Problem 555: Split Concatenated Strings

Problem Information

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

Problem Description

You are given an array of strings

strs

. You could concatenate these strings together into a loop, where for each string, you could choose to reverse it or not. Among all the possible loops

Return

the lexicographically largest string after cutting the loop, which will make the looped string into a regular one

.

Specifically, to find the lexicographically largest string, you need to experience two phases:

Concatenate all the strings into a loop, where you can reverse some strings or not and connect them in the same order as given.

Cut and make one breakpoint in any place of the loop, which will make the looped string into a regular one starting from the character at the cutpoint.

And your job is to find the lexicographically largest one among all the possible regular strings.

Example 1:

Input:

strs = ["abc","xyz"]

Output:

"zyxcba"

Explanation:

You can get the looped string "-abcxyz-", "-abczyx-", "-cbaxyz-", "-cbazyx-", where '-' represents the looped status. The answer string came from the fourth looped one, where you could cut from the middle character 'a' and get "zyxcba".

Example 2:

Input:

strs = ["abc"]

Output:

"cba"

Constraints:

1 <= strs.length <= 1000

1 <= strs[i].length <= 1000

1 <= sum(strs[i].length) <= 1000

strs[i]

consists of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
string splitLoopedString(vector<string>& strs) {

}
};
```

Java:

```
class Solution {
public String splitLoopedString(String[] strs) {

}
}
```

Python3:

```
class Solution:
def splitLoopedString(self, strs: List[str]) -> str:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function splitLoopedString(strs: string[]): string {

};
```

C#:

```
public class Solution {
public string SplitLoopedString(string[] strs) {

}
}
```

C:

```
char* splitLoopedString(char** strs, int strsSize) {

}
```

Go:

```
func splitLoopedString(strs []string) string {

}
```

Kotlin:

```
class Solution {
fun splitLoopedString(strs: Array<String>): String {

}
}
```

Swift:

```
class Solution {
func splitLoopedString(_ strs: [String]) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn split_looped_string(strs: Vec<String>) -> String {

}
}
```

Ruby:

```
# @param {String[]} strs
# @return {String}
def split_looped_string(strs)

end
```

PHP:

```
class Solution {

/**
* @param String[] $strs
* @return String
*/
function splitLoopedString($strs) {

}
}
```

Dart:

```
class Solution {
String splitLoopedString(List<String> strs) {

}
}
```

Scala:

```
object Solution {
def splitLoopedString(strs: Array[String]): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec split_looped_string(strs :: [String.t]) :: String.t
def split_looped_string(strs) do

end
end
```

Erlang:

```
-spec split_looped_string(Strs :: [unicode:unicode_binary()]) ->
unicode:unicode_binary().
split_looped_string(Strs) ->
.
```

Racket:

```
(define/contract (split-looped-string strs)
(-> (listof string?) string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Split Concatenated Strings
* Difficulty: Medium
* Tags: array, string, graph, greedy
*
* 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:
string splitLoopedString(vector<string>& strs) {

}
};
```

Java Solution:

```
/**
* Problem: Split Concatenated Strings
* Difficulty: Medium
* Tags: array, string, graph, greedy
*
* 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 String splitLoopedString(String[] strs) {

}
}
```

Python3 Solution:

```
"""
Problem: Split Concatenated Strings
Difficulty: Medium
Tags: array, string, graph, greedy

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 splitLoopedString(self, strs: List[str]) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Split Concatenated Strings
* Difficulty: Medium
* Tags: array, string, graph, greedy
*
* 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[]} strs
* @return {string}
*/
var splitLoopedString = function(strs) {

};
```

TypeScript Solution:

```
/**
* Problem: Split Concatenated Strings
* Difficulty: Medium
* Tags: array, string, graph, greedy
*
* 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 splitLoopedString(strs: string[]): string {

};
```

C# Solution:

```
/*
* Problem: Split Concatenated Strings
* Difficulty: Medium
* Tags: array, string, graph, greedy
*
* 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 string SplitLoopedString(string[] strs) {

}
}
```

C Solution:

```
/*
* Problem: Split Concatenated Strings
* Difficulty: Medium
* Tags: array, string, graph, greedy
*
* 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
*/

char* splitLoopedString(char** strs, int strsSize) {

}
```

Go Solution:

```
// Problem: Split Concatenated Strings
// Difficulty: Medium
// Tags: array, string, graph, greedy
//
// 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 splitLoopedString(strs []string) string {

}
```

Kotlin Solution:

```
class Solution {
fun splitLoopedString(strs: Array<String>): String {

}
}
```

Swift Solution:

```
class Solution {
func splitLoopedString(_ strs: [String]) -> String {

}
}
```

Rust Solution:

```
// Problem: Split Concatenated Strings
// Difficulty: Medium
// Tags: array, string, graph, greedy
//
// 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 split_looped_string(strs: Vec<String>) -> String {

}
}
```

Ruby Solution:

```
# @param {String[]} strs
# @return {String}
def split_looped_string(strs)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $strs
* @return String
*/
function splitLoopedString($strs) {

}
}
```

Dart Solution:

```
class Solution {
String splitLoopedString(List<String> strs) {

}
}
```

Scala Solution:

```
object Solution {
def splitLoopedString(strs: Array[String]): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec split_looped_string(strs :: [String.t]) :: String.t
def split_looped_string(strs) do

end
end
```

Erlang Solution:

```
-spec split_looped_string(Strs :: [unicode:unicode_binary()]) ->
unicode:unicode_binary().
split_looped_string(Strs) ->
.
```

Racket Solution:

```
(define/contract (split-looped-string strs)
(-> (listof string?) string?)
)
```