Problem 1237: Find Positive Integer Solution for a Given Equation

Problem Information

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

Problem Description

Given a callable function

f(x, y)

with a hidden formula

and a value

z

, reverse engineer the formula and return

all positive integer pairs

x

and

y

where

f(x,y) == z

. You may return the pairs in any order.

While the exact formula is hidden, the function is monotonically increasing, i.e.:

f(x, y) < f(x + 1, y)

f(x, y) < f(x, y + 1)

The function interface is defined like this:

interface CustomFunction { public: // Returns some positive integer f(x, y) for two positive integers x and y based on a formula. int f(int x, int y); };

We will judge your solution as follows:

The judge has a list of

9

hidden implementations of

CustomFunction

, along with a way to generate an

answer key

of all valid pairs for a specific

z

.

The judge will receive two inputs: a

function_id

(to determine which implementation to test your code with), and the target

z

.

The judge will call your

findSolution

and compare your results with the

answer key

.

If your results match the

answer key

, your solution will be

Accepted

.

Example 1:

Input:

function_id = 1, z = 5

Output:

[[1,4],[2,3],[3,2],[4,1]]

Explanation:

The hidden formula for function_id = 1 is f(x, y) = x + y. The following positive integer values of x and y make f(x, y) equal to 5: x=1, y=4 -> f(1, 4) = 1 + 4 = 5. x=2, y=3 -> f(2, 3) = 2 + 3 = 5. x=3, y=2 -> f(3, 2) = 3 + 2 = 5. x=4, y=1 -> f(4, 1) = 4 + 1 = 5.

Example 2:

Input:

function_id = 2, z = 5

Output:

[[1,5],[5,1]]

Explanation:

The hidden formula for function_id = 2 is f(x, y) = x * y. The following positive integer values of x and y make f(x, y) equal to 5: x=1, y=5 -> f(1, 5) = 1 * 5 = 5. x=5, y=1 -> f(5, 1) = 5 * 1 = 5.

Constraints:

1 <= function_id <= 9

1 <= z <= 100

It is guaranteed that the solutions of

f(x, y) == z

will be in the range

1 <= x, y <= 1000

.

It is also guaranteed that

f(x, y)

will fit in 32 bit signed integer if

1 <= x, y <= 1000

.

Code Snippets

C++:

```
/*
* // This is the custom function interface.
* // You should not implement it, or speculate about its implementation
* class CustomFunction {
* public:
* // Returns f(x, y) for any given positive integers x and y.
* // Note that f(x, y) is increasing with respect to both x and y.
* // i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
* int f(int x, int y);
* };
*/

class Solution {
public:
vector<vector<int>> findSolution(CustomFunction& customfunction, int z) {

}
};
```

Java:

```
/*
* // This is the custom function interface.
* // You should not implement it, or speculate about its implementation
* class CustomFunction {
* // Returns f(x, y) for any given positive integers x and y.
* // Note that f(x, y) is increasing with respect to both x and y.
* // i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
* public int f(int x, int y);
* };
*/

class Solution {
public List<List<Integer>> findSolution(CustomFunction customfunction, int z)
{

}
}
```

Python3:

```
"""
This is the custom function interface.
You should not implement it, or speculate about its implementation
class CustomFunction:
# Returns f(x, y) for any given positive integers x and y.
# Note that f(x, y) is increasing with respect to both x and y.
# i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
def f(self, x, y):

"""


class Solution:
def findSolution(self, customfunction: 'CustomFunction', z: int) ->
List[List[int]]:
```

Python:

```
"""
This is the custom function interface.
You should not implement it, or speculate about its implementation
class CustomFunction:
# Returns f(x, y) for any given positive integers x and y.
# Note that f(x, y) is increasing with respect to both x and y.
# i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
def f(self, x, y):
"""


class Solution(object):
def findSolution(self, customfunction, z):
"""
:type num: int
:type z: int
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* // This is the CustomFunction's API interface.
* // You should not implement it, or speculate about its implementation
* function CustomFunction() {
* @param {integer, integer} x, y
* @return {integer}
* this.f = function(x, y) {
* ...
* };
* };
*/

/**
* @param {CustomFunction} customfunction
* @param {integer} z
* @return {integer[][]}
*/
var findSolution = function(customfunction, z) {

};
```

TypeScript:

```
/**
* // This is the CustomFunction's API interface.
* // You should not implement it, or speculate about its implementation
* class CustomFunction {
* f(x: number, y: number): number {}
* }
*/

function findSolution(customfunction: CustomFunction, z: number): number[][]
{

};
```

C#:

```
/*
* // This is the custom function interface.
* // You should not implement it, or speculate about its implementation
* public class CustomFunction {
* // Returns f(x, y) for any given positive integers x and y.
* // Note that f(x, y) is increasing with respect to both x and y.
* // i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
* public int f(int x, int y);
* };
*/

public class Solution {
public IList<IList<int>> FindSolution(CustomFunction customfunction, int z) {

}
}
```

C:

```
/*
* // This is the definition for customFunction API.
* // You should not implement it, or speculate about its implementation
*
* // Returns f(x, y) for any given positive integers x and y.
* // Note that f(x, y) is increasing with respect to both x and y.
* // i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** findSolution(int (*customFunction)(int, int), int z, int* returnSize,
int** returnColumnSizes) {

}
```

Go:

```
/**
* This is the declaration of customFunction API.
* @param x int
* @param x int
* @return Returns f(x, y) for any given positive integers x and y.
* Note that f(x, y) is increasing with respect to both x and y.
* i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
*/

func findSolution(customFunction func(int, int) int, z int) [][]int {

}
```

Kotlin:

```
/*
* // This is the custom function interface.
* // You should not implement it, or speculate about its implementation
* class CustomFunction {
* // Returns f(x, y) for any given positive integers x and y.
* // Note that f(x, y) is increasing with respect to both x and y.
* // i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
* fun f(x:Int, y:Int):Int {}
* };
*/

class Solution {
fun findSolution(customfunction:CustomFunction, z:Int):List<List<Int>> {

}
}
```

Swift:

```
/*
* // This is the custom function interface.
* // You should not implement it, or speculate about its implementation
* class CustomFunction {
* // Returns f(x, y) for any given positive integers x and y.
* // Note that f(x, y) is increasing with respect to both x and y.
* // i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
* func f(_ x: Int, _ y: Int) -> Int {}
* }
*/

class Solution {
func findSolution(_ customfunction: CustomFunction, _ z: Int) -> [[Int]] {
}
}
```

Rust:

```
/*
* // This is the custom function interface.
* // You should not implement it, or speculate about its implementation
* struct CustomFunction;
* impl CustomFunction {
* pub fn f(x:i32,y:i32)->i32{}
* }
*/

impl Solution {
pub fn find_solution(customfunction: &CustomFunction, z: i32) ->
Vec<Vec<i32>> {

}
}
```

Ruby:

```
# This is the custom function interface.
# You should not implement it, or speculate about its implementation
# class CustomFunction:
# def f(self, x, y):
# Returns f(x, y) for any given positive integers x and y.
# Note that f(x, y) is increasing with respect to both x and y.
# i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
# end
# end
#

# @param {CustomFunction} customfunction
# @param {Integer} z
# @return {List[List[Integer]]}
def findSolution(customfunction, z)

end
```

PHP:

```
/*
* // This is the custom function interface.
* // You should not implement it, or speculate about its implementation
* class CustomFunction {
* // Returns f(x, y) for any given positive integers x and y.
* // Note that f(x, y) is increasing with respect to both x and y.
* // i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
* public function f($x, $y){}
* };
*/

class Solution {
/**
* @param CustomFunction $customfunction
* @param Integer $z
* @return Integer[][]
*/
function findSolution($customfunction, $n) {

}
}
```

Scala:

```
/*
* // This is the custom function interface.
* // You should not implement it, or speculate about its implementation
* class CustomFunction {
* // Returns f(x, y) for any given positive integers x and y.
* // Note that f(x, y) is increasing with respect to both x and y.
* // i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
* def f(x: Int, y: Int): Int = {}
* };
*/

object Solution {
def findSolution(customfunction: CustomFunction, z: Int): List[List[Int]] = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Find Positive Integer Solution for a Given Equation
* Difficulty: Medium
* Tags: array, math, 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
*/

/*
* // This is the custom function interface.
* // You should not implement it, or speculate about its implementation
* class CustomFunction {
* public:
* // Returns f(x, y) for any given positive integers x and y.
* // Note that f(x, y) is increasing with respect to both x and y.
* // i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
* int f(int x, int y);
* };
*/

class Solution {
public:
vector<vector<int>> findSolution(CustomFunction& customfunction, int z) {

}
};
```

Java Solution:

```
/**
* Problem: Find Positive Integer Solution for a Given Equation
* Difficulty: Medium
* Tags: array, math, 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
```

```
*/

/*
* // This is the custom function interface.
* // You should not implement it, or speculate about its implementation
* class CustomFunction {
* // Returns f(x, y) for any given positive integers x and y.
* // Note that f(x, y) is increasing with respect to both x and y.
* // i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
* public int f(int x, int y);
* };
*/

class Solution {
public List<List<Integer>> findSolution(CustomFunction customfunction, int z)
{

}
}
```

Python3 Solution:

```
"""
Problem: Find Positive Integer Solution for a Given Equation
Difficulty: Medium
Tags: array, math, 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
"""

"""
This is the custom function interface.
You should not implement it, or speculate about its implementation
class CustomFunction:
# Returns f(x, y) for any given positive integers x and y.
# Note that f(x, y) is increasing with respect to both x and y.
# i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
def f(self, x, y):
```

```
"""

class Solution:
def findSolution(self, customfunction: 'CustomFunction', z: int) ->
List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
"""
This is the custom function interface.
You should not implement it, or speculate about its implementation
class CustomFunction:
# Returns f(x, y) for any given positive integers x and y.
# Note that f(x, y) is increasing with respect to both x and y.
# i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
def f(self, x, y):
"""

class Solution(object):
def findSolution(self, customfunction, z):
"""
:type num: int
:type z: int
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Find Positive Integer Solution for a Given Equation
* Difficulty: Medium
* Tags: array, math, 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
*/

/**
```

```
* // This is the CustomFunction's API interface.
* // You should not implement it, or speculate about its implementation
* function CustomFunction() {
* @param {integer, integer} x, y
* @return {integer}
* this.f = function(x, y) {
* ...
* };
* };
*/

/**
* @param {CustomFunction} customfunction
* @param {integer} z
* @return {integer[][]}
*/
var findSolution = function(customfunction, z) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Positive Integer Solution for a Given Equation
* Difficulty: Medium
* Tags: array, math, 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
*/

/**
* // This is the CustomFunction's API interface.
* // You should not implement it, or speculate about its implementation
* class CustomFunction {
* f(x: number, y: number): number {}
* }
*/

function findSolution(customfunction: CustomFunction, z: number): number[][]
```

```
{

};
```

C# Solution:

```
/*
* Problem: Find Positive Integer Solution for a Given Equation
* Difficulty: Medium
* Tags: array, math, 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
*/

/*
* // This is the custom function interface.
* // You should not implement it, or speculate about its implementation
* public class CustomFunction {
* // Returns f(x, y) for any given positive integers x and y.
* // Note that f(x, y) is increasing with respect to both x and y.
* // i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
* public int f(int x, int y);
* };
*/

public class Solution {
public IList<IList<int>> FindSolution(CustomFunction customfunction, int z) {

}
}
```

C Solution:

```
/*
* Problem: Find Positive Integer Solution for a Given Equation
* Difficulty: Medium
* Tags: array, math, 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
*/

/*
* // This is the definition for customFunction API.
* // You should not implement it, or speculate about its implementation
*
* // Returns f(x, y) for any given positive integers x and y.
* // Note that f(x, y) is increasing with respect to both x and y.
* // i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** findSolution(int (*customFunction)(int, int), int z, int* returnSize,
int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Find Positive Integer Solution for a Given Equation
// Difficulty: Medium
// Tags: array, math, 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

/**
* This is the declaration of customFunction API.
* @param x int
* @param x int
* @return Returns f(x, y) for any given positive integers x and y.
* Note that f(x, y) is increasing with respect to both x and y.
* i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
*/
```

```


func findSolution(customFunction func(int, int) int, z int) [][]int {


}
```

Kotlin Solution:

```
/*
* // This is the custom function interface.
* // You should not implement it, or speculate about its implementation
* class CustomFunction {
* // Returns f(x, y) for any given positive integers x and y.
* // Note that f(x, y) is increasing with respect to both x and y.
* // i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
* fun f(x:Int, y:Int):Int {}
* };
*/

class Solution {
fun findSolution(customfunction:CustomFunction, z:Int):List<List<Int>> {


}
}
```

Swift Solution:

```
/*
* // This is the custom function interface.
* // You should not implement it, or speculate about its implementation
* class CustomFunction {
* // Returns f(x, y) for any given positive integers x and y.
* // Note that f(x, y) is increasing with respect to both x and y.
* // i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
* func f(_ x: Int, _ y: Int) -> Int {}
* }
*/

class Solution {
func findSolution(_ customfunction: CustomFunction, _ z: Int) -> [[Int]] {


}
}
```

Rust Solution:

```
// Problem: Find Positive Integer Solution for a Given Equation
// Difficulty: Medium
// Tags: array, math, 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

/*
* // This is the custom function interface.
* // You should not implement it, or speculate about its implementation
* struct CustomFunction;
* impl CustomFunction {
* pub fn f(x:i32,y:i32)->i32{}
* }
*/

impl Solution {
pub fn find_solution(customfunction: &CustomFunction, z: i32) ->
Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# This is the custom function interface.
# You should not implement it, or speculate about its implementation
# class CustomFunction:
# def f(self, x, y):
# Returns f(x, y) for any given positive integers x and y.
# Note that f(x, y) is increasing with respect to both x and y.
# i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
# end
# end
#
```

```
# @param {CustomFunction} customfunction
# @param {Integer} z
# @return {List[List[Integer]]}
def findSolution(customfunction, z)

end
```

PHP Solution:

```
/*
* // This is the custom function interface.
* // You should not implement it, or speculate about its implementation
* class CustomFunction {
* // Returns f(x, y) for any given positive integers x and y.
* // Note that f(x, y) is increasing with respect to both x and y.
* // i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
* public function f($x, $y){}
* };
*/

class Solution {
/**
* @param CustomFunction $customfunction
* @param Integer $z
* @return Integer[][]
*/
function findSolution($customfunction, $n) {

}
}
```

Scala Solution:

```
/*
* // This is the custom function interface.
* // You should not implement it, or speculate about its implementation
* class CustomFunction {
* // Returns f(x, y) for any given positive integers x and y.
* // Note that f(x, y) is increasing with respect to both x and y.
* // i.e. f(x, y) < f(x + 1, y), f(x, y) < f(x, y + 1)
* def f(x: Int, y: Int): Int = {}
* };
*/

object Solution {
def findSolution(customfunction: CustomFunction, z: Int): List[List[Int]] = {

}
}
```