Problem 1245: Tree Diameter

Problem Information

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

Problem Description

The

diameter

of a tree is

the number of edges

in the longest path in that tree.

There is an undirected tree of

n

nodes labeled from

0

to

n - 1

. You are given a 2D array

edges

where

edges.length == n - 1

and

edges[i] = [a

i

, b

i

]

indicates that there is an undirected edge between nodes

a

i

and

b

i

in the tree.

Return

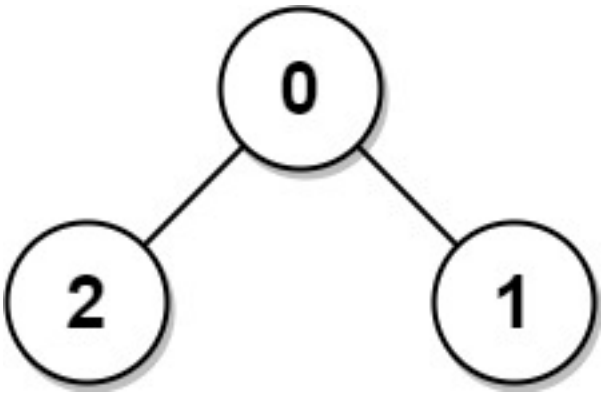
the

diameter

of the tree

.

Example 1:



Input:

edges = [[0,1],[0,2]]

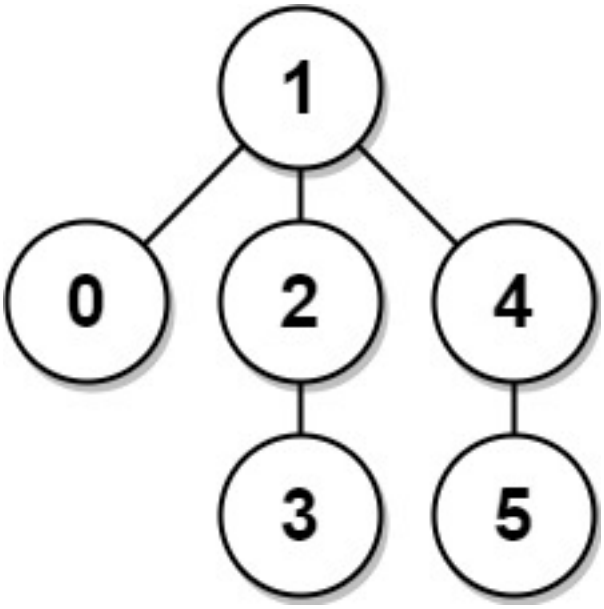
Output:

2

Explanation:

The longest path of the tree is the path 1 - 0 - 2.

Example 2:



Input:

edges = [[0,1],[1,2],[2,3],[1,4],[4,5]]

Output:

4

Explanation:

The longest path of the tree is the path 3 - 2 - 1 - 4 - 5.

Constraints:

n == edges.length + 1

1 <= n <= 10

4

0 <= a

i

, b

i

< n

a

i

!= b

i

Code Snippets

C++:

```
class Solution {
public:
int treeDiameter(vector<vector<int>>& edges) {

}
};
```

Java:

```
class Solution {
public int treeDiameter(int[][] edges) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function treeDiameter(edges: number[][]): number {

};
```

C#:

```
public class Solution {
public int TreeDiameter(int[][] edges) {

}
}
```

C:

```
int treeDiameter(int** edges, int edgesSize, int* edgesColSize) {

}
```

Go:

```
func treeDiameter(edges [][]int) int {

}
```

Kotlin:

```
class Solution {
fun treeDiameter(edges: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func treeDiameter(_ edges: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn tree_diameter(edges: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} edges
# @return {Integer}
def tree_diameter(edges)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $edges
* @return Integer
*/
function treeDiameter($edges) {

}
}
```

Dart:

```
class Solution {
int treeDiameter(List<List<int>> edges) {

}
}
```

Scala:

```
object Solution {
def treeDiameter(edges: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec tree_diameter(edges :: [[integer]]) :: integer
def tree_diameter(edges) do

end
end
```

Erlang:

```
-spec tree_diameter(Edges :: [[integer()]]) -> integer().
tree_diameter(Edges) ->
.
```

Racket:

```
(define/contract (tree-diameter edges)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Tree Diameter
* Difficulty: Medium
* Tags: array, tree, graph, sort, 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 treeDiameter(vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Tree Diameter
* Difficulty: Medium
* Tags: array, tree, graph, sort, 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 treeDiameter(int[][] edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Tree Diameter
Difficulty: Medium
Tags: array, tree, graph, sort, 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 treeDiameter(self, edges: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Tree Diameter
* Difficulty: Medium
* Tags: array, tree, graph, sort, 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[][]} edges
* @return {number}
*/
var treeDiameter = function(edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Tree Diameter
* Difficulty: Medium
* Tags: array, tree, graph, sort, 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 treeDiameter(edges: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Tree Diameter
* Difficulty: Medium
* Tags: array, tree, graph, sort, 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 TreeDiameter(int[][] edges) {

}
}
```

C Solution:

```
/*
* Problem: Tree Diameter
* Difficulty: Medium
* Tags: array, tree, graph, sort, 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 treeDiameter(int** edges, int edgesSize, int* edgesColSize) {

}
```

Go Solution:

```
// Problem: Tree Diameter
// Difficulty: Medium
// Tags: array, tree, graph, sort, 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 treeDiameter(edges [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun treeDiameter(edges: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func treeDiameter(_ edges: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Tree Diameter
// Difficulty: Medium
// Tags: array, tree, graph, sort, 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 tree_diameter(edges: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} edges
# @return {Integer}
def tree_diameter(edges)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $edges
* @return Integer
*/
function treeDiameter($edges) {

}
}
```

Dart Solution:

```
class Solution {
int treeDiameter(List<List<int>> edges) {

}
}
```

Scala Solution:

```
object Solution {
def treeDiameter(edges: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec tree_diameter(edges :: [[integer]]) :: integer
def tree_diameter(edges) do

end
end
```

Erlang Solution:

```
-spec tree_diameter(Edges :: [[integer()]]) -> integer().
tree_diameter(Edges) ->
.
```

Racket Solution:

```
(define/contract (tree-diameter edges)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```