Problem 802: Find Eventual Safe States

Problem Information

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

Problem Description

There is a directed graph of

n

nodes with each node labeled from

0

to

n - 1

. The graph is represented by a

0-indexed

2D integer array

graph

where

graph[i]

is an integer array of nodes adjacent to node

i

, meaning there is an edge from node

i

to each node in

graph[i]

.

A node is a

terminal node

if there are no outgoing edges. A node is a

safe node

if every possible path starting from that node leads to a

terminal node

(or another safe node).

Return

an array containing all the

safe nodes

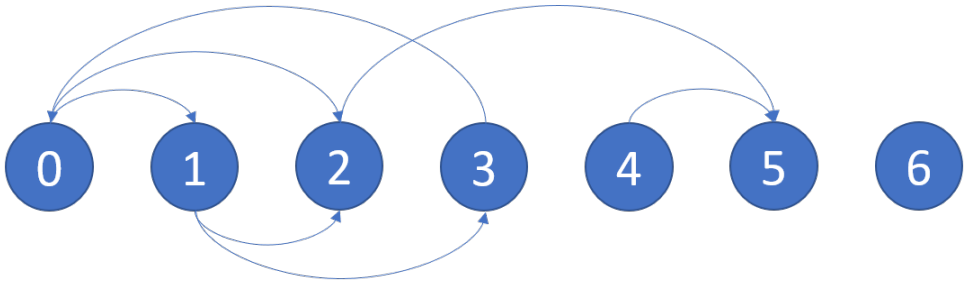
of the graph

. The answer should be sorted in

ascending

order.

Example 1:



Input:

graph = [[1,2],[2,3],[5],[0],[5],[],[]]

Output:

[2,4,5,6]

Explanation:

The given graph is shown above. Nodes 5 and 6 are terminal nodes as there are no outgoing edges from either of them. Every path starting at nodes 2, 4, 5, and 6 all lead to either node 5 or 6.

Example 2:

Input:

graph = [[1,2,3,4],[1,2],[3,4],[0,4],[]]

Output:

[4]

Explanation:

Only node 4 is a terminal node, and every path starting at node 4 leads to node 4.

Constraints:

n == graph.length

1 <= n <= 10

4

0 <= graph[i].length <= n

0 <= graph[i][j] <= n - 1

graph[i]

is sorted in a strictly increasing order.

The graph may contain self-loops.

The number of edges in the graph will be in the range

[1, 4 * 10

4

]

.

Code Snippets

C++:

```
class Solution {
public:
vector<int> eventualSafeNodes(vector<vector<int>>& graph) {

}
};
```

Java:

```
class Solution {
public List<Integer> eventualSafeNodes(int[][] graph) {

}
}
```

Python3:

```
class Solution:
def eventualSafeNodes(self, graph: List[List[int]]) -> List[int]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function eventualSafeNodes(graph: number[][]): number[] {

};
```

C#:

```
public class Solution {
public IList<int> EventualSafeNodes(int[][] graph) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* eventualSafeNodes(int** graph, int graphSize, int* graphColSize, int*
returnSize) {

}
```

Go:

```
func eventualSafeNodes(graph [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun eventualSafeNodes(graph: Array<IntArray>): List<Int> {

}
}
```

Swift:

```
class Solution {
func eventualSafeNodes(_ graph: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn eventual_safe_nodes(graph: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} graph
# @return {Integer[]}
def eventual_safe_nodes(graph)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $graph
* @return Integer[]
*/
function eventualSafeNodes($graph) {

}
}
```

Dart:

```
class Solution {
List<int> eventualSafeNodes(List<List<int>> graph) {

}
}
```

Scala:

```
object Solution {
def eventualSafeNodes(graph: Array[Array[Int]]): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec eventual_safe_nodes(graph :: [[integer]]) :: [integer]
def eventual_safe_nodes(graph) do

end
end
```

Erlang:

```
-spec eventual_safe_nodes(Graph :: [[integer()]]) -> [integer()].
eventual_safe_nodes(Graph) ->
.
```

Racket:

```
(define/contract (eventual-safe-nodes graph)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Eventual Safe States
* Difficulty: Medium
* Tags: array, graph, sort, search
*
* 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:
vector<int> eventualSafeNodes(vector<vector<int>>& graph) {

}
};
```

Java Solution:

```
/**
* Problem: Find Eventual Safe States
* Difficulty: Medium
* Tags: array, graph, sort, search
*
* 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 List<Integer> eventualSafeNodes(int[][] graph) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Eventual Safe States
Difficulty: Medium
Tags: array, graph, sort, search

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 eventualSafeNodes(self, graph: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Eventual Safe States
* Difficulty: Medium
```

```
* Tags: array, graph, sort, search
*
* 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[][]} graph
* @return {number[]}
*/
var eventualSafeNodes = function(graph) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Eventual Safe States
* Difficulty: Medium
* Tags: array, graph, sort, search
*
* 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 eventualSafeNodes(graph: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Find Eventual Safe States
* Difficulty: Medium
* Tags: array, graph, sort, search
*
* 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 IList<int> EventualSafeNodes(int[][] graph) {

}
}
```

C Solution:

```
/*
* Problem: Find Eventual Safe States
* Difficulty: Medium
* Tags: array, graph, sort, search
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* eventualSafeNodes(int** graph, int graphSize, int* graphColSize, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Find Eventual Safe States
// Difficulty: Medium
// Tags: array, graph, sort, search
//
// 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 eventualSafeNodes(graph [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun eventualSafeNodes(graph: Array<IntArray>): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func eventualSafeNodes(_ graph: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Find Eventual Safe States
// Difficulty: Medium
// Tags: array, graph, sort, search
//
// 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 eventual_safe_nodes(graph: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} graph
# @return {Integer[]}
def eventual_safe_nodes(graph)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $graph
* @return Integer[]
*/
function eventualSafeNodes($graph) {

}
}
```

Dart Solution:

```
class Solution {
List<int> eventualSafeNodes(List<List<int>> graph) {

}
}
```

Scala Solution:

```
object Solution {
def eventualSafeNodes(graph: Array[Array[Int]]): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec eventual_safe_nodes(graph :: [[integer]]) :: [integer]
def eventual_safe_nodes(graph) do

end
end
```

Erlang Solution:

```
-spec eventual_safe_nodes(Graph :: [[integer()]]) -> [integer()].
eventual_safe_nodes(Graph) ->
.
```

Racket Solution:

```
(define/contract (eventual-safe-nodes graph)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```