Problem 1489: Find Critical and Pseudo-Critical Edges in Minimum Spanning Tree

Problem Information

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

Problem Description

Given a weighted undirected connected graph with

n

vertices numbered from

0

to

n - 1

, and an array

edges

where

edges[i] = [a

i

, b

i

, weight

i

]

represents a bidirectional and weighted edge between nodes

a

i

and

b

i

. A minimum spanning tree (MST) is a subset of the graph's edges that connects all vertices without cycles and with the minimum possible total edge weight.

Find

all the critical and pseudo-critical edges in the given graph's minimum spanning tree (MST)

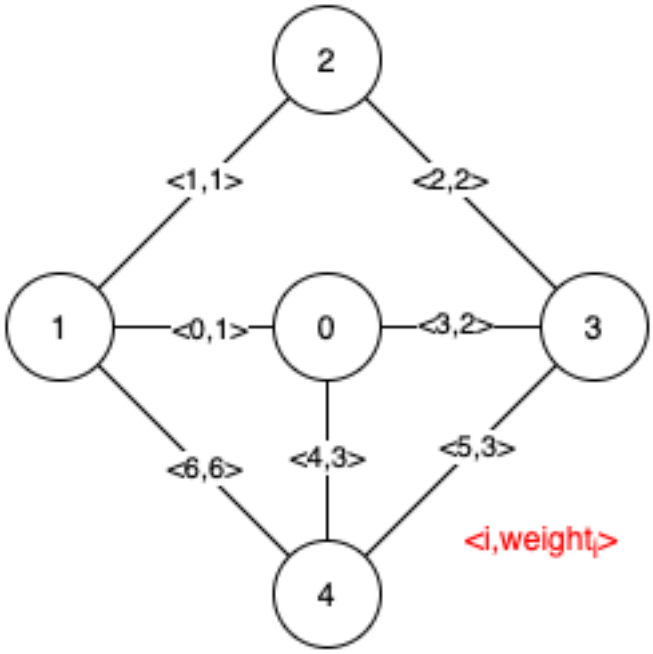
. An MST edge whose deletion from the graph would cause the MST weight to increase is called a

critical edge

. On the other hand, a pseudo-critical edge is that which can appear in some MSTs but not all.

Note that you can return the indices of the edges in any order.

Example 1:



Input:

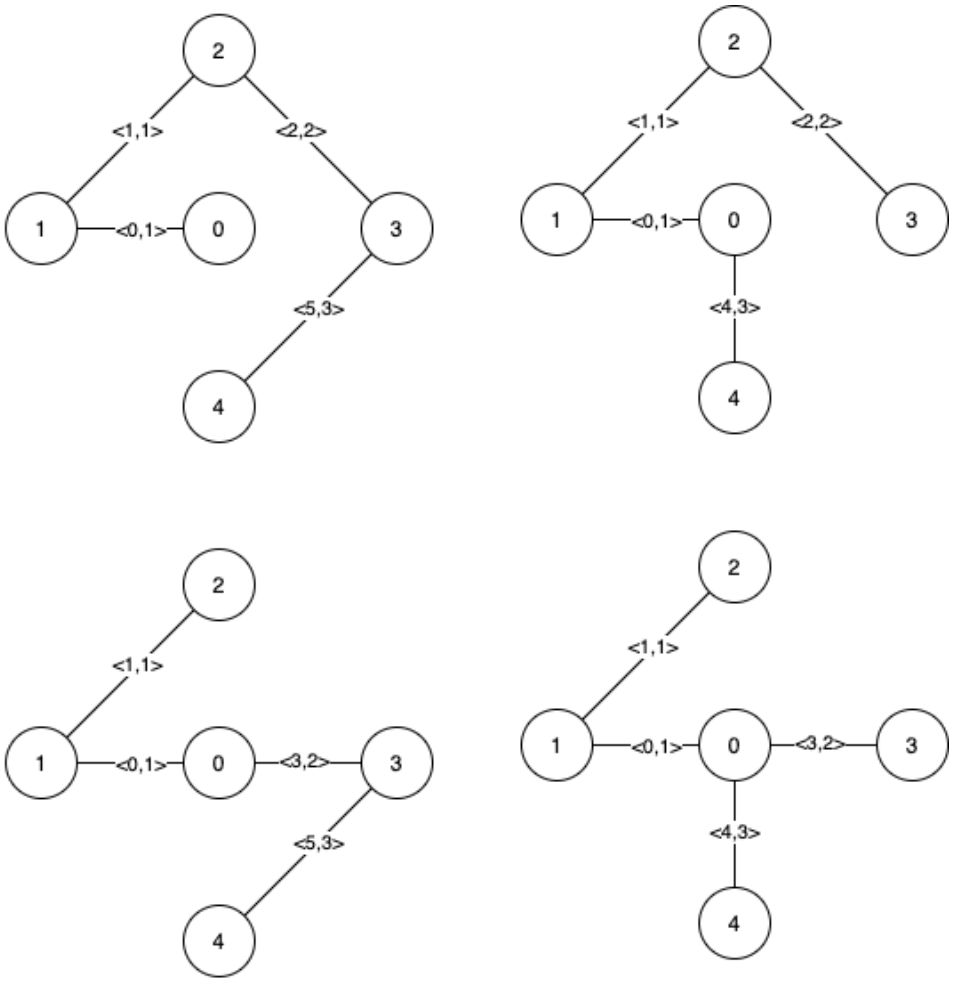
n = 5, edges = [[0,1,1],[1,2,1],[2,3,2],[0,3,2],[0,4,3],[3,4,3],[1,4,6]]

Output:

[[0,1],[2,3,4,5]]

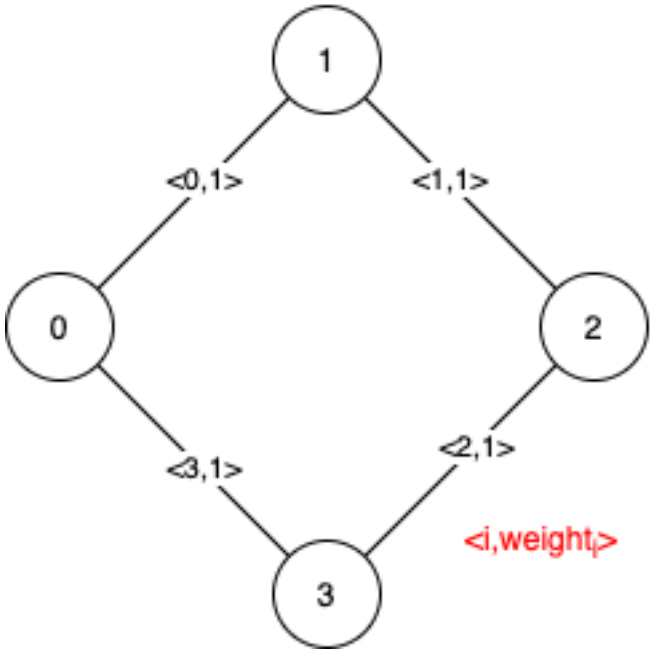
Explanation:

The figure above describes the graph. The following figure shows all the possible MSTs:



Notice that the two edges 0 and 1 appear in all MSTs, therefore they are critical edges, so we return them in the first list of the output. The edges 2, 3, 4, and 5 are only part of some MSTs, therefore they are considered pseudo-critical edges. We add them to the second list of the output.

Example 2:



Input:

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

Output:

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

Explanation:

We can observe that since all 4 edges have equal weight, choosing any 3 edges from the given 4 will yield an MST. Therefore all 4 edges are pseudo-critical.

Constraints:

2 <= n <= 100

1 <= edges.length <= min(200, n * (n - 1) / 2)

edges[i].length == 3

0 <= a

i

< b

i

< n

1 <= weight

i

<= 1000

All pairs

(a

i

, b

i

)

are

distinct

.

Code Snippets

C++:

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

```
}
};
```

Java:

```
class Solution {
public List<List<Integer>> findCriticalAndPseudoCriticalEdges(int n, int[][]
edges) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** findCriticalAndPseudoCriticalEdges(int n, int** edges, int edgesSize,
int* edgesColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func findCriticalAndPseudoCriticalEdges(n int, edges [][]int) [][]int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
List<List<int>> findCriticalAndPseudoCriticalEdges(int n, List<List<int>>
```

```
edges) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (find-critical-and-pseudo-critical-edges n edges)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Critical and Pseudo-Critical Edges in Minimum Spanning Tree
* Difficulty: Hard
* Tags: array, tree, graph, sort
*
* 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:
vector<vector<int>> findCriticalAndPseudoCriticalEdges(int n,
vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Find Critical and Pseudo-Critical Edges in Minimum Spanning Tree
* Difficulty: Hard
* Tags: array, tree, graph, sort
*
* 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 List<List<Integer>> findCriticalAndPseudoCriticalEdges(int n, int[][]
edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Critical and Pseudo-Critical Edges in Minimum Spanning Tree
Difficulty: Hard
```

```
Tags: array, tree, graph, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Critical and Pseudo-Critical Edges in Minimum Spanning Tree
* Difficulty: Hard
* Tags: array, tree, graph, sort
*
* 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
* @return {number[][]}
*/
var findCriticalAndPseudoCriticalEdges = function(n, edges) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Find Critical and Pseudo-Critical Edges in Minimum Spanning Tree
* Difficulty: Hard
* Tags: array, tree, graph, sort
*
* 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 findCriticalAndPseudoCriticalEdges(n: number, edges: number[][]):
number[][] {

};
```

C# Solution:

```
/*
* Problem: Find Critical and Pseudo-Critical Edges in Minimum Spanning Tree
* Difficulty: Hard
* Tags: array, tree, graph, sort
*
* 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 IList<IList<int>> FindCriticalAndPseudoCriticalEdges(int n, int[][]
edges) {

}
}
```

C Solution:

```
/*
* Problem: Find Critical and Pseudo-Critical Edges in Minimum Spanning Tree
* Difficulty: Hard
* Tags: array, tree, graph, sort
*
* 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** findCriticalAndPseudoCriticalEdges(int n, int** edges, int edgesSize,
int* edgesColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Find Critical and Pseudo-Critical Edges in Minimum Spanning Tree
// Difficulty: Hard
// Tags: array, tree, graph, sort
//
// 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 findCriticalAndPseudoCriticalEdges(n int, edges [][]int) [][]int {

}
```

Kotlin Solution:

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

}
```

```
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find Critical and Pseudo-Critical Edges in Minimum Spanning Tree
// Difficulty: Hard
// Tags: array, tree, graph, sort
//
// 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 find_critical_and_pseudo_critical_edges(n: i32, edges: Vec<Vec<i32>>)
-> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-critical-and-pseudo-critical-edges n edges)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)))
)
```