Problem 3547: Maximum Sum of Edge Values in a Graph

Problem Information

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

Problem Description

You are given an

undirected connected

graph of

n

nodes, numbered from

0

to

n - 1

. Each node is connected to

at most

2 other nodes.

The graph consists of

m

edges, represented by a 2D array

edges

, where

edges[i] = [a

i

, b

i

]

indicates that there is an edge between nodes

a

i

and

b

i

.

You have to assign a

unique

value from

1

to

n

to each node. The value of an edge will be the

product

of the values assigned to the two nodes it connects.

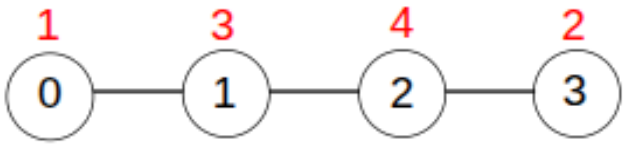
Your score is the sum of the values of all edges in the graph.

Return the

maximum

score you can achieve.

Example 1:



Input:

n = 4, edges =

[[0,1],[1,2],[2,3]]

Output:

23

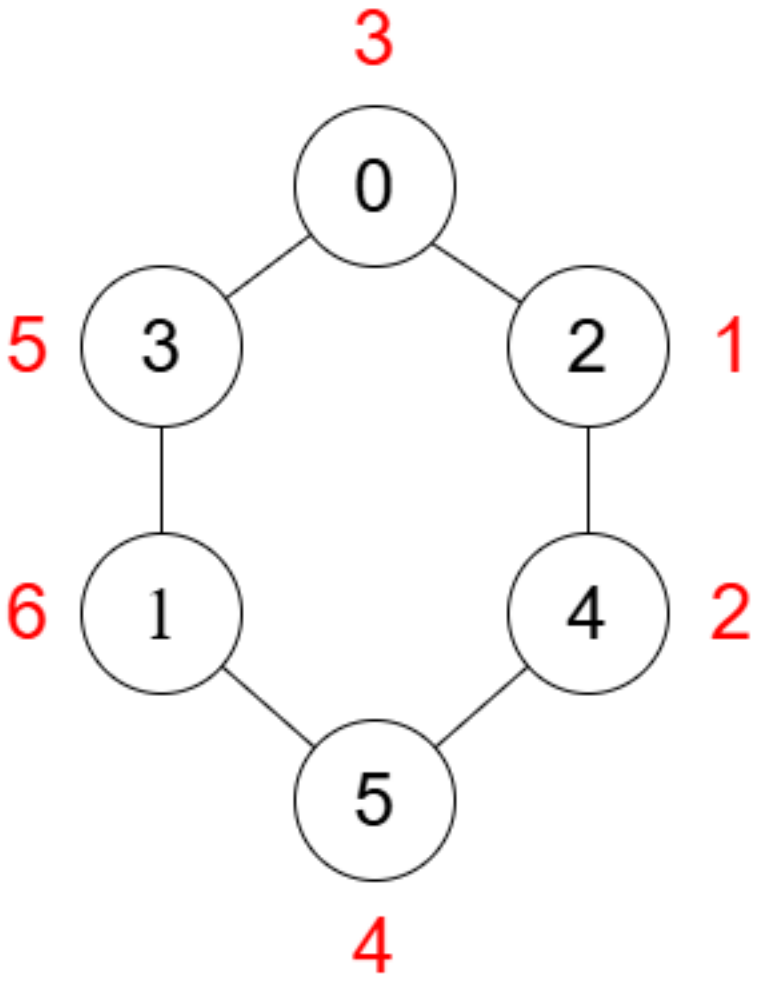
Explanation:

The diagram above illustrates an optimal assignment of values to nodes. The sum of the values of the edges is:

(1 * 3) + (3 * 4) + (4 * 2) = 23

.

Example 2:



Input:

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

Output:

82

Explanation:

The diagram above illustrates an optimal assignment of values to nodes. The sum of the values of the edges is:

(1 * 2) + (2 * 4) + (4 * 6) + (6 * 5) + (5 * 3) + (3 * 1) = 82

.

Constraints:

1 <= n <= 5 * 10

4

m == edges.length

1 <= m <= n

edges[i].length == 2

0 <= a

i

, b

i

< n

a

i

!= b

i

There are no repeated edges.

The graph is connected.

Each node is connected to at most 2 other nodes.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

```
}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Sum of Edge Values in a Graph
* Difficulty: Hard
```

```
* Tags: array, graph, greedy, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Maximum Sum of Edge Values in a Graph
* Difficulty: Hard
* Tags: array, graph, greedy, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Sum of Edge Values in a Graph
Difficulty: Hard
Tags: array, graph, greedy, math

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
```

```
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Sum of Edge Values in a Graph
* Difficulty: Hard
* Tags: array, graph, greedy, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Sum of Edge Values in a Graph
* Difficulty: Hard
* Tags: array, graph, greedy, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

C# Solution:

```
/*
* Problem: Maximum Sum of Edge Values in a Graph
* Difficulty: Hard
* Tags: array, graph, greedy, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

C Solution:

```
/*
* Problem: Maximum Sum of Edge Values in a Graph
* Difficulty: Hard
* Tags: array, graph, greedy, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

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

}
```

Go Solution:

```
// Problem: Maximum Sum of Edge Values in a Graph
// Difficulty: Hard
// Tags: array, graph, greedy, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Sum of Edge Values in a Graph
// Difficulty: Hard
// Tags: array, graph, greedy, math
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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