Problem 39: Combination Sum

Problem Information

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

Problem Description

Given an array of

distinct

integers

candidates

and a target integer

target

, return

a list of all

unique combinations

of

candidates

where the chosen numbers sum to

target

.

You may return the combinations in

any order

.

The

same

number may be chosen from

candidates

an

unlimited number of times

. Two combinations are unique if the

frequency

of at least one of the chosen numbers is different.

The test cases are generated such that the number of unique combinations that sum up to

target

is less than

150

combinations for the given input.

Example 1:

Input:

candidates = [2,3,6,7], target = 7

Output:

[[2,2,3],[7]]

Explanation:

2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times. 7 is a candidate, and 7 = 7. These are the only two combinations.

Example 2:

Input:

candidates = [2,3,5], target = 8

Output:

[[2,2,2,2],[2,3,3],[3,5]]

Example 3:

Input:

candidates = [2], target = 1

Output:

[]

Constraints:

1 <= candidates.length <= 30

2 <= candidates[i] <= 40

All elements of

candidates

are

distinct

.

1 <= target <= 40

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def combinationSum(self, candidates, target):
"""
```

```
:type candidates: List[int]
:type target: int
:rtype: List[List[int]]
"""
```

JavaScript:

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

};
```

TypeScript:

```
function combinationSum(candidates: number[], target: number): number[][] {

};
```

C#:

```
public class Solution {
public IList<IList<int>> CombinationSum(int[] candidates, int target) {

}
}
```

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** combinationSum(int* candidates, int candidatesSize, int target, int*
returnSize, int** returnColumnSizes) {

}
```

Go:

```
func combinationSum(candidates []int, target int) [][]int {

}
```

Kotlin:

```
class Solution {
fun combinationSum(candidates: IntArray, target: Int): List<List<Int>> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {
```

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Combination Sum
* 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>> combinationSum(vector<int>& candidates, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Combination Sum
* 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>> combinationSum(int[] candidates, int target) {
}
}
```

Python3 Solution:

```
"""
Problem: Combination Sum
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 combinationSum(self, candidates: List[int], target: int) ->
List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Combination Sum
* 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 combinationSum = function(candidates, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Combination Sum
* 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 combinationSum(candidates: number[], target: number): number[][] {

};
```

C# Solution:

```
/*
* Problem: Combination Sum
* 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>> CombinationSum(int[] candidates, int target) {

}
}
```

C Solution:

```
/*
* Problem: Combination Sum
* 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** combinationSum(int* candidates, int candidatesSize, int target, int*
returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Combination Sum
// 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 combinationSum(candidates []int, target int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun combinationSum(candidates: IntArray, target: Int): List<List<Int>> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Combination Sum
// 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_sum(candidates: Vec<i32>, target: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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