Problem 1258: Synonymous Sentences

Problem Information

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

Problem Description

You are given a list of equivalent string pairs

synonyms

where

synonyms[i] = [s

i

, t

i

]

indicates that

s

i

and

t

i

are equivalent strings. You are also given a sentence

text

.

Return

all possible synonymous sentences

sorted lexicographically

.

Example 1:

Input:

synonyms = [["happy","joy"],["sad","sorrow"],["joy","cheerful"]], text = "I am happy today but was sad yesterday"

Output:

["I am cheerful today but was sad yesterday","I am cheerful today but was sorrow yesterday","I am happy today but was sad yesterday","I am happy today but was sorrow yesterday","I am joy today but was sad yesterday","I am joy today but was sorrow yesterday"]

Example 2:

Input:

synonyms = [["happy","joy"],["cheerful","glad"]], text = "I am happy today but was sad yesterday"

Output:

["I am happy today but was sad yesterday","I am joy today but was sad yesterday"]

Constraints:

0 <= synonyms.length <= 10

synonyms[i].length == 2

1 <= s

i

.length,

t

i

.length <= 10

s

i

!= t

i

text

consists of at most

10

words.

All the pairs of

synonyms

are

unique

.

The words of

text

are separated by single spaces.

Code Snippets

C++:

```
class Solution {
public:
vector<string> generateSentences(vector<vector<string>>& synonyms, string
text) {

}
};
```

Java:

```
class Solution {
public List<String> generateSentences(List<List<String>> synonyms, String
text) {

}
}
```

Python3:

```
class Solution:
def generateSentences(self, synonyms: List[List[str]], text: str) ->
List[str]:
```

Python:

```
class Solution(object):
def generateSentences(self, synonyms, text):
"""
:type synonyms: List[List[str]]
:type text: str
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string[][]} synonyms
* @param {string} text
* @return {string[]}
*/
var generateSentences = function(synonyms, text) {

};
```

TypeScript:

```
function generateSentences(synonyms: string[][], text: string): string[] {

};
```

C#:

```
public class Solution {
public IList<string> GenerateSentences(IList<IList<string>> synonyms, string
text) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** generateSentences(char*** synonyms, int synonymsSize, int*
synonymsColSize, char* text, int* returnSize) {

}
```

Go:

```
func generateSentences(synonyms [][]string, text string) []string {

}
```

Kotlin:

```
class Solution {
fun generateSentences(synonyms: List<List<String>>, text: String):
List<String> {

}
}
```

Swift:

```
class Solution {
func generateSentences(_ synonyms: [[String]], _ text: String) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn generate_sentences(synonyms: Vec<Vec<String>>, text: String) ->
Vec<String> {

}
}
```

Ruby:

```
# @param {String[][]} synonyms
# @param {String} text
# @return {String[]}
def generate_sentences(synonyms, text)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $synonyms
* @param String $text
* @return String[]
*/
function generateSentences($synonyms, $text) {

}
}
```

Dart:

```
class Solution {
List<String> generateSentences(List<List<String>> synonyms, String text) {

}
}
```

Scala:

```
object Solution {
def generateSentences(synonyms: List[List[String]], text: String):
List[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec generate_sentences(synonyms :: [[String.t]], text :: String.t) ::
[String.t]
def generate_sentences(synonyms, text) do

end
end
```

Erlang:

```
-spec generate_sentences(Synonyms :: [[unicode:unicode_binary()]], Text ::
unicode:unicode_binary()) -> [unicode:unicode_binary()].
```

```
generate_sentences(Synonyms, Text) ->
.
```

Racket:

```
(define/contract (generate-sentences synonyms text)
(-> (listof (listof string?)) string? (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Synonymous Sentences
* Difficulty: Medium
* Tags: array, string, graph, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<string> generateSentences(vector<vector<string>>& synonyms, string
text) {

}
};
```

Java Solution:

```
/**
* Problem: Synonymous Sentences
* Difficulty: Medium
* Tags: array, string, graph, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) for hash map
*/

class Solution {
public List<String> generateSentences(List<List<String>> synonyms, String
text) {

}
}
```

Python3 Solution:

```
"""
Problem: Synonymous Sentences
Difficulty: Medium
Tags: array, string, graph, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

```
class Solution(object):
def generateSentences(self, synonyms, text):
"""
:type synonyms: List[List[str]]
:type text: str
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Synonymous Sentences
* Difficulty: Medium
* Tags: array, string, graph, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string[][]} synonyms
* @param {string} text
* @return {string[]}
*/
var generateSentences = function(synonyms, text) {

};
```

TypeScript Solution:

```
/**
* Problem: Synonymous Sentences
* Difficulty: Medium
* Tags: array, string, graph, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function generateSentences(synonyms: string[][], text: string): string[] {

};
```

C# Solution:

```
/*
* Problem: Synonymous Sentences
* Difficulty: Medium
* Tags: array, string, graph, hash, sort
*
```

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

public class Solution {
public IList<string> GenerateSentences(IList<IList<string>> synonyms, string
text) {

}
}
```

C Solution:

```
/*
* Problem: Synonymous Sentences
* Difficulty: Medium
* Tags: array, string, graph, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** generateSentences(char*** synonyms, int synonymsSize, int*
synonymsColSize, char* text, int* returnSize) {

}
```

Go Solution:

```
// Problem: Synonymous Sentences
// Difficulty: Medium
// Tags: array, string, graph, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map
```

```
func generateSentences(synonyms [][]string, text string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun generateSentences(synonyms: List<List<String>>, text: String):
List<String> {

}
}
```

Swift Solution:

```
class Solution {
func generateSentences(_ synonyms: [[String]], _ text: String) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Synonymous Sentences
// Difficulty: Medium
// Tags: array, string, graph, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn generate_sentences(synonyms: Vec<Vec<String>>, text: String) ->
Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String[][]} synonyms
# @param {String} text
# @return {String[]}
def generate_sentences(synonyms, text)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $synonyms
* @param String $text
* @return String[]
*/
function generateSentences($synonyms, $text) {

}
}
```

Dart Solution:

```
class Solution {
List<String> generateSentences(List<List<String>> synonyms, String text) {

}
}
```

Scala Solution:

```
object Solution {
def generateSentences(synonyms: List[List[String]], text: String):
List[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec generate_sentences(synonyms :: [[String.t]], text :: String.t) ::
```

```
[String.t]
def generate_sentences(synonyms, text) do

end
end
```

Erlang Solution:

```
-spec generate_sentences(Synonyms :: [[unicode:unicode_binary()]], Text ::
unicode:unicode_binary()) -> [unicode:unicode_binary()].
generate_sentences(Synonyms, Text) ->
.
```

Racket Solution:

```
(define/contract (generate-sentences synonyms text)
(-> (listof (listof string?)) string? (listof string?))
)
```