Problem 1743: Restore the Array From Adjacent Pairs

Problem Information

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

Problem Description

There is an integer array

nums

that consists of

n

unique

elements, but you have forgotten it. However, you do remember every pair of adjacent elements in

nums

.

You are given a 2D integer array

adjacentPairs

of size

n - 1

where each

adjacentPairs[i] = [u

i

, v

i

]

indicates that the elements

u

i

and

v

i

are adjacent in

nums

.

It is guaranteed that every adjacent pair of elements

nums[i]

and

nums[i+1]

will exist in

adjacentPairs

, either as

[nums[i], nums[i+1]]

or

[nums[i+1], nums[i]]

. The pairs can appear

in any order

.

Return

the original array

nums

. If there are multiple solutions, return

any of them

.

Example 1:

Input:

adjacentPairs = [[2,1],[3,4],[3,2]]

Output:

[1,2,3,4]

Explanation:

This array has all its adjacent pairs in adjacentPairs. Notice that adjacentPairs[i] may not be in left-to-right order.

Example 2:

Input:

adjacentPairs = [[4,-2],[1,4],[-3,1]]

Output:

[-2,4,1,-3]

Explanation:

There can be negative numbers. Another solution is [-3,1,4,-2], which would also be accepted.

Example 3:

Input:

adjacentPairs = [[100000,-100000]]

Output:

[100000,-100000]

Constraints:

nums.length == n

adjacentPairs.length == n - 1

adjacentPairs[i].length == 2

2 <= n <= 10

5

-10

5

<= nums[i], u

i

, v

i

<= 10

5

There exists some

nums

that has

adjacentPairs

as its pairs.

Code Snippets

C++:

```
class Solution {
public:
vector<int> restoreArray(vector<vector<int>>& adjacentPairs) {

}
};
```

Java:

```
class Solution {
public int[] restoreArray(int[][] adjacentPairs) {

}
}
```

Python3:

```
class Solution:
def restoreArray(self, adjacentPairs: List[List[int]]) -> List[int]:
```

Python:

```
class Solution(object):
def restoreArray(self, adjacentPairs):
"""
:type adjacentPairs: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[][]} adjacentPairs
* @return {number[]}
*/
var restoreArray = function(adjacentPairs) {

};
```

TypeScript:

```
function restoreArray(adjacentPairs: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] RestoreArray(int[][] adjacentPairs) {
```

```
}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* restoreArray(int** adjacentPairs, int adjacentPairsSize, int*
adjacentPairsColSize, int* returnSize) {

}
```

Go:

```
func restoreArray(adjacentPairs [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun restoreArray(adjacentPairs: Array<IntArray>): IntArray {

}
}
```

Swift:

```
class Solution {
func restoreArray(_ adjacentPairs: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn restore_array(adjacent_pairs: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} adjacent_pairs
# @return {Integer[]}
def restore_array(adjacent_pairs)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $adjacentPairs
* @return Integer[]
*/
function restoreArray($adjacentPairs) {

}
}
```

Dart:

```
class Solution {
List<int> restoreArray(List<List<int>> adjacentPairs) {

}
}
```

Scala:

```
object Solution {
def restoreArray(adjacentPairs: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec restore_array(adjacent_pairs :: [[integer]]) :: [integer]
def restore_array(adjacent_pairs) do
```

```
end
end
```

Erlang:

```
-spec restore_array(AdjacentPairs :: [[integer()]]) -> [integer()].
restore_array(AdjacentPairs) ->
.
```

Racket:

```
(define/contract (restore-array adjacentPairs)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Restore the Array From Adjacent Pairs
* Difficulty: Medium
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<int> restoreArray(vector<vector<int>>& adjacentPairs) {

}
};
```

Java Solution:

```
/**
* Problem: Restore the Array From Adjacent Pairs
```

```
* Difficulty: Medium
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int[] restoreArray(int[][] adjacentPairs) {

}
}
```

Python3 Solution:

```
"""
Problem: Restore the Array From Adjacent Pairs
Difficulty: Medium
Tags: array, hash, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def restoreArray(self, adjacentPairs: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def restoreArray(self, adjacentPairs):
"""
:type adjacentPairs: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Restore the Array From Adjacent Pairs
* Difficulty: Medium
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[][]} adjacentPairs
* @return {number[]}
*/
var restoreArray = function(adjacentPairs) {

};
```

TypeScript Solution:

```
/**
* Problem: Restore the Array From Adjacent Pairs
* Difficulty: Medium
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function restoreArray(adjacentPairs: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Restore the Array From Adjacent Pairs
* Difficulty: Medium
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int[] RestoreArray(int[][] adjacentPairs) {

}
}
```

C Solution:

```
/*
* Problem: Restore the Array From Adjacent Pairs
* Difficulty: Medium
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* restoreArray(int** adjacentPairs, int adjacentPairsSize, int*
adjacentPairsColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Restore the Array From Adjacent Pairs
// Difficulty: Medium
// Tags: array, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func restoreArray(adjacentPairs [][]int) []int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun restoreArray(adjacentPairs: Array<IntArray>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func restoreArray(_ adjacentPairs: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Restore the Array From Adjacent Pairs
// Difficulty: Medium
// Tags: array, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn restore_array(adjacent_pairs: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} adjacent_pairs
# @return {Integer[]}
def restore_array(adjacent_pairs)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $adjacentPairs
* @return Integer[]
*/
function restoreArray($adjacentPairs) {

}
}
```

Dart Solution:

```
class Solution {
List<int> restoreArray(List<List<int>> adjacentPairs) {

}
}
```

Scala Solution:

```
object Solution {
def restoreArray(adjacentPairs: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec restore_array(adjacent_pairs :: [[integer]]) :: [integer]
def restore_array(adjacent_pairs) do

end
end
```

Erlang Solution:

```
-spec restore_array(AdjacentPairs :: [[integer()]]) -> [integer()].
restore_array(AdjacentPairs) ->
.
```

Racket Solution:

```
(define/contract (restore-array adjacentPairs)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```