Problem 2242: Maximum Score of a Node Sequence

Problem Information

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

Problem Description

There is an

undirected

graph with

n

nodes, numbered from

0

to

n - 1

.

You are given a

0-indexed

integer array

scores

of length

n

where

scores[i]

denotes the score of node

i

. You are also given a 2D integer array

edges

where

edges[i] = [a

i

, b

i

]

denotes that there exists an

undirected

edge connecting nodes

a

i

and

b

i

.

A node sequence is

valid

if it meets the following conditions:

There is an edge connecting every pair of

adjacent

nodes in the sequence.

No node appears more than once in the sequence.

The score of a node sequence is defined as the

sum

of the scores of the nodes in the sequence.

Return

the

maximum score

of a valid node sequence with a length of

4

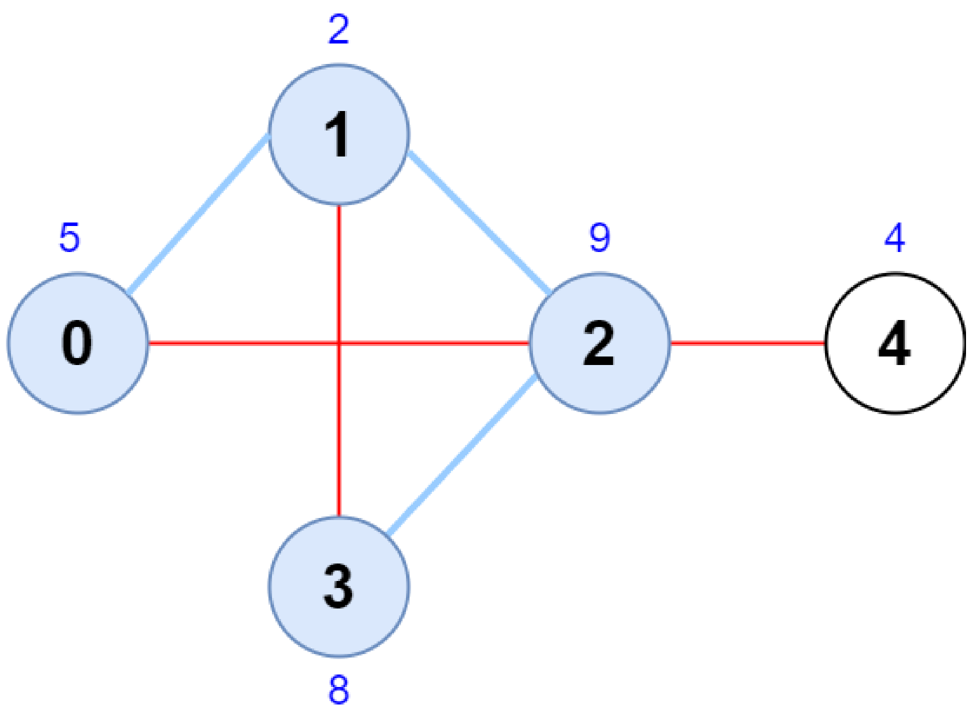
.

If no such sequence exists, return

-1

.

Example 1:



Input:

scores = [5,2,9,8,4], edges = [[0,1],[1,2],[2,3],[0,2],[1,3],[2,4]]

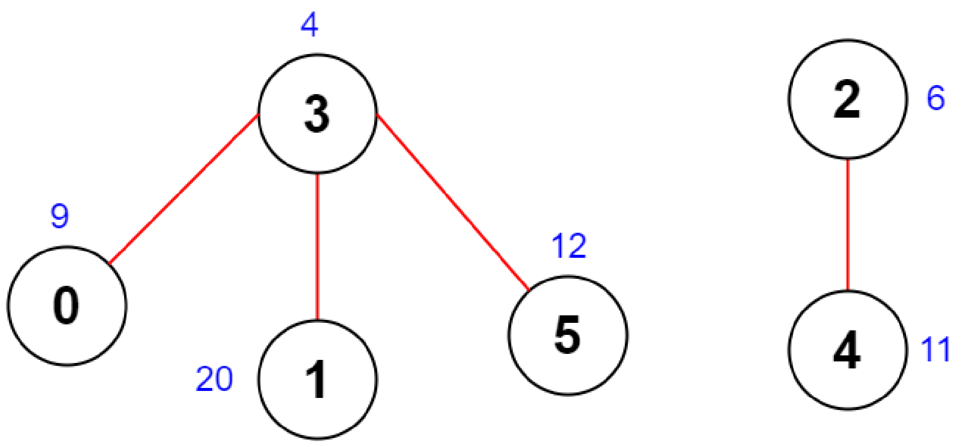
Output:

24

Explanation:

The figure above shows the graph and the chosen node sequence [0,1,2,3]. The score of the node sequence is 5 + 2 + 9 + 8 = 24. It can be shown that no other node sequence has a score of more than 24. Note that the sequences [3,1,2,0] and [1,0,2,3] are also valid and have a score of 24. The sequence [0,3,2,4] is not valid since no edge connects nodes 0 and 3.

Example 2:



Input:

scores = [9,20,6,4,11,12], edges = [[0,3],[5,3],[2,4],[1,3]]

Output:

-1

Explanation:

The figure above shows the graph. There are no valid node sequences of length 4, so we return -1.

Constraints:

n == scores.length

4 <= n <= 5 * 10

4

1 <= scores[i] <= 10

8

0 <= edges.length <= 5 * 10

4

edges[i].length == 2

0 <= a

i

, b

i

<= n - 1

a

i

!= b

i

There are no duplicate edges.

Code Snippets

C++:

```
class Solution {
public:
int maximumScore(vector<int>& scores, vector<vector<int>>& edges) {

}
};
```

Java:

```
class Solution {
public int maximumScore(int[] scores, int[][] edges) {
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximumScore(scores: number[], edges: number[][]): number {

};
```

C#:

```
public class Solution {
public int MaximumScore(int[] scores, int[][] edges) {

}
}
```

C:

```
int maximumScore(int* scores, int scoresSize, int** edges, int edgesSize,
int* edgesColSize) {

}
```

Go:

```
func maximumScore(scores []int, edges [][]int) int {

}
```

Kotlin:

```
class Solution {
fun maximumScore(scores: IntArray, edges: Array<IntArray>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn maximum_score(scores: Vec<i32>, edges: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} scores
# @param {Integer[][]} edges
# @return {Integer}
def maximum_score(scores, edges)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $scores
* @param Integer[][] $edges
* @return Integer
*/
function maximumScore($scores, $edges) {

}
}
```

Dart:

```
class Solution {
int maximumScore(List<int> scores, List<List<int>> edges) {

}
}
```

Scala:

```
object Solution {
def maximumScore(scores: Array[Int], edges: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_score(scores :: [integer], edges :: [[integer]]) :: integer
def maximum_score(scores, edges) do
```

```
end
end
```

Erlang:

```
-spec maximum_score(Scores :: [integer()], Edges :: [[integer()]]) ->
integer().
maximum_score(Scores, Edges) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Score of a Node Sequence
* Difficulty: Hard
* Tags: array, graph, sort
*
* 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:
int maximumScore(vector<int>& scores, vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Score of a Node Sequence
* Difficulty: Hard
* Tags: array, graph, sort
*
* 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 int maximumScore(int[] scores, int[][] edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Score of a Node Sequence
Difficulty: Hard
Tags: array, graph, sort

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 maximumScore(self, scores: List[int], edges: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Score of a Node Sequence
* Difficulty: Hard
* Tags: array, graph, sort
*
* 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[]} scores
* @param {number[][]} edges
* @return {number}
*/
var maximumScore = function(scores, edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Score of a Node Sequence
* Difficulty: Hard
* Tags: array, graph, sort
*
* 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 maximumScore(scores: number[], edges: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Score of a Node Sequence
* Difficulty: Hard
* Tags: array, graph, sort
*
* 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 int MaximumScore(int[] scores, int[][] edges) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Score of a Node Sequence
* Difficulty: Hard
* Tags: array, graph, sort
*
* 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
*/

int maximumScore(int* scores, int scoresSize, int** edges, int edgesSize,
int* edgesColSize) {

}
```

Go Solution:

```
// Problem: Maximum Score of a Node Sequence
// Difficulty: Hard
// Tags: array, graph, sort
//
// 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 maximumScore(scores []int, edges [][]int) int {
}
```

Kotlin Solution:

```
class Solution {
fun maximumScore(scores: IntArray, edges: Array<IntArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Score of a Node Sequence
// Difficulty: Hard
// Tags: array, graph, sort
//
// 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 maximum_score(scores: Vec<i32>, edges: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} scores
# @param {Integer[][]} edges
# @return {Integer}
def maximum_score(scores, edges)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $scores
* @param Integer[][] $edges
* @return Integer
*/
function maximumScore($scores, $edges) {

}
}
```

Dart Solution:

```
class Solution {
int maximumScore(List<int> scores, List<List<int>> edges) {

}
}
```

Scala Solution:

```
object Solution {
def maximumScore(scores: Array[Int], edges: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_score(scores :: [integer], edges :: [[integer]]) :: integer
def maximum_score(scores, edges) do

end
end
```

Erlang Solution:

```
-spec maximum_score(Scores :: [integer()], Edges :: [[integer()]]) ->
integer().
maximum_score(Scores, Edges) ->
.
```

Racket Solution:

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