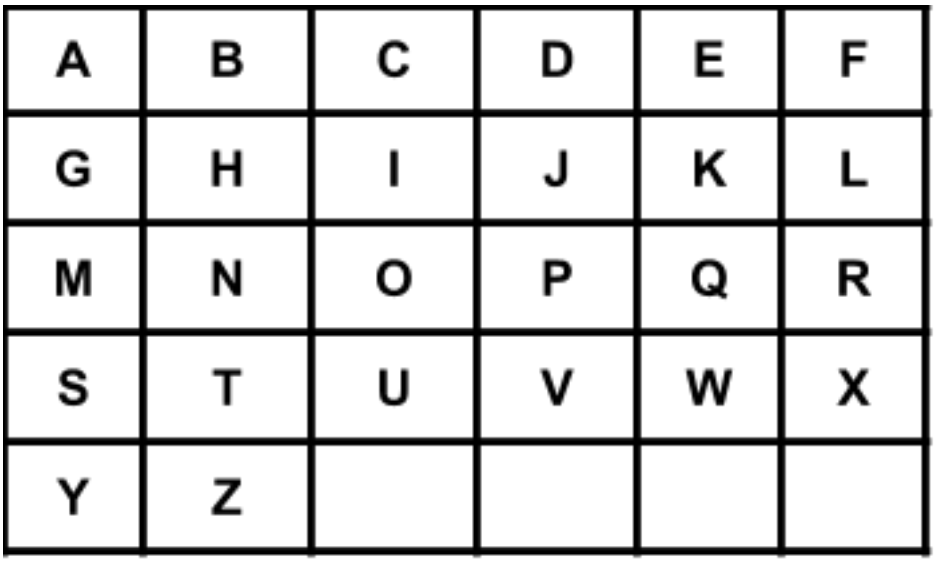
Problem 1320: Minimum Distance to Type a Word Using Two Fingers

Problem Information

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

Problem Description



You have a keyboard layout as shown above in the

X-Y

plane, where each English uppercase letter is located at some coordinate.

For example, the letter

'A'

is located at coordinate

(0, 0)

, the letter

'B'

is located at coordinate

(0, 1)

, the letter

'P'

is located at coordinate

(2, 3)

and the letter

'Z'

is located at coordinate

(4, 1)

.

Given the string

word

, return

the minimum total

distance

to type such string using only two fingers

.

The

distance

between coordinates

(x

1

, y

1

)

and

(x

2

, y

2

)

is

|x

1

- x

2

| + |y

1

- y

2

|

.

Note

that the initial positions of your two fingers are considered free so do not count towards your total distance, also your two fingers do not have to start at the first letter or the first two letters.

Example 1:

Input:

word = "CAKE"

Output:

3

Explanation:

Using two fingers, one optimal way to type "CAKE" is: Finger 1 on letter 'C' -> cost = 0 Finger 1 on letter 'A' -> cost = Distance from letter 'C' to letter 'A' = 2 Finger 2 on letter 'K' -> cost = 0 Finger 2 on letter 'E' -> cost = Distance from letter 'K' to letter 'E' = 1 Total distance = 3

Example 2:

Input:

word = "HAPPY"

Output:

6

Explanation:

Using two fingers, one optimal way to type "HAPPY" is: Finger 1 on letter 'H' -> cost = 0 Finger 1 on letter 'A' -> cost = Distance from letter 'H' to letter 'A' = 2 Finger 2 on letter 'P' -> cost = 0 Finger 2 on letter 'P' -> cost = Distance from letter 'P' to letter 'P' = 0 Finger 1 on letter 'Y' -> cost = Distance from letter 'A' to letter 'Y' = 4 Total distance = 6

Constraints:

2 <= word.length <= 300

word

consists of uppercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int minimumDistance(string word) {

}
};
```

Java:

```
class Solution {
public int minimumDistance(String word) {

}
}
```

Python3:

```
class Solution:
def minimumDistance(self, word: str) -> int:
```

Python:

```
class Solution(object):
def minimumDistance(self, word):
"""
:type word: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} word
* @return {number}
*/
var minimumDistance = function(word) {

};
```

TypeScript:

```
function minimumDistance(word: string): number {

};
```

C#:

```
public class Solution {
public int MinimumDistance(string word) {

}
}
```

C:

```
int minimumDistance(char* word) {

}
```

Go:

```
func minimumDistance(word string) int {

}
```

Kotlin:

```
class Solution {
fun minimumDistance(word: String): Int {

}
}
```

Swift:

```
class Solution {
func minimumDistance(_ word: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_distance(word: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} word
# @return {Integer}
def minimum_distance(word)

end
```

PHP:

```
class Solution {

/**
```

```
* @param String $word
* @return Integer
*/
function minimumDistance($word) {

}
}
```

Dart:

```
class Solution {
int minimumDistance(String word) {

}
}
```

Scala:

```
object Solution {
def minimumDistance(word: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_distance(word :: String.t) :: integer
def minimum_distance(word) do

end
end
```

Erlang:

```
-spec minimum_distance(Word :: unicode:unicode_binary()) -> integer().
minimum_distance(Word) ->
.
```

Racket:

```
(define/contract (minimum-distance word)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Distance to Type a Word Using Two Fingers
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int minimumDistance(string word) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Distance to Type a Word Using Two Fingers
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int minimumDistance(String word) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Minimum Distance to Type a Word Using Two Fingers
Difficulty: Hard
Tags: string, dp

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def minimumDistance(self, word: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumDistance(self, word):
"""
:type word: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Distance to Type a Word Using Two Fingers
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
```

```
* @param {string} word
* @return {number}
*/
var minimumDistance = function(word) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Distance to Type a Word Using Two Fingers
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function minimumDistance(word: string): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Distance to Type a Word Using Two Fingers
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* 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 MinimumDistance(string word) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Distance to Type a Word Using Two Fingers
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int minimumDistance(char* word) {

}
```

Go Solution:

```
// Problem: Minimum Distance to Type a Word Using Two Fingers
// Difficulty: Hard
// Tags: string, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func minimumDistance(word string) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumDistance(word: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumDistance(_ word: String) -> Int {
```

```
}
}
```

Rust Solution:

```
// Problem: Minimum Distance to Type a Word Using Two Fingers
// Difficulty: Hard
// Tags: string, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn minimum_distance(word: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} word
# @return {Integer}
def minimum_distance(word)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $word
* @return Integer
*/
function minimumDistance($word) {

}
}
```

Dart Solution:

```
class Solution {
int minimumDistance(String word) {

}
}
```

Scala Solution:

```
object Solution {
def minimumDistance(word: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_distance(word :: String.t) :: integer
def minimum_distance(word) do

end
end
```

Erlang Solution:

```
-spec minimum_distance(Word :: unicode:unicode_binary()) -> integer().
minimum_distance(Word) ->
.
```

Racket Solution:

```
(define/contract (minimum-distance word)
(-> string? exact-integer?)
)
```