Assignment 17 Solution - Queues | DSA

Question 1

Given a string `s`, find the first non-repeating character in it and return its index. If it does not exist, return `-1`.

Example 1:
Input: s = "leetcode"
Output: 0

Example 2:
Input: s = "loveleetcode"
Output: 2

Example 3:
Input: s = "aabb"
Output: -1

Solution Code:

```
package in.ineuron.pptAssignment17;

public class FirstNonRepeatingCharacter_1 {

	public static int firstUniqChar(String s) {
		int[] charCounts = new int[26];
		// Assuming only LOWERCASE English letters

		for (char c : s.toCharArray()) {
			charCounts[c - 'a']++;
		}
		for (int i = 0; i < s.length(); i++) {
			if (charCounts[s.charAt(i) - 'a'] == 1) {
				return i;
			}
		}
		return -1;
	}

	public static void main(String[] args) {
		String s = "loveleetcode"; // eg:leetcode,aabb.
		int index = firstUniqChar(s);
		System.out.println("Index of the first non-repeating character: " + index);
	}
}
```

Question 2

Given a circular integer array `nums` of length `n`, return the maximum possible sum of a non-empty subarray of `nums`.

A circular array means the end of the array connects to the beginning of the array. Formally, the next element of `nums[i]` is `nums[(i + 1) % n]` and the previous element of `nums[i]` is `nums[(i - 1 + n) % n]`.

A subarray may only include each element of the fixed buffer `nums` at most once. Formally, for a subarray `nums[i], nums[i + 1], ..., nums[j]`, there does not exist `i <= k1`, `k2 <= j` with `k1 % n == k2 % n`.

Example 1:
Input: nums = [1,-2,3,-2]
Output: 3
Explanation: Subarray [3] has maximum sum 3.

Example 2:
Input: nums = [5,-3,5]
Output: 10
Explanation: Subarray [5,5] has maximum sum 5 + 5 = 10.

Example 3:
Input: nums = [-3,-2,-3]
Output: -2
Explanation: Subarray [-2] has maximum sum -2.

Solution Code:

```
package in.ineuron.pptAssignment17;

public class MaximumCircularSubarraySum_2 {
	public static int maxSubarraySumCircular(int[] nums) {
		int totalSum = 0;
		int maxSum = Integer.MIN_VALUE;
		int minSum = Integer.MAX_VALUE;
		int currentMax = 0;
		int currentMin = 0;

		for (int num : nums) {
			totalSum += num;
			currentMax = Math.max(currentMax + num, num);
			maxSum = Math.max(maxSum, currentMax);
			currentMin = Math.min(currentMin + num, num);
			minSum = Math.min(minSum, currentMin);
		}

		if (maxSum > 0) {
			return Math.max(maxSum, totalSum - minSum);
```

```
                } else {
                        return maxSum;
                }
        }

        public static void main(String[] args) {
                int[] nums = { 5, -3, 5 };
                int maxSum = maxSubarraySumCircular(nums);
                System.out.println("Maximum circular subarray sum: " + maxSum);
        }
}
```

Question 3

The school cafeteria offers circular and square sandwiches at lunch break, referred to by numbers `0` and `1` respectively. All students stand in a queue. Each student either prefers square or circular sandwiches.

The number of sandwiches in the cafeteria is equal to the number of students. The sandwiches are placed in a stack. At each step:

- If the student at the front of the queue prefers the sandwich on the top of the stack, they will take it and leave the queue.
- Otherwise, they will leave it and go to the queue's end.

This continues until none of the queue students want to take the top sandwich and are thus unable to eat.

You are given two integer arrays `students` and `sandwiches` where `sandwiches[i]` is the type of the `ith` sandwich in the stack (`i = 0` is the top of the stack) and `students[j]` is the preference of the `jth` student in the initial queue (`j = 0` is the front of the queue). Return the number of students that are unable to eat.

Example 1:
Input: students = [1,1,0,0], sandwiches = [0,1,0,1]
Output: 0
Explanation:
- Front student leaves the top sandwich and returns to the end of the line making students = [1,0,0,1].
- Front student leaves the top sandwich and returns to the end of the line making students = [0,0,1,1].
- Front student takes the top sandwich and leaves the line making students = [0,1,1] and sandwiches = [1,0,1].
- Front student leaves the top sandwich and returns to the end of the line making students = [1,1,0].

- Front student takes the top sandwich and leaves the line making students = [1,0] and sandwiches = [0,1].
- Front student leaves the top sandwich and returns to the end of the line making students = [0,1].
- Front student takes the top sandwich and leaves the line making students = [1] and sandwiches = [1].
- Front student takes the top sandwich and leaves the line making students = [] and sandwiches = [].

Hence all students are able to eat.

Example 2:
Input: students = [1,1,1,0,0,1], sandwiches = [1,0,0,0,1,1]
Output: 3

Solution Code:

```
package in.ineuron.pptAssignment17;

public class UnableToEatSandwiches_3 {
	public static int countStudents(int[] students, int[] sandwiches) {
		int[] preferenceCount = new int[2];
			// Count of students preferring each type of sandwich

		for (int student : students) {
			preferenceCount[student]++;
		}

		for (int sandwich : sandwiches) {
			if (preferenceCount[sandwich] == 0) {
				break;
			}
			preferenceCount[sandwich]--;
		}

		return preferenceCount[0] + preferenceCount[1];
	}

	public static void main(String[] args) {
		int[] students = { 1, 1, 1, 0, 0, 1 };
		int[] sandwiches = { 1, 0, 0, 0, 1, 1 };
		int unableToEat = countStudents(students, sandwiches);
		System.out.println("Number of students unable to eat: " + unableToEat);
	}
}
```

Question 4

You have a `RecentCounter` class which counts the number of recent requests within a certain time frame.
Implement the `RecentCounter` class:
- `RecentCounter()` Initializes the counter with zero recent requests.
- `int ping(int t)` Adds a new request at time `t`, where `t` represents some time in milliseconds, and returns the number of requests that has happened in the past `3000` milliseconds (including the new request). Specifically, return the number of requests that have happened in the inclusive range `[t - 3000, t]`.
It is guaranteed that every call to `ping` uses a strictly larger value of `t` than the previous call.

Example 1:
Input
["RecentCounter", "ping", "ping", "ping", "ping"]
[[], [1], [100], [3001], [3002]]
Output
[null, 1, 2, 3, 3]

Explanation
RecentCounter recentCounter = new RecentCounter();
recentCounter.ping(1); // requests = [1], range is [-2999,1], return 1
recentCounter.ping(100); // requests = [1,100], range is [-2900,100], return 2
recentCounter.ping(3001); // requests = [1,100,3001], range is [1,3001], return 3
recentCounter.ping(3002); // requests = [1,100,3001,3002], range is [2,3002], return 3

Solution Code:

```
package in.ineuron.pptAssignment17;

import java.util.LinkedList;
import java.util.Queue;

public class RecentCounter_4 {
	private Queue<Integer> requests;

	public RecentCounter_4() {
		requests = new LinkedList<>();
	}

	public int ping(int t) {
		requests.offer(t); // Add the new request

		while (requests.peek() < t - 3000) {
			requests.poll(); // Remove requests older than t - 3000
		}

		return requests.size();
	}
```

```
        public static void main(String[] args) {
                RecentCounter_4 counter = new RecentCounter_4();
                System.out.println(counter.ping(1)); // Output: 1
                System.out.println(counter.ping(100)); // Output: 2
                System.out.println(counter.ping(3001)); // Output: 3
                System.out.println(counter.ping(3002)); // Output: 3
        }
}
```

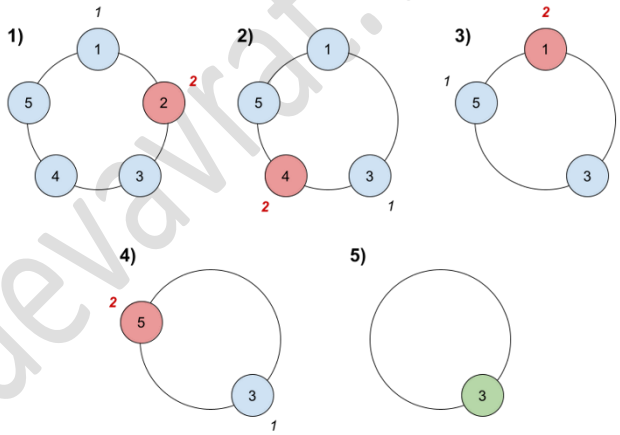
Question 5

There are `n` friends that are playing a game. The friends are sitting in a circle and are numbered from `1` to `n` in clockwise order. More formally, moving clockwise from the `ith` friend brings you to the `(i+1)th` friend for `1 <= i < n`, and moving clockwise from the `nth` friend brings you to the `1st` friend.

The rules of the game are as follows:
1. Start at the `1st` friend.
2. Count the next `k` friends in the clockwise direction including the friend you started at. The counting wraps around the circle and may count some friends more than once.
3. The last friend you counted leaves the circle and loses the game.
4. If there is still more than one friend in the circle, go back to step `2` starting from the friend immediately clockwise of the friend who just lost and repeat.
5. Else, the last friend in the circle wins the game.

Given the number of friends, `n`, and an integer `k`, return the winner of the game.

Example 1:



Input: n = 5, k = 2
Output: 3
Explanation: Here are the steps of the game:
1) Start at friend 1.
2) Count 2 friends clockwise, which are friends 1 and 2.
3) Friend 2 leaves the circle. Next start is friend 3.
4) Count 2 friends clockwise, which are friends 3 and 4.

5) Friend 4 leaves the circle. Next start is friend 5.
6) Count 2 friends clockwise, which are friends 5 and 1.
7) Friend 1 leaves the circle. Next start is friend 3.
8) Count 2 friends clockwise, which are friends 3 and 5.
9) Friend 5 leaves the circle. Only friend 3 is left, so they are the winner.

Example 2:
Input: n = 6, k = 5
Output: 1
Explanation: The friends leave in this order: 5, 4, 6, 2, 3. The winner is friend 1.

Solution Code:

```
package in.ineuron.pptAssignment17;

public class GameWinner_5 {
	public static int findWinner(int n, int k) {
		int winner = 0;
		for (int i = 1; i <= n; i++) {
			winner = (winner + k) % i;
		}
		return winner + 1;
	}

	public static void main(String[] args) {
		int n = 5;
		int k = 2;
		int winner = findWinner(n, k);
		System.out.println("Winner: " + winner);
	}
}
```

Question 6

You are given an integer array `deck`. There is a deck of cards where every card has a unique integer. The integer on the `ith` card is `deck[i]`.

You can order the deck in any order you want. Initially, all the cards start face down (unrevealed) in one deck.

You will do the following steps repeatedly until all cards are revealed:
1. Take the top card of the deck, reveal it, and take it out of the deck.
2. If there are still cards in the deck then put the next top card of the deck at the bottom of the deck.
3. If there are still unrevealed cards, go back to step 1. Otherwise, stop.

Return an ordering of the deck that would reveal the cards in increasing order.

Note that the first entry in the answer is considered to be the top of the deck.

Example 1:
Input: deck = [17,13,11,2,3,5,7]
Output: [2,13,3,11,5,17,7]
Explanation:
We get the deck in the order [17,13,11,2,3,5,7] (this order does not matter), and reorder it.
After reordering, the deck starts as [2,13,3,11,5,17,7], where 2 is the top of the deck.
We reveal 2, and move 13 to the bottom. The deck is now [3,11,5,17,7,13].
We reveal 3, and move 11 to the bottom. The deck is now [5,17,7,13,11].
We reveal 5, and move 17 to the bottom. The deck is now [7,13,11,17].
We reveal 7, and move 13 to the bottom. The deck is now [11,17,13].
We reveal 11, and move 17 to the bottom. The deck is now [13,17].
We reveal 13, and move 17 to the bottom. The deck is now [17].
We reveal 17.
Since all the cards revealed are in increasing order, the answer is correct.

Example 2:
Input: deck = [1,1000]
Output: [1,1000]

Solution Code:

```
package in.ineuron.pptAssignment17;

import java.util.Arrays;
import java.util.Deque;
import java.util.LinkedList;

public class CardRevealOrder_6 {
	public static int[] deckRevealedIncreasing(int[] deck) {
		int n = deck.length;
		Arrays.sort(deck); // Sort the deck in increasing order
```

```
                Deque<Integer> indices = new LinkedList<>();
                for (int i = 0; i < n; i++) {
                        indices.offer(i); // Add indices to a deque
                }

                int[] result = new int[n];
                for (int card : deck) {
                        result[indices.poll()] = card; // Reveal the cards at the deque indices
                        if (!indices.isEmpty()) {
                                indices.offer(indices.poll());
                                        // Move the next top index to the bottom
                        }
                }

                return result;
        }

        public static void main(String[] args) {
                int[] deck = { 17, 13, 11, 2, 3, 5, 7 };
                int[] order = deckRevealedIncreasing(deck);
                System.out.println("Revealed Card Order: " + Arrays.toString(order));
        }
}
```

Question 7

Design a queue that supports `push` and `pop` operations in the front, middle, and back. Implement the `FrontMiddleBack` class:

- `FrontMiddleBack()` Initializes the queue.
- `void pushFront(int val)` Adds `val` to the front of the queue.
- `void pushMiddle(int val)` Adds `val` to the middle of the queue.
- `void pushBack(int val)` Adds `val` to the back of the queue.
- `int popFront()` Removes the front element of the queue and returns it. If the queue is empty, return `1`.
- `int popMiddle()` Removes the middle element of the queue and returns it. If the queue is empty, return `1`.
- `int popBack()` Removes the back element of the queue and returns it. If the queue is empty, return `1`.

Notice that when there are two middle position choices, the operation is performed on the frontmost middle position choice. For example:

- Pushing `6` into the middle of `[1, 2, 3, 4, 5]` results in `[1, 2, 6, 3, 4, 5]`.
- Popping the middle from `[1, 2, 3, 4, 5, 6]` returns `3` and results in `[1, 2, 4, 5, 6]`.

Example 1:
Input:
["FrontMiddleBackQueue", "pushFront", "pushBack", "pushMiddle", "pushMiddle", "popFront", "popMiddle", "popMiddle", "popBack", "popFront"]
[[], [1], [2], [3], [4], [], [], [], [], []]
Output:
[null, null, null, null, null, 1, 3, 4, 2, -1]

Explanation:
FrontMiddleBackQueue q = new FrontMiddleBackQueue();
q.pushFront(1); // [1]
q.pushBack(2); // [1,2]
q.pushMiddle(3); // [1,3, 2]
q.pushMiddle(4); // [1,4, 3, 2]
q.popFront(); // return 1 -> [4, 3, 2]
q.popMiddle(); // return 3 -> [4, 2]
q.popMiddle(); // return 4 -> [2]
q.popBack(); // return 2 -> []
q.popFront(); // return -1 -> [] (The queue is empty)

Solution Code:

```
package in.ineuron.pptAssignment17;

public class FrontMiddleBackQueue_7 {
	private class Node {
		int val;
```

```
        Node prev;
        Node next;

        public Node(int val) {
            this.val = val;
            this.prev = null;
            this.next = null;
        }
    }

    private Node head;
    private Node tail;
    private int size;

    public FrontMiddleBackQueue_7() {
        this.head = null;
        this.tail = null;
        this.size = 0;
    }

    public void pushFront(int val) {
        Node newNode = new Node(val);
        if (isEmpty()) {
            head = newNode;
            tail = newNode;
        } else {
            newNode.next = head;
            head.prev = newNode;
            head = newNode;
        }
        size++;
    }

    public void pushMiddle(int val) {
        Node newNode = new Node(val);
        if (isEmpty()) {
            head = newNode;
            tail = newNode;
        } else if (size % 2 == 0) {
            Node middle = getNodeAtIndex(size / 2 - 1);
            newNode.prev = middle;
            newNode.next = middle.next;
            if (middle.next != null) {
                middle.next.prev = newNode;
            }
            middle.next = newNode;
        } else {
```

```
                Node middle = getNodeAtIndex(size / 2);
                newNode.prev = middle.prev;
                newNode.next = middle;
                if (middle.prev != null) {
                        middle.prev.next = newNode;
                }
                middle.prev = newNode;
        }
        size++;
}

public void pushBack(int val) {
        Node newNode = new Node(val);
        if (isEmpty()) {
                head = newNode;
                tail = newNode;
        } else {
                newNode.prev = tail;
                tail.next = newNode;
                tail = newNode;
        }
        size++;
}

public int popFront() {
        if (isEmpty()) {
                return -1;
        }

        int front = head.val;
        head = head.next;
        if (head == null) {
                tail = null;
        } else {
                head.prev = null;
        }
        size--;

        return front;
}

public int popMiddle() {
        if (isEmpty()) {
                return -1;
        }

        int middle;
```

```
            if (size % 2 == 0) {
                Node middleNode = getNodeAtIndex(size / 2 - 1);
                middle = middleNode.next.val;
                middleNode.next = middleNode.next.next;
                if (middleNode.next != null) {
                    middleNode.next.prev = middleNode;
                } else {
                    tail = middleNode;
                }
            } else {
                Node middleNode = getNodeAtIndex(size / 2);
                middle = middleNode.val;
                if (middleNode.prev != null) {
                    middleNode.prev.next = middleNode.next;
                }
                if (middleNode.next != null) {
                    middleNode.next.prev = middleNode.prev;
                }
                if (middleNode == head) {
                    head = head.next;
                }
                if (middleNode == tail) {
                    tail = tail.prev;
                }
            }
            size--;

            return middle;
        }

        public int popBack() {
            if (isEmpty()) {
                return -1;
            }

            int back = tail.val;
            tail = tail.prev;
            if (tail == null) {
                head = null;
            } else {
                tail.next = null;
            }
            size--;

            return back;
        }
```

```
    private boolean isEmpty() {
        return size == 0;
    }

    private Node getNodeAtIndex(int index) {
        if (index < 0 || index >= size) {
            return null;
        }

        Node current = head;
        for (int i = 0; i < index; i++) {
            current = current.next;
        }
        return current;
    }

    public static void main(String[] args) {
        FrontMiddleBackQueue_7 queue = new FrontMiddleBackQueue_7();
        queue.pushFront(1);
        queue.pushBack(2);
        queue.pushMiddle(3);
        queue.pushMiddle(4);

        System.out.println(queue.popFront()); // 1
        System.out.println(queue.popMiddle()); // 3
        System.out.println(queue.popMiddle()); // 4
        System.out.println(queue.popBack()); // 2
        System.out.println(queue.popFront()); // -1
    }
}
```

Question 8

For a stream of integers, implement a data structure that checks if the last `k` integers parsed in the stream are equal to `value`.
Implement the DataStream class:

- `DataStream(int value, int k)` Initializes the object with an empty integer stream and the two integers `value` and `k`.
- `boolean consec(int num)` Adds `num` to the stream of integers. Returns `true` if the last `k` integers are equal to `value`, and `false` otherwise. If there are less than `k` integers, the condition does not hold true, so returns `false`.

Example 1:
Input
["DataStream", "consec", "consec", "consec", "consec"]
[[4, 3], [4], [4], [4], [3]]
Output :
[null, false, false, true, false]

Explanation
DataStream dataStream = new DataStream(4, 3); //value = 4, k = 3
dataStream.consec(4); // Only 1 integer is parsed, so returns False.
dataStream.consec(4); // Only 2 integers are parsed.
// Since 2 is less than k, returns False.
dataStream.consec(4); // The 3 integers parsed are all equal to value, so returns True.
dataStream.consec(3); // The last k integers parsed in the stream are [4,4,3].
// Since 3 is not equal to value, it returns False.

Solution Code:

```
package in.ineuron.pptAssignment17;

import java.util.LinkedList;
import java.util.Queue;

public class DataStream_8 {
	private int value;
	private int k;
	private Queue<Integer> stream;

	public DataStream_8(int value, int k) {
		this.value = value;
		this.k = k;
		this.stream = new LinkedList<>();
	}

	public boolean consec(int num) {
		stream.offer(num);
		if (stream.size() > k) {
```

```
                stream.poll();
            }
            return isConsecutive();
        }

        private boolean isConsecutive() {
            if (stream.size() != k) {
                return false;
            }
            for (int num : stream) {
                if (num != value) {
                    return false;
                }
            }
            return true;
        }

        public static void main(String[] args) {
            DataStream_8 dataStream = new DataStream_8(4, 3);
            System.out.println(dataStream.consec(4)); // false
            System.out.println(dataStream.consec(4)); // false
            System.out.println(dataStream.consec(4)); // true
            System.out.println(dataStream.consec(3)); // false
        }
    }
```


GitHub Link : https://github.com/devavratwadekar/ineuron_ppt_ProgramAssignmentCode/tree/master/PPTAssignment-17