Problem 2573: Find the String with LCP

Problem Information

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

Problem Description

We define the

lcp

matrix of any

0-indexed

string

word

of

n

lowercase English letters as an

n x n

grid such that:

lcp[i][j]

is equal to the length of the

longest common prefix

between the substrings

word[i,n-1]

and

word[j,n-1]

.

Given an

n x n

matrix

lcp

, return the alphabetically smallest string

word

that corresponds to

lcp

. If there is no such string, return an empty string.

A string

a

is lexicographically smaller than a string

b

(of the same length) if in the first position where a

and

b

differ, string

a

has a letter that appears earlier in the alphabet than the corresponding letter in

b

. For example,

"aabd"

is lexicographically smaller than

"aaca"

because the first position they differ is at the third letter, and

'b'

comes before

'c'

.

Example 1:

Input:

lcp = [[4,0,2,0],[0,3,0,1],[2,0,2,0],[0,1,0,1]] Output:

"abab"

Explanation:

lcp corresponds to any 4 letter string with two alternating letters. The lexicographically smallest of them is "abab".

Example 2:

Input:

lcp = [[4,3,2,1],[3,3,2,1],[2,2,2,1],[1,1,1,1]]

Output:

"aaaa"

Explanation:

lcp corresponds to any 4 letter string with a single distinct letter. The lexicographically smallest of them is "aaaa".

Example 3:

Input:

lcp = [[4,3,2,1],[3,3,2,1],[2,2,2,1],[1,1,1,3]]

Output:

""

Explanation:

lcp[3][3] cannot be equal to 3 since word[3,...,3] consists of only a single letter; Thus, no answer exists.

Constraints:

1 <= n ==

lcp.length ==

lcp[i].length

<= 1000

0 <= lcp[i][j] <= n

Code Snippets

C++:

```
class Solution {
public:
string findTheString(vector<vector<int>>& lcp) {

}
};
```

Java:

```
class Solution {
public String findTheString(int[][] lcp) {

}
}
```

Python3:

```
class Solution:
def findTheString(self, lcp: List[List[int]]) -> str:
```

Python:

```
class Solution(object):
def findTheString(self, lcp):
```

```
"""
:type lcp: List[List[int]]
:rtype: str
"""
```

JavaScript:

```
/**
* @param {number[][]} lcp
* @return {string}
*/
var findTheString = function(lcp) {

};
```

TypeScript:

```
function findTheString(lcp: number[][]): string {

};
```

C#:

```
public class Solution {
public string FindTheString(int[][] lcp) {

}
}
```

C:

```
char* findTheString(int** lcp, int lcpSize, int* lcpColSize) {

}
```

Go:

```
func findTheString(lcp [][]int) string {

}
```

Kotlin:

```
class Solution {
fun findTheString(lcp: Array<IntArray>): String {

}
}
```

Swift:

```
class Solution {
func findTheString(_ lcp: [[Int]]) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn find_the_string(lcp: Vec<Vec<i32>>) -> String {

}
}
```

Ruby:

```
# @param {Integer[][]} lcp
# @return {String}
def find_the_string(lcp)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $lcp
* @return String
*/
function findTheString($lcp) {

}
}
```

Dart:

```
class Solution {
String findTheString(List<List<int>> lcp) {

}
}
```

Scala:

```
object Solution {
def findTheString(lcp: Array[Array[Int]]): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_the_string(lcp :: [[integer]]) :: String.t
def find_the_string(lcp) do

end
end
```

Erlang:

```
-spec find_the_string(Lcp :: [[integer()]]) -> unicode:unicode_binary().
find_the_string(Lcp) ->
.
```

Racket:

```
(define/contract (find-the-string lcp)
(-> (listof (listof exact-integer?)) string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the String with LCP
* Difficulty: Hard
* Tags: array, string, tree, graph, dp, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
string findTheString(vector<vector<int>>& lcp) {

}
};
```

Java Solution:

```
/**
* Problem: Find the String with LCP
* Difficulty: Hard
* Tags: array, string, tree, graph, dp, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public String findTheString(int[][] lcp) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the String with LCP
Difficulty: Hard
Tags: array, string, tree, graph, dp, greedy
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

```
class Solution(object):
def findTheString(self, lcp):
"""
:type lcp: List[List[int]]
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Find the String with LCP
* Difficulty: Hard
* Tags: array, string, tree, graph, dp, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[][]} lcp
* @return {string}
*/
var findTheString = function(lcp) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the String with LCP
* Difficulty: Hard
* Tags: array, string, tree, graph, dp, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function findTheString(lcp: number[][]): string {

};
```

C# Solution:

```
/*
* Problem: Find the String with LCP
* Difficulty: Hard
* Tags: array, string, tree, graph, dp, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public string FindTheString(int[][] lcp) {

}
}
```

C Solution:

```
/*
* Problem: Find the String with LCP
* Difficulty: Hard
* Tags: array, string, tree, graph, dp, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

char* findTheString(int** lcp, int lcpSize, int* lcpColSize) {

}
```

Go Solution:

```
// Problem: Find the String with LCP
// Difficulty: Hard
// Tags: array, string, tree, graph, dp, greedy
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func findTheString(lcp [][]int) string {

}
```

Kotlin Solution:

```
class Solution {
fun findTheString(lcp: Array<IntArray>): String {

}
}
```

Swift Solution:

```
class Solution {
func findTheString(_ lcp: [[Int]]) -> String {

}
}
```

Rust Solution:

```
// Problem: Find the String with LCP
// Difficulty: Hard
// Tags: array, string, tree, graph, dp, greedy
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn find_the_string(lcp: Vec<Vec<i32>>) -> String {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} lcp
# @return {String}
def find_the_string(lcp)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $lcp
* @return String
*/
function findTheString($lcp) {

}
}
```

Dart Solution:

```
class Solution {
String findTheString(List<List<int>> lcp) {

}
}
```

Scala Solution:

```
object Solution {
def findTheString(lcp: Array[Array[Int]]): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_the_string(lcp :: [[integer]]) :: String.t
def find_the_string(lcp) do

end
end
```

Erlang Solution:

```
-spec find_the_string(Lcp :: [[integer()]]) -> unicode:unicode_binary().
find_the_string(Lcp) ->
.
```

Racket Solution:

```
(define/contract (find-the-string lcp)
(-> (listof (listof exact-integer?)) string?)
)
```