Problem 1257: Smallest Common Region

Problem Information

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

Problem Description

You are given some lists of

regions

where the first region of each list

directly

contains all other regions in that list.

If a region

x

contains a region

y

directly

, and region

y

contains region

z

directly

, then region

x

is said to contain region

z

indirectly

. Note that region

x

also

indirectly

contains all regions

indirectly

containd in

y

.

Naturally, if a region

x

contains (either

directly

or

indirectly

) another region

y

, then

x

is bigger than or equal to

y

in size. Also, by definition, a region

x

contains itself.

Given two regions:

region1

and

region2

, return

the smallest region that contains both of them

.

It is guaranteed the smallest region exists.

Example 1:

Input:

regions = [["Earth","North America","South America"], ["North America","United States","Canada"], ["United States","New York","Boston"], ["Canada","Ontario","Quebec"], ["South America","Brazil"]], region1 = "Quebec", region2 = "New York"

Output:

"North America"

Example 2:

Input:

regions = [["Earth", "North America", "South America"],["North America", "United States", "Canada"],["United States", "New York", "Boston"],["Canada", "Ontario", "Quebec"],["South America", "Brazil"]], region1 = "Canada", region2 = "South America"

Output:

"Earth"

Constraints:

2 <= regions.length <= 10

4

2 <= regions[i].length <= 20

1 <= regions[i][j].length, region1.length, region2.length <= 20

region1 != region2

regions[i][j]

,

region1

, and

region2

consist of English letters.

The input is generated such that there exists a region which contains all the other regions, either directly or indirectly.

A region cannot be directly contained in more than one region.

Code Snippets

C++:

```
class Solution {
public:
string findSmallestRegion(vector<vector<string>>& regions, string region1,
string region2) {

}
};
```

Java:

```
class Solution {
public String findSmallestRegion(List<List<String>> regions, String region1,
String region2) {

}
}
```

Python3:

```
class Solution:
def findSmallestRegion(self, regions: List[List[str]], region1: str, region2:
str) -> str:
```

Python:

```
class Solution(object):
def findSmallestRegion(self, regions, region1, region2):
"""
:type regions: List[List[str]]
:type region1: str
:type region2: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string[][]} regions
* @param {string} region1
* @param {string} region2
* @return {string}
*/
var findSmallestRegion = function(regions, region1, region2) {

};
```

TypeScript:

```
function findSmallestRegion(regions: string[][], region1: string, region2:
string): string {

};
```

C#:

```
public class Solution {
public string FindSmallestRegion(IList<IList<string>> regions, string
region1, string region2) {

}
}
```

C:

```
char* findSmallestRegion(char*** regions, int regionsSize, int*
regionsColSize, char* region1, char* region2) {
}
```

Go:

```
func findSmallestRegion(regions [][]string, region1 string, region2 string)
string {

}
```

Kotlin:

```
class Solution {
fun findSmallestRegion(regions: List<List<String>>, region1: String, region2:
String): String {

}
}
```

Swift:

```
class Solution {
func findSmallestRegion(_ regions: [[String]], _ region1: String, _ region2:
String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn find_smallest_region(regions: Vec<Vec<String>>, region1: String,
region2: String) -> String {

}
}
```

Ruby:

```
# @param {String[][]} regions
# @param {String} region1
# @param {String} region2
# @return {String}
def find_smallest_region(regions, region1, region2)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $regions
* @param String $region1
* @param String $region2
* @return String
*/
function findSmallestRegion($regions, $region1, $region2) {

}
}
```

Dart:

```
class Solution {
String findSmallestRegion(List<List<String>> regions, String region1, String
region2) {

}
}
```

Scala:

```
object Solution {
def findSmallestRegion(regions: List[List[String]], region1: String, region2:
String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_smallest_region(regions :: [[String.t]], region1 :: String.t,
region2 :: String.t) :: String.t
def find_smallest_region(regions, region1, region2) do

end
end
```

Erlang:

```
-spec find_smallest_region(Regions :: [[unicode:unicode_binary()]], Region1
:: unicode:unicode_binary(), Region2 :: unicode:unicode_binary()) ->
unicode:unicode_binary().
find_smallest_region(Regions, Region1, Region2) ->
.
```

Racket:

```
(define/contract (find-smallest-region regions region1 region2)
(-> (listof (listof string?)) string? string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Smallest Common Region
* Difficulty: Medium
* Tags: array, string, tree, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
string findSmallestRegion(vector<vector<string>>& regions, string region1,
string region2) {

}
};
```

Java Solution:

```
/**
* Problem: Smallest Common Region
* Difficulty: Medium
* Tags: array, string, tree, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public String findSmallestRegion(List<List<String>> regions, String region1,
String region2) {

}
}
```

Python3 Solution:

```
"""
Problem: Smallest Common Region
Difficulty: Medium
Tags: array, string, tree, hash, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def findSmallestRegion(self, regions: List[List[str]], region1: str, region2:
str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findSmallestRegion(self, regions, region1, region2):
"""
```

```
:type regions: List[List[str]]
:type region1: str
:type region2: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Smallest Common Region
* Difficulty: Medium
* Tags: array, string, tree, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {string[][]} regions
* @param {string} region1
* @param {string} region2
* @return {string}
*/
var findSmallestRegion = function(regions, region1, region2) {

};
```

TypeScript Solution:

```
/**
* Problem: Smallest Common Region
* Difficulty: Medium
* Tags: array, string, tree, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function findSmallestRegion(regions: string[][], region1: string, region2:
```

```
string): string {

};
```

C# Solution:

```
/*
* Problem: Smallest Common Region
* Difficulty: Medium
* Tags: array, string, tree, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public string FindSmallestRegion(IList<IList<string>> regions, string
region1, string region2) {

}
}
```

C Solution:

```
/*
* Problem: Smallest Common Region
* Difficulty: Medium
* Tags: array, string, tree, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

char* findSmallestRegion(char*** regions, int regionsSize, int*
regionsColSize, char* region1, char* region2) {

}
```

Go Solution:

```
// Problem: Smallest Common Region
// Difficulty: Medium
// Tags: array, string, tree, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func findSmallestRegion(regions [][]string, region1 string, region2 string)
string {

}
```

Kotlin Solution:

```
class Solution {
fun findSmallestRegion(regions: List<List<String>>, region1: String, region2:
String): String {

}
}
```

Swift Solution:

```
class Solution {
func findSmallestRegion(_ regions: [[String]], _ region1: String, _ region2:
String) -> String {

}
}
```

Rust Solution:

```
// Problem: Smallest Common Region
// Difficulty: Medium
// Tags: array, string, tree, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
```

```
pub fn find_smallest_region(regions: Vec<Vec<String>>, region1: String,
region2: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String[][]} regions
# @param {String} region1
# @param {String} region2
# @return {String}
def find_smallest_region(regions, region1, region2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $regions
* @param String $region1
* @param String $region2
* @return String
*/
function findSmallestRegion($regions, $region1, $region2) {

}
}
```

Dart Solution:

```
class Solution {
String findSmallestRegion(List<List<String>> regions, String region1, String
region2) {

}
}
```

Scala Solution:

```
object Solution {
def findSmallestRegion(regions: List[List[String]], region1: String, region2:
String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_smallest_region(regions :: [[String.t]], region1 :: String.t,
region2 :: String.t) :: String.t
def find_smallest_region(regions, region1, region2) do

end
end
```

Erlang Solution:

```
-spec find_smallest_region(Regions :: [[unicode:unicode_binary()]], Region1
:: unicode:unicode_binary(), Region2 :: unicode:unicode_binary()) ->
unicode:unicode_binary().
find_smallest_region(Regions, Region1, Region2) ->
.
```

Racket Solution:

```
(define/contract (find-smallest-region regions region1 region2)
(-> (listof (listof string?)) string? string? string?)
)
```