Problem 1203: Sort Items by Groups Respecting Dependencies

Problem Information

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

Problem Description

There are

n

items each belonging to zero or one of

m

groups where

group[i]

is the group that the

i

-th item belongs to and it's equal to

-1

if the

i

-th item belongs to no group. The items and the groups are zero indexed. A group can have no item belonging to it.

Return a sorted list of the items such that:

The items that belong to the same group are next to each other in the sorted list.

There are some relations between these items where

beforeItems[i]

is a list containing all the items that should come before the

i

-th item in the sorted array (to the left of the

i

-th item).

Return any solution if there is more than one solution and return an

empty list

if there is no solution.

Example 1:

Item	Group	Before
0	-1	
1	-1	6
2	1	5
3	0	6
4	0	3, 6
5	1	
6	0	
7	-1	

Input:

n = 8, m = 2, group = [-1,-1,1,0,0,1,0,-1], beforeItems = [[],[6],[5],[6],[3,6],[],[],[]]

Output:

[6,3,4,1,5,2,0,7]

Example 2:

Input:

n = 8, m = 2, group = [-1,-1,1,0,0,1,0,-1], beforeItems = [[],[6],[5],[6],[3],[],[4],[]]

Output:

[]

Explanation:

This is the same as example 1 except that 4 needs to be before 6 in the sorted list.

Constraints:

1 <= m <= n <= 3 * 10

4

group.length == beforeItems.length == n

-1 <= group[i] <= m - 1

0 <= beforeItems[i].length <= n - 1

0 <= beforeItems[i][j] <= n - 1

i != beforeItems[i][j]

beforeItems[i]

does not contain duplicates elements.

Code Snippets

C++:

```
class Solution {
public:
vector<int> sortItems(int n, int m, vector<int>& group, vector<vector<int>>&
beforeItems) {

}
};
```

Java:

```
class Solution {
public int[] sortItems(int n, int m, int[] group, List<List<Integer>>
beforeItems) {

}
}
```

Python3:

```
class Solution:
def sortItems(self, n: int, m: int, group: List[int], beforeItems:
List[List[int]]) -> List[int]:
```

Python:

```
class Solution(object):
def sortItems(self, n, m, group, beforeItems):
"""
:type n: int
:type m: int
:type group: List[int]
:type beforeItems: List[List[int]]
:rtype: List[int]
```

```
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} m
* @param {number[]} group
* @param {number[][]} beforeItems
* @return {number[]}
*/
var sortItems = function(n, m, group, beforeItems) {

};
```

TypeScript:

```
function sortItems(n: number, m: number, group: number[], beforeItems:
number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] SortItems(int n, int m, int[] group, IList<IList<int>>
beforeItems) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* sortItems(int n, int m, int* group, int groupSize, int** beforeItems,
int beforeItemsSize, int* beforeItemsColSize, int* returnSize) {

}
```

Go:

```
func sortItems(n int, m int, group []int, beforeItems [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun sortItems(n: Int, m: Int, group: IntArray, beforeItems: List<List<Int>>):
IntArray {

}
}
```

Swift:

```
class Solution {
func sortItems(_ n: Int, _ m: Int, _ group: [Int], _ beforeItems: [[Int]]) ->
[Int] {

}
}
```

Rust:

```
impl Solution {
pub fn sort_items(n: i32, m: i32, group: Vec<i32>, before_items:
Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} m
# @param {Integer[]} group
# @param {Integer[][]} before_items
# @return {Integer[]}
def sort_items(n, m, group, before_items)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $m
* @param Integer[] $group
* @param Integer[][] $beforeItems
* @return Integer[]
*/
function sortItems($n, $m, $group, $beforeItems) {

}
}
```

Dart:

```
class Solution {
List<int> sortItems(int n, int m, List<int> group, List<List<int>>
beforeItems) {

}
}
```

Scala:

```
object Solution {
def sortItems(n: Int, m: Int, group: Array[Int], beforeItems:
List[List[Int]]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec sort_items(n :: integer, m :: integer, group :: [integer], before_items
:: [[integer]]) :: [integer]
def sort_items(n, m, group, before_items) do

end
end
```

Erlang:

```
-spec sort_items(N :: integer(), M :: integer(), Group :: [integer()],
BeforeItems :: [[integer()]]) -> [integer()].
sort_items(N, M, Group, BeforeItems) ->
.
```

Racket:

```
(define/contract (sort-items n m group beforeItems)
(-> exact-integer? exact-integer? (listof exact-integer?) (listof (listof
exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sort Items by Groups Respecting Dependencies
* Difficulty: Hard
* Tags: array, graph, 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:
vector<int> sortItems(int n, int m, vector<int>& group, vector<vector<int>>&
beforeItems) {

}
};
```

Java Solution:

```
/**
* Problem: Sort Items by Groups Respecting Dependencies
* Difficulty: Hard
* Tags: array, graph, 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[] sortItems(int n, int m, int[] group, List<List<Integer>>
beforeItems) {

}
}
```

Python3 Solution:

```
"""
Problem: Sort Items by Groups Respecting Dependencies
Difficulty: Hard
Tags: array, graph, 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 sortItems(self, n: int, m: int, group: List[int], beforeItems:
List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def sortItems(self, n, m, group, beforeItems):
"""
:type n: int
:type m: int
:type group: List[int]
:type beforeItems: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Sort Items by Groups Respecting Dependencies
* Difficulty: Hard
* Tags: array, graph, 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} n
* @param {number} m
* @param {number[]} group
* @param {number[][]} beforeItems
* @return {number[]}
*/
var sortItems = function(n, m, group, beforeItems) {

};
```

TypeScript Solution:

```
/**
* Problem: Sort Items by Groups Respecting Dependencies
* Difficulty: Hard
* Tags: array, graph, 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 sortItems(n: number, m: number, group: number[], beforeItems:
number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Sort Items by Groups Respecting Dependencies
* Difficulty: Hard
* Tags: array, graph, 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[] SortItems(int n, int m, int[] group, IList<IList<int>>
beforeItems) {

}
}
```

C Solution:

```
/*
* Problem: Sort Items by Groups Respecting Dependencies
* Difficulty: Hard
* Tags: array, graph, 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* sortItems(int n, int m, int* group, int groupSize, int** beforeItems,
int beforeItemsSize, int* beforeItemsColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Sort Items by Groups Respecting Dependencies
// Difficulty: Hard
// Tags: array, graph, 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 sortItems(n int, m int, group []int, beforeItems [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun sortItems(n: Int, m: Int, group: IntArray, beforeItems: List<List<Int>>):
IntArray {

}
}
```

Swift Solution:

```
class Solution {
func sortItems(_ n: Int, _ m: Int, _ group: [Int], _ beforeItems: [[Int]]) ->
[Int] {

}
}
```

Rust Solution:

```
// Problem: Sort Items by Groups Respecting Dependencies
// Difficulty: Hard
// Tags: array, graph, 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 sort_items(n: i32, m: i32, group: Vec<i32>, before_items:
Vec<Vec<i32>>) -> Vec<i32> {
```

```
}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} m
# @param {Integer[]} group
# @param {Integer[][]} before_items
# @return {Integer[]}
def sort_items(n, m, group, before_items)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $m
* @param Integer[] $group
* @param Integer[][] $beforeItems
* @return Integer[]
*/
function sortItems($n, $m, $group, $beforeItems) {

}
}
```

Dart Solution:

```
class Solution {
List<int> sortItems(int n, int m, List<int> group, List<List<int>>
beforeItems) {

}
}
```

Scala Solution:

```
object Solution {
def sortItems(n: Int, m: Int, group: Array[Int], beforeItems:
List[List[Int]]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec sort_items(n :: integer, m :: integer, group :: [integer], before_items
:: [[integer]]) :: [integer]
def sort_items(n, m, group, before_items) do

end
end
```

Erlang Solution:

```
-spec sort_items(N :: integer(), M :: integer(), Group :: [integer()],
BeforeItems :: [[integer()]]) -> [integer()].
sort_items(N, M, Group, BeforeItems) ->
.
```

Racket Solution:

```
(define/contract (sort-items n m group beforeItems)
(-> exact-integer? exact-integer? (listof exact-integer?) (listof (listof
exact-integer?)) (listof exact-integer?))
)
```