Problem 2325: Decode the Message

Problem Information

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

Problem Description

You are given the strings

key

and

message

, which represent a cipher key and a secret message, respectively. The steps to decode

message

are as follows:

Use the

first

appearance of all 26 lowercase English letters in

key

as the

order

of the substitution table.

Align the substitution table with the regular English alphabet.

Each letter in

message

is then

substituted

using the table.

Spaces

' '

are transformed to themselves.

For example, given

key = "

hap

p

y

bo

y"

(actual key would have

at least one

instance of each letter in the alphabet), we have the partial substitution table of (

'h' -> 'a'

,

'a' -> 'b'

,

'p' -> 'c'

,

'y' -> 'd'

,

'b' -> 'e'

,

'o' -> 'f'

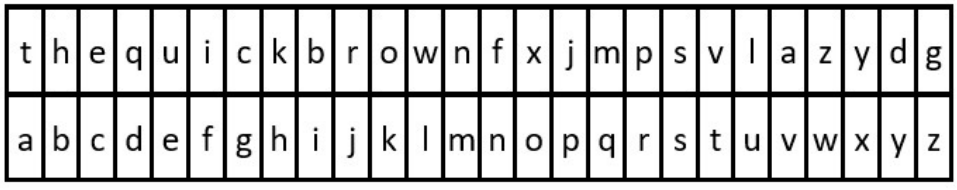
).

Return

the decoded message

.

Example 1:



Input:

key = "the quick brown fox jumps over the lazy dog", message = "vkbs bs t suepuv"

Output:

"this is a secret"

Explanation:

The diagram above shows the substitution table. It is obtained by taking the first appearance of each letter in "

the

quick

brown

f

o

x

j

u

mps

o

v

er the

lazy

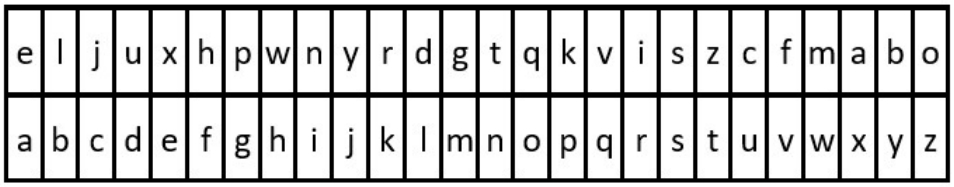
d

o

g

".

Example 2:



Input:

key = "eljuxhpwnyrdgtqkviszcfmabo", message = "zwx hnfx lqantp mnoeius ycgk vcnjrdb"

Output:

"the five boxing wizards jump quickly"

Explanation:

The diagram above shows the substitution table. It is obtained by taking the first appearance of each letter in "

eljuxhpwnyrdgtqkviszcfmabo

".

Constraints:

26 <= key.length <= 2000

key

consists of lowercase English letters and

' '

.

key

contains every letter in the English alphabet (

'a'

to

'z'

)

at least once

.

1 <= message.length <= 2000

message

consists of lowercase English letters and

' '

.

Code Snippets

C++:

```
class Solution {
public:
string decodeMessage(string key, string message) {

}
};
```

Java:

```
class Solution {
public String decodeMessage(String key, String message) {

}
}
```

Python3:

```
class Solution:
def decodeMessage(self, key: str, message: str) -> str:
```

Python:

```
class Solution(object):
def decodeMessage(self, key, message):
"""
:type key: str
:type message: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} key
* @param {string} message
* @return {string}
*/
var decodeMessage = function(key, message) {

};
```

TypeScript:

```
function decodeMessage(key: string, message: string): string {

};
```

C#:

```
public class Solution {
public string DecodeMessage(string key, string message) {

}
}
```

C:

```
char* decodeMessage(char* key, char* message) {

}
```

Go:

```
func decodeMessage(key string, message string) string {

}
```

Kotlin:

```
class Solution {
fun decodeMessage(key: String, message: String): String {

}
}
```

Swift:

```
class Solution {
func decodeMessage(_ key: String, _ message: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn decode_message(key: String, message: String) -> String {

}
}
```

Ruby:

```
# @param {String} key
# @param {String} message
# @return {String}
def decode_message(key, message)

end
```

PHP:

```
class Solution {

/**
* @param String $key
* @param String $message
* @return String
*/
function decodeMessage($key, $message) {

}
}
```

Dart:

```
class Solution {
String decodeMessage(String key, String message) {

}
}
```

Scala:

```
object Solution {
def decodeMessage(key: String, message: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec decode_message(key :: String.t, message :: String.t) :: String.t
def decode_message(key, message) do
```

```
end
end
```

Erlang:

```
-spec decode_message(Key :: unicode:unicode_binary(), Message ::
unicode:unicode_binary()) -> unicode:unicode_binary().
decode_message(Key, Message) ->
.
```

Racket:

```
(define/contract (decode-message key message)
(-> string? string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Decode the Message
* Difficulty: Easy
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
string decodeMessage(string key, string message) {

}
};
```

Java Solution:

```
/**
* Problem: Decode the Message
* Difficulty: Easy
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public String decodeMessage(String key, String message) {

}
}
```

Python3 Solution:

```
"""
Problem: Decode the Message
Difficulty: Easy
Tags: string, tree, hash

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def decodeMessage(self, key: str, message: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def decodeMessage(self, key, message):
"""
:type key: str
:type message: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Decode the Message
* Difficulty: Easy
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {string} key
* @param {string} message
* @return {string}
*/
var decodeMessage = function(key, message) {

};
```

TypeScript Solution:

```
/**
* Problem: Decode the Message
* Difficulty: Easy
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function decodeMessage(key: string, message: string): string {

};
```

C# Solution:

```
/*
* Problem: Decode the Message
* Difficulty: Easy
```

```
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* 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 DecodeMessage(string key, string message) {

}
}
```

C Solution:

```
/*
* Problem: Decode the Message
* Difficulty: Easy
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

char* decodeMessage(char* key, char* message) {

}
```

Go Solution:

```
// Problem: Decode the Message
// Difficulty: Easy
// Tags: string, tree, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func decodeMessage(key string, message string) string {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun decodeMessage(key: String, message: String): String {

}
}
```

Swift Solution:

```
class Solution {
func decodeMessage(_ key: String, _ message: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Decode the Message
// Difficulty: Easy
// Tags: string, tree, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn decode_message(key: String, message: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} key
# @param {String} message
# @return {String}
def decode_message(key, message)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $key
* @param String $message
* @return String
*/
function decodeMessage($key, $message) {

}
}
```

Dart Solution:

```
class Solution {
String decodeMessage(String key, String message) {

}
}
```

Scala Solution:

```
object Solution {
def decodeMessage(key: String, message: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec decode_message(key :: String.t, message :: String.t) :: String.t
def decode_message(key, message) do

end
end
```

Erlang Solution:

```
-spec decode_message(Key :: unicode:unicode_binary(), Message ::
unicode:unicode_binary()) -> unicode:unicode_binary().
decode_message(Key, Message) ->
.
```

Racket Solution:

```
(define/contract (decode-message key message)
(-> string? string? string?)
)
```