Problem 816: Ambiguous Coordinates

Problem Information

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

Problem Description

We had some 2-dimensional coordinates, like

"(1, 3)"

or

"(2, 0.5)"

. Then, we removed all commas, decimal points, and spaces and ended up with the string s.

For example,

"(1, 3)"

becomes

s = "(13)"

and

"(2, 0.5)"

becomes

s = "(205)"

.

Return

a list of strings representing all possibilities for what our original coordinates could have been

.

Our original representation never had extraneous zeroes, so we never started with numbers like

"00"

,

"0.0"

,

"0.00"

,

"1.0"

,

"001"

,

"00.01"

, or any other number that can be represented with fewer digits. Also, a decimal point within a number never occurs without at least one digit occurring before it, so we never started with numbers like

".1"

.

The final answer list can be returned in any order. All coordinates in the final answer have exactly one space between them (occurring after the comma.)

Example 1:

Input:

s = "(123)"

Output:

["(1, 2.3)","(1, 23)","(1.2, 3)","(12, 3)"]

Example 2:

Input:

s = "(0123)"

Output:

["(0, 1.23)","(0, 12.3)","(0, 123)","(0.1, 2.3)","(0.1, 23)","(0.12, 3)"]

Explanation:

0.0, 00, 0001 or 00.01 are not allowed.

Example 3:

Input:

s = "(00011)"

Output:

["(0, 0.011)","(0.001, 1)"]

Constraints:

4 <= s.length <= 12

s[0] == '('

and

s[s.length - 1] == ')'

.

The rest of

s

are digits.

Code Snippets

C++:

```
class Solution {
public:
vector<string> ambiguousCoordinates(string s) {

}
};
```

Java:

```
class Solution {
public List<String> ambiguousCoordinates(String s) {

}
}
```

Python3:

```
class Solution:
def ambiguousCoordinates(self, s: str) -> List[str]:
```

Python:

```
class Solution(object):
def ambiguousCoordinates(self, s):
"""
:type s: str
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string} s
* @return {string[]}
*/
var ambiguousCoordinates = function(s) {

};
```

TypeScript:

```
function ambiguousCoordinates(s: string): string[] {

};
```

C#:

```
public class Solution {
public IList<string> AmbiguousCoordinates(string s) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** ambiguousCoordinates(char* s, int* returnSize) {
```

```
}
```

Go:

```
func ambiguousCoordinates(s string) []string {

}
```

Kotlin:

```
class Solution {
fun ambiguousCoordinates(s: String): List<String> {

}
}
```

Swift:

```
class Solution {
func ambiguousCoordinates(_ s: String) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn ambiguous_coordinates(s: String) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String} s
# @return {String[]}
def ambiguous_coordinates(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return String[]
*/
function ambiguousCoordinates($s) {

}
}
```

Dart:

```
class Solution {
List<String> ambiguousCoordinates(String s) {

}
}
```

Scala:

```
object Solution {
def ambiguousCoordinates(s: String): List[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec ambiguous_coordinates(s :: String.t) :: [String.t]
def ambiguous_coordinates(s) do

end
end
```

Erlang:

```
-spec ambiguous_coordinates(S :: unicode:unicode_binary()) ->
[unicode:unicode_binary()].
ambiguous_coordinates(S) ->
.
```

Racket:

```
(define/contract (ambiguous-coordinates s)
(-> string? (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Ambiguous Coordinates
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<string> ambiguousCoordinates(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Ambiguous Coordinates
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public List<String> ambiguousCoordinates(String s) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Ambiguous Coordinates
Difficulty: Medium
Tags: string

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def ambiguousCoordinates(self, s: str) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def ambiguousCoordinates(self, s):
"""
:type s: str
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Ambiguous Coordinates
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
/**
* @param {string} s
* @return {string[]}
*/
var ambiguousCoordinates = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Ambiguous Coordinates
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function ambiguousCoordinates(s: string): string[] {

};
```

C# Solution:

```
/*
* Problem: Ambiguous Coordinates
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public IList<string> AmbiguousCoordinates(string s) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Ambiguous Coordinates
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* 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().
*/
char** ambiguousCoordinates(char* s, int* returnSize) {

}
```

Go Solution:

```
// Problem: Ambiguous Coordinates
// Difficulty: Medium
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func ambiguousCoordinates(s string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun ambiguousCoordinates(s: String): List<String> {

}
```

```
}
```

Swift Solution:

```
class Solution {
func ambiguousCoordinates(_ s: String) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Ambiguous Coordinates
// Difficulty: Medium
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn ambiguous_coordinates(s: String) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {String[]}
def ambiguous_coordinates(s)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return String[]
```

```
*/
function ambiguousCoordinates($s) {

}
}
```

Dart Solution:

```
class Solution {
List<String> ambiguousCoordinates(String s) {

}
}
```

Scala Solution:

```
object Solution {
def ambiguousCoordinates(s: String): List[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec ambiguous_coordinates(s :: String.t) :: [String.t]
def ambiguous_coordinates(s) do

end
end
```

Erlang Solution:

```
-spec ambiguous_coordinates(S :: unicode:unicode_binary()) ->
[unicode:unicode_binary()].
ambiguous_coordinates(S) ->
.
```

Racket Solution:

```
(define/contract (ambiguous-coordinates s)
(-> string? (listof string?))
)
```