Problem 1307: Verbal Arithmetic Puzzle

Problem Information

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

Problem Description

Given an equation, represented by

words

on the left side and the

result

on the right side.

You need to check if the equation is solvable under the following rules:

Each character is decoded as one digit (0 - 9).

No two characters can map to the same digit.

Each

words[i]

and

result

are decoded as one number

without

leading zeros.

Sum of numbers on the left side (

words

) will equal to the number on the right side (

result

).

Return

true

if the equation is solvable, otherwise return

false

.

Example 1:

Input:

words = ["SEND","MORE"], result = "MONEY"

Output:

true

Explanation:

Map 'S'-> 9, 'E'->5, 'N'->6, 'D'->7, 'M'->1, 'O'->0, 'R'->8, 'Y'->'2' Such that: "SEND" + "MORE" = "MONEY" , 9567 + 1085 = 10652

Example 2:

Input:

words = ["SIX","SEVEN","SEVEN"], result = "TWENTY"

Output:

true

Explanation:

Map 'S'-> 6, 'I'->5, 'X'->0, 'E'->8, 'V'->7, 'N'->2, 'T'->1, 'W'->'3', 'Y'->4 Such that: "SIX" + "SEVEN" + "SEVEN" = "TWENTY" , 650 + 68782 + 68782 = 138214

Example 3:

Input:

words = ["LEET","CODE"], result = "POINT"

Output:

false

Explanation:

There is no possible mapping to satisfy the equation, so we return false. Note that two different characters cannot map to the same digit.

Constraints:

2 <= words.length <= 5

1 <= words[i].length, result.length <= 7

words[i], result

contain only uppercase English letters.

The number of different characters used in the expression is at most

10

.

Code Snippets

C++:

```
class Solution {
public:
bool isSolvable(vector<string>& words, string result) {

}
};
```

Java:

```
class Solution {
public boolean isSolvable(String[] words, String result) {

}
}
```

Python3:

```
class Solution:
def isSolvable(self, words: List[str], result: str) -> bool:
```

Python:

```
class Solution(object):
def isSolvable(self, words, result):
"""
:type words: List[str]
:type result: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string[]} words
* @param {string} result
* @return {boolean}
*/
var isSolvable = function(words, result) {

};
```

TypeScript:

```
function isSolvable(words: string[], result: string): boolean {

};
```

C#:

```
public class Solution {
public bool IsSolvable(string[] words, string result) {

}
}
```

C:

```
bool isSolvable(char** words, int wordsSize, char* result) {

}
```

Go:

```
func isSolvable(words []string, result string) bool {

}
```

Kotlin:

```
class Solution {
fun isSolvable(words: Array<String>, result: String): Boolean {

}
}
```

Swift:

```
class Solution {
func isSolvable(_ words: [String], _ result: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn is_solvable(words: Vec<String>, result: String) -> bool {

}
}
```

Ruby:

```
# @param {String[]} words
# @param {String} result
# @return {Boolean}
def is_solvable(words, result)

end
```

PHP:

```
class Solution {

/**
* @param String[] $words
* @param String $result
* @return Boolean
*/
function isSolvable($words, $result) {

}
}
```

Dart:

```
class Solution {
bool isSolvable(List<String> words, String result) {

}
}
```

Scala:

```
object Solution {
def isSolvable(words: Array[String], result: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec is_solvable(words :: [String.t], result :: String.t) :: boolean
def is_solvable(words, result) do

end
end
```

Erlang:

```
-spec is_solvable(Words :: [unicode:unicode_binary()], Result ::
unicode:unicode_binary()) -> boolean().
is_solvable(Words, Result) ->
.
```

Racket:

```
(define/contract (is-solvable words result)
(-> (listof string?) string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Verbal Arithmetic Puzzle
* Difficulty: Hard
* Tags: array, string, math
*
* 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 isSolvable(vector<string>& words, string result) {

}
};
```

Java Solution:

```
/**
* Problem: Verbal Arithmetic Puzzle
* Difficulty: Hard
* Tags: array, string, math
*
* 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 isSolvable(String[] words, String result) {

}
}
```

Python3 Solution:

```
"""
Problem: Verbal Arithmetic Puzzle
Difficulty: Hard
Tags: array, string, math
```

```
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 isSolvable(self, words: List[str], result: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def isSolvable(self, words, result):
"""
:type words: List[str]
:type result: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Verbal Arithmetic Puzzle
* Difficulty: Hard
* Tags: array, string, math
*
* 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[]} words
* @param {string} result
* @return {boolean}
*/
var isSolvable = function(words, result) {

};
```

TypeScript Solution:

```
/**
* Problem: Verbal Arithmetic Puzzle
* Difficulty: Hard
* Tags: array, string, math
*
* 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 isSolvable(words: string[], result: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Verbal Arithmetic Puzzle
* Difficulty: Hard
* Tags: array, string, math
*
* 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 IsSolvable(string[] words, string result) {

}
}
```

C Solution:

```
/*
* Problem: Verbal Arithmetic Puzzle
* Difficulty: Hard
* Tags: array, string, math
*
* 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 isSolvable(char** words, int wordsSize, char* result) {

}
```

Go Solution:

```go
// Problem: Verbal Arithmetic Puzzle
// Difficulty: Hard
// Tags: array, string, math
//
// 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 isSolvable(words []string, result string) bool {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun isSolvable(words: Array<String>, result: String): Boolean {

}
}
```

Swift Solution:

```swift
class Solution {
func isSolvable(_ words: [String], _ result: String) -> Bool {

}
}
```

Rust Solution:

```rust
// Problem: Verbal Arithmetic Puzzle
// Difficulty: Hard
// Tags: array, string, math
//
// 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 is_solvable(words: Vec<String>, result: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String[]} words
# @param {String} result
# @return {Boolean}
def is_solvable(words, result)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $words
* @param String $result
* @return Boolean
*/
function isSolvable($words, $result) {

}
}
```

Dart Solution:

```
class Solution {
bool isSolvable(List<String> words, String result) {

}
}
```

Scala Solution:

```
object Solution {
def isSolvable(words: Array[String], result: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec is_solvable(words :: [String.t], result :: String.t) :: boolean
def is_solvable(words, result) do

end
end
```

Erlang Solution:

```
-spec is_solvable(Words :: [unicode:unicode_binary()], Result ::
unicode:unicode_binary()) -> boolean().
is_solvable(Words, Result) ->
.
```

Racket Solution:

```
(define/contract (is-solvable words result)
(-> (listof string?) string? boolean?)
)
```