Problem 3016: Minimum Number of Pushes to Type Word II

Problem Information

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

Problem Description

You are given a string

word

containing lowercase English letters.

Telephone keypads have keys mapped with

distinct

collections of lowercase English letters, which can be used to form words by pushing them. For example, the key

2

is mapped with

["a","b","c"]

, we need to push the key one time to type

"a"

, two times to type

"b"

, and three times to type

"c"

.

It is allowed to remap the keys numbered

2

to

9

to

distinct

collections of letters. The keys can be remapped to

any

amount of letters, but each letter

must

be mapped to

exactly

one key. You need to find the

minimum

number of times the keys will be pushed to type the string

word

.

Return

the

minimum

number of pushes needed to type

word

after remapping the keys

.

An example mapping of letters to keys on a telephone keypad is given below. Note that

1

,

*

,

#

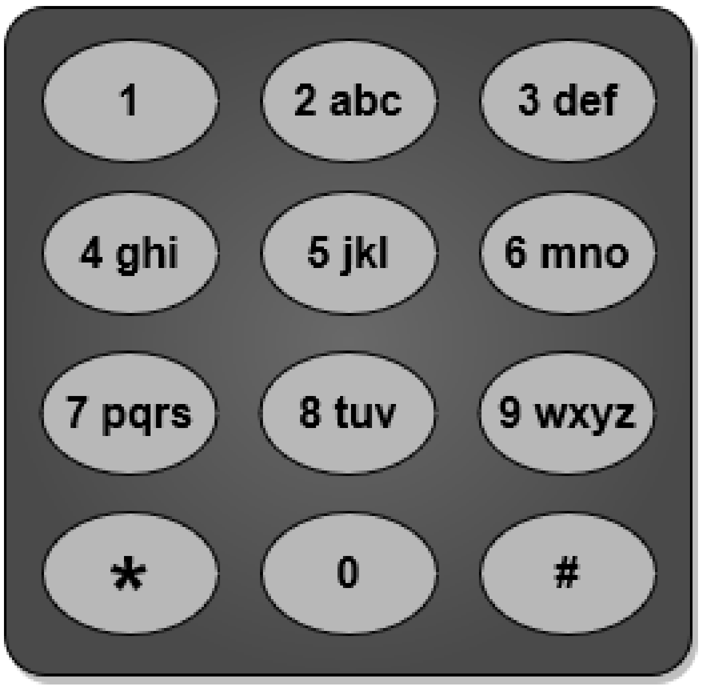
, and

0

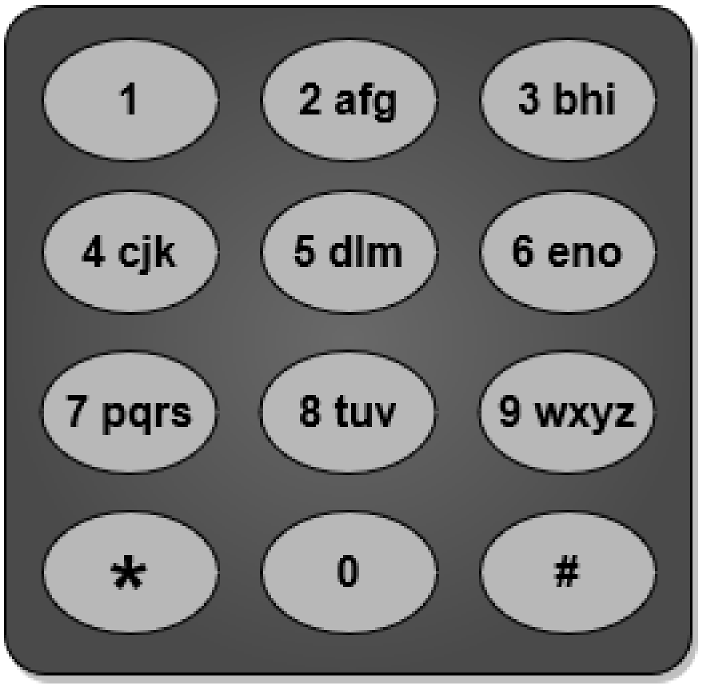
do

not

map to any letters.


1
2 abc
3 def
4 ghi
5 jkl
6 mno
7 pqrs
8 tuv
9 wxyz
*
0
#

Example 1:


1
2 afg
3 bhi
4 cjk
5 dlm
6 eno
7 pqrs
8 tuv
9 wxyz
*
0
#

Input:

word = "abcde"

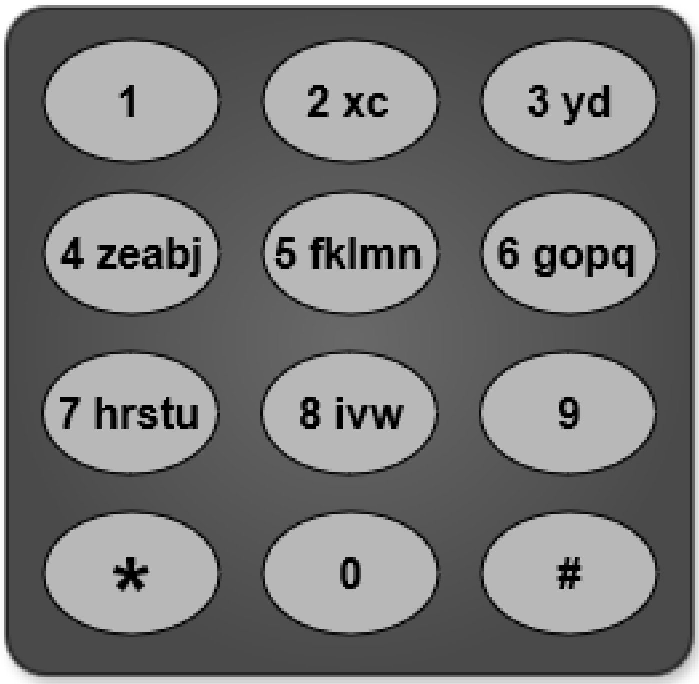
Output:

5

Explanation:

The remapped keypad given in the image provides the minimum cost. "a" -> one push on key 2 "b" -> one push on key 3 "c" -> one push on key 4 "d" -> one push on key 5 "e" -> one push on key 6 Total cost is 1 + 1 + 1 + 1 + 1 = 5. It can be shown that no other mapping can provide a lower cost.

Example 2:



Input:

word = "xyzxyzxyzxyz"

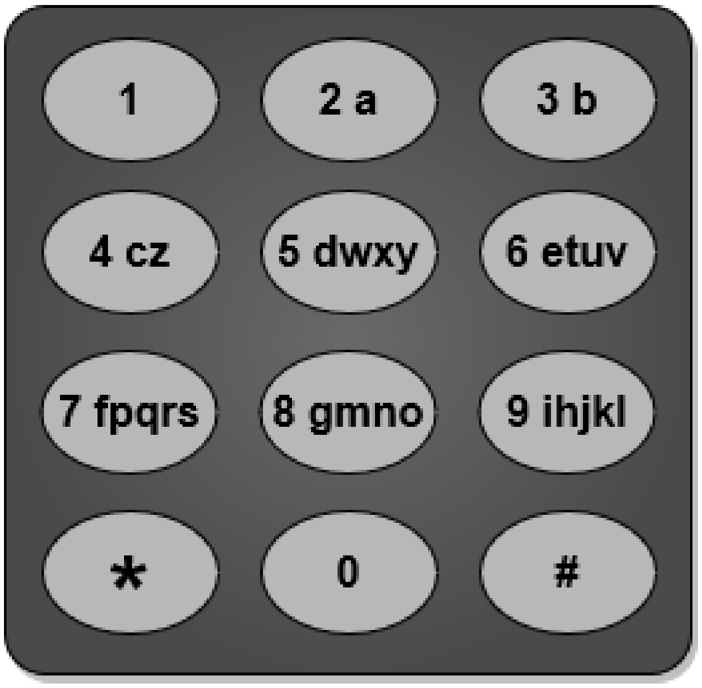
Output:

12

Explanation:

The remapped keypad given in the image provides the minimum cost. "x" -> one push on key 2 "y" -> one push on key 3 "z" -> one push on key 4 Total cost is 1 * 4 + 1 * 4 + 1 * 4 = 12 It can be shown that no other mapping can provide a lower cost. Note that the key 9 is not mapped to any letter: it is not necessary to map letters to every key, but to map all the letters.

Example 3:



Input:

word = "aabbccddeeffgghhiiiiii"

Output:

24

Explanation:

The remapped keypad given in the image provides the minimum cost. "a" -> one push on key 2 "b" -> one push on key 3 "c" -> one push on key 4 "d" -> one push on key 5 "e" -> one push on key 6 "f" -> one push on key 7 "g" -> one push on key 8 "h" -> two pushes on key 9 "i" -> one push on key 9 Total cost is 1 * 2 + 1 * 2 + 1 * 2 + 1 * 2 + 1 * 2 + 1 * 2 + 1 * 2 + 2 * 2 + 6 * 1 = 24. It can be shown that no other mapping can provide a lower cost.

Constraints:

1 <= word.length <= 10

5

word

consists of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int minimumPushes(char* word) {

}
```

Go:

```
func minimumPushes(word string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Pushes to Type Word II
* Difficulty: Medium
* Tags: string, greedy, hash, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Pushes to Type Word II
* Difficulty: Medium
* Tags: string, greedy, hash, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int minimumPushes(String word) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Pushes to Type Word II
Difficulty: Medium
Tags: string, greedy, hash, sort

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

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Number of Pushes to Type Word II
* Difficulty: Medium
* Tags: string, greedy, hash, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

TypeScript Solution:

```
/**
* Problem: Minimum Number of Pushes to Type Word II
* Difficulty: Medium
* Tags: string, greedy, hash, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function minimumPushes(word: string): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Pushes to Type Word II
* Difficulty: Medium
* Tags: string, greedy, hash, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int MinimumPushes(string word) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Pushes to Type Word II
* Difficulty: Medium
* Tags: string, greedy, hash, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int minimumPushes(char* word) {

}
```

Go Solution:

```
// Problem: Minimum Number of Pushes to Type Word II
// Difficulty: Medium
// Tags: string, greedy, hash, sort
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func minimumPushes(word string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Pushes to Type Word II
// Difficulty: Medium
// Tags: string, greedy, hash, sort
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int minimumPushes(String word) {
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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