Problem 2307: Check for Contradictions in Equations

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given a 2D array of strings

equations

and an array of real numbers

values

, where

equations[i] = [A

i

, B

i

]

and

values[i]

means that

A

i

/ B

i

= values[i]

.

Determine if there exists a contradiction in the equations. Return

true

if there is a contradiction, or

false

otherwise

.

Note

:

When checking if two numbers are equal, check that their

absolute difference

is less than

10

-5

.

The testcases are generated such that there are no cases targeting precision, i.e. using

double

is enough to solve the problem.

Example 1:

Input:

equations = [["a","b"],["b","c"],["a","c"]], values = [3,0.5,1.5]

Output:

false

Explanation:

The given equations are: a / b = 3, b / c = 0.5, a / c = 1.5 There are no contradictions in the equations. One possible assignment to satisfy all equations is: a = 3, b = 1 and c = 2.

Example 2:

Input:

equations = [["le","et"],["le","code"],["code","et"]], values = [2,5,0.5]

Output:

true

Explanation:

The given equations are: le / et = 2, le / code = 5, code / et = 0.5 Based on the first two equations, we get code / et = 0.4. Since the third equation is code / et = 0.5, we get a contradiction.

Constraints:

1 <= equations.length <= 100

equations[i].length == 2

1 <= A

i

.length, B

i

.length <= 5

A

i

,

B

i

consist of lowercase English letters.

equations.length == values.length

0.0 < values[i] <= 10.0

values[i]

has a maximum of 2 decimal places.

Code Snippets

C++:

```
class Solution {
public:
bool checkContradictions(vector<vector<string>>& equations, vector<double>&
values) {

}
};
```

Java:

```
class Solution {
public boolean checkContradictions(List<List<String>> equations, double[]
values) {

}
}
```

Python3:

```
class Solution:
def checkContradictions(self, equations: List[List[str]], values:
List[float]) -> bool:
```

Python:

```
class Solution(object):
def checkContradictions(self, equations, values):
"""
:type equations: List[List[str]]
:type values: List[float]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string[][]} equations
* @param {number[]} values
* @return {boolean}
*/
var checkContradictions = function(equations, values) {
};
```

TypeScript:

```
function checkContradictions(equations: string[][], values: number[]):
boolean {

};
```

C#:

```
public class Solution {
public bool CheckContradictions(IList<IList<string>> equations, double[]
values) {

}
}
```

C:

```
bool checkContradictions(char*** equations, int equationsSize, int*
equationsColSize, double* values, int valuesSize) {

}
```

Go:

```
func checkContradictions(equations [][]string, values []float64) bool {

}
```

Kotlin:

```
class Solution {
fun checkContradictions(equations: List<List<String>>, values: DoubleArray):
Boolean {

}
}
```

Swift:

```
class Solution {
func checkContradictions(_ equations: [[String]], _ values: [Double]) -> Bool
{

}
}
```

Rust:

```
impl Solution {
pub fn check_contradictions(equations: Vec<Vec<String>>, values: Vec<f64>) ->
bool {

}
}
```

Ruby:

```
# @param {String[][]} equations
# @param {Float[]} values
# @return {Boolean}
def check_contradictions(equations, values)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $equations
* @param Float[] $values
* @return Boolean
*/
function checkContradictions($equations, $values) {

}
}
```

Dart:

```
class Solution {
bool checkContradictions(List<List<String>> equations, List<double> values) {



}
}
```

Scala:

```
object Solution {
def checkContradictions(equations: List[List[String]], values:
Array[Double]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec check_contradictions(equations :: [[String.t]], values :: [float]) ::
boolean
def check_contradictions(equations, values) do

end
end
```

Erlang:

```
-spec check_contradictions(Equations :: [[unicode:unicode_binary()]], Values
:: [float()]) -> boolean().
check_contradictions(Equations, Values) ->
.
```

Racket:

```
(define/contract (check-contradictions equations values)
(-> (listof (listof string?)) (listof flonum?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check for Contradictions in Equations
* Difficulty: Hard
* Tags: array, string, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
bool checkContradictions(vector<vector<string>>& equations, vector<double>&
values) {

}
};
```

Java Solution:

```
/**
* Problem: Check for Contradictions in Equations
* Difficulty: Hard
* Tags: array, string, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public boolean checkContradictions(List<List<String>> equations, double[]
values) {

}
}
```

Python3 Solution:

```
"""
Problem: Check for Contradictions in Equations
Difficulty: Hard
```

```
Tags: array, string, graph, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def checkContradictions(self, equations: List[List[str]], values:
List[float]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def checkContradictions(self, equations, values):
"""
:type equations: List[List[str]]
:type values: List[float]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Check for Contradictions in Equations
* Difficulty: Hard
* Tags: array, string, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string[][]} equations
* @param {number[]} values
* @return {boolean}
*/
var checkContradictions = function(equations, values) {
};
```

TypeScript Solution:

```
/**
* Problem: Check for Contradictions in Equations
* Difficulty: Hard
* Tags: array, string, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function checkContradictions(equations: string[][], values: number[]):
boolean {

};
```

C# Solution:

```
/*
* Problem: Check for Contradictions in Equations
* Difficulty: Hard
* Tags: array, string, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool CheckContradictions(IList<IList<string>> equations, double[]
values) {

}
}
```

C Solution:

```
/*
* Problem: Check for Contradictions in Equations
* Difficulty: Hard
* Tags: array, string, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

bool checkContradictions(char*** equations, int equationsSize, int*
equationsColSize, double* values, int valuesSize) {

}
```

Go Solution:

```
// Problem: Check for Contradictions in Equations
// Difficulty: Hard
// Tags: array, string, graph, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func checkContradictions(equations [][]string, values []float64) bool {

}
```

Kotlin Solution:

```
class Solution {
fun checkContradictions(equations: List<List<String>>, values: DoubleArray):
Boolean {

}
}
```

Swift Solution:

```
class Solution {
func checkContradictions(_ equations: [[String]], _ values: [Double]) -> Bool
{

}
}
```

Rust Solution:

```
// Problem: Check for Contradictions in Equations
// Difficulty: Hard
// Tags: array, string, graph, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn check_contradictions(equations: Vec<Vec<String>>, values: Vec<f64>) ->
bool {

}
}
```

Ruby Solution:

```
# @param {String[][]} equations
# @param {Float[]} values
# @return {Boolean}
def check_contradictions(equations, values)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $equations
* @param Float[] $values
* @return Boolean
*/
function checkContradictions($equations, $values) {
}
}
```

Dart Solution:

```
class Solution {
bool checkContradictions(List<List<String>> equations, List<double> values) {

}
}
```

Scala Solution:

```
object Solution {
def checkContradictions(equations: List[List[String]], values:
Array[Double]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec check_contradictions(equations :: [[String.t]], values :: [float]) ::
boolean
def check_contradictions(equations, values) do

end
end
```

Erlang Solution:

```
-spec check_contradictions(Equations :: [[unicode:unicode_binary()]], Values
:: [float()]) -> boolean().
check_contradictions(Equations, Values) ->
.
```

Racket Solution:

```
(define/contract (check-contradictions equations values)
(-> (listof (listof string?)) (listof flonum?) boolean?)
)
```