Problem 3493: Properties Graph

Problem Information

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

Problem Description

You are given a 2D integer array

properties

having dimensions

n x m

and an integer

k

.

Define a function

intersect(a, b)

that returns the

number of distinct integers

common to both arrays

a

and

b

.

Construct an

undirected

graph where each index

i

corresponds to

properties[i]

. There is an edge between node

i

and node

j

if and only if

intersect(properties[i], properties[j]) >= k

, where

i

and

j

are in the range

[0, n - 1]

and

i != j

.

Return the number of

connected components

in the resulting graph.

Example 1:

Input:

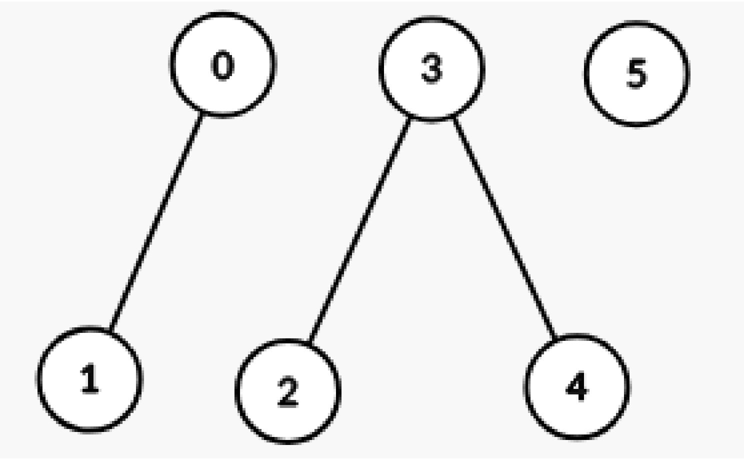
properties = [[1,2],[1,1],[3,4],[4,5],[5,6],[7,7]], k = 1

Output:

3

Explanation:

The graph formed has 3 connected components:



Example 2:

Input:

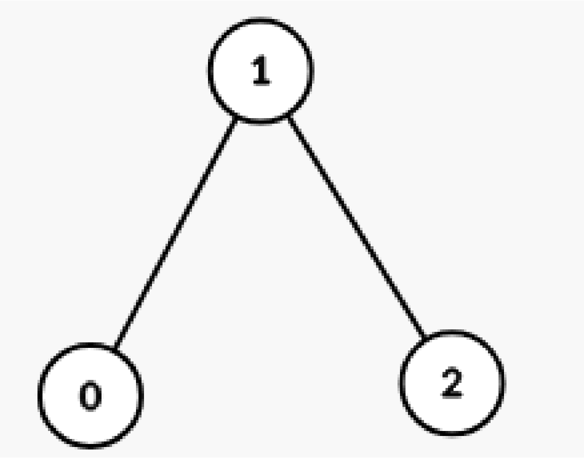
properties = [[1,2,3],[2,3,4],[4,3,5]], k = 2

Output:

1

Explanation:

The graph formed has 1 connected component:



Example 3:

Input:

properties = [[1,1],[1,1]], k = 2

Output:

2

Explanation:

intersect(properties[0], properties[1]) = 1

, which is less than

k

. This means there is no edge between

properties[0]

and

properties[1]

in the graph.

Constraints:

1 <= n == properties.length <= 100

1 <= m == properties[i].length <= 100

1 <= properties[i][j] <= 100

1 <= k <= m

Code Snippets

C++:

```
class Solution {
public:
int numberOfComponents(vector<vector<int>>& properties, int k) {

}
};
```

Java:

```
class Solution {
public int numberOfComponents(int[][] properties, int k) {
}
}
```

Python3:

```
class Solution:
def numberOfComponents(self, properties: List[List[int]], k: int) -> int:
```

Python:

```
class Solution(object):
def numberOfComponents(self, properties, k):
"""
:type properties: List[List[int]]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} properties
* @param {number} k
* @return {number}
*/
var numberOfComponents = function(properties, k) {

};
```

TypeScript:

```
function numberOfComponents(properties: number[][], k: number): number {

};
```

C#:

```
public class Solution {
public int NumberOfComponents(int[][] properties, int k) {

}

}
```

C:

```
int numberOfComponents(int** properties, int propertiesSize, int*
propertiesColSize, int k) {

}
```

Go:

```
func numberOfComponents(properties [][]int, k int) int {

}
```

Kotlin:

```
class Solution {
fun numberOfComponents(properties: Array<IntArray>, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func numberOfComponents(_ properties: [[Int]], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn number_of_components(properties: Vec<Vec<i32>>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} properties
# @param {Integer} k
# @return {Integer}
def number_of_components(properties, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $properties
* @param Integer $k
* @return Integer
*/
function numberOfComponents($properties, $k) {

}
}
```

Dart:

```
class Solution {
int numberOfComponents(List<List<int>> properties, int k) {

}
}
```

Scala:

```
object Solution {
def numberOfComponents(properties: Array[Array[Int]], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec number_of_components(properties :: [[integer]], k :: integer) :: integer
def number_of_components(properties, k) do
```

```
end
end
```

Erlang:

```
-spec number_of_components(Properties :: [[integer()]], K :: integer()) ->
integer().
number_of_components(Properties, K) ->
.
```

Racket:

```
(define/contract (number-of-components properties k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Properties Graph
* Difficulty: Medium
* Tags: array, graph, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int numberOfComponents(vector<vector<int>>& properties, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Properties Graph
* Difficulty: Medium
* Tags: array, graph, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int numberOfComponents(int[][] properties, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Properties Graph
Difficulty: Medium
Tags: array, graph, hash, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

```
class Solution(object):
def numberOfComponents(self, properties, k):
"""
:type properties: List[List[int]]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```javascript
/**
* Problem: Properties Graph
* Difficulty: Medium
* Tags: array, graph, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[][]} properties
* @param {number} k
* @return {number}
*/
var numberOfComponents = function(properties, k) {

};
```

TypeScript Solution:

```typescript
/**
* Problem: Properties Graph
* Difficulty: Medium
* Tags: array, graph, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function numberOfComponents(properties: number[][], k: number): number {

};
```

C# Solution:

```csharp
/*
* Problem: Properties Graph
* Difficulty: Medium
* Tags: array, graph, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int NumberOfComponents(int[][] properties, int k) {

}
}
```

C Solution:

```
/*
* Problem: Properties Graph
* Difficulty: Medium
* Tags: array, graph, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int numberOfComponents(int** properties, int propertiesSize, int*
propertiesColSize, int k) {

}
```

Go Solution:

```
// Problem: Properties Graph
// Difficulty: Medium
// Tags: array, graph, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func numberOfComponents(properties [][]int, k int) int {
}
```

Kotlin Solution:

```
class Solution {
fun numberOfComponents(properties: Array<IntArray>, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func numberOfComponents(_ properties: [[Int]], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Properties Graph
// Difficulty: Medium
// Tags: array, graph, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn number_of_components(properties: Vec<Vec<i32>>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} properties
# @param {Integer} k
# @return {Integer}
def number_of_components(properties, k)
```

```


end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $properties
* @param Integer $k
* @return Integer
*/
function numberOfComponents($properties, $k) {

}
}
```

Dart Solution:

```
class Solution {
int numberOfComponents(List<List<int>> properties, int k) {

}
}
```

Scala Solution:

```
object Solution {
def numberOfComponents(properties: Array[Array[Int]], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec number_of_components(properties :: [[integer]], k :: integer) ::
integer
def number_of_components(properties, k) do

end
```

```
end
```

Erlang Solution:

```
-spec number_of_components(Properties :: [[integer()]], K :: integer()) ->
integer().
number_of_components(Properties, K) ->
.
```

Racket Solution:

```
(define/contract (number-of-components properties k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```