Problem 2861: Maximum Number of Alloys

Problem Information

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

Problem Description

You are the owner of a company that creates alloys using various types of metals. There are

n

different types of metals available, and you have access to

k

machines that can be used to create alloys. Each machine requires a specific amount of each metal type to create an alloy.

For the

i

th

machine to create an alloy, it needs

composition[i][j]

units of metal of type

j

. Initially, you have

stock[i]

units of metal type

i

, and purchasing one unit of metal type

i

costs

cost[i]

coins.

Given integers

n

,

k

,

budget

, a

1-indexed

2D array

composition

, and

1-indexed

arrays

stock

and

cost

, your goal is to

maximize

the number of alloys the company can create while staying within the budget of

budget

coins.

All alloys must be created with the same machine.

Return

the maximum number of alloys that the company can create

.

Example 1:

Input:

n = 3, k = 2, budget = 15, composition = [[1,1,1],[1,1,10]], stock = [0,0,0], cost = [1,2,3]

Output:

2

Explanation:

It is optimal to use the 1

st

machine to create alloys. To create 2 alloys we need to buy the: - 2 units of metal of the 1

st

type. - 2 units of metal of the 2

nd

type. - 2 units of metal of the 3

rd

type. In total, we need 2 * 1 + 2 * 2 + 2 * 3 = 12 coins, which is smaller than or equal to budget = 15. Notice that we have 0 units of metal of each type and we have to buy all the required units of metal. It can be proven that we can create at most 2 alloys.

Example 2:

Input:

n = 3, k = 2, budget = 15, composition = [[1,1,1],[1,1,10]], stock = [0,0,100], cost = [1,2,3]

Output:

5

Explanation:

It is optimal to use the 2

nd

machine to create alloys. To create 5 alloys we need to buy: - 5 units of metal of the 1

st

type. - 5 units of metal of the 2

nd

type. - 0 units of metal of the 3

rd

type. In total, we need 5 * 1 + 5 * 2 + 0 * 3 = 15 coins, which is smaller than or equal to budget = 15. It can be proven that we can create at most 5 alloys.

Example 3:

Input:

n = 2, k = 3, budget = 10, composition = [[2,1],[1,2],[1,1]], stock = [1,1], cost = [5,5]

Output:

2

Explanation:

It is optimal to use the 3

rd

machine to create alloys. To create 2 alloys we need to buy the: - 1 unit of metal of the 1

st

type. - 1 unit of metal of the 2

nd

type. In total, we need 1 * 5 + 1 * 5 = 10 coins, which is smaller than or equal to budget = 10. It can be proven that we can create at most 2 alloys.

Constraints:

1 <= n, k <= 100

0 <= budget <= 10

8

composition.length == k

composition[i].length == n

1 <= composition[i][j] <= 100

stock.length == cost.length == n

0 <= stock[i] <= 10

8

1 <= cost[i] <= 100

Code Snippets

C++:

```
class Solution {
public:
int maxNumberOfAlloys(int n, int k, int budget, vector<vector<int>>&
composition, vector<int>& stock, vector<int>& cost) {

}
};
```

Java:

```
class Solution {
public int maxNumberOfAlloys(int n, int k, int budget, List<List<Integer>>
composition, List<Integer> stock, List<Integer> cost) {

}
```

```
}
```

Python3:

```
class Solution:
def maxNumberOfAlloys(self, n: int, k: int, budget: int, composition:
List[List[int]], stock: List[int], cost: List[int]) -> int:
```

Python:

```
class Solution(object):
def maxNumberOfAlloys(self, n, k, budget, composition, stock, cost):
"""
:type n: int
:type k: int
:type budget: int
:type composition: List[List[int]]
:type stock: List[int]
:type cost: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} k
* @param {number} budget
* @param {number[][]} composition
* @param {number[]} stock
* @param {number[]} cost
* @return {number}
*/
var maxNumberOfAlloys = function(n, k, budget, composition, stock, cost) {

};
```

TypeScript:

```
function maxNumberOfAlloys(n: number, k: number, budget: number, composition:
number[][], stock: number[], cost: number[]): number {
```

```
};
```

C#:

```
public class Solution {
public int MaxNumberOfAlloys(int n, int k, int budget, IList<IList<int>>
composition, IList<int> stock, IList<int> cost) {

}
}
```

C:

```
int maxNumberOfAlloys(int n, int k, int budget, int** composition, int
compositionSize, int* compositionColSize, int* stock, int stockSize, int*
cost, int costSize) {

}
```

Go:

```
func maxNumberOfAlloys(n int, k int, budget int, composition [][]int, stock
[]int, cost []int) int {

}
```

Kotlin:

```
class Solution {
fun maxNumberOfAlloys(n: Int, k: Int, budget: Int, composition:
List<List<Int>>, stock: List<Int>, cost: List<Int>): Int {

}
}
```

Swift:

```
class Solution {
func maxNumberOfAlloys(_ n: Int, _ k: Int, _ budget: Int, _ composition:
[[Int]], _ stock: [Int], _ cost: [Int]) -> Int {

}
```

```
}
```

Rust:

```
impl Solution {
pub fn max_number_of_alloys(n: i32, k: i32, budget: i32, composition:
Vec<Vec<i32>>, stock: Vec<i32>, cost: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} k
# @param {Integer} budget
# @param {Integer[][]} composition
# @param {Integer[]} stock
# @param {Integer[]} cost
# @return {Integer}
def max_number_of_alloys(n, k, budget, composition, stock, cost)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @param Integer $budget
* @param Integer[][] $composition
* @param Integer[] $stock
* @param Integer[] $cost
* @return Integer
*/
function maxNumberOfAlloys($n, $k, $budget, $composition, $stock, $cost) {

}
}
```

Dart:

```
class Solution {
int maxNumberOfAlloys(int n, int k, int budget, List<List<int>> composition,
List<int> stock, List<int> cost) {

}
}
```

Scala:

```
object Solution {
def maxNumberOfAlloys(n: Int, k: Int, budget: Int, composition:
List[List[Int]], stock: List[Int], cost: List[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_number_of_alloys(n :: integer, k :: integer, budget :: integer,
composition :: [[integer]], stock :: [integer], cost :: [integer]) :: integer
def max_number_of_alloys(n, k, budget, composition, stock, cost) do

end
end
```

Erlang:

```
-spec max_number_of_alloys(N :: integer(), K :: integer(), Budget ::
integer(), Composition :: [[integer()]], Stock :: [integer()], Cost ::
[integer()]) -> integer().
max_number_of_alloys(N, K, Budget, Composition, Stock, Cost) ->
.
```

Racket:

```
(define/contract (max-number-of-alloys n k budget composition stock cost)
(-> exact-integer? exact-integer? exact-integer? (listof (listof
exact-integer?)) (listof exact-integer?) (listof exact-integer?)
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Alloys
* Difficulty: Medium
* Tags: array, search
*
* 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:
int maxNumberOfAlloys(int n, int k, int budget, vector<vector<int>>&
composition, vector<int>& stock, vector<int>& cost) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Alloys
* Difficulty: Medium
* Tags: array, search
*
* 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 int maxNumberOfAlloys(int n, int k, int budget, List<List<Integer>>
composition, List<Integer> stock, List<Integer> cost) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Alloys
Difficulty: Medium
Tags: array, search

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 maxNumberOfAlloys(self, n: int, k: int, budget: int, composition:
List[List[int]], stock: List[int], cost: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxNumberOfAlloys(self, n, k, budget, composition, stock, cost):
"""
:type n: int
:type k: int
:type budget: int
:type composition: List[List[int]]
:type stock: List[int]
:type cost: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Number of Alloys
* Difficulty: Medium
* Tags: array, search
*
* 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} n
* @param {number} k
* @param {number} budget
* @param {number[][]} composition
* @param {number[]} stock
* @param {number[]} cost
* @return {number}
*/
var maxNumberOfAlloys = function(n, k, budget, composition, stock, cost) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Alloys
* Difficulty: Medium
* Tags: array, search
*
* 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 maxNumberOfAlloys(n: number, k: number, budget: number, composition:
number[][], stock: number[], cost: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Alloys
* Difficulty: Medium
* Tags: array, search
*
* 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 int MaxNumberOfAlloys(int n, int k, int budget, IList<IList<int>>
composition, IList<int> stock, IList<int> cost) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Alloys
* Difficulty: Medium
* Tags: array, search
*
* 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
*/

int maxNumberOfAlloys(int n, int k, int budget, int** composition, int
compositionSize, int* compositionColSize, int* stock, int stockSize, int*
cost, int costSize) {

}
```

Go Solution:

```
// Problem: Maximum Number of Alloys
// Difficulty: Medium
// Tags: array, search
//
// 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 maxNumberOfAlloys(n int, k int, budget int, composition [][]int, stock
[]int, cost []int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun maxNumberOfAlloys(n: Int, k: Int, budget: Int, composition:
List<List<Int>>, stock: List<Int>, cost: List<Int>): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxNumberOfAlloys(_ n: Int, _ k: Int, _ budget: Int, _ composition:
[[Int]], _ stock: [Int], _ cost: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Alloys
// Difficulty: Medium
// Tags: array, search
//
// 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_number_of_alloys(n: i32, k: i32, budget: i32, composition:
Vec<Vec<i32>>, stock: Vec<i32>, cost: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} k
```

```
# @param {Integer} budget
# @param {Integer[][]} composition
# @param {Integer[]} stock
# @param {Integer[]} cost
# @return {Integer}
def max_number_of_alloys(n, k, budget, composition, stock, cost)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @param Integer $budget
* @param Integer[][] $composition
* @param Integer[] $stock
* @param Integer[] $cost
* @return Integer
*/
function maxNumberOfAlloys($n, $k, $budget, $composition, $stock, $cost) {

}
}
```

Dart Solution:

```
class Solution {
int maxNumberOfAlloys(int n, int k, int budget, List<List<int>> composition,
List<int> stock, List<int> cost) {

}
}
```

Scala Solution:

```
object Solution {
def maxNumberOfAlloys(n: Int, k: Int, budget: Int, composition:
List[List[Int]], stock: List[Int], cost: List[Int]): Int = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_number_of_alloys(n :: integer, k :: integer, budget :: integer,
composition :: [[integer]], stock :: [integer], cost :: [integer]) :: integer
def max_number_of_alloys(n, k, budget, composition, stock, cost) do

end
end
```

Erlang Solution:

```
-spec max_number_of_alloys(N :: integer(), K :: integer(), Budget ::
integer(), Composition :: [[integer()]], Stock :: [integer()], Cost ::
[integer()]) -> integer().
max_number_of_alloys(N, K, Budget, Composition, Stock, Cost) ->
.
```

Racket Solution:

```
(define/contract (max-number-of-alloys n k budget composition stock cost)
(-> exact-integer? exact-integer? exact-integer? (listof (listof
exact-integer?)) (listof exact-integer?) (listof exact-integer?)
exact-integer?)
)
```