

Problem 2926: Maximum Balanced Subsequence Sum

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a

0-indexed

integer array

nums

.

A

subsequence

of

nums

having length

k

and consisting of

indices

i

0

< i

1

< ... < i

k-1

is

balanced

if the following holds:

nums[i

j

] - nums[i

j-1

] >= i

j

- i

j-1

, for every

j

in the range

[1, k - 1]

.

A

subsequence

of

nums

having length

1

is considered balanced.

Return

an integer denoting the

maximum

possible

sum of elements

in a

balanced

subsequence of

nums

.

A

subsequence

of an array is a new

non-empty

array that is formed from the original array by deleting some (

possibly none

) of the elements without disturbing the relative positions of the remaining elements.

Example 1:

Input:

nums = [3,3,5,6]

Output:

14

Explanation:

In this example, the subsequence [3,5,6] consisting of indices 0, 2, and 3 can be selected. $\text{nums}[2] - \text{nums}[0] \geq 2 - 0$. $\text{nums}[3] - \text{nums}[2] \geq 3 - 2$. Hence, it is a balanced subsequence, and its sum is the maximum among the balanced subsequences of nums. The subsequence consisting of indices 1, 2, and 3 is also valid. It can be shown that it is not possible to get a balanced subsequence with a sum greater than 14.

Example 2:

Input:

nums = [5,-1,-3,8]

Output:

13

Explanation:

In this example, the subsequence [5,8] consisting of indices 0 and 3 can be selected. $\text{nums}[3] - \text{nums}[0] \geq 3 - 0$. Hence, it is a balanced subsequence, and its sum is the maximum among the balanced subsequences of nums. It can be shown that it is not possible to get a balanced subsequence with a sum greater than 13.

Example 3:

Input:

nums = [-2,-1]

Output:

-1

Explanation:

In this example, the subsequence [-1] can be selected. It is a balanced subsequence, and its sum is the maximum among the balanced subsequences of nums.

Constraints:

$1 \leq \text{nums.length} \leq 10$

5

-10

9

$\leq \text{nums}[i] \leq 10$

9

Code Snippets

C++:

```
class Solution {  
public:  
    long long maxBalancedSubsequenceSum(vector<int>& nums) {  
  
    }  
};
```

Java:

```
class Solution {  
    public long maxBalancedSubsequenceSum(int[] nums) {  
  
    }  
}
```

Python3:

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

Python:

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

JavaScript:

```
/**  
 * @param {number[]} nums  
 * @return {number}  
 */  
var maxBalancedSubsequenceSum = function(nums) {  
  
};
```

TypeScript:

```
function maxBalancedSubsequenceSum(nums: number[]): number {  
}  
};
```

C#:

```
public class Solution {  
    public long MaxBalancedSubsequenceSum(int[] nums) {  
  
    }  
}
```

C:

```
long long maxBalancedSubsequenceSum(int* nums, int numssSize) {  
  
}
```

Go:

```
func maxBalancedSubsequenceSum(nums []int) int64 {  
  
}
```

Kotlin:

```
class Solution {  
    fun maxBalancedSubsequenceSum(nums: IntArray): Long {  
  
    }  
}
```

Swift:

```
class Solution {  
    func maxBalancedSubsequenceSum(_ nums: [Int]) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {
    pub fn max_balanced_subsequence_sum(nums: Vec<i32>) -> i64 {
        }
    }
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def max_balanced_subsequence_sum(nums)

end
```

PHP:

```
class Solution {

    /**
     * @param Integer[] $nums
     * @return Integer
     */
    function maxBalancedSubsequenceSum($nums) {

    }
}
```

Dart:

```
class Solution {
    int maxBalancedSubsequenceSum(List<int> nums) {
        }
    }
```

Scala:

```
object Solution {
    def maxBalancedSubsequenceSum(nums: Array[Int]): Long = {
        }
    }
```

Elixir:

```
defmodule Solution do
  @spec max_balanced_subsequence_sum(nums :: [integer]) :: integer
  def max_balanced_subsequence_sum(nums) do
    end
  end
end
```

Erlang:

```
-spec max_balanced_subsequence_sum(Nums :: [integer()]) -> integer().
max_balanced_subsequence_sum(Nums) ->
  .
```

Racket:

```
(define/contract (max-balanced-subsequence-sum nums)
  (-> (listof exact-integer?) exact-integer?))
```

Solutions

C++ Solution:

```
/*
 * Problem: Maximum Balanced Subsequence Sum
 * Difficulty: Hard
 * Tags: array, tree, dp, search
 *
 * 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:
  long long maxBalancedSubsequenceSum(vector<int>& nums) {
    }
};
```

Java Solution:

```
/**  
 * Problem: Maximum Balanced Subsequence Sum  
 * Difficulty: Hard  
 * Tags: array, tree, dp, search  
 *  
 * 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 long maxBalancedSubsequenceSum(int[] nums) {  
  
    }  
}
```

Python3 Solution:

```
"""  
Problem: Maximum Balanced Subsequence Sum  
Difficulty: Hard  
Tags: array, tree, dp, search  
  
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 maxBalancedSubsequenceSum(self, nums: List[int]) -> int:  
        # TODO: Implement optimized solution  
        pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**  
 * Problem: Maximum Balanced Subsequence Sum  
 * Difficulty: Hard  
 * Tags: array, tree, dp, search  
 *  
 * 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  
 * @return {number}  
 */  
var maxBalancedSubsequenceSum = function(nums) {  
  
};
```

TypeScript Solution:

```
/**  
 * Problem: Maximum Balanced Subsequence Sum  
 * Difficulty: Hard  
 * Tags: array, tree, dp, search  
 *  
 * 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 maxBalancedSubsequenceSum(nums: number[]): number {  
  
};
```

C# Solution:

```

/*
 * Problem: Maximum Balanced Subsequence Sum
 * Difficulty: Hard
 * Tags: array, tree, dp, search
 *
 * 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 long MaxBalancedSubsequenceSum(int[] nums) {

    }
}

```

C Solution:

```

/*
 * Problem: Maximum Balanced Subsequence Sum
 * Difficulty: Hard
 * Tags: array, tree, dp, search
 *
 * 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
 */

long long maxBalancedSubsequenceSum(int* nums, int numssSize) {

}

```

Go Solution:

```

// Problem: Maximum Balanced Subsequence Sum
// Difficulty: Hard
// Tags: array, tree, dp, search
//
// 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 maxBalancedSubsequenceSum(nums []int) int64 {  
    }  
}
```

Kotlin Solution:

```
class Solution {  
    fun maxBalancedSubsequenceSum(nums: IntArray): Long {  
        }  
    }  
}
```

Swift Solution:

```
class Solution {  
    func maxBalancedSubsequenceSum(_ nums: [Int]) -> Int {  
        }  
    }  
}
```

Rust Solution:

```
// Problem: Maximum Balanced Subsequence Sum  
// Difficulty: Hard  
// Tags: array, tree, dp, search  
//  
// 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 max_balanced_subsequence_sum(nums: Vec<i32>) -> i64 {  
        }  
    }  
}
```

Ruby Solution:

```
# @param {Integer[]} nums  
# @return {Integer}  
def max_balanced_subsequence_sum(nums)
```

```
end
```

PHP Solution:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums  
     * @return Integer  
     */  
    function maxBalancedSubsequenceSum($nums) {  
  
    }  
}
```

Dart Solution:

```
class Solution {  
int maxBalancedSubsequenceSum(List<int> nums) {  
  
}  
}
```

Scala Solution:

```
object Solution {  
def maxBalancedSubsequenceSum(nums: Array[Int]): Long = {  
  
}  
}
```

Elixir Solution:

```
defmodule Solution do  
@spec max_balanced_subsequence_sum(nums :: [integer]) :: integer  
def max_balanced_subsequence_sum(nums) do  
  
end  
end
```

Erlang Solution:

```
-spec max_balanced_subsequence_sum(Nums :: [integer()]) -> integer().  
max_balanced_subsequence_sum(Nums) ->  
. 
```

Racket Solution:

```
(define/contract (max-balanced-subsequence-sum nums)  
(-> (listof exact-integer?) exact-integer?)  
) 
```