

Problem 2421: Number of Good Paths

Problem Information

Difficulty: **Hard**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

There is a tree (i.e. a connected, undirected graph with no cycles) consisting of

n

nodes numbered from

0

to

$n - 1$

and exactly

$n - 1$

edges.

You are given a

0-indexed

integer array

vals

of length

n

where

$vals[i]$

denotes the value of the

i

th

node. You are also given a 2D integer array

edges

where

$edges[i] = [a$

i

, b

i

]

denotes that there exists an

undirected

edge connecting nodes

a

i

and

b

i

.

A

good path

is a simple path that satisfies the following conditions:

The starting node and the ending node have the

same

value.

All nodes between the starting node and the ending node have values

less than or equal to

the starting node (i.e. the starting node's value should be the maximum value along the path).

Return

the number of distinct good paths

.

Note that a path and its reverse are counted as the

same

path. For example,

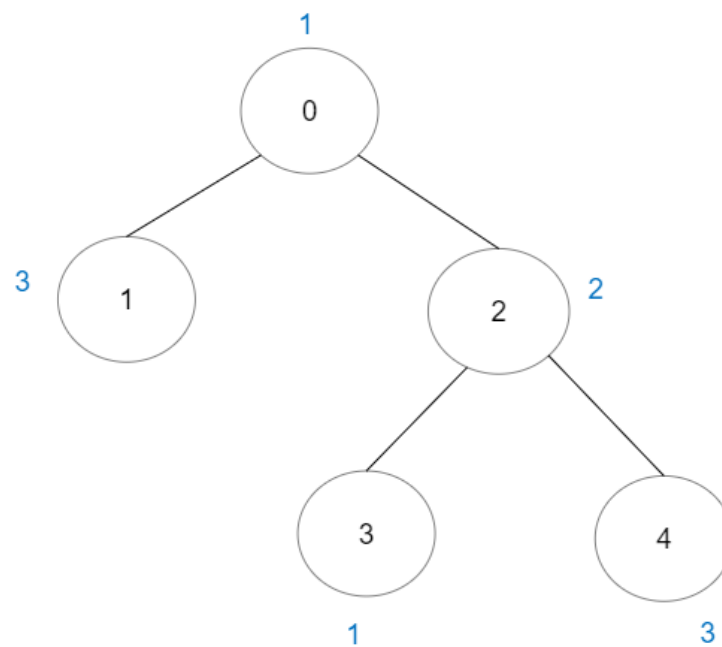
0 -> 1

is considered to be the same as

1 -> 0

. A single node is also considered as a valid path.

Example 1:



Input:

vals = [1,3,2,1,3], edges = [[0,1],[0,2],[2,3],[2,4]]

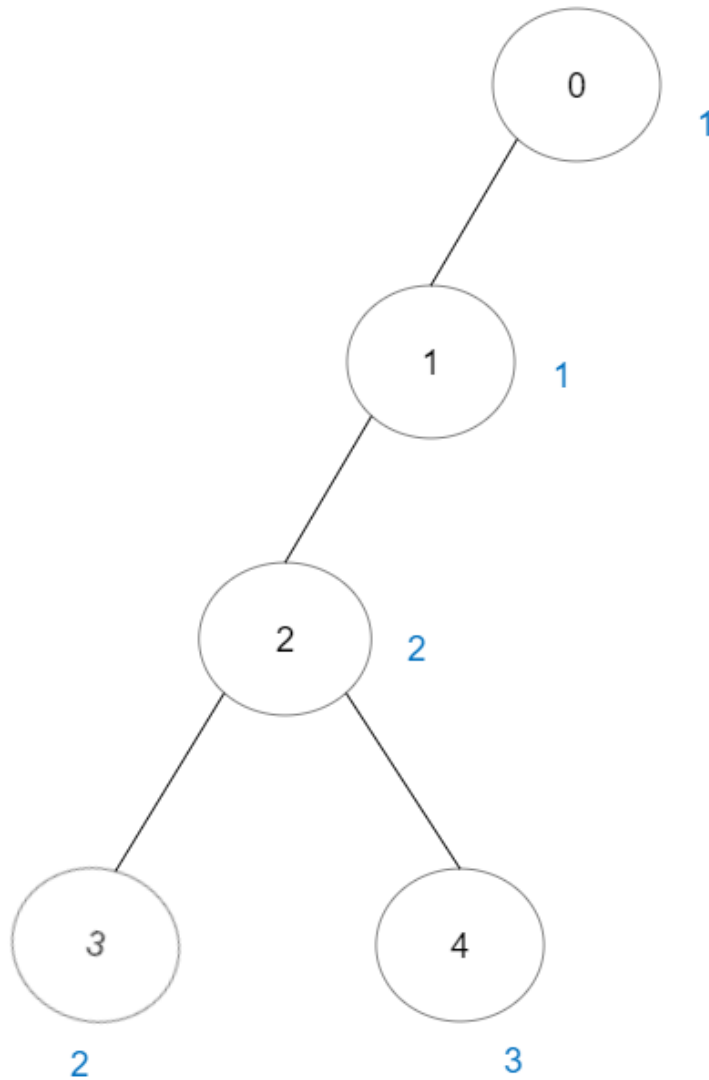
Output:

6

Explanation:

There are 5 good paths consisting of a single node. There is 1 additional good path: 1 -> 0 -> 2 -> 4. (The reverse path 4 -> 2 -> 0 -> 1 is treated as the same as 1 -> 0 -> 2 -> 4.) Note that 0 -> 2 -> 3 is not a good path because $\text{vals}[2] > \text{vals}[0]$.

Example 2:



Input:

`vals = [1,1,2,2,3], edges = [[0,1],[1,2],[2,3],[2,4]]`

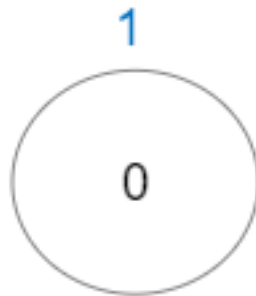
Output:

7

Explanation:

There are 5 good paths consisting of a single node. There are 2 additional good paths: 0 -> 1 and 2 -> 3.

Example 3:



Input:

vals = [1], edges = []

Output:

1

Explanation:

The tree consists of only one node, so there is one good path.

Constraints:

$n == \text{vals.length}$

$1 \leq n \leq 3 * 10$

4

`0 <= vals[i] <= 10`

`5`

`edges.length == n - 1`

`edges[i].length == 2`

`0 <= a`

`i`

`, b`

`i`

`< n`

`a`

`i`

`!= b`

`i`

`edges`

represents a valid tree.

Code Snippets

C++:

```
class Solution {  
public:  
    int numberOfGoodPaths(vector<int>& vals, vector<vector<int>>& edges) {
```

```
}  
};
```

Java:

```
class Solution {  
    public int numberOfGoodPaths(int[] vals, int[][] edges) {  
  
    }  
}
```

Python3:

```
class Solution:  
    def numberOfGoodPaths(self, vals: List[int], edges: List[List[int]]) -> int:
```

Python:

```
class Solution(object):  
    def numberOfGoodPaths(self, vals, edges):  
        """  
        :type vals: List[int]  
        :type edges: List[List[int]]  
        :rtype: int  
        """
```

JavaScript:

```
/**  
 * @param {number[]} vals  
 * @param {number[][]} edges  
 * @return {number}  
 */  
var numberOfGoodPaths = function(vals, edges) {  
  
};
```

TypeScript:

```
function numberOfGoodPaths(vals: number[], edges: number[][]): number {  
  
};
```

C#:

```
public class Solution {  
    public int NumberOfGoodPaths(int[] vals, int[][] edges) {  
  
    }  
}
```

C:

```
int numberOfGoodPaths(int* vals, int valsSize, int** edges, int edgesSize,  
int* edgesColSize) {  
  
}
```

Go:

```
func numberOfGoodPaths(vals []int, edges [][]int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun numberOfGoodPaths(vals: IntArray, edges: Array<IntArray>): Int {  
  
    }  
}
```

Swift:

```
class Solution {  
    func numberOfGoodPaths(_ vals: [Int], _ edges: [[Int]]) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn number_of_good_paths(vals: Vec<i32>, edges: Vec<Vec<i32>> ) -> i32 {  
  
    }  
}
```

```
}
```

Ruby:

```
# @param {Integer[]} vals
# @param {Integer[][]} edges
# @return {Integer}
def number_of_good_paths(vals, edges)

end
```

PHP:

```
class Solution {

    /**
     * @param Integer[] $vals
     * @param Integer[][] $edges
     * @return Integer
     */
    function numberOfGoodPaths($vals, $edges) {

    }

}
```

Dart:

```
class Solution {
  int numberOfGoodPaths(List<int> vals, List<List<int>> edges) {

  }

}
```

Scala:

```
object Solution {
  def numberOfGoodPaths(vals: Array[Int], edges: Array[Array[Int]]): Int = {

  }

}
```

Elixir:

```

defmodule Solution do
  @spec number_of_good_paths(vals :: [integer], edges :: [[integer]]) ::
    integer
  def number_of_good_paths(vals, edges) do

  end

end

```

Erlang:

```

-spec number_of_good_paths(Vals :: [integer()], Edges :: [[integer()]]) ->
integer().
number_of_good_paths(Vals, Edges) ->
.

```

Racket:

```

(define/contract (number-of-good-paths vals edges)
  (-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?)
  )

```

Solutions

C++ Solution:

```

/*
 * Problem: Number of Good Paths
 * Difficulty: Hard
 * Tags: array, tree, graph, hash, sort
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

class Solution {
public:
    int numberOfGoodPaths(vector<int>& vals, vector<vector<int>>& edges) {

    }

};

```

Java Solution:

```
/**
 * Problem: Number of Good Paths
 * Difficulty: Hard
 * Tags: array, tree, graph, hash, sort
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

class Solution {
    public int numberOfGoodPaths(int[] vals, int[][] edges) {

    }
}
```

Python3 Solution:

```
"""
Problem: Number of Good Paths
Difficulty: Hard
Tags: array, tree, graph, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

```
class Solution(object):
    def numberOfGoodPaths(self, vals, edges):
        """
        :type vals: List[int]
        :type edges: List[List[int]]
```

```
:rtype: int
"""
```

JavaScript Solution:

```
/**
 * Problem: Number of Good Paths
 * Difficulty: Hard
 * Tags: array, tree, graph, hash, sort
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

/**
 * @param {number[]} vals
 * @param {number[][]} edges
 * @return {number}
 */
var numberOfGoodPaths = function(vals, edges) {

};
```

TypeScript Solution:

```
/**
 * Problem: Number of Good Paths
 * Difficulty: Hard
 * Tags: array, tree, graph, hash, sort
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

function numberOfGoodPaths(vals: number[], edges: number[][]): number {

};
```

C# Solution:

```

/*
 * Problem: Number of Good Paths
 * Difficulty: Hard
 * Tags: array, tree, graph, hash, sort
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

public class Solution {
    public int NumberOfGoodPaths(int[] vals, int[][] edges) {

    }
}

```

C Solution:

```

/*
 * Problem: Number of Good Paths
 * Difficulty: Hard
 * Tags: array, tree, graph, hash, sort
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(h) for recursion stack where h is height
 */

int numberOfGoodPaths(int* vals, int valsSize, int** edges, int edgesSize,
int* edgesColSize) {

}

```

Go Solution:

```

// Problem: Number of Good Paths
// Difficulty: Hard
// Tags: array, tree, graph, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

```

```

func numberOfGoodPaths(vals []int, edges [][]int) int {

}

```

Kotlin Solution:

```

class Solution {
    fun numberOfGoodPaths(vals: IntArray, edges: Array<IntArray>): Int {

    }
}

```

Swift Solution:

```

class Solution {
    func numberOfGoodPaths(_ vals: [Int], _ edges: [[Int]]) -> Int {

    }
}

```

Rust Solution:

```

// Problem: Number of Good Paths
// Difficulty: Hard
// Tags: array, tree, graph, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
    pub fn number_of_good_paths(vals: Vec<i32>, edges: Vec<Vec<i32>> ) -> i32 {

    }
}

```

Ruby Solution:

```

# @param {Integer[]} vals
# @param {Integer[][]} edges

```

```
# @return {Integer}
def number_of_good_paths(vals, edges)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[] $vals
     * @param Integer[][] $edges
     * @return Integer
     */
    function numberOfGoodPaths($vals, $edges) {

    }

}
```

Dart Solution:

```
class Solution {
  int numberOfGoodPaths(List<int> vals, List<List<int>> edges) {

  }
}
```

Scala Solution:

```
object Solution {
  def numberOfGoodPaths(vals: Array[Int], edges: Array[Array[Int]]): Int = {

  }
}
```

Elixir Solution:

```
defmodule Solution do
  @spec number_of_good_paths(vals :: [integer], edges :: [[integer]]) ::
    integer
  def number_of_good_paths(vals, edges) do
```

```
end  
end
```

Erlang Solution:

```
-spec number_of_good_paths(Vals :: [integer()], Edges :: [[integer()]]) ->  
integer().  
number_of_good_paths(Vals, Edges) ->  
.
```

Racket Solution:

```
(define/contract (number-of-good-paths vals edges)  
  (-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?)  
  )
```