Problem 320: Generalized Abbreviation

Problem Information

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

Problem Description

A word's

generalized abbreviation

can be constructed by taking any number of

non-overlapping

and

non-adjacent

substrings

and replacing them with their respective lengths.

For example,

"abcde"

can be abbreviated into:

"a3e"

(

"bcd"

turned into

"3"

)

"1bcd1"

(

"a"

and

"e"

both turned into

"1"

)

"5"

(

"abcde"

turned into

"5"

)

"abcde"

(no substrings replaced)

However, these abbreviations are

invalid

:

"23"

(

"ab"

turned into

"2"

and

"cde"

turned into

"3"

) is invalid as the substrings chosen are adjacent.

"22de"

(

"ab"

turned into

"2"

and

"bc"

turned into

"2"

) is invalid as the substring chosen overlap.

Given a string

word

, return

a list of all the possible

generalized abbreviations

of

word

. Return the answer in

any order

.

Example 1:

Input:

word = "word"

Output:

["4","3d","2r1","2rd","1o2","1o1d","1or1","1ord","w3","w2d","w1r1","w1rd","wo2","wo1d","wor1",
"word"]

Example 2:

Input:

word = "a"

Output:

["1","a"]

Constraints:

1 <= word.length <= 15

word

consists of only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
vector<string> generateAbbreviations(string word) {

}
};
```

Java:

```
class Solution {
public List<String> generateAbbreviations(String word) {

}
}
```

Python3:

```
class Solution:
def generateAbbreviations(self, word: str) -> List[str]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function generateAbbreviations(word: string): string[] {

};
```

C#:

```
public class Solution {
public IList<string> GenerateAbbreviations(string word) {

}
}
```

C:

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

}
```

Go:

```
func generateAbbreviations(word string) []string {

}
```

Kotlin:

```
class Solution {
fun generateAbbreviations(word: String): List<String> {

}
}
```

Swift:

```
class Solution {
func generateAbbreviations(_ word: String) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn generate_abbreviations(word: String) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String} word
# @return {String[]}
def generate_abbreviations(word)

end
```

PHP:

```
class Solution {

/**
* @param String $word
* @return String[]
*/
function generateAbbreviations($word) {

}
}
```

Dart:

```
class Solution {
List<String> generateAbbreviations(String word) {

}
}
```

Scala:

```
object Solution {
def generateAbbreviations(word: String): List[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec generate_abbreviations(word :: String.t) :: [String.t]
def generate_abbreviations(word) do

end
end
```

Erlang:

```
-spec generate_abbreviations(Word :: unicode:unicode_binary()) ->
[unicode:unicode_binary()].
generate_abbreviations(Word) ->
.
```

Racket:

```
(define/contract (generate-abbreviations word)
(-> string? (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Generalized Abbreviation
* Difficulty: Medium
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
vector<string> generateAbbreviations(string word) {

}
};
```

Java Solution:

```
/**
* Problem: Generalized Abbreviation
* Difficulty: Medium
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public List<String> generateAbbreviations(String word) {
}
}
```

Python3 Solution:

```
"""
Problem: Generalized Abbreviation
Difficulty: Medium
Tags: string, tree

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Generalized Abbreviation
* Difficulty: Medium
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Generalized Abbreviation
* Difficulty: Medium
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function generateAbbreviations(word: string): string[] {

};
```

C# Solution:

```
/*
* Problem: Generalized Abbreviation
* Difficulty: Medium
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public IList<string> GenerateAbbreviations(string word) {

}
}
```

C Solution:

```
/*
* Problem: Generalized Abbreviation
* Difficulty: Medium
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
```

Go Solution:

```
// Problem: Generalized Abbreviation
// Difficulty: Medium
// Tags: string, tree
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func generateAbbreviations(word string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun generateAbbreviations(word: String): List<String> {

}

}
```

Swift Solution:

```
class Solution {
func generateAbbreviations(_ word: String) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Generalized Abbreviation
// Difficulty: Medium
// Tags: string, tree
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn generate_abbreviations(word: String) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String} word
# @return {String[]}
def generate_abbreviations(word)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $word
* @return String[]
*/
function generateAbbreviations($word) {

}
}
```

Dart Solution:

```
class Solution {
List<String> generateAbbreviations(String word) {

}
}
```

Scala Solution:

```
object Solution {
def generateAbbreviations(word: String): List[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec generate_abbreviations(word :: String.t) :: [String.t]
def generate_abbreviations(word) do

end
end
```

Erlang Solution:

```
-spec generate_abbreviations(Word :: unicode:unicode_binary()) ->
[unicode:unicode_binary()].
generate_abbreviations(Word) ->
.
```

Racket Solution:

```
(define/contract (generate-abbreviations word)
(-> string? (listof string?))
)
```