# Problem 399: Evaluate Division

## Problem Information

**Difficulty:** Medium
**Acceptance Rate:** 0.00%
**Paid Only:** No

## Problem Description

You are given an array of variable pairs

equations

and an array of real numbers

values

, where

equations[i] = [A

i

, B

i

]

and

values[i]

represent the equation

A

i

/ B

i

= values[i]

. Each

A

i

or

B

i

is a string that represents a single variable.

You are also given some

queries

, where

queries[j] = [C

j

, D

j

]

represents the

j

th

query where you must find the answer for

C

j

/ D

j

= ?

.

Return

the answers to all queries

. If a single answer cannot be determined, return

-1.0

.

Note:

The input is always valid. You may assume that evaluating the queries will not result in division by zero and that there is no contradiction.

Note:

The variables that do not occur in the list of equations are undefined, so the answer cannot be determined for them.

Example 1:

Input:

equations = [["a","b"],["b","c"]], values = [2.0,3.0], queries = [["a","c"],["b","a"],["a","e"],["a","a"],["x","x"]]

Output:

[6.00000,0.50000,-1.00000,1.00000,-1.00000]

Explanation:

Given:

a / b = 2.0

,

b / c = 3.0

queries are:

a / c = ?

,

b / a = ?

,

a / e = ?

,

a / a = ?

,

x / x = ?

return: [6.0, 0.5, -1.0, 1.0, -1.0 ] note: x is undefined => -1.0

Example 2:

Input:

equations = [["a","b"],["b","c"],["bc","cd"]], values = [1.5,2.5,5.0], queries = [["a","c"],["c","b"],["bc","cd"],["cd","bc"]]

Output:

[3.75000,0.40000,5.00000,0.20000]

Example 3:

Input:

equations = [["a","b"]], values = [0.5], queries = [["a","b"],["b","a"],["a","c"],["x","y"]]

Output:

[0.50000,2.00000,-1.00000,-1.00000]

Constraints:

1 <= equations.length <= 20

equations[i].length == 2

1 <= A

i

.length, B

i

.length <= 5

values.length == equations.length

0.0 < values[i] <= 20.0

1 <= queries.length <= 20

queries[i].length == 2

1 <= C

j

.length, D

j

.length <= 5

A

i

, B

i

, C

j

, D

j

consist of lower case English letters and digits.

## Code Snippets

**C++:**

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

}
};
```

**Java:**

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

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

```
/**
* @param {string[][]} equations
```

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

};
```

**TypeScript:**

```
function calcEquation(equations: string[][], values: number[], queries:
string[][]): number[] {

};
```

**C#:**

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

}
}
```

**C:**

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
double* calcEquation(char*** equations, int equationsSize, int*
equationsColSize, double* values, int valuesSize, char*** queries, int
queriesSize, int* queriesColSize, int* returnSize) {

}
```

**Go:**

```
func calcEquation(equations [][]string, values []float64, queries [][]string)
[]float64 {

}
```

**Kotlin:**

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

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

/**
* @param String[][] $equations
```

```
* @param Float[] $values
* @param String[][] $queries
* @return Float[]
*/
function calcEquation($equations, $values, $queries) {

}
}
```

**Dart:**

```
class Solution {
List<double> calcEquation(List<List<String>> equations, List<double> values,
List<List<String>> queries) {

}
}
```

**Scala:**

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

}
}
```

**Elixir:**

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

end
end
```

**Erlang:**

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

```
.
```

**Racket:**

```
(define/contract (calc-equation equations values queries)
(-> (listof (listof string?)) (listof flonum?) (listof (listof string?))
(listof flonum?))
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Evaluate Division
* Difficulty: Medium
* 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:
vector<double> calcEquation(vector<vector<string>>& equations,
vector<double>& values, vector<vector<string>>& queries) {

}
};
```

### Java Solution:

```
/**
* Problem: Evaluate Division
* Difficulty: Medium
* 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 double[] calcEquation(List<List<String>> equations, double[] values,
List<List<String>> queries) {

}
}
```

**Python3 Solution:**

```
"""
Problem: Evaluate Division
Difficulty: Medium
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 calcEquation(self, equations: List[List[str]], values: List[float],
queries: List[List[str]]) -> List[float]:
# TODO: Implement optimized solution
pass
```

**Python Solution:**

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

**JavaScript Solution:**

```
/**
* Problem: Evaluate Division
* Difficulty: Medium
* 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
* @param {string[][]} queries
* @return {number[]}
*/
var calcEquation = function(equations, values, queries) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Evaluate Division
* Difficulty: Medium
* 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 calcEquation(equations: string[][], values: number[], queries:
string[][]): number[] {

};
```

**C# Solution:**

```
/*
* Problem: Evaluate Division
* Difficulty: Medium
```

```
* 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 double[] CalcEquation(IList<IList<string>> equations, double[] values,
IList<IList<string>> queries) {

}
}
```

### C Solution:

```
/*
* Problem: Evaluate Division
* Difficulty: Medium
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
double* calcEquation(char*** equations, int equationsSize, int*
equationsColSize, double* values, int valuesSize, char*** queries, int
queriesSize, int* queriesColSize, int* returnSize) {

}
```

### Go Solution:

```
// Problem: Evaluate Division
// Difficulty: Medium
// 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 calcEquation(equations [][]string, values []float64, queries [][]string)
[]float64 {

}
```

**Kotlin Solution:**

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

}
}
```

**Swift Solution:**

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

}
}
```

**Rust Solution:**

```
// Problem: Evaluate Division
// Difficulty: Medium
// 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 calc_equation(equations: Vec<Vec<String>>, values: Vec<f64>, queries:
Vec<Vec<String>>) -> Vec<f64> {
```

```
}
}
```

**Ruby Solution:**

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

end
```

**PHP Solution:**

```
class Solution {

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

}
}
```

**Dart Solution:**

```
class Solution {
List<double> calcEquation(List<List<String>> equations, List<double> values,
List<List<String>> queries) {

}
}
```

**Scala Solution:**

```
object Solution {
def calcEquation(equations: List[List[String]], values: Array[Double],
```

```
queries: List[List[String]]): Array[Double] = {

}
}
```

### Elixir Solution:

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

end
end
```

### Erlang Solution:

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

### Racket Solution:

```
(define/contract (calc-equation equations values queries)
(-> (listof (listof string?)) (listof flonum?) (listof (listof string?))
(listof flonum?))
)
```