Codility_

CodeCheck Report: trainingV763Y6-4NG

Test Name:

Check out Codility training tasks

Summary Timeline AI Assistant Transcript

Tasks summary

Task	Time spent	Score
MaxNonoverlappingSegments ⚠ C++	15 min	100%

Total score

100%

Tasks Details

Easy

1. MaxNonoverlappingSegments
Find a maximal set of non-overlapping segments.

Task Score	Correctness	Performance
100%	100%	100%

Task description

Located on a line are N segments, numbered from 0 to N − 1, whose positions are given in arrays A and B. For each I (0 ≤ I < N) the position of segment I is from A[I] to B[I] (inclusive). The segments are sorted by their ends, which means that B[K] ≤ B[K + 1] for K such that 0 ≤ K < N − 1.

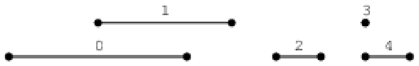
Two segments I and J, such that I ≠ J, are *overlapping* if they share at least one common point. In other words, A[I] ≤ A[J] ≤ B[I] or A[J] ≤ A[I] ≤ B[J].

We say that the set of segments is *non-overlapping* if it contains no two overlapping segments. The goal is to find the size of a non-overlapping set containing the maximal number of segments.

For example, consider arrays A, B such that:

```
    A[0] = 1    B[0] = 5
    A[1] = 3    B[1] = 6
    A[2] = 7    B[2] = 8
    A[3] = 9    B[3] = 9
    A[4] = 9    B[4] = 10
```

The segments are shown in the figure below.

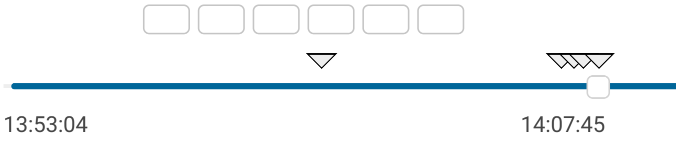


The size of a non-overlapping set containing a maximal number of segments is 3. For example, possible sets are {0, 2, 3}, {0, 2, 4}, {1, 2,

Solution

Programming language used:	C++
Total time used:	15 minutes
Effective time used:	15 minutes
Notes:	*not defined yet*

Task timeline



Code: 14:07:44 UTC, cpp, final, score: **100**

show code in pop-up

```
// you can use includes, for example:
#include <algorithm>

// you can write to stdout for debugging purpose
// cout << "this is a debug message" << endl;

```

3} or {1, 2, 4}. There is no non-overlapping set with four segments.

Write a function:

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

that, given two arrays A and B consisting of N integers, returns the size of a non-overlapping set containing a maximal number of segments.

For example, given arrays A, B shown above, the function should return 3, as explained above.

Write an **efficient** algorithm for the following assumptions:

- N is an integer within the range [0..30,000];
- each element of arrays A and B is an integer within the range [0..1,000,000,000];
- A[I] ≤ B[I], for each I (0 ≤ I < N);
- B[K] ≤ B[K + 1], for each K (0 ≤ K < N − 1).

```
int solution(vector<int> &A, vector<int> &B) {
    int N = A.size();
    if (N == 0) return 0;

    vector<pair<int, int>> v(N);

    for (int i = 0; i < A.size(); i++) {
        v[i] = make_pair(B[i], A[i]);
    }
    sort(v.begin(), v.end()); //끝나는 지점을 기준으로

    // for (auto& e:v){
    //     cout << e.first << " " << e.second<<
    // }

    int count = 1;

    int last = v[0].first;
    for (int i = 1; i < N; i++) {
        if (v[i].second > last) { //2번째 선 부터
            count++;
            last = v[i].first;
        }
    }

    return count;
}
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: **O(N)**

expand all **Example tests**

▶ example ✔ OK
example test

expand all **Correctness tests**

▶ extreme_empty_and_single ✔ OK
empty and single element

▶ small_functional ✔ OK
many overlapping

▶ small_non_overlapping ✔ OK
all non-overlapping

▶ small_all_overlapping ✔ OK
small functional

▶ small_random_same_length ✔ OK
small random, length = ~40

expand all **Performance tests**

▶ medium_random_differ_length ✔ OK
medium random, length = ~300

▶ large_points ✔ OK
all points, length = ~30,000

▶ large_random_many_overlapping ✔ OK
large random, length = ~30,000

▶ large_random_few_overlapping ✔ OK
large random, length = ~30,000

▶ extreme_large ✔ OK
large size of intervals, length = ~30,000