

Problem 17: Letter Combinations of a Phone Number

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given a string containing digits from

2-9

inclusive, return all possible letter combinations that the number could represent. Return the answer in

any order

.

A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.



Example 1:

Input:

digits = "23"

Output:

["ad","ae","af","bd","be","bf","cd","ce","cf"]

Example 2:

Input:

digits = "2"

Output:

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

Constraints:

1 <= digits.length <= 4

digits[i]

is a digit in the range

['2', '9']

.

Code Snippets

C++:

```
class Solution {
public:
    vector<string> letterCombinations(string digits) {

    }
};
```

Java:

```
class Solution {
    public List<String> letterCombinations(String digits) {

    }
}
```

Python3:

```
class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
```

Python:

```
class Solution(object):
    def letterCombinations(self, digits):
        """
        :type digits: str
```

```
:rtype: List[str]
"""
```

JavaScript:

```
/**
 * @param {string} digits
 * @return {string[]}
 */
var letterCombinations = function(digits) {

};
```

TypeScript:

```
function letterCombinations(digits: string): string[] {

};
```

C#:

```
public class Solution {
    public IList<string> LetterCombinations(string digits) {

    }
}
```

C:

```
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
char** letterCombinations(char* digits, int* returnSize) {

}
```

Go:

```
func letterCombinations(digits string) []string {

}
```

Kotlin:

```
class Solution {  
    fun letterCombinations(digits: String): List<String> {  
  
    }  
}
```

Swift:

```
class Solution {  
    func letterCombinations(_ digits: String) -> [String] {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn letter_combinations(digits: String) -> Vec<String> {  
  
    }  
}
```

Ruby:

```
# @param {String} digits  
# @return {String[]}  
def letter_combinations(digits)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param String $digits  
     * @return String[]  
     */  
    function letterCombinations($digits) {  
  
    }  
}
```

```
}
```

Dart:

```
class Solution {  
  List<String> letterCombinations(String digits) {  
  
  }  
}
```

Scala:

```
object Solution {  
  def letterCombinations(digits: String): List[String] = {  
  
  }  
}
```

Elixir:

```
defmodule Solution do  
  @spec letter_combinations(digits :: String.t) :: [String.t]  
  def letter_combinations(digits) do  
  
  end  
end
```

Erlang:

```
-spec letter_combinations(Digits :: unicode:unicode_binary()) ->  
[unicode:unicode_binary()].  
letter_combinations(Digits) ->  
.
```

Racket:

```
(define/contract (letter-combinations digits)  
  (-> string? (listof string?))  
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Letter Combinations of a Phone Number
 * Difficulty: Medium
 * Tags: string, hash
 *
 * 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:
    vector<string> letterCombinations(string digits) {

    }
};
```

Java Solution:

```
/**
 * Problem: Letter Combinations of a Phone Number
 * Difficulty: Medium
 * Tags: string, hash
 *
 * 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 List<String> letterCombinations(String digits) {

    }
}
```

Python3 Solution:

```
"""
Problem: Letter Combinations of a Phone Number
```

Difficulty: Medium

Tags: string, hash

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 letterCombinations(self, digits: str) -> List[str]:
```

```
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
```

```
def letterCombinations(self, digits):
```

```
"""
```

```
:type digits: str
```

```
:rtype: List[str]
```

```
"""
```

JavaScript Solution:

```
/**
```

```
 * Problem: Letter Combinations of a Phone Number
```

```
 * Difficulty: Medium
```

```
 * Tags: string, hash
```

```
 *
```

```
 * 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} digits
```

```
 * @return {string[]}
```

```
 */
```

```
var letterCombinations = function(digits) {
```

```
};
```

TypeScript Solution:

```
/**
 * Problem: Letter Combinations of a Phone Number
 * Difficulty: Medium
 * Tags: string, hash
 *
 * 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 letterCombinations(digits: string): string[] {

};
```

C# Solution:

```
/*
 * Problem: Letter Combinations of a Phone Number
 * Difficulty: Medium
 * Tags: string, hash
 *
 * 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 IList<string> LetterCombinations(string digits) {

    }
}
```

C Solution:

```
/*
 * Problem: Letter Combinations of a Phone Number
 * Difficulty: Medium
 * Tags: string, hash
 *
 * 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** letterCombinations(char* digits, int* returnSize) {

}

```

Go Solution:

```

// Problem: Letter Combinations of a Phone Number
// Difficulty: Medium
// Tags: string, hash
//
// 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 letterCombinations(digits string) []string {

}

```

Kotlin Solution:

```

class Solution {
    fun letterCombinations(digits: String): List<String> {

    }
}

```

Swift Solution:

```

class Solution {
    func letterCombinations(_ digits: String) -> [String] {

    }
}

```

Rust Solution:

```
// Problem: Letter Combinations of a Phone Number
// Difficulty: Medium
// Tags: string, hash
//
// 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 letter_combinations(digits: String) -> Vec<String> {

    }
}
```

Ruby Solution:

```
# @param {String} digits
# @return {String[]}
def letter_combinations(digits)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param String $digits
     * @return String[]
     */
    function letterCombinations($digits) {

    }
}
```

Dart Solution:

```
class Solution {
    List<String> letterCombinations(String digits) {
```

```
}  
}
```

Scala Solution:

```
object Solution {  
  def letterCombinations(digits: String): List[String] = {  
  
  }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec letter_combinations(digits :: String.t) :: [String.t]  
  def letter_combinations(digits) do  
  
  end  
end
```

Erlang Solution:

```
-spec letter_combinations(Digits :: unicode:unicode_binary()) ->  
[unicode:unicode_binary()].  
letter_combinations(Digits) ->  
.
```

Racket Solution:

```
(define/contract (letter-combinations digits)  
  (-> string? (listof string?))  
)
```