Problem 3310: Remove Methods From Project

Problem Information

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

Problem Description

You are maintaining a project that has

n

methods numbered from

0

to

n - 1

.

You are given two integers

n

and

k

, and a 2D integer array

invocations

, where

invocations[i] = [a

i

, b

i

]

indicates that method

a

i

invokes method

b

i

.

There is a known bug in method

k

. Method

k

, along with any method invoked by it, either

directly

or

indirectly

, are considered

suspicious

and we aim to remove them.

A group of methods can only be removed if no method

outside

the group invokes any methods

within

it.

Return an array containing all the remaining methods after removing all the

suspicious

methods. You may return the answer in

any order

. If it is not possible to remove

all

the suspicious methods,

none

should be removed.

Example 1:

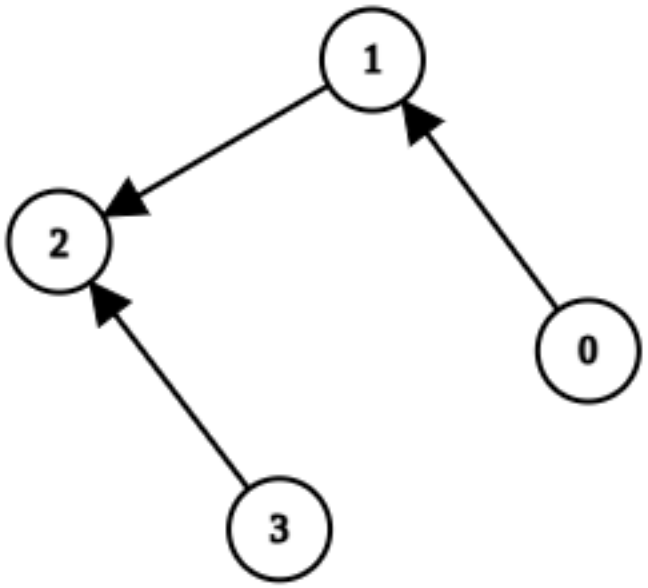
Input:

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

Output:

[0,1,2,3]

Explanation:



Method 2 and method 1 are suspicious, but they are directly invoked by methods 3 and 0, which are not suspicious. We return all elements without removing anything.

Example 2:

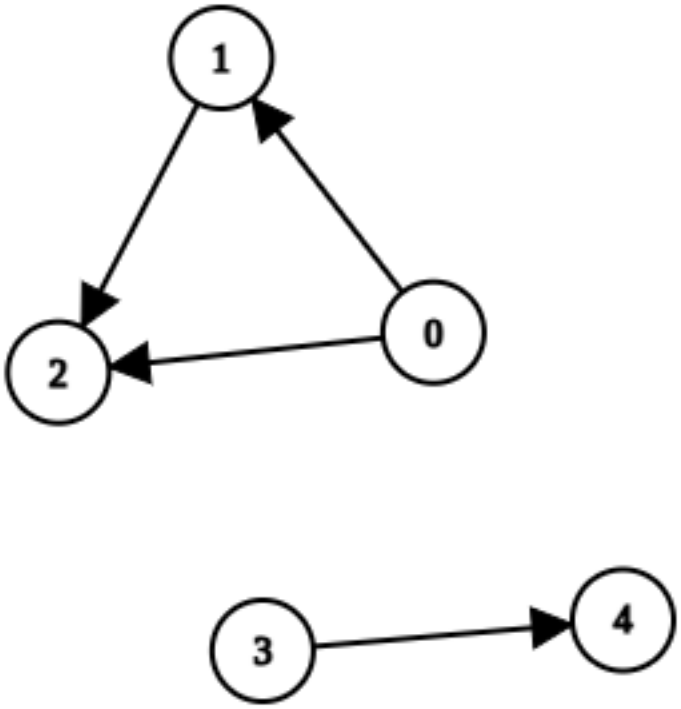
Input:

n = 5, k = 0, invocations = [[1,2],[0,2],[0,1],[3,4]]

Output:

[3,4]

Explanation:



Methods 0, 1, and 2 are suspicious and they are not directly invoked by any other method. We can remove them.

Example 3:

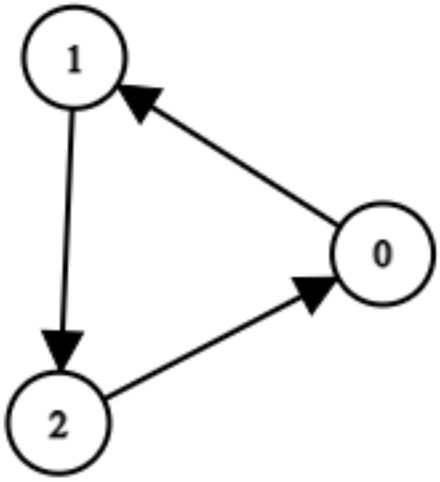
Input:

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

Output:

[]

Explanation:



All methods are suspicious. We can remove them.

Constraints:

1 <= n <= 10

5

0 <= k <= n - 1

0 <= invocations.length <= 2 * 10

5

invocations[i] == [a

i

, b

i

]

0 <= a

i

, b

i

<= n - 1

a

i

!= b

i

invocations[i] != invocations[j]

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<Integer> remainingMethods(int n, int k, int[][] invocations) {
```

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function remainingMethods(n: number, k: number, invocations: number[][]):
number[] {

};
```

C#:

```
public class Solution {
public IList<int> RemainingMethods(int n, int k, int[][] invocations) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* remainingMethods(int n, int k, int** invocations, int invocationsSize,
int* invocationsColSize, int* returnSize) {

}
```

Go:

```
func remainingMethods(n int, k int, invocations [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun remainingMethods(n: Int, k: Int, invocations: Array<IntArray>): List<Int>
{

}
}
```

Swift:

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

}
}
```

Rust:

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

```
Vec<i32> {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
List<int> remainingMethods(int n, int k, List<List<int>> invocations) {

}
}
```

Scala:

```
object Solution {
def remainingMethods(n: Int, k: Int, invocations: Array[Array[Int]]):
List[Int] = {
```

```
}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec remaining_methods(N :: integer(), K :: integer(), Invocations ::
[[integer()]]) -> [integer()].
remaining_methods(N, K, Invocations) ->
.
```

Racket:

```
(define/contract (remaining-methods n k invocations)
(-> exact-integer? exact-integer? (listof (listof exact-integer?)) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Remove Methods From Project
* Difficulty: Medium
* Tags: array, graph, 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> remainingMethods(int n, int k, vector<vector<int>>& invocations)
{

}
};
```

Java Solution:

```
/**
* Problem: Remove Methods From Project
* Difficulty: Medium
* Tags: array, graph, 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> remainingMethods(int n, int k, int[][] invocations) {

}
}
```

Python3 Solution:

```
"""
Problem: Remove Methods From Project
Difficulty: Medium
Tags: array, graph, 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 remainingMethods(self, n: int, k: int, invocations: List[List[int]]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Remove Methods From Project
* Difficulty: Medium
* Tags: array, graph, 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} n
* @param {number} k
* @param {number[][]} invocations
* @return {number[]}
*/
var remainingMethods = function(n, k, invocations) {

};
```

TypeScript Solution:

```
/**
* Problem: Remove Methods From Project
* Difficulty: Medium
* Tags: array, graph, 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 remainingMethods(n: number, k: number, invocations: number[][]):
number[] {

};
```

C# Solution:

```
/*
* Problem: Remove Methods From Project
* Difficulty: Medium
* Tags: array, graph, 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> RemainingMethods(int n, int k, int[][] invocations) {

}
}
```

C Solution:

```
/*
* Problem: Remove Methods From Project
* Difficulty: Medium
* Tags: array, graph, 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* remainingMethods(int n, int k, int** invocations, int invocationsSize,
int* invocationsColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Remove Methods From Project
// Difficulty: Medium
// Tags: array, graph, 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 remainingMethods(n int, k int, invocations [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun remainingMethods(n: Int, k: Int, invocations: Array<IntArray>): List<Int>
{

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Remove Methods From Project
// Difficulty: Medium
// Tags: array, graph, 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 remaining_methods(n: i32, k: i32, invocations: Vec<Vec<i32>>) ->
Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
List<int> remainingMethods(int n, int k, List<List<int>> invocations) {

}
}
```

Scala Solution:

```
object Solution {
def remainingMethods(n: Int, k: Int, invocations: Array[Array[Int]]):
List[Int] = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec remaining_methods(N :: integer(), K :: integer(), Invocations ::
[[integer()]]) -> [integer()].
remaining_methods(N, K, Invocations) ->
.
```

Racket Solution:

```
(define/contract (remaining-methods n k invocations)
(-> exact-integer? exact-integer? (listof (listof exact-integer?)) (listof
exact-integer?))
)
```