you can write to stdout for debugging purposes, e.g.

// printf(this is a debug message\n);

int solution(char \*S) {

// write your code in C99 (gcc 4.8.2)

}// you can write to stdout for debugging purposes, e.g.

// printf(this is a debug message\n);

int solution(char \*S) {

// write your code in C99 (gcc 4.8.2)

}

You would like to set a password for an email account. However, there are two restrictions on the format of the password. It has to contain at least one uppercase character and it cannot contain any digits.

You are given a string S consisting of N alphanumerical characters. You would like to find the longest substring of S that is a valid password. A substring is defined as a contiguous segment of a string.

For example, given “a0Ba”, the substrings that are valid passwords are “B” and “Ba”. Note that “aBa” is not a substring and “a0B” is not a valid password.

Write a function:

int solution(char \*S);

that, given a non-empty string S consisting of N characters, returns the length of its longest substring that is a valid password. If there is no such substring, your function shod return −1.

For example, given “a0Ba”, your function should return 2, as explained above. Given “a0bb”, your function shod return −1, since there is no substring that satisfies the restrictions on the format of a valid password.

Assume that:

N is an integer within the range [1-200];

string S consists only of alphanumerical characters (a-z and/or A-Z and/or 0-9).

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

you can write to stdout for debugging purposes, e.g.

// printf(this is a debug message\n);

int solution(struct tree \* T) {

// write your code in C99 (gcc 4.8.2)

}// you can write to stdout for debugging purposes, e.g.

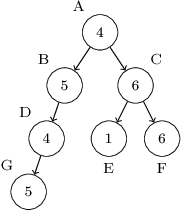
// printf(this is a debug message\n);

int solution(struct tree \* T) {

// write your code in C99 (gcc 4.8.2)

}

In this problem we consider binary trees. The figure below shows an example binary tree consisting of seven nodes.

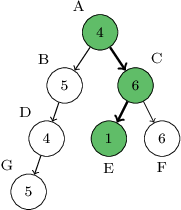


pimg src=https://codility-frontend-prod.s3.amazonaws.com/media/task\_img/tree\_most\_distinct\_path/media/auto/tikzb16ee9a593f3a1a989dc7d38fb205a0e.png /

A binary tree is either an empty tree or a node (called the root) containing a single integer value and linked to two further binary trees. We are interested in paths (sequences of linked adjacent nodes) that start at the root and follow the tree edges (marked as arrows in the figure above). For example, the sequence of nodes A, B, D is a valid path, but the sequence A, B, G is not.

Problem

We would like to find the maximum number of distinct values that appear on a path starting at the root of the tree. For example, on the path consisting of nodes A, B, D, G there are two distinct values (4 and 5). On the path A, C, E there are three distinct values (1, 4 and 6). There is no path that contains four or more distinct values.



pimg src=https://codility-frontend-prod.s3.amazonaws.com/media/task\_img/tree\_most\_distinct\_path/media/auto/tikz3e7e23ff482d5c78f41af53bec605035.png /

Write a function:

int solution(struct tree \* T);

that, given a binary tree T consisting of N nodes, returns the maximum number of distinct values that appear on a path starting at the root of tree T. For example, given the tree shown above, the function should return 3.

Technical details

A binary tree is given using a pointer data structure. Assume that the following declarations are given:

struct tree {

int x;

struct tree \* l;

struct tree \* r;

};

An empty tree is represented by an empty pointer (denoted by pre-wrapNL/tt). 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.

Assumptions

Assume that:

N is an integer within the range [1-50,000];i

the height of tree T (number of edges on the longest path from root to leaf) is within the range [0.. 3,500];

each value in tree T is an integer within the range [1.. N].

Complexity:

expected worst-case time complexity is O(N);

expected worst-case space complexity is O(N).

you can write to stdout for debugging purposes, e.g.

// printf(this is a debug message\n);

int solution(int A[], int N) {

// write your code in C99 (gcc 4.8.2)

}// you can write to stdout for debugging purposes, e.g.

// printf(this is a debug message\n);

int solution(int A[], int N) {

// write your code in C99 (gcc 4.8.2)

}

A non-empty zero-indexed array A consisting of N integers is given. The array is sorted in ascending order and it does not contain duplicate values. The array describes a number K as follows:

K = pow2(A[0]) + pow2(A[1]) + ... + pow2(A[N−1])

pow2(L) = 2L

For example, consider array A consisting of three elements such that

A[0] = 1 A[1] = 4 A[2] = 5

It describes the number K = 50, because

pow2(A[0]) + pow2(A[1]) + pow2(A[2]) =

pow2(1) + pow2(4) + pow2(5) =

2 + 16 + 32 = 50

Write a function

int solution(int A[], int N);

that, given a non-empty zero-indexed array A consisting of N non-negative integers, returns the number of bits set to 1 in the binary representation of the number 3\*K, where K is the number described by array A.

Assume that:

N is an integer within the range [1.. 10,000];

each element of array A is an integer within the range [0.. 1,000,000,000];

array A is sorted in non-decreasing order;

array A does not contain duplicates.

For example, given array A consisting of three elements such that

A[0] = 1 A[1] = 4 A[2] = 5

the function shod return 4 because:

array A represents number K = 50, as explained above;

3 \* K = 3 \* 50 = 150;

the binary representation of 150 is 10010110

it contains four bits set to 1.

Complexity:

expected worst-case time complexity is O(N);

expected worst-case space complexity is O(1),

beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

write your code in PostgreSQL 9.4

SELECT ...-- write your code in PostgreSQL 9.4

SELECT ...

Given a table invoice\_items with the following structure:

create table invoice\_items (

inv\_num integer not null,

item varchar(10) not null,

price integer not null

);

pwrite an SQL query that returns a list of invoices (inv\_num) with the total price of each. The order of the invoices in the list does not matter.

For example, given:

inv\_num | item | price

--------+------+------

3 | a | 10

3 | b | 15

1 | c | 7

your query shod return:

inv\_num | sum

--------+----

1 | 7

3 | 25