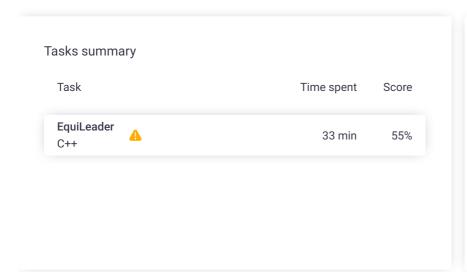
Codility_

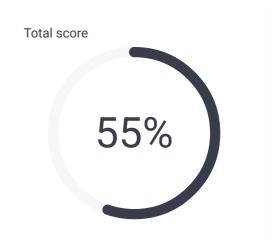
CodeCheck Report: training5NDMHJ-P23

Test Name:

Check out Codility training tasks

Summary Timeline 🛕 Al Assistant Transcript





Tasks Details

1. EquiLeader

Find the index S such that the leaders of the sequences A[0], A[1], ..., A[S] and A[S + 1], A[S + 2], ..., A[N - 1] are the same.

Task Score 55%

Correctness Performance

Task description

A non-empty array A consisting of N integers is given.

The *leader* of this array is the value that occurs in more than half of the elements of A.

An equi leader is an index S such that $0 \le S < N - 1$ and two sequences A[0], A[1], ..., A[S] and A[S + 1], A[S + 2], ..., A[N - 1] have leaders of the same value.

For example, given array A such that:

A[0] = 4

A[1] = 3

A[2] = 4

A[3] = 4

A[4] = 4

A[5] = 2

we can find two equi leaders:

- 0, because sequences: (4) and (3, 4, 4, 4, 2) have the same leader, whose value is 4.
- 2, because sequences: (4, 3, 4) and (4, 4, 2) have the same leader, whose value is 4.

Solution

Programming language used: C++

Total time used: 33 minutes

Effective time used: 33 minutes **3**

Notes: not defined yet



Code: 08:09:10 UTC, cpp,

final, score: 55

show code in pop-up

The goal is to count the number of equi leaders.

Write a function:

```
int solution(vector<int> &A);
```

that, given a non-empty array A consisting of N integers, returns the number of equi leaders.

For example, given:

A[0] = 4

A[1] = 3

A[2] = 4

A[3] = 4

A[4] = 4

A[5] = 2

the function should return 2, as explained above.

Write an efficient algorithm for the following assumptions:

- N is an integer within the range [1..100,000];
- each element of array A is an integer within the range [-1,000,000,000..1,000,000,000].

Copyright 2009–2024 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

Test results - Codility

```
// you can use includes, for example:
2
     // #include <algorithm>
3
     #include <climits>
 4
 5
     // you can write to stdout for debugging purpo
     // cout << "this is a debug message" << endl;</pre>
 6
 7
8
     int findLeader(vector<int>& A) {
9
         int leader = INT_MAX;
10
11
         int size = 0;
12
         int c;
13
         for (int a : A) {
             if (size == 0) {
14
15
                  c = a;
16
                  size = 1;
             }
17
18
             else {
                  if (c == a)
19
20
                      size++;
21
                  else
22
                      size--;
23
24
         }
25
         int count = 0;
26
27
         for (int i = 0; i < A.size(); i++) {</pre>
28
             if (A[i] == c) {
29
30
                  count++;
31
32
33
         if (count > A.size()/2)
             return c;
34
35
         return leader;
     }
36
37
38
     int solution(vector<int> &A) {
39
         // The leader of this array is the value t
40
         //n을 기준으로 0~n / n+1~end 으로 나눈 후에도 lea
41
         int leader = findLeader(A);
42
43
         int count = 0;
44
         for (int i = 0; i < A.size(); i++) {</pre>
45
             vector<int> v1(A.begin(), A.begin()+i+
46
             vector<int> v2(A.begin()+i+1, A.end())
47
             // for (auto e: v1) {cout << e << " ";
             // cout << endl;</pre>
48
49
             // for (auto e: v2) {cout << e << " ";
             // cout << endl;</pre>
50
51
             // int v1L = findLeader(v1);
52
             // int v2L = findLeader(v2);
             // cout << "leaders:" << endl;
53
             // cout << v1L <<" "<< v2L << " " << :
54
             if (findLeader(v1) != INT_MAX && findl
55
56
                  count++;
         }
57
58
59
     return count;
60
     }
```

Analysis summary

The following issues have been detected: timeout errors.

Analysis

Detected time complexity: O(N ** 2)

```
expand all Example tests
```

Test results - Codility

| length = ~50,000 6.000 sec. large | 1est 1 | results - Codinty | | | | |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|---------------------------|-------------|----------|-----------------------------|--|
| ▶ single single single element ▶ double two elements ▶ simple simple test ▶ small_random small random test with two values, length = ~100 ▶ small random test with two values, length = ~300 ▶ small random test with two values, length = ~300 expand all Performance tests ▶ large_random | • | • | | • | OK | |
| single element double two elements simple simple test small_random small random test with two values, length = ~100 small random + 200 * [MIN_INT] + random ,length = ~300 expand all Performance tests large_random large random test with two values, length = ~50,000 large random(0,1) + 50000 * [0] + random(0, 1), length = ~100,000 large_range 1, 2,, N, length = ~100,000 extreme_large x TIMEOUT ERROR Killed. Hard limit reached. 6.000 sec. x TIMEOUT ERROR Killed. Hard limit reached. 6.000 sec. x TIMEOUT ERROR Killed. Hard limit reached. 6.000 sec. x TIMEOUT ERROR Killed. Hard limit reached. 6.000 sec. | expand all Correctness te | | | sts | | |
| two elements Simple Simpl | • | • | | ~ | OK | |
| simple test ▶ small_random small random test with two values, length = ~100 ▶ small random + 200 * [MIN_INT] + random length = ~300 expand all Performance tests ▶ large_random large random test with two values, length = ~50,000 ▶ large random(0,1) + 50000 * [0] + random(0, 1), length = ~100,000 ▶ large_range 1, 2,, N, length = ~100,000 ▶ extreme_large x TIMEOUT ERROR Killed. Hard limit reached. 6.000 sec. x TIMEOUT ERROR Killed. Hard limit reached. 6.000 sec. x TIMEOUT ERROR Killed. Hard limit reached. 6.000 sec. | • | | | ~ | OK | |
| small random test with two values, length = ~100 Small random + 200 * [MIN_INT] + random ,length = ~300 expand all Performance tests Iarge_random large random test with two values, length = ~50,000 Iarge random(0,1) + 50000 * [0] + random(0, 1), length = ~100,000 Iarge_range 1, 2,, N, length = ~100,000 expand all Performance tests X TIMEOUT ERROR Killed. Hard limit reached 6.000 sec. X TIMEOUT ERROR Killed. Hard limit reached 6.000 sec. X TIMEOUT ERROR Killed. Hard limit reached 6.000 sec. X TIMEOUT ERROR X TIMEOUT ERROR Killed. Hard limit reached 6.000 sec. X TIMEOUT ERROR X TIMEOUT ERROR X TIMEOUT ERROR X TIMEOUT ERROR | • | • | | ~ | OK | |
| random + 200 * [MIN_INT] + random ,length = ~300 expand all Performance tests Parge_random large_random test with two values, length = ~50,000 Parge random(0,1) + 50000 * [0] + random(0, 1), length = ~100,000 Parge_range 1, 2,, N, length = ~100,000 Rilled. Hard limit reached 6.000 sec. TIMEOUT ERROR Killed. Hard limit reached 6.000 sec. Rilled. Hard limit reached 6.000 sec. Rilled. Hard limit reached 6.000 sec. Pextreme_large X TIMEOUT ERROR Killed. Hard limit reached 6.000 sec. | • | small random test with to | wo values, | V | OK | |
| ▶ large_random x TIMEOUT ERROR large random test with two values, Killed. Hard limit reach length = ~50,000 6.000 sec. ▶ large x TIMEOUT ERROR random(0,1) + 50000 * [0] + random(0, Killed. Hard limit reach 1), length = ~100,000 6.000 sec. ▶ large_range x TIMEOUT ERROR 1, 2,, N, length = ~100,000 Killed. Hard limit reach 6.000 sec. ▶ extreme_large x TIMEOUT ERROR | • | random + 200 * [MIN_IN] | Γ] + random | V | OK | |
| large random test with two values, length = ~50,000 6.000 sec. ▶ large random(0,1) + 50000 * [0] + random(0, 1), length = ~100,000 6.000 sec. ▶ large_range 1, 2,, N, length = ~100,000 Killed. Hard limit reached. 6.000 sec. ▶ extreme_large x TIMEOUT ERROR Killed. Hard limit reached. 6.000 sec. | expand all Performance tests | | | | | |
| random(0,1) + 50000 * [0] + random(0, 1), length = ~100,000 large_range 1, 2,, N, length = ~100,000 X TIMEOUT ERROR Killed. Hard limit reached | • | large random test with tw | | × | Killed. Hard limit reached: | |
| 1, 2,, N, length = ~100,000 Killed. Hard limit reach 6.000 sec. ▶ extreme_large | • | random(0,1) + 50000 * [0 | | × | Killed. Hard limit reached: | |
| | • | 0 0 | 000 | × | Killed. Hard limit reached: | |
| 6.000 sec. | • | • | | × | Killed. Hard limit reached: | |