Problem 1443: Minimum Time to Collect All Apples in a Tree

Problem Information

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

Problem Description

Given an undirected tree consisting of

n

vertices numbered from

0

to

n-1

, which has some apples in their vertices. You spend 1 second to walk over one edge of the tree.

Return the minimum time in seconds you have to spend to collect all apples in the tree, starting at

vertex 0

and coming back to this vertex.

The edges of the undirected tree are given in the array

edges

, where

edges[i] = [a

i

, b

i

]

means that exists an edge connecting the vertices

a

i

and

b

i

. Additionally, there is a boolean array

hasApple

, where

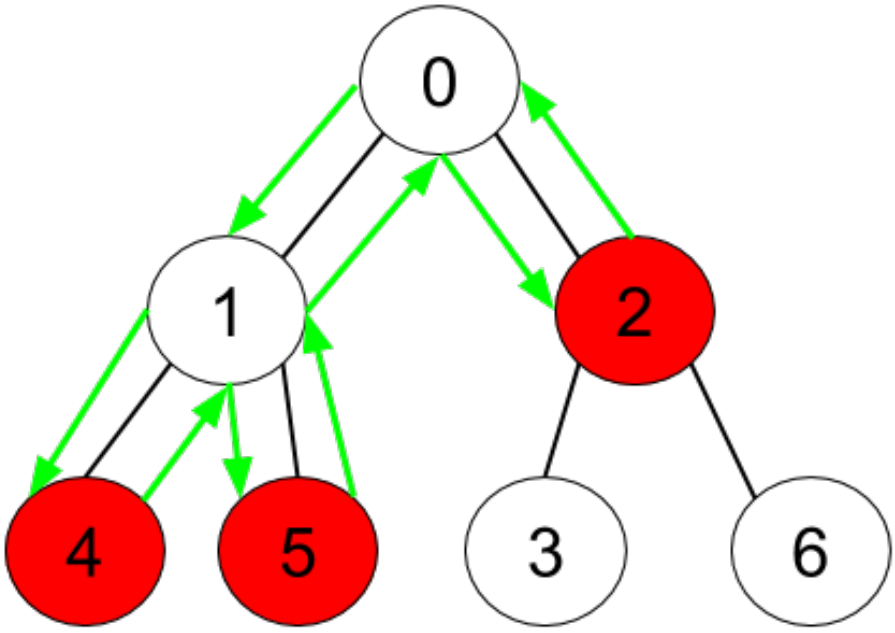
hasApple[i] = true

means that vertex

i

has an apple; otherwise, it does not have any apple.

Example 1:



Input:

n = 7, edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]], hasApple = [false,false,true,false,true,true,false]

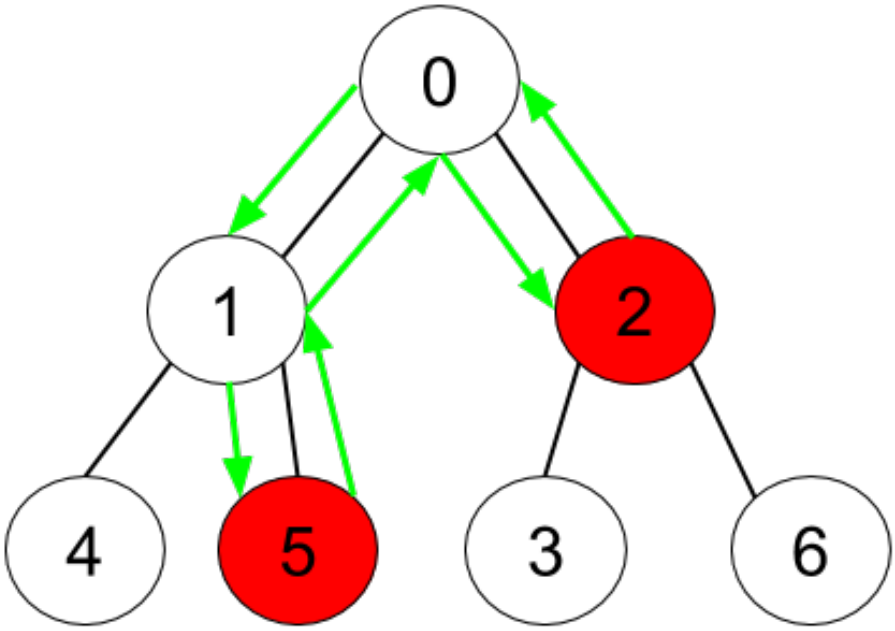
Output:

8

Explanation:

The figure above represents the given tree where red vertices have an apple. One optimal path to collect all apples is shown by the green arrows.

Example 2:



Input:

n = 7, edges = [[0,1],[0,2],[1,4],[1,5],[2,3],[2,6]], hasApple = [false,false,true,false,false,true,false]

Output:

6

Explanation:

The figure above represents the given tree where red vertices have an apple. One optimal path to collect all apples is shown by the green arrows.

Example 3:

Input:

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

Output:

0

Constraints:

1 <= n <= 10

5

edges.length == n - 1

edges[i].length == 2

0 <= a

i

< b

i

<= n - 1

hasApple.length == n

Code Snippets

C++:

```
class Solution {
public:
int minTime(int n, vector<vector<int>>& edges, vector<bool>& hasApple) {

}
};
```

Java:

```
class Solution {
public int minTime(int n, int[][] edges, List<Boolean> hasApple) {

}
```

```
}
```

Python3:

```
class Solution:
def minTime(self, n: int, edges: List[List[int]], hasApple: List[bool]) ->
int:
```

Python:

```
class Solution(object):
def minTime(self, n, edges, hasApple):
"""
:type n: int
:type edges: List[List[int]]
:type hasApple: List[bool]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} edges
* @param {boolean[]} hasApple
* @return {number}
*/
var minTime = function(n, edges, hasApple) {

};
```

TypeScript:

```
function minTime(n: number, edges: number[][], hasApple: boolean[]): number {

};
```

C#:

```
public class Solution {
public int MinTime(int n, int[][] edges, IList<bool> hasApple) {
```

```
}
}
```

C:

```
int minTime(int n, int** edges, int edgesSize, int* edgesColSize, bool*
hasApple, int hasAppleSize) {

}
```

Go:

```
func minTime(n int, edges [][]int, hasApple []bool) int {

}
```

Kotlin:

```
class Solution {
fun minTime(n: Int, edges: Array<IntArray>, hasApple: List<Boolean>): Int {

}
}
```

Swift:

```
class Solution {
func minTime(_ n: Int, _ edges: [[Int]], _ hasApple: [Bool]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_time(n: i32, edges: Vec<Vec<i32>>, has_apple: Vec<bool>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Boolean[]} has_apple
# @return {Integer}
def min_time(n, edges, has_apple)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Boolean[] $hasApple
* @return Integer
*/
function minTime($n, $edges, $hasApple) {

}
}
```

Dart:

```
class Solution {
int minTime(int n, List<List<int>> edges, List<bool> hasApple) {

}
}
```

Scala:

```
object Solution {
def minTime(n: Int, edges: Array[Array[Int]], hasApple: List[Boolean]): Int =
{

}
}
```

Elixir:

```
defmodule Solution do
@spec min_time(n :: integer, edges :: [[integer]], has_apple :: [boolean]) ::
integer
def min_time(n, edges, has_apple) do

end
end
```

Erlang:

```
-spec min_time(N :: integer(), Edges :: [[integer()]], HasApple ::
[boolean()]) -> integer().
min_time(N, Edges, HasApple) ->
.
```

Racket:

```
(define/contract (min-time n edges hasApple)
(-> exact-integer? (listof (listof exact-integer?)) (listof boolean?)
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Time to Collect All Apples in a Tree
* Difficulty: Medium
* Tags: array, tree, 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 minTime(int n, vector<vector<int>>& edges, vector<bool>& hasApple) {

}
```

```
};
```

Java Solution:

```
/**
* Problem: Minimum Time to Collect All Apples in a Tree
* Difficulty: Medium
* Tags: array, tree, 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 minTime(int n, int[][] edges, List<Boolean> hasApple) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Time to Collect All Apples in a Tree
Difficulty: Medium
Tags: array, tree, 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 minTime(self, n: int, edges: List[List[int]], hasApple: List[bool]) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minTime(self, n, edges, hasApple):
"""
:type n: int
:type edges: List[List[int]]
:type hasApple: List[bool]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Time to Collect All Apples in a Tree
* Difficulty: Medium
* Tags: array, tree, 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 {boolean[]} hasApple
* @return {number}
*/
var minTime = function(n, edges, hasApple) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Time to Collect All Apples in a Tree
* Difficulty: Medium
* Tags: array, tree, 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 minTime(n: number, edges: number[][], hasApple: boolean[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Time to Collect All Apples in a Tree
* Difficulty: Medium
* Tags: array, tree, 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 MinTime(int n, int[][] edges, IList<bool> hasApple) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Time to Collect All Apples in a Tree
* Difficulty: Medium
* Tags: array, tree, 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 minTime(int n, int** edges, int edgesSize, int* edgesColSize, bool*
hasApple, int hasAppleSize) {

}
```

Go Solution:

```
// Problem: Minimum Time to Collect All Apples in a Tree
// Difficulty: Medium
// Tags: array, tree, 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 minTime(n int, edges [][]int, hasApple []bool) int {

}
```

Kotlin Solution:

```
class Solution {
fun minTime(n: Int, edges: Array<IntArray>, hasApple: List<Boolean>): Int {

}
}
```

Swift Solution:

```
class Solution {
func minTime(_ n: Int, _ edges: [[Int]], _ hasApple: [Bool]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Time to Collect All Apples in a Tree
// Difficulty: Medium
// Tags: array, tree, 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 min_time(n: i32, edges: Vec<Vec<i32>>, has_apple: Vec<bool>) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Boolean[]} has_apple
# @return {Integer}
def min_time(n, edges, has_apple)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Boolean[] $hasApple
* @return Integer
*/
function minTime($n, $edges, $hasApple) {

}
}
```

Dart Solution:

```
class Solution {
int minTime(int n, List<List<int>> edges, List<bool> hasApple) {

}
}
```

Scala Solution:

```
object Solution {
def minTime(n: Int, edges: Array[Array[Int]], hasApple: List[Boolean]): Int =
{
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_time(n :: integer, edges :: [[integer]], has_apple :: [boolean]) ::
integer
def min_time(n, edges, has_apple) do

end
end
```

Erlang Solution:

```
-spec min_time(N :: integer(), Edges :: [[integer()]], HasApple ::
[boolean()]) -> integer().
min_time(N, Edges, HasApple) ->
.
```

Racket Solution:

```
(define/contract (min-time n edges hasApple)
(-> exact-integer? (listof (listof exact-integer?)) (listof boolean?)
exact-integer?)
)
```