Problem 3600: Maximize Spanning Tree Stability with Upgrades

Problem Information

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

Problem Description

You are given an integer

n

, representing

n

nodes numbered from 0 to

n - 1

and a list of

edges

, where

edges[i] = [u

i

, v

i

, s

i

, must

i

]

:

u

i

and

v

i

indicates an undirected edge between nodes

u

i

and

v

i

.

s

i

is the strength of the edge.

must

i

is an integer (0 or 1). If

must

i

== 1

, the edge

must

be included in the

spanning tree

. These edges

cannot

be

upgraded

.

You are also given an integer

k

, the

maximum

number of upgrades you can perform. Each upgrade

doubles

the strength of an edge, and each eligible edge (with

must

i

== 0

) can be upgraded

at most

once.

The

stability

of a spanning tree is defined as the

minimum

strength score among all edges included in it.

Return the

maximum

possible stability of any valid spanning tree. If it is impossible to connect all nodes, return

-1

.

Note

: A

spanning tree

of a graph with

n

nodes is a subset of the edges that connects all nodes together (i.e. the graph is

connected

)

without

forming any cycles, and uses

exactly

n - 1

edges.

Example 1:

Input:

n = 3, edges = [[0,1,2,1],[1,2,3,0]], k = 1

Output:

2

Explanation:

Edge

[0,1]

with strength = 2 must be included in the spanning tree.

Edge

[1,2]

is optional and can be upgraded from 3 to 6 using one upgrade.

The resulting spanning tree includes these two edges with strengths 2 and 6.

The minimum strength in the spanning tree is 2, which is the maximum possible stability.

Example 2:

Input:

n = 3, edges = [[0,1,4,0],[1,2,3,0],[0,2,1,0]], k = 2

Output:

6

Explanation:

Since all edges are optional and up to

k = 2

upgrades are allowed.

Upgrade edges

[0,1]

from 4 to 8 and

[1,2]

from 3 to 6.

The resulting spanning tree includes these two edges with strengths 8 and 6.

The minimum strength in the tree is 6, which is the maximum possible stability.

Example 3:

Input:

n = 3, edges = [[0,1,1,1],[1,2,1,1],[2,0,1,1]], k = 0

Output:

-1

Explanation:

All edges are mandatory and form a cycle, which violates the spanning tree property of acyclicity. Thus, the answer is -1.

Constraints:

2 <= n <= 10

5

1 <= edges.length <= 10

5

edges[i] = [u

i

, v

i

, s

i

, must

i

]

0 <= u

i

, v

i

< n

u

i

!= v

i

1 <= s

i

<= 10

5

must

i

is either

0

or

1

.

0 <= k <= n

There are no duplicate edges.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int maxStability(int n, int[][] edges, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int MaxStability(int n, int[][] edges, int k) {

}
}
```

C:

```
int maxStability(int n, int** edges, int edgesSize, int* edgesColSize, int k)
{

}
```

Go:

```
func maxStability(n int, edges [][]int, k int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

```
end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximize Spanning Tree Stability with Upgrades
* Difficulty: Hard
* Tags: tree, graph, greedy, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
int maxStability(int n, vector<vector<int>>& edges, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize Spanning Tree Stability with Upgrades
* Difficulty: Hard
* Tags: tree, graph, greedy, search
```

```
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int maxStability(int n, int[][] edges, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize Spanning Tree Stability with Upgrades
Difficulty: Hard
Tags: tree, graph, greedy, search

Approach: DFS or BFS traversal
Time Complexity: O(n) where n is number of nodes
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```javascript
/**
* Problem: Maximize Spanning Tree Stability with Upgrades
* Difficulty: Hard
* Tags: tree, graph, greedy, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

TypeScript Solution:

```typescript
/**
* Problem: Maximize Spanning Tree Stability with Upgrades
* Difficulty: Hard
* Tags: tree, graph, greedy, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

function maxStability(n: number, edges: number[][], k: number): number {

};
```

C# Solution:

```csharp
/*
* Problem: Maximize Spanning Tree Stability with Upgrades
* Difficulty: Hard
* Tags: tree, graph, greedy, search
```

```
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int MaxStability(int n, int[][] edges, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximize Spanning Tree Stability with Upgrades
* Difficulty: Hard
* Tags: tree, graph, greedy, search
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

int maxStability(int n, int** edges, int edgesSize, int* edgesColSize, int k)
{

}
```

Go Solution:

```
// Problem: Maximize Spanning Tree Stability with Upgrades
// Difficulty: Hard
// Tags: tree, graph, greedy, search
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

func maxStability(n int, edges [][]int, k int) int {
```

```
}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximize Spanning Tree Stability with Upgrades
// Difficulty: Hard
// Tags: tree, graph, greedy, search
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

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

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

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