Problem 3462: Maximum Sum With at Most K Elements

Problem Information

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

Problem Description

You are given a 2D integer matrix

grid

of size

n x m

, an integer array

limits

of length

n

, and an integer

k

. The task is to find the

maximum sum

of

at most

k

elements from the matrix

grid

such that:

The number of elements taken from the

i

th

row of

grid

does not exceed

limits[i]

.

Return the

maximum sum

.

Example 1:

Input:

grid = [[1,2],[3,4]], limits = [1,2], k = 2

Output:

7

Explanation:

From the second row, we can take at most 2 elements. The elements taken are 4 and 3.

The maximum possible sum of at most 2 selected elements is

4 + 3 = 7

.

Example 2:

Input:

grid = [[5,3,7],[8,2,6]], limits = [2,2], k = 3

Output:

21

Explanation:

From the first row, we can take at most 2 elements. The element taken is 7.

From the second row, we can take at most 2 elements. The elements taken are 8 and 6.

The maximum possible sum of at most 3 selected elements is

7 + 8 + 6 = 21

.

Constraints:

n == grid.length == limits.length

m == grid[i].length

1 <= n, m <= 500

0 <= grid[i][j] <= 10

5

0 <= limits[i] <= m

0 <= k <= min(n * m, sum(limits))

Code Snippets

C++:

```
class Solution {
public:
long long maxSum(vector<vector<int>>& grid, vector<int>& limits, int k) {

}
};
```

Java:

```
class Solution {
public long maxSum(int[][] grid, int[] limits, int k) {

}
}
```

Python3:

```
class Solution:
def maxSum(self, grid: List[List[int]], limits: List[int], k: int) -> int:
```

Python:

```
class Solution(object):
def maxSum(self, grid, limits, k):
```

```
"""
:type grid: List[List[int]]
:type limits: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} grid
* @param {number[]} limits
* @param {number} k
* @return {number}
*/
var maxSum = function(grid, limits, k) {

};
```

TypeScript:

```
function maxSum(grid: number[][], limits: number[], k: number): number {

};
```

C#:

```
public class Solution {
public long MaxSum(int[][] grid, int[] limits, int k) {

}
}
```

C:

```
long long maxSum(int** grid, int gridSize, int* gridColSize, int* limits, int
limitsSize, int k) {

}
```

Go:

```
func maxSum(grid [][]int, limits []int, k int) int64 {

}
```

Kotlin:

```
class Solution {
fun maxSum(grid: Array<IntArray>, limits: IntArray, k: Int): Long {

}
}
```

Swift:

```
class Solution {
func maxSum(_ grid: [[Int]], _ limits: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_sum(grid: Vec<Vec<i32>>, limits: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @param {Integer[]} limits
# @param {Integer} k
# @return {Integer}
def max_sum(grid, limits, k)

end
```

PHP:

```
class Solution {

/**
```

```
* @param Integer[][] $grid
* @param Integer[] $limits
* @param Integer $k
* @return Integer
*/
function maxSum($grid, $limits, $k) {

}
}
```

Dart:

```
class Solution {
int maxSum(List<List<int>> grid, List<int> limits, int k) {

}
}
```

Scala:

```
object Solution {
def maxSum(grid: Array[Array[Int]], limits: Array[Int], k: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_sum(grid :: [[integer]], limits :: [integer], k :: integer) ::
integer
def max_sum(grid, limits, k) do

end
end
```

Erlang:

```
-spec max_sum(Grid :: [[integer()]], Limits :: [integer()], K :: integer())
-> integer().
max_sum(Grid, Limits, K) ->
.
```

Racket:

```
(define/contract (max-sum grid limits k)
(-> (listof (listof exact-integer?)) (listof exact-integer?) exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Sum With at Most K Elements
* Difficulty: Medium
* Tags: array, greedy, sort, queue, heap
*
* 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:
long long maxSum(vector<vector<int>>& grid, vector<int>& limits, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Sum With at Most K Elements
* Difficulty: Medium
* Tags: array, greedy, sort, queue, heap
*
* 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 long maxSum(int[][] grid, int[] limits, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Sum With at Most K Elements
Difficulty: Medium
Tags: array, greedy, sort, queue, heap

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 maxSum(self, grid: List[List[int]], limits: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxSum(self, grid, limits, k):
"""
:type grid: List[List[int]]
:type limits: List[int]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Sum With at Most K Elements
* Difficulty: Medium
* Tags: array, greedy, sort, queue, heap
*
* 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[][]} grid
* @param {number[]} limits
* @param {number} k
* @return {number}
*/
var maxSum = function(grid, limits, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Sum With at Most K Elements
* Difficulty: Medium
* Tags: array, greedy, sort, queue, heap
*
* 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 maxSum(grid: number[][], limits: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Sum With at Most K Elements
* Difficulty: Medium
* Tags: array, greedy, sort, queue, heap
*
* 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 long MaxSum(int[][] grid, int[] limits, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Sum With at Most K Elements
* Difficulty: Medium
* Tags: array, greedy, sort, queue, heap
*
* 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
*/

long long maxSum(int** grid, int gridSize, int* gridColSize, int* limits, int
limitsSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum Sum With at Most K Elements
// Difficulty: Medium
// Tags: array, greedy, sort, queue, heap
//
// 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 maxSum(grid [][]int, limits []int, k int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maxSum(grid: Array<IntArray>, limits: IntArray, k: Int): Long {

}
}
```

Swift Solution:

```
class Solution {
func maxSum(_ grid: [[Int]], _ limits: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Sum With at Most K Elements
// Difficulty: Medium
// Tags: array, greedy, sort, queue, heap
//
// 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 max_sum(grid: Vec<Vec<i32>>, limits: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @param {Integer[]} limits
# @param {Integer} k
# @return {Integer}
def max_sum(grid, limits, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @param Integer[] $limits
* @param Integer $k
* @return Integer
*/
function maxSum($grid, $limits, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maxSum(List<List<int>> grid, List<int> limits, int k) {

}
}
```

Scala Solution:

```
object Solution {
def maxSum(grid: Array[Array[Int]], limits: Array[Int], k: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_sum(grid :: [[integer]], limits :: [integer], k :: integer) ::
integer
def max_sum(grid, limits, k) do

end
end
```

Erlang Solution:

```
-spec max_sum(Grid :: [[integer()]], Limits :: [integer()], K :: integer())
-> integer().
max_sum(Grid, Limits, K) ->
.
```

Racket Solution:

```
(define/contract (max-sum grid limits k)
(-> (listof (listof exact-integer?)) (listof exact-integer?) exact-integer?
exact-integer?)
)
```