Problem 1298: Maximum Candies You Can Get from Boxes

Problem Information

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

Problem Description

You have

n

boxes labeled from

0

to

n - 1

. You are given four arrays:

status

,

candies

,

keys

, and

containedBoxes

where:

status[i]

is

1

if the

i

th

box is open and

0

if the

i

th

box is closed,

candies[i]

is the number of candies in the

i

th

box,

keys[i]

is a list of the labels of the boxes you can open after opening the

i

th

box.

containedBoxes[i]

is a list of the boxes you found inside the

i

th

box.

You are given an integer array

initialBoxes

that contains the labels of the boxes you initially have. You can take all the candies in

any open box

and you can use the keys in it to open new boxes and you also can use the boxes you find in it.

Return

the maximum number of candies you can get following the rules above

.

Example 1:

Input:

status = [1,0,1,0], candies = [7,5,4,100], keys = [[],[],[1],[]], containedBoxes = [[1,2],[3],[],[]], initialBoxes = [0]

Output:

16

Explanation:

You will be initially given box 0. You will find 7 candies in it and boxes 1 and 2. Box 1 is closed and you do not have a key for it so you will open box 2. You will find 4 candies and a key to box 1 in box 2. In box 1, you will find 5 candies and box 3 but you will not find a key to box 3 so box 3 will remain closed. Total number of candies collected = 7 + 4 + 5 = 16 candy.

Example 2:

Input:

status = [1,0,0,0,0,0], candies = [1,1,1,1,1,1], keys = [[1,2,3,4,5],[],[],[],[],[]], containedBoxes = [[1,2,3,4,5],[],[],[],[],[]], initialBoxes = [0]

Output:

6

Explanation:

You have initially box 0. Opening it you can find boxes 1,2,3,4 and 5 and their keys. The total number of candies will be 6.

Constraints:

n == status.length == candies.length == keys.length == containedBoxes.length

1 <= n <= 1000

status[i]

is either

0

or

1

.

1 <= candies[i] <= 1000

0 <= keys[i].length <= n

0 <= keys[i][j] < n

All values of

keys[i]

are

unique

.

0 <= containedBoxes[i].length <= n

0 <= containedBoxes[i][j] < n

All values of

containedBoxes[i]

are unique.

Each box is contained in one box at most.

0 <= initialBoxes.length <= n

0 <= initialBoxes[i] < n

Code Snippets

C++:

```
class Solution {
public:
int maxCandies(vector<int>& status, vector<int>& candies,
vector<vector<int>>& keys, vector<vector<int>>& containedBoxes, vector<int>&
initialBoxes) {

}
};
```

Java:

```
class Solution {
public int maxCandies(int[] status, int[] candies, int[][] keys, int[][]
containedBoxes, int[] initialBoxes) {

}
}
```

Python3:

```
class Solution:
def maxCandies(self, status: List[int], candies: List[int], keys:
List[List[int]], containedBoxes: List[List[int]], initialBoxes: List[int]) ->
int:
```

Python:

```
class Solution(object):
def maxCandies(self, status, candies, keys, containedBoxes, initialBoxes):
"""
:type status: List[int]
:type candies: List[int]
:type keys: List[List[int]]
```

```
:type containedBoxes: List[List[int]]
:type initialBoxes: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} status
* @param {number[]} candies
* @param {number[][]} keys
* @param {number[][]} containedBoxes
* @param {number[]} initialBoxes
* @return {number}
*/
var maxCandies = function(status, candies, keys, containedBoxes,
initialBoxes) {

};
```

TypeScript:

```
function maxCandies(status: number[], candies: number[], keys: number[][],
containedBoxes: number[][], initialBoxes: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxCandies(int[] status, int[] candies, int[][] keys, int[][]
containedBoxes, int[] initialBoxes) {

}
}
```

C:

```
int maxCandies(int* status, int statusSize, int* candies, int candiesSize,
int** keys, int keysSize, int* keysColSize, int** containedBoxes, int
containedBoxesSize, int* containedBoxesColSize, int* initialBoxes, int
initialBoxesSize) {
```

```
}
```

Go:

```
func maxCandies(status []int, candies []int, keys [][]int, containedBoxes
[][]int, initialBoxes []int) int {

}
```

Kotlin:

```
class Solution {
fun maxCandies(status: IntArray, candies: IntArray, keys: Array<IntArray>,
containedBoxes: Array<IntArray>, initialBoxes: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxCandies(_ status: [Int], _ candies: [Int], _ keys: [[Int]], _
containedBoxes: [[Int]], _ initialBoxes: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_candies(status: Vec<i32>, candies: Vec<i32>, keys: Vec<Vec<i32>>,
contained_boxes: Vec<Vec<i32>>, initial_boxes: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} status
# @param {Integer[]} candies
# @param {Integer[][]} keys
```

```
# @param {Integer[][]} contained_boxes
# @param {Integer[]} initial_boxes
# @return {Integer}
def max_candies(status, candies, keys, contained_boxes, initial_boxes)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $status
* @param Integer[] $candies
* @param Integer[][] $keys
* @param Integer[][] $containedBoxes
* @param Integer[] $initialBoxes
* @return Integer
*/
function maxCandies($status, $candies, $keys, $containedBoxes, $initialBoxes)
{

}
}
```

Dart:

```
class Solution {
int maxCandies(List<int> status, List<int> candies, List<List<int>> keys,
List<List<int>> containedBoxes, List<int> initialBoxes) {

}
}
```

Scala:

```
object Solution {
def maxCandies(status: Array[Int], candies: Array[Int], keys:
Array[Array[Int]], containedBoxes: Array[Array[Int]], initialBoxes:
Array[Int]): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec max_candies(status :: [integer], candies :: [integer], keys ::
[[integer]], contained_boxes :: [[integer]], initial_boxes :: [integer]) ::
integer
def max_candies(status, candies, keys, contained_boxes, initial_boxes) do

end
end
```

Erlang:

```
-spec max_candies(Status :: [integer()], Candies :: [integer()], Keys ::
[[integer()]], ContainedBoxes :: [[integer()]], InitialBoxes :: [integer()])
-> integer().
max_candies(Status, Candies, Keys, ContainedBoxes, InitialBoxes) ->
.
```

Racket:

```
(define/contract (max-candies status candies keys containedBoxes
initialBoxes)
(-> (listof exact-integer?) (listof exact-integer?) (listof (listof
exact-integer?)) (listof (listof exact-integer?)) (listof exact-integer?)
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Candies You Can Get from Boxes
* Difficulty: Hard
* Tags: array, graph, 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 maxCandies(vector<int>& status, vector<int>& candies,
vector<vector<int>>& keys, vector<vector<int>>& containedBoxes, vector<int>&
initialBoxes) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Candies You Can Get from Boxes
* Difficulty: Hard
* Tags: array, graph, 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 maxCandies(int[] status, int[] candies, int[][] keys, int[][]
containedBoxes, int[] initialBoxes) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Candies You Can Get from Boxes
Difficulty: Hard
Tags: array, graph, 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 maxCandies(self, status: List[int], candies: List[int], keys:
List[List[int]], containedBoxes: List[List[int]], initialBoxes: List[int]) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxCandies(self, status, candies, keys, containedBoxes, initialBoxes):
"""
:type status: List[int]
:type candies: List[int]
:type keys: List[List[int]]
:type containedBoxes: List[List[int]]
:type initialBoxes: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Candies You Can Get from Boxes
* Difficulty: Hard
* Tags: array, graph, 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[]} status
* @param {number[]} candies
* @param {number[][]} keys
* @param {number[][]} containedBoxes
* @param {number[]} initialBoxes
```

```
* @return {number}
*/
var maxCandies = function(status, candies, keys, containedBoxes,
initialBoxes) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Candies You Can Get from Boxes
* Difficulty: Hard
* Tags: array, graph, 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 maxCandies(status: number[], candies: number[], keys: number[][],
containedBoxes: number[][], initialBoxes: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Candies You Can Get from Boxes
* Difficulty: Hard
* Tags: array, graph, 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 MaxCandies(int[] status, int[] candies, int[][] keys, int[][]
containedBoxes, int[] initialBoxes) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Maximum Candies You Can Get from Boxes
* Difficulty: Hard
* Tags: array, graph, 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 maxCandies(int* status, int statusSize, int* candies, int candiesSize,
int** keys, int keysSize, int* keysColSize, int** containedBoxes, int
containedBoxesSize, int* containedBoxesColSize, int* initialBoxes, int
initialBoxesSize) {

}
```

Go Solution:

```
// Problem: Maximum Candies You Can Get from Boxes
// Difficulty: Hard
// Tags: array, graph, 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 maxCandies(status []int, candies []int, keys [][]int, containedBoxes
[][]int, initialBoxes []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxCandies(status: IntArray, candies: IntArray, keys: Array<IntArray>,
containedBoxes: Array<IntArray>, initialBoxes: IntArray): Int {
```

```
}
}
```

Swift Solution:

```
class Solution {
func maxCandies(_ status: [Int], _ candies: [Int], _ keys: [[Int]], _
containedBoxes: [[Int]], _ initialBoxes: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Candies You Can Get from Boxes
// Difficulty: Hard
// Tags: array, graph, 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_candies(status: Vec<i32>, candies: Vec<i32>, keys: Vec<Vec<i32>>,
contained_boxes: Vec<Vec<i32>>, initial_boxes: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} status
# @param {Integer[]} candies
# @param {Integer[][]} keys
# @param {Integer[][]} contained_boxes
# @param {Integer[]} initial_boxes
# @return {Integer}
def max_candies(status, candies, keys, contained_boxes, initial_boxes)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $status
* @param Integer[] $candies
* @param Integer[][] $keys
* @param Integer[][] $containedBoxes
* @param Integer[] $initialBoxes
* @return Integer
*/
function maxCandies($status, $candies, $keys, $containedBoxes, $initialBoxes)
{

}
}
```

Dart Solution:

```
class Solution {
int maxCandies(List<int> status, List<int> candies, List<List<int>> keys,
List<List<int>> containedBoxes, List<int> initialBoxes) {

}
}
```

Scala Solution:

```
object Solution {
def maxCandies(status: Array[Int], candies: Array[Int], keys:
Array[Array[Int]], containedBoxes: Array[Array[Int]], initialBoxes:
Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_candies(status :: [integer], candies :: [integer], keys ::
[[integer]], contained_boxes :: [[integer]], initial_boxes :: [integer]) ::
```

```
integer
def max_candies(status, candies, keys, contained_boxes, initial_boxes) do

end
end
```

Erlang Solution:

```
-spec max_candies(Status :: [integer()], Candies :: [integer()], Keys ::
[[integer()]], ContainedBoxes :: [[integer()]], InitialBoxes :: [integer()])
-> integer().
max_candies(Status, Candies, Keys, ContainedBoxes, InitialBoxes) ->
.
```

Racket Solution:

```
(define/contract (max-candies status candies keys containedBoxes
initialBoxes)
(-> (listof exact-integer?) (listof exact-integer?) (listof (listof
exact-integer?)) (listof (listof exact-integer?)) (listof exact-integer?)
exact-integer?)
)
```