

Problem 40: Combination Sum II

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given a collection of candidate numbers (

candidates

) and a target number (

target

), find all unique combinations in

candidates

where the candidate numbers sum to

target

.

Each number in

candidates

may only be used

once

in the combination.

Note:

The solution set must not contain duplicate combinations.

Example 1:

Input:

candidates = [10,1,2,7,6,1,5], target = 8

Output:

[[1,1,6], [1,2,5], [1,7], [2,6]]

Example 2:

Input:

candidates = [2,5,2,1,2], target = 5

Output:

[[1,2,2], [5]]

Constraints:

$1 \leq \text{candidates.length} \leq 100$

$1 \leq \text{candidates}[i] \leq 50$

$1 \leq \text{target} \leq 30$

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {

}
};
```

Java:

```
class Solution {
public List<List<Integer>> combinationSum2(int[] candidates, int target) {

}
}
```

Python3:

```
class Solution:
def combinationSum2(self, candidates: List[int], target: int) ->
List[List[int]]:
```

Python:

```
class Solution(object):
def combinationSum2(self, candidates, target):
"""
:type candidates: List[int]
:type target: int
:rtype: List[List[int]]
"""

"
```

JavaScript:

```
/**
 * @param {number[]} candidates
 * @param {number} target
 * @return {number[][]}
 */
var combinationSum2 = function(candidates, target) {

};
```

TypeScript:

```
function combinationSum2(candidates: number[], target: number): number[][] {  
}  
};
```

C#:

```
public class Solution {  
    public IList<IList<int>> CombinationSum2(int[] candidates, int target) {  
        return null;  
    }  
}
```

C:

```
/**  
 * Return an array of arrays of size *returnSize.  
 * The sizes of the arrays are returned as *returnColumnSizes array.  
 * Note: Both returned array and *columnSizes array must be malloced, assume  
 caller calls free().  
 */  
int** combinationSum2(int* candidates, int candidatesSize, int target, int*  
returnSize, int** returnColumnSizes) {  
  
}
```

Go:

```
func combinationSum2(candidates []int, target int) [][]int {  
  
}
```

Kotlin:

```
class Solution {  
    fun combinationSum2(candidates: IntArray, target: Int): List<List<Int>> {  
        return emptyList()  
    }  
}
```

Swift:

```
class Solution {  
    func combinationSum2(_ candidates: [Int], _ target: Int) -> [[Int]] {  
        return []  
    }  
}
```

```
}
```

```
}
```

Rust:

```
impl Solution {
    pub fn combination_sum2(candidates: Vec<i32>, target: i32) -> Vec<Vec<i32>> {
        }
    }
```

Ruby:

```
# @param {Integer[]} candidates
# @param {Integer} target
# @return {Integer[][]}
def combination_sum2(candidates, target)

end
```

PHP:

```
class Solution {

    /**
     * @param Integer[] $candidates
     * @param Integer $target
     * @return Integer[][]
     */
    function combinationSum2($candidates, $target) {

    }
}
```

Dart:

```
class Solution {
    List<List<int>> combinationSum2(List<int> candidates, int target) {
        }
    }
```

Scala:

```
object Solution {  
    def combinationSum2(candidates: Array[Int], target: Int): List[List[Int]] = {  
        }  
        }  
}
```

Elixir:

```
defmodule Solution do  
    @spec combination_sum2(candidates :: [integer], target :: integer) ::  
        [[integer]]  
    def combination_sum2(candidates, target) do  
  
        end  
        end
```

Erlang:

```
-spec combination_sum2(Candidates :: [integer()], Target :: integer()) ->  
    [[integer()]].  
combination_sum2(Candidates, Target) ->  
    .
```

Racket:

```
(define/contract (combination-sum2 candidates target)  
  (-> (listof exact-integer?) exact-integer? (listof (listof exact-integer?)))  
  )
```

Solutions

C++ Solution:

```
/*  
 * Problem: Combination Sum II  
 * Difficulty: Medium  
 * Tags: array  
 *  
 * Approach: Use two pointers or sliding window technique
```

```

* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

```

```

class Solution {
public:
vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {

}
};

```

Java Solution:

```

/**
 * Problem: Combination Sum II
 * Difficulty: Medium
 * Tags: array
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
*/

```

```

class Solution {
public List<List<Integer>> combinationSum2(int[] candidates, int target) {

}
}

```

Python3 Solution:

```

"""
Problem: Combination Sum II
Difficulty: Medium
Tags: array

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

```

```
class Solution:

def combinationSum2(self, candidates: List[int], target: int) ->
List[List[int]]:

# TODO: Implement optimized solution

pass
```

Python Solution:

```
class Solution(object):

def combinationSum2(self, candidates, target):
    """
:type candidates: List[int]
:type target: int
:rtype: List[List[int]]
"""


```

JavaScript Solution:

```
/**
 * Problem: Combination Sum II
 * Difficulty: Medium
 * Tags: array
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

/**
 * @param {number[]} candidates
 * @param {number} target
 * @return {number[][][]}
 */
var combinationSum2 = function(candidates, target) {

};
```

TypeScript Solution:

```
/**
 * Problem: Combination Sum II
```

```

* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
function combinationSum2(candidates: number[], target: number): number[][] {
}

```

C# Solution:

```

/*
* Problem: Combination Sum II
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
public class Solution {
    public IList<IList<int>> CombinationSum2(int[] candidates, int target) {
        return null;
    }
}

```

C Solution:

```

/*
* Problem: Combination Sum II
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

```

```

/**
 * Return an array of arrays of size *returnSize.
 * The sizes of the arrays are returned as *returnColumnSizes array.
 * Note: Both returned array and *columnSizes array must be malloced, assume
 caller calls free().
 */
int** combinationSum2(int* candidates, int candidatesSize, int target, int*
returnSize, int** returnColumnSizes) {

}

```

Go Solution:

```

// Problem: Combination Sum II
// Difficulty: Medium
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func combinationSum2(candidates []int, target int) [][]int {
}

```

Kotlin Solution:

```

class Solution {
    fun combinationSum2(candidates: IntArray, target: Int): List<List<Int>> {
        }
    }
}

```

Swift Solution:

```

class Solution {
    func combinationSum2(_ candidates: [Int], _ target: Int) -> [[Int]] {
        }
    }
}

```

Rust Solution:

```
// Problem: Combination Sum II
// Difficulty: Medium
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
    pub fn combination_sum2(candidates: Vec<i32>, target: i32) -> Vec<Vec<i32>> {
        }

    }
}
```

Ruby Solution:

```
# @param {Integer[]} candidates
# @param {Integer} target
# @return {Integer[][]}
def combination_sum2(candidates, target)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[] $candidates
     * @param Integer $target
     * @return Integer[][]
     */
    function combinationSum2($candidates, $target) {

    }
}
```

Dart Solution:

```
class Solution {  
    List<List<int>> combinationSum2(List<int> candidates, int target) {  
        }  
    }  
}
```

Scala Solution:

```
object Solution {  
    def combinationSum2(candidates: Array[Int], target: Int): List[List[Int]] = {  
        }  
    }
```

Elixir Solution:

```
defmodule Solution do  
    @spec combination_sum2(candidates :: [integer], target :: integer) ::  
        [[integer]]  
    def combination_sum2(candidates, target) do  
  
    end  
    end
```

Erlang Solution:

```
-spec combination_sum2(Candidates :: [integer()], Target :: integer()) ->  
    [[integer()]].  
combination_sum2(Candidates, Target) ->  
    .
```

Racket Solution:

```
(define/contract (combination-sum2 candidates target)  
  (-> (listof exact-integer?) exact-integer? (listof (listof exact-integer?)))  
)
```