Problem 1125: Smallest Sufficient Team

Problem Information

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

Problem Description

In a project, you have a list of required skills

req_skills

, and a list of people. The

i

th

person

people[i]

contains a list of skills that the person has.

Consider a sufficient team: a set of people such that for every required skill in

req_skills

, there is at least one person in the team who has that skill. We can represent these teams by the index of each person.

For example,

team = [0, 1, 3]

represents the people with skills

people[0]

,

people[1]

, and

people[3]

.

Return

any sufficient team of the smallest possible size, represented by the index of each person

. You may return the answer in

any order

.

It is

guaranteed

an answer exists.

Example 1:

Input:

req_skills = ["java","nodejs","reactjs"], people = [["java"],["nodejs"],["nodejs","reactjs"]]

Output:

[0,2]

Example 2:

Input:

req_skills = ["algorithms","math","java","reactjs","csharp","aws"], people = [["algorithms","math","java"],["algorithms","math","reactjs"],["java","csharp","aws"],["reactjs","csharp"],["csharp","math"],["aws","java"]]

Output:

[1,2]

Constraints:

1 <= req_skills.length <= 16

1 <= req_skills[i].length <= 16

req_skills[i]

consists of lowercase English letters.

All the strings of

req_skills

are

unique

.

1 <= people.length <= 60

0 <= people[i].length <= 16

1 <= people[i][j].length <= 16

people[i][j]

consists of lowercase English letters.

All the strings of

people[i]

are

unique

.

Every skill in

people[i]

is a skill in

req_skills

.

It is guaranteed a sufficient team exists.

Code Snippets

C++:

```
class Solution {
public:
vector<int> smallestSufficientTeam(vector<string>& req_skills,
vector<vector<string>>& people) {

}
};
```

Java:

```
class Solution {
public int[] smallestSufficientTeam(String[] req_skills, List<List<String>>
people) {

}
}
```

Python3:

```
class Solution:
def smallestSufficientTeam(self, req_skills: List[str], people:
List[List[str]]) -> List[int]:
```

Python:

```
class Solution(object):
def smallestSufficientTeam(self, req_skills, people):
"""
:type req_skills: List[str]
:type people: List[List[str]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {string[]} req_skills
* @param {string[][]} people
* @return {number[]}
*/
var smallestSufficientTeam = function(req_skills, people) {

};
```

TypeScript:

```
function smallestSufficientTeam(req_skills: string[], people: string[][]):
number[] {

};
```

C#:

```
public class Solution {
public int[] SmallestSufficientTeam(string[] req_skills, IList<IList<string>>
people) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* smallestSufficientTeam(char** req_skills, int req_skillsSize, char***
people, int peopleSize, int* peopleColSize, int* returnSize) {

}
```

Go:

```
func smallestSufficientTeam(req_skills []string, people [][]string) []int {

}
```

Kotlin:

```
class Solution {
fun smallestSufficientTeam(req_skills: Array<String>, people:
List<List<String>>): IntArray {

}
}
```

Swift:

```
class Solution {
func smallestSufficientTeam(_ req_skills: [String], _ people: [[String]]) ->
[Int] {

}
}
```

Rust:

```
impl Solution {
pub fn smallest_sufficient_team(req_skills: Vec<String>, people:
Vec<Vec<String>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {String[]} req_skills
# @param {String[][]} people
# @return {Integer[]}
def smallest_sufficient_team(req_skills, people)

end
```

PHP:

```
class Solution {

/**
* @param String[] $req_skills
* @param String[][] $people
* @return Integer[]
*/
function smallestSufficientTeam($req_skills, $people) {

}
}
```

Dart:

```
class Solution {
List<int> smallestSufficientTeam(List<String> req_skills, List<List<String>>
people) {

}
}
```

Scala:

```
object Solution {
def smallestSufficientTeam(req_skills: Array[String], people:
List[List[String]]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec smallest_sufficient_team(req_skills :: [String.t], people ::
[[String.t]]) :: [integer]
def smallest_sufficient_team(req_skills, people) do

end
end
```

Erlang:

```
-spec smallest_sufficient_team(Req_skills :: [unicode:unicode_binary()],
People :: [[unicode:unicode_binary()]]) -> [integer()].
smallest_sufficient_team(Req_skills, People) ->
.
```

Racket:

```
(define/contract (smallest-sufficient-team req_skills people)
(-> (listof string?) (listof (listof string?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Smallest Sufficient Team
* Difficulty: Hard
* Tags: array, string, dp, math
*
* 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:
vector<int> smallestSufficientTeam(vector<string>& req_skills,
vector<vector<string>>& people) {

}
};
```

Java Solution:

```
/**
* Problem: Smallest Sufficient Team
* Difficulty: Hard
* Tags: array, string, dp, math
*
* 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 int[] smallestSufficientTeam(String[] req_skills, List<List<String>>
people) {

}
}
```

Python3 Solution:

```
"""
Problem: Smallest Sufficient Team
Difficulty: Hard
Tags: array, string, dp, math

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 smallestSufficientTeam(self, req_skills: List[str], people:
List[List[str]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def smallestSufficientTeam(self, req_skills, people):
"""
:type req_skills: List[str]
:type people: List[List[str]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Smallest Sufficient Team
* Difficulty: Hard
* Tags: array, string, dp, math
*
* 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 {string[]} req_skills
* @param {string[][]} people
* @return {number[]}
*/
var smallestSufficientTeam = function(req_skills, people) {

};
```

TypeScript Solution:

```
/**
* Problem: Smallest Sufficient Team
* Difficulty: Hard
* Tags: array, string, dp, math
*
* 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 smallestSufficientTeam(req_skills: string[], people: string[][]):
number[] {

};
```

C# Solution:

```
/*
* Problem: Smallest Sufficient Team
* Difficulty: Hard
* Tags: array, string, dp, math
*
* 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 int[] SmallestSufficientTeam(string[] req_skills, IList<IList<string>>
people) {

}
}
```

C Solution:

```
/*
* Problem: Smallest Sufficient Team
* Difficulty: Hard
* Tags: array, string, dp, math
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* smallestSufficientTeam(char** req_skills, int req_skillsSize, char***
people, int peopleSize, int* peopleColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Smallest Sufficient Team
// Difficulty: Hard
// Tags: array, string, dp, math
//
// 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 smallestSufficientTeam(req_skills []string, people [][]string) []int {

}
```

Kotlin Solution:

```
class Solution {
fun smallestSufficientTeam(req_skills: Array<String>, people:
List<List<String>>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func smallestSufficientTeam(_ req_skills: [String], _ people: [[String]]) ->
[Int] {
```

```
}
}
```

Rust Solution:

```
// Problem: Smallest Sufficient Team
// Difficulty: Hard
// Tags: array, string, dp, math
//
// 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 smallest_sufficient_team(req_skills: Vec<String>, people:
Vec<Vec<String>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {String[]} req_skills
# @param {String[][]} people
# @return {Integer[]}
def smallest_sufficient_team(req_skills, people)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $req_skills
* @param String[][] $people
* @return Integer[]
*/
function smallestSufficientTeam($req_skills, $people) {

}
```

```
}
```

Dart Solution:

```
class Solution {
List<int> smallestSufficientTeam(List<String> req_skills, List<List<String>>
people) {

}
}
```

Scala Solution:

```
object Solution {
def smallestSufficientTeam(req_skills: Array[String], people:
List[List[String]]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec smallest_sufficient_team(req_skills :: [String.t], people ::
[[String.t]]) :: [integer]
def smallest_sufficient_team(req_skills, people) do

end
end
```

Erlang Solution:

```
-spec smallest_sufficient_team(Req_skills :: [unicode:unicode_binary()],
People :: [[unicode:unicode_binary()]]) -> [integer()].
smallest_sufficient_team(Req_skills, People) ->
.
```

Racket Solution:

```
(define/contract (smallest-sufficient-team req_skills people)
(-> (listof string?) (listof (listof string?)) (listof exact-integer?))
```

```
)
```