PW SKILLS

Lesson Plan

Stack-2

PW SKILLS

Q.Check for Balanced Brackets in an expression (well-formedness).

Input: exp = "[()]{}{[()()]()}"
Output: Balanced
Explanation: all the brackets are well-formed

Input: exp = "[(])"
Output: Not Balanced
Explanation: 1 and 4 brackets are not balanced because
there is a closing ']' before the closing '('

Code:

```
import java.util.Stack;

public class BalancedBrackets {
    public static boolean areBracketsBalanced(String expr) {
        // Declare a stack to hold the previous brackets.
        Stack<Character> temp = new Stack<>();
        for (int i = 0; i < expr.length(); i++) {
            if (temp.empty()) {
                // If the stack is empty
                // just push the current bracket
                temp.push(expr.charAt(i));
            } else if ((temp.peek() == '(' && expr.charAt(i) == ')')
                    || (temp.peek() == '{' && expr.charAt(i) == '}')
                    || (temp.peek() == '[' && expr.charAt(i) == ']')) {
                // If we found any complete pair of bracket
                // then pop
                temp.pop();
            } else {
                temp.push(expr.charAt(i));
            }
        }
        return temp.empty();
    }

    public static void main(String[] args) {
        String expression = "{[()]}";
        if (areBracketsBalanced(expression)) {
            System.out.println("Brackets are balanced.");
        } else {
            System.out.println("Brackets are not balanced.");
        }
    }
}
```

Q. Given a string S, The task is to remove all the consecutive duplicate characters of the string and return the resultant string.

Input: S= “aaaaabbbbbb”
Output: ab

Code:

```
public class DeleteConsecutiveStrings {
    public static String deleteConsecutiveStrings(String s) {
        // Initialize start and stop pointers
        int i = 0;
        int j = 0;

        // Initialize an empty string for new elements
        StringBuilder newElements = new StringBuilder();

        // Iterate string using j pointer
        while (j < s.length()) {
            // If both elements are same then skip
            if (s.charAt(i) == s.charAt(j)) {
                j++;
            }
            // If both elements are not same then append new element
            else {
                newElements.append(s.charAt(i));

                // After appending, slide over the window
                i = j;
            }
            j++;
        }

        // Append the last string
        newElements.append(s.charAt(j - 1));
        return newElements.toString();
    }

    public static void main(String[] args) {
        String input = "aabbcdeeff";
        String result = deleteConsecutiveStrings(input);
        System.out.println(result);
    }
}
```

Ques: Given an array, print the Next Greater Element (NGE) for every element.
Input: arr[] = [4 , 5 , 2 , 25]
Output: 4 –> 5
5 –> 25
2 –> 25
25 –> -1
Explanation: except 25 every element has an element greater than them present on the right side

Code:

```
import java.util.Stack;
public class NextGreaterElement {
    static void printNGE(int[] arr, int n) {
        Stack<Integer> s = new Stack<>();

        /* push the first element to stack */
        s.push(arr[0]);

        // iterate for rest of the elements
        for (int i = 1; i < n; i++) {

            if (s.empty()) {
                s.push(arr[i]);
                continue;
            }

            /*
             * if stack is not empty, then pop an element from stack. If the popped
             * element is smaller than next, then a) print the pair b) keep popping while
             * elements are smaller and stack is not empty
             */
            while (s.empty() == false && s.peek() < arr[i]) {
                System.out.println(s.pop() + " ⟶ " + arr[i]);
            }

            /* push next to stack so that we can find next greater for it */
            s.push(arr[i]);
        }

        while (s.empty() == false) {
            System.out.println(s.pop() + " ⟶ " + -1);
        }
    }

    public static void main(String[] args) {
        int[] arr = { 4, 5, 2, 10, 8 };
        int n = arr.length;
        printNGE(arr, n);
    }
}
```

Ques: Given an array, print the Next Greater Element (NGE) for every element.
Input: arr[] = [4 , 5 , 2 , 25]
Output: 4 –> 5
5 –> 25
2 –> 25
25 –> -1
Explanation: except 25 every element has an element greater than them present on the right side

Code:

```
import java.util.Stack;
public class NextGreaterElement {
    static void printNGE(int[] arr, int n) {
        Stack<Integer> s = new Stack<>();

        /* push the first element to stack */
        s.push(arr[0]);

        // iterate for rest of the elements
        for (int i = 1; i < n; i++) {

            if (s.empty()) {
                s.push(arr[i]);
                continue;
            }

            /*
             * if stack is not empty, then pop an element from stack. If the popped
             * element is smaller than next, then a) print the pair b) keep popping while
             * elements are smaller and stack is not empty
             */
            while (s.empty() == false && s.peek() < arr[i]) {
                System.out.println(s.pop() + " ⟶ " + arr[i]);
            }

            /* push next to stack so that we can find next greater for it */
            s.push(arr[i]);
        }

        while (s.empty() == false) {
            System.out.println(s.pop() + " ⟶ " + -1);
        }
    }

    public static void main(String[] args) {
        int[] arr = { 4, 5, 2, 10, 8 };
        int n = arr.length;
        printNGE(arr, n);
    }
}
```

Q. Given an array of distinct elements, find the previous greater element for every element. If the previous greater element does not exist, print -1.
Input : arr[] = {10, 4, 2, 20, 40, 12, 30}
Output : -1, 10, 4, -1, -1, 40, 40

Code:

```
import java.util.Stack;

public class PreviousGreaterElement {
    static void prevGreater(int[] arr, int n) {
        // Create a stack and push index of the first element to it
        Stack<Integer> s = new Stack<>();
        s.push(arr[0]);

        // Previous greater for the first element is always -1.
        System.out.print("-1, ");

        // Traverse remaining elements
        for (int i = 1; i < n; i++) {
            // Pop elements from the stack while the stack is not empty
            // and the top of the stack is smaller than arr[i].
            // We always have elements in decreasing order in a stack.
            while (!s.empty() && s.peek() < arr[i])
                s.pop();

            // If the stack becomes empty, then no element is greater
            // on the left side. Else, the top of the stack is the previous greater.
            System.out.print(s.empty() ? "-1, " : s.peek() + ", ");

            s.push(arr[i]);
        }
    }

    public static void main(String[] args) {
        int[] arr = { 10, 4, 2, 20, 40, 12, 30 };
        int n = arr.length;
        prevGreater(arr, n);
    }
}
```

Q. The stock span problem is a financial problem where we have a series of N daily price quotes for a stock and we need to calculate the span of the stock's price for all N days. The span Si of the stock's price on a given day i is defined as the maximum number of consecutive days just before the given day, for which the price of the stock on the current day is less than or equal to its price on the given day.

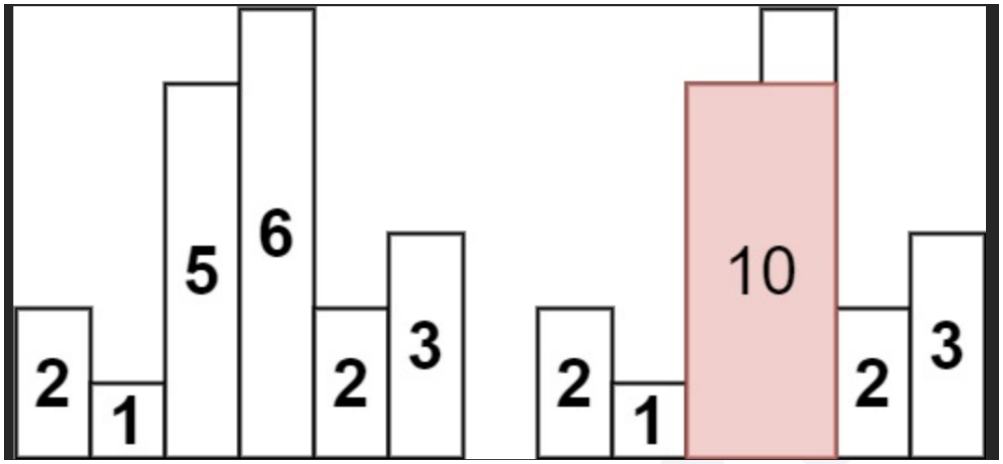
Input: N = 7, price[] = [100 80 60 70 60 75 85]
Output: 1 1 1 2 1 4 6
Explanation: Traversing the given input span for 100 will be 1, 80 is smaller than 100 so the span is 1, 60 is smaller than 80 so the span is 1, 70 is greater than 60 so the span is 2 and so on. Hence the output will be 1 1 1 2 1 4 6.

Code:

```
import java.util.Stack;

public class StockSpan {
    static void calculateSpan(int[] price, int n, int[] S) {
        // Create a stack and push the index of the first element to it
        Stack<Integer> st = new Stack<>();
        st.push(0);

        // Span value of the first element is always 1
        S[0] = 1;

        // Calculate span values for the rest of the elements
        for (int i = 1; i < n; i++) {
            // Pop elements from the stack while the stack is not
            // empty and the top of the stack is smaller than
            // price[i]
            while (!st.empty() && price[st.peek()] ⩽ price[i])
                st.pop();

            // If the stack becomes empty, then price[i] is
            // greater than all elements on the left of it,
            // i.e., price[0], price[1], ..price[i-1].
            // Else, price[i] is greater than elements after
            // the top of the stack
            S[i] = (st.empty()) ? (i + 1) : (i - st.peek());

            // Push this element to the stack
            st.push(i);
        }
    }

    public static void main(String[] args) {
        int[] price = { 10, 4, 5, 90, 120, 80 };
        int n = price.length;
        int[] span = new int[n];

        calculateSpan(price, n, span);

        for (int i = 0; i < n; i++) {
            System.out.print(span[i] + " ");
        }
    }
}
```

Ques. Given an array of integers heights representing the histogram's bar height where the width of each bar is 1, return the area of the largest rectangle in the histogram.(leetcode 84)

Input: heights = [2,1,5,6,2,3]
Output: 10
Explanation: The above is a histogram where width of each bar is 1.
The largest rectangle is shown in the red area, which has an area = 10 units.



Code:

```
import java.util.Stack;

public class Solution {
    public int largestRectangleArea(int[] heights) {
        // Method 2 Optimized

        // Store the distance of the next smaller element in the left
        Stack<Integer> leftStack = new Stack<>();
        int n = heights.length;
        int[] left = new int[n];
        for (int i = 0; i < n; i++) {
            while (!leftStack.isEmpty() && heights[leftStack.peek()] ⩾ heights[i]) {
                leftStack.pop();
            }
            left[i] = leftStack.isEmpty() ? i : i - leftStack.peek() - 1;
            leftStack.push(i);
        }

        // Similarly store the distance of the next smaller element in the right
        int[] right = new int[n];
        leftStack.clear();
        for (int i = n - 1; i ⩾ 0; i--) {
            while (!leftStack.isEmpty() && heights[leftStack.peek()] ⩾ heights[i]) {
                leftStack.pop();
            }
```

```
            right[i] = leftStack.isEmpty() ? n - 1 - i : leftStack.peek() - i - 1;
            leftStack.push(i);
        }

        // Now we have the next and right smaller element
        int ans = -1;
        for (int i = 0; i < n; i++) {
            int area = (left[i] + right[i] + 1) * heights[i];
            ans = Math.max(ans, area);
        }
        return ans;

        // Time → O(n), Space → O(3n)
    }
}
```

PW SKILLS

THANK YOU !