Grafos: Representación y Algormtimos Elementales

José Ortiz-Bejar

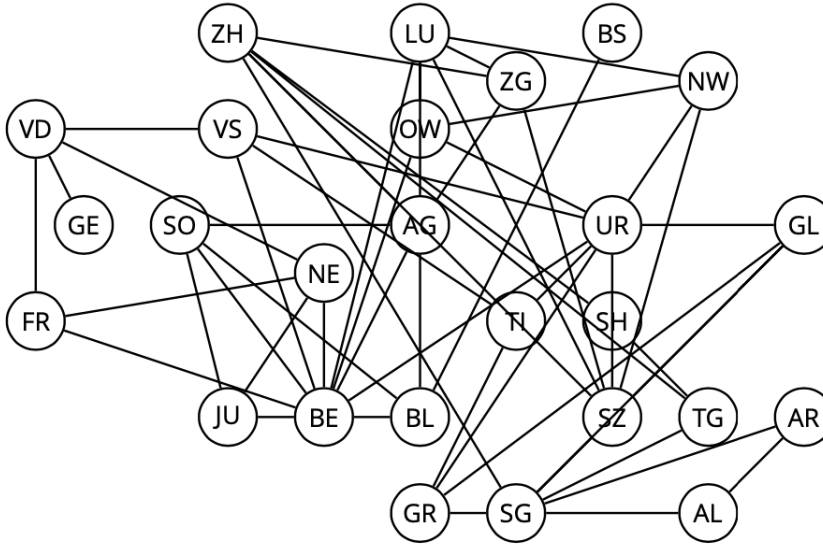
Facultad de Ingeniería Eléctrica
Universidad Michoacana de San Nicolás de Hidalgo

Abril, 2022

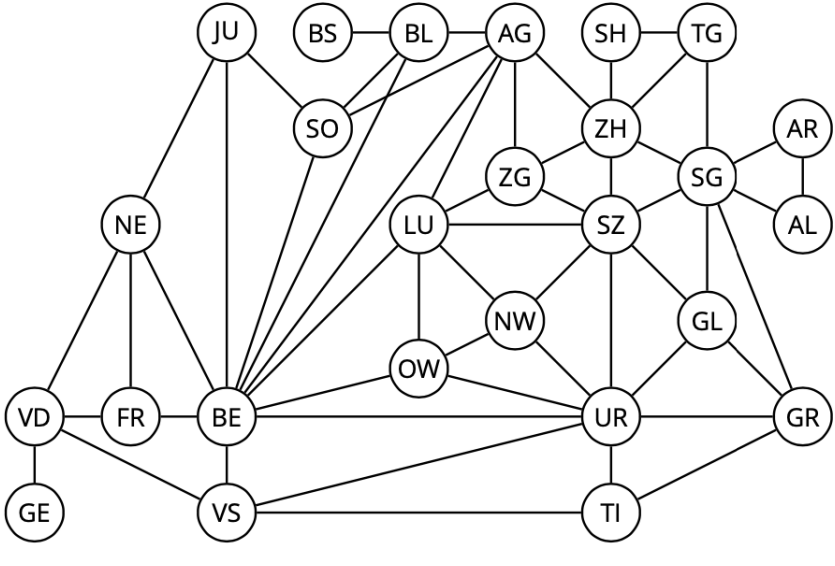
Definiciones

Representación

Ejemplo


ZH
LU
BS
ZG
NW
VD
VS
OW
GE
SO
AG
UR
GL
NE
FR
TI
SH
JU
BE
BL
SZ
TG
AR
GR
SG
AL

Mismo ejemplo (mejor organizado)



Aplicaciones

- Redes sociales: *quién conoce a quién*
- Grafo Web : *que páginas se enlazan*
- Grafo de internet: *como se enlazan las rutas*
- Grafo de citas: *referencias entres publicaciones*
- Grafos planos: *que países comparten frontera*
- Mallas: *figuras construidas con triangulos*
- Grafos geométricos: *quién es similar a quién*
- Grafos aleatorios: *cualquiera...*

Ejemplos y descripciones del curso de Daniel A. Spielman's course "Graphs and Networks."

Definiciones

- Un ***grafo***

$$G = (V, E)$$

- V es el conjunto de vértices ***vertices*** (o ***nodos***)
- E es el conjunto de ***aristas***

Definiciones

- Un ***grafo***

$$G = (V, E)$$

- V es el conjunto de vértices ***vertices*** (o ***nodos***)
- E es el conjunto de ***aristas***
  - $E \subseteq V \times V$, i.e., E es una ***relación entre vértices***
  - una arista $e = (u, v) \in V$ es un para de vértices tal que $u \in V$ y $v \in V$

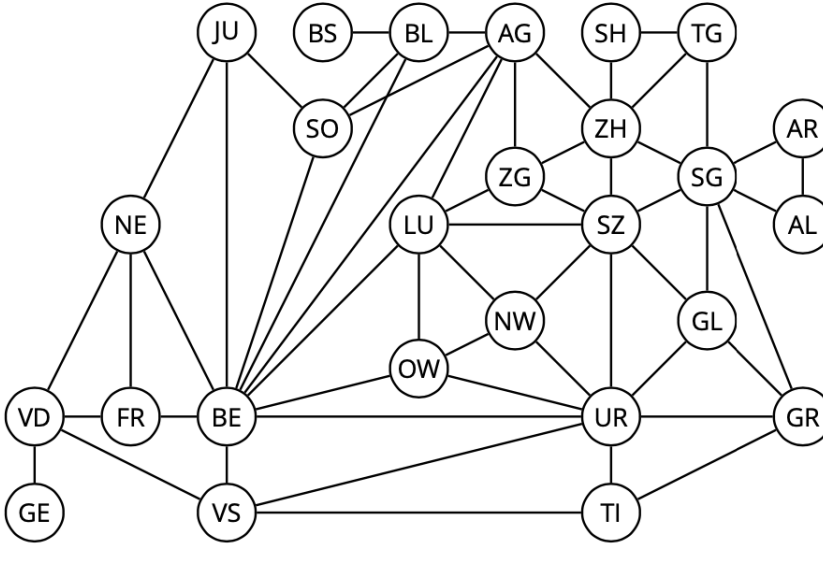
Definiciones

- Un ***grafo***

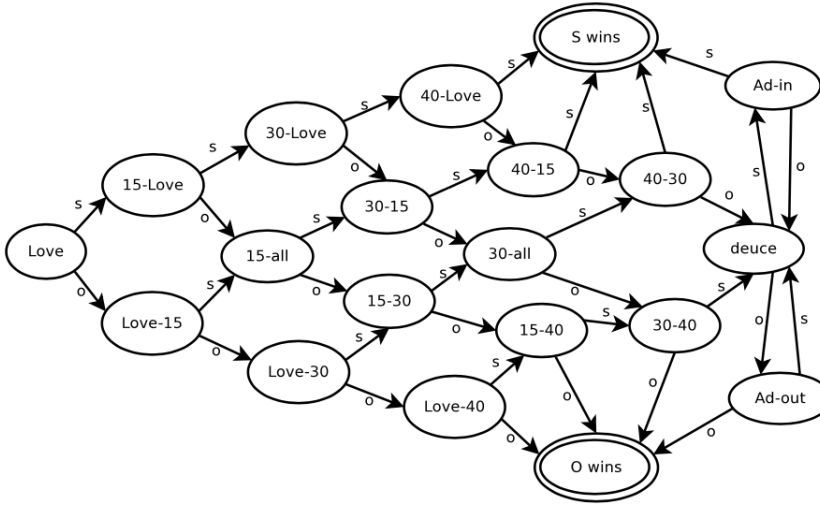
$$G = (V, E)$$

- V es el conjunto de vértices ***vertices*** (o ***nodos***)
- E es el conjunto de ***aristas***
  - $E \subseteq V \times V$, i.e., E es una ***relación entre vértices***
  - una arista $e = (u, v) \in V$ es un para de vértices tal que $u \in V$ y $v \in V$
- Un grafo *no-dirigido* es caracterrizado por una relación *simétrica* entre los vértices.
  - una arista en un conjunto de dos vértices $e = \{u, v\}$

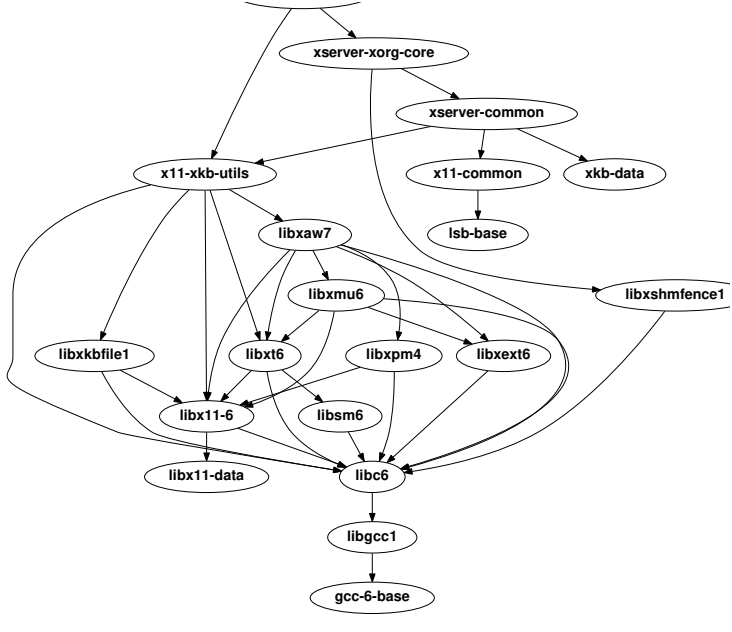
Ejemplo (1)


JU
BS
BL
AG
SH
TG
SO
ZH
AR
ZG
SG
NE
LU
SZ
AL
NW
GL
OW
VD
FR
BE
UR
GR
GE
VS
TI

Ejemplo (2)


S wins
Ad-in
40-Love
30-Love
40-15
40-30
15-Love
30-15
deuce
Love
15-all
30-all
15-30
Love-15
15-40
30-40
Love-30
Ad-out
Love-40
O wins
s
o

Ejemplo (3)


xserver-xorg-core
xserver-common
x11-xkb-utils
x11-common
xkb-data
lsb-base
libxaw7
libxmu6
libxshmfence1
libxkbfile1
libxt6
libxpm4
libxext6
libx11-6
libsm6
libx11-data
libc6
libgcc1
gcc-6-base

Representación de grafos

- ¿Cómo representamos un grafo $G = (E, V)$ en una computadora?

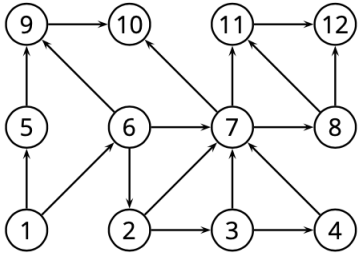
Representación de grafos

- ¿Cómo representamos un grafo $G = (E, V)$ en una computadora?
- *Lista de adyacencias*
- $V = \{1, 2, \ldots |V|\}$
- G consiste de un array *Adj*
- Un vértice $u \in V$ represntado por un elemento en *Adj*

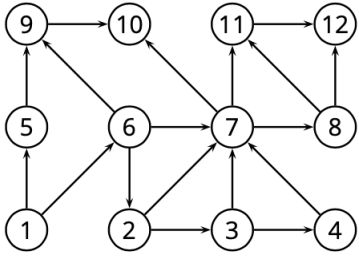
Representación de grafos

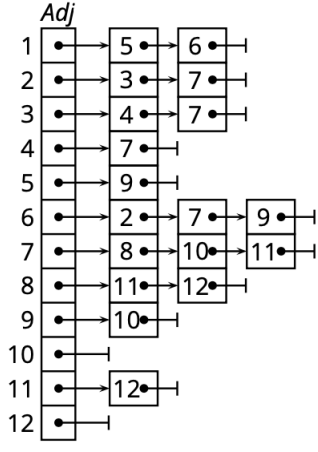
- ¿Cómo representamos un grafo $G = (E, V)$ en una computadora?
- *Lista de adyacencias*
- $V = \{1, 2, \ldots |V|\}$
- G consiste de un array Adj
- Un vértice $u \in V$ represntado por un elemento en Adj
- $Adj[u]$ es la ***lista de adyacencia*** de vértice u
  - la lista de vértices que son adyacentes a u
  - e.i., la lista de todos vértices v tal que $(u, v) \in E$

Ejemplo

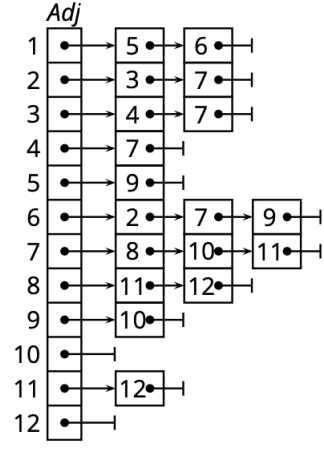


Ejemplo


9
10
11
12
5
6
7
8
1
2
3
4

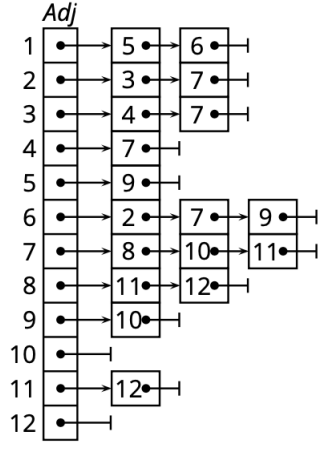

Adj
1
2
3
4
5
6
7
8
9
10
11
12
5
6
3
7
4
7
7
9
2
7
9
8
10
11
11
12
10
12

Lista de adyacencia complejidad



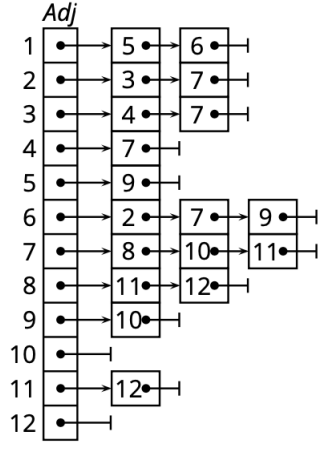
Lista de adyacencia complejidad

- Acceder al vértice *u*?



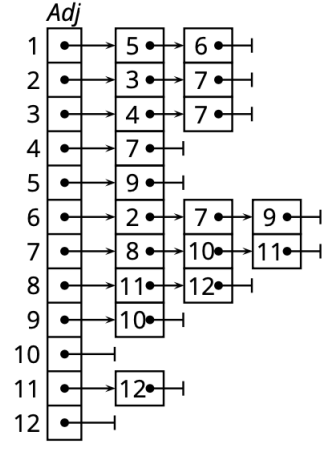
Lista de adyacencia complejidad

- Acceder al vértice u? $O(1)$
  - óptimo



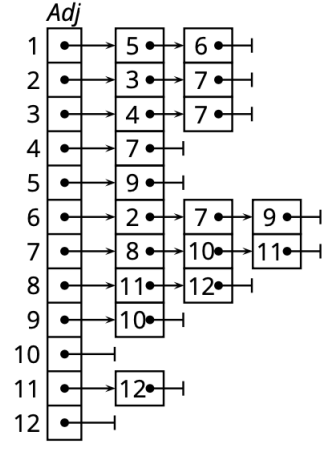
Lista de adyacencia complejidad

- Acceder al vértice u? $O(1)$
  - óptimo
- ¿Iterara V?



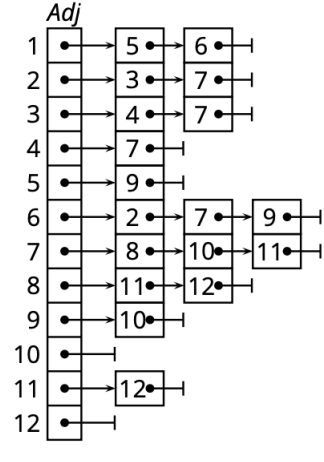
Lista de adyacencia complejidad

- Acceder al vértice u? $O(1)$
  - óptimo
- ¿Iterara V? $\Theta(|V|)$
  - óptimo



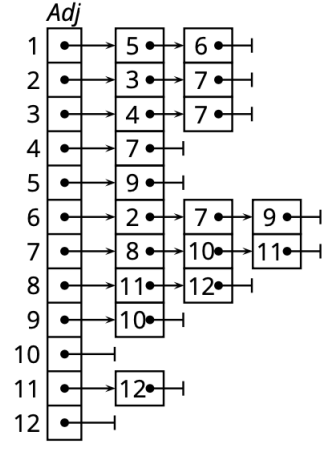
Lista de adyacencia complejidad

- Acceder al vértice u? $O(1)$
  - óptimo
- ¿Iterara V? $\Theta(|V|)$
  - óptimo
- ¿Iterar E?



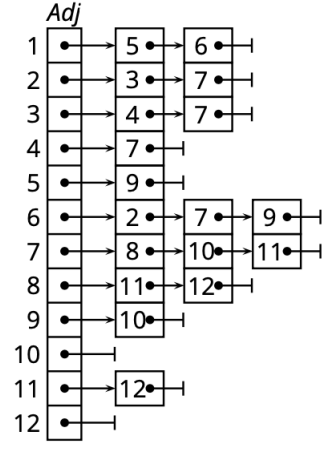
Lista de adyacencia complejidad

- Acceder al vértice u? $O(1)$
  - óptimo
- ¿Iterara V? $\Theta(|V|)$
  - óptimo
- ¿Iterar E? $\Theta(|V| + |E|)$
  - no óptimo



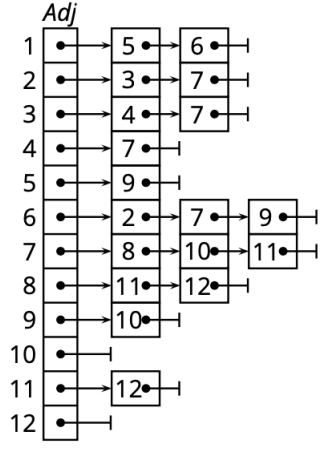
Lista de adyacencia complejidad

- Acceder al vértice u? $O(1)$
  - óptimo
- ¿Iterara V? $\Theta(|V|)$
  - óptimo
- ¿Iterar E? $\Theta(|V| + |E|)$
  - no óptimo
- ¿Verificar si $(u, v) \in E$?



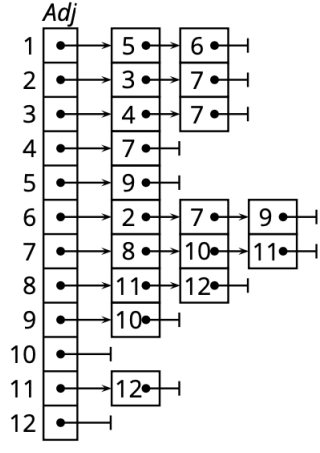
Lista de adyacencia complejidad

- Acceder al vértice u? $O(1)$
  - óptimo
- ¿Iterara V? $\Theta(|V|)$
  - óptimo
- ¿Iterar E? $\Theta(|V| + |E|)$
  - no óptimo
- ¿Verificar si $(u, v) \in E$? $O(|V|)$



Lista de adyacencia complejidad

- Acceder al vértice u? $O(1)$
  - óptimo
- ¿Iterara V? $\Theta(|V|)$
  - óptimo
- ¿Iterar E? $\Theta(|V| + |E|)$
  - no óptimo
- ¿Verificar si $(u, v) \in E$? $O(|V|)$
  - mal caso



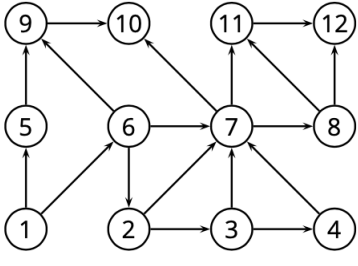
Representación de grafos (2)

Representación de grafos (2)

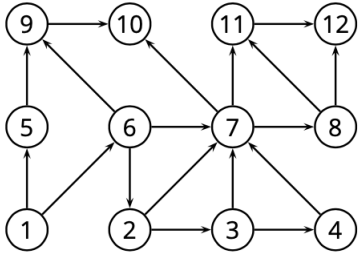
- *Representación de matriz de adyaciencias*
- $V = \{1, 2, \ldots |V|\}$
- G se representa por una matriz A de $|V| \times |V|$
- $A = (a_{ij})$ tal que

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

Ejemplo

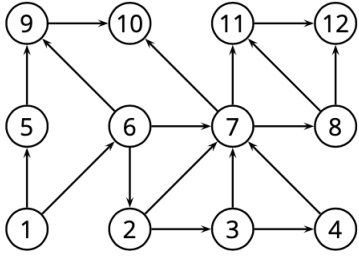


Ejemplo



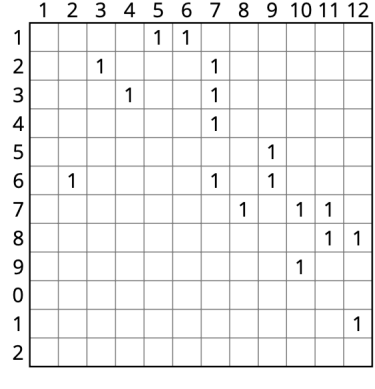
	1	2	3	4	5	6	7	8	9	10	11	12
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												

Ejemplo



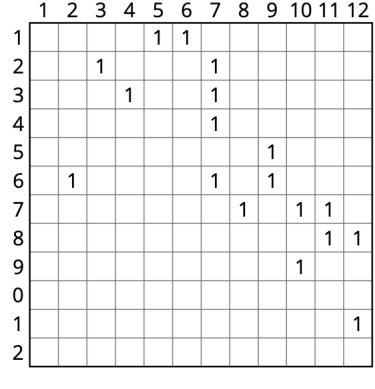
	1	2	3	4	5	6	7	8	9	10	11	12
1					1	1						
2			1				1					
3				1			1					
4							1					
5									1			
6		1					1		1			
7								1		1	1	
8											1	1
9										1		
10												
11												1
12												

Utilizando la lista de adyacencia



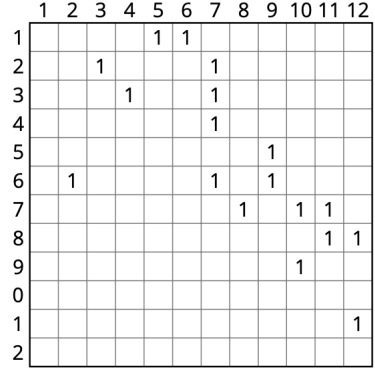
Utilizando la lista de adyacencia

- ¿Acceso al vértice *u*?



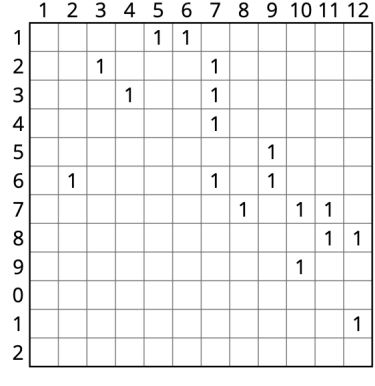
Utilizando la lista de adyacencia

- ¿Acceso al vértice u? $O(1)$
  - óptimo



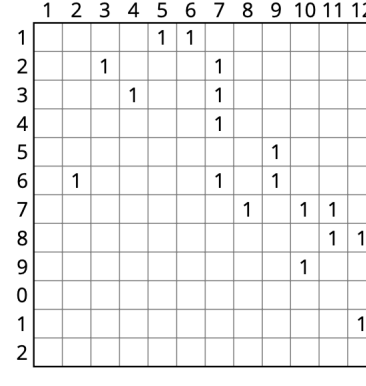
Utilizando la lista de adyacencia

- ¿Acceso al vértice u? $O(1)$
  - óptimo
- ¿Iterar V?



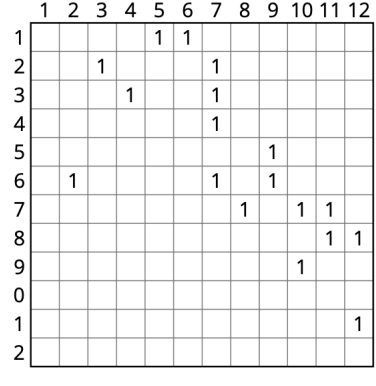
Utilizando la lista de adyacencia

- ¿Acceso al vértice u? $O(1)$
  - óptimo
- ¿Iterar V? $\Theta(|V|)$
  - óptimo



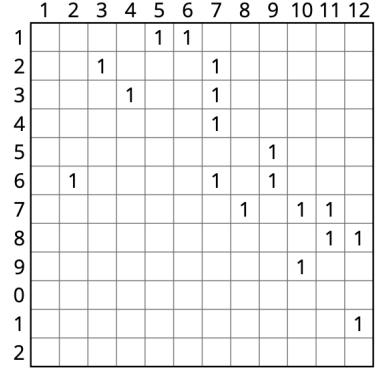
Utilizando la lista de adyacencia

- ¿Acceso al vértice u? $O(1)$
  - óptimo
- ¿Iterar V? $\Theta(|V|)$
  - óptimo
- ¿Iterar E?



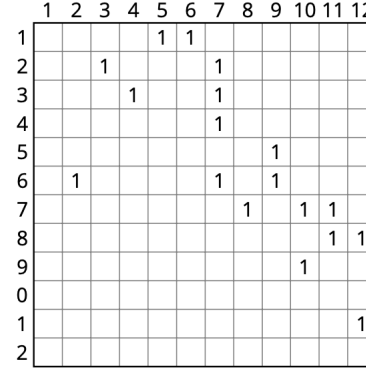
Utilizando la lista de adyacencia

- ¿Acceso al vértice u? $O(1)$
  - óptimo
- ¿Iterar V? $\Theta(|V|)$
  - óptimo
- ¿Iterar E? $\Theta(|V|^2)$
  - posiblmente muy malo



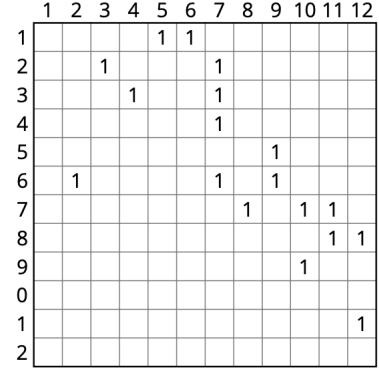
Utilizando la lista de adyacencia

- ¿Acceso al vértice u? $O(1)$
  - óptimo
- ¿Iterar V? $\Theta(|V|)$
  - óptimo
- ¿Iterar E? $\Theta(|V|^2)$
  - posiblmente muy malo
- ¿Verificar $(u, v) \in E$?



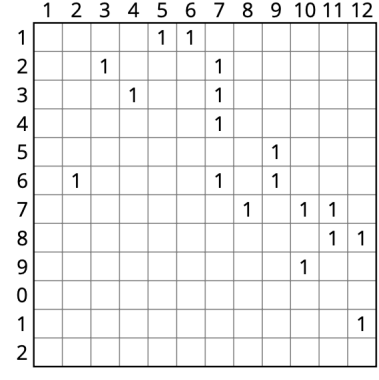
Utilizando la lista de adyacencia

- ¿Acceso al vértice u? $O(1)$
  - óptimo
- ¿Iterar V? $\Theta(|V|)$
  - óptimo
- ¿Iterar E? $\Theta(|V|^2)$
  - posiblmente muy malo
- ¿Verificar $(u, v) \in E$? $O(1)$



Utilizando la lista de adyacencia

- ¿Acceso al vértice u? $O(1)$
  - óptimo
- ¿Iterar V? $\Theta(|V|)$
  - óptimo
- ¿Iterar E? $\Theta(|V|^2)$
  - posiblmente muy malo
- ¿Verificar $(u, v) \in E$? $O(1)$
  - óptimo



Complejidad en espacio

Complejidad en espacio

- representación como lista de adyacencia

Complejidad en espacio

- representación como lista de adyacencia

$$\Theta(|V| + |E|)$$

Complejidad en espacio

- representación como lista de adyacencia

$$\Theta(|V| + |E|)$$

  óptimo

Complejidad en espacio

- representación como lista de adyacencia

$$\Theta(|V| + |E|)$$

  óptimo

- representación como matriz de adyacencia

Complejidad en espacio

- representación como lista de adyacencia

$$\Theta(|V| + |E|)$$

  óptimo

- representación como matriz de adyacencia

$$\Theta(|V|^2)$$

Complejidad en espacio

- representación como lista de adyacencia

$$\Theta(|V| + |E|)$$

  óptimo

- representación como matriz de adyacencia

$$\Theta(|V|^2)$$

  posiblemente muy malo

Complejidad en espacio

- representación como lista de adyacencia

$$\Theta(|V| + |E|)$$

  óptimo

- representación como matriz de adyacencia

$$\Theta(|V|^2)$$

  posiblemente muy malo

- ¿Cuándo la matriz de adyacencia “muy malo”?

Qué representación elegir

- Representación como lista de adyacencia
  - generalmente bueno, especialmente es óptimo para complejidad en espacio
  - malo para grafos ***densos*** que requieren acceso aleatorio a las aristas
  - preferible para grafos ***dispersos*** con un ***grado bajo***

Qué representación elegir

- Representación como lista de adyacencia
  - generalmente bueno, especialmente es óptimo para complejidad en espacio
  - malo para grafos ***densos*** que requieren acceso aleatorio a las aristas
  - preferible para grafos ***dispersos*** con un ***grado bajo***
- Representación como matriz de adyacencia
  - alta complejidad espacial
  - bueno para algoritmos que requieren acceso aleatorio a las aristas.
  - recomendable para grafos ***densos***

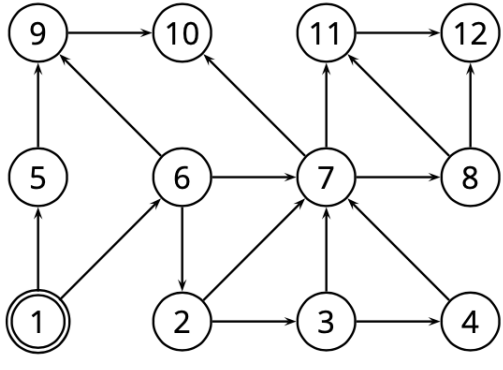
Búsqueda primero por amplitud (Breadth-First Search)

- Uno de los algoritmos fundamentales para grafos

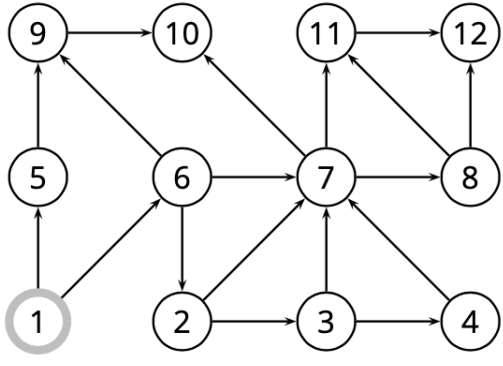
Búsqueda primero por amplitud (Breadth-First Search)

- Uno de los algoritmos fundamentales para grafos
- *Entrada:* $G = (V, E)$ un vértice $s \in V$
  - explora el grafo, descubre todos los nodos alcanzables desde s
  - itera todos lo vértices de forma incremental (distancia en aristas)
  - computa la distancia de cada vértice desde s
  - produce un árbol ***breadth-first tree*** con raíz en s
  - funciona en grafos *dirigidos* y *no-dirigidos*

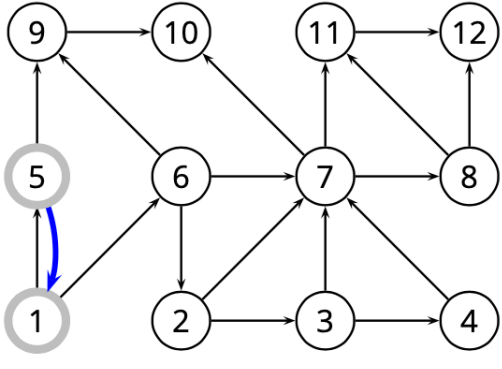
Ejemplo



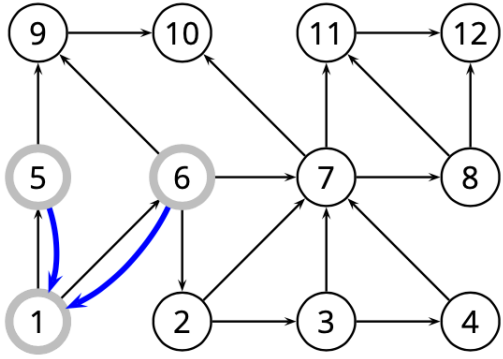
Ejemplo



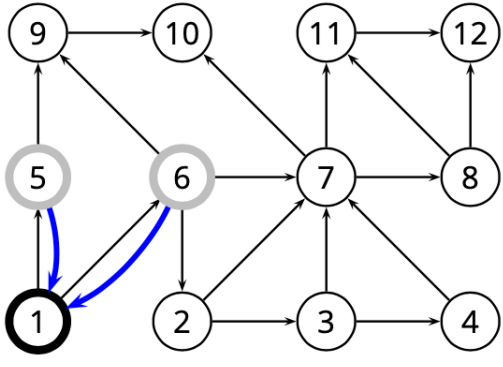
Ejemplo



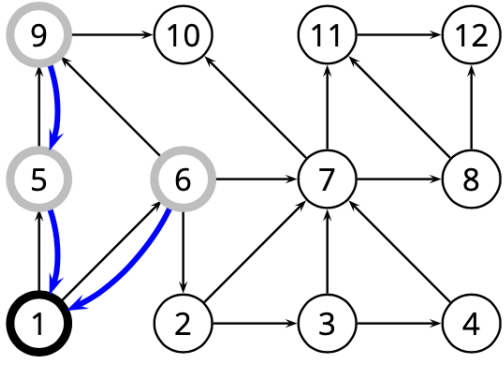
Ejemplo



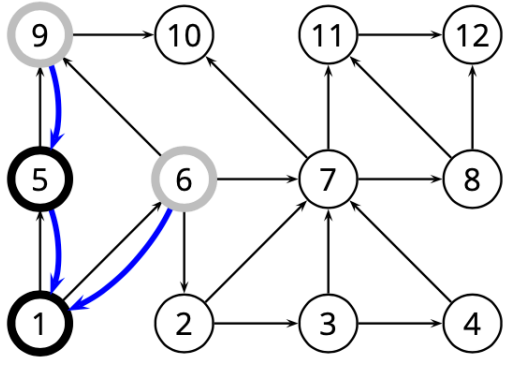
Ejemplo



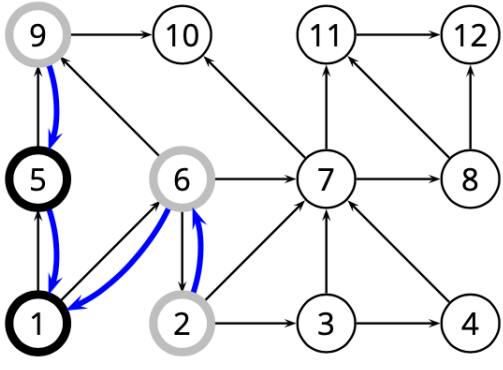
Ejemplo



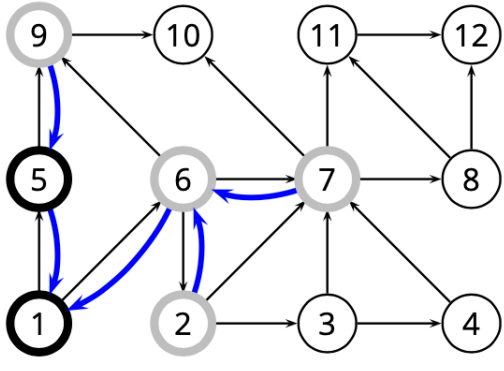
Ejemplo



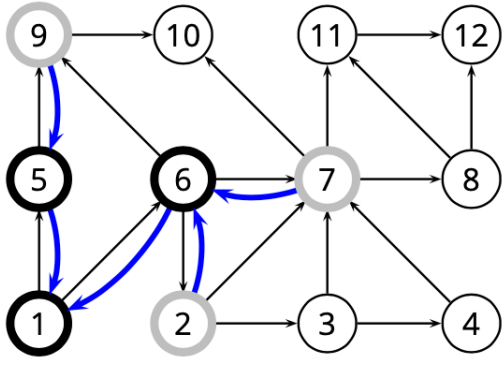
Ejemplo



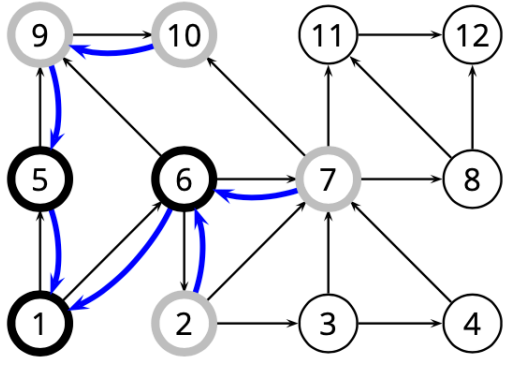
Ejemplo



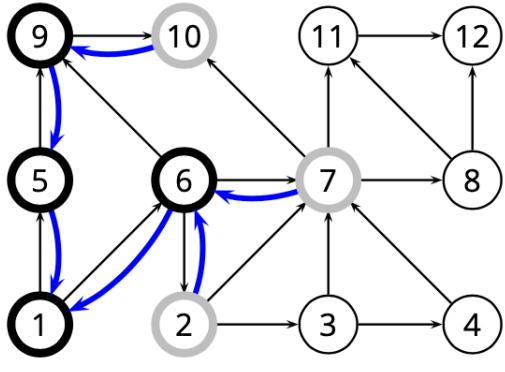
Ejemplo



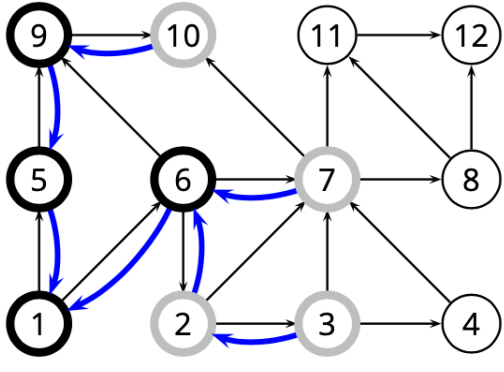
Ejemplo



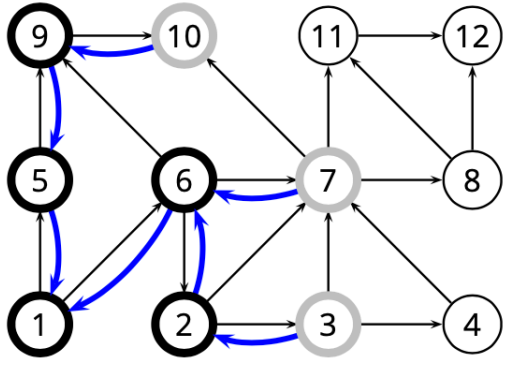
Ejemplo



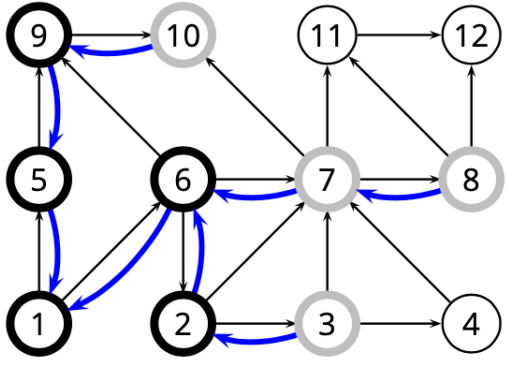
Ejemplo



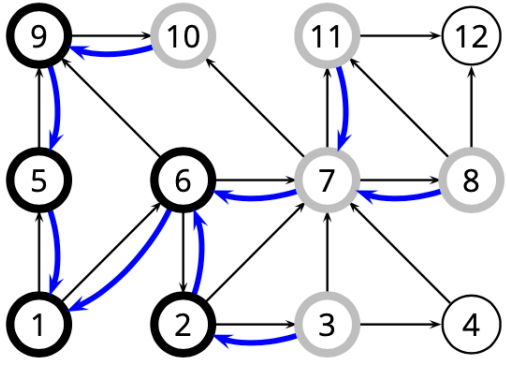
Ejemplo



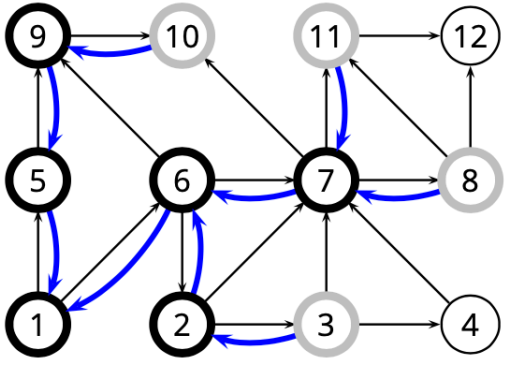
Ejemplo



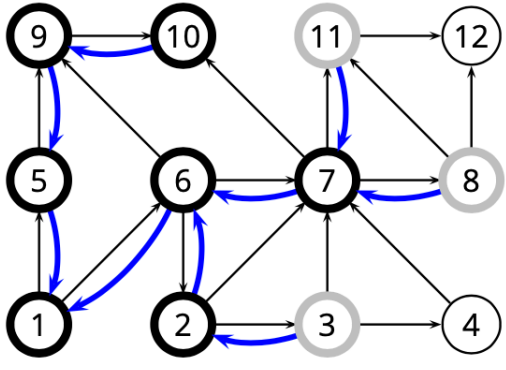
Ejemplo



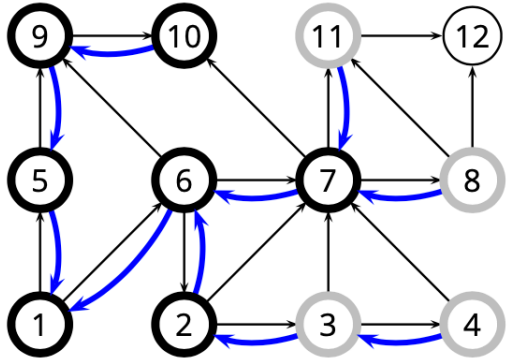
Ejemplo



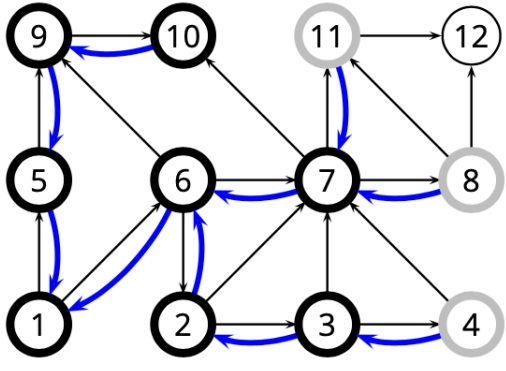
Ejemplo



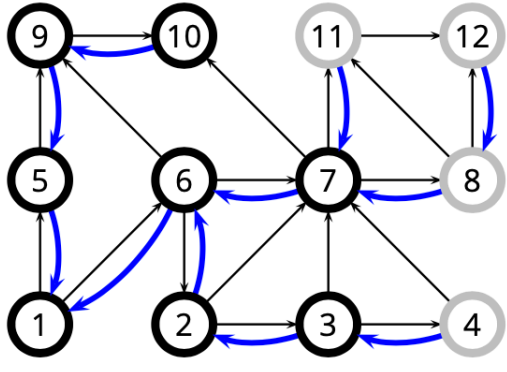
Ejemplo



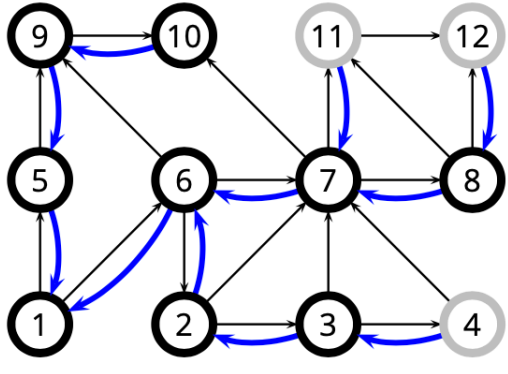
Ejemplo



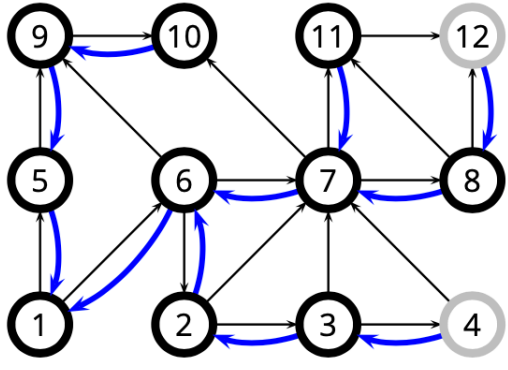
Ejemplo



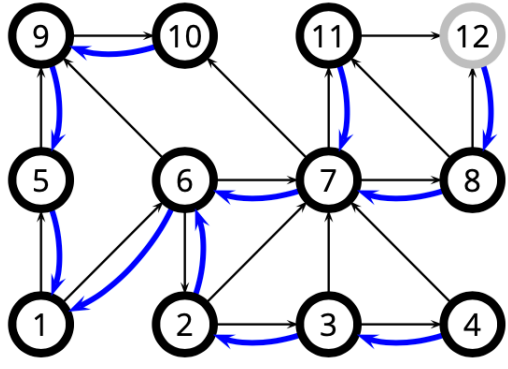
Ejemplo



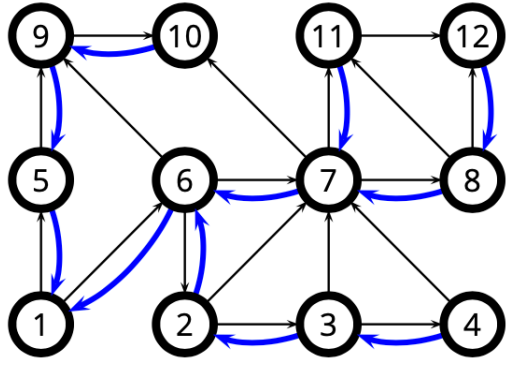
Ejemplo



Ejemplo

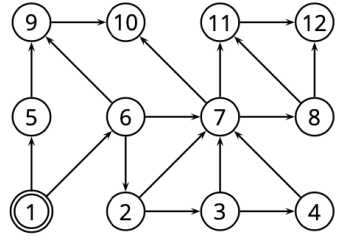


Ejemplo



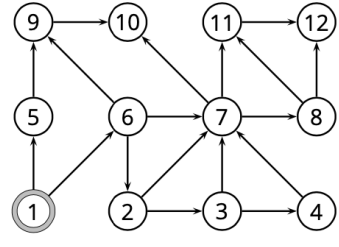
Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2       color[u] = white
 3       d[u] = ∞
 4       π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11       u = Dequeue(Q)
12       for each v ∈ Adj[u]
13            if color[v] == white
14                 color[v] = gray
15                 d[v] = d[u] + 1
16                 π[v] = u
17                 Enqueue(Q, v)
18       color[u] = black
```



Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2      color[u] = white
 3      d[u] = ∞
 4      π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11      u = Dequeue(Q)
12      for each v ∈ Adj[u]
13          if color[v] == white
14              color[v] = gray
15              d[v] = d[u] + 1
16              π[v] = u
17              Enqueue(Q, v)
18      color[u] = black
```

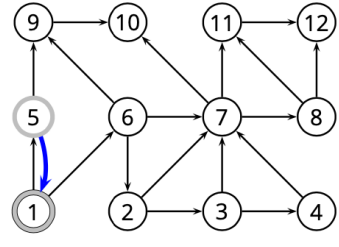


$u = 1$

$Q = \emptyset$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2        color[u] = white
 3        d[u] = ∞
 4        π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11        u = Dequeue(Q)
12        for each v ∈ Adj[u]
13              if color[v] == white
14                    color[v] = gray
15                    d[v] = d[u] + 1
16                    π[v] = u
17                    Enqueue(Q, v)
18        color[u] = black
```

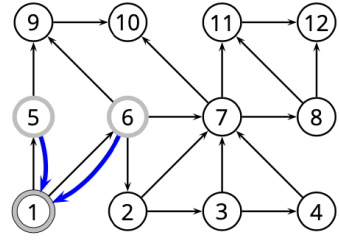


$u = 1$

$Q = \{5\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2        color[u] = white
 3        d[u] = ∞
 4        π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11        u = Dequeue(Q)
12        for each v ∈ Adj[u]
13              if color[v] == white
14                    color[v] = gray
15                    d[v] = d[u] + 1
16                    π[v] = u
17                    Enqueue(Q, v)
18        color[u] = black
```

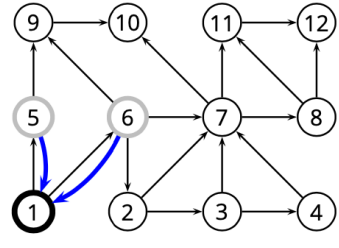


$u = 1$

$Q = \{5, 6\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2      color[u] = white
 3      d[u] = ∞
 4      π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11      u = Dequeue(Q)
12      for each v ∈ Adj[u]
13          if color[v] == white
14              color[v] = gray
15              d[v] = d[u] + 1
16              π[v] = u
17              Enqueue(Q, v)
18      color[u] = black
```

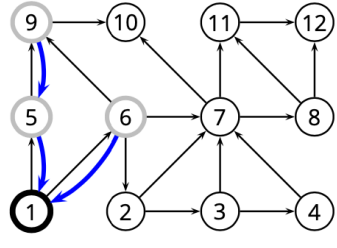


$u = 5$

$Q = \{6\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2        color[u] = white
 3        d[u] = ∞
 4        π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11        u = Dequeue(Q)
12        for each v ∈ Adj[u]
13              if color[v] == white
14                    color[v] = gray
15                    d[v] = d[u] + 1
16                    π[v] = u
17                    Enqueue(Q, v)
18        color[u] = black
```

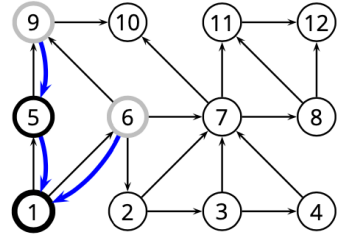


$u = 5$

$Q = \{6, 9\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2      color[u] = white
 3      d[u] = ∞
 4      π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11      u = Dequeue(Q)
12      for each v ∈ Adj[u]
13          if color[v] == white
14              color[v] = gray
15              d[v] = d[u] + 1
16              π[v] = u
17              Enqueue(Q, v)
18      color[u] = black
```

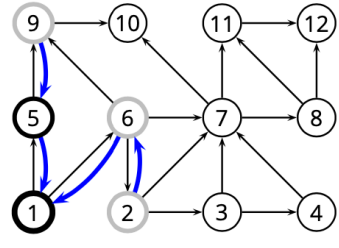


$u = 6$

$Q = \{9\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2      color[u] = white
 3      d[u] = ∞
 4      π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11      u = Dequeue(Q)
12      for each v ∈ Adj[u]
13          if color[v] == white
14              color[v] = gray
15              d[v] = d[u] + 1
16              π[v] = u
17              Enqueue(Q, v)
18      color[u] = black
```

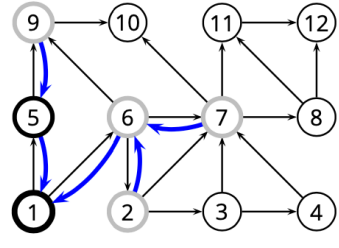


$u = 6$

$Q = \{9, 2, 7\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2      color[u] = white
 3      d[u] = ∞
 4      π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11      u = Dequeue(Q)
12      for each v ∈ Adj[u]
13          if color[v] == white
14              color[v] = gray
15              d[v] = d[u] + 1
16              π[v] = u
17              Enqueue(Q, v)
18      color[u] = black
```

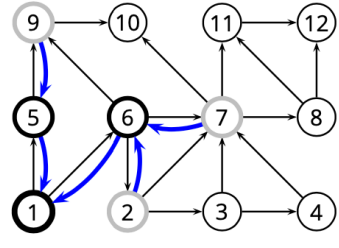


$u = 6$

$Q = \{9, 2, 7\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2      color[u] = white
 3      d[u] = ∞
 4      π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11      u = Dequeue(Q)
12      for each v ∈ Adj[u]
13          if color[v] == white
14              color[v] = gray
15              d[v] = d[u] + 1
16              π[v] = u
17              Enqueue(Q, v)
18      color[u] = black
```

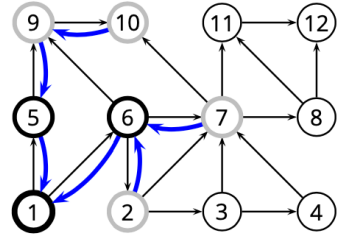


$u = 9$

$Q = \{2, 7\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2        color[u] = white
 3        d[u] = ∞
 4        π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11        u = Dequeue(Q)
12        for each v ∈ Adj[u]
13              if color[v] == white
14                    color[v] = gray
15                    d[v] = d[u] + 1
16                    π[v] = u
17                    Enqueue(Q, v)
18        color[u] = black
```

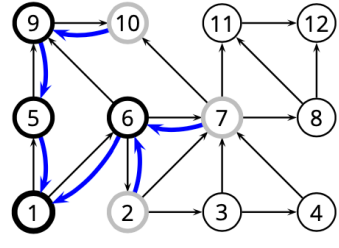


$u = 9$

$Q = \{2, 7, 10\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2      color[u] = white
 3      d[u] = ∞
 4      π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11      u = Dequeue(Q)
12      for each v ∈ Adj[u]
13          if color[v] == white
14              color[v] = gray
15              d[v] = d[u] + 1
16              π[v] = u
17              Enqueue(Q, v)
18      color[u] = black
```

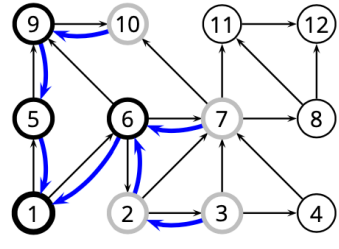


$u = 2$

$Q = \{7, 10\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2        color[u] = white
 3        d[u] = ∞
 4        π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11        u = Dequeue(Q)
12        for each v ∈ Adj[u]
13              if color[v] == white
14                    color[v] = gray
15                    d[v] = d[u] + 1
16                    π[v] = u
17                    Enqueue(Q, v)
18        color[u] = black
```

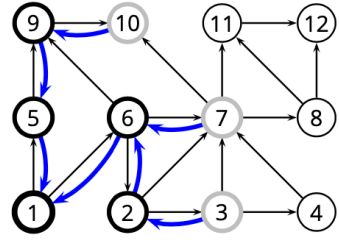


$u = 2$

$Q = \{7, 10, 3\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2        color[u] = white
 3        d[u] = ∞
 4        π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11        u = Dequeue(Q)
12        for each v ∈ Adj[u]
13              if color[v] == white
14                    color[v] = gray
15                    d[v] = d[u] + 1
16                    π[v] = u
17                    Enqueue(Q, v)
18        color[u] = black
```

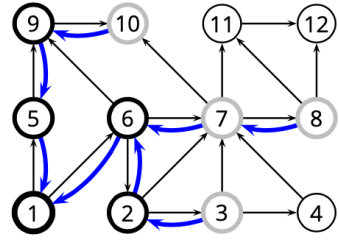


$u = 7$

$Q = \{10, 3\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2        color[u] = white
 3        d[u] = ∞
 4        π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11        u = Dequeue(Q)
12        for each v ∈ Adj[u]
13              if color[v] == white
14                    color[v] = gray
15                    d[v] = d[u] + 1
16                    π[v] = u
17                    Enqueue(Q, v)
18        color[u] = black
```

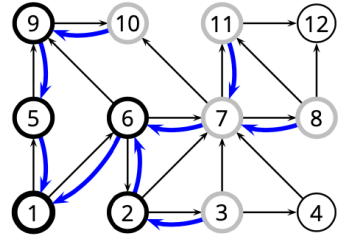


$u = 7$

$Q = \{10, 3, 8\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2        color[u] = white
 3        d[u] = ∞
 4        π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11        u = Dequeue(Q)
12        for each v ∈ Adj[u]
13              if color[v] == white
14                    color[v] = gray
15                    d[v] = d[u] + 1
16                    π[v] = u
17                    Enqueue(Q, v)
18        color[u] = black
```

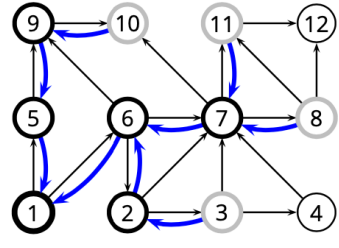


$u = 7$

$Q = \{10, 3, 8, 11\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2      color[u] = white
 3      d[u] = ∞
 4      π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11      u = Dequeue(Q)
12      for each v ∈ Adj[u]
13          if color[v] == white
14              color[v] = gray
15              d[v] = d[u] + 1
16              π[v] = u
17              Enqueue(Q, v)
18      color[u] = black
```

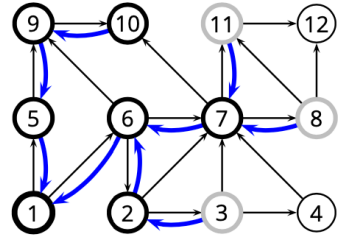


$u = 10$

$Q = \{3, 8, 11\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2        color[u] = white
 3        d[u] = ∞
 4        π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11        u = Dequeue(Q)
12        for each v ∈ Adj[u]
13              if color[v] == white
14                    color[v] = gray
15                    d[v] = d[u] + 1
16                    π[v] = u
17                    Enqueue(Q, v)
18        color[u] = black
```

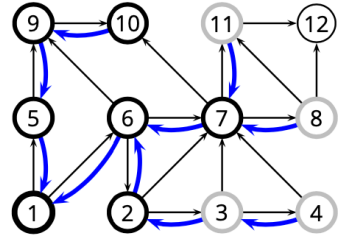


$u = 3$

$Q = \{8, 11\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2      color[u] = white
 3      d[u] = ∞
 4      π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11      u = Dequeue(Q)
12      for each v ∈ Adj[u]
13          if color[v] == white
14              color[v] = gray
15              d[v] = d[u] + 1
16              π[v] = u
17              Enqueue(Q, v)
18      color[u] = black
```

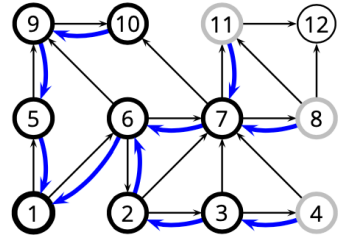


$u = 3$

$Q = \{8, 11, 4\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2      color[u] = white
 3      d[u] = ∞
 4      π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11      u = Dequeue(Q)
12      for each v ∈ Adj[u]
13          if color[v] == white
14              color[v] = gray
15              d[v] = d[u] + 1
16              π[v] = u
17              Enqueue(Q, v)
18      color[u] = black
```

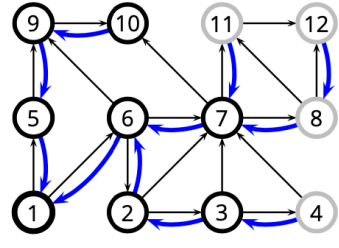


$u = 8$

$Q = \{11, 4\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2      color[u] = white
 3      d[u] = ∞
 4      π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11      u = Dequeue(Q)
12      for each v ∈ Adj[u]
13          if color[v] == white
14              color[v] = gray
15              d[v] = d[u] + 1
16              π[v] = u
17              Enqueue(Q, v)
18      color[u] = black
```

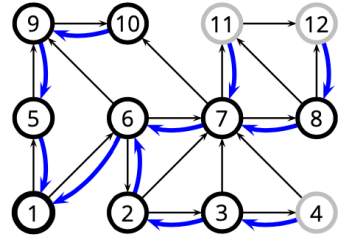


$u = 8$

$Q = \{11, 4, 12\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2        color[u] = white
 3        d[u] = ∞
 4        π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11        u = Dequeue(Q)
12        for each v ∈ Adj[u]
13              if color[v] == white
14                    color[v] = gray
15                    d[v] = d[u] + 1
16                    π[v] = u
17                    Enqueue(Q, v)
18        color[u] = black
```

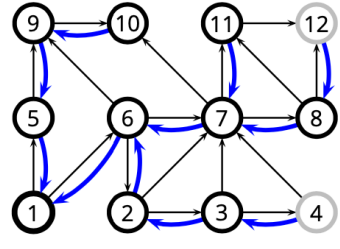


$u = 11$

$Q = \{4, 12\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2        color[u] = white
 3        d[u] = ∞
 4        π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11        u = Dequeue(Q)
12        for each v ∈ Adj[u]
13              if color[v] == white
14                    color[v] = gray
15                    d[v] = d[u] + 1
16                    π[v] = u
17                    Enqueue(Q, v)
18        color[u] = black
```

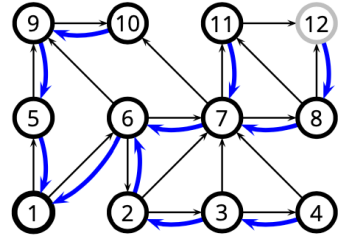


$u = 4$

$Q = \{12\}$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2        color[u] = white
 3        d[u] = ∞
 4        π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11        u = Dequeue(Q)
12        for each v ∈ Adj[u]
13              if color[v] == white
14                    color[v] = gray
15                    d[v] = d[u] + 1
16                    π[v] = u
17                    Enqueue(Q, v)
18        color[u] = black
```

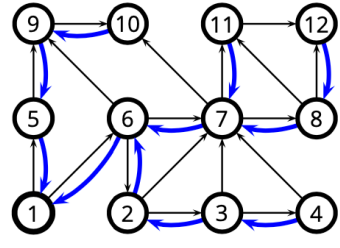


$u = 12$

$Q = \emptyset$

Algoritmo BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2        color[u] = white
 3        d[u] = ∞
 4        π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11        u = Dequeue(Q)
12        for each v ∈ Adj[u]
13              if color[v] == white
14                    color[v] = gray
15                    d[v] = d[u] + 1
16                    π[v] = u
17                    Enqueue(Q, v)
18        color[u] = black
```



Complejidad de BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2      color[u] = white
 3      d[u] = ∞
 4      π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11      u = Dequeue(Q)
12      for each v ∈ Adj[u]
13          if color[v] == white
14              color[v] = gray
15              d[v] = d[u] + 1
16              π[v] = u
17              Enqueue(Q, v)
18      color[u] = black
```

Complejidad de BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2      color[u] = white
 3      d[u] = ∞
 4      π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11      u = Dequeue(Q)
12      for each v ∈ Adj[u]
13          if color[v] == white
14              color[v] = gray
15              d[v] = d[u] + 1
16              π[v] = u
17              Enqueue(Q, v)
18      color[u] = black
```

- Un vértice se agrega a la cola (enqueue) solo si es blanco, se colorea de gris; cada vértice se agrega *a lo más una vez*

Complejidad de BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2      color[u] = white
 3      d[u] = ∞
 4      π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11      u = Dequeue(Q)
12      for each v ∈ Adj[u]
13          if color[v] == white
14              color[v] = gray
15              d[v] = d[u] + 1
16              π[v] = u
17              Enqueue(Q, v)
18      color[u] = black
```

- Un vértice se agrega a la cola (enqueue) solo si es blanco, se colorea de gris; cada vértice se agrega *a lo más una vez*
- El ciclo while se ejecuta $O(|V|)$ veces

Complejidad de BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2      color[u] = white
 3      d[u] = ∞
 4      π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11      u = Dequeue(Q)
12      for each v ∈ Adj[u]
13          if color[v] == white
14              color[v] = gray
15              d[v] = d[u] + 1
16              π[v] = u
17              Enqueue(Q, v)
18      color[u] = black
```

- Un vértice se agrega a la cola (enqueue) solo si es blanco, se colorea de gris; cada vértice se agrega *a lo más una vez*
- El ciclo while se ejecuta $O(|V|)$ veces
- Para cada vértice u, el ciclo externo se ejecuta $\Theta(|E_u|)$, para un total de $O(|E|)$ pasos

Complejidad de BFS

```
BFS(G, s)
 1  for each vertex u ∈ V(G) \ {s}
 2      color[u] = white
 3      d[u] = ∞
 4      π[u] = nil
 5  color[s] = gray
 6  d[s] = 0
 7  π[s] = nil
 8  Q = ∅
 9  Enqueue(Q, s)
10  while Q ≠ ∅
11      u = Dequeue(Q)
12      for each v ∈ Adj[u]
13          if color[v] == white
14              color[v] = gray
15              d[v] = d[u] + 1
16              π[v] = u
17              Enqueue(Q, v)
18      color[u] = black
```

- Un vértice se agrega a la cola (enqueue) solo si es blanco, se colorea de gris; cada vértice se agrega *a lo más una vez*
- El ciclo while se ejecuta $O(|V|)$ veces
- Para cada vértice u, el ciclo externo se ejecuta $\Theta(|E_u|)$, para un total de $O(|E|)$ pasos
- La complejidad total es $O(|V| + |E|)$

Búsqueda primero por profundidad (Depth-First Search)

Búsqueda primero por profundidad (Depth-First Search)

- Se siguen los enlaces de los nodos visitados más recientemente, se retroceda cuando llegue a un callejón sin salida)
  - e.i., backtrack cuándo el vértice actual no tiene más vértices adyacentes para visitar (ya no hay más nodos que colorear).

Búsqueda primero por profundidad (Depth-First Search)

- Se siguen los enlaces de los nodos visitados más recientemente, se retroceda cuando llegue a un callejón sin salida)
  - e.i., backtrack cuándo el vértice actual no tiene más vértices adyacentes para visitar (ya no hay más nodos que colorear).

- *Entrada:* $G = (V, E)$
  - recorre el grafo, visitando *todos los vértices*

Búsqueda primero por profundidad (Depth-First Search)

- Se siguen los enlaces de los nodos visitados más recientemente, se retroceda cuando llegue a un callejón sin salida)
  - e.i., backtrack cuándo el vértice actual no tiene más vértices adyacentes para visitar (ya no hay más nodos que colorear).

- *Entrada:* $G = (V, E)$
  - recorre el grafo, visitando *todos los vértices*
  - genera un ***depth-first forest***, este consiste de todos todos los ***depth-first trees*** definido por DFS

Búsqueda primero por profundidad (Depth-First Search)

- Se siguen los enlaces de los nodos visitados más recientemente, se retroceda cuando llegue a un callejón sin salida)
  - e.i., backtrack cuándo el vértice actual no tiene más vértices adyacentes para visitar (ya no hay más nodos que colorear).

- *Entrada:* $G = (V, E)$
  - recorre el grafo, visitando *todos los vértices*
  - genera un ***depth-first forest***, este consiste de todos todos los ***depth-first trees*** definido por DFS
  - asociar ***dos marcas de tiempo*** a cada vértice
    - $d[u]$ indica cuando u es descubierto
    - $f[u]$ indica cuando DFS terminda de examinar todas las aristas de u, entonces hace backtrack desde u

Algoritmo DFS

```
DFS(G)
1  for each vertex u ∈ V(G)
2      color[u] = white
3      π[u] = nil
4  time = 0 // "global" variable
5  for each vertex u ∈ V(G)
6      if color[u] == white
7          DFS-Visit(u)
```

```
DFS-Visit(u)
1  color[u] = grey
2  time = time + 1
3  d[u] = time
4  for each v ∈ Adj[u]
5      if color[v] == white
6          π[v] = u
7          DFS-Visit(v)
8  color[u] = black
9  time = time + 1
10 f[u] = time
```

Complejidad DFS

Complejidad DFS

- El ciclo en **DFS-Visit**(u) (lineas 4–7) se ejecuta $\Theta(|E_u|)$

Complejidad DFS

- El ciclo en **DFS-Visit**(u) (lineas 4–7) se ejecuta $\Theta(|E_u|)$
- Se llama a **DFS-Visit**(u) *una vez* por cada vértice u
  - ya sea en **DFS**, o recursivamente desde **DFS-Visit**
  - se ejecuta solo sí $color[u] =$ white, pero inmediatamente se realiza la asignación $color[u] =$ grey

Complejidad DFS

- El ciclo en **DFS-Visit**(u) (lineas 4–7) se ejecuta $\Theta(|E_u|)$
- Se llama a **DFS-Visit**(u) *una vez* por cada vértice u
  - ya sea en **DFS**, o recursivamente desde **DFS-Visit**
  - se ejecuta solo sí $color[u] =$ white, pero inmediatamente se realiza la asignación $color[u] =$ grey
- Así, al complejidad total está dada por $\Theta(|V| + |E|)$

Aplicación de DFS: Ordenamiento topológico

- **Problema:** (ordenamiento topológico)

  Dado un *grafo no dirigido acíclico* (DAG)

  - encontrar un ordenamiento de los vértices tal que solo se sigan *aristas hacia adelante*

Aplicación de DFS: Ordenamiento topológico

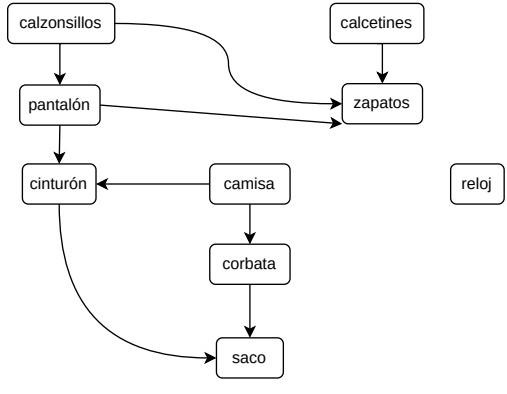
- **Problema:** (ordenamiento topológico)

  Dado un *grafo no dirigido acíclico* (DAG)

  - encontrar un ordenamiento de los vértices tal que solo se sigan *aristas hacia adelante*

- **Ejemplo:** dependencias de paquetes de software
  - encontrar el orden de instalación de un conjunto de paquetes de software
  - de forma que cada paquete es instalado después de todos los paquetque de los cuales depende.

Algoritmo para el ordenamiento topológico



Algoritmo para el ordenamiento topológico

Topological-Sort(G)

1 **DFS**(G)
2 regresa V ordenado de forma invertida con respecto del tiempo de terminación $f[\cdot]$