

IIC-2133 — Estructuras de Datos y Algoritmos Grafos

Jorge A. Baier

Departamento de Ciencia de la Computación
Pontificia Universidad Católica de Chile
Santiago, Chile



Representando Grafos en Memoria

Dado un grafo $G = (V, E)$ estas son posibles representaciones:

- 1 *Listas de adyacencia*. Para cada $u \in V$, $Adj[u]$ es una lista con todos los v tales que $(u, v) \in E$.



Representando Grafos en Memoria

Dado un grafo $G = (V, E)$ estas son posibles representaciones:

- 1 *Listas de adyacencia.* Para cada $u \in V$, $Adj[u]$ es una lista con todos los v tales que $(u, v) \in E$.
- 2 *Matriz de adyacencia.* Definimos una matriz A , tal que

$$a_{ij} = \begin{cases} 1 & \text{si } (i, j) \in E \\ 0 & \text{en caso contrario} \end{cases}$$

¿Cuánta memoria se requiere usando estas representaciones?



Definiremos un algoritmo (BFS) para recorrer un grafo a partir de un nodo s .

- $color[u]$ puede ser alguno de estos:
 - *blanco* si no hemos visto nunca u .
 - *gris* hemos encontrado un camino hasta u .
 - *negro* hemos terminado de generar los nodos adyacentes a u .
- $\pi[u]$ el predecesor de u .
- $d[u]$ número de aristas en el camino descubierto hasta u
- Q estructura de datos que contiene a los nodos grises.



Búsqueda en Amplitud (Breadth-First Search)

```
1 function BFS(G, s)
2   for each  $u \in V[G]$  do
3      $color[u] \leftarrow blanco$ ;  $d[u] \leftarrow \infty$ ;  $\pi[u] \leftarrow nil$ 
4   Inserte s a Q
5    $color[s] \leftarrow gris$ ;  $d[s] \leftarrow 0$ 
6   while Q no está vacía do
7     Extraiga un elemento u desde Q
8     for each  $v \in Adj[u]$  do
9       if  $color[v] = blanco$  then
10          $\pi[v] \leftarrow u$ 
11          $d[v] \leftarrow d[u] + 1$ 
12         Inserte v a Q
13          $color[v] \leftarrow gris$ 
14      $color[u] \leftarrow negro$ 
```



Definición: Decimos que v es alcanzable desde u , o, simplemente, $u \rightsquigarrow v$, si existe una camino que comienza en u y termina en v .

Teorema: $s \rightsquigarrow t$ ssi después de una llamada a $\text{BFS}(G, s)$ se cumple que $\text{color}[t] = \text{negro}$.



Definición: Decimos que v es alcanzable desde u , o, simplemente, $u \rightsquigarrow v$, si existe una camino que comienza en u y termina en v .

Teorema: $s \rightsquigarrow t$ ssi después de una llamada a $\text{BFS}(G, s)$ se cumple que $\text{color}[t] = \text{negro}$.

Teorema: El tiempo de ejecución de $\text{BFS}(G, s)$ es $O(|E|)$.



Theorem

Dado un grafo $G = (V, E)$ y una función de peso unitaria ($w(e) = 1$, para todo $e \in E$), luego de una llamada a $BFS(G, s)$, $d[t] = \delta(s, t)$ para todo $t \in V$.

Definimos V_k como el conjunto de vértices a distancia k desde u .
Luego se procede por inducción en k .
Antes demostramos:

Lemma

Si durante una ejecución de $BFS(G, s)$ la cola contiene los elementos $\langle v_1, v_2, \dots, v_n \rangle$, entonces $d[v_n] \leq d[v_1] + 1$ y $d[v_i] \leq d[v_{i+1}]$ para cualquier $i \in \{1, \dots, n-1\}$.



Otros problemas que nos interesan:

- Encontrar ciclos en grafos.
- Encontrar componentes fuertemente conexas.
- Computar un orden topológico.

¿Podemos usar BFS o DFS para estos problemas?



Búsqueda en Profundidad (Depth-First Search)

```
1 procedure Init-DFS( $G$ )
2    $t \leftarrow 0$ 
3   for each  $u \in V[G]$  do
4      $\lfloor$   $color[u] \leftarrow blanco; d[u] \leftarrow \infty; \pi[u] \leftarrow nil$ 
5 procedure DFS( $G, s$ )
6    $\lfloor$  Init-DFS( $G$ )
7    $\lfloor$  DFS-visit( $G, s$ )
```



DFS (parte 2)

```
1 procedure DFS-visit( $G, s$ )
2    $color[s] \leftarrow gris$ 
3    $t \leftarrow t + 1$ 
4    $d[s] \leftarrow t$ 
5   for each  $t \in Adj[s]$  do
6     if  $color[t] = blanco$  then
7        $\pi[t] \leftarrow s$ 
8       DFS-visit( $G, t$ )
9    $color[s] \leftarrow negro$ 
10   $t \leftarrow t + 1$ 
11   $f[s] \leftarrow t$ 
```

Observación: podemos interpretar a $d[s]$ y $f[s]$ como los tiempos de “inicio” y “finalización” de s .



Cuando ejecutamos DFS completamente sobre un grafo, generamos lo que se conoce como un *bosque DFS*.

```
1 procedure DFS( $G$ )
2   Init-DFS( $G$ )
3   for each  $s \in V[G]$  do
4     if  $color[s] = blanco$  then
5       DFS-visit( $G, s$ )
```



¿Qué sucede cuando?

Si u, v son dos vértices en $G = (V, E)$, ¿es posible que:

- 1 $[d[u], f[u]]$ y $[d[v], f[v]]$ sean intervalos disjuntos?
- 2 $[d[u], f[u]]$ esté contenido en $[d[v], f[v]]$ o $[d[v], f[v]]$ esté contenido en $[d[u], f[u]]$?
- 3 $[d[u], f[u]]$ y $[d[v], f[v]]$ tengan intersección no vacía y no están contenidos el uno en el otro?



¿Qué sucede cuando?

Si u, v son dos vértices en $G = (V, E)$, ¿es posible que:

- 1 $[d[u], f[u]]$ y $[d[v], f[v]]$ sean intervalos disjuntos?
- 2 $[d[u], f[u]]$ esté contenido en $[d[v], f[v]]$ o $[d[v], f[v]]$ esté contenido en $[d[u], f[u]]$?
- 3 $[d[u], f[u]]$ y $[d[v], f[v]]$ tengan intersección no vacía y no están contenidos el uno en el otro?

Teorema del paréntesis: Sólo 1) y 2) se pueden dar.



Tree/Back Edges

Después de ejecutar $\text{DFS}(G)$

- 1 Decimos que (u, v) es un *tree edge* si v fue descubierto por primera vez desde u .
- 2 Decimos que (u, v) es un *back edge* si v es un ancestro de u en un árbol depth-first.

Propiedad: (u, v) es un *back edge* si v es gris cuando u es expandido. (Es decir, DFS se puede modificar para encontrar back-edges eficientemente)

Teorema: G tiene un ciclo ssi $\text{DFS}(G)$ descubre un *back edge*



Diga cómo usar o modificar DFS para:

- 1 Decidir si un grafo G tiene un ciclo.
- 2 Generar un orden topológico de $G = (V, E)$. (Definición: $<\subseteq V \times V$ es un orden topológico ssi $<$ es un orden total y para todo $(u, v) \in E$ se tiene $u < v$.)



Definición: Una componente fuertemente conexa (CFC) de un grafo $G = (V, E)$ es un subconjunto C maximal de V tal que para todo $u, v \in C$ $u \rightsquigarrow v$ (y $v \rightsquigarrow u$).



Definición: Una componente fuertemente conexa (CFC) de un grafo $G = (V, E)$ es un subconjunto C maximal de V tal que para todo $u, v \in C$ $u \rightsquigarrow v$ (y $v \rightsquigarrow u$).

Algoritmo de Kosarju (para calcular todas las CFC)

- 1 Llame a $\text{DFS}(G)$ para computar los tiempos de término $f[u]$ para cada vértice u .
- 2 Compute G^T .
- 3 Llame a $\text{DFS}(G^T)$, ordenando los nodos decrecientemente según f en el loop principal (línea 3, del pseudocódigo de $\text{DFS}(G)$).
- 4 Imprimir cada árbol DFS encontrado como un CFC.

