Codility_

Dmitry Sermyagin

Candidate

E-mail:
dmitry_sermyagin@epam.com

Session

ID: VWKYUX-MHB
Time limit: 90 min.
Report recipients: Auto_EPM-PLX_Codility@epam.com dzmitry_kurch@epam.com
Accessed from: 37.214.33.90, 37.214.33.90
Invited by: Auto_EPM-PLX_Codility@epam.com

Status: completed

Invitation: sent
Created on: 2021-11-13 06:41 UTC
Started on: 2021-11-14 12:06 UTC
Finished on: 2021-11-14 13:25 UTC

Notes:
N/A

Similarity Check

Status: not found
No similar solutions have been detected.

Test score

100%

Tasks in test	Score	Impact
1 ArrayFilterVar (Quartz variant) Submitted in: Python	100%	Low
2 ABString Submitted in: Python	100%	Regular
3 BlocksEqualLength Submitted in: Python	100%	Regular
4 ConsecutiveDecomposition Submitted in: Python	100%	Regular
5 FinancialPlan Submitted in: Python	100%	High

Tasks Details

Elementary

1. ArrayFilterVar (Quartz variant)

Given an array of integers, filter them according to a simple condition and return an aggregate of the results.

Task Score	Correctness	Performance
100	100	Not assessed

Task description

Write a function:

```
def solution(A)
```

that, given an array A consisting of N integers, returns the maximum among all integers which are multiples of 3.

For example, given array A as follows:

```
[-6, -91, 1011, -100, 84, -22, 0, 1, 473]
```

the function should return 1011.

Assume that:

- N is an integer within the range [1..1,000];
- each element of array A is an integer within the range [−10,000..10,000];
- there is at least one element in array A which satisfies the condition in the task statement.

In your solution, focus on **correctness**. The performance of your solution will not be the focus of the assessment.

Solution

See Live Version

Programming language used:	Python
Total time used:	10 minutes
Effective time used:	10 minutes
Notes:	*not defined yet*

Source code

Code: 12:15:38 UTC, py, final, score: **200**

```python
# you can write to stdout for debugging purposes, e.g.
# print("this is a debug message")

def solution(A):
    # write your code in Python 3.6
    max_ = -10001
    for a in A:
        if a % 3 == 0 and a > max_:
            max_ = a
    return max_
```

Analysis summary

The solution obtained perfect score.

Analysis

Example tests		
	example example test	✔ OK
Correctness tests		
	short_1 one-element array	✔ OK
	short_2 two-element array	✔ OK
	all_satisfy all elements satisfy the condition	✔ OK
	one_satisfies exactly one element satisfies the condition	✔ OK
	two_satisfies exactly two element satisfies the condition	✔ OK
	nonnegative all elements are non-negative	✔ OK
	small_random small random	✔ OK
	large_random medium random	✔ OK
	include_min_max_element Include minimum and maximum value possible in array	✔ OK

Easy

2. ABString

Check, whether in a given string all letters 'a' occur before all letters 'b'.

Task Score	Correctness	Performance
100	100	100

Task description

Write a function `solution` that, given a string S consisting of N letters 'a' and/or 'b' returns `True` when all occurrences of letter 'a' are before all occurrences of letter 'b' and returns `False` otherwise.

Examples:

1. Given S = "aabbb", the function should return `True`.
2. Given S = "ba", the function should return `False`.
3. Given S = "aaa", the function should return `True`. Note that 'b' does not need to occur in S.
4. Given S = "b", the function should return `True`. Note that 'a' does not need to occur in S.
5. Given S = "abba", the function should return `False`.

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

- N is an integer within the range [1..300,000];
- string S consists only of the characters 'a' and/or 'b'.

Solution

See Live Version

Programming language used:	Python
Total time used:	16 minutes
Effective time used:	16 minutes
Notes:	*not defined yet*

Source code

Code: 12:31:20 UTC, py, final, score: **500**

```python
# you can write to stdout for debugging purposes, e.g.
# print("this is a debug message")

def solution(S):
    # write your code in Python 3.6
    seen_a = False
    for i in range(len(S)-1, -1, -1):
        if S[i] == 'a':
            seen_a = True
        if seen_a and S[i] == 'b':
            return False
    return True
```

Analysis summary

The solution obtained perfect score.

Analysis

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

Example tests	
example1 First example test.	✔ OK
example2 Second example test.	✔ OK
example3 Third example test.	✔ OK
example4 Fourth example test.	✔ OK
example5 Fifth example test.	✔ OK
Correctness tests	
one_letter Strings with only one character repeated N times.	✔ OK
first Strings that detect if solution omits first character.	✔ OK
last Strings that detect if solution omits last character.	✔ OK
true Strings with positive answer (aaa..ab..bbb).	✔ OK
swapped Strings in form 'bbb..ba..aaa'.	✔ OK
flip Strings in form 'aaa..abbb..b' with flipped letters on some substring.	✔ OK
alteration Strings where 'a's and 'b's interlace.	✔ OK
anti_random Strings in form 'aaa..abab..bbb'. It is hard to determine the answer by taking random pairs of indices.	✔ OK
tiny_difficult N <= 20. Score x 2.	✔ OK
Performance tests	
small_random Random strings. N <= 1000.	✔ OK
small_difficult N <= 1000.	✔ OK
medium_random Random strings. N <= 100'000.	✔ OK
medium_difficult N <= 100'000. Score x 2.	✔ OK
large_random Random strings. N <= 300'000.	✔ OK
big_difficult N <= 300'000. Score x 2.	✔ OK

Easy

3. BlocksEqualLength

Count the minimum number of additional letters needed to obtain a string with blocks of equal lengths.

Task Score	Correctness	Performance
100	100	100

Task description

You are given a string S consisting of letters 'a' and/or 'b'. A block is a consecutive fragment of S composed of the same letters and surrounded by different letters or string endings. For example, S = "abbabbaaa" has five blocks: "a", "bb", "a", "bb" and "aaa".

What is the minimum number of additional letters needed to obtain a string containing blocks of equal lengths? Letters can be added only at the beginning or at the end of an existing block.

Write a function:

```
def solution(S)
```

that, given a string S of length N, returns the minimum number of additional letters needed to obtain a string containing blocks of equal lengths.

Examples:

1. Given S = "babaa", the function should return 3. There are four blocks: "b", "a", "b", "aa". One letter each should be added to the first, second and third blocks, therefore obtaining a string "bbaabbaa", in which every block is of equal length.

2. Given S = "bbbab", the function should return 4. Two letters each should be added to the second and third blocks, therefore obtaining a string "bbbaaabbb", in which every block is of equal length.

3. Given S = "bbbaaabbb", the function should return 0. All blocks are already of equal lengths.

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

- N is an integer within the range [1..40,000];
- string S consists only of the characters "a" and/or "b".

Solution

See Live Version

Programming language used:	Python
Total time used:	54 minutes
Effective time used:	25 minutes
Notes:	*not defined yet*

Source code

Code: 13:25:02 UTC, py, final, score: **500**

```python
# you can write to stdout for debugging purposes, e.g.
# print("this is a debug message")
from itertools import groupby

def solution(S):
    # write your code in Python 3.6
    lengths = [len(list(g)) for k, g in groupby(S)]
    max_len = max(lengths)
    return sum(max_len - l for l in lengths)
```

Analysis summary

The solution obtained perfect score.

Analysis

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

Example tests		
	example1 First example test.	✔ OK
	example2 Second example test.	✔ OK
	example3 Third example test.	✔ OK
Correctness tests		
	small N <= 3.	✔ OK
	random_small S is randomly generated. N = 40.	✔ OK
	one_big_block_small S contains long block. N <= 80.	✔ OK
	answer_0_small Blocks have equal lengths. N <= 100.	✔ OK
Performance tests		
	one_big_block_medium S contains long block. N <= 1,000.	✔ OK
	many_occurrences_of_one_letter_medium One letter is much more common than the other. N = 1,000.	✔ OK
	one_big_block_large S contains long block. Score x 2.	✔ OK
	random_large S is randomly generated. Score x 2.	✔ OK

Easy

4. ConsecutiveDecomposition

Count the number of integers within the given interval that can be expressed as X * (X + 1).

Task Score	Correctness	Performance
100	100	100

Task description

Write a function `solution` that, given two integers A and B, returns the number of integers from the range [A..B] (ends are included) which can be expressed as the product of two consecutive integers, that is X * (X + 1), for some integer X.

Examples:

1. Given A = 6 and B = 20, the function should return 3. These integers are: 6 = 2 * 3, 12 = 3 * 4 and 20 = 4 * 5.

2. Given A = 21 and B = 29, the function should return 0. There are no integers of the form X * (X + 1) in this interval.

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

- A and B are integers within the range [1..1,000,000,000];
- A ≤ B.

Solution

See Live Version

Programming language used:	Python
Total time used:	9 minutes
Effective time used:	9 minutes
Notes:	*not defined yet*

Source code

Code: 13:01:08 UTC, py, final, score: **500**

```
# you can write to stdout for debugging purposes, e.g.
# print("this is a debug message")

def solution(A, B):
    # write your code in Python 3.6
    products = [i*(i+1) for i in range(31623)]
    numbers = [n for n in products if A<= n <= B]
    return len(numbers)
```

Analysis summary

The solution obtained perfect score.

Analysis

Detected time complexity: **O(sqrt(B)) or O(1)**

Example tests

example1 First example test.	✔ OK
example2 Second example test.	✔ OK
Correctness tests	
simple A, B <= 10.	✔ OK
n_100_corners A, B <= 100. The corners belong to the result.	✔ OK
n_2_000_corners A, B <= 2,000. The corners belong to the result.	✔ OK
n_2_000 A, B <= 2,000.	✔ OK
Performance tests	
n_1_000_000_000_interval_100 A, B <= 1,000,000,000, B - A <= 100.	✔ OK
n_1_000_000_000_interval_100_000 A, B <= 1,000,000,000, B - A <= 100,000.	✔ OK
n_1_000_000_000 A, B <= 1,000,000,000.	✔ OK
n_1_000_000_000_corners A, B <= 1,000,000,000. The corners belong to the result.	✔ OK

Easy

5. FinancialPlan

How many expenses must be rescheduled to the end of the year so that the company doesn't fall into debt?

Task Score	Correctness	Performance
100	100	100

Task description

A company has a list of expected revenues and payments for the upcoming year in chronological order. The problem is that at some moments in time the sum of previous payments can be larger than the total previous revenue, which would put the company in debt. To avoid this problem the company takes a very simple approach: it reschedules some expenses to the end of the year.

You are given an array of integers, where positive numbers represent revenues and negative numbers represent expenses, all in chronological order. In one move you can relocate any expense (negative number) to the end of the array. What is the minimum number of such relocations to make sure that the company never falls into debt (in other words: you need to ensure that there is no consecutive sequence of elements starting from the beginning of the array, that sums up to a negative number)?

You can assume that the sum of all elements in A is nonnegative.

Write a function:

```
def solution(A)
```

that, given an array A of N integers, returns the minimum number of relocations, so that company never falls into debt.

Examples:

1. Given A = [10, -10, -1, -1, 10], the function should return 1. It is enough to move -10 to the end of the array.

2. Given A = [-1, -1, -1, 1, 1, 1, 1], your function should return 3. The negative elements at the beginning must be moved to the end to avoid the debt at the start of the year.

3. Given A = [5, -2, -3, 1], the answer is 0. The company balance is always nonnegative.

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];
- sum of all elements in A is greater than or equal to 0.

Solution

See Live Version

Programming language used:	Python
Total time used:	20 minutes
Effective time used:	20 minutes
Notes:	*not defined yet*

Source code

Code: 13:21:09 UTC, py, final, score: **1000**

```python
# you can write to stdout for debugging purposes, e.g.
# print("this is a debug message")
from heapq import heappop, heappush

def solution(A):
    # write your code in Python 3.6
    sum_ = 0
    heap = []
    cnt = 0
    for n in A:
```

```
11        if n < 0:
12            heappush(heap, n)
13        while sum_ + n < 0:
14            cnt += 1
15            neg = heappop(heap)
16            sum_ -= neg
17        sum_ += n
18    return cnt
```

Analysis summary

The solution obtained perfect score.

Analysis

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

Example tests	
example1 First example test.	✔ OK
example2 Second example test.	✔ OK
example3 Third example test.	✔ OK
Correctness tests	
small_corner_cases Small corner cases.	✔ OK
almost_all_prefix_sums_negative All prefix sums except the entire array are negative.	✔ OK
negative_in_the_middle Many negative values in the middle of the array.	✔ OK
negative_nonnegative Negative elements followed by nonnegative elements.	✔ OK
random Random arrays.	✔ OK
one_positive All elements but one are negative. N ~ 100,000.	✔ OK
Correctness/performance tests	
negative_in_the_middle_large Many negative values in the middle of the array. N ~ 100,000.	✔ OK
Performance tests	
cyclic_large Large tests with repeating sequences. N ~ 100,000.	✔ OK
one_positive_large All elements but one are negative. N ~ 100,000.	✔ OK