DAA Skill 12

NAME: K. SAI SADWIIK

ID: 2300032291

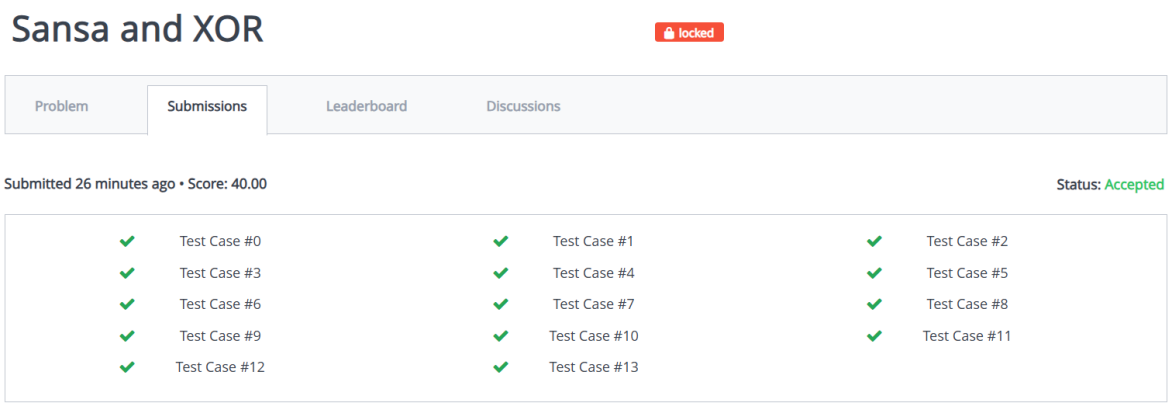
Sansa and XOR

Implementation:

```
 1. import java.io.*;
 2. import java.math.*;
 3. import java.security.*;
 4. import java.text.*;
 5. import java.util.*;
 6. import java.util.concurrent.*;
 7. import java.util.function.*;
 8. import java.util.regex.*;
 9. import java.util.stream.*;
10. import static java.util.stream.Collectors.joining;
11. import static java.util.stream.Collectors.toList;
12.
13. class Result {
14.
15.     /*
16.      * Complete the 'sansaXor' function below.
17.      *
18.      * The function is expected to return an INTEGER.
19.      * The function accepts INTEGER_ARRAY arr as parameter.
20.      */
21.
22.     public static int sansaXor(List<Integer> arr) {
23.     // Write your code here
24.         int n = arr.size();
25.         int c = 0;
26.
27.     if (n % 2 == 0) {
28.         return 0;
29.     }
30.
31.     for (int i = 0; i < n; i += 2) {
32.         c ^= arr.get(i);
33.     }
34.
35.     return c;
36.
37.     }
38.
39. }
40.
41. public class Solution {
42.     public static void main(String[] args) throws IOException {
43.         BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));
44.         BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(System.getenv("OUTPUT_PATH")));
45.
46.         int t = Integer.parseInt(bufferedReader.readLine().trim());
47.
48.         IntStream.range(0, t).forEach(tItr -> {
49.             try {
50.                 int n = Integer.parseInt(bufferedReader.readLine().trim());
51.
```

```
52.                 List<Integer> arr = Stream.of(bufferedReader.readLine().replaceAll("\\s+$",
"").split(" "))
53.                     .map(Integer::parseInt)
54.                     .collect(toList());
55.
56.                 int result = Result.sansaXor(arr);
57.
58.                 bufferedWriter.write(String.valueOf(result));
59.                 bufferedWriter.newLine();
60.             } catch (IOException ex) {
61.                 throw new RuntimeException(ex);
62.             }
63.         });
64.
65.         bufferedReader.close();
66.         bufferedWriter.close();
67.     }
68. }
69.
```

TESTCASES:

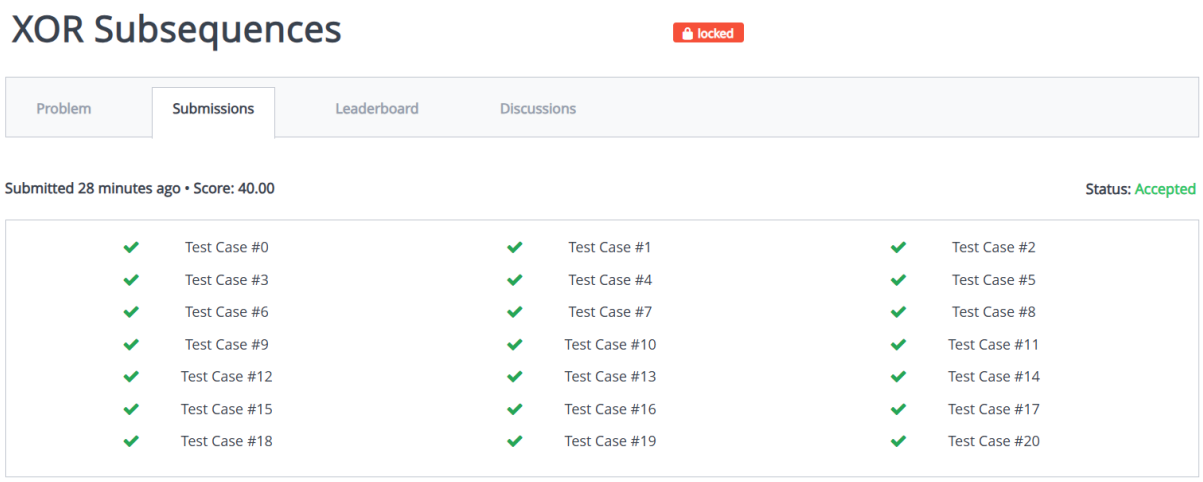


XOR Subsequences

Implementation:

```java
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

public class Solution {

    public static void main(String[] args) {
    Solution sol1 = new Solution();
        sol1.process();
    }
    public void process() {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] num = new int[n+1];
        int[] xor = new int[n+1];
        int[] counts = new int[1<<16];
        int max_count = Integer.MIN_VALUE;
        counts[0] = 1;
        for (int i = 1;i < n+1; i++) {
            num[i] = sc.nextInt();
            if (i > 0)
                xor[i] = xor[i-1] ^num[i];
            else
                xor[i] = num[i];
            counts[xor[i]] ++;
            if (xor[i] > max_count)
                max_count = xor[i];

        }
        int[] results = new int[1<<16];
        for (int i = 0;i <= max_count ; i++) {
            for (int j = i+1; j <= max_count ; j++) {
                results[i^j] += counts[i] * counts[j];
            }
        }
        int max = Integer.MIN_VALUE;
        int max_freq = Integer.MIN_VALUE;
        for (int  i =0 ;i < results.length; i++) {
            if (max_freq < results[i]) {
                max_freq = results[i];
                max = i;
            }

        }
        System.out.println(max + " " + max_freq);
    }
}

```

TESTCASES:



Maximizing XOR

Implementation:

```java
import java.io.*;
import java.math.*;
import java.security.*;
import java.text.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.function.*;
import java.util.regex.*;
import java.util.stream.*;
import static java.util.stream.Collectors.joining;
import static java.util.stream.Collectors.toList;

class Result {

    /*
     * Complete the 'maximizingXor' function below.
     *
     * The function is expected to return an INTEGER.
     * The function accepts following parameters:
     *  1. INTEGER l
     *  2. INTEGER r
     */

    public static int maximizingXor(int l, int r) {
    // Write your code here
        int result = 0;
    for(int a = l; a <= r; a++){
        for(int b = a; b<= r; b++){
            result = Math.max(result, a ^ b);
        }
    }
    return result;

    }

}

public class Solution {
    public static void main(String[] args) throws IOException {
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));
        BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(System.getenv("OUTPUT_PATH")));

        int l = Integer.parseInt(bufferedReader.readLine().trim());

        int r = Integer.parseInt(bufferedReader.readLine().trim());

        int result = Result.maximizingXor(l, r);

        bufferedWriter.write(String.valueOf(result));
        bufferedWriter.newLine();

        bufferedReader.close();
        bufferedWriter.close();
    }
}

```

TESTCASES:

Maximizing XOR

locked

Problem | Submissions | Leaderboard | Discussions

Submitted 27 minutes ago • Score: 20.00

Status: Accepted

✔ Test Case #0	✔ Test Case #1	✔ Test Case #2
✔ Test Case #3	✔ Test Case #4	✔ Test Case #5
✔ Test Case #6	✔ Test Case #7	✔ Test Case #8
✔ Test Case #9	✔ Test Case #10	✔ Test Case #11
✔ Test Case #12	✔ Test Case #13	