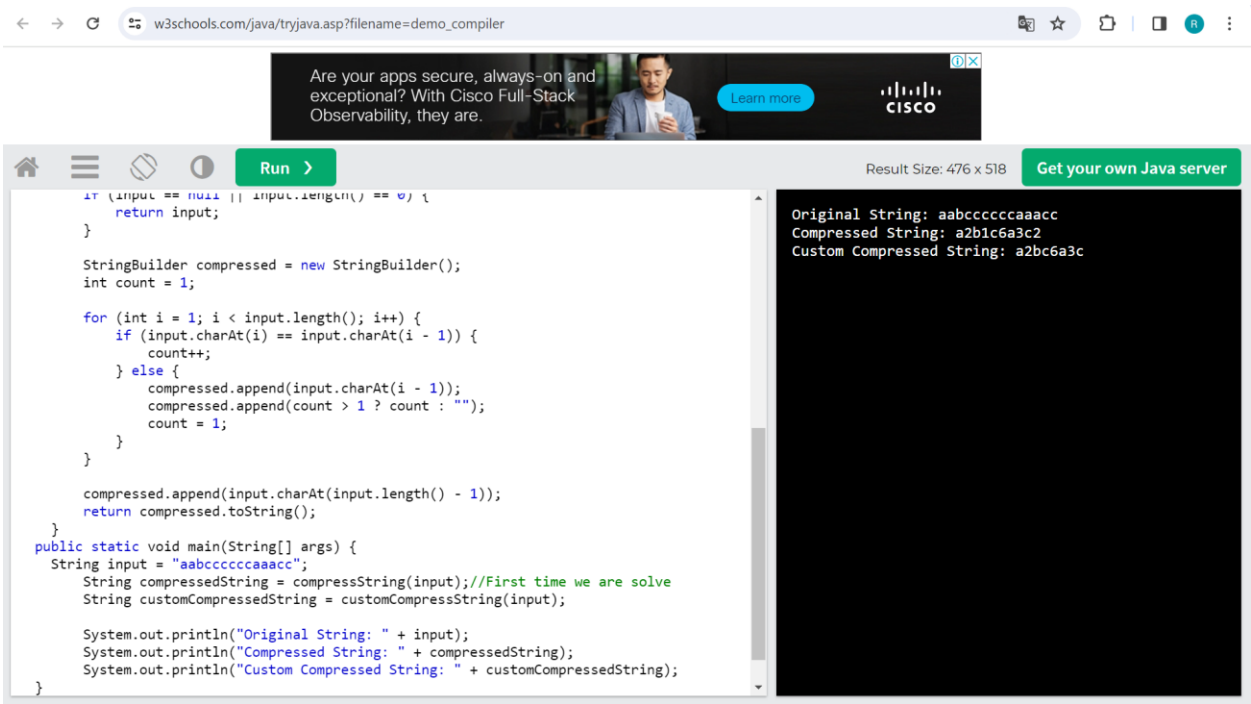
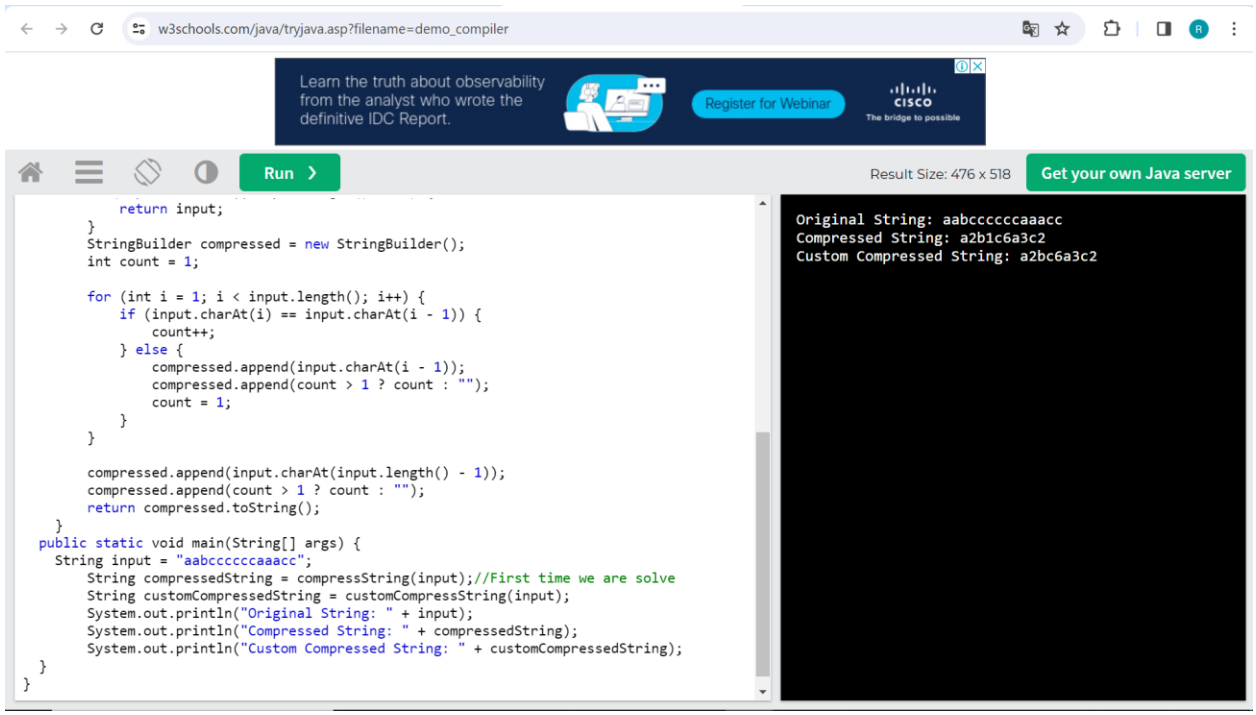
A. String Compression → I attach online editor code there also

```
public class Main {

public static String compressString(String input) {
    if (input == null || input.length() == 0) {
      return input;
    }

    StringBuilder compressed = new StringBuilder();
    int count = 1;

    for (int i = 1; i < input.length(); i++) {
      if (input.charAt(i) == input.charAt(i - 1)) {
        count++;
      } else {
        compressed.append(input.charAt(i - 1));
        compressed.append(count);
        count = 1;
      }
    }
    compressed.append(input.charAt(input.length() - 1));
    compressed.append(count);
    String result = compressed.toString();
    return result.length() < input.length() ? result : input;
  }
  public static String customCompressString(String input) {
    if (input == null || input.length() == 0) {
      return input;
    }
    StringBuilder compressed = new StringBuilder();
    int count = 1;

    for (int i = 1; i < input.length(); i++) {
      if (input.charAt(i) == input.charAt(i - 1)) {
        count++;
      } else {
        compressed.append(input.charAt(i - 1));
        compressed.append(count > 1 ? count : "");
        count = 1;
      }
    }

    compressed.append(input.charAt(input.length() - 1));
    compressed.append(count > 1 ? count : "");
```

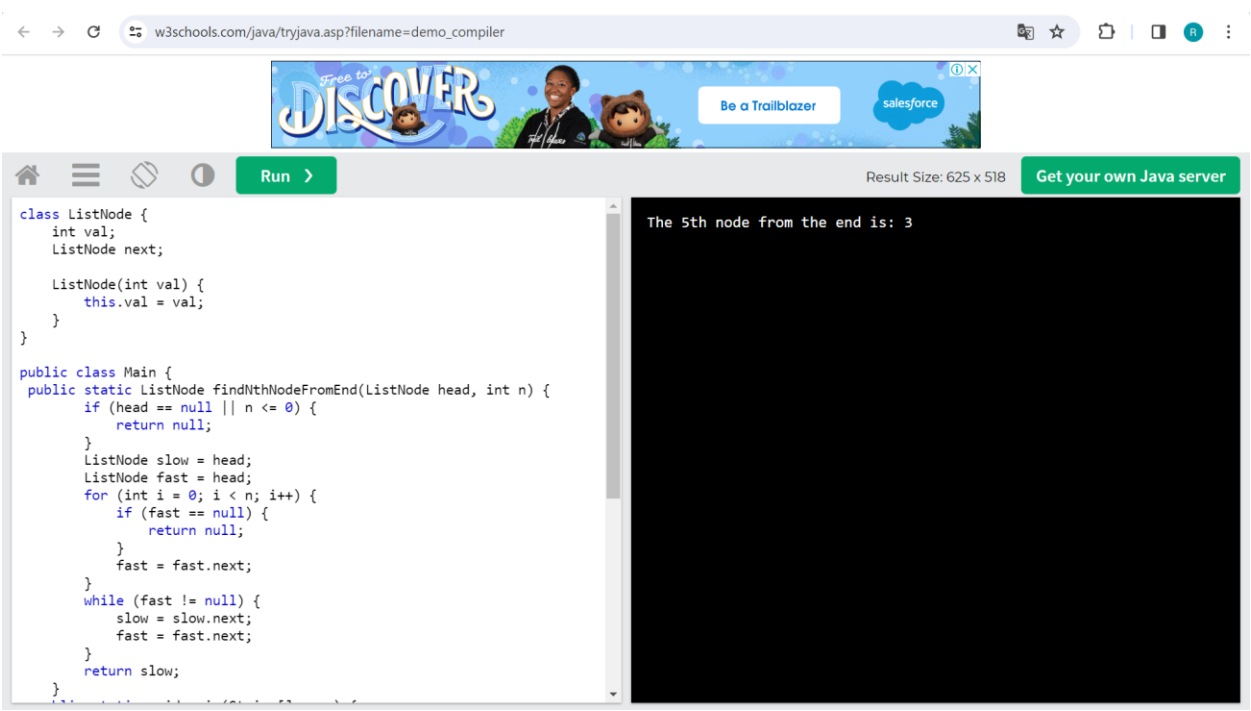
```
    return compressed.toString();
  }
 public static void main(String[] args) {
  String input = "aabccccccaaacc";
    String compressedString = compressString(input);//First time we are solve
    String customCompressedString = customCompressString(input);
    System.out.println("Original String: " + input);
    System.out.println("Compressed String: " + compressedString);
    System.out.println("Custom Compressed String: " + customCompressedString);
 }
}
```



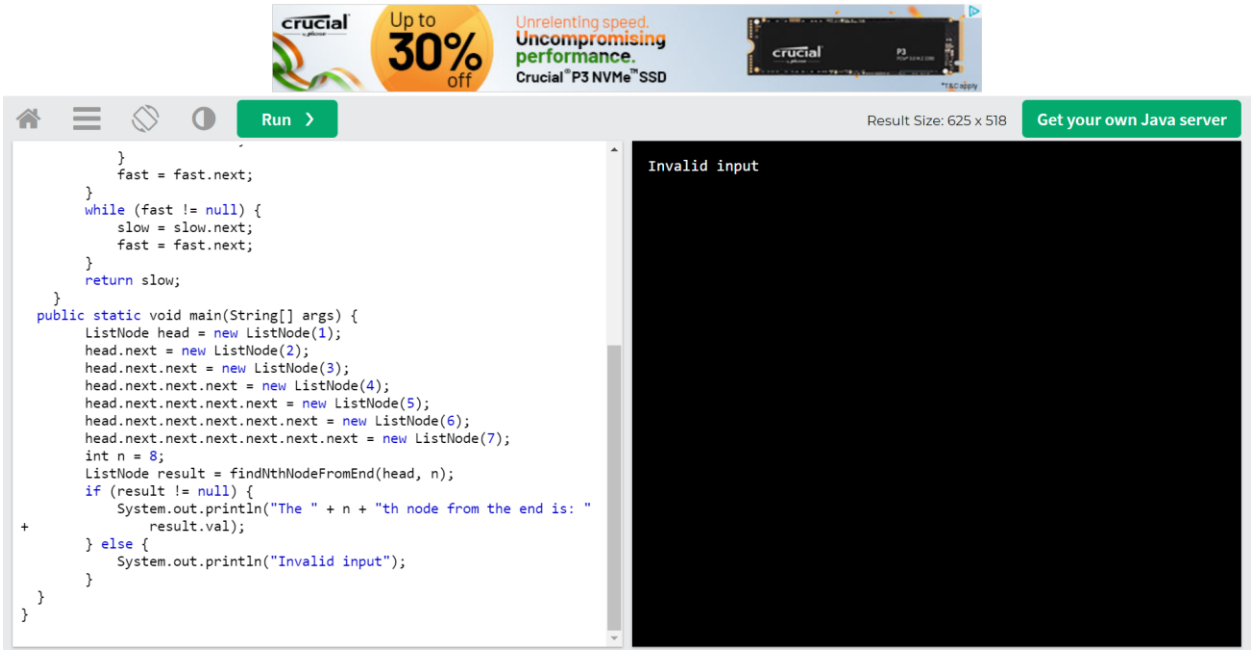


B. The Link Shows a Program to find the nth element of linked list→

Positive Scenario →



Negative Scenario→




```
class ListNode {
    int val;
    ListNode next;
```

```
  ListNode(int val) {
    this.val = val;
  }
}

public class Main {
 public static ListNode findNthNodeFromEnd(ListNode head, int n) {
    if (head == null || n <= 0) {
      return null;
    }
    ListNode slow = head;
    ListNode fast = head;
    for (int i = 0; i < n; i++) {
      if (fast == null) {
        return null;
      }
      fast = fast.next;
    }
    while (fast != null) {
      slow = slow.next;
      fast = fast.next;
    }
    return slow;
  }
 public static void main(String[] args) {
                    ListNode head = new ListNode(1);
    head.next = new ListNode(2);
    head.next.next = new ListNode(3);
```

```
    head.next.next.next = new ListNode(4);

    head.next.next.next.next = new ListNode(5);

    head.next.next.next.next.next = new ListNode(6);

    head.next.next.next.next.next.next = new ListNode(7);

    int n = 8;

    ListNode result = findNthNodeFromEnd(head, n);

    if (result != null) {

      System.out.println("The " + n + "th node from the end is: " +
        result.val);

    } else {

      System.out.println("Invalid input");

    }

  }

}
```

C. Water will Be Trapped

Run › Result Size: 625 x 518 Get your own Java server

```
public class Main {
    public static int trap(int[] height) {
        int n = height.length;
        if (n <= 2) {
            return 0;
        }
        int first = 0;
        int last = n - 1;
        int firstMax = 0;
        int lastMax = 0;
        int water = 0;
        while (first < last) {
            firstMax = Math.max(firstMax, height[first]);
            lastMax = Math.max(lastMax, height[last]);
            if (height[first] < height[last]) {
                water += firstMax - height[first];
                first++;
            } else {
                water += lastMax - height[last];
                last--;
            }
        }

        return water;
    }

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

```
The amount of water trapped is: 7
```

Run

Result Size: 625 x 518

Get your own Java server

```
        int first = 0;
        int last = n - 1;
        int firstMax = 0;
        int lastMax = 0;
        int water = 0;
        while (first < last) {
            firstMax = Math.max(firstMax, height[first]);
            lastMax = Math.max(lastMax, height[last]);
            if (height[first] < height[last]) {
                water += firstMax - height[first];
                first++;
            } else {
                water += lastMax - height[last];
                last--;
            }
        }

        return water;
    }

  public static void main(String[] args) {
        int[] elevation = {2,1,3,0,1,2,3};
        int result = trap(elevation);
        System.out.println("The amount of water trapped is: " +
result);
    }
}
```

```
The amount of water trapped is: 7
```

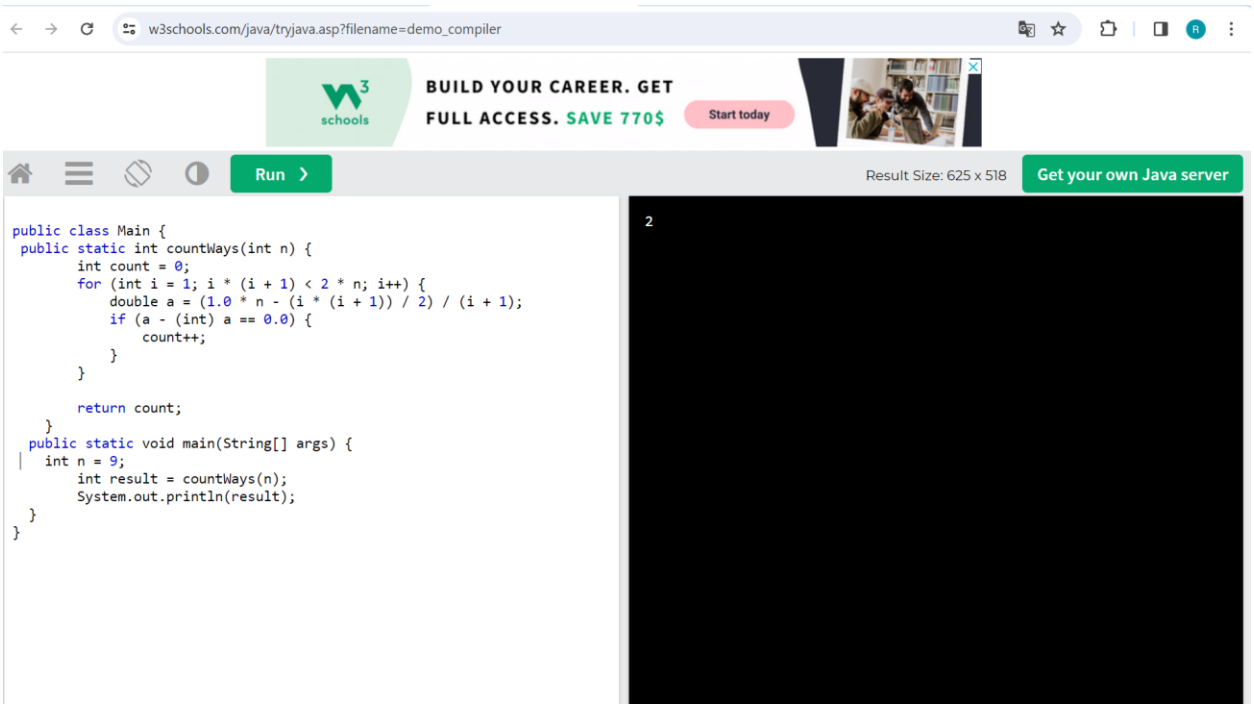
```
public class Main {
  public static int trap(int[] height) {
    int n = height.length;
    if (n <= 2) {
      return 0;
    }
    int first = 0;
    int last = n - 1;
    int firstMax = 0;
    int lastMax = 0;
    int water = 0;
    while (first < last) {
      firstMax = Math.max(firstMax, height[first]);
      lastMax = Math.max(lastMax, height[last]);
      if (height[first] < height[last]) {
        water += firstMax - height[first];
```

```
            first++;
        } else {
            water += lastMax - height[last];
            last--;
        }
    }

    return water;
  }

 public static void main(String[] args) {
                    int[] elevation = {2,1,3,0,1,2,3};
    int result = trap(elevation);
    System.out.println("The amount of water trapped is: " + result);
            }
}
```

D. N consecutive Number of ways →

```
public class Main {
 public static int countWays(int n) {
    int count = 0;
    for (int i = 1; i * (i + 1) < 2 * n; i++) {
        double a = (1.0 * n - (i * (i + 1)) / 2) / (i + 1);
        if (a - (int) a == 0.0) {
            count++;
        }
    }

    return count;
```

```
  }
 public static void main(String[] args) {
  int n = 9;
     int result = countWays(n);
     System.out.println(result);
 }
}
```



I am trying one another method but I got fall in recursive approach

```
import java.util.HashMap;
import java.util.Map;
public class Main {


private static Map<String, Integer> map = new HashMap<>();
private static int countWays(int n){
```

```
map.clear();
return countWaysRecursive(n,1);
}
 private static int countWaysRecursive(int n, int start) {
      if (n <= 0) {
         return 0;
      }
      if (n == start) {
         return 1;
      }
      String key = n+"-"+start;
     int ways = countWaysRecursive(n - start, start + 1) + countWaysRecursive(n, start + 1);
     map.put(key,ways);
      return ways;
   }
  public static void main(String[] args) {
   int n = 15;

      int result = countWays(n);
      System.out.println(result);
  }
}
```

But I get Stack overflow error so there time have less so in future I again try to another approach.

E. I am not able solve overself because I am not get meaning of Pi.
F. The dot product and cross product are mathematical operations used in vector algebra to calculate the relationship between two vectors. They have many real-life applications in fields such as physics, engineering, and computer graphics.

Reference: https://www.physicsforums.com/threads/what-is-the-real-life-utility-of-the-dot-product-and-cross-product.914549/

G. String compression and Linked List code
H. I have provided a code snippet above for a recursive approach to find the sum of consecutive numbers. However, I am encountering a stack overflow error. I need assistance in debugging the code and finding the most effective solution. The recursive approach is preferred, and I would like to focus on resolving the specific issue causing the stack overflow. Once this initial step is addressed, the rest of the recursion should handle the task efficiently.