Problem 2378: Choose Edges to Maximize Score in a Tree

Problem Information

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

Problem Description

You are given a

weighted

tree consisting of

n

nodes numbered from

0

to

n - 1

.

The tree is

rooted

at node

0

and represented with a

2D

array

edges

of size

n

where

edges[i] = [par

i

, weight

i

]

indicates that node

par

i

is the

parent

of node

i

, and the edge between them has a weight equal to

weight

i

. Since the root does

not

have a parent, you have

edges[0] = [-1, -1]

.

Choose some edges from the tree such that no two chosen edges are

adjacent

and the

sum

of the weights of the chosen edges is maximized.

Return

the

maximum

sum of the chosen edges

.

Note

:

You are allowed to

not

choose any edges in the tree, the sum of weights in this case will be

0

.

Two edges

Edge

1

and

Edge

2

in the tree are

adjacent

if they have a

common

node.

In other words, they are adjacent if

Edge

1

connects nodes

a

and

b

and

Edge

2

connects nodes

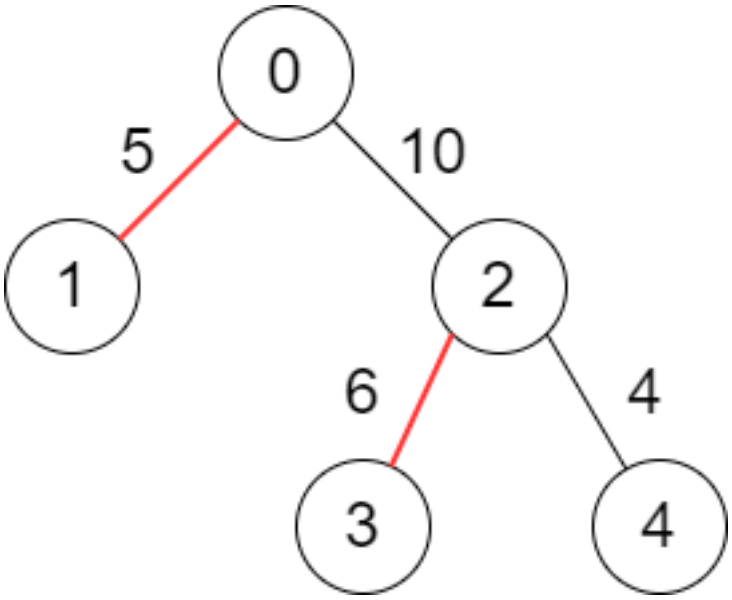
b

and

c

.

Example 1:



Input:

edges = [[-1,-1],[0,5],[0,10],[2,6],[2,4]]

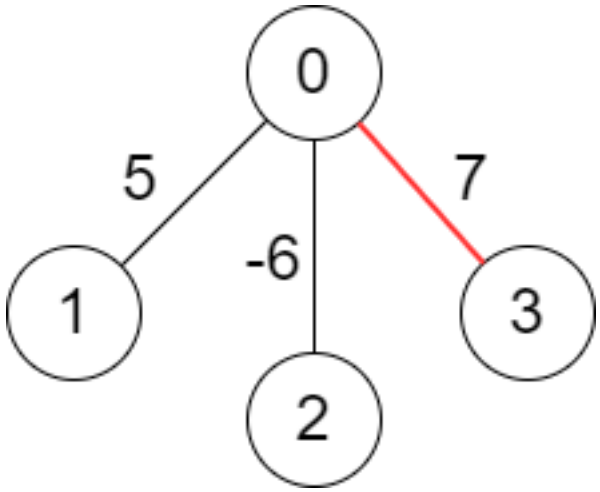
Output:

11

Explanation:

The above diagram shows the edges that we have to choose colored in red. The total score is 5 + 6 = 11. It can be shown that no better score can be obtained.

Example 2:



Input:

edges = [[-1,-1],[0,5],[0,-6],[0,7]]

Output:

7

Explanation:

We choose the edge with weight 7. Note that we cannot choose more than one edge because all edges are adjacent to each other.

Constraints:

n == edges.length

1 <= n <= 10

5

edges[i].length == 2

par

0

== weight

0

== -1

0 <= par

i

<= n - 1

for all

i >= 1

.

par

i

!= i

-10

6

<= weight

i

<= 10

6

for all

i >= 1

.

edges

represents a valid tree.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long maxScore(int[][] edges) {

}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long MaxScore(int[][] edges) {

}
}
```

C:

```
long long maxScore(int** edges, int edgesSize, int* edgesColSize) {

}
```

Go:

```
func maxScore(edges [][]int) int64 {

}
```

Kotlin:

```
class Solution {
fun maxScore(edges: Array<IntArray>): Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn max_score(edges: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-score edges)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Choose Edges to Maximize Score in a Tree
* Difficulty: Medium
* Tags: array, tree, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
long long maxScore(vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Choose Edges to Maximize Score in a Tree
* Difficulty: Medium
* Tags: array, tree, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public long maxScore(int[][] edges) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Choose Edges to Maximize Score in a Tree
Difficulty: Medium
Tags: array, tree, dp, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Choose Edges to Maximize Score in a Tree
* Difficulty: Medium
* Tags: array, tree, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Choose Edges to Maximize Score in a Tree
* Difficulty: Medium
* Tags: array, tree, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxScore(edges: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Choose Edges to Maximize Score in a Tree
* Difficulty: Medium
* Tags: array, tree, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public long MaxScore(int[][] edges) {

}
}
```

C Solution:

```
/*
* Problem: Choose Edges to Maximize Score in a Tree
* Difficulty: Medium
* Tags: array, tree, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

long long maxScore(int** edges, int edgesSize, int* edgesColSize) {

}
```

Go Solution:

```
// Problem: Choose Edges to Maximize Score in a Tree
// Difficulty: Medium
// Tags: array, tree, dp, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxScore(edges [][]int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maxScore(edges: Array<IntArray>): Long {

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Choose Edges to Maximize Score in a Tree
// Difficulty: Medium
// Tags: array, tree, dp, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn max_score(edges: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-score edges)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```