " + decodeString(s2)); // Output: "accaccacc"

System.out.println("Decoded String 3: " + decodeString(s3)); // Output: "abcabccdcdcdef"

}

}

Q3. You are keeping the scores for a baseball game with strange rules. At the beginning of the

game, you start with an empty record.

You are given a list of strings operations, where operations[i] is the ith operation you must apply

to the record and is one of the following:

An integer x.

Record a new score of x.

'+'.

Record a new score that is the sum of the previous two scores.

'D'.

Record a new score that is the double of the previous score.

'C'.

Invalidate the previous score, removing it from the record.

Return the sum of all the scores on the record after applying all the operations.

Example 1:

ops = ["5","2","C","D","+"]

30

Explanation:

"5" - Add 5 to the record, record is now [5].

"2" - Add 2 to the record, record is now [5, 2].

"C" - Invalidate and remove the previous score, record is now [5].

"D" - Add 2 \* 5 = 10 to the record, record is now [5, 10].

"+" - Add 5 + 10 = 15 to the record, record is now [5, 10, 15].

The total sum is 5 + 10 + 15 = 30.

Example 2:

ops = ["5","-2","4","C","D","9","+","+"]

27

Explanation:

"5" - Add 5 to the record, record is now [5].

"-2" - Add -2 to the record, record is now [5, -2].

"4" - Add 4 to the record, record is now [5, -2, 4].

"C" - Invalidate and remove the previous score, record is now [5, -2].

"D" - Add 2 \* -2 = -4 to the record, record is now [5, -2, -4].

"9" - Add 9 to the record, record is now [5, -2, -4, 9].

"+" - Add -4 + 9 = 5 to the record, record is now [5, -2, -4, 9, 5].

"+" - Add 9 + 5 = 14 to the record, record is now [5, -2, -4, 9, 5, 14].

The total sum is 5 + -2 + -4 + 9 + 5 + 14 = 27.

Example 3:

ops = ["1","C"]

import java.util.Stack;

public class BaseballGame {

// Method to compute the total score based on operations

public static int calPoints(String[] ops) {

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

for (String op : ops) {

switch (op) {

case "+":

int top = stack.pop();

int newTop = top + stack.peek();

stack.push(top); // Push the popped value back

stack.push(newTop);

break;

case "D":

stack.push(stack.peek() \* 2);

break;

case "C":

stack.pop();

break;

default:

stack.push(Integer.parseInt(op));

break;

}

}

// Compute the sum of all values in the stack

int sum = 0;

for (int score : stack) {

sum += score;

}

return sum;

}

public static void main(String[] args) {

String[] ops1 = {"5", "2", "C", "D", "+"};

String[] ops2 = {"5", "-2", "4", "C", "D", "9", "+", "+"};

String[] ops3 = {"1", "C"};

System.out.println("Total Score 1: " + calPoints(ops1)); // Output: 30

System.out.println("Total Score 2: " + calPoints(ops2)); // Output: 27

System.out.println("Total Score 3: " + calPoints(ops3)); // Output: 0

}

}

Q4. We are given an array of asteroids of integers representing asteroids in a row.For each

asteroid, the absolute value represents its size, and the sign represents its direction (positive

meaning right, negative meaning left). Each asteroid moves at the same speed.

Find out the state of the asteroids after all collisions. If two asteroids meet, the smaller one will

explode. If both are the same size, both will explode. Two asteroids moving in the same direction

will never meet.

Example 1:

asteroids = [5,10,-5]

[5,10]

Explanation: The 10 and -5 collide resulting in 10. The 5 and 10 never collide.

Example 2:

asteroids = [8,-8]

[]

Explanation: The 8 and -8 collide exploding each other.

Example 3:

asteroids = [10,2,-5]

[10]

Explanation: The 2 and -5 collide resulting in -5. The 10 and -5 collide resulting in 10.

Input:

Output:

import java.util.Stack;

public class AsteroidCollision {

// Method to find the state of asteroids after all collisions

public static int[] asteroidCollision(int[] asteroids) {

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

for (int asteroid : asteroids) {

// Handle the case where the asteroid is moving right or the stack is empty

if (asteroid > 0 || stack.isEmpty() || stack.peek() < 0) {

stack.push(asteroid);

} else {

// Handle collision

while (!stack.isEmpty() && stack.peek() > 0 && stack.peek() < Math.abs(asteroid)) {

stack.pop();

}

if (stack.isEmpty() || stack.peek() < 0) {

stack.push(asteroid);

} else if (stack.peek() == Math.abs(asteroid)) {

stack.pop(); // Both asteroids explode

}

// If the stack's top is greater than the current asteroid, do nothing

}

}

// Convert stack to array

int[] result = new int[stack.size()];

for (int i = stack.size() - 1; i >= 0; i--) {

result[i] = stack.pop();

}

return result;

}

public static void main(String[] args) {

int[] asteroids1 = {5, 10, -5};

int[] asteroids2 = {8, -8};

int[] asteroids3 = {10, 2, -5};

System.out.println("Asteroids after collision 1: " + arrayToString(asteroidCollision(asteroids1))); // [5, 10]

System.out.println("Asteroids after collision 2: " + arrayToString(asteroidCollision(asteroids2))); // []

System.out.println("Asteroids after collision 3: " + arrayToString(asteroidCollision(asteroids3))); // [10]

}

// Helper method to convert array to string

private static String arrayToString(int[] arr) {

if (arr.length == 0) return "[]";

StringBuilder sb = new StringBuilder();

sb.append("[");

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

if (i > 0) sb.append(", ");

sb.append(arr[i]);

}

sb.append("]");

return sb.toString();

}

}

Q5. Given an array of integers temperatures represents the daily temperatures, return an array

answer such that answer[i] is the number of days you have to wait after the ith day to get a

warmer temperature. If there is no future day for which this is possible, keep answer[i] == 0

instead.

Example 1:

temperatures = [73,74,75,71,69,72,76,73]

[1,1,4,2,1,1,0,0]

Example 2:

temperatures = [30,40,50,60]

[1,1,1,0]

Example 3:

temperatures = [30,60,90]

[1,1,0]

Input:

Output:

Input:

Output:

import java.util.Stack;

public class DailyTemperatures {

// Method to compute the number of days until a warmer temperature

public static int[] dailyTemperatures(int[] temperatures) {

int n = temperatures.length;

int[] answer = new int[n];

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

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

// Process each day

while (!stack.isEmpty() && temperatures[i] > temperatures[stack.peek()]) {

int index = stack.pop();

answer[index] = i - index;

}

stack.push(i);

}

// The stack contains indices of days with no warmer temperature in the future

return answer;

}

public static void main(String[] args) {

int[] temperatures1 = {73, 74, 75, 71, 69, 72, 76, 73};

int[] temperatures2 = {30, 40, 50, 60};

int[] temperatures3 = {30, 60, 90};

printArray("Days to warmer temperature 1: ", dailyTemperatures(temperatures1)); // [1, 1, 4, 2, 1, 1, 0, 0]

printArray("Days to warmer temperature 2: ", dailyTemperatures(temperatures2)); // [1, 1, 1, 0]

printArray("Days to warmer temperature 3: ", dailyTemperatures(temperatures3)); // [1, 1, 0]

}

// Helper method to print the array

private static void printArray(String message, int[] array) {

System.out.print(message);

System.out.print("[");

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

if (i > 0) System.out.print(", ");

System.out.print(array[i]);

}

System.out.println("]");

}

}