Algoritmo de Kosaraju

Componentes fuertemente conectados en grafos dirigidos

Un componente fuertemente conectado (SCC) en un grafo dirigido $G=(V, E)$ es un conjunto de nodos $C \leq V$, tal que para cada par de nodos u y v en C se tiene que $u \rightarrow v$ y $v \rightarrow u$, es decir que ambos nodos son “alcanzables” el uno desde el otro.

En otras palabras la idea es encontrar el mayor número de “regiones” de un grafo dentro de las cuales se pueda ir desde cualquier punto hacia cualquier otro y viceversa.

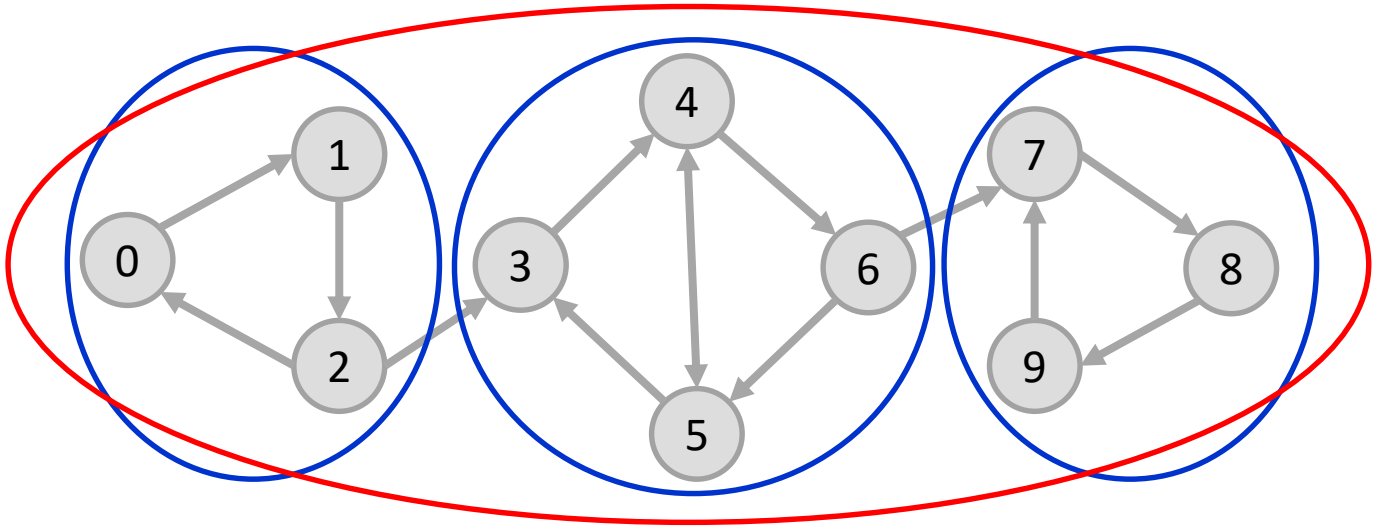
Componentes fuertemente conectados en grafos dirigidos

¿Qué sucedería si usamos DFS partiendo directamente del nodo 2?

Terminaríamos recorriendo todo el grafo sin “separar” los SCC

¿Qué sucedería si usamos DFS partiendo del nodo 8, luego del 5 y luego del 2?

Encontraríamos exactamente los 3 SCC del grafo



Componentes fuertemente conectados en grafos dirigidos

Parece ser entonces que el “éxito” a la hora de usar DFS para encontrar los SCC depende del orden en que se recorran los nodos.

La pregunta clave es entonces: ¿Cómo determinar ese orden?

Y la respuesta es … Mediante un “pre-recorrido” usando DFS

Ese doble recorrido se conoce como “algoritmo de Kosaraju”, el cual tiene una eficiencia $O(M+N)$

Algoritmo de Kosaraju

Dado un grafo dirigido G:

1. Sea G^i = G con todos sus aristas invertidas

2. Ejecutar DFS sobre G^i calculando los *f*(*v*) de todos los nodos, siendo *f*(*v*) el “tiempo de finalización” de *v*

3. Ejecutar DFS sobre G usando el orden decreciente de los *f*(*v*) y agrupando en SCC los nodos con el mismo punto de partida

Algoritmo de Kosaraju

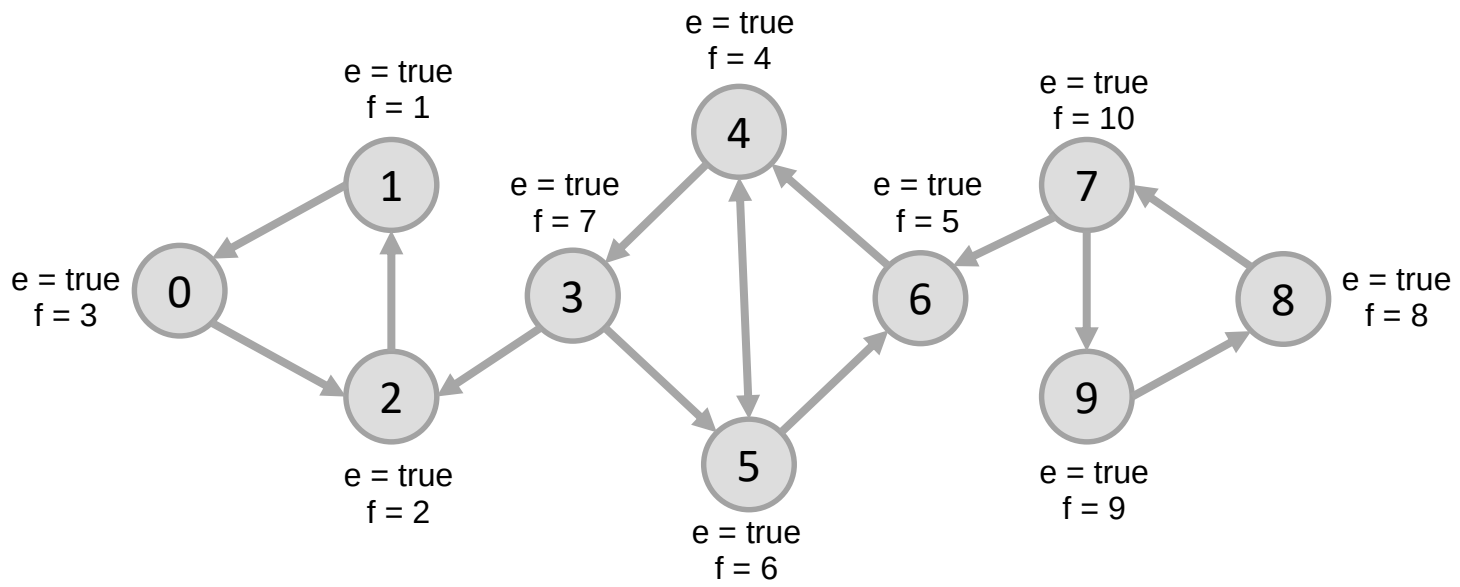
```
function kosaraju(grafo G):

   G = reverse(G)
   for each vertex u:
      u.explored = false
   t = 0
   for each vertex u (en cualquier orden):
      if u.explored = false:
         DFS1(G, u)

   G = reverse(G)
   for each vertex u:
      u.explored = false
   s = 0
   for each vertex u (desde f=N hasta 1):
      if u.explored = false:
         s++
         DFS2(G, u)
```

```
function DFS1(grafo G, nodo u):
   u.explored = true
   for each edge (u, v):
      if v.explored = false:
         DFS1(G, v)
   t++
   u.f = t
```

```
function DFS2(grafo G, nodo u):
   u.explored = true
   u.c = s
   for each edge (u, v):
      if v.explored = false:
         DFS2(G, v)
```

Algoritmo de Kosaraju

```
function DFS1(grafo G, nodo u):
   u.explored = true
   for each edge (u, v):
      if v.explored = false:
         DFS1(G, v)
   t++
   u.f = t
```



Algoritmo de Kosaraju

```
function DFS2(grafo G, nodo u){
   u.explored = true
   u.leader = s
   for each edge (u, v){
      if v.explored = false:
         DFS2(G, v)
```