Problem 2368: Reachable Nodes With Restrictions

Problem Information

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

Problem Description

There is an undirected tree with

n

nodes labeled from

0

to

n - 1

and

n - 1

edges.

You are given a 2D integer array

edges

of length

n - 1

where

edges[i] = [a

i

, b

i

]

indicates that there is an edge between nodes

a

i

and

b

i

in the tree. You are also given an integer array

restricted

which represents

restricted

nodes.

Return

the

maximum

number of nodes you can reach from node

0

without visiting a restricted node.

Note that node

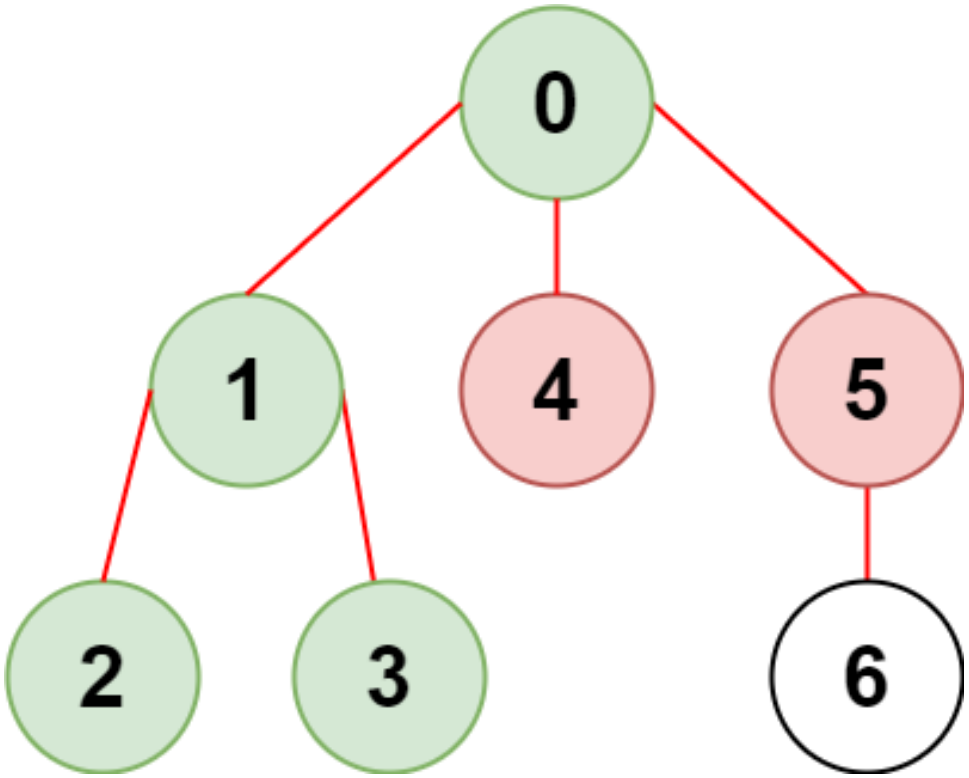
0

will

not

be a restricted node.

Example 1:



Input:

n = 7, edges = [[0,1],[1,2],[3,1],[4,0],[0,5],[5,6]], restricted = [4,5]

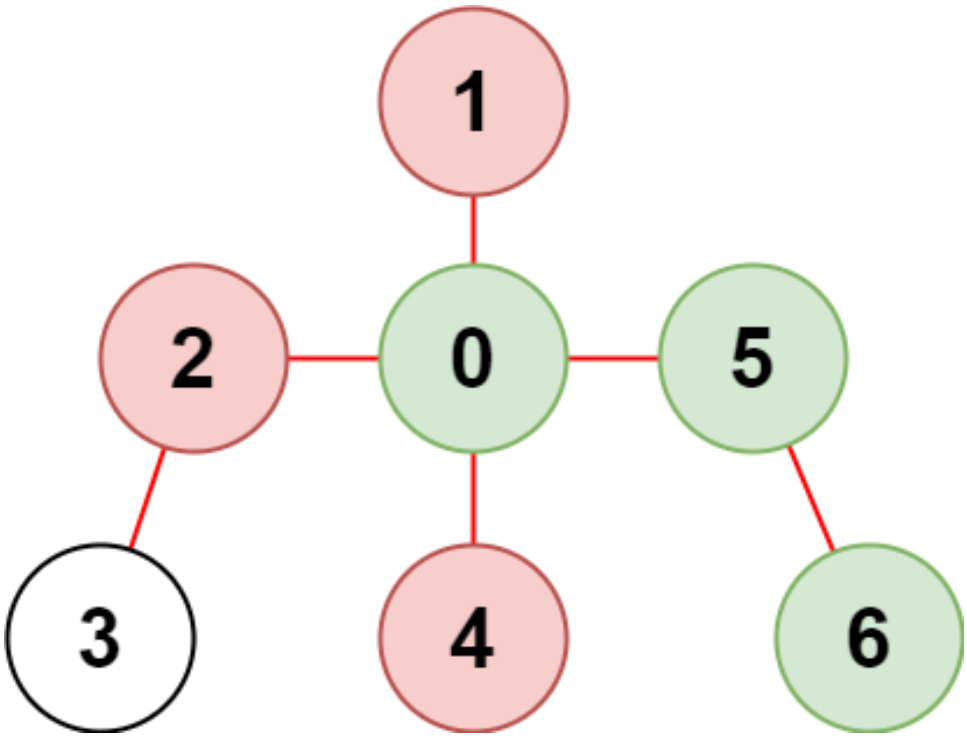
Output:

4

Explanation:

The diagram above shows the tree. We have that [0,1,2,3] are the only nodes that can be reached from node 0 without visiting a restricted node.

Example 2:



Input:

n = 7, edges = [[0,1],[0,2],[0,5],[0,4],[3,2],[6,5]], restricted = [4,2,1]

Output:

3

Explanation:

The diagram above shows the tree. We have that [0,5,6] are the only nodes that can be reached from node 0 without visiting a restricted node.

Constraints:

2 <= n <= 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.

1 <= restricted.length < n

1 <= restricted[i] < n

All the values of

restricted

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
int reachableNodes(int n, vector<vector<int>>& edges, vector<int>&
restricted) {

}
};
```

Java:

```
class Solution {
public int reachableNodes(int n, int[][] edges, int[] restricted) {

}
}
```

Python3:

```
class Solution:
def reachableNodes(self, n: int, edges: List[List[int]], restricted:
List[int]) -> int:
```

Python:

```
class Solution(object):
def reachableNodes(self, n, edges, restricted):
```

```
"""
:type n: int
:type edges: List[List[int]]
:type restricted: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} edges
* @param {number[]} restricted
* @return {number}
*/
var reachableNodes = function(n, edges, restricted) {

};
```

TypeScript:

```
function reachableNodes(n: number, edges: number[][], restricted: number[]):
number {

};
```

C#:

```
public class Solution {
public int ReachableNodes(int n, int[][] edges, int[] restricted) {

}
}
```

C:

```
int reachableNodes(int n, int** edges, int edgesSize, int* edgesColSize, int*
restricted, int restrictedSize) {

}
```

Go:

```
func reachableNodes(n int, edges [][]int, restricted []int) int {

}
```

Kotlin:

```
class Solution {
fun reachableNodes(n: Int, edges: Array<IntArray>, restricted: IntArray): Int
{

}
}
```

Swift:

```
class Solution {
func reachableNodes(_ n: Int, _ edges: [[Int]], _ restricted: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn reachable_nodes(n: i32, edges: Vec<Vec<i32>>, restricted: Vec<i32>) ->
i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer[]} restricted
# @return {Integer}
def reachable_nodes(n, edges, restricted)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer[] $restricted
* @return Integer
*/
function reachableNodes($n, $edges, $restricted) {

}
}
```

Dart:

```
class Solution {
int reachableNodes(int n, List<List<int>> edges, List<int> restricted) {

}
}
```

Scala:

```
object Solution {
def reachableNodes(n: Int, edges: Array[Array[Int]], restricted: Array[Int]):
Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec reachable_nodes(n :: integer, edges :: [[integer]], restricted ::
[integer]) :: integer
def reachable_nodes(n, edges, restricted) do

end
end
```

Erlang:

```
-spec reachable_nodes(N :: integer(), Edges :: [[integer()]], Restricted ::
[integer()]) -> integer().
reachable_nodes(N, Edges, Restricted) ->
.
```

Racket:

```
(define/contract (reachable-nodes n edges restricted)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?)
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Reachable Nodes With Restrictions
* Difficulty: Medium
* Tags: array, tree, graph, hash, search
*
* 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 reachableNodes(int n, vector<vector<int>>& edges, vector<int>&
restricted) {

}
};
```

Java Solution:

```
/**
* Problem: Reachable Nodes With Restrictions
* Difficulty: Medium
* Tags: array, tree, graph, hash, search
*
```

```
* 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 reachableNodes(int n, int[][] edges, int[] restricted) {

}
}
```

Python3 Solution:

```
"""
Problem: Reachable Nodes With Restrictions
Difficulty: Medium
Tags: array, tree, graph, hash, search

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 reachableNodes(self, n: int, edges: List[List[int]], restricted:
List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def reachableNodes(self, n, edges, restricted):
"""
:type n: int
:type edges: List[List[int]]
:type restricted: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Reachable Nodes With Restrictions
* Difficulty: Medium
* Tags: array, tree, graph, hash, search
*
* 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} n
* @param {number[][]} edges
* @param {number[]} restricted
* @return {number}
*/
var reachableNodes = function(n, edges, restricted) {

};
```

TypeScript Solution:

```
/**
* Problem: Reachable Nodes With Restrictions
* Difficulty: Medium
* Tags: array, tree, graph, hash, search
*
* 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 reachableNodes(n: number, edges: number[][], restricted: number[]):
number {

};
```

C# Solution:

```
/*
* Problem: Reachable Nodes With Restrictions
* Difficulty: Medium
```

```
* Tags: array, tree, graph, hash, search
*
* 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 ReachableNodes(int n, int[][] edges, int[] restricted) {

}
}
```

C Solution:

```
/*
* Problem: Reachable Nodes With Restrictions
* Difficulty: Medium
* Tags: array, tree, graph, hash, search
*
* 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 reachableNodes(int n, int** edges, int edgesSize, int* edgesColSize, int*
restricted, int restrictedSize) {

}
```

Go Solution:

```
// Problem: Reachable Nodes With Restrictions
// Difficulty: Medium
// Tags: array, tree, graph, hash, search
//
// 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 reachableNodes(n int, edges [][]int, restricted []int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun reachableNodes(n: Int, edges: Array<IntArray>, restricted: IntArray): Int
{

}
}
```

Swift Solution:

```
class Solution {
func reachableNodes(_ n: Int, _ edges: [[Int]], _ restricted: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Reachable Nodes With Restrictions
// Difficulty: Medium
// Tags: array, tree, graph, hash, search
//
// 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 reachable_nodes(n: i32, edges: Vec<Vec<i32>>, restricted: Vec<i32>) ->
i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
```

```
# @param {Integer[]} restricted
# @return {Integer}
def reachable_nodes(n, edges, restricted)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer[] $restricted
* @return Integer
*/
function reachableNodes($n, $edges, $restricted) {

}
}
```

Dart Solution:

```
class Solution {
int reachableNodes(int n, List<List<int>> edges, List<int> restricted) {

}
}
```

Scala Solution:

```
object Solution {
def reachableNodes(n: Int, edges: Array[Array[Int]], restricted: Array[Int]):
Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec reachable_nodes(n :: integer, edges :: [[integer]], restricted ::
[integer]) :: integer
def reachable_nodes(n, edges, restricted) do

end
end
```

Erlang Solution:

```
-spec reachable_nodes(N :: integer(), Edges :: [[integer()]], Restricted ::
[integer()]) -> integer().
reachable_nodes(N, Edges, Restricted) ->
.
```

Racket Solution:

```
(define/contract (reachable-nodes n edges restricted)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?)
exact-integer?)
)
```