Codility_

CodeCheck Report: trainingBYV735-59J

Test Name:

Check out Codility training tasks

Summary Timeline AI Assistant Transcript

Tasks summary

Task	Time spent	Score
MinMaxDivision C++ ⚠	3 min	100%

Total score

100%

Tasks Details

Medium

1. MinMaxDivision
Divide array A into K blocks and minimize the largest sum of any block.

Task Score	Correctness	Performance
100%	100%	100%

Task description

You are given integers K, M and a non-empty array A consisting of N integers. Every element of the array is not greater than M.

You should divide this array into K blocks of consecutive elements. The size of the block is any integer between 0 and N. Every element of the array should belong to some block.

The sum of the block from X to Y equals A[X] + A[X + 1] + ... + A[Y]. The sum of empty block equals 0.

The *large sum* is the maximal sum of any block.

For example, you are given integers K = 3, M = 5 and array A such that:

```
  A[0] = 2
  A[1] = 1
  A[2] = 5
  A[3] = 1
  A[4] = 2
  A[5] = 2
  A[6] = 2
```

The array can be divided, for example, into the following blocks:

- [2, 1, 5, 1, 2, 2, 2], [], [] with a large sum of 15;

Solution

Programming language used:	C++
Total time used:	3 minutes
Effective time used:	3 minutes
Notes:	*not defined yet*

Task timeline

12:03:20 12:05:33

Code: 12:05:33 UTC, cpp, final, score: 100 show code in pop-up

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

- [2], [1, 5, 1, 2], [2, 2] with a large sum of 9;
- [2, 1, 5], [], [1, 2, 2, 2] with a large sum of 8;
- [2, 1], [5, 1], [2, 2, 2] with a large sum of 6.

The goal is to minimize the large sum. In the above example, 6 is the minimal large sum.

Write a function:

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

that, given integers K, M and a non-empty array A consisting of N integers, returns the minimal large sum.

For example, given K = 3, M = 5 and array A such that:

```
A[0] = 2
A[1] = 1
A[2] = 5
A[3] = 1
A[4] = 2
A[5] = 2
A[6] = 2
```

the function should return 6, as explained above.

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

- N and K are integers within the range [1..100,000];
- M is an integer within the range [0..10,000];
- each element of array A is an integer within the range [0..M].

```

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

bool division(vector<int> &A, int max, int K)
    int cur = 0;
    int count = 1;

    for (int a : A) {
        if (cur + a > max) {
            cur = a;
            count++;
            if (count > K) {
                return false;
            }
        } else {
            cur += a;
        }
    }
    return true;
}

int solution(int K, int M, vector<int> &A) {
    int lower = *max_element(A.begin(), A.end(
    int upper = accumulate(A.begin(), A.end(),

    int result = upper;

    while (lower <= upper) {
        int mid = lower + (upper - lower) / 2;
        if (division(A, mid, K)) { //합이 mid보
            result = mid; //k개로 나누기 성공했으니
            upper = mid - 1; //이진탐색
        } else { //k개보다 많게 나눠야하면
            lower = mid + 1; //이진탐색
        }
    }

    return result;
}
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: **O(N*log(N+M))**

expand all **Example tests**

▶ example ✔ OK
example test

expand all **Correctness tests**

▶ extreme_single ✔ OK
single elements

▶ extreme_double ✔ OK
single and double elements

▶ extreme_min_max ✔ OK
maximal / minimal values

▶ simple1 ✔ OK
simple tests

▶ simple2 ✔ OK
simple tests

▶ tiny_random_ones ✔ OK
random values {0, 1}, N = 100

expand all	Performance tests
▶ small_ruandom_ones random values {0, 1}, N = 100	✔ OK
▶ medium_zeros many zeros and 99 in the middle, length = 15,000	✔ OK
▶ medium_random random values {1, 100}, N = 20,000	✔ OK
▶ large_random random values {0, ..., MAX_INT}, N = 100,000	✔ OK
▶ large_random_ones random values {0, 1}, N = 100,000	✔ OK
▶ all_the_same all the same values, N = 100,000	✔ OK