CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING

ASSIGNMENT 1

STUDENT NAME: LAKSHIT MALHOTRA **UID:** 22BCS13047
BRANCH: CSE **SECTION:** 22BCS_FL_IOT_601A
SEMESTER: 6 **DATE OF PERFORMANCE:** 10/04/25
SUBJECT NAME: AP LAB -2 **SUBJECT CODE:** 22CSP-351

73 - SET MATRIX ZEROES

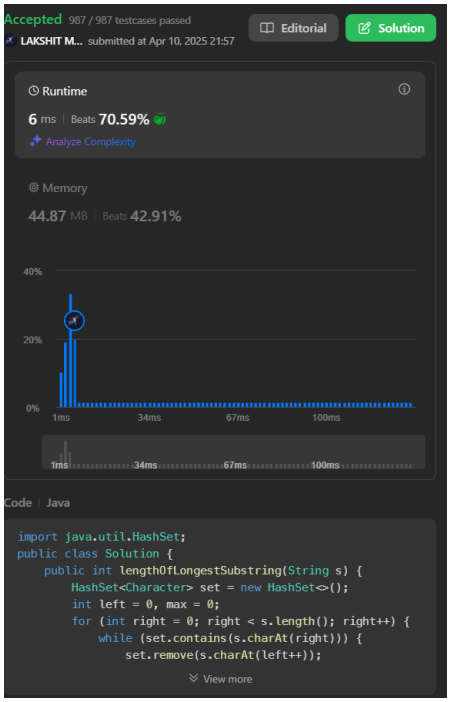
🔗 https://leetcode.com/problems/set-matrix-zeroes/

```
public class Solution {
    public void setZeroes(int[][] matrix) {
        int m = matrix.length, n = matrix[0].length;
        boolean firstRow = false, firstCol = false;
        for (int i = 0; i < m; i++)
            if (matrix[i][0] == 0) firstCol = true;
        for (int j = 0; j < n; j++)
            if (matrix[0][j] == 0) firstRow = true;
        for (int i = 1; i < m; i++)
            for (int j = 1; j < n; j++)
                if (matrix[i][j] == 0)
                    matrix[i][0] = matrix[0][j] = 0;
        for (int i = 1; i < m; i++)
            for (int j = 1; j < n; j++)
                if (matrix[i][0] == 0 || matrix[0][j] == 0)
                    matrix[i][j] = 0;
        if (firstCol)
            for (int i = 0; i < m; i++) matrix[i][0] = 0;
        if (firstRow)
            for (int j = 0; j < n; j++) matrix[0][j] = 0;
    }
}
```

3 - LONGEST SUBSTRING WITHOUT REPEATING CHARACTERS

🔗 https://leetcode.com/problems/longest-substring-without-repeating-characters/

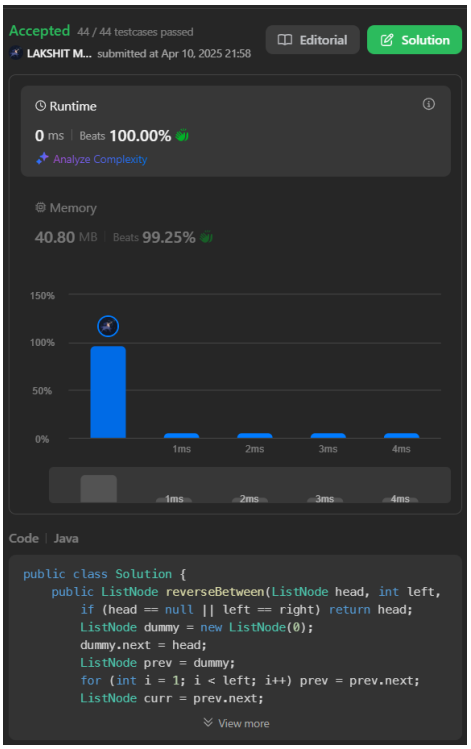
```
import java.util.HashSet;
public class Solution {
    public int lengthOfLongestSubstring(String s) {
        HashSet<Character> set = new HashSet<>();
        int left = 0, max = 0;
        for (int right = 0; right < s.length(); right++) {
            while (set.contains(s.charAt(right))) {
                set.remove(s.charAt(left++));
            }
            set.add(s.charAt(right));
            max = Math.max(max, right - left + 1);
        }
        return max;
    }
}
```



92 - REVERSE LINKED LIST II

🔗 https://leetcode.com/problems/reverse-linked-list-ii/

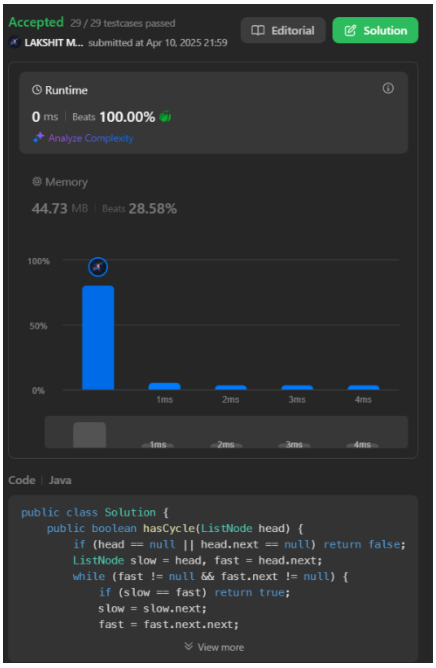
```
public class Solution {
    public ListNode reverseBetween(ListNode head, int
left, int right) {
        if (head == null || left == right) return head;
        ListNode dummy = new ListNode(0);
        dummy.next = head;
        ListNode prev = dummy;
        for (int i = 1; i < left; i++) prev = prev.next;
        ListNode curr = prev.next;
        for (int i = 0; i < right - left; i++) {
            ListNode temp = curr.next;
            curr.next = temp.next;
            temp.next = prev.next;
            prev.next = temp;
        }
        return dummy.next;
    }
}
```



141 - DETECT CYCLE IN A LINKED LIST

🔗 https://leetcode.com/problems/linked-list-cycle/

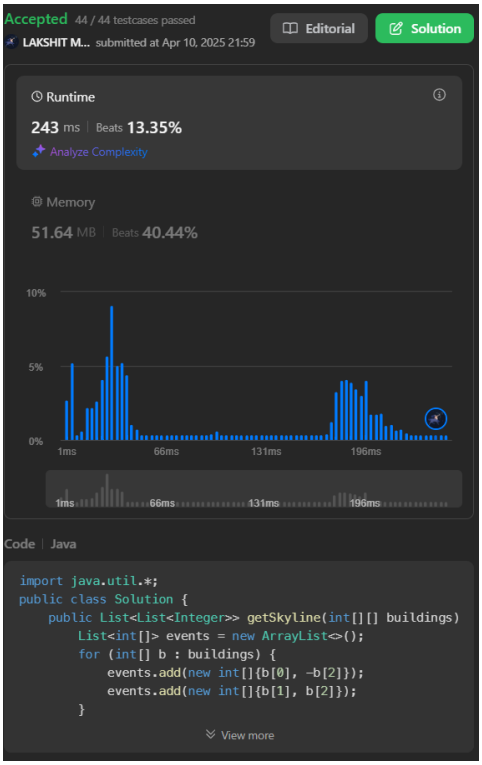
```
public class Solution {
    public boolean hasCycle(ListNode head) {
        if (head == null || head.next == null) return false;
        ListNode slow = head, fast = head.next;
        while (fast != null && fast.next != null) {
            if (slow == fast) return true;
            slow = slow.next;
            fast = fast.next.next;
        }
        return false;
    }
}
```



218 - THE SKYLINE PROBLEM

🔗 https://leetcode.com/problems/the-skyline-problem/

```
import java.util.*;
public class Solution {
    public List<List<Integer>> getSkyline(int[][]
buildings) {
        List<int[]> events = new ArrayList<>();
        for (int[] b : buildings) {
            events.add(new int[]{b[0], -b[2]});
            events.add(new int[]{b[1], b[2]});
        }
        events.sort((a, b) -> {
            if (a[0] != b[0]) return a[0] - b[0];
            return a[1] - b[1];
        });
        List<List<Integer>> result = new ArrayList<>();
        PriorityQueue<Integer> pq = new
PriorityQueue<>(Collections.reverseOrder());
        pq.add(0);
        int prev = 0;
        for (int[] e : events) {
            if (e[1] < 0) pq.add(-e[1]);
```

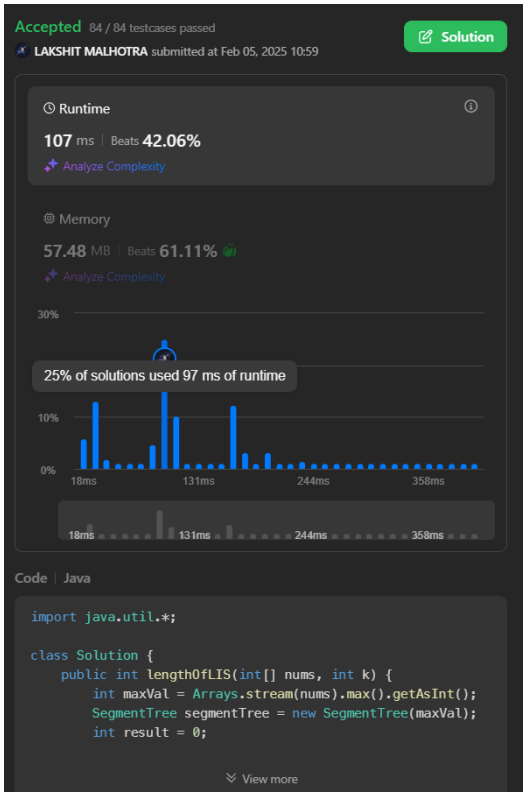


```
 else pq.remove(e[1]);
            int curr = pq.peek();
            if (curr != prev) {
                result.add(Arrays.asList(e[0], curr));
                prev = curr;
            }
        }
        return result;
    }
}
```

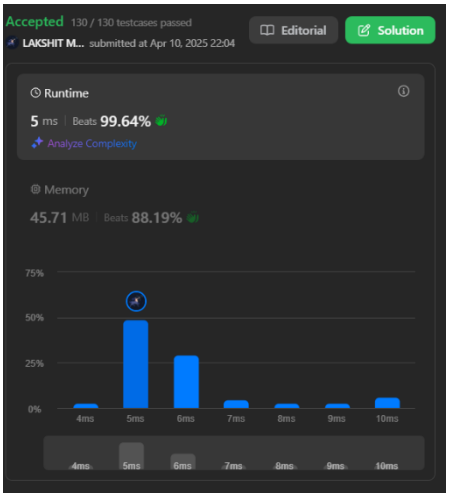
2407 - LONGEST INCREASING SUBSEQUENCE II

🔗 https://leetcode.com/problems/longest-increasing-subsequence-ii/

```
import java.util.*;
class Solution {
    public int lengthOfLIS(int[] nums, int k) {
        int maxVal = Arrays.stream(nums).max().getAsInt();
        SegmentTree segmentTree = new SegmentTree(maxVal);
        int result = 0;
        for (int num : nums) {
            int maxPrev = 0;
            if (num > 1) {
                maxPrev = segmentTree.query(Math.max(1,
num - k), num - 1);
            }
            segmentTree.update(num, maxPrev + 1);
            result = Math.max(result, maxPrev + 1);
        }
        return result;
    }
    static class SegmentTree {
        int[] tree;
        int size;
        public SegmentTree(int size) {
            this.size = size;
            tree = new int[4 * size];
        }
        public void update(int index, int value) {
            update(1, 1, size, index, value);
        }
```



```
    private void update(int node, int start, int end, int index, int value) {
        if (start == end) {
            tree[node] = Math.max(tree[node], value);
        } else {
            int mid = start + (end - start) / 2;
            if (index <= mid) {
                update(2 * node, start, mid, index, value);
            } else {
                update(2 * node + 1, mid + 1, end, index, value);
            }
            tree[node] = Math.max(tree[2 * node], tree[2 * node + 1]);
        }
    }
    public int query(int left, int right) {
        return query(1, 1, size, left, right);
    }
    private int query(int node, int start, int end, int left, int right) {
        if (left > end || right < start) {
            return 0;
        }
        if (left <= start && end <= right) {
            return tree[node];
        }
        int mid = start + (end - start) / 2;
        int leftQuery = query(2 * node, start, mid, left, right);
        int rightQuery = query(2 * node + 1, mid + 1, end, left, right);
        return Math.max(leftQuery, rightQuery);
    }
}
}
```



240 - SEARCH A 2D MATRIX II

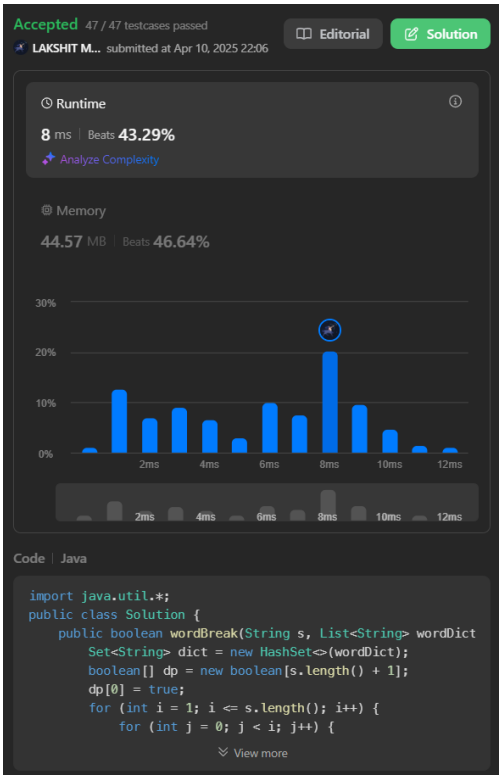
🔗 https://leetcode.com/problems/search-a-2d-matrix-ii/

```
public class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int row = 0, col = matrix[0].length - 1;
        while (row < matrix.length && col >= 0) {
            if (matrix[row][col] == target) return true;
            else if (matrix[row][col] < target) row++;
```

```
            else col--;
        }
        return false;
    }
}
```

139 - WORD BREAK

🔗 https://leetcode.com/problems/word-break/



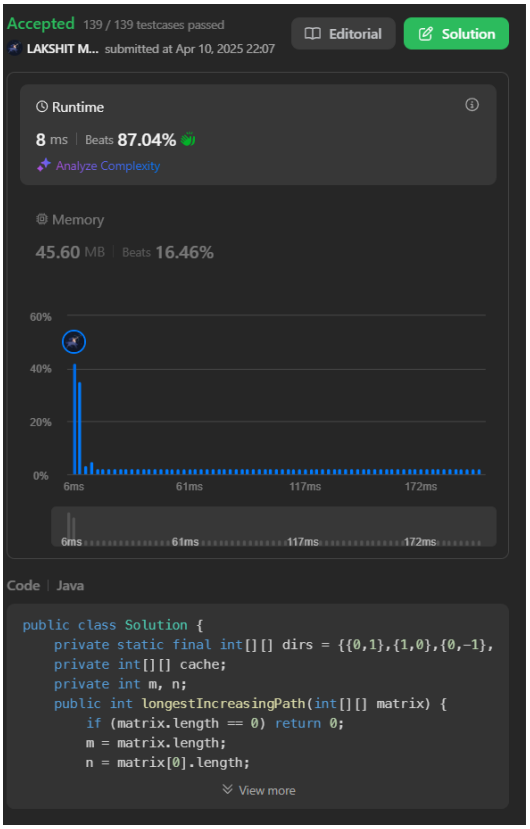

```
import java.util.*;
public class Solution {
    public boolean wordBreak(String s, List<String> wordDict) {
        Set<String> dict = new HashSet<>(wordDict);
        boolean[] dp = new boolean[s.length() + 1];
        dp[0] = true;
        for (int i = 1; i <= s.length(); i++) {
            if (dp[j] && dict.contains(s.substring(j, i))) {
                dp[i] = true;
                break;
            }
        }
        return dp[s.length()];
    }
}
```

329 - LONGEST INCREASING PATH IN A MATRIX

🔗 https://leetcode.com/problems/longest-increasing-path-in-a-matrix/

```
public class Solution {
    private static final int[][] dirs = {{0,1},{1,0},{0,-1},{-1,0}};
    private int[][] cache;
    private int m, n;
    public int longestIncreasingPath(int[][] matrix) {
        if (matrix.length == 0) return 0;
        m = matrix.length;
        n = matrix[0].length;
        cache = new int[m][n];
```

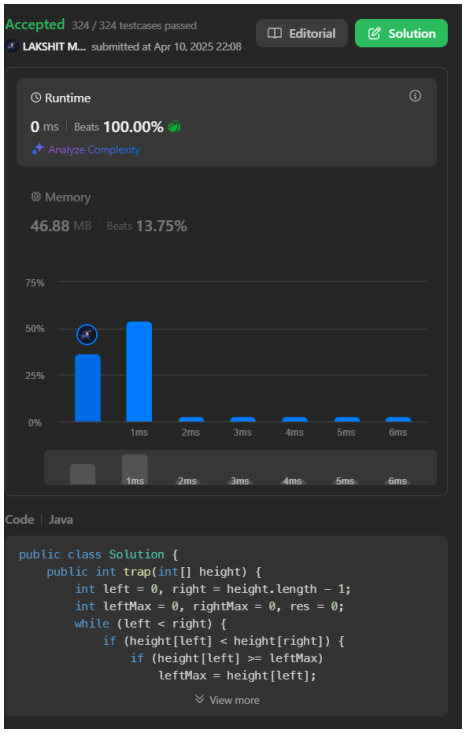
```
    int max = 0;
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
                max = Math.max(max, dfs(matrix, i, j));
        return max;
    }
    private int dfs(int[][] matrix, int i, int j) {
        if (cache[i][j] != 0) return cache[i][j];
        int max = 1;
        for (int[] d : dirs) {
            int x = i + d[0], y = j + d[1];
            if (x >= 0 && x < m && y >= 0 && y < n
&& matrix[x][y] > matrix[i][j]) {
                int len = 1 + dfs(matrix, x, y);
                max = Math.max(max, len);
            }
        }
        cache[i][j] = max;
        return max;
    }
}
```



42 - TRAPPING RAIN WATER

🔗 https://leetcode.com/problems/trapping-rain-water/

```
public class Solution {
    public int trap(int[] height) {
        int left = 0, right = height.length - 1;
        int leftMax = 0, rightMax = 0, res = 0;
        while (left < right) {
            if (height[left] < height[right]) {
                if (height[left] >= leftMax)
                    leftMax = height[left];
                else
                    res += leftMax - height[left];
                left++;
            } else {
                if (height[right] >= rightMax)
                    rightMax = height[right];
```



```
    else
                    res += rightMax - height[right];
                right--;
            }
        }
        return res;
    }
}
```