Problem 2218: Maximum Value of K Coins From Piles

Problem Information

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

Problem Description

There are

n

piles

of coins on a table. Each pile consists of a

positive number

of coins of assorted denominations.

In one move, you can choose any coin on

top

of any pile, remove it, and add it to your wallet.

Given a list

piles

, where

piles[i]

is a list of integers denoting the composition of the

i

th

pile from

top to bottom

, and a positive integer

k

, return

the

maximum total value

of coins you can have in your wallet if you choose

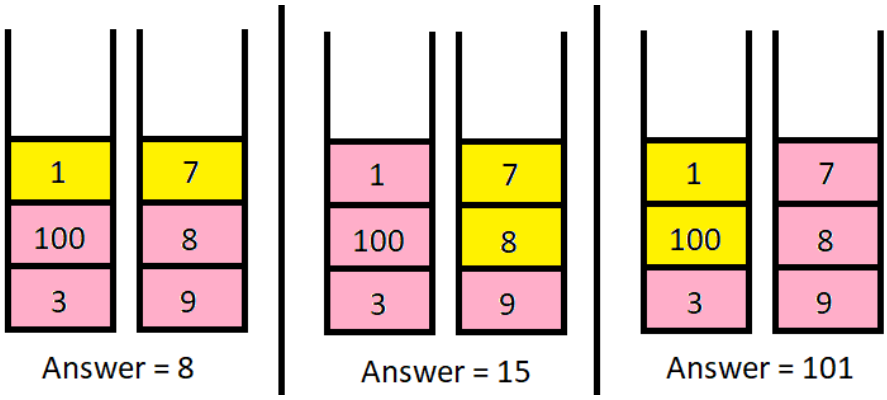
exactly

k

coins optimally

.

Example 1:



Input:

piles = [[1,100,3],[7,8,9]], k = 2

Output:

101

Explanation:

The above diagram shows the different ways we can choose k coins. The maximum total we can obtain is 101.

Example 2:

Input:

piles = [[100],[100],[100],[100],[100],[100],[1,1,1,1,1,1,700]], k = 7

Output:

706

Explanation:

The maximum total can be obtained if we choose all coins from the last pile.

Constraints:

n == piles.length

1 <= n <= 1000

1 <= piles[i][j] <= 10

5

1 <= k <= sum(piles[i].length) <= 2000

Code Snippets

C++:

```
class Solution {
public:
int maxValueOfCoins(vector<vector<int>>& piles, int k) {

}
};
```

Java:

```
class Solution {
public int maxValueOfCoins(List<List<Integer>> piles, int k) {

}
}
```

Python3:

```
class Solution:
def maxValueOfCoins(self, piles: List[List[int]], k: int) -> int:
```

Python:

```
class Solution(object):
def maxValueOfCoins(self, piles, k):
"""
:type piles: List[List[int]]
:type k: int
:rtype: int
```

```
"""
```

JavaScript:

```
/**
* @param {number[][]} piles
* @param {number} k
* @return {number}
*/
var maxValueOfCoins = function(piles, k) {

};
```

TypeScript:

```
function maxValueOfCoins(piles: number[][], k: number): number {

};
```

C#:

```
public class Solution {
public int MaxValueOfCoins(IList<IList<int>> piles, int k) {

}
}
```

C:

```
int maxValueOfCoins(int** piles, int pilesSize, int* pilesColSize, int k) {

}
```

Go:

```
func maxValueOfCoins(piles [][]int, k int) int {

}
```

Kotlin:

```
class Solution {
fun maxValueOfCoins(piles: List<List<Int>>, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func maxValueOfCoins(_ piles: [[Int]], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_value_of_coins(piles: Vec<Vec<i32>>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} piles
# @param {Integer} k
# @return {Integer}
def max_value_of_coins(piles, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $piles
* @param Integer $k
* @return Integer
*/
function maxValueOfCoins($piles, $k) {

}
```

```
}
```

Dart:

```
class Solution {
int maxValueOfCoins(List<List<int>> piles, int k) {

}
}
```

Scala:

```
object Solution {
def maxValueOfCoins(piles: List[List[Int]], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_value_of_coins(piles :: [[integer]], k :: integer) :: integer
def max_value_of_coins(piles, k) do

end
end
```

Erlang:

```
-spec max_value_of_coins(Piles :: [[integer()]], K :: integer()) ->
integer().
max_value_of_coins(Piles, K) ->
.
```

Racket:

```
(define/contract (max-value-of-coins piles k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Value of K Coins From Piles
* Difficulty: Hard
* 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:
int maxValueOfCoins(vector<vector<int>>& piles, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Value of K Coins From Piles
* Difficulty: Hard
* 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 int maxValueOfCoins(List<List<Integer>> piles, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Value of K Coins From Piles
```

```
Difficulty: Hard
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 maxValueOfCoins(self, piles: List[List[int]], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxValueOfCoins(self, piles, k):
"""
:type piles: List[List[int]]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Value of K Coins From Piles
* Difficulty: Hard
* 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[][]} piles
* @param {number} k
* @return {number}
*/
var maxValueOfCoins = function(piles, k) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Maximum Value of K Coins From Piles
* Difficulty: Hard
* 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 maxValueOfCoins(piles: number[][], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Value of K Coins From Piles
* Difficulty: Hard
* 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 int MaxValueOfCoins(IList<IList<int>> piles, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Value of K Coins From Piles
```

```
* Difficulty: Hard
* 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
*/

int maxValueOfCoins(int** piles, int pilesSize, int* pilesColSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum Value of K Coins From Piles
// Difficulty: Hard
// 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 maxValueOfCoins(piles [][]int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxValueOfCoins(piles: List<List<Int>>, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxValueOfCoins(_ piles: [[Int]], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Value of K Coins From Piles
// Difficulty: Hard
// 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 max_value_of_coins(piles: Vec<Vec<i32>>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} piles
# @param {Integer} k
# @return {Integer}
def max_value_of_coins(piles, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $piles
* @param Integer $k
* @return Integer
*/
function maxValueOfCoins($piles, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maxValueOfCoins(List<List<int>> piles, int k) {

}
}
```

Scala Solution:

```
object Solution {
def maxValueOfCoins(piles: List[List[Int]], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_value_of_coins(piles :: [[integer]], k :: integer) :: integer
def max_value_of_coins(piles, k) do

end
end
```

Erlang Solution:

```
-spec max_value_of_coins(Piles :: [[integer()]], K :: integer()) ->
integer().
max_value_of_coins(Piles, K) ->
.
```

Racket Solution:

```
(define/contract (max-value-of-coins piles k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```