0h 37min

Submi

Task 1 | Java 8 | English

Find the bug(s) and modify **one** line of code in the incorrect implementation of a function `solution` that is supposed to return the smallest element of the given non-empty array `A` which contains at most `1000` integers within range `[-1000..1000]`.

Notice that for the example test case `A = [-1, 1, -2, 2]` the attached code is already returning the correct answer (-2).

Files

- task1
  - solution.java
  - test-input.txt

solution.java

```
import java.util.*;

class Solution {
    int solution(int[] A) {
        int ans = 0;
        int ans = A[0];
        for (int i = 1; i < A.length; i++) {
            if (ans > A[i]) {
                ans = A[i];
            }
        }
        return ans;
    }
}
```

Test Output

Run

```
Compilation successful.

Example test:   [-1, 1, -2, 2]
OK
```

Your code is syntactically correct and works properly on the example test.

Note that the example tests are not part of your score. On submission at least 8 test cases not shown here will assess your solution.

All changes saved

Any problems with the editor? Switch to basic e

0h 36min

Submit Task

Task 2

Java 8

Write a function `solution` that, given an array A of N integers (between −100 and 100), returns the sign (−1, 0, 1) of the product of all the numbers in the array multiplied together. Assume that N is between 1 and 1000.

For example, given A = [1, −2, −3, 5], the function should return 1 (the multiplication equals 30).
Given A = [1, 2, 3, −5] your function should return −1 (the multiplication equals −30).
Given A = [1, 2, 0, −5] your function should return 0 (the multiplication equals 0).

Files

- task2
  - solution.java
  - test-input.txt

solution.java

```java
// you can also use imports, for example:
// import java.util.*;

// you can write to stdout for debugging purposes, e.g.
// System.out.println("this is a debug message");

class Solution {
    public int solution(int[] A) {
        // write your code in Java SE 8
        int res = 0;

        for (int num: A) {
            if (num == 0) return 0;

            if (num < 0) res += 1;
        }

        return res % 2== 0? 1: -1;
    }
}
```

Test Output

Run Tests

D: 6

2. You are given an array with 10^8 elements and you must compute the sum A[J] + A[K] for every possible pair of elements in the array. How long will this computation take on a single mainstream computer?

- A: milliseconds
- B: seconds
- C: minutes
- D: hours
- E: more than a few hours

3. How do you check whether a number is odd in most programming languages?

- A: if (x) ...
- B: if ((x & 1) != 0) ...
- C: if (x % 2 == 0) ...
- D: if (x >> 1 == 0) ...
- E: none of the above

4. Given a sorted array, which of the following operations can be performed the fastest?

- A: checking whether the number 42 is in the array
- B: counting the number of different elements
- C: computing the sum of all elements
- D: all of the above operations require linear time, so they are approximately the same

5. A common data structure used to implement a relational database is a

- A: queue
- B: stack
- C: priority queue
- D: B-tree

6. Given a list of 1000 elements, which of the following numbers is the largest?

- A: The number of all possible subsets of a given list

Task 3

4. Given a sorted array, which of the following operations can be performed the fastest?

- (●) A: checking whether the number 42 is in the array
- ( ) B: counting the number of different elements
- ( ) C: computing the sum of all elements
- ( ) D: all of the above operations require linear time, so they are approximately the same

5. A common data structure used to implement a relational database is a

- ( ) A: queue
- ( ) B: stack
- ( ) C: priority queue
- (●) D: B-tree

6. Given a list of 1000 elements, which of the following numbers is the largest?

- ( ) A: The number of all possible subsets of a given list
- ( ) B: The number of all possible pairs of elements from a given list
- (●) C: The number of all possible permutations of a given list
- ( ) D: 10,000,000,000,000,000

7. Given a tree with four nodes, how many edges have to be added for it to cease being a tree?

- (●) A: 1
- ( ) B: 2
- ( ) C: 3
- ( ) D: 4

8. Which of the following statements is correct (choosing one answer means that single answer is correct):

```
def f(A):
    for i := 0; i < A.length - 1; i++ do:
        if (A[i] > A[i + 1]):
            swap A[i] with A[i+1]
    return A
```

Task 4 | Java 8 | English

An empty tree is represented by an empty pointer (denoted by `null`). A non-empty tree is represented by a pointer to an object representing its root. The attribute `x` holds the integer contained in the root, whereas attributes `l` and `r` hold the left and right subtrees of the binary tree, respectively.

A *path* in a binary tree is a non-empty sequence of nodes that one can traverse by following the pointers. The *length* of a path is the number of pointers it traverses. More formally, a path of length K is a sequence of nodes P[0], P[1], ..., P[K], such that node P[I + 1] is the root of the left or right subtree of P[I], for 0 ≤ I < K. For example, the sequence of nodes with values 5, 3, 21 is a path of length 2 in the tree from the above figure. The sequence of nodes with values 10, 1 is a path of length 1. The sequence of nodes with values 20, 3, 21 is not a valid path.

The *height* of a binary tree is defined as the length of the longest possible path in the tree. In particular, a tree consisting of only one node has height 0 and, conventionally, an empty tree has height −1. For example, the tree shown in the above figure is of height 2.

A binary tree T is given. A node of tree T containing value V is described as *visible* if the path from the root of the tree to that node does not contain a node with any value exceeding V. In particular, the root is always visible and nodes with values lower than that of the root are never visible.

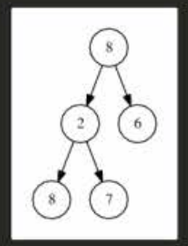
For example, the tree shown in the above figure has four visible nodes: namely, those with values 5, 10, 20 and 21. The node with value 1 is not visible because there is a node with value 10 on the path from the root to that node. The node with value 3 is not visible because its value is lower than that of the root, which has value 5.

Write a function:

```
class Solution { public int solution(Tree T); }
```

that, given a binary tree T consisting of N nodes, returns its number of visible nodes. For example, given the tree shown in the figure above, the function should return 4, as explained above.

Given tree T with the following structure:



the function should return 2, because the only visible nodes are those with value 8.

For the purpose of entering your own test cases, you can denote a tree recursively in the following way. An empty binary tree is denoted by `None`. A non-empty tree is denoted as (X, L, R), where X is the value contained in the root and L and R denote the left and right subtrees, respectively. The trees from the above two figures can be denoted as:

```
(5, (3, (20, None, None), (21, None, None)), (10, (1, None, None), None))
```

and:

```
(8, (2, (8, None, None), (7, None, None)), (6, None, None))
```

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

- N is an integer within the range [0..50,000];
- each value in tree T is an integer within the range [−100,000..100,000];
- the height of tree T (number of edges on the longest path from root to leaf) is within the range [−1..500].

Files

- task4
  - Tests data
    - test-input-01.txt
  - solution.java

solution.java

```java
// you can also use imports, for example:
// import java.util.*;

// you can write to stdout for debugging purposes, e.g.
// System.out.println("this is a debug message");

class Solution {
    public int solution(Tree T) {
        // write your code in Java SE 8

    }
}
```

Test Output

Autocomplete is connected | All changes saved

Any problems with the editor? Switch t

在这里输入你要搜索的内容

0h 38min

Task 4 | Java 8 | English

An empty tree is represented by an empty pointer (denoted by `null`). A non-empty tree is represented by a pointer to an object representing its root. The attribute `x` holds the integer contained in the root, whereas attributes `l` and `r` hold the left and right subtrees of the binary tree, respectively.

A *path* in a binary tree is a non-empty sequence of nodes that one can traverse by following the pointers. The *length* of a path is the number of pointers it traverses. More formally, a path of length K is a sequence of nodes P[0], P[1], ..., P[K], such that node P[I + 1] is the root of the left or right subtree of P[I], for 0 ≤ I < K. For example, the sequence of nodes with values 5, 3, 21 is a path of length 2 in the tree from the above figure. The sequence of nodes with values 10, 1 is a path of length 1. The sequence of nodes with values 20, 3, 21 is not a valid path.

The *height* of a binary tree is defined as the length of the longest possible path in the tree. In particular, a tree consisting of only one node has height 0 and, conventionally, an empty tree has height −1. For example, the tree shown in the above figure is of height 2.

A binary tree T is given. A node of tree T containing value V is described as *visible* if the path from the root of the tree to that node does not contain a node with any value exceeding V. In particular, the root is always visible and nodes with values lower than that of the root are never visible.

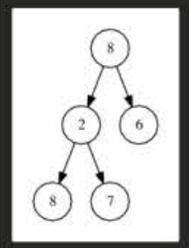
For example, the tree shown in the above figure has four visible nodes: namely, those with values 5, 10, 20 and 21. The node with value 1 is not visible because there is a node with value 10 on the path from the root to that node. The node with value 3 is not visible because its value is lower than that of the root, which has value 5.

Write a function:

```
class Solution { public int solution(Tree T); }
```

that, given a binary tree T consisting of N nodes, returns its number of visible nodes. For example, given the tree shown in the figure above, the function should return 4, as explained above.

Given tree T with the following structure:



the function should return 2, because the only visible nodes are those with value 8.

For the purpose of entering your own test cases, you can denote a tree recursively in the following way. An empty binary tree is denoted by `None`. A non-empty tree is denoted as (X, L, R), where X is the value contained in the root and L and R denote the left and right subtrees, respectively. The trees from the above two figures can be denoted as:

```
(5, (3, (20, None, None), (21, None, None)), (10, (1, None, None), None))
```

and:

```
(8, (2, (8, None, None), (7, None, None)), (6, None, None))
```

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

- N is an integer within the range [0..50,000];
- each value in tree T is an integer within the range [−100,000..100,000];
- the height of tree T (number of edges on the longest path from root to leaf) is within the range [−1..500].

Files
- task4
  - Tests data
    - test-input-01.txt
  - solution.java

solution.java

```java
// you can also use imports, for example:
// import java.util.*;

// you can write to stdout for debugging purposes, e.g.
// System.out.println("this is a debug message");

class Solution {
    int ret = 0;
    public int solution(Tree T) {
        // write your code in Java SE 8
        dfs(T, Integer.MIN_VALUE);
        return ret;
    }

    private void dfs(Tree node, int curMax) {
        if (node == null) return;
        if (curMax <= node.x) {
            ret ++;
            curMax = node.x;
        }

        dfs(node.l, curMax);
        dfs(node.r, curMax);
    }
}
```

Test Output

```
Compilation successful.

Example test:   (5, (3, (20, None, None), (21, None, None)), (10, (1, None, None), None))
OK

Example test:   (8, (2, (8, None, None), (7, None, None)), (6, None, None))
OK

Your test case: (5, (3, (20, None, None), (21, None, None)), (10, (1, None, None), None))
Returned value: 4
```

Your code is syntactically correct and works properly on the example test.
Note that the example tests are not part of your score. On submission at least 8 test cases not shown here will assess your solution.