Problem 547: Number of Provinces

Problem Information

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

Problem Description

There are

n

cities. Some of them are connected, while some are not. If city

a

is connected directly with city

b

, and city

b

is connected directly with city

c

, then city

a

is connected indirectly with city

c

.

A

province

is a group of directly or indirectly connected cities and no other cities outside of the group.

You are given an

n x n

matrix

isConnected

where

isConnected[i][j] = 1

if the

i

th

city and the

j

th

city are directly connected, and

isConnected[i][j] = 0

otherwise.

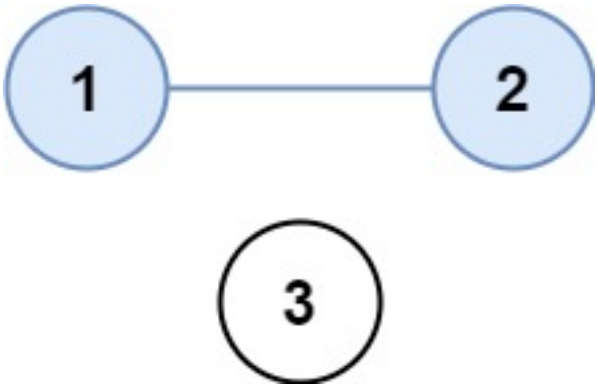
Return

the total number of

provinces

.

Example 1:



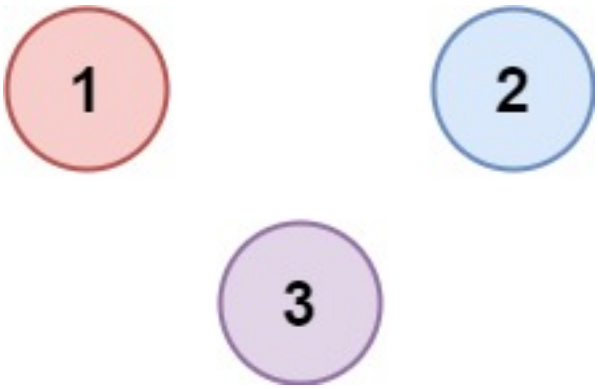
Input:

isConnected = [[1,1,0],[1,1,0],[0,0,1]]

Output:

2

Example 2:



Input:

isConnected = [[1,0,0],[0,1,0],[0,0,1]]

Output:

3

Constraints:

1 <= n <= 200

n == isConnected.length

n == isConnected[i].length

isConnected[i][j]

is

1

or

0

.

isConnected[i][i] == 1

isConnected[i][j] == isConnected[j][i]

Code Snippets

C++:

```
class Solution {
public:
int findCircleNum(vector<vector<int>>& isConnected) {
```

```
}
};
```

Java:

```
class Solution {
public int findCircleNum(int[][] isConnected) {

}
}
```

Python3:

```
class Solution:
def findCircleNum(self, isConnected: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function findCircleNum(isConnected: number[][]): number {

};
```

C#:

```
public class Solution {
public int FindCircleNum(int[][] isConnected) {

}
}
```

C:

```
int findCircleNum(int** isConnected, int isConnectedSize, int*
isConnectedColSize) {

}
```

Go:

```
func findCircleNum(isConnected [][]int) int {

}
```

Kotlin:

```
class Solution {
fun findCircleNum(isConnected: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func findCircleNum(_ isConnected: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_circle_num(is_connected: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} is_connected
# @return {Integer}
def find_circle_num(is_connected)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $isConnected
* @return Integer
*/
function findCircleNum($isConnected) {

}
}
```

Dart:

```
class Solution {
int findCircleNum(List<List<int>> isConnected) {

}
}
```

Scala:

```
object Solution {
def findCircleNum(isConnected: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_circle_num(is_connected :: [[integer]]) :: integer
def find_circle_num(is_connected) do
```

```
end
end
```

Erlang:

```
-spec find_circle_num(IsConnected :: [[integer()]]) -> integer().
find_circle_num(IsConnected) ->
.
```

Racket:

```
(define/contract (find-circle-num isConnected)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Provinces
* Difficulty: Medium
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int findCircleNum(vector<vector<int>>& isConnected) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Provinces
```

```
* Difficulty: Medium
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int findCircleNum(int[][] isConnected) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Provinces
Difficulty: Medium
Tags: graph, search

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Provinces
* Difficulty: Medium
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[][]} isConnected
* @return {number}
*/
var findCircleNum = function(isConnected) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Provinces
* Difficulty: Medium
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function findCircleNum(isConnected: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Provinces
* Difficulty: Medium
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
```

```
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int FindCircleNum(int[][] isConnected) {

}
}
```

C Solution:

```
/*
* Problem: Number of Provinces
* Difficulty: Medium
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int findCircleNum(int** isConnected, int isConnectedSize, int*
isConnectedColSize) {

}
```

Go Solution:

```
// Problem: Number of Provinces
// Difficulty: Medium
// Tags: graph, search
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func findCircleNum(isConnected [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findCircleNum(isConnected: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func findCircleNum(_ isConnected: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Provinces
// Difficulty: Medium
// Tags: graph, search
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn find_circle_num(is_connected: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} is_connected
# @return {Integer}
def find_circle_num(is_connected)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $isConnected
* @return Integer
*/
function findCircleNum($isConnected) {

}
}
```

Dart Solution:

```
class Solution {
int findCircleNum(List<List<int>> isConnected) {

}
}
```

Scala Solution:

```
object Solution {
def findCircleNum(isConnected: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_circle_num(is_connected :: [[integer]]) :: integer
def find_circle_num(is_connected) do

end
end
```

Erlang Solution:

```
-spec find_circle_num(IsConnected :: [[integer()]]) -> integer().
find_circle_num(IsConnected) ->
.
```

Racket Solution:

```
(define/contract (find-circle-num isConnected)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```