RECORRIDOS BÁSICOS EN GRAFOS

Programación 3
Javier Miranda

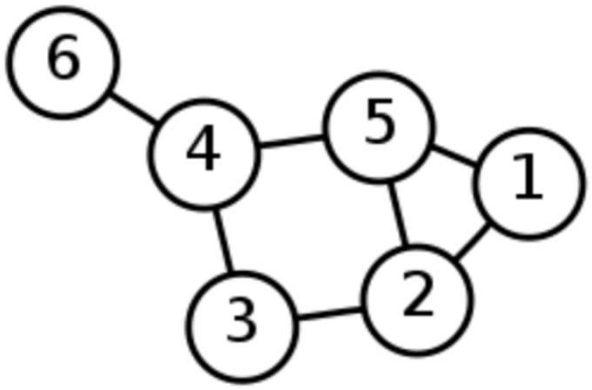
Escuela de Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

Contenido

- Estrategias de búsqueda en un grafo
  - A lo ancho
    - Aplicaciones
  - En profundidad
    - Aplicaciones
- Resumen

Búsqueda en un Grafo

- **Objetivo genérico**: Partiendo de un determinado vértice del grafo buscar la "mejor" ruta para llegar a otro vértice (o al resto de los vertices).



Algoritmo genérico

Marcar el vértice origen como visitado

mientras queden vértices por procesar.
- Elegir una arista (U,V) con U visitado y V no visitado
- Procesar esta ruta
- Marcar el vértice V como visitado

Es un algoritmo eficiente ya que evita procesar los vértices varias veces

Estrategias básicas de búsqueda

Recorrido a lo ancho

- *Bread-First Search (BFS)*
- Exploración por niveles
- Se programa con una cola

Aplicaciones:

- Camino más corto
- Componentes conectados

Recorrido en profundidad

- *Depth-First Search (DFS)*
- Exploración agresiva
- Se programa con una pila (o con recursividad).

Aplicaciones:

- Orden topológico
- Componentes conectados

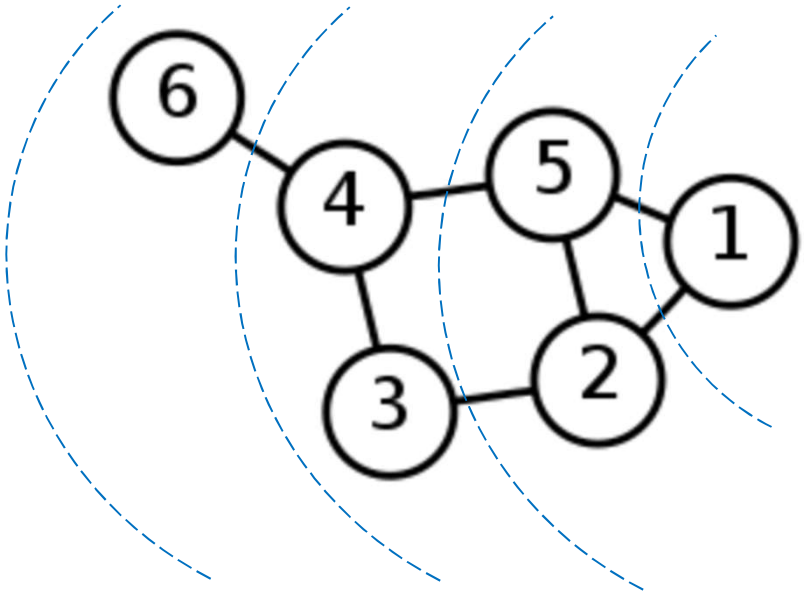
https://www.youtube.com/watch?v=x-VTfcmrLEQ

https://www.youtube.com/watch?v=NUgMa5coCoE

Estrategias básicas de búsqueda

Recorrido a lo ancho

- *Bread-First Search (BFS)*
- Exploración por niveles
- Se programa con una cola



Aplicaciones:

- Camino más corto
- Componentes conectados

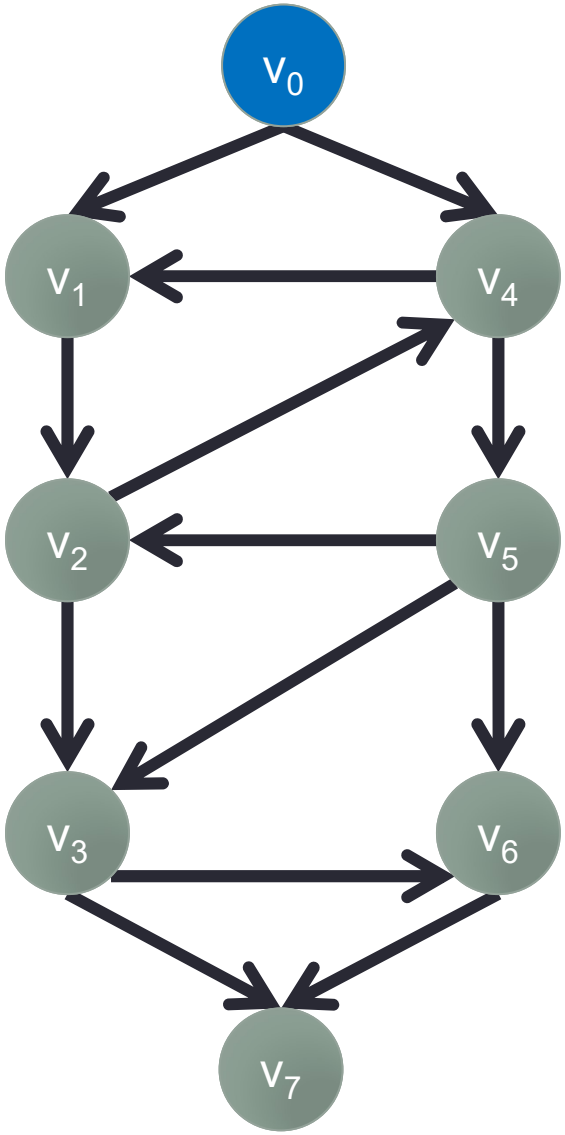
Recorrido a lo ancho (Cola)

```
class Queue:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def enqueue(self, item):
        self.items.insert(0,item)

    def dequeue(self):
        return self.items.pop()

    def size(self):
        return len(self.items)
```

```
obj = Queue()
obj.enqueue(1)
obj.enqueue(2)
obj.enqueue(3)


print(obj.dequeue())  # 1
print(obj.dequeue())  # 2
print(obj.dequeue())  # 3
print(obj.isEmpty())  # True
```

FIFO – First In First Out

Recorrido a lo ancho (*BFS*)



Marcamos el vértice origen como visible
… y lo añadimos a la cola

Cola (vértices pendientes de procesar)

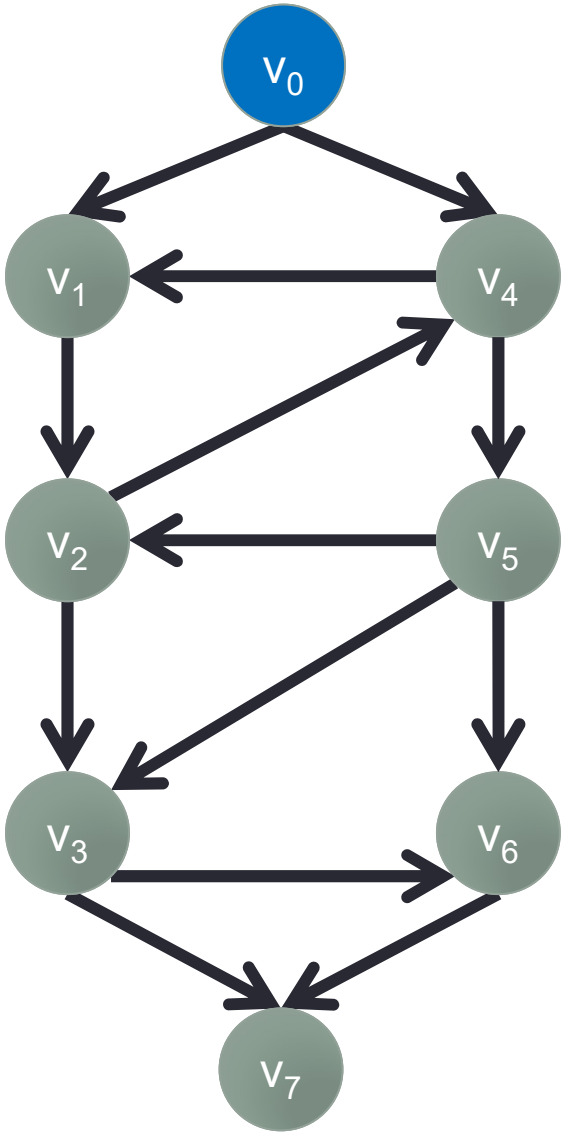
v_0							

Vértices visibles

v_0							

Recorrido a lo ancho (*BFS*)

Paso 1



v_0

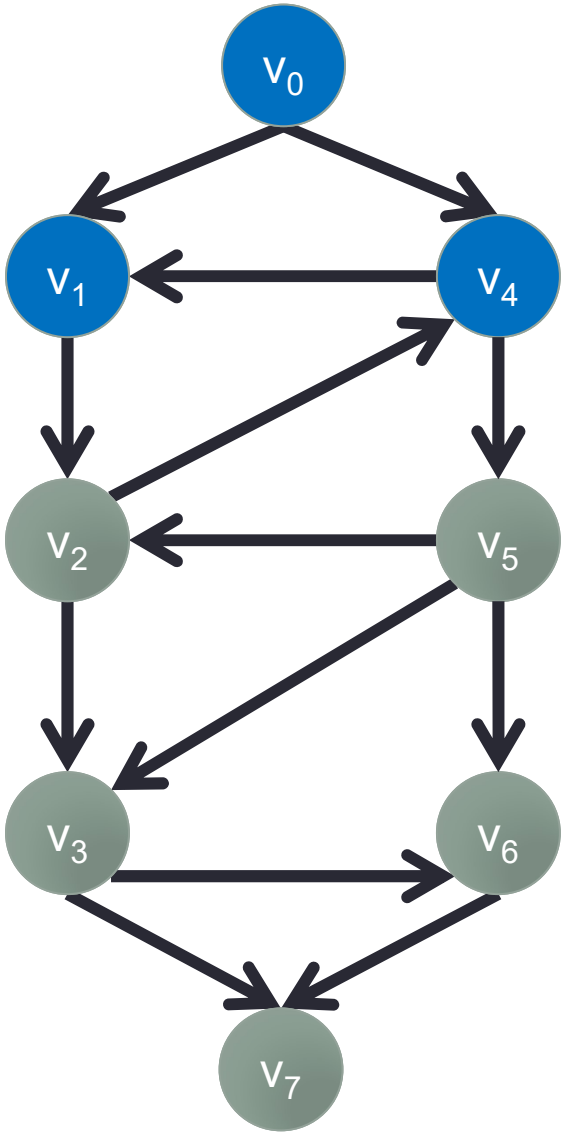
Sacamos un elemento de la cola

Cola (vértices pendientes de procesar)

Vértices visibles

v_0							

Recorrido a lo ancho (*BFS*)



V_0

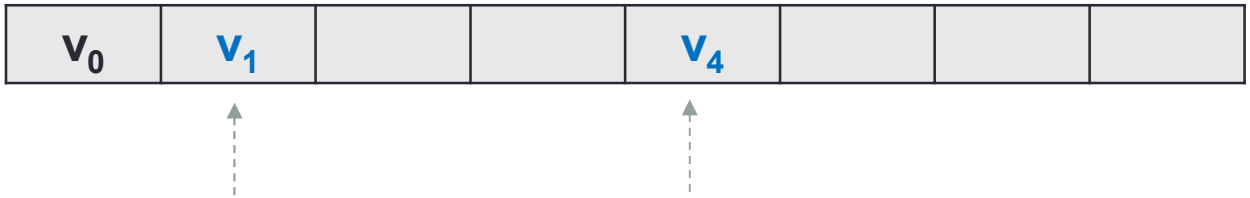
Sacamos un elemento de la cola

… marcamos sus vecinos como visibles:

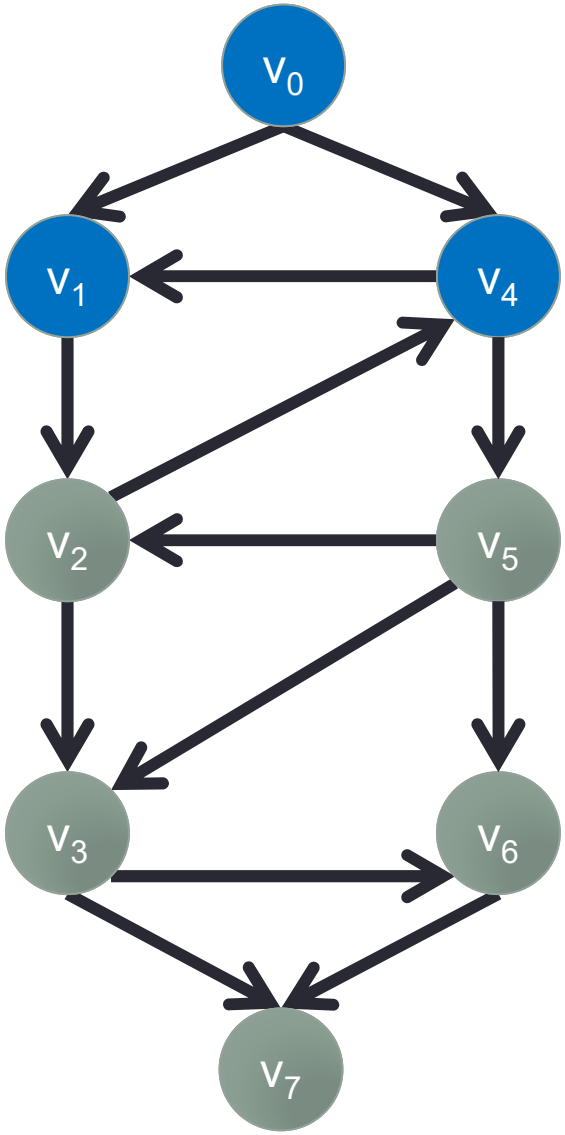
V_1 y V_4

Cola (vértices pendientes de procesar)

Vértices visibles



Recorrido a lo ancho (*BFS*)



V_0

Sacamos un elemento de la cola

… marcamos sus vecinos como visibles:

V_1 y V_4

… y los metemos en la cola para procesarlos

Cola (vértices pendientes de procesar)

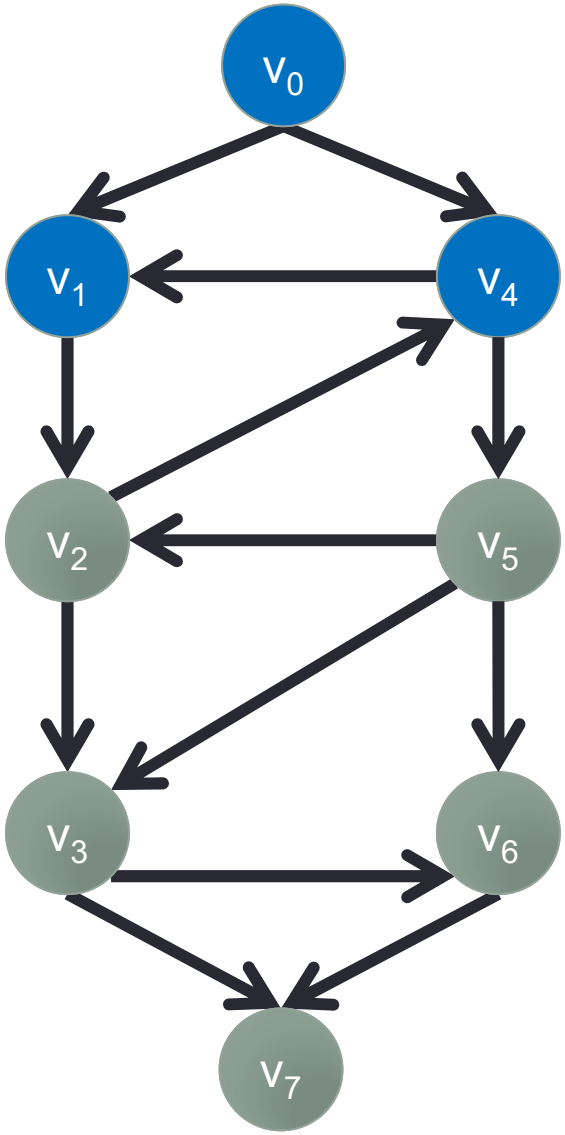
v_1	v_4					

Vértices visibles

v_0	v_1			v_4			

Recorrido a lo ancho (*BFS*)

Paso 1



V_0 V_1

Sacamos un elemento de la cola:

Cola (vértices pendientes de procesar)

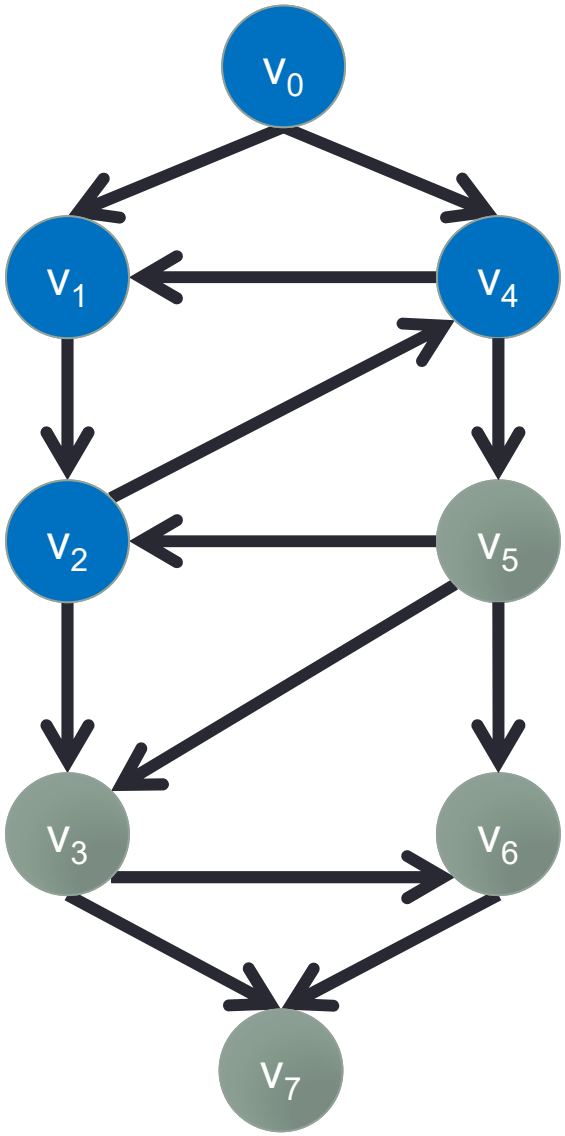
v_4					

Vértices visibles

v_0	v_1			v_4			

Recorrido a lo ancho (*BFS*)

Paso 2



V_0 V_1

Sacamos un elemento de la cola

… marcamos sus vecinos como visibles:

V_2

Cola (vértices pendientes de procesar)

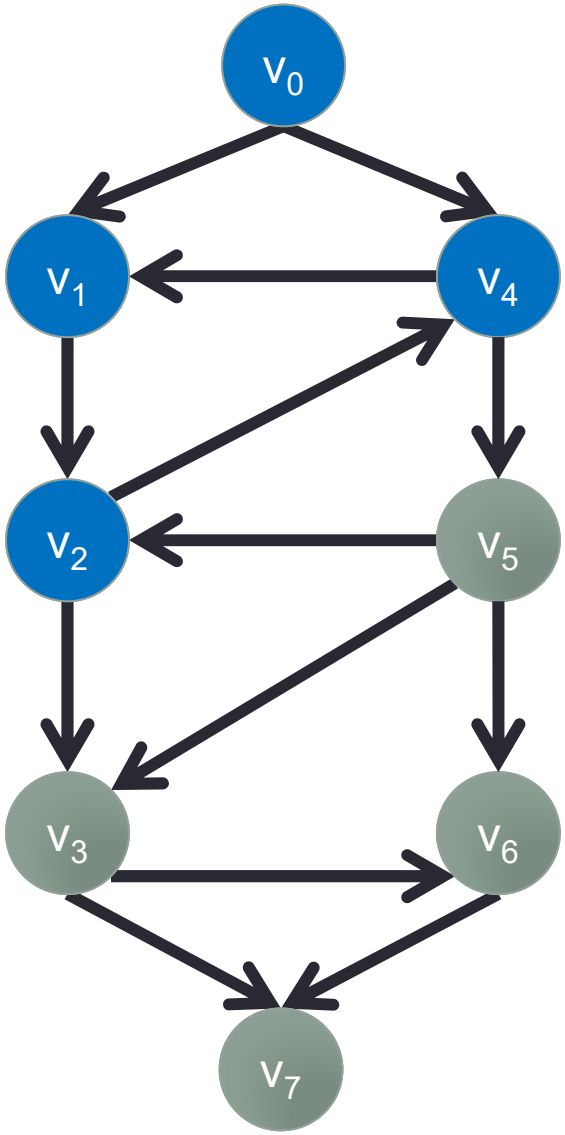
v_4					

Vértices visibles

v_0	v_1	v_2		v_4			

Recorrido a lo ancho (*BFS*)

Paso 3



V_0 V_1

Sacamos un elemento de la cola

… marcamos sus vecinos como visibles:

V_2

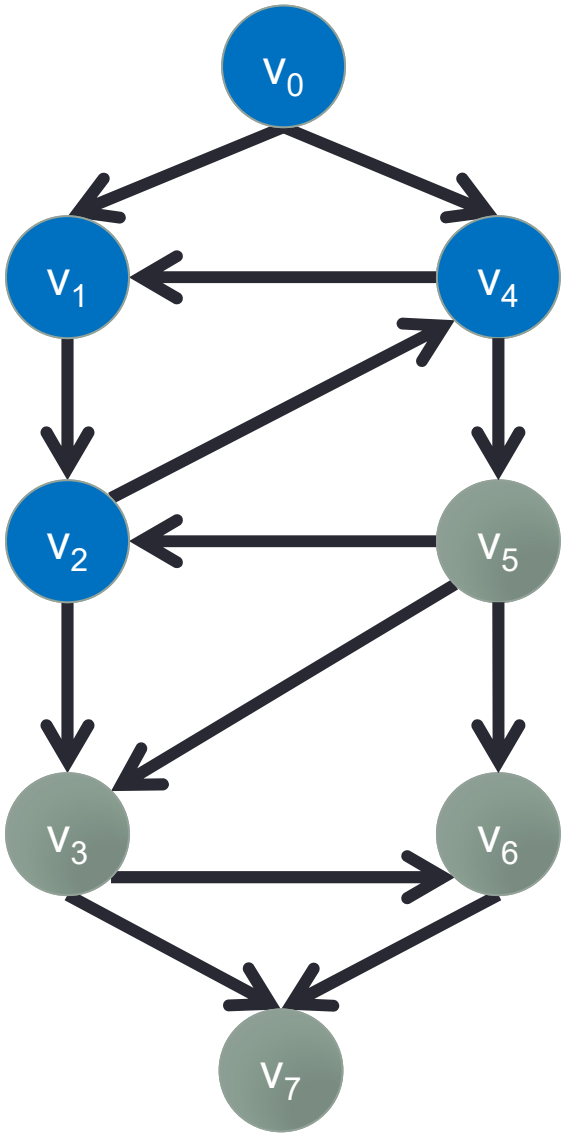
… y los metemos en la cola para procesarlos

Cola (vértices pendientes de procesar)

Vértices visibles

Recorrido a lo ancho (*BFS*)

Paso 1



V_0 V_1 V_4

Sacamos un elemento de la cola

Cola (vértices pendientes de procesar)

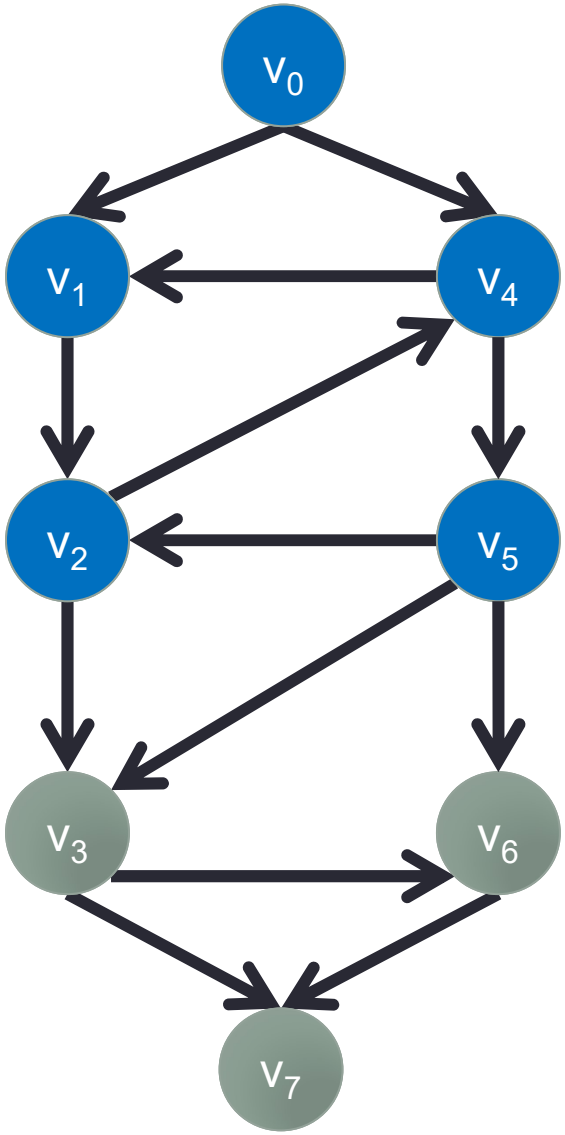
v_2				

Vértices visibles

v_0	v_1	v_2		v_4			

Recorrido a lo ancho (*BFS*)

Paso 2



V_0 V_1 V_4

Sacamos un elemento de la cola

… marcamos sus vecinos como visibles:

V_5

Cola (vértices pendientes de procesar)

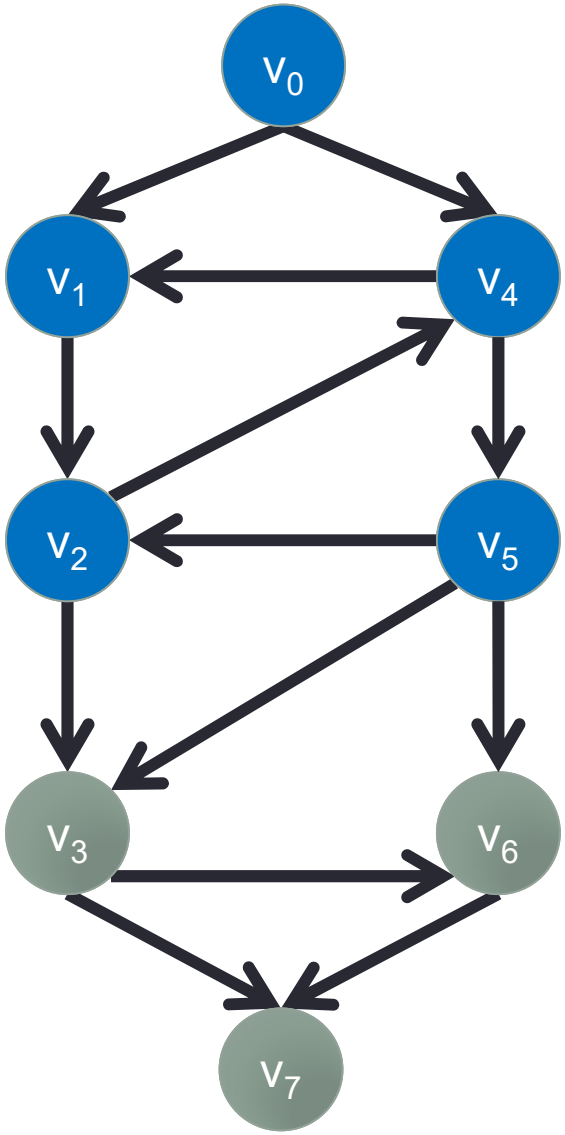
v_2				

Vértices visibles

v_0	v_1	v_2		v_4	v_5		

Recorrido a lo ancho (*BFS*)

Paso 3



V_0 V_1 V_4

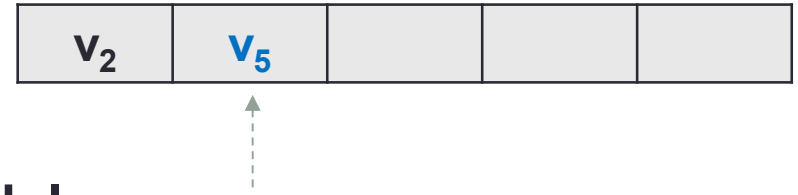
Sacamos un elemento de la cola

… marcamos sus vecinos como visibles:

V_5

… y los metemos en la cola para procesarlos

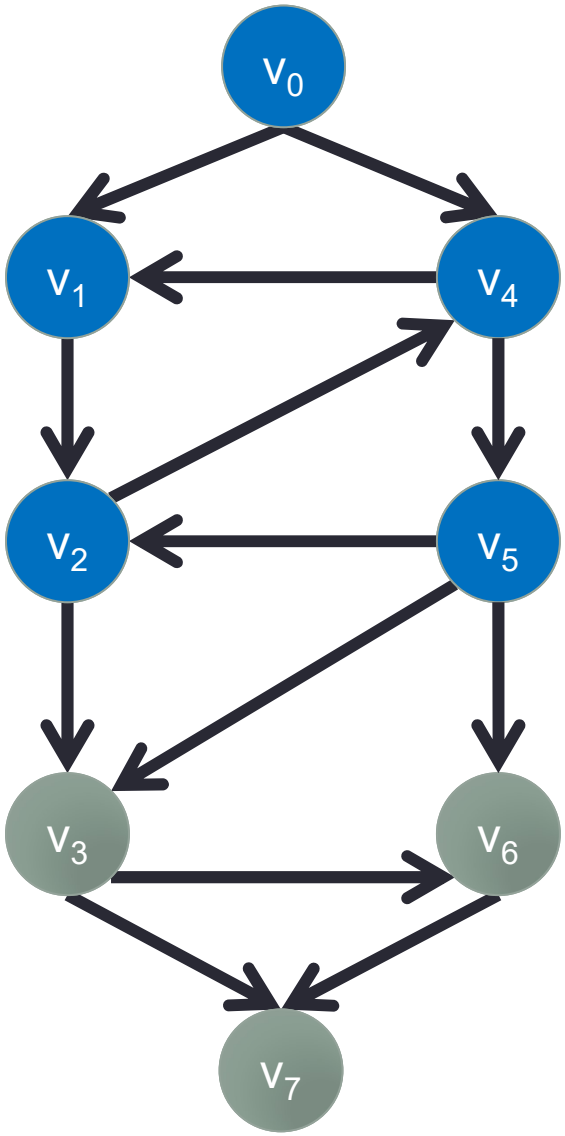
Cola (vértices pendientes de procesar)



Vértices visibles

Recorrido a lo ancho (*BFS*)

Paso 1



V_0 V_1 V_4 V_2

Sacamos un elemento de la cola

Cola (vértices pendientes de procesar)

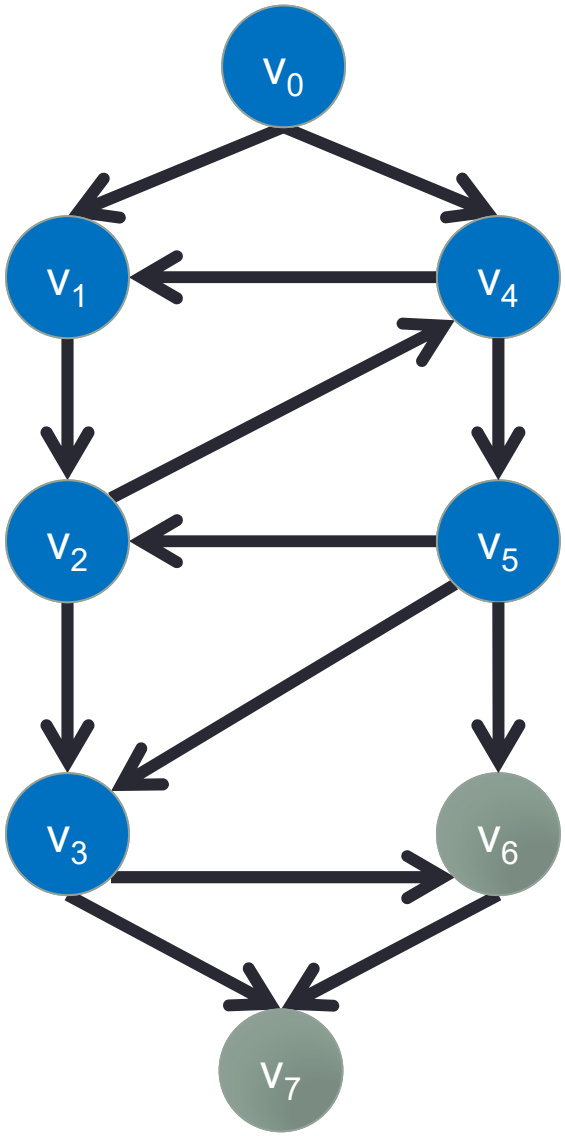
v_5			

Vértices visibles

v_0	v_1	v_2		v_4			

Recorrido a lo ancho (*BFS*)

Paso 2



V_0 V_1 V_4 V_2

Sacamos un elemento de la cola

… marcamos sus vecinos como visibles:

V_3

Cola (vértices pendientes de procesar)

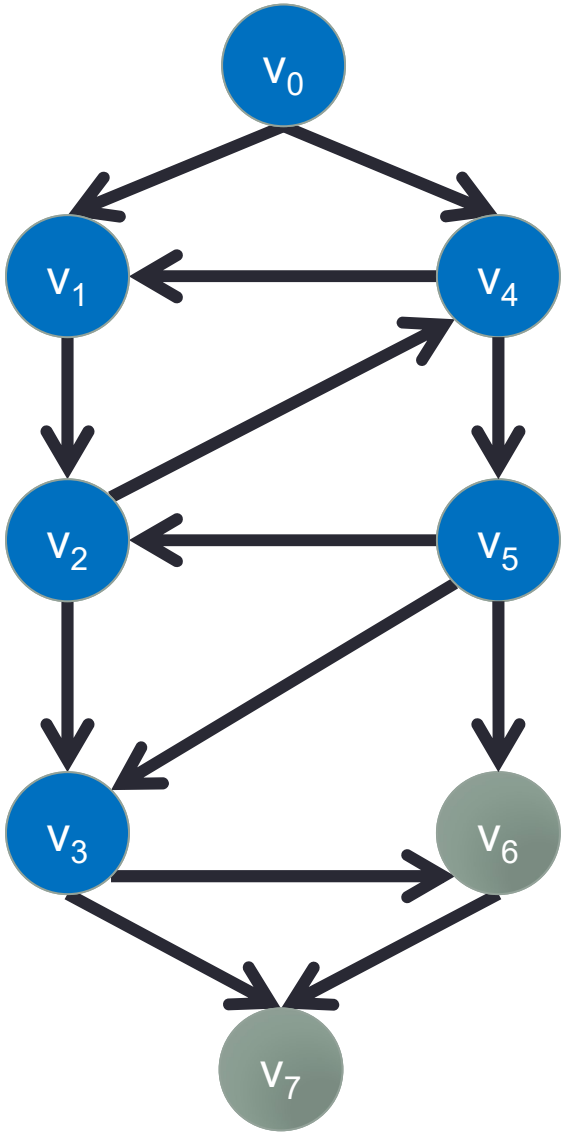
v_5			

Vértices visibles

v_0	v_1	v_2	v_3	v_4			

Recorrido a lo ancho (*BFS*)

Paso 3



V_0 V_1 V_4 V_2

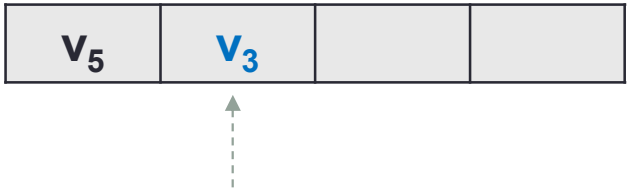
Sacamos un elemento de la cola

… marcamos sus vecinos como visibbles:

V_3

… y los metemos en la cola para procesarlos

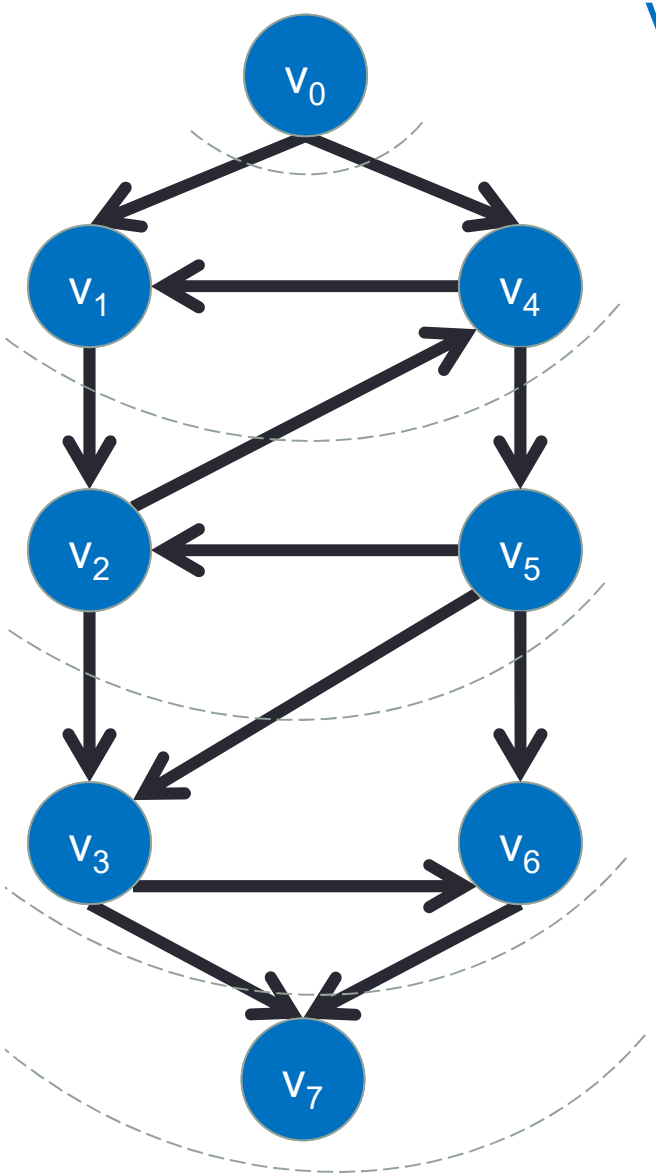
Cola (vértices pendientes de procesar)



Vértices visibles

Recorrido a lo ancho (*BFS*)

V_0 V_1 V_4 V_2 V_5 V_3 V_6 V_7



… complentando el algoritmo éste es el recorrido obtenido

Recorrido a lo ancho (BFS)

BFS (grafo G, vertice inicial vi)

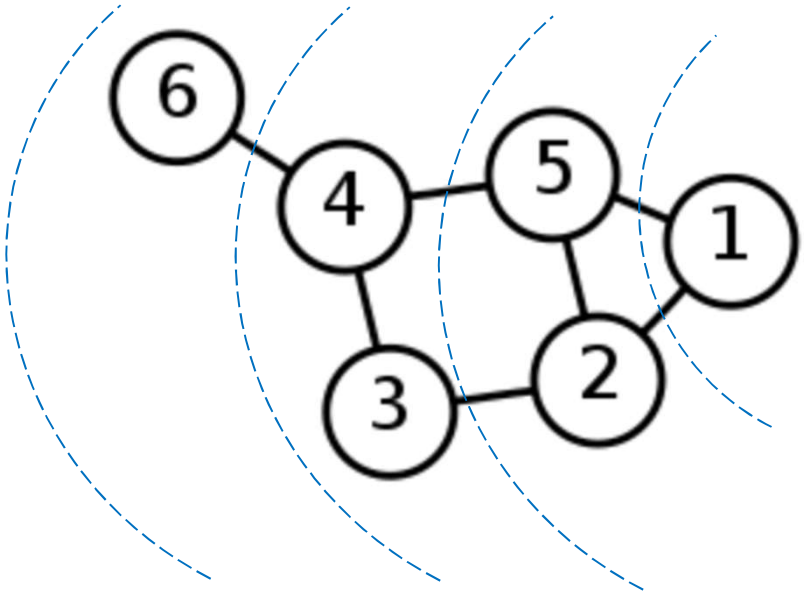
- Q = Queue()
- Marcar vi como visible;
- Q.enqueue(vi)

- Mientras not Q.is_empty():
  - v = Q.dequeue()
  - Para todas las aristas (v,w):
    - Si w no es visible:
      - Marcar w como visible
      - Q.enqueue(w)

Estrategias básicas de búsqueda

Recorrido a lo ancho

- *Bread-First Search (BFS)*
- Exploración por niveles
- Se programa con una cola



Aplicaciones:

- Camino más corto
- Componentes conectados

Aplicación 1: Camino más corto

- ***Calcular la distancia minima (en número de saltos) desde un determinado vértice a todos los demás***

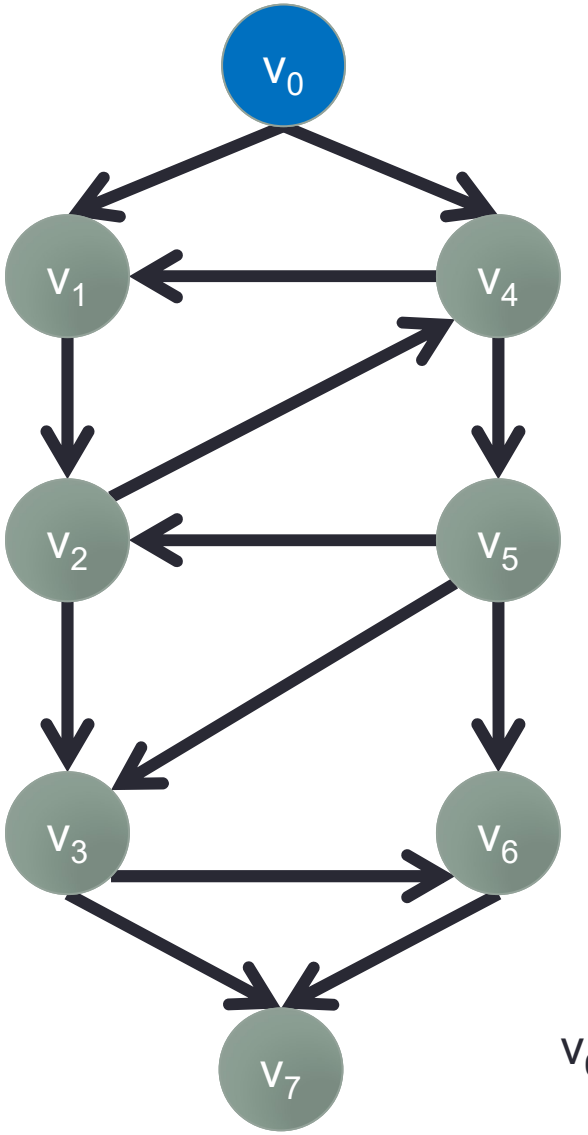
Shortest_Distance (grafo G, vertice inicial vi)

- Q = Queue();
- Marcar vi como visitado; Q.enqueue(vi)
- Dist = [0 si k = vi; infinito si k != vi]

- Mientras not Q.is_empty():
  - v = Q.dequeue ()

  - Para todas las aristas (v,w):
    - Si w no es visible:
      - Marcar w como visitado
      - Dist (w) = Dist(v) + 1
      - Q.enqueue(w)

Camino más corto (*BFS*)

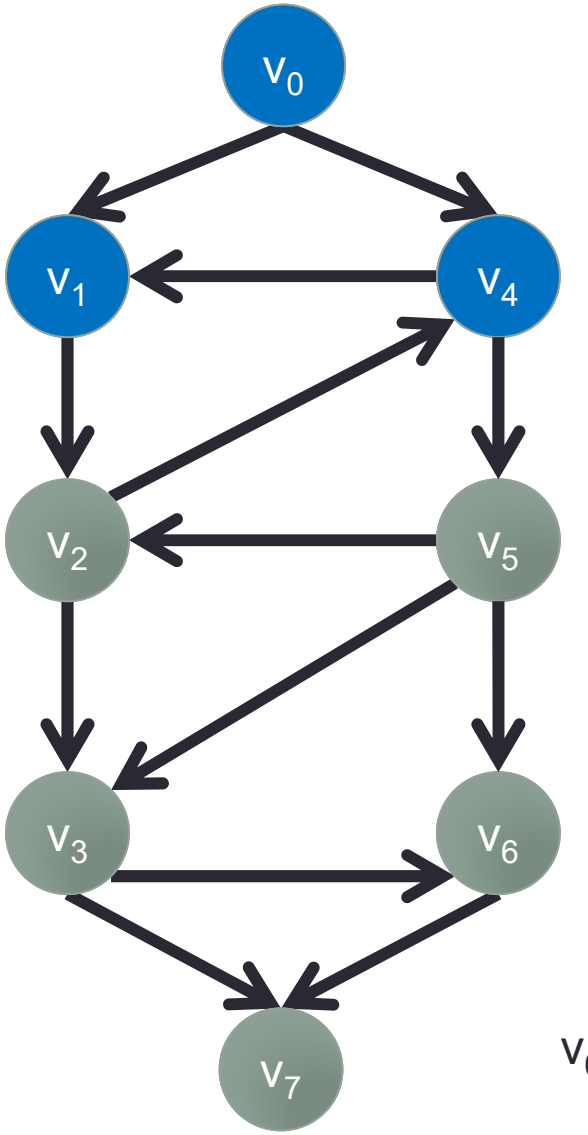


Marcamos el vértice origen como visible
… y lo añadimos a la cola

Distancia

v_0	0	inf	inf	inf	inf	inf	inf	inf
	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7

Camino más corto (*BFS*)



Sacamos un elemento de la cola: V_0

… marcamos sus vecinos visibles:

V_1 y V_4

… y actualizamos dist(V_1), dist(V_4)

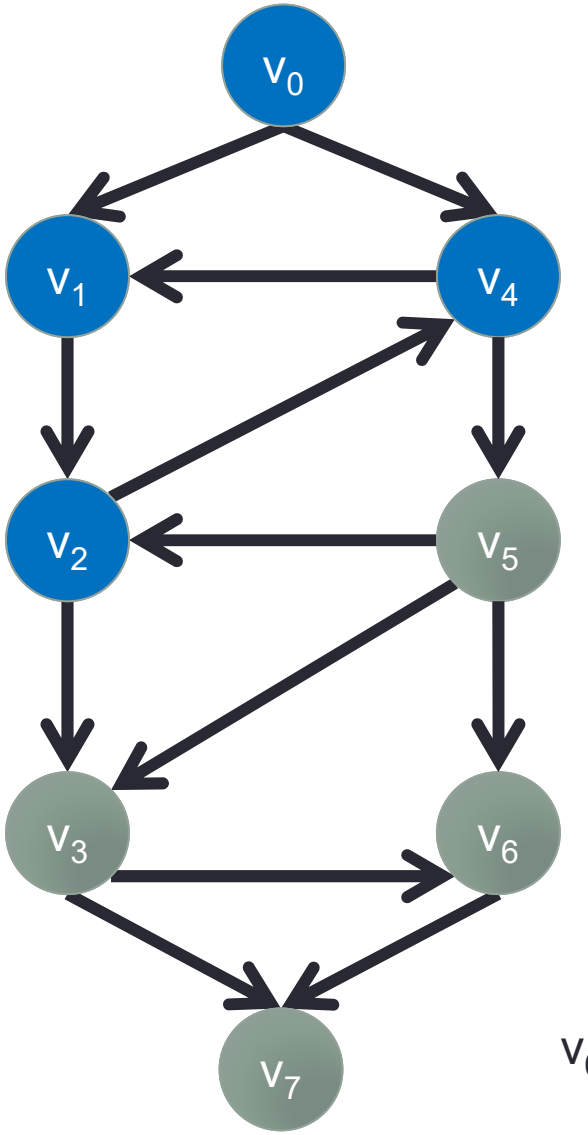
dist(V_1) = dist(V_0) + 1

dist(V_4) = dist(V_0) + 1

Distancia

v_0	0	1	inf	inf	1	inf	inf	inf
	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7

Camino más corto (*BFS*)



Sacamos un elemento de la cola: V_1

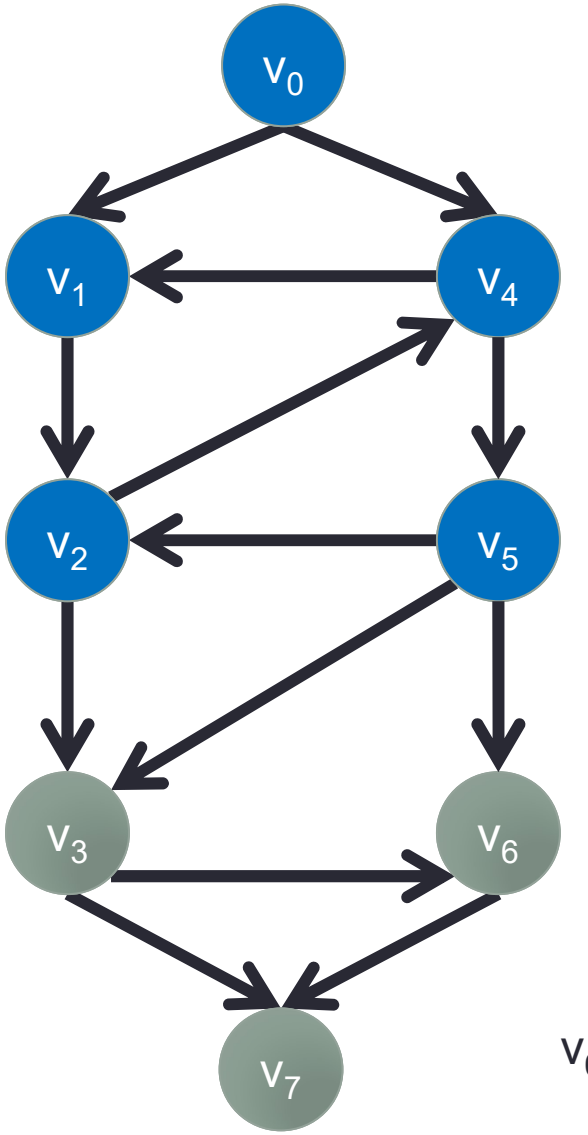
… marcamos sus vecinos como visibles:
V_2

… y actualizamos dist(V_2)
dist(V_2) = dist(V_1) + 1

Distancia

v_0	0	1	2	inf	1	inf	inf	inf
	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7

Camino más corto (*BFS*)



Sacamos un elemento de la cola: V_4

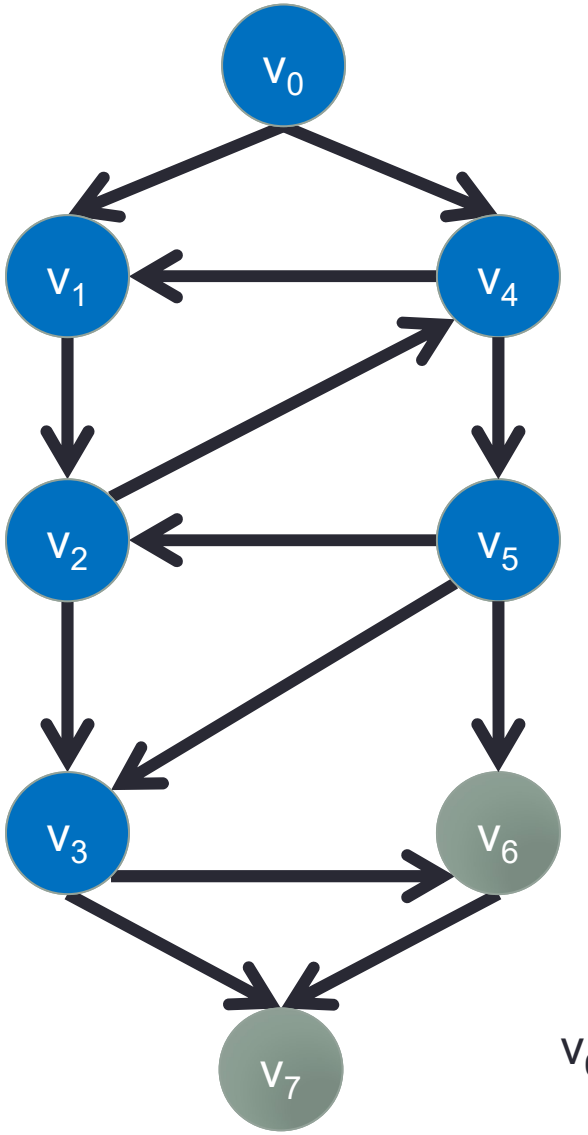
… marcamos sus vecinos como visibles:
V_5

… y actualizamos dist(V_5)
dist(V_5) = dist(V_4) + 1

Distancia

v_0	0	1	2	inf	1	2	inf	inf
	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7

Camino más corto (*BFS*)



Sacamos un elemento de la cola: V_2

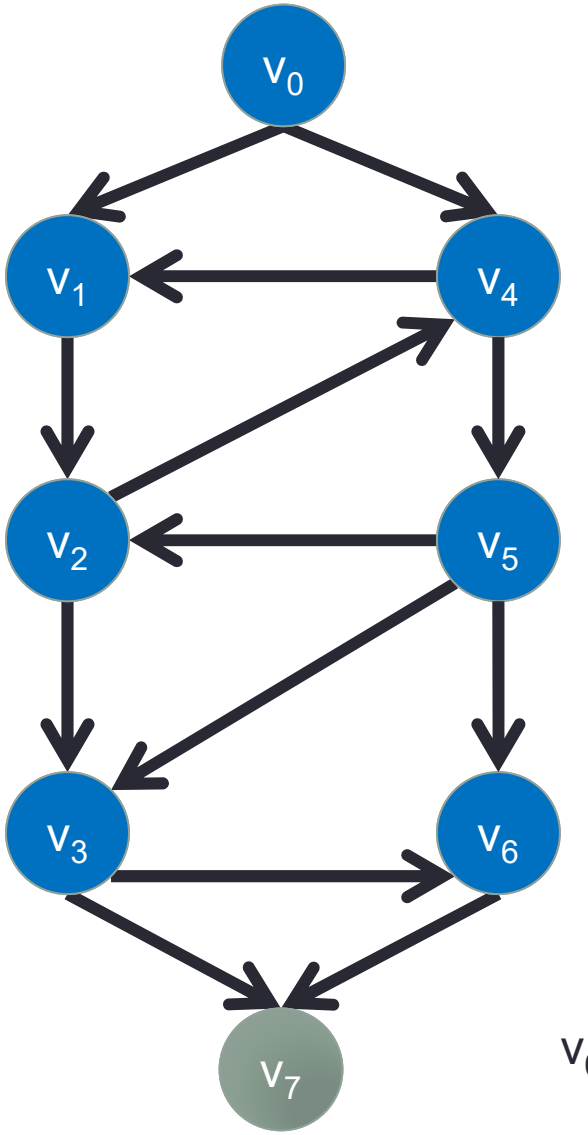
… marcamos sus vecinos como visibles:
V_3

… y actualizamos dist(V_3)
dist(V_3) = dist(V_2) + 1

Distancia

v_0	0	1	2	3	1	2	inf	inf
	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7

Camino más corto (*BFS*)



Sacamos un elemento de la cola: V_5

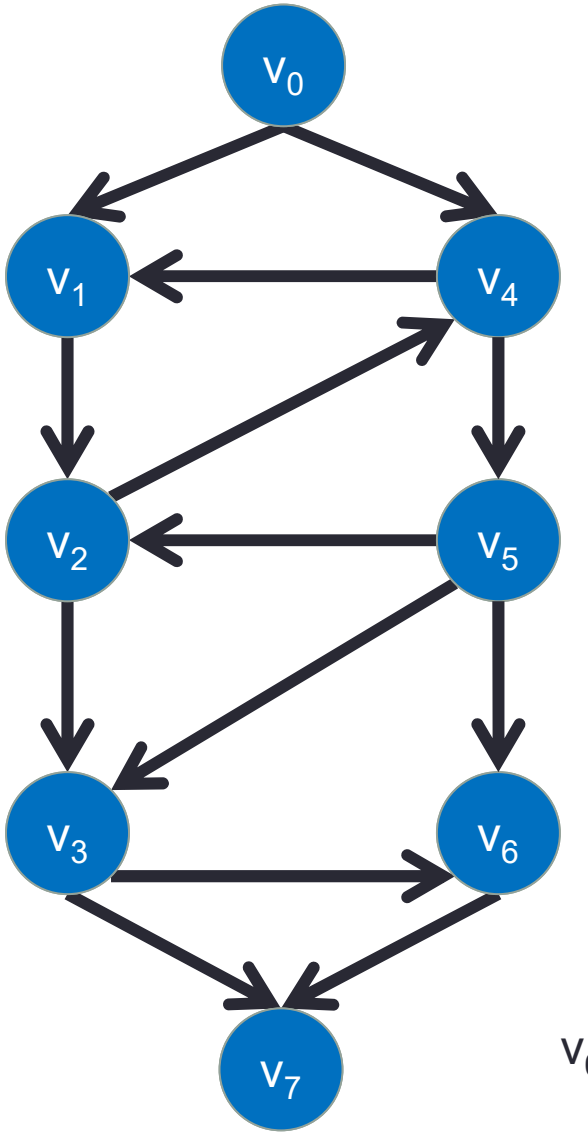
… marcamos sus vecinos como visibles:
V_6

… y actualizamos dist(V_6)
dist(V_6) = dist(V_5) + 1

Distancia

v_0	0	1	2	3	1	2	3	inf
	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7

Camino más corto (*BFS*)



Sacamos un elemento de la cola: V_3

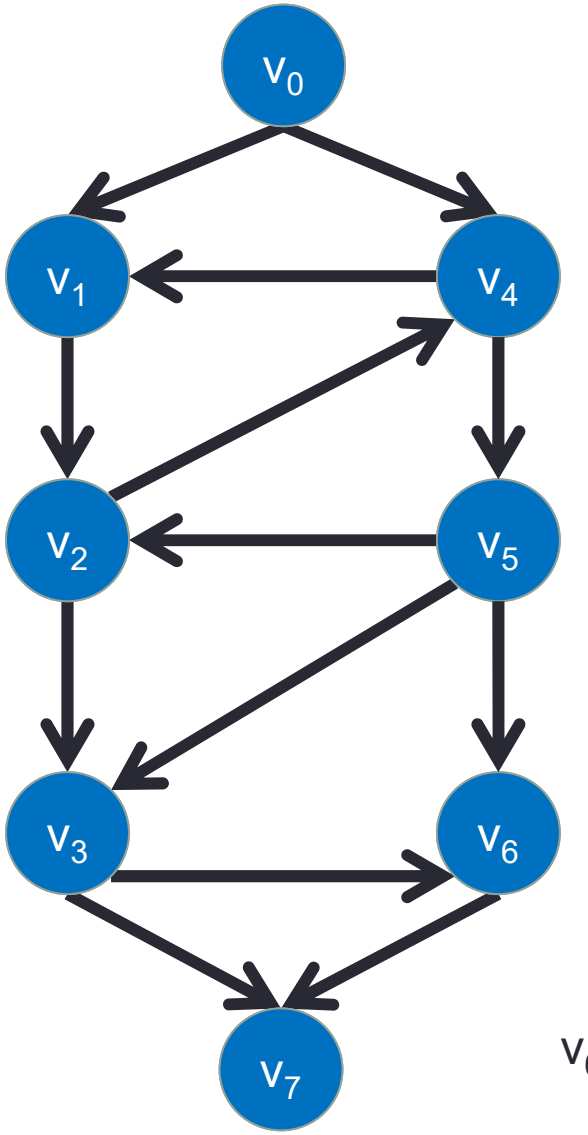
… marcamos sus vecinos como visibles:
V_7

… y actualizamos dist(V_7)
dist(V_7) = dist(V_3) + 1

Distancia

v_0	0	1	2	3	1	2	3	4
	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7

Camino más corto (*BFS*)

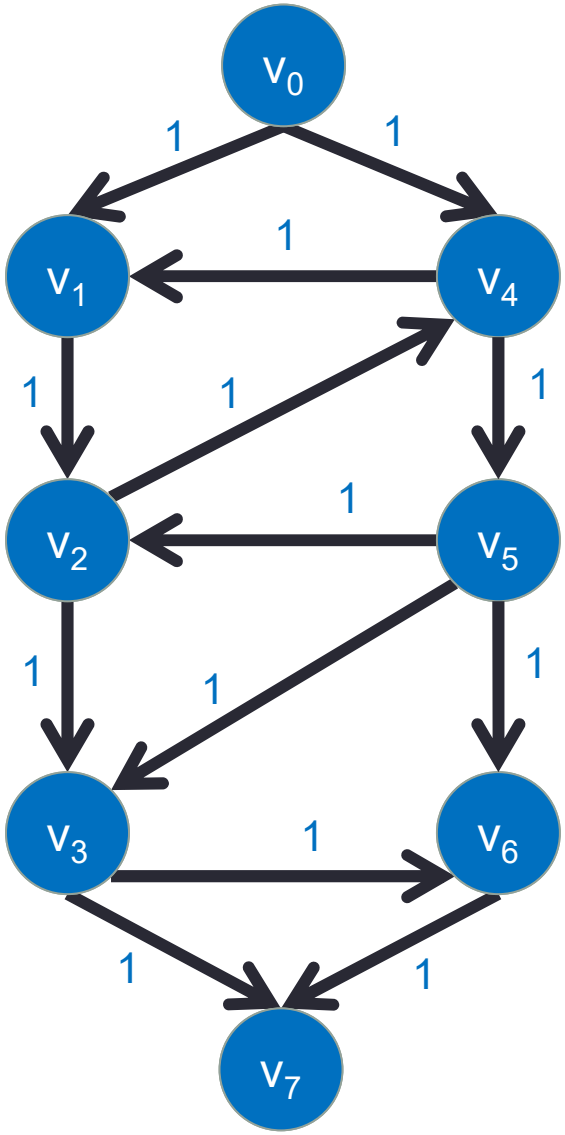


… complentando el algoritmo estas son las distancias calculadas

Distancia

v_0	0	1	2	3	1	2	3	4
	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7

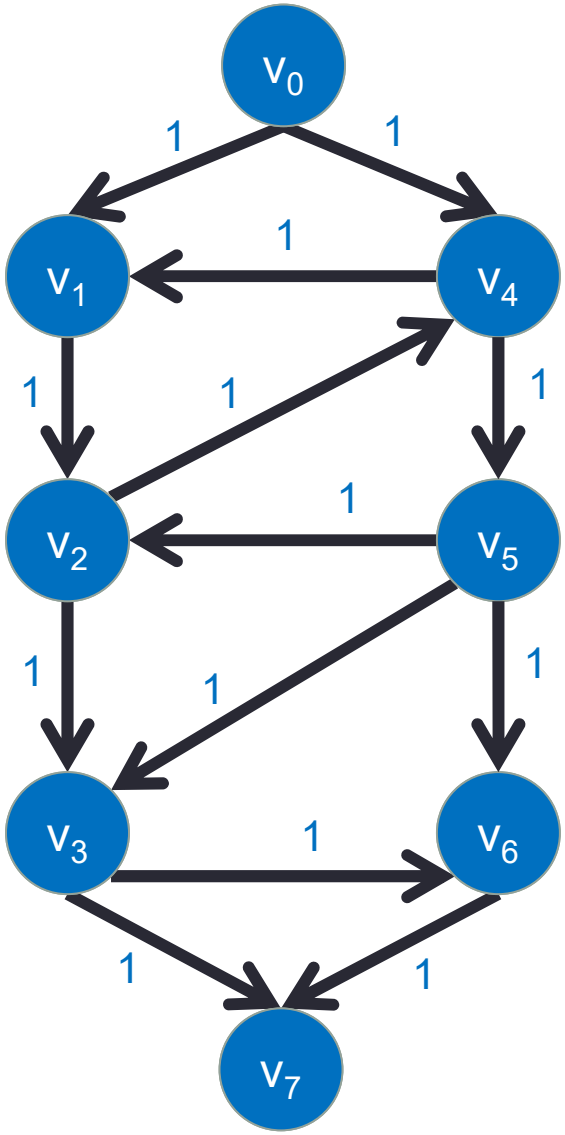
Camino más corto (*BFS*)



Limitación

Sólo funciona bien cuando todas las aristas tienen peso 1

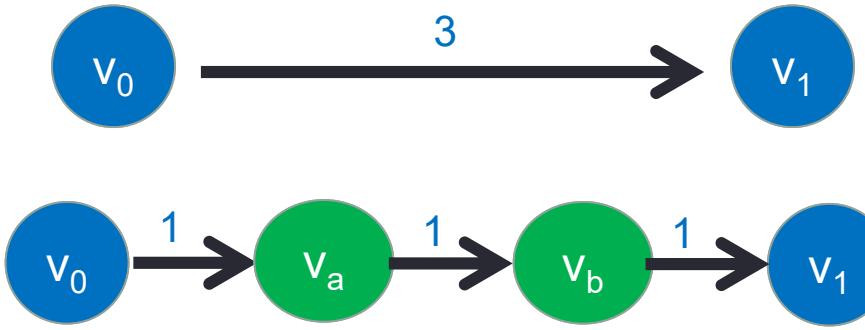
Camino más corto (*BFS*)



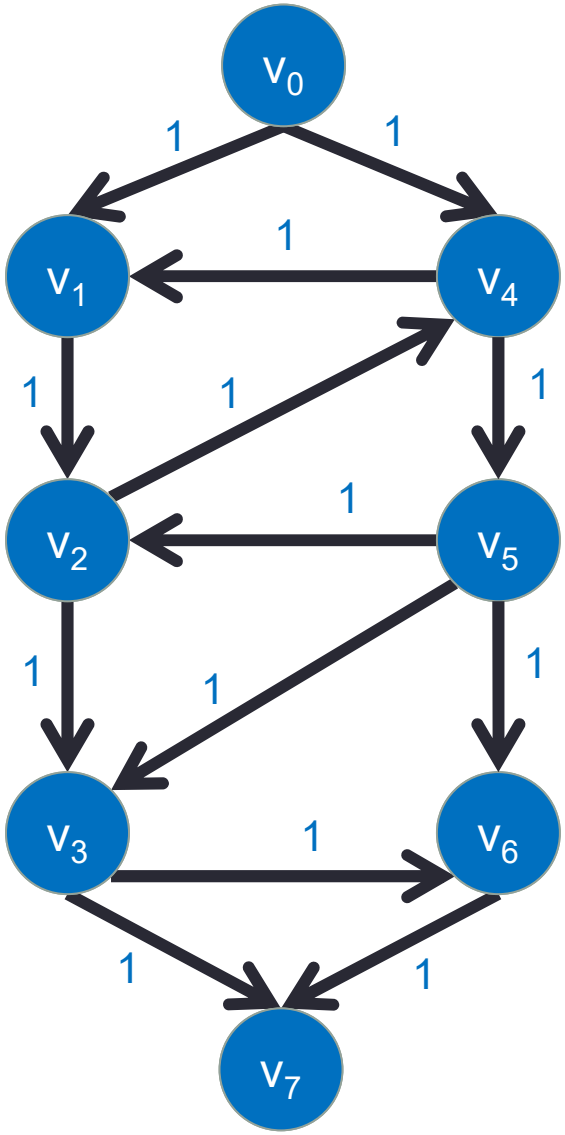
Limitación

Sólo funciona bien cuando todas las aristas tienen peso 1

¿ Podríamos sustituir pesos mayores por caminos de nodos con peso 1 ? (*reducción*)



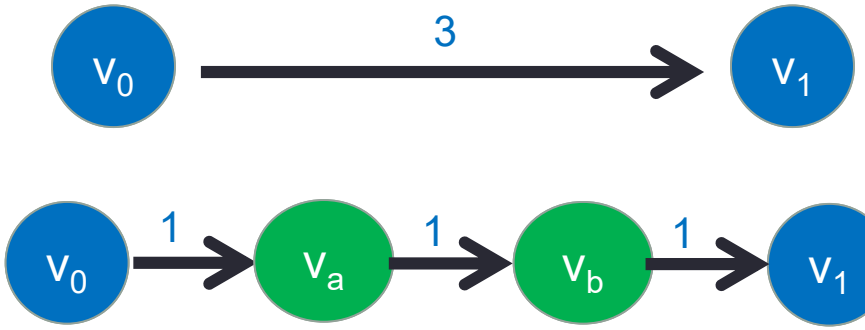
Camino más corto (*BFS*)



Limitación

Sólo funciona bien cuando todas las aristas tienen peso 1

¿ Podríamos sustituir pesos mayores por caminos de nodos con peso 1 ? (*reducción*)



¿ Hay una solución mejor ?

Algoritmo Dijskstra ruta más corta

Teniendo un grafo dirigido ponderado de N nodos no aislados, sea x el nodo inicial. Un vector D de tamaño N guardará al final del algoritmo las distancias desde x hasta el resto de los nodos.

1) Inicializar todas las distancias en D con un valor infinito relativo, ya que son desconocidas al principio, exceptuando la de x, que se debe colocar en 0, debido a que la distancia de x a x sería 0.

2) Sea a = x (Se toma a como nodo actual.)

3) Se recorren todos los nodos adyacentes de a, excepto los nodos marcados. Se les llamará nodos no marcados V_i.

4) Para el nodo actual, se calcula la distancia tentativa desde dicho nodo hasta sus vecinos con la siguiente fórmula: $dt(V_i) = Da + d(a,V_i)$. Es decir, la distancia tentativa del nodo 'V_i' es la distancia que actualmente tiene el nodo en el vector D más la distancia desde dicho nodo 'a' (el actual) hasta el nodo vi. Si la distancia tentativa es menor que la distancia almacenada en el vector, entonces se actualiza el vector con esta distancia tentativa. Es decir, si dt(vi) < Dvi → Dvi = dt(vi)

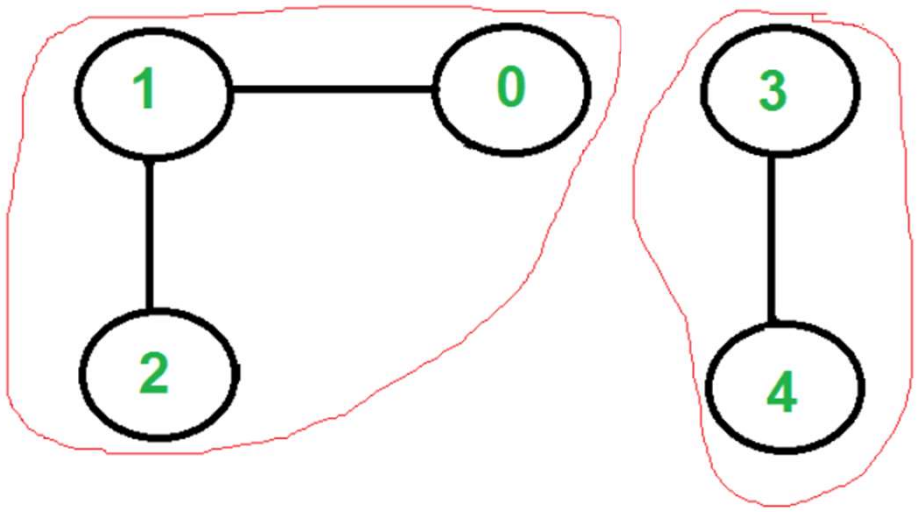
5) Se marca como completo el nodo a.

6) Se toma como próximo nodo actual el de menor valor en D (puede hacerse almacenando los valores en una cola de prioridad) y se regresa al paso 3, mientras existan nodos no marcados.

Una vez terminado al algoritmo, D estará completamente lleno.

Aplicación 2: Conectividad

- ***Componentes conectados de un grafo no dirigido***



- Marcamos todos los vertices como no-visibles

- Recorremos todos los vertices J
  - Si J no es visible:
    - **BFS** (G, J)

Estrategias básicas de búsqueda

Recorrido a lo ancho

- *Bread-First Search (BFS)*
- Exploración por niveles
- Se programa con una cola

Aplicaciones:

- Camino más corto
- Componentes conectados

Recorrido en profundidad

- *Depth-First Search (DFS)*
- Exploración agresiva
- Se programa con una pila (o con recursividad).

Aplicaciones:

- Orden topológico
- Componentes conectados

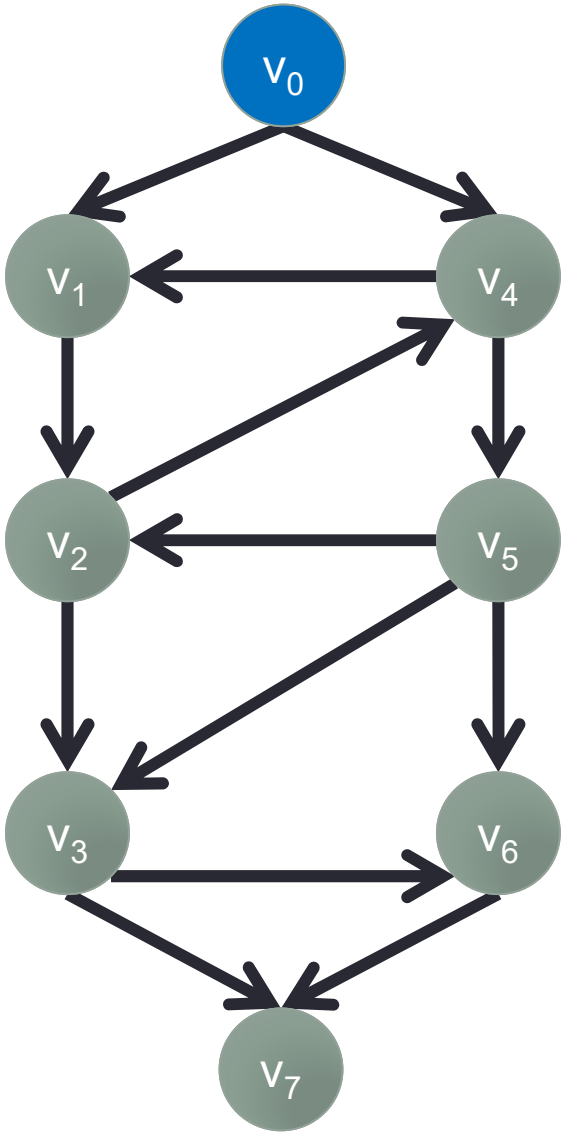
Recorrido en Profundidad (Pila)

```
class Stack:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()

    def peek(self):
        return self.items[len(self.items)-1]

    def size(self):
        return len(self.items)
```

```
obj = Stack()
obj.push(1)
obj.push(2)


print(obj.peek())    # 2
obj.pop()
print(obj.peek())    # 1
obj.pop()
print(obj.isEmpty()) # True
```

LIFO – Last In First Out

Recorrido en profundidad (*DFS*)



Marcamos el vértice origen como visible
… y lo añadimos a la pila

Pila (vértices pendientes de procesar)

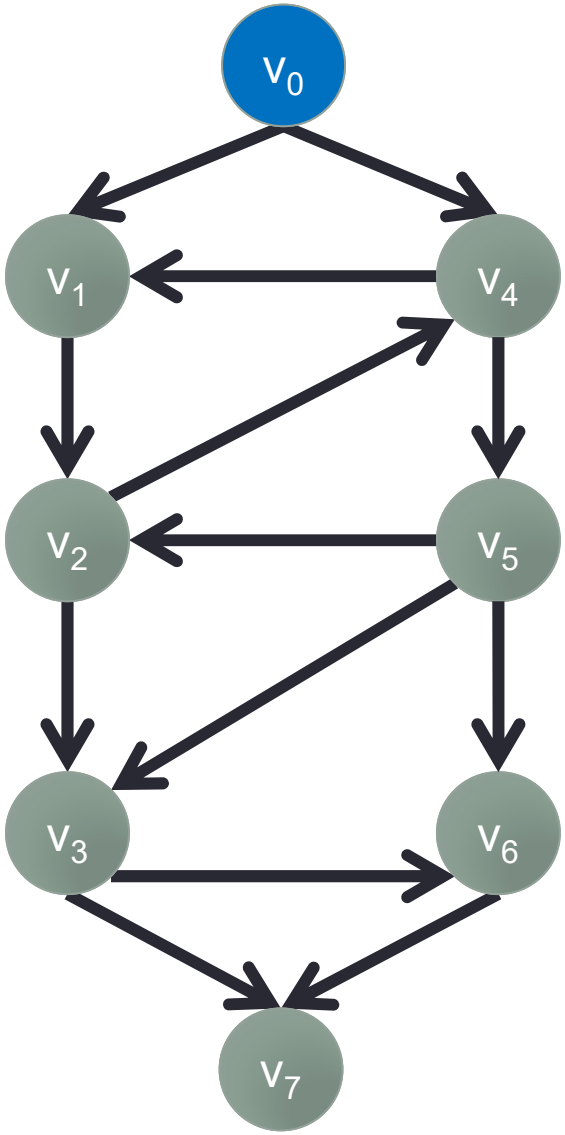
v_0							

Vértices visibles

v_0							

Recorrido en profundidad (*DFS*)

Paso 1



v_0

Sacamos un elemento de la pila

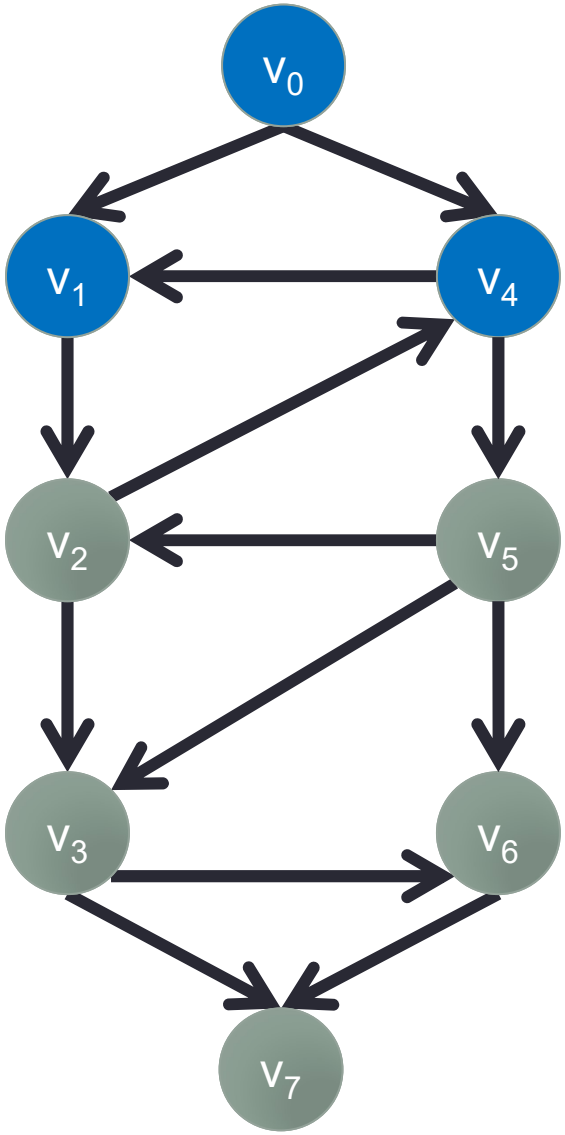
Pila (vértices pendientes de procesar)

Vértices visibles

v_0							

Recorrido en profundidad (*DFS*)

Paso 2



V_0

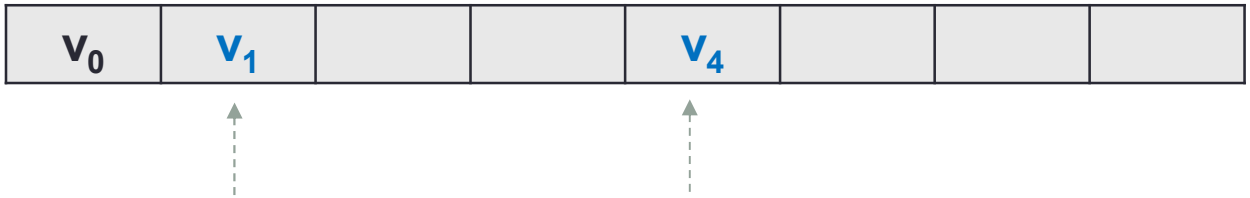
Sacamos un elemento de la pila

… marcamos sus vecinos como visibles:

V_1 y V_4

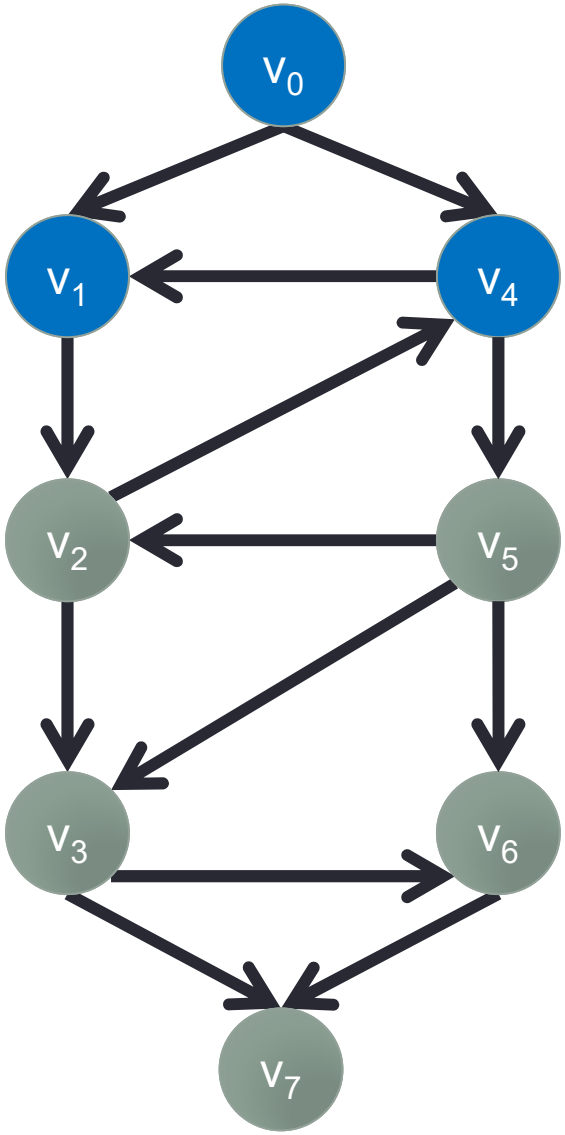
Pila (vértices pendientes de procesar)

Vértices visibles



Recorrido en profundidad (*DFS*)

Paso 3



V_0

Sacamos un elemento de la pila

… marcamos sus vecinos como visitados:

V_1 y V_4

… y los metemos en la pila para procesarlos

Pila (vértices pendientes de procesar)

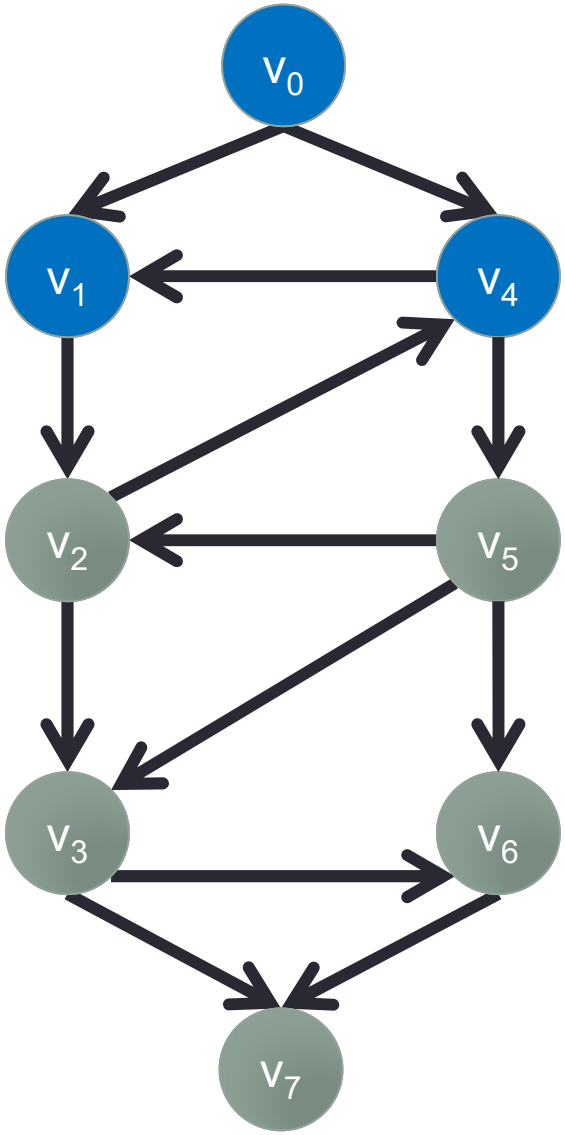
v_4	v_1						

Vértices visibles

v_0	v_1			v_4			

Recorrido en profundidad (*DFS*)

Paso 1



V_0 V_1

Sacamos un elemento de la pila:

Pila (vértices pendientes de procesar)

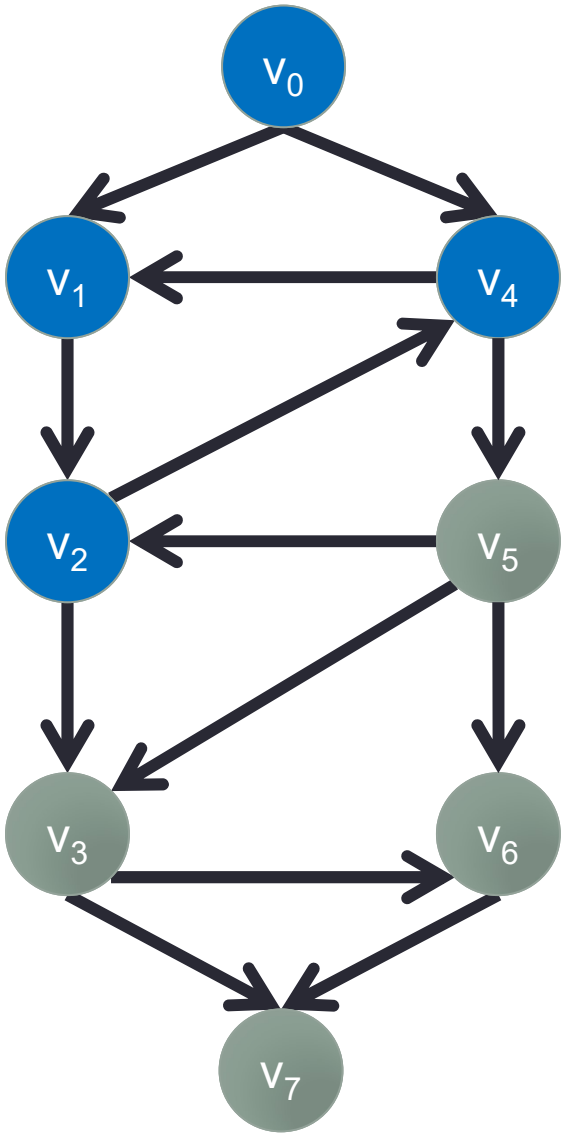
v_4							

Vértices visibles

v_0	v_1			v_4			

Recorrido en profundidad (*DFS*)

Paso 2



V_0 V_1

Sacamos un elemento de la pila

… marcamos sus vecinos como visibles:

V_2

Pila (vértices pendientes de procesar)

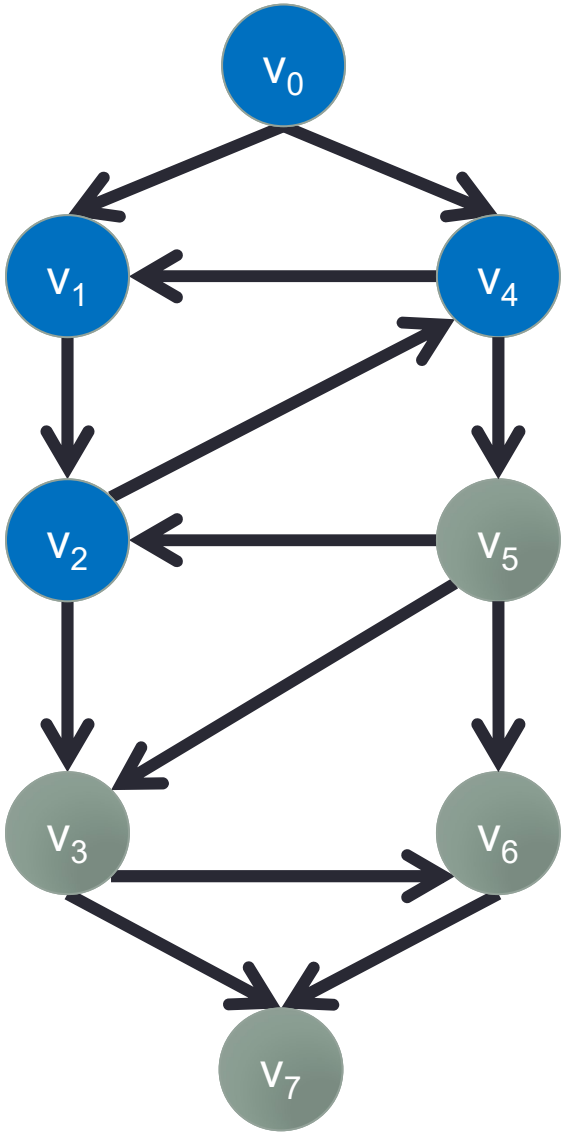
v_4							

Vértices visibles

v_0	v_1	v_2		v_4			

Recorrido en profundidad (*DFS*)

Paso 3



V_0 V_1

Sacamos un elemento de la pila

… marcamos sus vecinos como visibles:

V_2

… y los metemos en la pila para procesarlos

Pila (vértices pendientes de procesar)

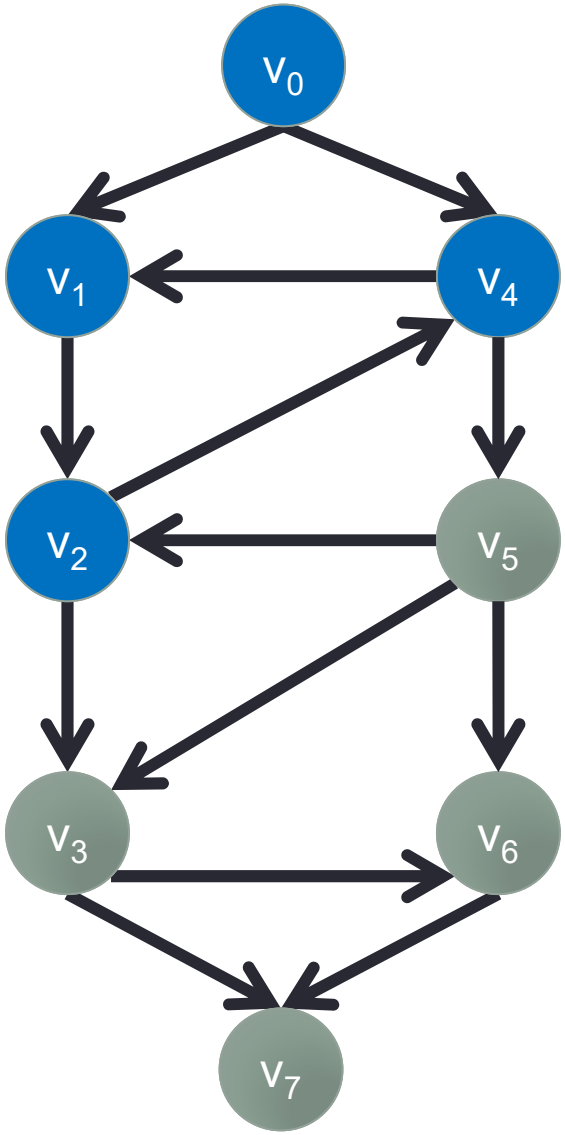
v_4	v_2						

Vértices visibles

v_0	v_1	v_2		v_4			

Recorrido en profundidad (*DFS*)

Paso 1



V_0 V_1 V_2

Sacamos un elemento de la pila

Pila (vértices pendientes de procesar)

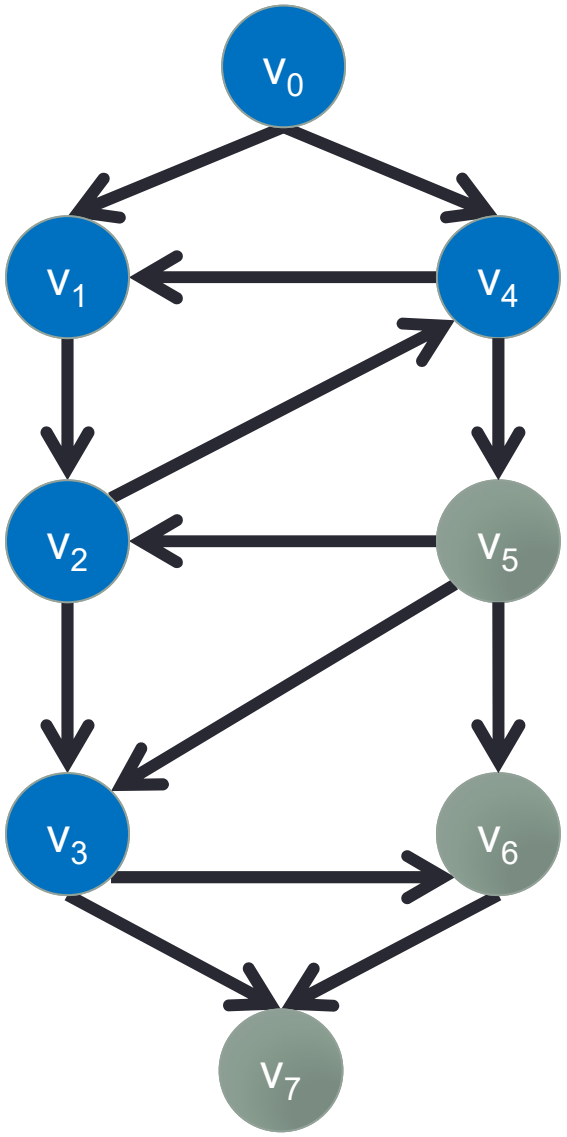
v_4							

Vértices visibles

v_0	v_1	v_2		v_4			

Recorrido en profundidad (*DFS*)

Paso 2



V_0 V_1 V_2

Sacamos un elemento de la pila

… marcamos sus vecinos como visibles:

V_3

Pila (vértices pendientes de procesar)

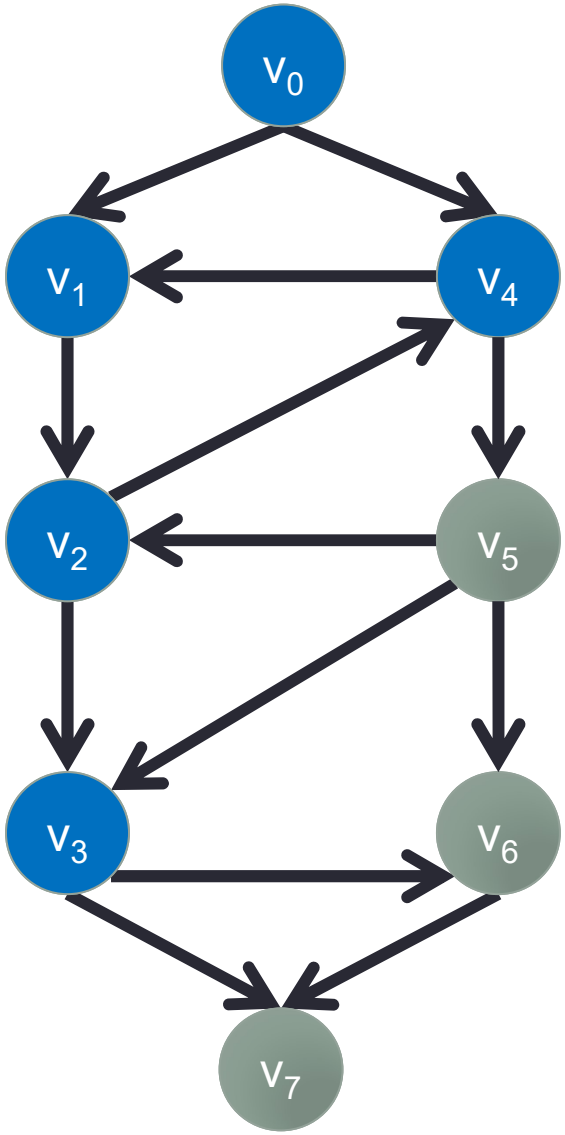
v_4							

Vértices visibles

v_0	v_1	v_2	v_3	v_4			

Recorrido en profundidad (*DFS*)

Paso 3



V_0 V_1 V_2

Sacamos un elemento de la pila

… marcamos sus vecinos como visibles:

V_3

… y los metemos en la pila para procesarlos

Pila (vértices pendientes de procesar)

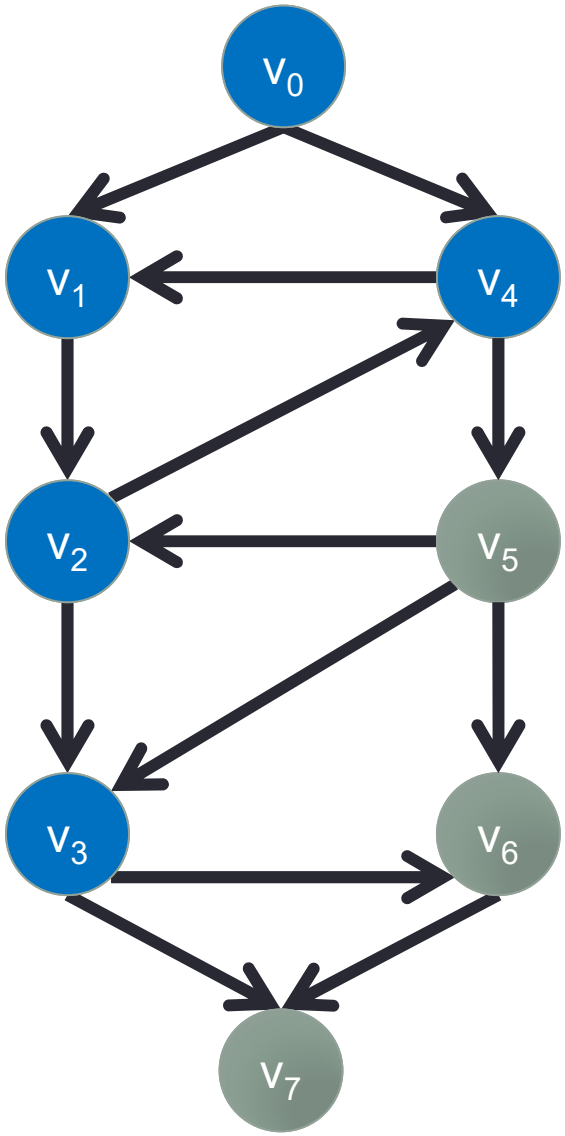
v_4	v_3						

Vértices visibles

v_0	v_1	v_2	v_3	v_4			

Recorrido en profundidad (*DFS*)

Paso 1



$V_0\ V_1\ V_2\ V_3$

Sacamos un elemento de la pila

Pila (vértices pendientes de procesar)

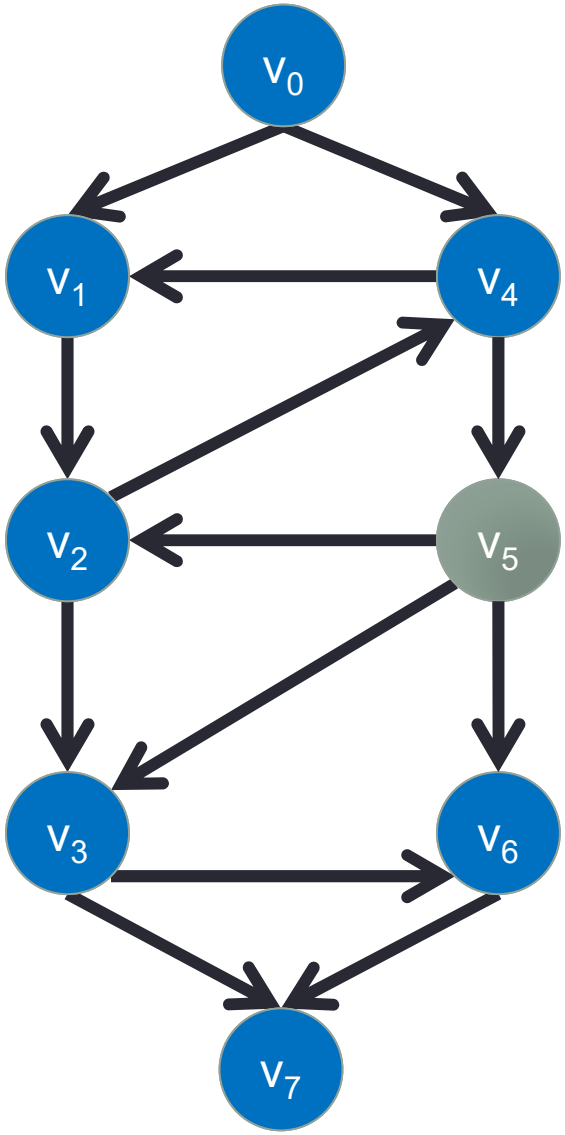
v_4							

Vértices visibles

v_0	v_1	v_2	v_3	v_4			

Recorrido en profundidad (*DFS*)

Paso 2



V_0 V_1 V_2 V_3

Sacamos un elemento de la pila

… marcamos sus vecinos como visibles:

V_6 V_7

Pila (vértices pendientes de procesar)

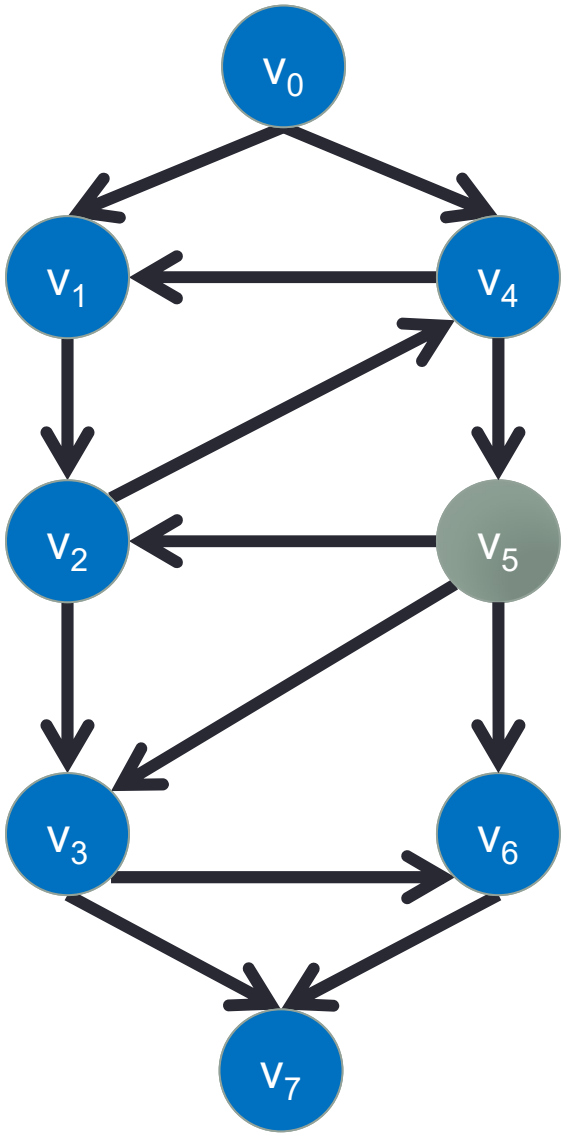
v_4							

Vértices visibles

v_0	v_1	v_2	v_3	v_4		v_6	v_7

Recorrido en profundidad (*DFS*)

Paso 3



V_0 V_1 V_2 V_3

Sacamos un elemento de la pila

… marcamos sus vecinos como visibles:

V_6 V_7

… y los metemos en la pila para procesarlos

Pila (vértices pendientes de procesar)

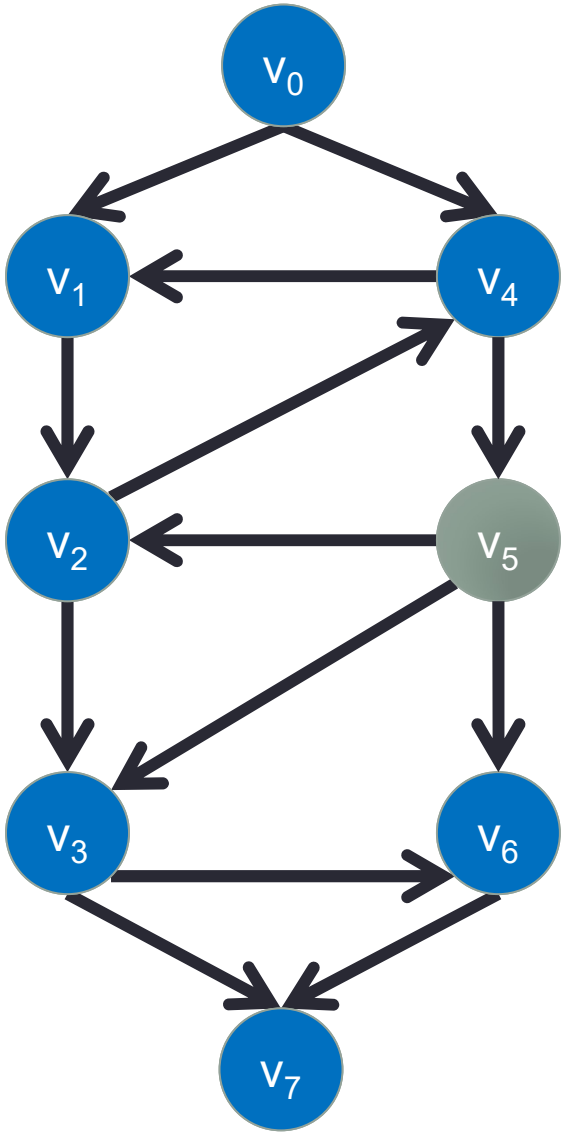
v_4	v_6	v_7					

Vértices visibles

v_0	v_1	v_2	v_3	v_4		v_6	v_7

Recorrido en profundidad (*DFS*)

Paso 1



V_0 V_1 V_2 V_3 V_7

Sacamos un elemento de la pila

Pila (vértices pendientes de procesar)

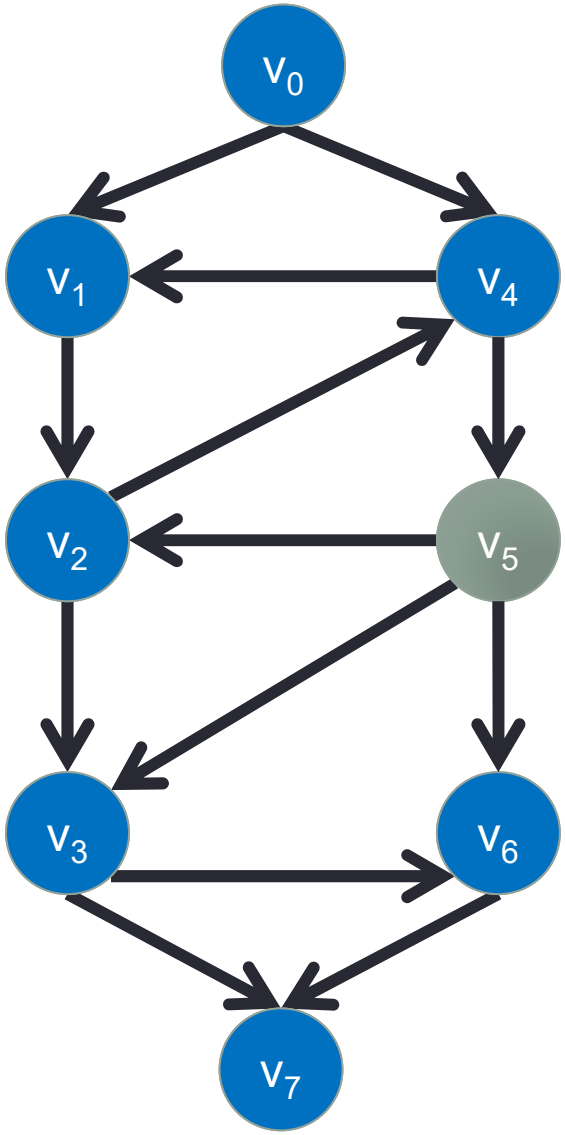
v_4	v_6						

Vértices visibles

v_0	v_1	v_2	v_3	v_4		v_6	v_7

Recorrido en profundidad (*DFS*)

Paso 2



V_0 V_1 V_2 V_3 V_7

Sacamos un elemento de la pila

… no tiene vecinos no visibles

Pila (vértices pendientes de procesar)

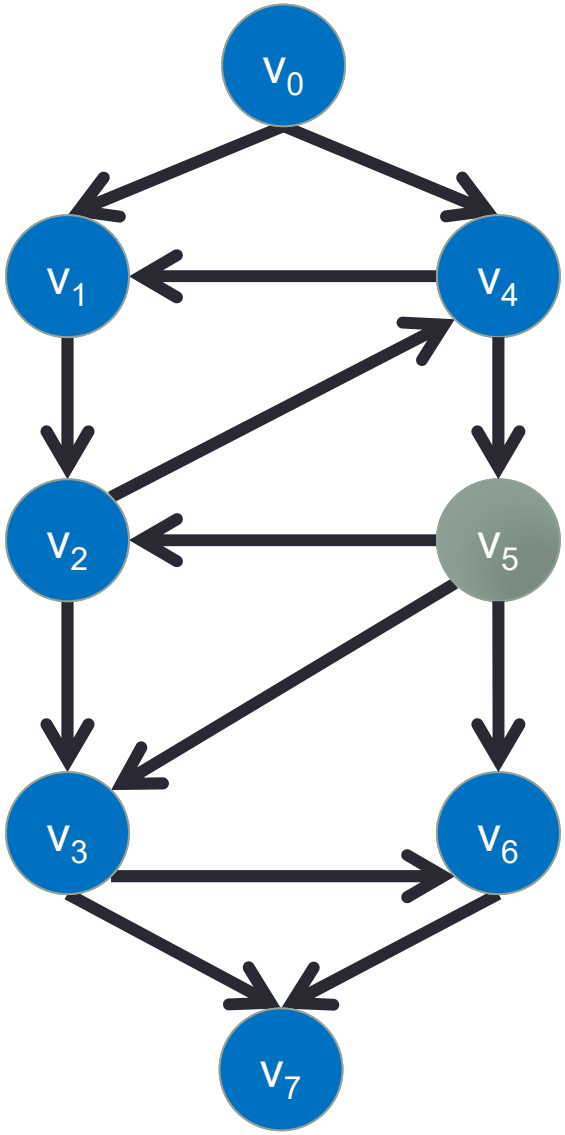
v_4	v_6						

Vértices visibles

v_0	v_1	v_2	v_3	v_4		v_6	v_7

Recorrido en profundidad (*DFS*)

Paso 2



V_0 V_1 V_2 V_3 V_7 V_6

Sacamos un elemento de la pila

Pila (vértices pendientes de procesar)

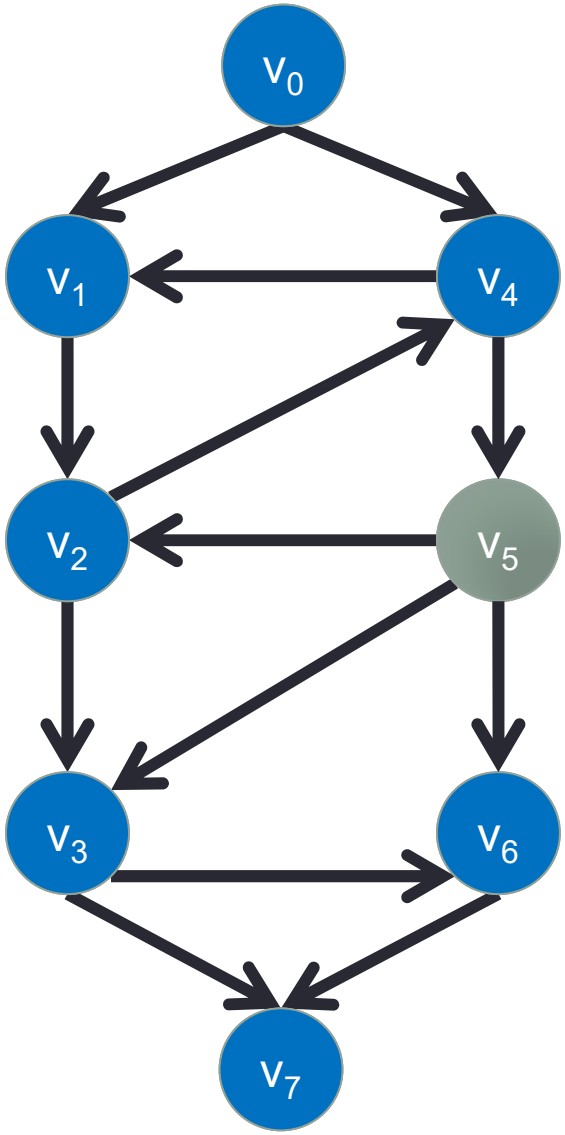
v_4							

Vértices visibles

v_0	v_1	v_2	v_3	v_4		v_6	v_7

Recorrido en profundidad (*DFS*)

Paso 2



V_0 V_1 V_2 V_3 V_7 V_6

Sacamos un elemento de la pila

… no tiene vecinos no visibles

Pila (vértices pendientes de procesar)

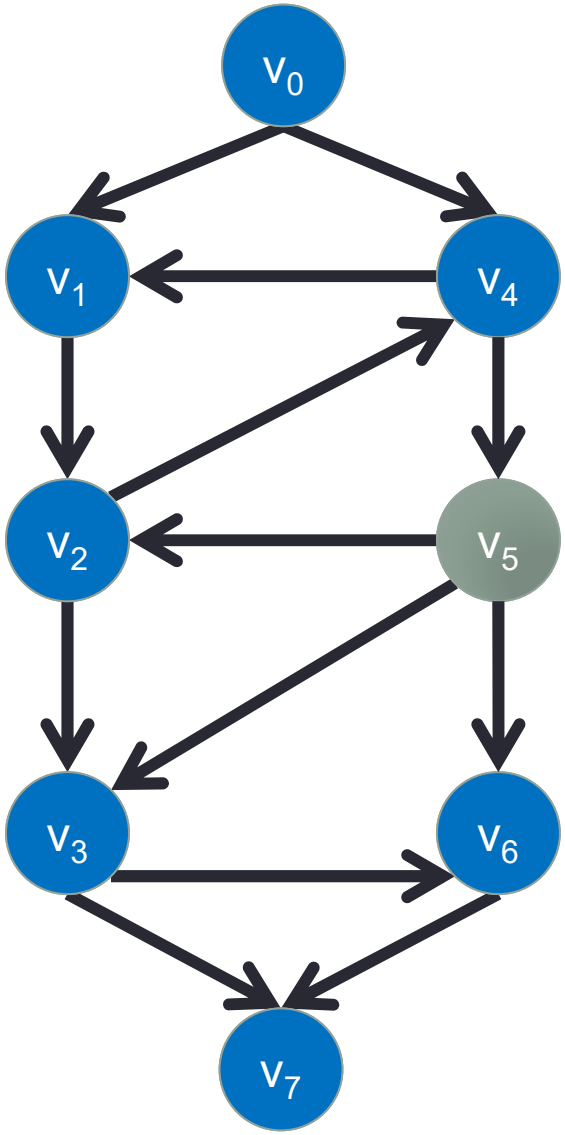
v_4							

Vértices visibles

v_0	v_1	v_2	v_3	v_4		v_6	v_7

Recorrido en profundidad (*DFS*)

Paso 2



$V_0\ V_1\ V_2\ V_3\ V_7\ V_6\ V_4$

Sacamos un elemento de la pila

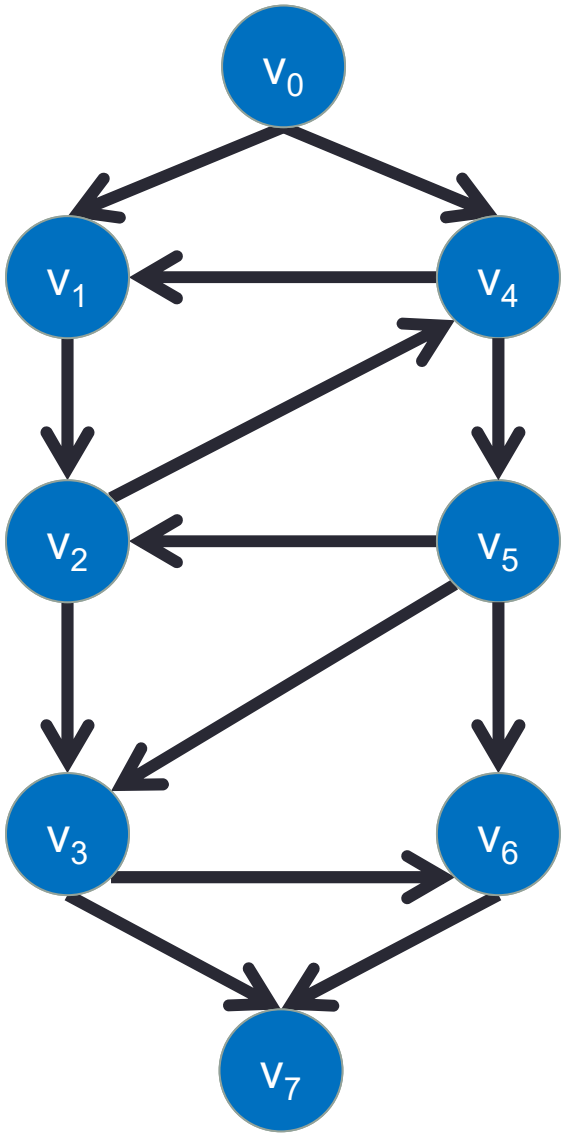
Pila (vértices pendientes de procesar)

Vértices visibles

v_0	v_1	v_2	v_3	v_4		v_6	v_7

Recorrido en profundidad (*DFS*)

Paso 1



$V_0\ V_1\ V_2\ V_3\ V_7\ V_6\ V_4$

Sacamos un elemento de la pila

… marcamos sus vecinos como visibles:

V_5

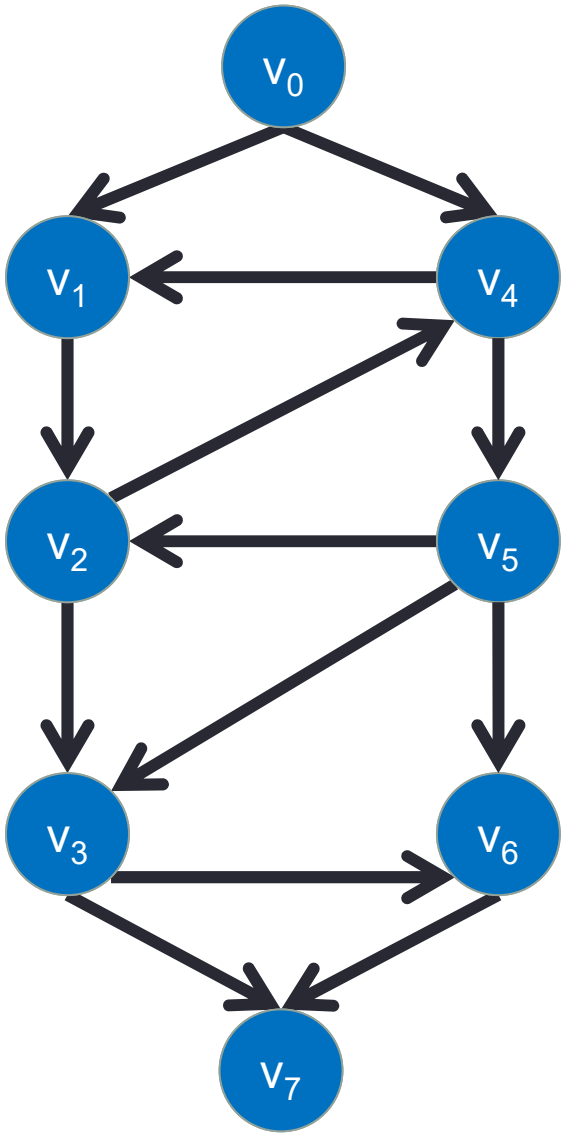
Pila (vértices pendientes de procesar)

Vértices visibles

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7

Recorrido en profundidad (*DFS*)

Paso 3



V_0 V_1 V_2 V_3 V_7 V_6 V_4

Sacamos un elemento de la pila

… marcamos sus vecinos como visibles:

V_6

… y los metemos en la pila para procesarlos

Pila (vértices pendientes de procesar)

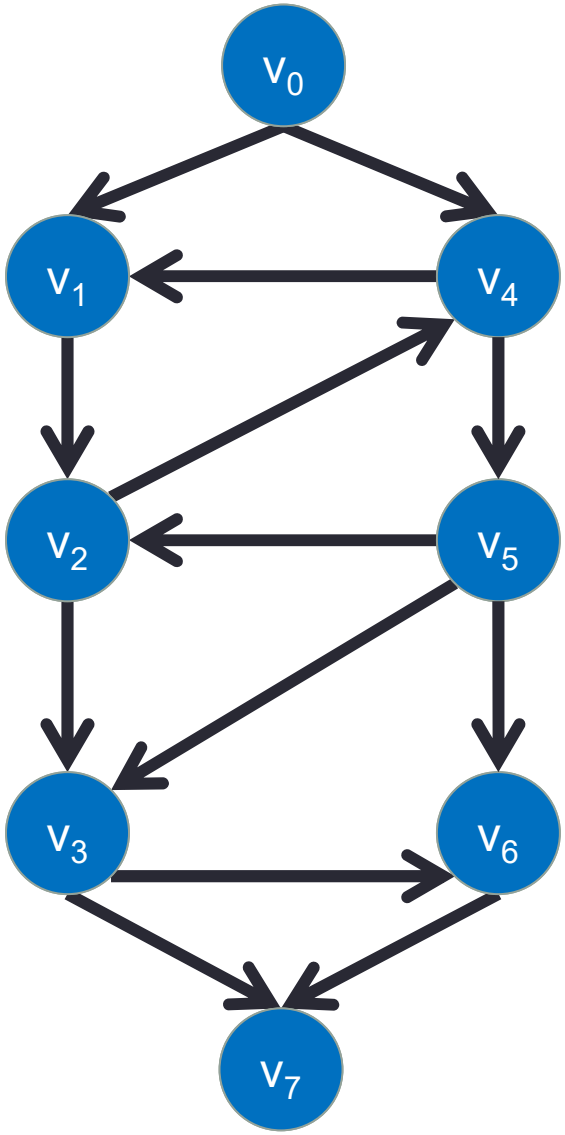
v_5							

Vértices visibles

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7

Recorrido en profundidad (*DFS*)

Paso 1



$V_0\ V_1\ V_2\ V_3\ V_7\ V_6\ V_4\ V_5$

Sacamos un elemento de la pila

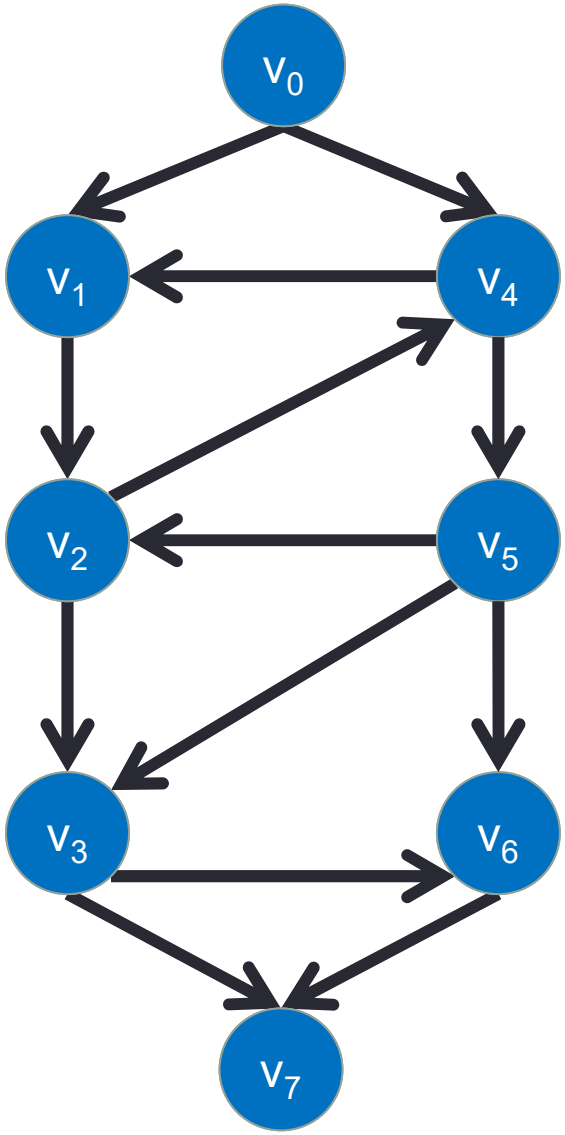
Pila (vértices pendientes de procesar)

Vértices visibles

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7

Recorrido en profundidad (*DFS*)

Paso 2



$V_0\ V_1\ V_2\ V_3\ V_7\ V_6\ V_4\ V_5$

Sacamos un elemento de la pila

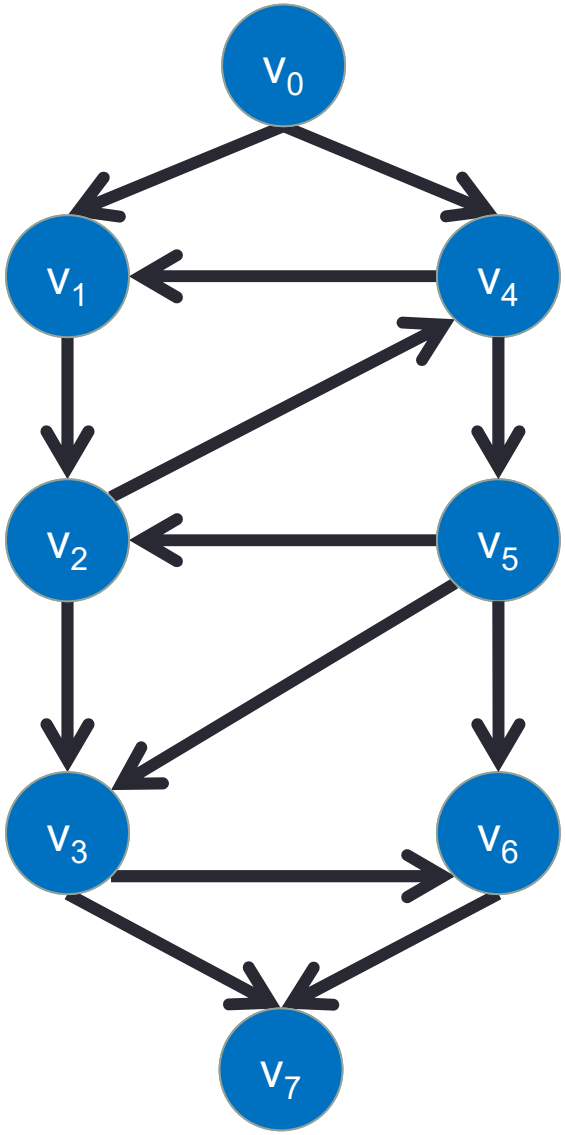
… no tiene vecinos no visibles

Pila (vértices pendientes de procesar)

Vértices visibles

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7

Recorrido en profundidad (*DFS*)



$V_0\ V_1\ V_2\ V_3\ V_7\ V_6\ V_4\ V_5$

… *recorrido en profundidad*

Recorrido en profundidad (DFS)

DFS (grafo G, vertice inicial vi)

- S = Stack()
- Marcar vi como visible;
- S.push(vi)

- Mientras not S.isEmpty ():
  - v = S.pop()

  - Para todas las aristas (v,w):
    - Si w no es visible:
      - Marcar w como visible
      - S.push(w)

Implementación Iterativa

Recorrido en profundidad (DFS)

DFS (grafo G, vertice inicial vi)

- Marcar vi como visible

- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)

Llamada recursiva

Implementación Recursiva

Estrategias básicas de búsqueda

Recorrido a lo ancho

- *Bread-First Search (BFS)*
- Exploración por niveles
- Se programa con una cola

Aplicaciones:

- Camino más corto
- Componentes conectados *(grafo no dirigido)*

Recorrido en profundidad

- *Depth-First Search (DFS)*
- Exploración agresiva
- Se programa con una pila (o con recursividad).

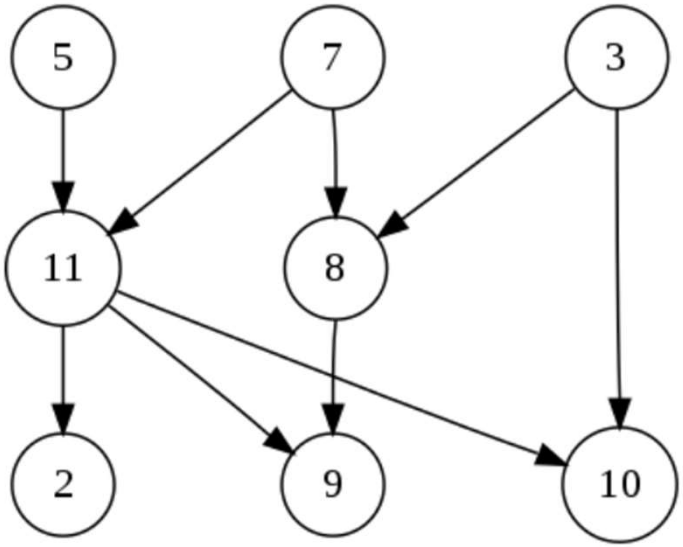
Aplicaciones:

- Orden topológico
- Componentes conectados *(grafo dirigido)*

Aplicación: Orden topológico

- ***Dado un grafo acíclico dirigido calcular un orden válido de recorrido topológico***

- Definición: el órden topológico de un grafo dirigido es una ordenación lineal en la que cada arista (u,v) establece que u precede v en el orden final.



Soluciones válidas:

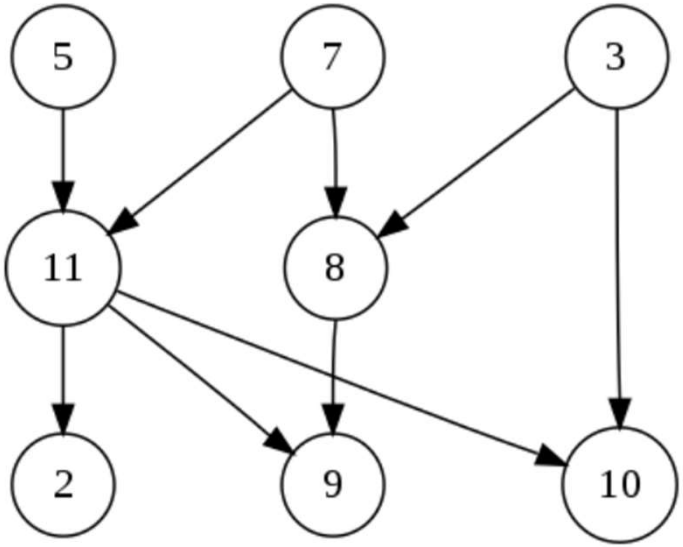
- 5, 7, 3, 11, 8, 2, 9, 10
- 3, 5, 7, 8, 11, 2, 9, 10
- 5, 7, 3, 8, 11, 10, 9, 2
- 7, 5, 11, 3, 10, 8, 9, 2
- 5, 7, 11, 2, 3, 8, 9, 10
- 3, 7, 8, 5, 11, 10, 2, 9

https://en.wikipedia.org/wiki/Topological_sorting

Aplicación: Orden topológico con DFS

n = número de vertices

Con todos los vertices v:
- Si v no es visible:
  - DFS (G, v)



DFS (grafo G, vertice inicial v)
- Marcar v como visible

- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)

- **orden(v) = n**
- **n = n – 1**

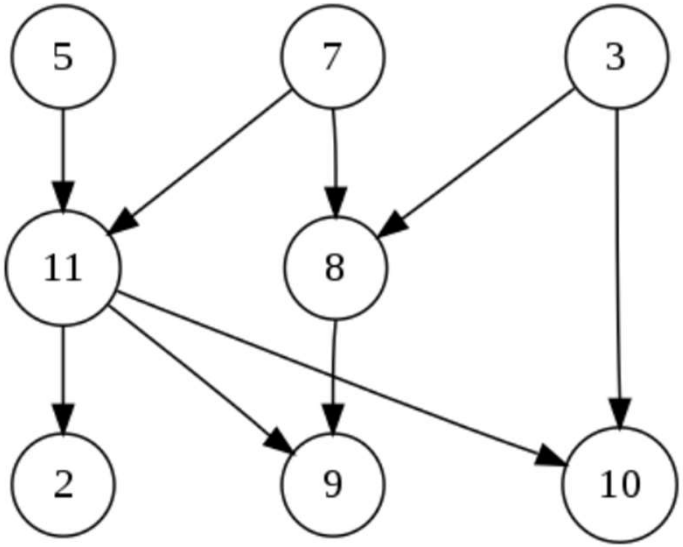
return

Aplicación: Orden topológico con DFS

n = número de vertices

Con todos los vertices v:

- Si v no es visible:
  - DFS (G, v)



DFS (grafo G, vertice inicial v)

- Marcar v como visible

- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)

- **orden(v) = n**
- **n = n – 1**

return

n=8

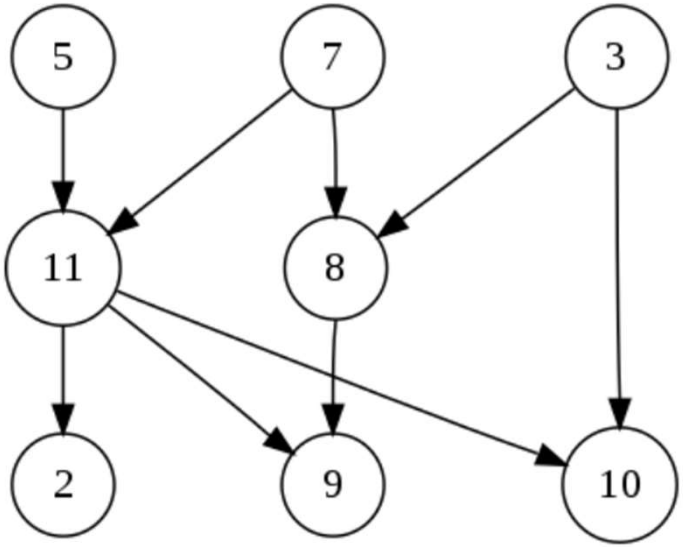
Aplicación: Orden topológico con DFS

n = número de vertices

Con todos los vertices v:

- Si v no es visible:
  - DFS (G, v)

dfs()



DFS (grafo G, vertice inicial v)

- Marcar v como visible

- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)

- **orden(v) = n**
- **n = n – 1**

return

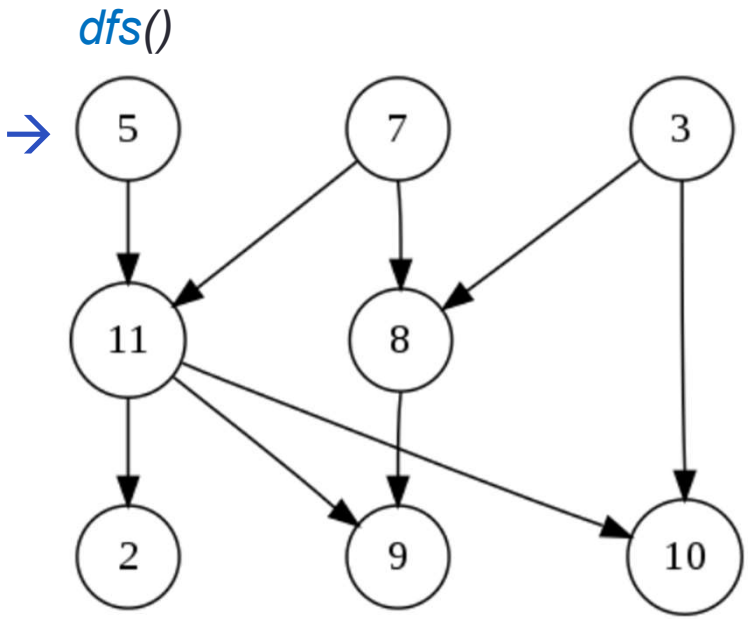
n=8

Aplicación: Orden topológico con DFS

n = número de vertices

Con todos los vertices v:

- Si v no es visible:
  - DFS (G, v)



DFS (grafo G, vertice inicial v)

- Marcar v como visible

- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)

- **orden(v) = n**
- **n = n – 1**

return

n=8

Aplicación: Orden topológico con DFS

n = número de vertices

Con todos los vertices v:

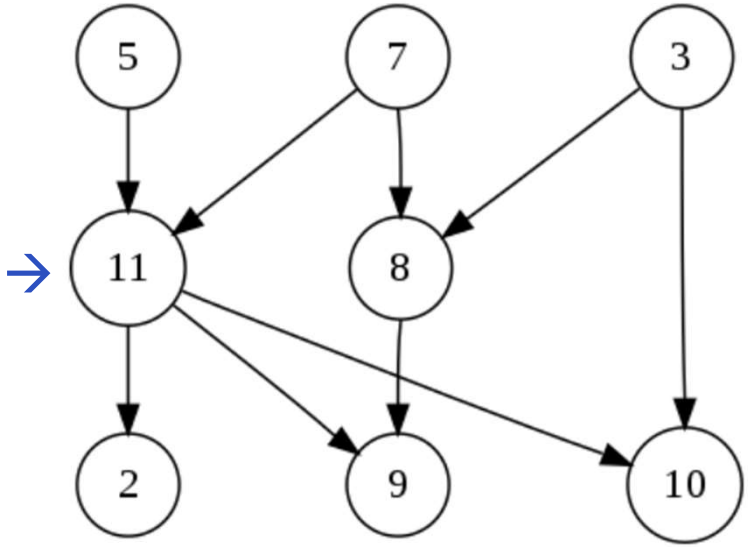
- Si v no es visible:
  - DFS (G, v)

DFS (grafo G, vertice inicial v)

- Marcar v como visible
- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)
- **orden(v) = n**
- **n = n – 1**

return

dfs()



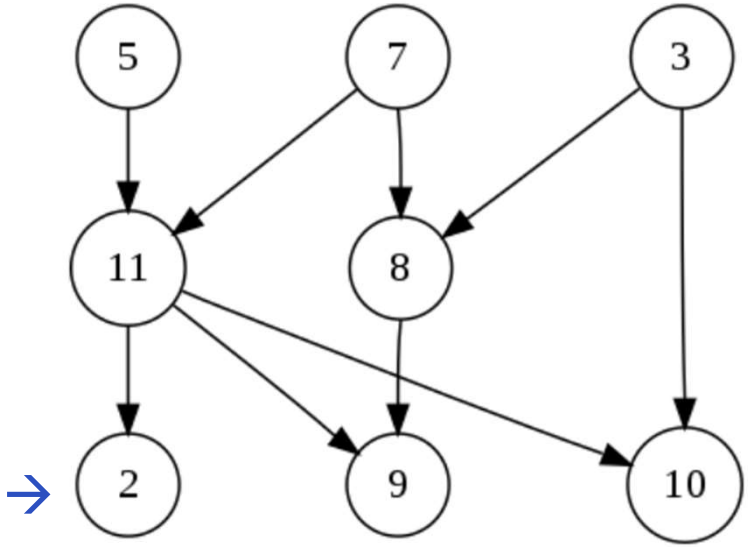
n=8

Aplicación: Orden topológico con DFS

n = número de vertices

Con todos los vertices v:
- Si v no es visible:
  - DFS (G, v)

dfs()



DFS (grafo G, vertice inicial v)
- Marcar v como visible

- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)

- **orden(v) = n**
- **n = n – 1**

return

n=8

Aplicación: Orden topológico con DFS

n = número de vertices

Con todos los vertices v:

- Si v no es visible:
  - DFS (G, v)

DFS (grafo G, vertice inicial v)

- Marcar v como visible

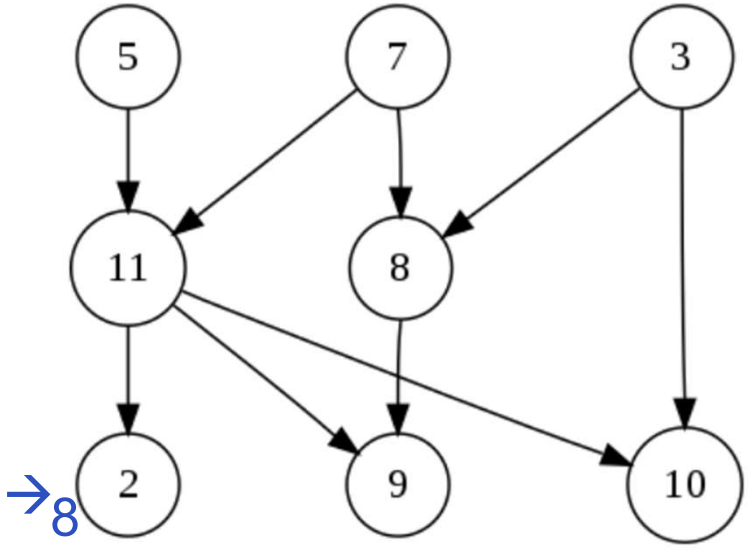
- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)

- **orden(v) = n**
- **n = n – 1**

return

n=7

dfs()



Aplicación: Orden topológico con DFS

n = número de vertices

Con todos los vertices v:

- Si v no es visible:
  - DFS (G, v)

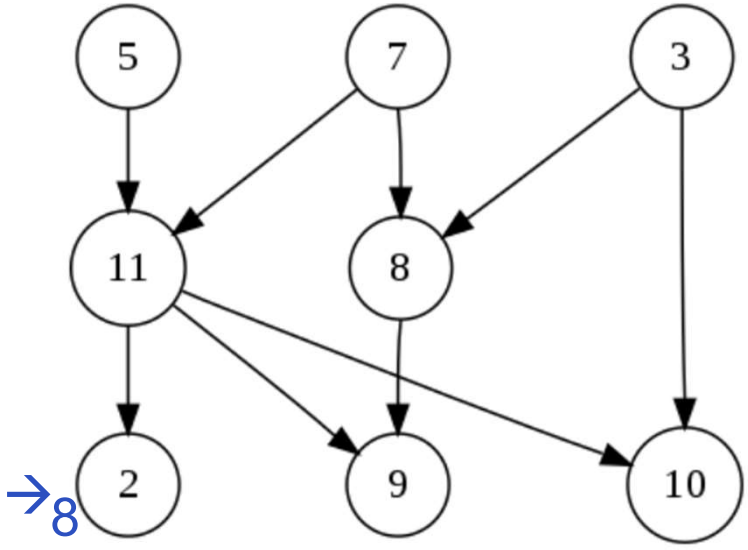
DFS (grafo G, vertice inicial v)

- Marcar v como visible
- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)
- **orden(v) = n**
- **n = n – 1**

return

n=7

dfs()



Aplicación: Orden topológico con DFS

n = número de vertices

Con todos los vertices v:

- Si v no es visible:
  - DFS (G, v)

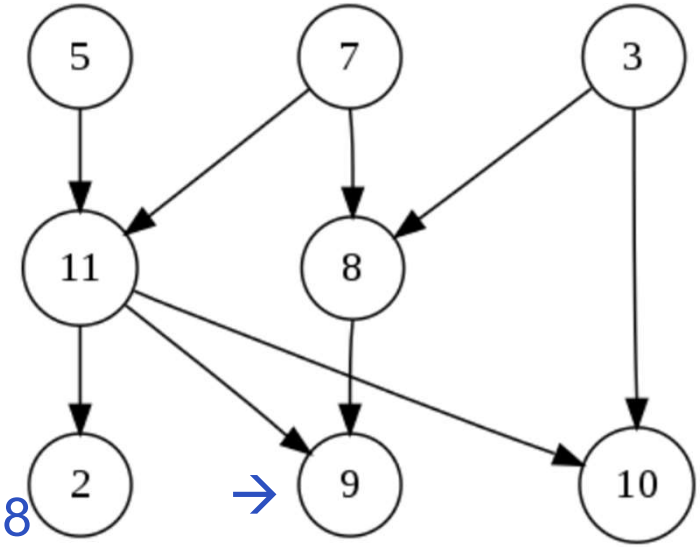
DFS (grafo G, vertice inicial v)

- Marcar v como visible
- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)
- **orden(v) = n**
- **n = n – 1**

return

n=7

dfs()



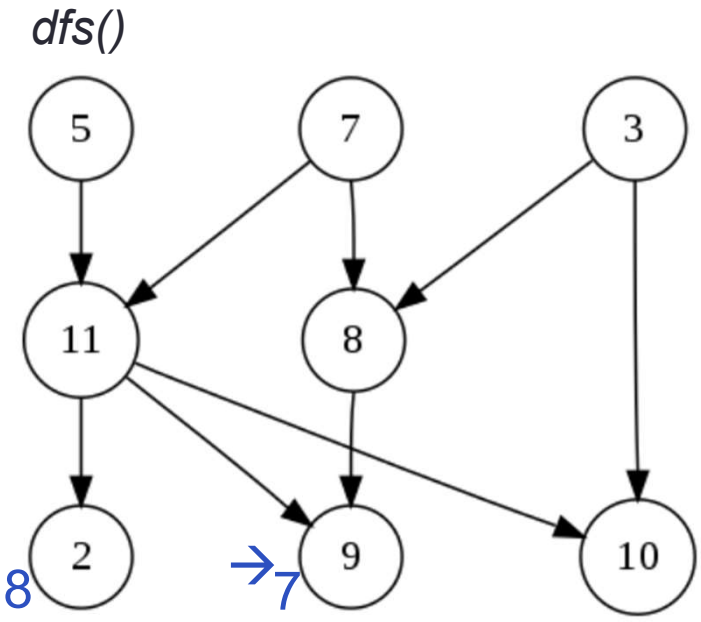
8 →

Aplicación: Orden topológico con DFS

n = número de vertices

Con todos los vertices v:

- Si v no es visible:
  - DFS (G, v)



DFS (grafo G, vertice inicial v)

- Marcar v como visible

- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)

- **orden(v) = n**
- **n = n – 1**

return

n=6

Aplicación: Orden topológico con DFS

n = número de vertices

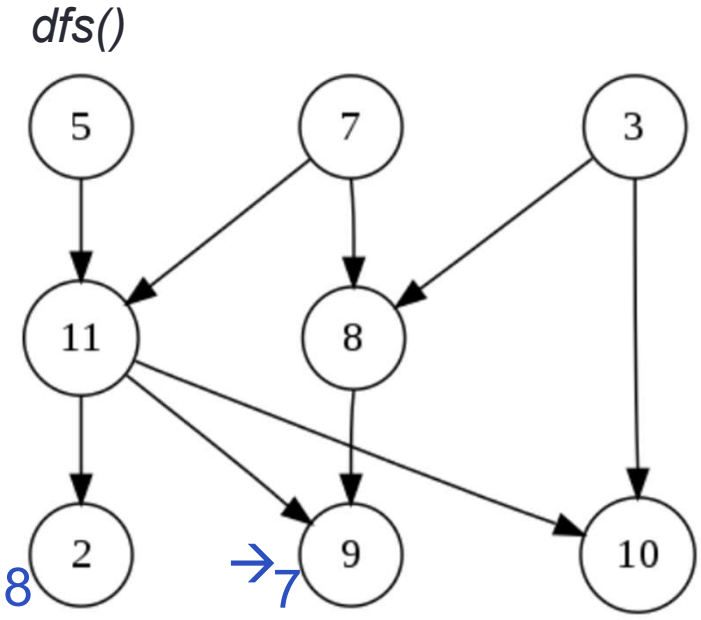
Con todos los vertices v:

- Si v no es visible:
  - DFS (G, v)

DFS (grafo G, vertice inicial v)

- Marcar v como visible
- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)
- **orden(v) = n**
- **n = n – 1**

return



n=6

Aplicación: Orden topológico con DFS

n = número de vertices

Con todos los vertices v:

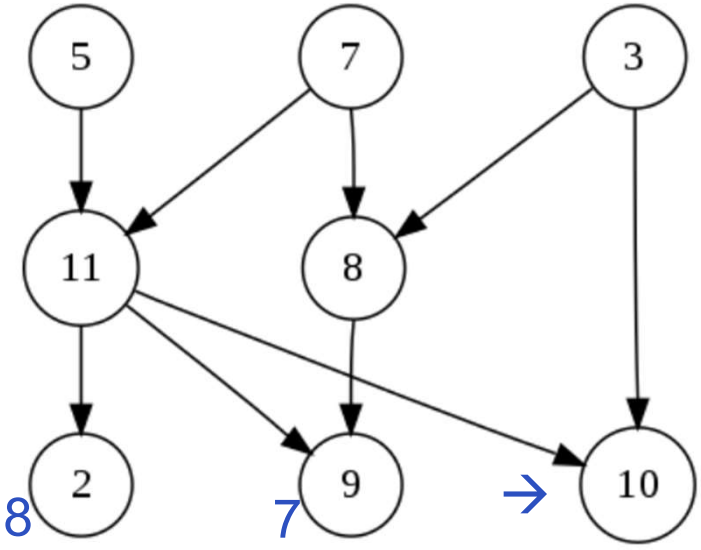
- Si v no es visible:
  - DFS (G, v)

DFS (grafo G, vertice inicial v)

- Marcar v como visible
- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)
- **orden(v) = n**
- **n = n – 1**

return

dfs()



n=6

Aplicación: Orden topológico con DFS

n = número de vertices

Con todos los vertices v:

- Si v no es visible:
  - DFS (G, v)

DFS (grafo G, vertice inicial v)

- Marcar v como visible

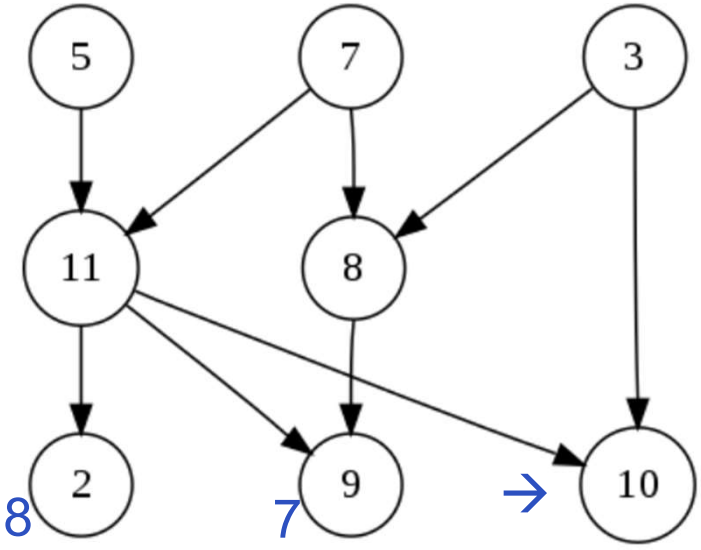
- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)

- **orden(v) = n**
- **n = n – 1**

return

n=6

dfs()



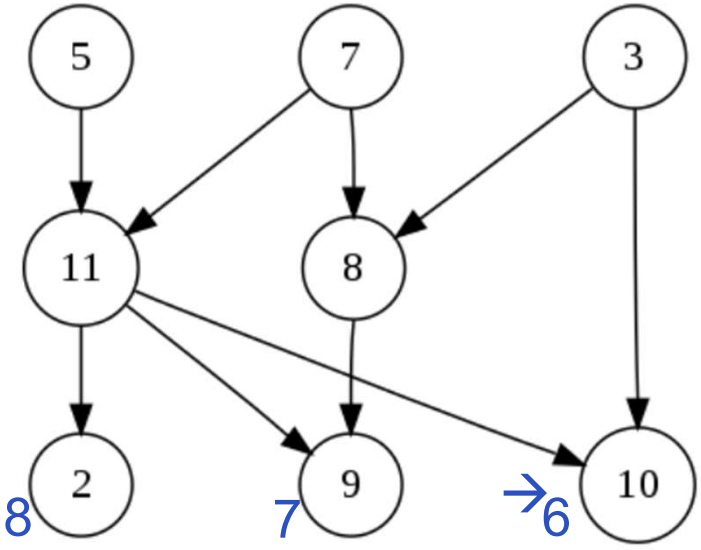
Aplicación: Orden topológico con DFS

n = número de vertices

Con todos los vertices v:

- Si v no es visible:
  - DFS (G, v)

dfs()



DFS (grafo G, vertice inicial v)

- Marcar v como visible

- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)

- **orden(v) = n**
- **n = n – 1**

return

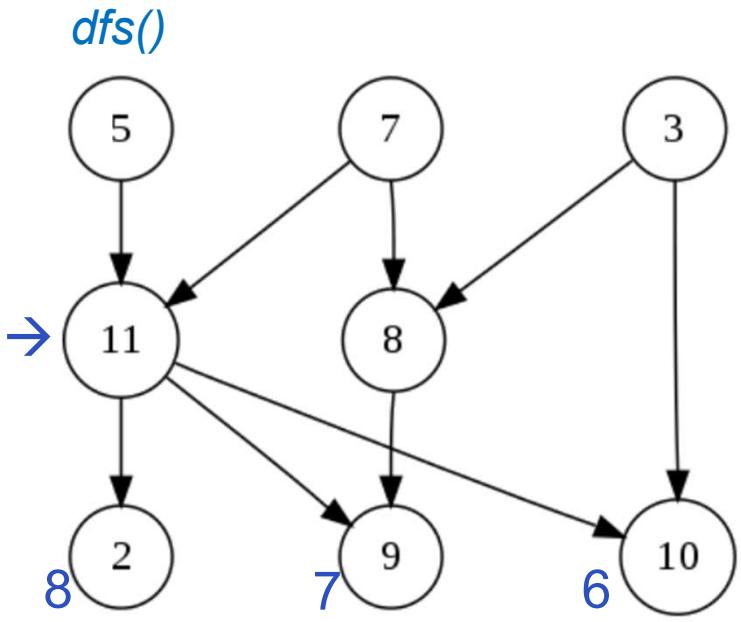
n=5

Aplicación: Orden topológico con DFS

n = número de vertices

Con todos los vertices v:

- Si v no es visible:
  - DFS (G, v)



DFS (grafo G, vertice inicial v)

- Marcar v como visible

- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)

- **orden(v) = n**
- **n = n – 1**

return

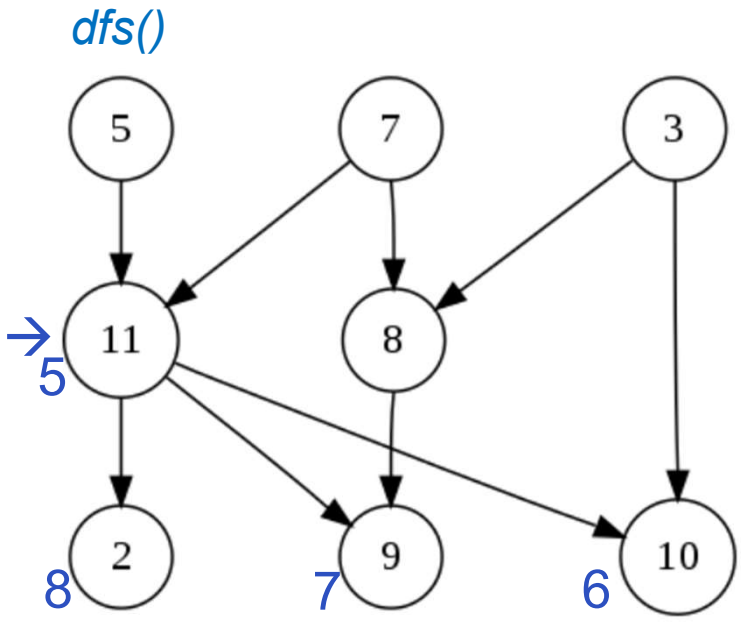
n=5

Aplicación: Orden topológico con DFS

n = número de vertices

Con todos los vertices v:

- Si v no es visible:
  - DFS (G, v)



DFS (grafo G, vertice inicial v)

- Marcar v como visible

- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)

- **orden(v) = n**
- **n = n – 1**

return

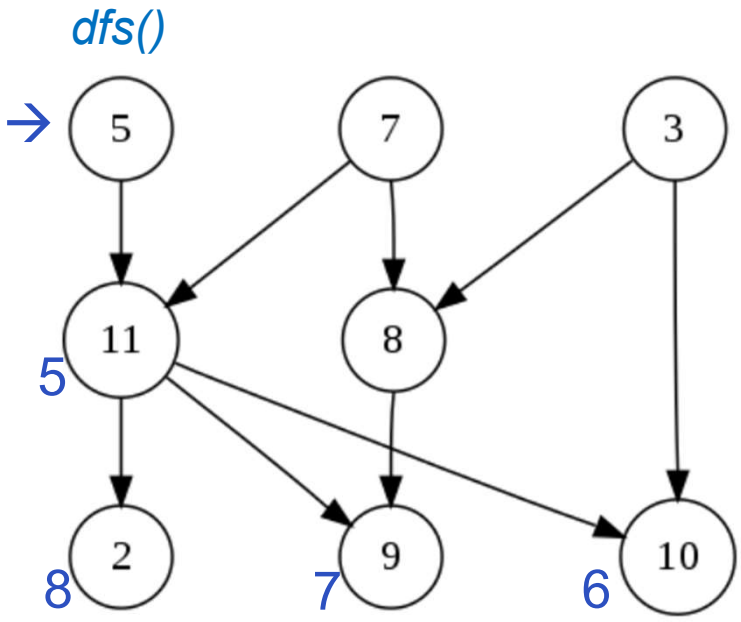
n=4

Aplicación: Orden topológico con DFS

n = número de vertices

Con todos los vertices v:

- Si v no es visible:
  - DFS (G, v)



DFS (grafo G, vertice inicial v)

- Marcar v como visible

- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)

- **orden(v) = n**
- **n = n – 1**

return

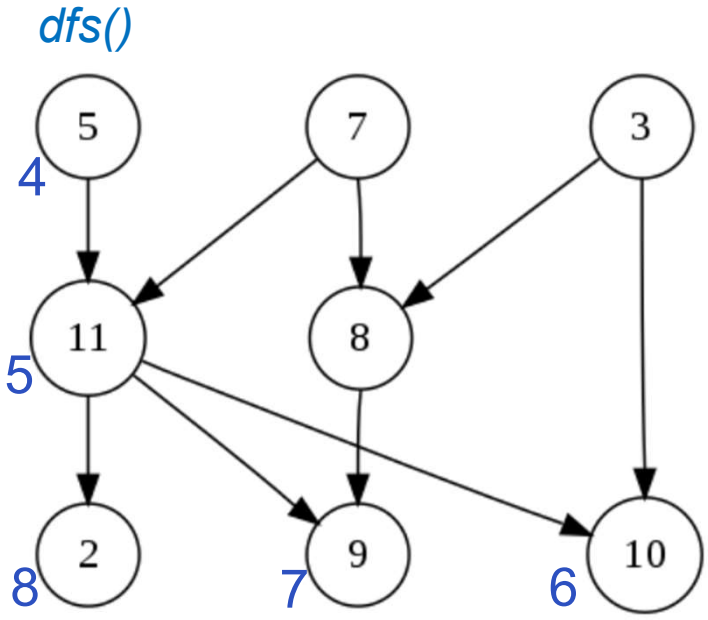
n=4

Aplicación: Orden topológico con DFS

n = número de vertices

Con todos los vertices v:

- Si v no es visible:
  - DFS (G, v)



DFS (grafo G, vertice inicial v)

- Marcar v como visible

- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)

- **orden(v) = n**
- **n = n – 1**

return

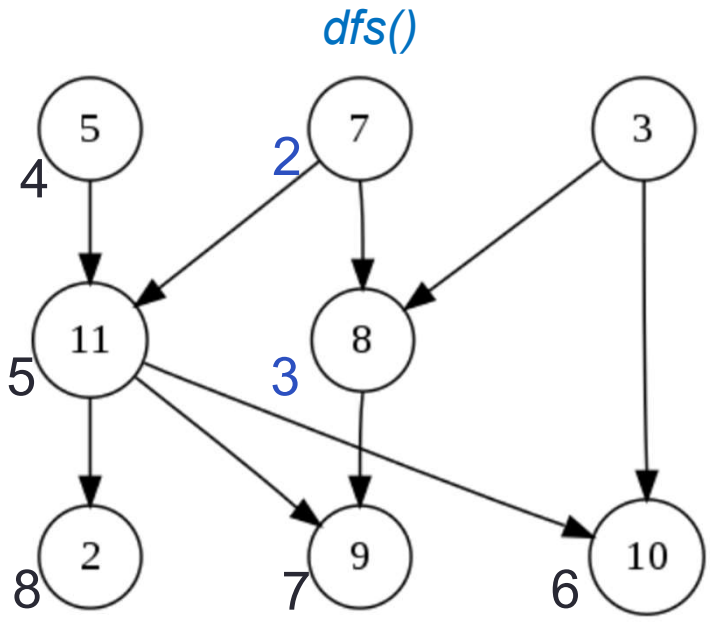
n=3

Aplicación: Orden topológico con DFS

n = número de vertices

Con todos los vertices v:

- Si v no es visible:
  - DFS (G, v)



DFS (grafo G, vertice inicial v)

- Marcar v como visible

- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)

- **orden(v) = n**
- **n = n – 1**

return

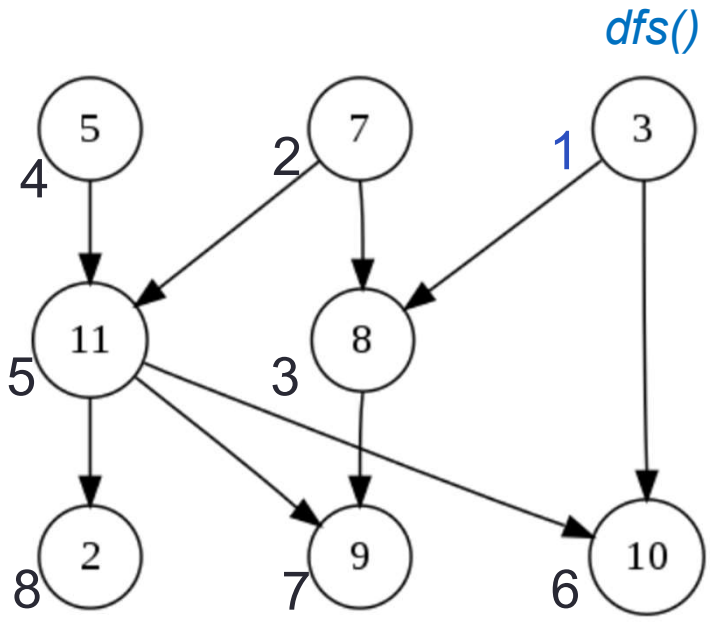
n=1

Aplicación: Orden topológico con DFS

n = número de vertices

Con todos los vertices v:

- Si v no es visible:
  - DFS (G, v)



DFS (grafo G, vertice inicial v)

- Marcar v como visible

- Para todas las aristas (v,w):
  - Si w no es visible:
    - DFS (G, w)

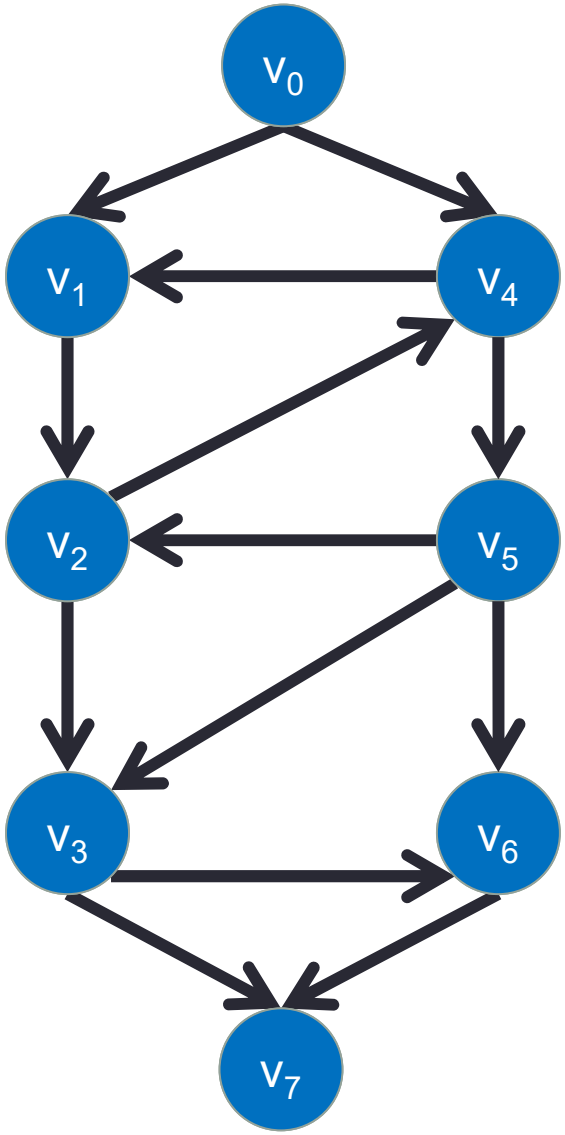
- **orden(v) = n**
- **n = n – 1**

return

n=0

Solución: 3 7 8 5 11 10 9 2

Recorridos en Grafos



Para recorrer un grafo no siempre es necesario construir el grafo.

Podemos recorrerlo utilizando sólamente las estructuras de datos auxiliares

Importante para recorrer grafos grandes!

Pila / Cola

Vértices visibles

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7

Resumen

Recorrido a lo ancho

- *Bread-First Search (BFS)*
- Exploración por niveles
- Se programa con una cola

Aplicaciones:

- Camino más corto
- Componentes conectados (en grafo no dirigido)

Recorrido en profundidad

- *Depth-First Search (DFS)*
- Exploración agresiva
- Se programa con una pila (o con recursividad)

Aplicaciones:

- Orden topológico
- Componentes conectados (en grafo dirigido)