Ayudantía 10

Grafos y DFS

Carlos Paredes - cparedesr@uc.cl Sergio Matamala Appel - matamalaappels@uc.cl

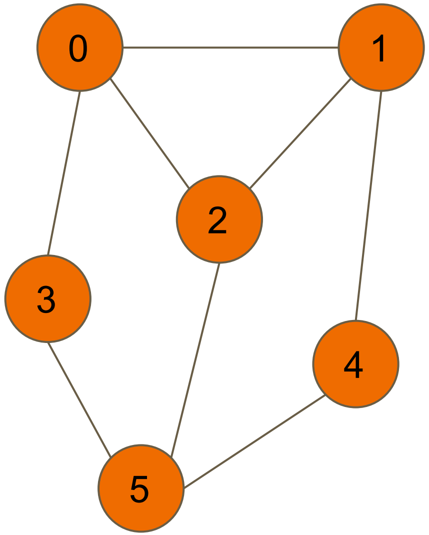
DFS (Iterative)

```
dfs(graph G, node start)
    stack s
    s.push(start)
    label start as discovered
    while not s.empty()
        node u = s.pop()
        for v in G.adjacent[u]
            if v is not discovered
                s.push(v)
                label v as discovered
```

DFS (Recursive)

```
dfs(graph G, node u)
    label u as discovered
    for v in G.adjacent[u]
        if v not discovered
            dfs(G, v)
```

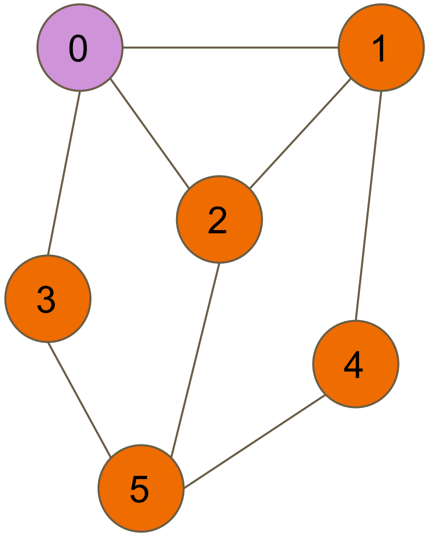
Ejemplo




```
dfs(G, 0)

dfs(graph G, node u)
    label u as discovered
    for v in G.adjacent[u]
        if v not discovered
            dfs(G, v)
```

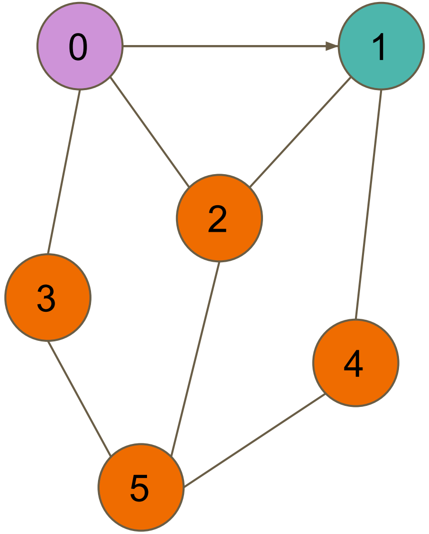
Ejemplo




```
dfs(G, 0)

dfs(graph G, node u)
    label u as discovered
    for v in G.adjacent[u]
        if v not discovered
            dfs(G, v)
```

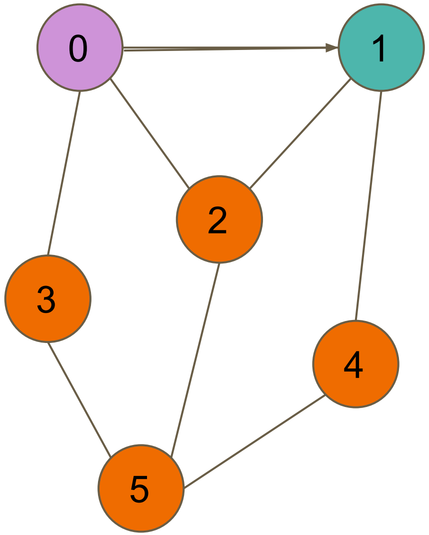
Ejemplo




```
dfs(G, 0)

dfs(graph G, node u)
    label u as discovered
    for v in G.adjacent[u]
        if v not discovered
            dfs(G, v)
```

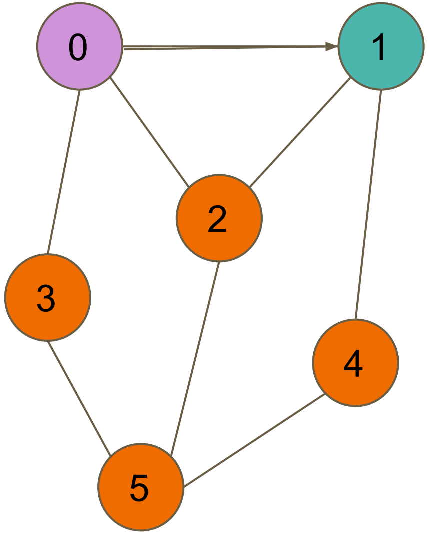
Ejemplo




```
dfs(G, 0)

dfs(graph G, node u)
    label u as discovered
    for v in G.adjacent[u]
        if v not discovered
            dfs(G, v)
```

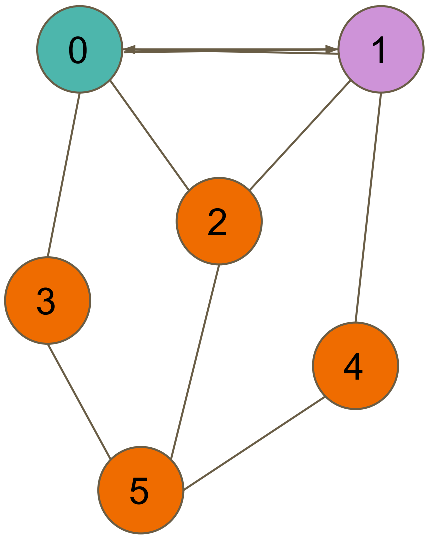
Ejemplo




```
dfs(G, 0)

dfs(graph G, node u)
    label u as discovered
    for v in G.adjacent[u]
        if v not discovered
            dfs(G, v)
```

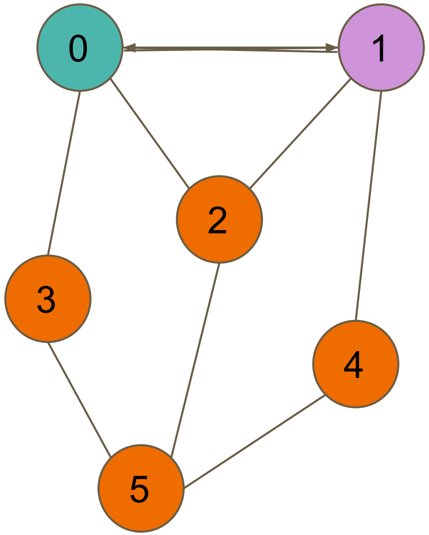
Ejemplo




```
dfs(G, 0)

dfs(graph G, node u)
    label u as discovered
    for v in G.adjacent[u]
        if v not discovered
            dfs(G, v)
```

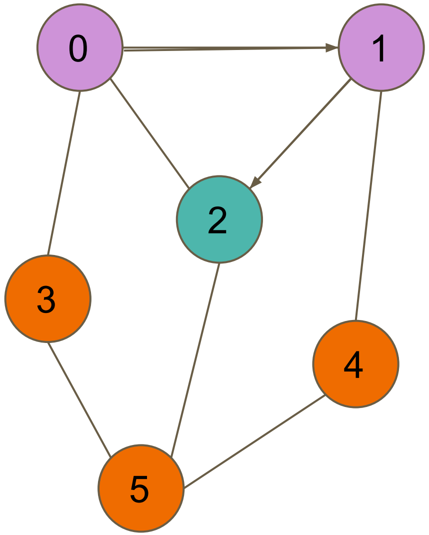
Ejemplo




```
dfs(G, 0)

dfs(graph G, node u)
    label u as discovered
    for v in G.adjacent[u]
        if v not discovered
            dfs(G, v)
```

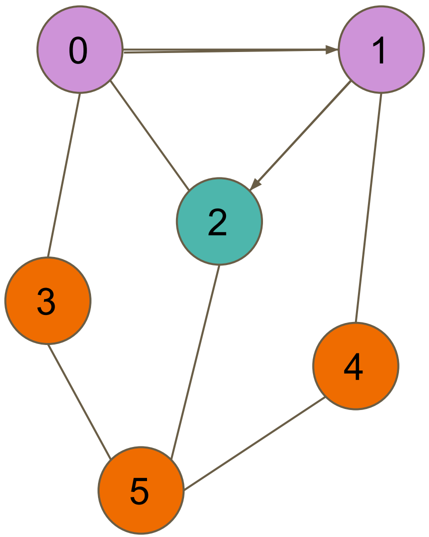
Ejemplo




```
dfs(G, 0)

dfs(graph G, node u)
    label u as discovered
    for v in G.adjacent[u]
        if v not discovered
            dfs(G, v)
```

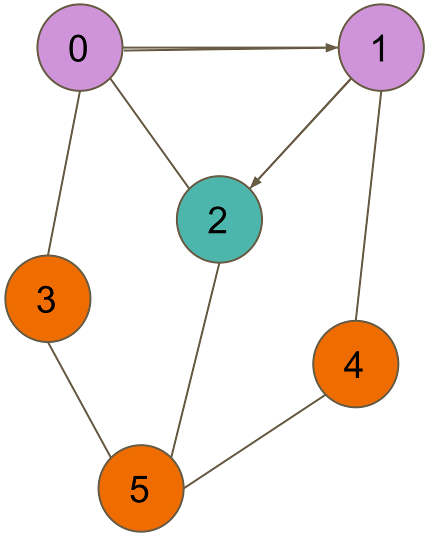
Ejemplo




```
dfs(G, 0)

dfs(graph G, node u)
    label u as discovered
    for v in G.adjacent[u]
        if v not discovered
            dfs(G, v)
```

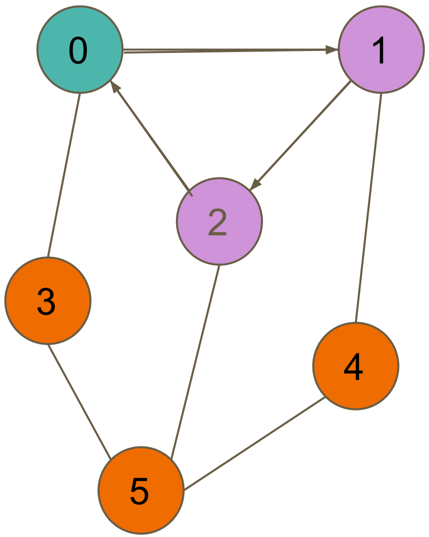
Ejemplo




```
dfs(G, 0)

dfs(graph G, node u)
    label u as discovered
    for v in G.adjacent[u]
        if v not discovered
            dfs(G, v)
```

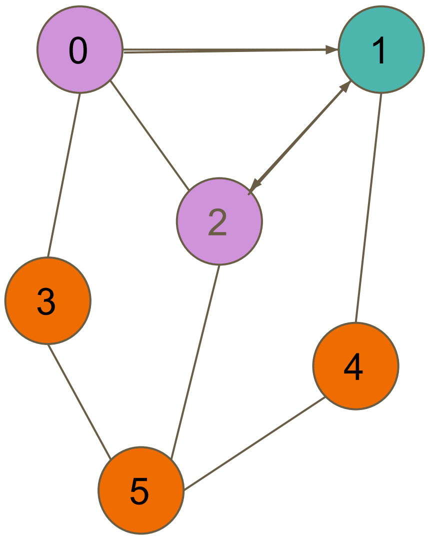
Ejemplo




```
dfs(G, 0)

dfs(graph G, node u)
    label u as discovered
    for v in G.adjacent[u]
        if v not discovered
            dfs(G, v)
```

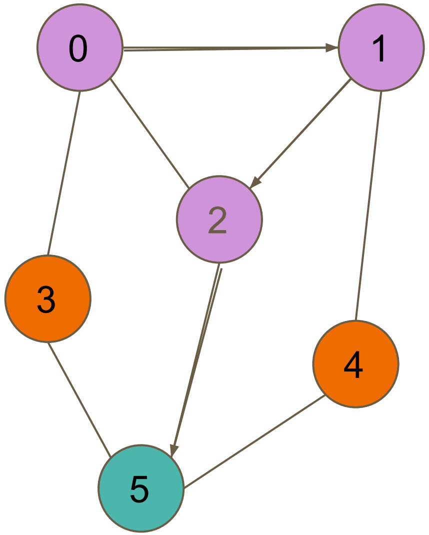
Ejemplo




```
dfs(G, 0)

dfs(graph G, node u)
    label u as discovered
    for v in G.adjacent[u]
        if v not discovered
            dfs(G, v)
```

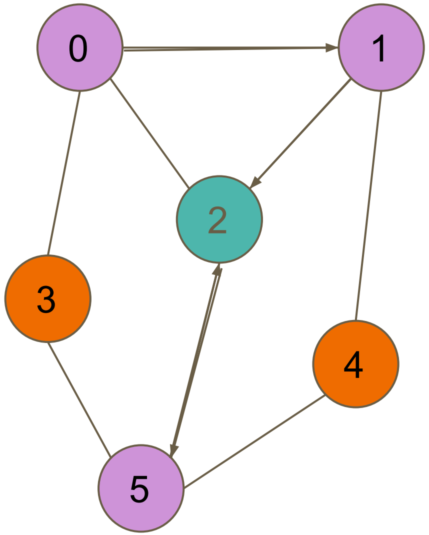
Ejemplo




```
dfs(G, 0)

dfs(graph G, node u)
    label u as discovered
    for v in G.adjacent[u]
        if v not discovered
            dfs(G, v)
```

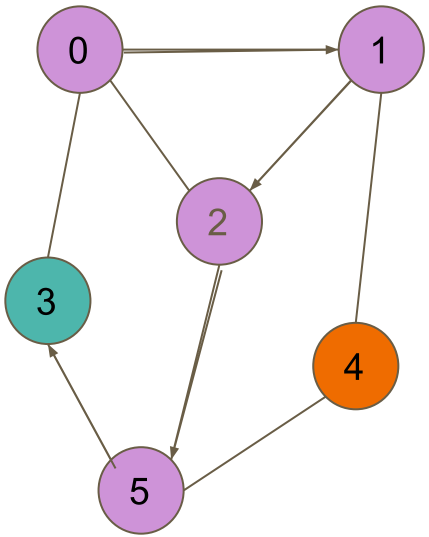
Ejemplo




```
dfs(G, 0)

dfs(graph G, node u)
    label u as discovered
    for v in G.adjacent[u]
        if v not discovered
            dfs(G, v)
```

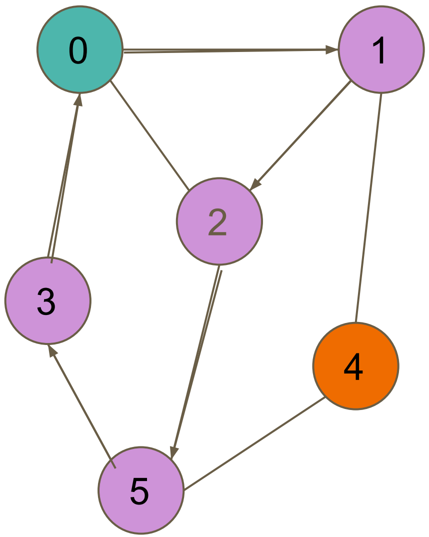
Ejemplo




```
dfs(G, 0)

dfs(graph G, node u)
    label u as discovered
    for v in G.adjacent[u]
        if v not discovered
            dfs(G, v)
```

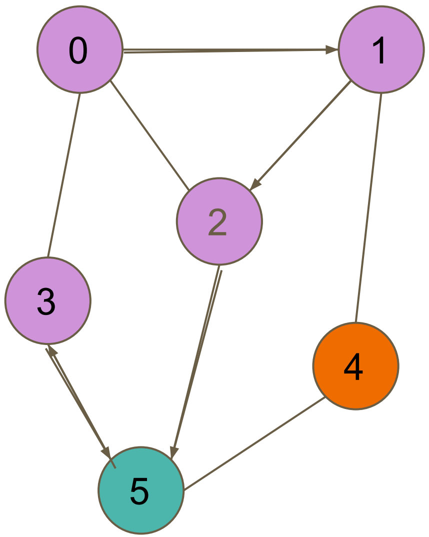
Ejemplo




```
dfs(G, 0)

dfs(graph G, node u)
    label u as discovered
    for v in G.adjacent[u]
        if v not discovered
            dfs(G, v)
```

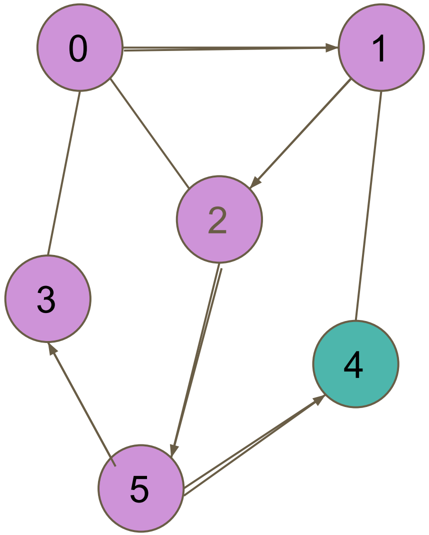
Ejemplo




```
dfs(G, 0)

dfs(graph G, node u)
    label u as discovered
    for v in G.adjacent[u]
        if v not discovered
            dfs(G, v)
```

Ejemplo



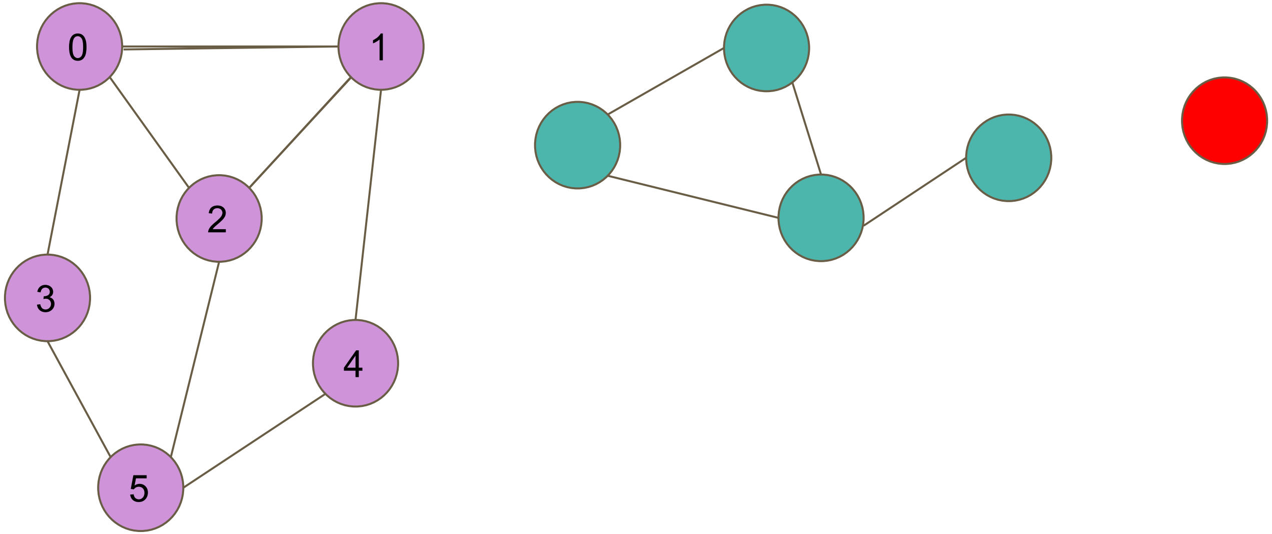

```
dfs(G, 0)


dfs(graph G, node u)
    label u as discovered
    for v in G.adjacent[u]
        if v not discovered
            dfs(G, v)
```

Nota: El algoritmo continúa por un más, pero no se mostrará el resto

Problema 1

Contar componentes conexas en un grafo no dirigido G



Problema 1

Notar: DFS recorre toda una componente conexa.

Problema 1

Idea: Ir por cada nodo y ver si ha sido visitado, si no correr DFS desde el nodo y sumar 1

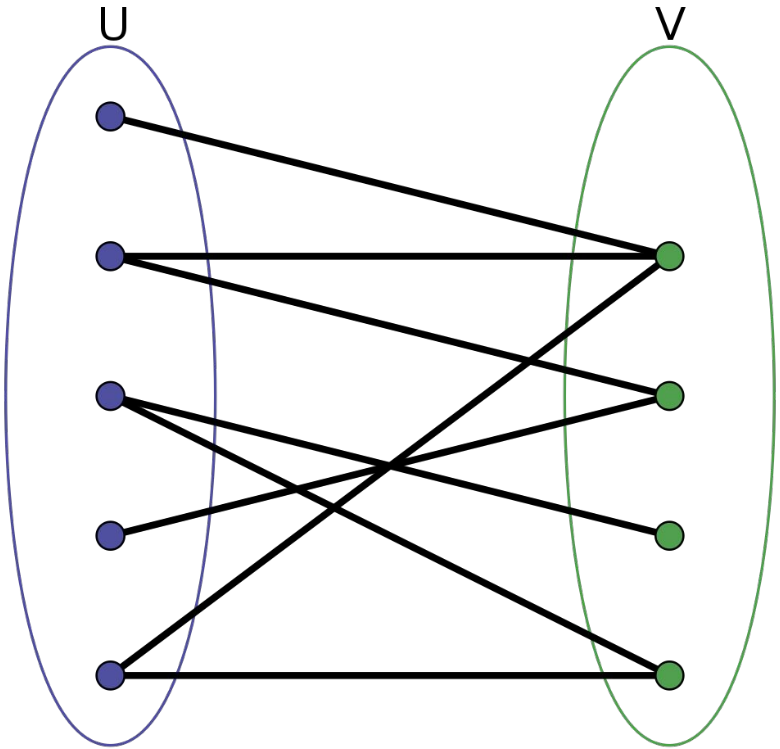
Problema 2

Dado un grafo G verificar si es bipartito

Definición: Un grafo G es bipartito, si es que se puede particionar en dos conjuntos U y V, tal que ningún nodo en U es vecino de otro nodo en U, y análogamente para V.

Problema 2

Bipartito



Problema 2

Definición Bi-coloración de un Grafo: Sea un grafo G, se llama a un función f de los nodos del grafo a {0,1} una bi-coloración de G, si cada nodo tiene un color distinto a sus vecinos.

Nota: Un grafo es bi-coloreable si y sólo si es bipartito.

Problema 2

Idea: Usar DFS para intentar bi-colorear un grafo.

Componente Fuertemente Conexa

Dado un grafo dirigido G, un subconjunto de nodos A es una componente fuertemente conexa (SCC (Strongly Connected Component)) si para cada par de nodos u, v en A existe un camino de u a v y un camino de v a u, y si es maximal (no está contenido en una SCC más grande).

Nota 1: Si G es no dirigido, se tiene que una componente es conexa si y sólo si es fuertemente conexa.

Nota 2: Dado dos SCC distintas A y B de un grafo dirigido G, la intersección de A con B es vacía.

Problema 3

Dado un grafo dirigido G, encontrar todas las SCC.

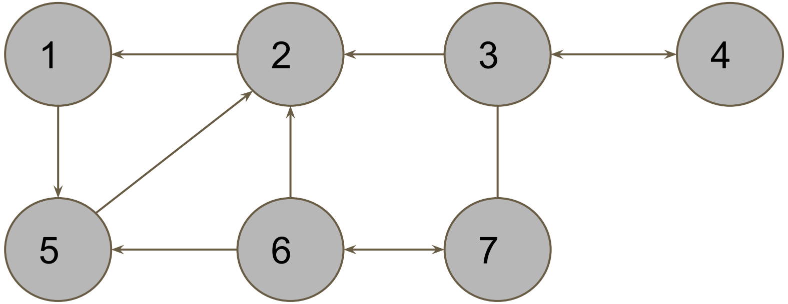
Problema 3

Solución: Algoritmo de Tarjan para SCC.

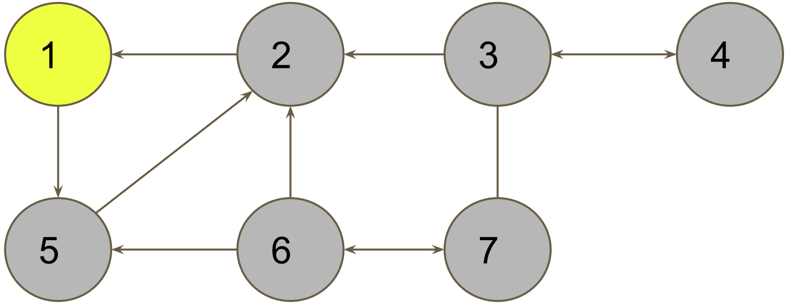
Idea general: Usar un stack e índices con un DFS para marcar las componentes

Tarjan

Stack



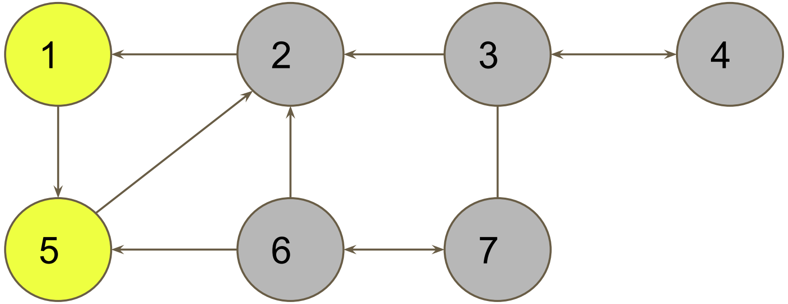
Tarjan



Stack

1

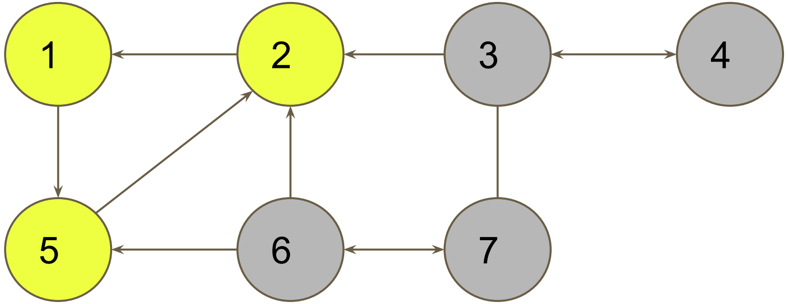
Tarjan



Stack

1
5

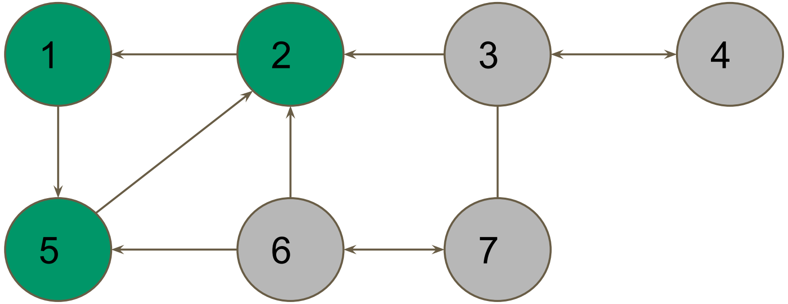
Tarjan



Stack

1
5
2

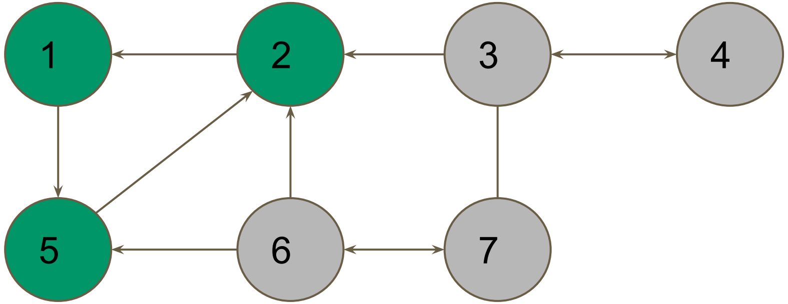
Tarjan



Stack

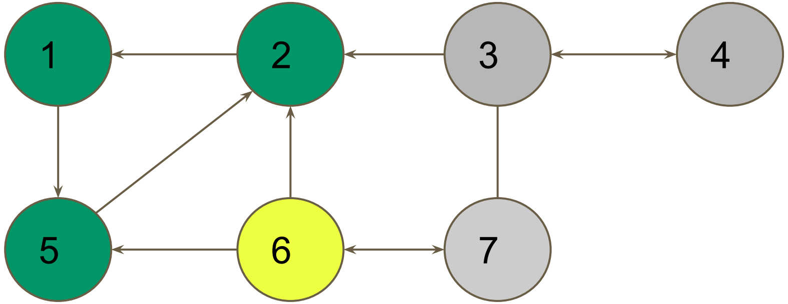
1
5
2

Tarjan



Stack

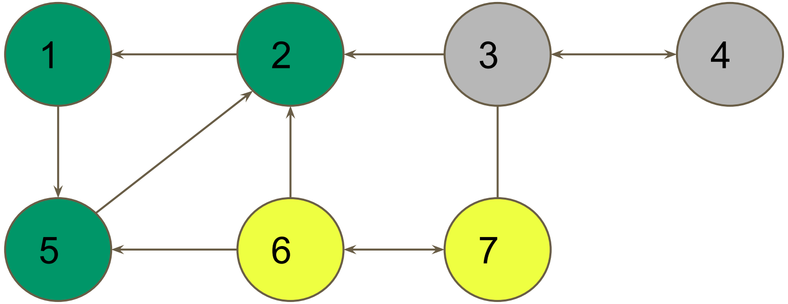
Tarjan



Stack

6

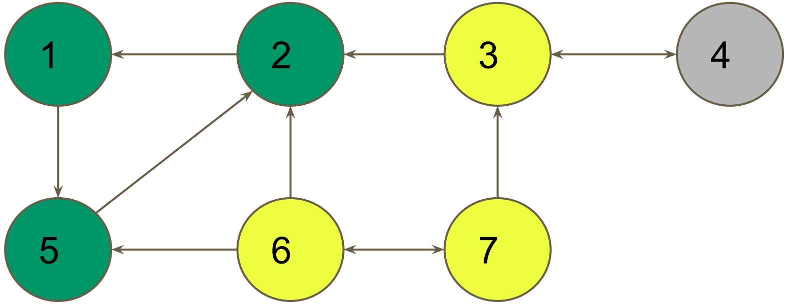
Tarjan



Stack

6
7

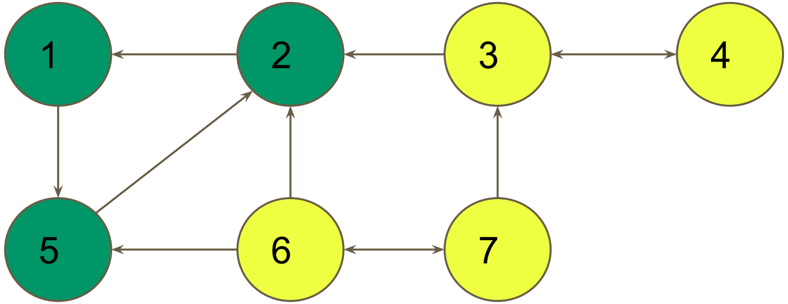
Tarjan



Stack

6
7
3

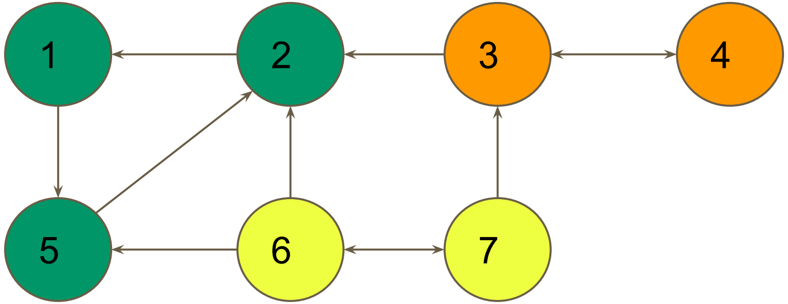
Tarjan



Stack

6
7
3
4

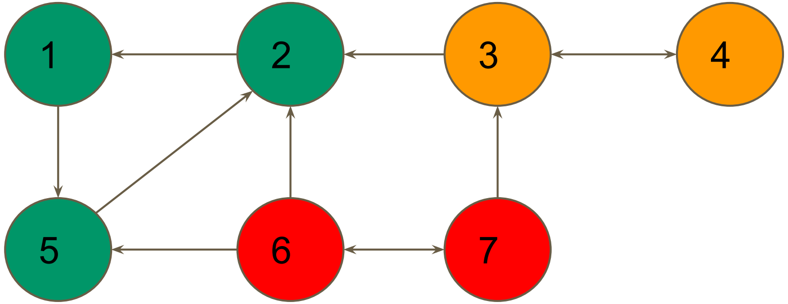
Tarjan



Stack

6
7

Tarjan

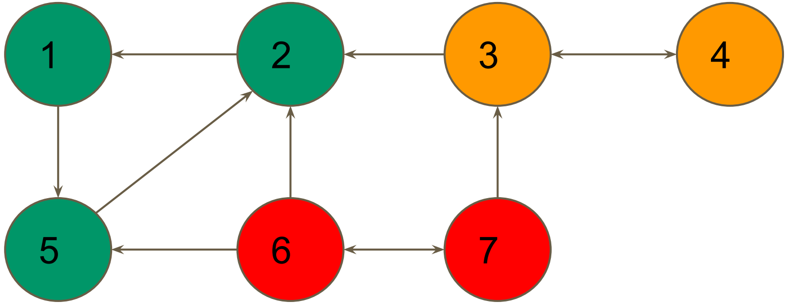


Stack

6
7

Tarjan

Stack



Tarjan

```
stack s
id = 0
tarjan(graph G)
	s = empty stack
	id = 0
	for v in G
		if v.id == -1
			dfs(G,v)
dfs(graph G, node v)
	v.id = id
	v.lowlink = id
	id++
	s.push(v)
	v.onStack = true
```

```
for u in G.adjacent[v]
	if u.id == -1
		dfs(G, v)
		v.lowlink =min(v.lowlink,u.lowlink)
	else if v.onStack
		v.lowlink =min(v.lowlink,u.lowlink)

if v.lowlink == v.id
	do
		u = s.pop()
		u.onStack = false
		add u to current SCC
	while u≠v
	output current SCC
```