21TH AUGUST HEAP, STACK AND QUEUE

Q1. Find the largest rectangle of '1's in a binary matrix:

```java
public int maximalRectangle(char[][] matrix) {
    if (matrix == null || 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++) {
            if (matrix[i][j] == '1') {
                heights[j]++;
            } else {
                heights[j] = 0;
            }
        }
        maxArea = Math.max(maxArea, largestRectangleArea(heights));
    }

    return maxArea;
}
```

```
private int largestRectangleArea(int[] heights) {
    Stack<Integer> stack = new Stack<>();
    int maxArea = 0;

    for (int i = 0; i <= heights.length; i++) {
        int h = (i == heights.length) ? 0 : heights[i];

        while (!stack.isEmpty() && h < heights[stack.peek()]) {
            int height = heights[stack.pop()];
            int width = stack.isEmpty() ? i : i - stack.peek() - 1;
            maxArea = Math.max(maxArea, height * width);
        }

        stack.push(i);
    }

    return maxArea;
}
```

Q2. Decode an encoded string:

```java
public String decodeString(String s) {
    Stack<Integer> countStack = new Stack<>();
    Stack<String> stringStack = new Stack<>();
    String currentString = "";
    int count = 0;

    for (char ch : s.toCharArray()) {
        if (Character.isDigit(ch)) {
            count = count * 10 + (ch - '0');
```

```
        } else if (ch == '[') {
            countStack.push(count);
            stringStack.push(currentString);
            count = 0;
            currentString = "";
        } else if (ch == ']') {
            StringBuilder temp = new StringBuilder(stringStack.pop());
            int repeatTimes = countStack.pop();
            for (int i = 0; i < repeatTimes; i++) {
                temp.append(currentString);
            }
            currentString = temp.toString();
        } else {
            currentString += ch;
        }
    }

    return currentString;
}
```

Q3. Sum of baseball game scores:

```java
public int calPoints(String[] ops) {
    Stack<Integer> stack = new Stack<>();

    for (String op : ops) {
        if (op.equals("C")) {
            stack.pop();
        } else if (op.equals("D")) {
            stack.push(stack.peek() * 2);
```

```
        } else if (op.equals("+")) {
            int top = stack.pop();
            int newTop = top + stack.peek();
            stack.push(top);
            stack.push(newTop);
        } else {
            stack.push(Integer.parseInt(op));
        }
    }

    int sum = 0;
    for (int score : stack) {
        sum += score;
    }

    return sum;
}
```

Q4. Asteroid collision:

```java
public int[] asteroidCollision(int[] asteroids) {
    Stack<Integer> stack = new Stack<>();

    for (int asteroid : asteroids) {
        while (!stack.isEmpty() && asteroid < 0 && stack.peek() > 0) {
            if (Math.abs(asteroid) > stack.peek()) {
                stack.pop();
                continue;
            } else if (Math.abs(asteroid) == stack.peek()) {
                stack.pop();
```

```
            }
            break;
        }
        if (asteroid != 0 || (asteroid == 0 && stack.isEmpty())) {
            stack.push(asteroid);
        }
    }

    int[] result = new int[stack.size()];
    for (int i = result.length - 1; i >= 0; i--) {
        result[i] = stack.pop();
    }

    return result;
}
```

Q5. Wait for warmer temperatures:

```java
public int[] dailyTemperatures(int[] temperatures) {
    int[] result = new int[temperatures.length];
    Stack<Integer> stack = new Stack<>();

    for (int i = 0; i < temperatures.length; i++) {
        while (!stack.isEmpty() && temperatures[i] > temperatures[stack.peek()]) {
            int idx = stack.pop();
            result[idx] = i - idx;
        }
        stack.push(i);
    }
```

```
    return result;
}
```

Q6. Implement a Map in Java with sorted keys:

```java
import java.util.*;

public class SortedKeyMap<K, V> extends TreeMap<K, V> {
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder("{");
        boolean first = true;
        for (Map.Entry<K, V> entry : entrySet()) {
            if (!first) {
                sb.append(", ");
            }
            sb.append(entry.getKey()).append("=").append(entry.getValue());
            first = false;
        }
        sb.append("}");
        return sb.toString();
    }
}
```

Q7. Implement a Map in Java with sorted values:

```java
import java.util.*;
```

```
public class SortedValueMap<K, V> extends TreeMap<K, V> {
    private Comparator<? super V> valueComparator;

    public SortedValueMap(Comparator<? super V> valueComparator) {
        this.valueComparator = valueComparator;
    }

    @Override
    public String toString() {
        List<Map.Entry<K, V>> entries = new ArrayList<>(entrySet());
        entries.sort((entry1, entry2) -> valueComparator.compare(entry1.getValue(),
entry2.getValue()));

        StringBuilder sb = new StringBuilder("{");
        boolean first = true;
        for (Map.Entry<K, V> entry : entries) {
            if (!first) {
                sb.append(", ");
            }
            sb.append(entry.getKey()).append("=").append(entry.getValue());
            first = false;
        }
        sb.append("}");
        return sb.toString();
    }
}
```

Q8. Detect duplicate element in an array:

```java
public boolean containsDuplicate(int[] nums) {
    Set<Integer> numSet = new HashSet<>();
```

```
    for (int num : nums) {
        if (numSet.contains(num)) {
            return true;
        }
        numSet.add(num);
    }

    return false;
}
```

Q9. Find majority element in an array:

```java
public int majorityElement(int[] nums) {
    int majority = nums[0];
    int count = 1;

    for (int i = 1; i < nums.length; i++) {
        if (count == 0) {
            majority = nums[i];
        }

        if (nums[i] == majority) {
            count++;
        } else {
            count--;
        }
    }

    return majority;
```

}
```

**Q10. Check if ransomNote can be constructed from magazine:**

```java
public boolean canConstruct(String ransomNote, String magazine) {
    int[] counts = new int[26];



 for (char ch : magazine.toCharArray()) {
        counts[ch - 'a']++;
    }

    for (char ch : ransomNote.toCharArray()) {
        if (counts[ch - 'a'] > 0) {
            counts[ch - 'a']--;
        } else {
            return false;
        }
    }

    return true;
}
```
```