Problem 1449: Form Largest Integer With Digits That Add up to Target

Problem Information

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

Problem Description

Given an array of integers

cost

and an integer

target

, return

the

maximum

integer you can paint under the following rules

:

The cost of painting a digit

(i + 1)

is given by

cost[i]

(

0-indexed

).

The total cost used must be equal to

target

.

The integer does not have

0

digits.

Since the answer may be very large, return it as a string. If there is no way to paint any integer given the condition, return

"0"

.

Example 1:

Input:

cost = [4,3,2,5,6,7,2,5,5], target = 9

Output:

"7772"

Explanation:

The cost to paint the digit '7' is 2, and the digit '2' is 3. Then cost("7772") = 2*3+ 3*1 = 9. You could also paint "977", but "7772" is the largest number.

Digit cost

1 -> 4 2 -> 3 3 -> 2 4 -> 5 5 -> 6 6 -> 7 7 -> 2 8 -> 5 9 -> 5

Example 2:

Input:

cost = [7,6,5,5,5,6,8,7,8], target = 12

Output:

"85"

Explanation:

The cost to paint the digit '8' is 7, and the digit '5' is 5. Then cost("85") = 7 + 5 = 12.

Example 3:

Input:

cost = [2,4,6,2,4,6,4,4,4], target = 5

Output:

"0"

Explanation:

It is impossible to paint any integer with total cost equal to target.

Constraints:

cost.length == 9

1 <= cost[i], target <= 5000

Code Snippets

C++:

```
class Solution {
public:
string largestNumber(vector<int>& cost, int target) {

}
};
```

Java:

```
class Solution {
public String largestNumber(int[] cost, int target) {

}
}
```

Python3:

```
class Solution:
def largestNumber(self, cost: List[int], target: int) -> str:
```

Python:

```
class Solution(object):
def largestNumber(self, cost, target):
"""
:type cost: List[int]
:type target: int
:rtype: str
"""
```

JavaScript:

```
/**
* @param {number[]} cost
* @param {number} target
* @return {string}
*/
var largestNumber = function(cost, target) {
};
```

TypeScript:

```
function largestNumber(cost: number[], target: number): string {

};
```

C#:

```
public class Solution {
public string LargestNumber(int[] cost, int target) {

}
}
```

C:

```
char* largestNumber(int* cost, int costSize, int target) {

}
```

Go:

```
func largestNumber(cost []int, target int) string {

}
```

Kotlin:

```
class Solution {
fun largestNumber(cost: IntArray, target: Int): String {

}
}
```

Swift:

```
class Solution {
func largestNumber(_ cost: [Int], _ target: Int) -> String {
}
}
```

Rust:

```
impl Solution {
pub fn largest_number(cost: Vec<i32>, target: i32) -> String {

}
}
```

Ruby:

```
# @param {Integer[]} cost
# @param {Integer} target
# @return {String}
def largest_number(cost, target)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $cost
* @param Integer $target
* @return String
*/
function largestNumber($cost, $target) {

}
}
```

Dart:

```
class Solution {
String largestNumber(List<int> cost, int target) {

}
}
```

Scala:

```
object Solution {
def largestNumber(cost: Array[Int], target: Int): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec largest_number(cost :: [integer], target :: integer) :: String.t
def largest_number(cost, target) do

end
end
```

Erlang:

```
-spec largest_number(Cost :: [integer()], Target :: integer()) ->
unicode:unicode_binary().
largest_number(Cost, Target) ->
.
```

Racket:

```
(define/contract (largest-number cost target)
(-> (listof exact-integer?) exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Form Largest Integer With Digits That Add up to Target
* Difficulty: Hard
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

class Solution {
public:
string largestNumber(vector<int>& cost, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Form Largest Integer With Digits That Add up to Target
* Difficulty: Hard
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public String largestNumber(int[] cost, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Form Largest Integer With Digits That Add up to Target
Difficulty: Hard
Tags: array, string, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def largestNumber(self, cost: List[int], target: int) -> str:
```

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

Python Solution:

```
class Solution(object):
def largestNumber(self, cost, target):
"""
:type cost: List[int]
:type target: int
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Form Largest Integer With Digits That Add up to Target
* Difficulty: Hard
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} cost
* @param {number} target
* @return {string}
*/
var largestNumber = function(cost, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Form Largest Integer With Digits That Add up to Target
* Difficulty: Hard
* Tags: array, string, dp
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function largestNumber(cost: number[], target: number): string {

};
```

C# Solution:

```
/*
* Problem: Form Largest Integer With Digits That Add up to Target
* Difficulty: Hard
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public string LargestNumber(int[] cost, int target) {

}
}
```

C Solution:

```
/*
* Problem: Form Largest Integer With Digits That Add up to Target
* Difficulty: Hard
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

char* largestNumber(int* cost, int costSize, int target) {

}
```

Go Solution:

```
// Problem: Form Largest Integer With Digits That Add up to Target
// Difficulty: Hard
// Tags: array, string, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func largestNumber(cost []int, target int) string {

}
```

Kotlin Solution:

```
class Solution {
fun largestNumber(cost: IntArray, target: Int): String {

}
}
```

Swift Solution:

```
class Solution {
func largestNumber(_ cost: [Int], _ target: Int) -> String {

}
}
```

Rust Solution:

```
// Problem: Form Largest Integer With Digits That Add up to Target
// Difficulty: Hard
// Tags: array, string, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table
```

```
impl Solution {
pub fn largest_number(cost: Vec<i32>, target: i32) -> String {

}
}
```

Ruby Solution:

```
# @param {Integer[]} cost
# @param {Integer} target
# @return {String}
def largest_number(cost, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $cost
* @param Integer $target
* @return String
*/
function largestNumber($cost, $target) {

}
}
```

Dart Solution:

```
class Solution {
String largestNumber(List<int> cost, int target) {

}
}
```

Scala Solution:

```
object Solution {
def largestNumber(cost: Array[Int], target: Int): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec largest_number(cost :: [integer], target :: integer) :: String.t
def largest_number(cost, target) do

end
end
```

Erlang Solution:

```
-spec largest_number(Cost :: [integer()], Target :: integer()) ->
unicode:unicode_binary().
largest_number(Cost, Target) ->
.
```

Racket Solution:

```
(define/contract (largest-number cost target)
(-> (listof exact-integer?) exact-integer? string?)
)
```