Intech Additive || ROUND 1 || TASK RESPONSE

QUESTION-A:

Implement a method to perform string compression. E.g. ‘aabccccccaaa’ should be a2b1c5a3. The

code to implement this is given in the link - https://www.educative.io/answers/string-compression-using-run-length-encoding

Think about memory occupied and how it can be improved.

Bonus 1:

The answer should be taken into second compressor and compress further.

E.g. a2b2c1a3c3 should become ab2c1ac3

Bonus 2: decompress2

ab2c1ac3 should return aabbcaaaccc.

Think about how you will test this code.

SOLUTION-A:

```
public class StringCompression {

    // String compression method
    public static String compress(String input) {
        StringBuilder result = new StringBuilder();
        int count = 1;

        for (int i = 0; i < input.length(); i++) {
            if (i + 1 < input.length() && input.charAt(i) == input.charAt(i + 1)) {
                count++;
            } else {
                result.append(input.charAt(i)).append(count);
```

```
            count = 1;
        }
    }

    return result.length() < input.length() ? result.toString() : input;
}

//Compress further
public static String compressFurther(String input) {
    StringBuilder result = new StringBuilder();
    int count = 1;

    for (int i = 0; i < input.length(); i += 2) {
        char character = input.charAt(i);
        int frequency = Character.getNumericValue(input.charAt(i + 1));

        for (int j = 0; j < frequency; j++) {
            result.append(character);
        }
    }

    return result.toString();
}

// Decompress further
public static String decompressFurther(String input) {
    StringBuilder result = new StringBuilder();

    for (int i = 0; i < input.length(); i++) {
        char character = input.charAt(i);
```

```
        int count = 0;

        while (i + 1 < input.length() && Character.isDigit(input.charAt(i + 1))) {
            count = count * 10 + Character.getNumericValue(input.charAt(++i));
        }

        for (int j = 0; j < count; j++) {
            result.append(character);
        }
    }

    return result.toString();
}

// Test the code
public static void main(String[] args) {
    String input = "aabccccccaaa";
    System.out.println("Original String: " + input);

    // Task: String Compression
    String compressedString = compress(input);
    System.out.println("Compressed String: " + compressedString);

    // Bonus 1: Compress Further
    String furtherCompressed = compressFurther(compressedString);
    System.out.println("Further Compressed: " + furtherCompressed);

    // Bonus 2: Decompress Further
    String decompressedString = decompressFurther(furtherCompressed);
    System.out.println("Decompressed String: " + decompressedString);
```

```
    }
}
```

QUESTION-B:

Linked List - The link shows a program to find the nth element of a linked list.

https://www.geeksforgeeks.org/nth-node-from-the-end-of-a-linked-list/

Find a way to find the kth to the last element of linked list (assume length of linked list is not

known)

Bonus 1:

Can you minimize the number of times you run through the loop.

SOLUTION-B:

```
class Node {
    int data;
    Node next;
    Node(int data) {
        this.data = data;
        this.next = null;
    }
}

public class LinkedList {

    Node head;

    // Function to find the kth to the last element of the linked list
    void printKthToLast(int k) {
        Node slow = head;
        Node fast = head;
```

```
    // Move fast pointer k nodes ahead
    for (int i = 0; i < k; i++) {
        if (fast == null) {
            System.out.println("List has fewer than k nodes");
            return;
        }
        fast = fast.next;
    }

    // Move both pointers until fast reaches the end
    while (fast != null) {
        slow = slow.next;
        fast = fast.next;
    }

    // At this point, slow is k nodes from the end
    System.out.println("Kth to the last element: " + slow.data);
}

public static void main(String[] args) {
    LinkedList list = new LinkedList();
    list.head = new Node(1);
    list.head.next = new Node(2);
    list.head.next.next = new Node(3);
    list.head.next.next.next = new Node(4);
    list.head.next.next.next.next = new Node(5);

    int k = 2; // Change k as needed
    list.printKthToLast(k);
```

```
        // Bonus 1: Minimize the number of times you run through the loop
        list.printKthToLastOptimized(k);
    }

    // Bonus 1: Optimize to minimize the number of times you run through the loop
    void printKthToLastOptimized(int k) {
        Node slow = head;
        Node fast = head;

        // Move fast pointer k nodes ahead
        for (int i = 0; i < k; i++) {
            if (fast == null) {
                System.out.println("List has fewer than k nodes");
                return;
            }
            fast = fast.next;
        }

        // Move both pointers until fast reaches the end
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next;
        }

        // At this point, slow is k nodes from the end
        System.out.println("Kth to the last element (Optimized): " + slow.data);
    }
}
```

QUESTION-C:

Stack minimum- Details of stack data structure is available in

https://www.geeksforgeeks.org/stack-data-structure/

Stack has functions of push and pop. Can you also add a function ‘min’ to the stack and it

should also execute in O(1).

If you are not aware of O(1), refer to some videos online. E.g.

https://en.wikipedia.org/wiki/Big_O_notation

Bonus 1 –

Explain one real world use case where stack is better used data structure than arrays.

SOLUTION-C:

```
import java.util.Stack;


class MinStack {
    Stack<Integer> mainStack;
    Stack<Integer> minStack;

    public MinStack() {
        mainStack = new Stack<>();
        minStack = new Stack<>();
    }

    // Push element onto the stack
    public void push(int x) {
        mainStack.push(x);

        // If minStack is empty or the current element is smaller than the top of minStack, push it onto minStack
        if (minStack.isEmpty() || x <= minStack.peek()) {
```

```
            minStack.push(x);
        }
    }

    // Pop element from the stack
    public void pop() {
        // If the element being popped from mainStack is the minimum, also pop it from minStack
        if (mainStack.peek().equals(minStack.peek())) {
            minStack.pop();
        }
        mainStack.pop();
    }

    // Get the minimum element from the stack
    public int min() {
        if (!minStack.isEmpty()) {
            return minStack.peek();
        }
        throw new IllegalStateException("Stack is empty");
    }
}

public class StackWithMinFunction {
    public static void main(String[] args) {
        MinStack stack = new MinStack();

        stack.push(3);
        stack.push(5);
        System.out.println("Minimum element: " + stack.min()); // Output: 3
```

```
        stack.push(2);
        stack.push(1);
        System.out.println("Minimum element: " + stack.min()); // Output: 1

        stack.pop();
        System.out.println("Minimum element: " + stack.min()); // Output: 2
    }
}
```

The **push** operation checks whether the element being pushed is smaller than or equal to the current minimum, and if so, it is also pushed onto **minStack**. The **pop** operation ensures that if the element being popped is the minimum, it is also popped from **minStack**.

Bonus 1: Real-world use case where a stack is better than arrays: One real-world use case is the management of function calls in a programming language. The call stack is a stack data structure that keeps track of the active function calls during the execution of a program. Each time a function is called, its context (local variables, return address, etc.) is pushed onto the call stack, and when the function returns, its context is popped. This allows for efficient management of function calls, especially in recursive scenarios, where each function call has its own context. Using an array for this purpose would be less efficient due to dynamic resizing and copying.

QUESTION-D:

Given an array of integers representing the elevation of a roof structure at various positions, each position is separated by a unit length, Write a program to determine the amount of water that will be trapped on the roof after heavy rainfall

Example:

input : [2 1 3 0 1 2 3]

SOLUTION-D

```
public class TrappingRainWater {

    public static int trap(int[] height) {
```

```
    int n = height.length;

    if (n <= 2) {
        return 0; // No trapping is possible with less than 3 elements
    }

    int left = 0;
    int right = n - 1;
    int leftMax = 0;
    int rightMax = 0;
    int result = 0;

    while (left < right) {
        leftMax = Math.max(leftMax, height[left]);
        rightMax = Math.max(rightMax, height[right]);

        if (height[left] < height[right]) {
            result += Math.max(0, leftMax - height[left]);
            left++;
        } else {
            result += Math.max(0, rightMax - height[right]);
            right--;
        }
    }

    return result;
}

public static void main(String[] args) {
    int[] height = {2, 1, 3, 0, 1, 2, 3};
```

```
        System.out.println("Amount of water trapped: " + trap(height));
    }
}
```

Explanation:

1. The **trap** function takes an array of integers representing the elevation of the roof.
2. Two pointers, **left** and **right**, start from both ends of the array.
3. **leftMax** and **rightMax** are used to keep track of the maximum elevation encountered from the left and right, respectively.
4. While the pointers haven't crossed each other:
    - If **height[left]** is less than **height[right]**, calculate the trapped water based on the left side, and move the **left** pointer.
    - If **height[left]** is greater than or equal to **height[right]**, calculate the trapped water based on the right side, and move the **right** pointer.
5. The result variable accumulates the total trapped water.
6. Print the result.

QUESTION-F:

What is dot product and cross product? Explain use cases of where dot product is used and

cross product is used in graphics environment. Add links to places where you studied this

information and get back with the understanding.

Bonus - How do you calculate the intersection between a ray and a plane/sphere/triangle?

SOLUTION-F:

Dot Product: The dot product, also known as the scalar product, is an algebraic operation that takes two equal-length sequences of numbers (usually coordinate vectors) and returns a single number. It is calculated by multiplying corresponding components of the two vectors and summing up the results.

Mathematically, for two vectors

A = [a1, a2, a3] and

B = [b1, b2, b3],

the dot product (A · B) is given by:

A.B =a1 xb1 + a2 x b2 + a3xb3

Use Cases in Graphics Environment:

- **Illumination Calculation:** Dot product is often used in computer graphics to calculate the cosine of the angle between the light direction and the surface normal. This is crucial for determining how much light a surface receives.
- **Shading:** In shading models, the dot product is used to compute the intensity of light on a surface, affecting its color and brightness.
- **Reflection and Refraction:** Dot product is employed in reflection and refraction calculations, helping determine the behavior of light rays when they interact with surfaces.

Cross Product: The cross product, also known as the vector product, is another binary operation on two vectors in three-dimensional space. It results in a vector that is perpendicular to both input vectors. Mathematically, for two vectors A = [a1, a2, a3] and B = [b1, b2, b3], the cross product (A x B) is given by:

A x B = [a2 x b3 – a3 x b2 , a3 x b1 – a1 x b3 , a1 x b2 – a2 x b1]

Use Cases in Graphics Environment:

- **Surface Normal Calculation:** Cross product is extensively used to calculate surface normals. Knowing the normal is crucial for proper lighting calculations.
- **Orientation Determination:** Cross product can be used to determine the orientation of an object in 3D space, which is useful for various graphics operations.
- **Generating Tangent Vectors:** In 3D computer graphics, tangent vectors are often needed for mapping textures onto surfaces. Cross product can help generate tangent vectors.

QUESTION – G:

Explain a piece of code that you wrote which you are proud of? If you have not written any

code, please write your favorite subject in engineering studies. We can go deep into that

subject.

SOLUTION -G

```
import java.util.Scanner;


public class FibonacciSeries {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);


        System.out.print("Enter the number of terms in the Fibonacci series: ");
        int n = scanner.nextInt();


        generateFibonacciSeries(n);
    }


    static void generateFibonacciSeries(int n) {
        int firstTerm = 0, secondTerm = 1;


        System.out.println("Fibonacci Series:");


        for (int i = 0; i < n; i++) {
            System.out.print(firstTerm + " ");


            int nextTerm = firstTerm + secondTerm;
            firstTerm = secondTerm;
            secondTerm = nextTerm;
```

```
        }
    }
}
```

In this program, the generateFibonacciSeries method takes the number of terms (n) as an argument and prints the Fibonacci series up to the specified number of terms. The series starts with 0 and 1, and each subsequent term is the sum of the two preceding ones.