Problem 2106: Maximum Fruits Harvested After at Most K Steps

Problem Information

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

Problem Description

Fruits are available at some positions on an infinite x-axis. You are given a 2D integer array

fruits

where

fruits[i] = [position

i

, amount

i

]

depicts

amount

i

fruits at the position

position

i

.

fruits

is already

sorted

by

position

i

in

ascending order

, and each

position

i

is

unique

.

You are also given an integer

startPos

and an integer

k

. Initially, you are at the position

startPos

. From any position, you can either walk to the

left or right

. It takes

one step

to move

one unit

on the x-axis, and you can walk

at most

k

steps in total. For every position you reach, you harvest all the fruits at that position, and the fruits will disappear from that position.

Return

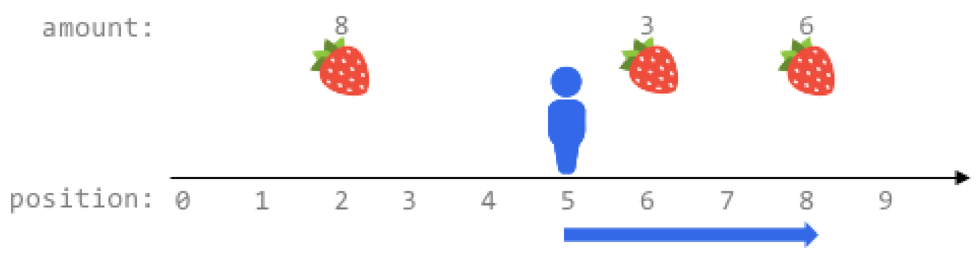
the

maximum total number

of fruits you can harvest

.

Example 1:



Input:

fruits = [[2,8],[6,3],[8,6]], startPos = 5, k = 4

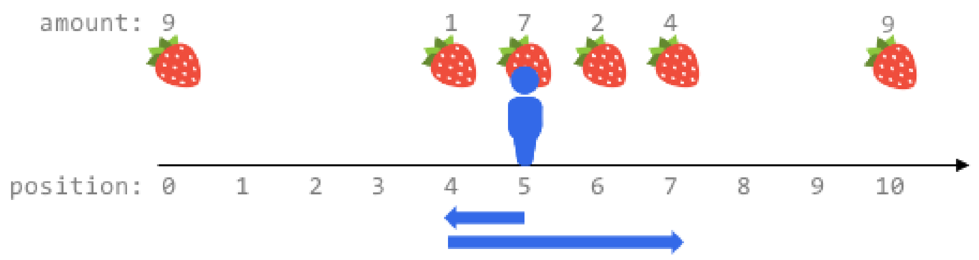
Output:

9

Explanation:

The optimal way is to: - Move right to position 6 and harvest 3 fruits - Move right to position 8 and harvest 6 fruits You moved 3 steps and harvested 3 + 6 = 9 fruits in total.

Example 2:



Input:

fruits = [[0,9],[4,1],[5,7],[6,2],[7,4],[10,9]], startPos = 5, k = 4

Output:

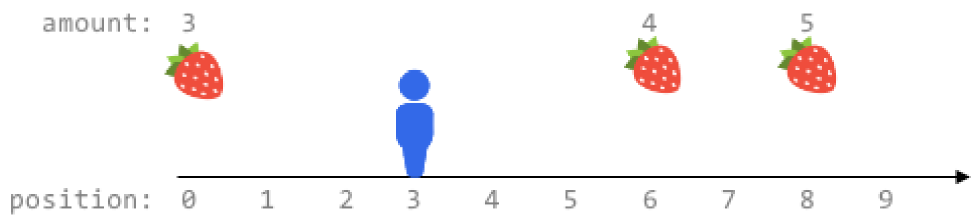
14

Explanation:

You can move at most k = 4 steps, so you cannot reach position 0 nor 10. The optimal way is to: - Harvest the 7 fruits at the starting position 5 - Move left to position 4 and harvest 1 fruit -

Move right to position 6 and harvest 2 fruits - Move right to position 7 and harvest 4 fruits You moved 1 + 3 = 4 steps and harvested 7 + 1 + 2 + 4 = 14 fruits in total.

Example 3:



Input:

fruits = [[0,3],[6,4],[8,5]], startPos = 3, k = 2

Output:

0

Explanation:

You can move at most k = 2 steps and cannot reach any position with fruits.

Constraints:

1 <= fruits.length <= 10

5

fruits[i].length == 2

0 <= startPos, position

i

<= 2 * 10

5

position

i-1

< position

i

for any

i > 0

(

0-indexed

)

1 <= amount

i

<= 10

4

0 <= k <= 2 * 10

5

Code Snippets

C++:

```
class Solution {
public:
int maxTotalFruits(vector<vector<int>>& fruits, int startPos, int k) {

}
};
```

Java:

```
class Solution {
public int maxTotalFruits(int[][] fruits, int startPos, int k) {

}
}
```

Python3:

```
class Solution:
def maxTotalFruits(self, fruits: List[List[int]], startPos: int, k: int) ->
int:
```

Python:

```
class Solution(object):
def maxTotalFruits(self, fruits, startPos, k):
"""
:type fruits: List[List[int]]
:type startPos: int
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} fruits
* @param {number} startPos
* @param {number} k
* @return {number}
*/
var maxTotalFruits = function(fruits, startPos, k) {

};
```

TypeScript:

```
function maxTotalFruits(fruits: number[][], startPos: number, k: number):
number {

};
```

C#:

```
public class Solution {
public int MaxTotalFruits(int[][] fruits, int startPos, int k) {

}
}
```

C:

```
int maxTotalFruits(int** fruits, int fruitsSize, int* fruitsColSize, int
startPos, int k) {

}
```

Go:

```
func maxTotalFruits(fruits [][]int, startPos int, k int) int {

}
```

Kotlin:

```
class Solution {
fun maxTotalFruits(fruits: Array<IntArray>, startPos: Int, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func maxTotalFruits(_ fruits: [[Int]], _ startPos: Int, _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_total_fruits(fruits: Vec<Vec<i32>>, start_pos: i32, k: i32) -> i32
{
}
}
```

Ruby:

```
# @param {Integer[][]} fruits
# @param {Integer} start_pos
# @param {Integer} k
# @return {Integer}
def max_total_fruits(fruits, start_pos, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $fruits
* @param Integer $startPos
* @param Integer $k
* @return Integer
*/
function maxTotalFruits($fruits, $startPos, $k) {

}
}
```

Dart:

```
class Solution {
int maxTotalFruits(List<List<int>> fruits, int startPos, int k) {

}
}
```

Scala:

```
object Solution {
def maxTotalFruits(fruits: Array[Array[Int]], startPos: Int, k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_total_fruits(fruits :: [[integer]], start_pos :: integer, k ::
integer) :: integer
def max_total_fruits(fruits, start_pos, k) do

end
end
```

Erlang:

```
-spec max_total_fruits(Fruits :: [[integer()]], StartPos :: integer(), K ::
integer()) -> integer().
max_total_fruits(Fruits, StartPos, K) ->
.
```

Racket:

```
(define/contract (max-total-fruits fruits startPos k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Fruits Harvested After at Most K Steps
* Difficulty: Hard
* Tags: array, sort, 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 maxTotalFruits(vector<vector<int>>& fruits, int startPos, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Fruits Harvested After at Most K Steps
* Difficulty: Hard
* Tags: array, sort, 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 maxTotalFruits(int[][] fruits, int startPos, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Fruits Harvested After at Most K Steps
Difficulty: Hard
Tags: array, sort, 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 maxTotalFruits(self, fruits: List[List[int]], startPos: int, k: int) ->
int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def maxTotalFruits(self, fruits, startPos, k):
"""
:type fruits: List[List[int]]
:type startPos: int
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Fruits Harvested After at Most K Steps
* Difficulty: Hard
* Tags: array, sort, 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[][]} fruits
* @param {number} startPos
* @param {number} k
* @return {number}
*/
var maxTotalFruits = function(fruits, startPos, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Fruits Harvested After at Most K Steps
* Difficulty: Hard
* Tags: array, sort, 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 maxTotalFruits(fruits: number[][], startPos: number, k: number):
number {

};
```

C# Solution:

```
/*
* Problem: Maximum Fruits Harvested After at Most K Steps
* Difficulty: Hard
* Tags: array, sort, 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 MaxTotalFruits(int[][] fruits, int startPos, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Fruits Harvested After at Most K Steps
* Difficulty: Hard
* Tags: array, sort, 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 maxTotalFruits(int** fruits, int fruitsSize, int* fruitsColSize, int
startPos, int k) {

}
```

Go Solution:

```
// Problem: Maximum Fruits Harvested After at Most K Steps
// Difficulty: Hard
// Tags: array, sort, 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 maxTotalFruits(fruits [][]int, startPos int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxTotalFruits(fruits: Array<IntArray>, startPos: Int, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxTotalFruits(_ fruits: [[Int]], _ startPos: Int, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Fruits Harvested After at Most K Steps
// Difficulty: Hard
// Tags: array, sort, 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_total_fruits(fruits: Vec<Vec<i32>>, start_pos: i32, k: i32) -> i32
{

}
}
```

Ruby Solution:

```
# @param {Integer[][]} fruits
# @param {Integer} start_pos
# @param {Integer} k
# @return {Integer}
def max_total_fruits(fruits, start_pos, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $fruits
* @param Integer $startPos
* @param Integer $k
* @return Integer
*/
function maxTotalFruits($fruits, $startPos, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maxTotalFruits(List<List<int>> fruits, int startPos, int k) {
}
}
```

Scala Solution:

```
object Solution {
def maxTotalFruits(fruits: Array[Array[Int]], startPos: Int, k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_total_fruits(fruits :: [[integer]], start_pos :: integer, k ::
integer) :: integer
def max_total_fruits(fruits, start_pos, k) do

end
end
```

Erlang Solution:

```
-spec max_total_fruits(Fruits :: [[integer()]], StartPos :: integer(), K ::
integer()) -> integer().
max_total_fruits(Fruits, StartPos, K) ->
.
```

Racket Solution:

```
(define/contract (max-total-fruits fruits startPos k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?
exact-integer?)
)
```