

I. ¿Qué es un grafo y para qué sirve?

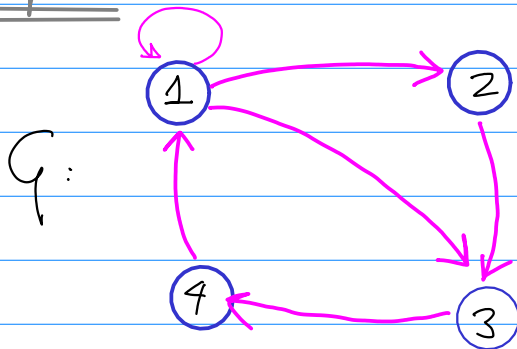
Def. Un grafo dirigido G es una pareja $(V(G), E(G))$ donde

$V(G)$ = conjunto finito llamado "vértices" de G .

$$E(G) \subseteq V(G) \times V(G)$$


conjunto de parejas ordenadas llamado las "aristas" de G

Ejemplo:



$$V(G) = \{1, 2, 3, 4\}$$

$$E(G) = \{(1, 2), (1, 3), (2, 3), (3, 4), (4, 1), (1, 1)\}$$

Note: A veces nos restringimos a grafos
no dirigidos ($\{u, v\}$ es arista $\Leftrightarrow (u, v)$ y (v, u)
se dibujan sin flechas  $\in E(G)$)
sin loops (no permitimos $(u, u) \in E(G)$)

Las siguientes nociones básicas van a ser importantes, para nosotros: camino, camino simple, ciclo, distancia entre vértices, grafo conexo, grafo fuertemente conexo.

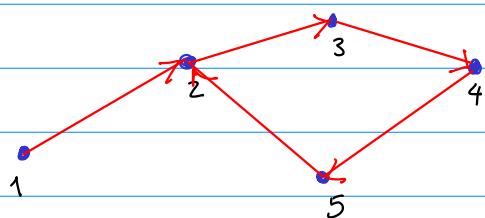
Sea G un grafo y sean $u, v \in V(G)$.

Def: Un camino desde u hasta v es una sucesión $u = w_0, w_1, w_2, \dots, w_k = v$ con $w_i \in V(G)$ y $\forall i \in \{0, \dots, k-1\} (w_i, w_{i+1}) \in E(G)$.

La longitud de este camino es k .

El camino es simple si ningún vértice se repite.

Un ciclo en G es un camino w_0, w_1, \dots, w_k con $k \geq 2$, $w_0 = w_k$ y todos los demás vértices distintos.



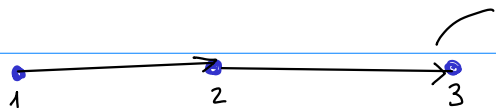
(i) $(1, 2, 3, 2, 3)$ es un camino de 1 a 3 [NO simple] de longitud 4.

(ii) $(2, 3, 4, 5)$ es un ciclo

Def: Un grafo NO DIRIGIDO es conexo si para todo par de vértices u, v existe un camino de u a v .

Un grafo DIRIGIDO es fuerentemente conexo si para todo $u, v \in V(G)$ existe un camino de u a v y un camino de v a u .

Obs: En este grafo dirigido hay un camino de 1 a 3 pero no de 3 a 1.



Def: La distancia desde u hasta v

$d(u, v) :=$ mínima longitud de un camino desde u hasta v .

(se define $d(u, v) = \infty$ si NO hay caminos desde u hasta v).

Por qué son importantes los grafos?

Porque sirven para modelar relaciones binarias entre objetos y estas relaciones son muy comunes como muestran los siguientes ejemplos:

Ejemplo 1: Redes físicas : T:

$V(T) = \text{"Ciudades del mundo"}$

$(c_1, c_2) \in E(T) \Leftrightarrow \text{"hay al menos un vuelo de } c_1 \text{ a } c_2 \text{ cada día"}$

Ejemplo 2: Grafos de conocimiento A

$V(A) = \text{"Actores de cine"}$

$(a_1, a_2) \in E(A) \Leftrightarrow \text{"} a_1 \text{ y } a_2 \text{ actúan en al menos una película"}$

World Wide Web W

$V(W) = \text{"páginas web"}$

$(p, q) \in E(W) \Leftrightarrow \text{"hay un link de } p \text{ a } q"$

Ejemplo 3: Grafos de planeación: R

$V(R) = \text{"Posibles configuraciones de un cubo de rubik"}$

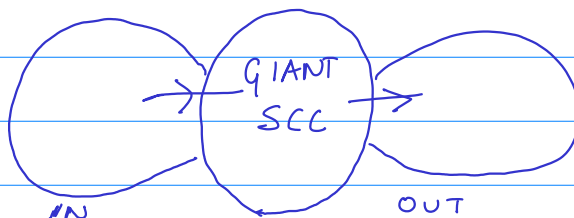
$(p_1, p_2) \in E(R) \Leftrightarrow \text{"es posible pasar de } a \text{ a } b \text{ mediante un movimiento básico."}$

¿Qué podemos aprender de organizar la información mediante grafos? Mucho...

[www.oracleofbacon.org /](http://www.oracleofbacon.org/)

¿Cómo imaginar el INTERNET?

$|V| = 200 \times 10^6$ millones
 $|E| = 1.5 \times 10^9$ billones



ISLAS
[Dibujos de cinco islas pequeñas representadas por rectángulos rojos]

Cómo representar un grafo G en una computadora?
Hay dos maneras principales:

(i) Matriz de adyacencia:

$$A \in \{0,1\}^{V(G) \times V(G)}$$

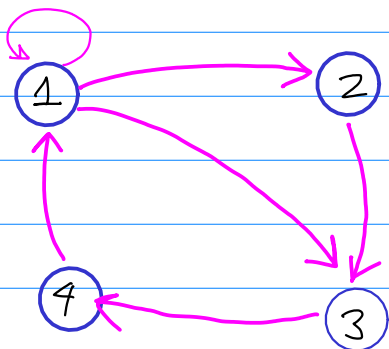
$$A_{ij} = \begin{cases} 1 & \text{si } (i,j) \in E(G) \\ 0 & \text{de lo contrario} \end{cases}$$

(ii) Lista de adyacencia: Recordamos dos estructuras:

- $V(G) = [1, 2, \dots, n]$ (lista)
- Un diccionario con $\text{Out}(a)$
(keys, Values) = $(a, \{b_1, \dots, b_t\})$

Def: Para $a \in V(G)$, $\text{Out}(a) = \{b \in V(G) : (a,b) \in E(G)\}$
 (a,b) son las aristas que salen de a .

Ejemplo:



(i)

	1	2	3	4
1	1	1	1	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0

(ii) $V(G) = [1, 2, 3, 4]$

dict: $\{1: \{1, 2, 3\},$
 $2: \{3\},$
 $3: \{4\},$
 $4: \{1\}\}$

Cuánta memoria se necesita para guardar un grafo G en función de sus parámetros principales $n := |V(G)|$ y $m := |E(G)|$?

[Memoria requerida para representar un grafo]

Lema: (i) Una matriz de adyacencia requiere $O(n^2)$

Una lista de adyacencia requiere $O(m+n)$

Dem: (i) entodos de una matriz $n \times n = n^2$.
(ii) $\text{out}(v) = \#$ de aristas que salen de v

$$\sum_{v \in V(G)} \text{out}(v) = |E| = m$$

luego la rep. en lista de adyacencia
requiere $n + m$ enteros.

Por eso la representación como lista de adyacencia
es típicamente preferida.

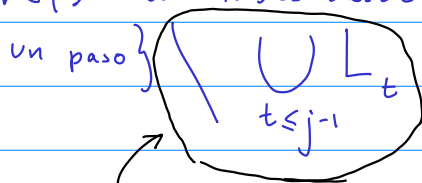
II. Breadth-First-Search (BFS) búsqueda por amplitud:

Sea G un grafo y sea $s \in V(G)$ fijo. En el
algoritmo BFS recorremos los vértices de G
iniciando en s y recorriendo G mediante "capas"
sucesivas cada vez más lejos de s , denotadas L_j

Concretamente definimos $L_0 := \{s\}$

y para $j \geq 1$

$$L_j = \{u \in V(G) \mid \text{alcantables desde } L_{j-1} \text{ en un paso}\}$$



Queremos dejar sólo los vértices no
vistos, en capas anteriores.

Teorema: [¿Qué es BFS?]

BFS calcula las distancias
más cortas entre s y todos
los demás vértices.

$$L_j = \{v \in V(G) : d(s, v) = j\}$$

$S_j := \{v \in V(G) : d(s, v) = j\}$. Mostremos $L_j = S_j$ por

Dem: inducción en j .

($j=0$) $S_0 = \{s\} = L_0$ ✓

si $u \in L_{j+1}$ existe $v \in L_j$ y $(v, u) \in E(G)$

por ind. $L_j = S_j$ luego hay un camino de long. j de s a v y conectándolo con (v, u)

concluimos $d(s, u) \leq j+1$. Si $d(s, u) \leq j$

por ind. $u \in L_t$, $t \leq j$ luego $d(s, u) = j+1$

y $u \in S_{j+1}$.

$L_{j+1} \subseteq S_{j+1}$

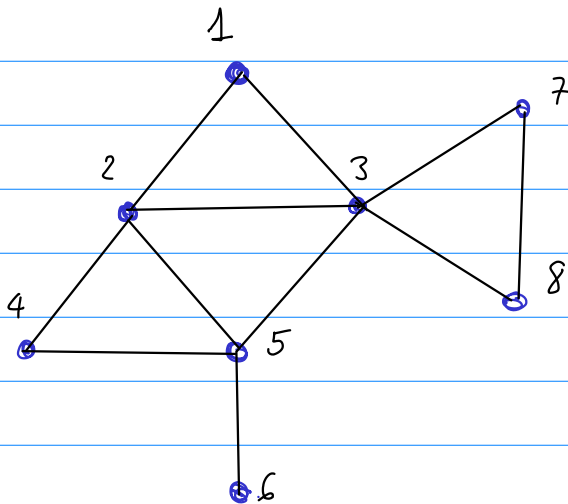
Recíprocamente, si $u \in S_{j+1}$ existe un camino

$s = v_0, v_1, \dots, v_{j+1} = u$ con $d(s, v_j) = j$

luego por ind. $v_j \in L_j$ y $u \notin L_t$, $t \leq j$

luego $u \in L_{j+1}$ demostrando la igualdad

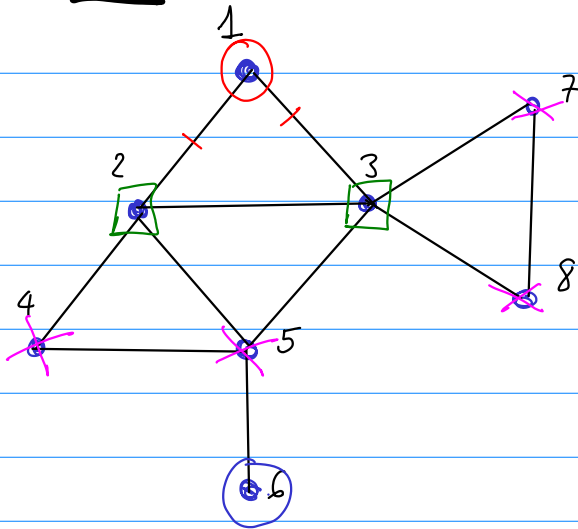
$S_{j+1} \subseteq L_{j+1}$



← Calcule los capos
de BFS iniciando
en ①

[Sigue abajo ...]

Sol.



$$L_0 = \{1\}$$

$$L_1 = \{2, 3\}$$

$$L_2 = \{4, 5, 7, 8\}$$

$$L_3 = \{6\}$$

IMPLEMENTACIÓN:

vertices-visitados = {s}

i = 0

L[0] = [s]

while L[i] not empty:

 L[i+1] = []

 for v in L[i]:

(*)

 for u in Out(v):

 if u not in vertices-visitados:

 vertices-visitados ← vertices-visitados ∪ {u}

 L[i+1].append(u)

 i = i + 1.

escogemos un
orden de visita
de los vecinos Out(v)



Lema: El algoritmo de arriba requiere tiempo $O(m)$

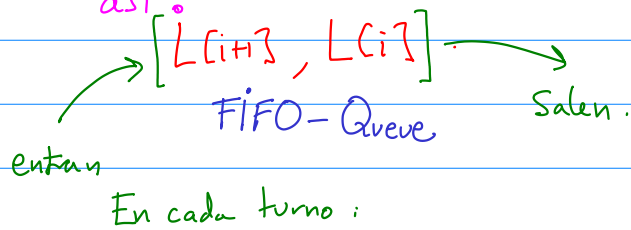
Dem: El ciclo principal (*) recorre a lo más todos los vértices (los L_j son una partición) y por cada vértice v recorre su conjunto de vecinos $Out(v)$ de tamaño d_v realizando $O(1)$ operaciones.

$$\sum_{v \in V} d_v O(1) \leq \sum_{v \in V} d_v O(1) = O(m).$$

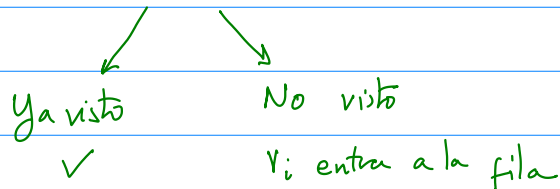
↑
MUY, MUY
rápido,
casi tanto
como leer
los datos
de entrada

[FOR FREE
ALGOS]

Obs: En el algoritmo anterior solo $L[i]$ y $L[i+1]$ son importantes en cada instante (la información relevante de $L[t]$, $t \leq i-1$ está en vertices-vistos). Esto permite limitar el espacio (memoria total utilizada) a máximo 2 capas así:



- (1) Sacamos un vértice de adelante u
- (2) Buscamos $Out(u) = \{v_1, \dots, v_j\}$
- (3) Para cada $v_i \in Out(u)$



Usando BFS iniciando en s podemos calcular rápidamente los números $d(s, v)$. Cómo encontrar un camino de s a v de longitud $d(s, v)$? (camino más corto).

Def: Un árbol es un grafo no dirigido conexo sin ciclos.

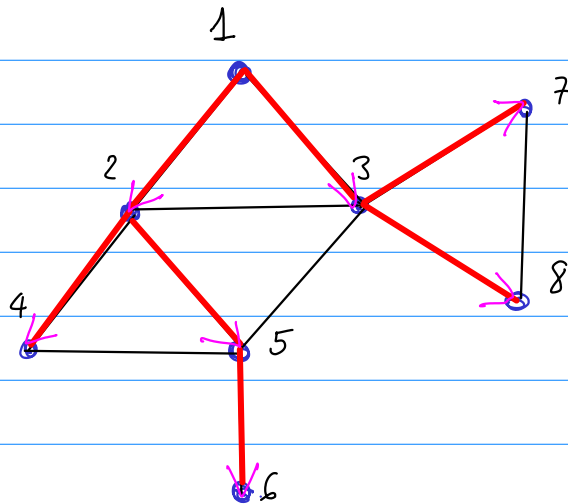
Def: Un árbol BFS T para G iniciando en s se construye así:

$$V(T) = \bigcup_{j \geq 0} L_j$$

$(v, u) \in E(T) \Leftrightarrow u$ se pone en vertices-vistos gracias a la arista (v, u)
[es decir a $u \in Out(v)$].

Obs: Lo podemos dirigir de s hacia abajo (ver flechas en la siguiente página)

Ejemplo:



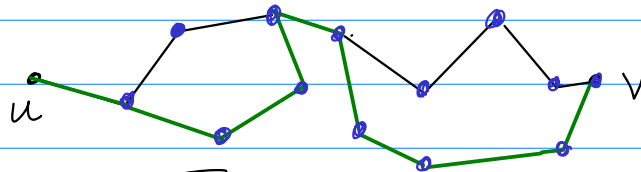
$$L_0 = \{1\}$$

$$L_1 = \{2, 3\}$$

T
(aristas
en rojo)

Lema: Todo par de vértices en un árbol T puede conectarse mediante un único camino simple.

Dem: Sean $u, v \in V(T)$ y suponga que hay dos caminos simples distintos



entonces T tiene ciclos, así que no es un árbol. (CONTRADICCIÓN!)

Teorema [Caminos óptimos desde s]

Sea G un grafo y sea T un árbol BFS iniciando en $s \in V(G)$. Para todo $u \in V(T)$ el camino (único) que une a s y u en T es un camino de longitud mínima de s hasta u en G .

Dem: Mostaremos primero que T es un árbol.
Por inducción en el índice L_j es fácil probar que todo vértice del árbol $en L_j$ admite un camino en T de longitud $\leq j$

desde s así que T es conexo. Si
hubiera un ciclo el elemento más lejano
a s habría sido descubierto dos
veces. (imposible) luego T es árbol.

$E(T) \subseteq E(G)$ así que

$$d_T(s, v) \geq d_G(s, v)$$

luego para $v \in L_j$ concluimos

$$d_T(s, v) = d_G(s, v)$$

demostrando el Teorema.