Problem 1557: Minimum Number of Vertices to Reach All Nodes

Problem Information

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

Problem Description

Given a

directed acyclic graph

, with

n

vertices numbered from

0

to

n-1

, and an array

edges

where

edges[i] = [from

i

, to

i

]

represents a directed edge from node

from

i

to node

to

i

.

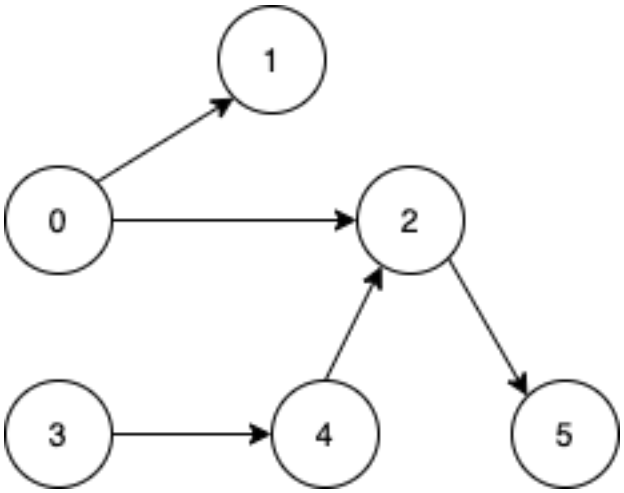
Find

the smallest set of vertices from which all nodes in the graph are reachable

. It's guaranteed that a unique solution exists.

Notice that you can return the vertices in any order.

Example 1:



Input:

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

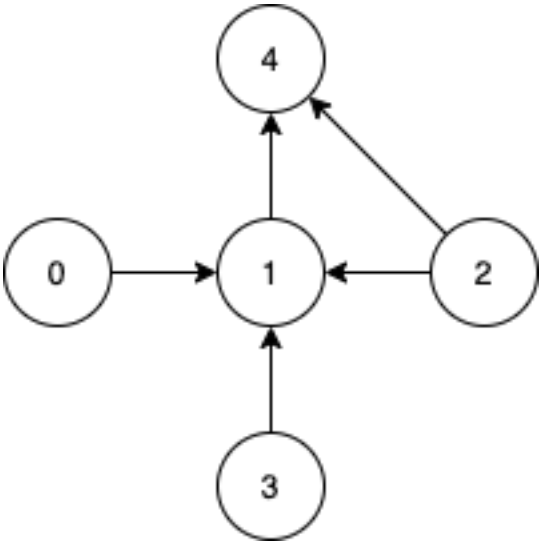
Output:

[0,3]

Explanation:

It's not possible to reach all the nodes from a single vertex. From 0 we can reach [0,1,2,5]. From 3 we can reach [3,4,2,5]. So we output [0,3].

Example 2:



Input:

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

Output:

[0,2,3]

Explanation:

Notice that vertices 0, 3 and 2 are not reachable from any other node, so we must include them. Also any of these vertices can reach nodes 1 and 4.

Constraints:

2 <= n <= 10^5

1 <= edges.length <= min(10^5, n * (n - 1) / 2)

edges[i].length == 2

0 <= from

i,

to

i

< n

All pairs

(from

i

, to

i

)

are distinct.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<Integer> findSmallestSetOfVertices(int n, List<List<Integer>>
edges) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public IList<int> FindSmallestSetOfVertices(int n, IList<IList<int>> edges) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findSmallestSetOfVertices(int n, int** edges, int edgesSize, int*
edgesColSize, int* returnSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun findSmallestSetOfVertices(n: Int, edges: List<List<Int>>): List<Int> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

```
}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (find-smallest-set-of-vertices n edges)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Vertices to Reach All Nodes
* Difficulty: Medium
* Tags: array, graph
*
* 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> findSmallestSetOfVertices(int n, vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Vertices to Reach All Nodes
* Difficulty: Medium
* Tags: array, graph
*
* 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> findSmallestSetOfVertices(int n, List<List<Integer>>
edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Vertices to Reach All Nodes
Difficulty: Medium
Tags: array, graph

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Number of Vertices to Reach All Nodes
* Difficulty: Medium
* Tags: array, graph
*
* 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 findSmallestSetOfVertices = function(n, edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Vertices to Reach All Nodes
* Difficulty: Medium
* Tags: array, graph
*
* 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 findSmallestSetOfVertices(n: number, edges: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Vertices to Reach All Nodes
* Difficulty: Medium
* Tags: array, graph
*
* 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> FindSmallestSetOfVertices(int n, IList<IList<int>> edges) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Vertices to Reach All Nodes
* Difficulty: Medium
* Tags: array, graph
*
* 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* findSmallestSetOfVertices(int n, int** edges, int edgesSize, int*
edgesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Minimum Number of Vertices to Reach All Nodes
// Difficulty: Medium
// Tags: array, graph
//
// 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 findSmallestSetOfVertices(n int, edges [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun findSmallestSetOfVertices(n: Int, edges: List<List<Int>>): List<Int> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Vertices to Reach All Nodes
// Difficulty: Medium
// Tags: array, graph
//
// 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 find_smallest_set_of_vertices(n: i32, edges: Vec<Vec<i32>>) ->
Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

```
}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-smallest-set-of-vertices n edges)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
```

```
)
```