Problem 3685: Subsequence Sum After Capping Elements

Problem Information

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

Problem Description

You are given an integer array

nums

of size

n

and a positive integer

k

.

An array

capped

by value

x

is obtained by replacing every element

nums[i]

with

min(nums[i], x)

.

For each integer

x

from 1 to

n

, determine whether it is possible to choose a

subsequence

from the array capped by

x

such that the sum of the chosen elements is

exactly

k

.

Return a

0-indexed

boolean array

answer

of size

n

, where

answer[i]

is

true

if it is possible when using

x = i + 1

, and

false

otherwise.

Example 1:

Input:

nums = [4,3,2,4], k = 5

Output:

[false,false,true,true]

Explanation:

For

x = 1

, the capped array is

[1, 1, 1, 1]

. Possible sums are

1, 2, 3, 4

, so it is impossible to form a sum of

5

.

For

x = 2

, the capped array is

[2, 2, 2, 2]

. Possible sums are

2, 4, 6, 8

, so it is impossible to form a sum of

5

.

For

x = 3

, the capped array is

[3, 3, 2, 3]

. A subsequence

[2, 3]

sums to

5

, so it is possible.

For

x = 4

, the capped array is

[4, 3, 2, 4]

. A subsequence

[3, 2]

sums to

5

, so it is possible.

Example 2:

Input:

nums = [1,2,3,4,5], k = 3

Output:

[true,true,true,true,true]

Explanation:

For every value of

x

, it is always possible to select a subsequence from the capped array that sums exactly to

3

.

Constraints:

1 <= n == nums.length <= 4000

1 <= nums[i] <= n

1 <= k <= 4000

Code Snippets

C++:

```
class Solution {
public:
vector<bool> subsequenceSumAfterCapping(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public boolean[] subsequenceSumAfterCapping(int[] nums, int k) {

}
}
```

Python3:

```
class Solution:
def subsequenceSumAfterCapping(self, nums: List[int], k: int) -> List[bool]:
```

Python:

```
class Solution(object):
def subsequenceSumAfterCapping(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: List[bool]
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} k
* @return {boolean[]}
*/
var subsequenceSumAfterCapping = function(nums, k) {

};
```

TypeScript:

```
function subsequenceSumAfterCapping(nums: number[], k: number): boolean[] {

};
```

C#:

```
public class Solution {
public bool[] SubsequenceSumAfterCapping(int[] nums, int k) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
```

```
bool* subsequenceSumAfterCapping(int* nums, int numsSize, int k, int*
returnSize) {

}
```

Go:

```
func subsequenceSumAfterCapping(nums []int, k int) []bool {

}
```

Kotlin:

```
class Solution {
fun subsequenceSumAfterCapping(nums: IntArray, k: Int): BooleanArray {

}
}
```

Swift:

```
class Solution {
func subsequenceSumAfterCapping(_ nums: [Int], _ k: Int) -> [Bool] {

}
}
```

Rust:

```
impl Solution {
pub fn subsequence_sum_after_capping(nums: Vec<i32>, k: i32) -> Vec<bool> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Boolean[]}
def subsequence_sum_after_capping(nums, k)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Boolean[]
*/
function subsequenceSumAfterCapping($nums, $k) {

}
}
```

Dart:

```
class Solution {
List<bool> subsequenceSumAfterCapping(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def subsequenceSumAfterCapping(nums: Array[Int], k: Int): Array[Boolean] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec subsequence_sum_after_capping(nums :: [integer], k :: integer) ::
[boolean]
def subsequence_sum_after_capping(nums, k) do

end
end
```

Erlang:

```
-spec subsequence_sum_after_capping(Nums :: [integer()], K :: integer()) ->
[boolean()].
subsequence_sum_after_capping(Nums, K) ->
.
```

Racket:

```
(define/contract (subsequence-sum-after-capping nums k)
(-> (listof exact-integer?) exact-integer? (listof boolean?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Subsequence Sum After Capping Elements
* Difficulty: Medium
* Tags: array, dp, sort
*
* 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:
vector<bool> subsequenceSumAfterCapping(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Subsequence Sum After Capping Elements
* Difficulty: Medium
* Tags: array, dp, sort
*
```

```
* 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 boolean[] subsequenceSumAfterCapping(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Subsequence Sum After Capping Elements
Difficulty: Medium
Tags: array, dp, sort

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 subsequenceSumAfterCapping(self, nums: List[int], k: int) -> List[bool]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def subsequenceSumAfterCapping(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: List[bool]
"""
```

JavaScript Solution:

```
/**
* Problem: Subsequence Sum After Capping Elements
* Difficulty: Medium
* Tags: array, dp, sort
*
* 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[]} nums
* @param {number} k
* @return {boolean[]}
*/
var subsequenceSumAfterCapping = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Subsequence Sum After Capping Elements
* Difficulty: Medium
* Tags: array, dp, sort
*
* 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 subsequenceSumAfterCapping(nums: number[], k: number): boolean[] {

};
```

C# Solution:

```
/*
* Problem: Subsequence Sum After Capping Elements
* Difficulty: Medium
* Tags: array, dp, sort
*
```

```
* 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 bool[] SubsequenceSumAfterCapping(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Subsequence Sum After Capping Elements
* Difficulty: Medium
* Tags: array, dp, sort
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
bool* subsequenceSumAfterCapping(int* nums, int numsSize, int k, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Subsequence Sum After Capping Elements
// Difficulty: Medium
// Tags: array, dp, sort
//
// 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 subsequenceSumAfterCapping(nums []int, k int) []bool {

}
```

Kotlin Solution:

```
class Solution {
fun subsequenceSumAfterCapping(nums: IntArray, k: Int): BooleanArray {

}
}
```

Swift Solution:

```
class Solution {
func subsequenceSumAfterCapping(_ nums: [Int], _ k: Int) -> [Bool] {

}
}
```

Rust Solution:

```
// Problem: Subsequence Sum After Capping Elements
// Difficulty: Medium
// Tags: array, dp, sort
//
// 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 subsequence_sum_after_capping(nums: Vec<i32>, k: i32) -> Vec<bool> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Boolean[]}
```

```
def subsequence_sum_after_capping(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Boolean[]
*/
function subsequenceSumAfterCapping($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
List<bool> subsequenceSumAfterCapping(List<int> nums, int k) {

}
}
```

Scala Solution:

```
object Solution {
def subsequenceSumAfterCapping(nums: Array[Int], k: Int): Array[Boolean] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec subsequence_sum_after_capping(nums :: [integer], k :: integer) ::
[boolean]
def subsequence_sum_after_capping(nums, k) do
```

```
end
end
```

Erlang Solution:

```
-spec subsequence_sum_after_capping(Nums :: [integer()], K :: integer()) ->
[boolean()].
subsequence_sum_after_capping(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (subsequence-sum-after-capping nums k)
(-> (listof exact-integer?) exact-integer? (listof boolean?))
)
```