RECORRIDOS BÁSICOS EN GRAFOS

Algoritmos y Programación

Javier Miranda

Escuela de Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

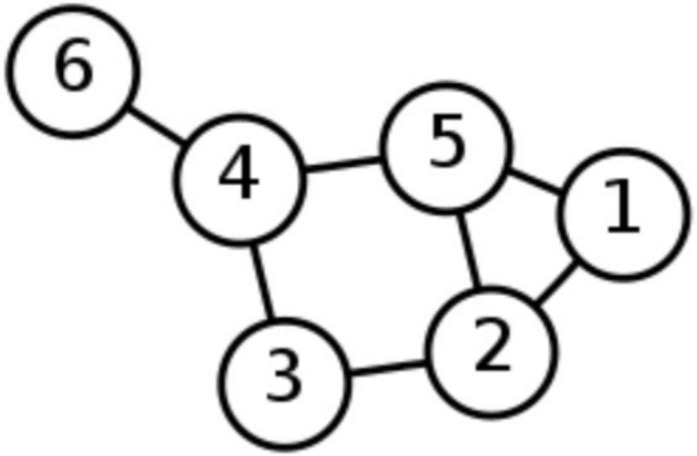
14 de Septiembre de 2023

Contenido

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

Grafo: Definición

- Conjunto no vacío de objetos llamados vértices o nodos, conectados entre algunos de ellos por enlaces llamados aristas, que pueden ser o no orientadas



Variantes:
- Dirigidos o no dirigidos
- Con pesos en las aristas

- Ejemplos de uso: redes sociales, rutas de vehículos, la Web, restricciones de precedencia, etc.

Representación de grafos

Matriz de adyacencia

- Matriz cuadrada (NxN)

$$\begin{array}{cc} & \begin{array}{ccc} V1 & V2 & V3 \end{array} \\ \begin{array}{c} V1 \\ V2 \\ V3 \end{array} & \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}$$

Listas de adyacencia

- Array (o lista) de vertices
- Array (o lista) de aristas

V = {V1, V2, V3}

E = {(1,2), (1,3), (2,3)}

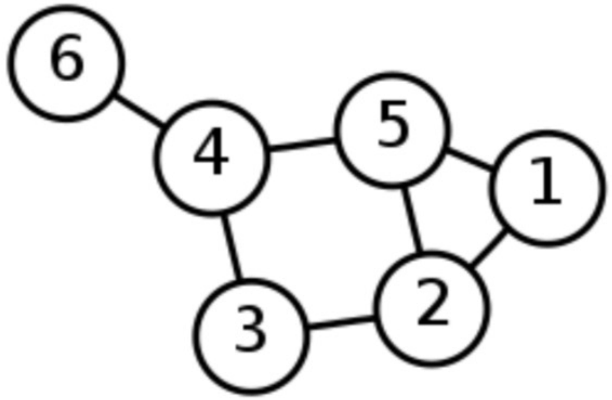
En esta asignatura vamos a utilizar el soporte que proporciona NetworkX

https://networkx.org

NetworkX
Network Analysis in Python

Búsqueda en un Grafo

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



<u>Algoritmo genérico</u>

Marcar el vértice origen como visitado

mientras queden vértices por procesar.

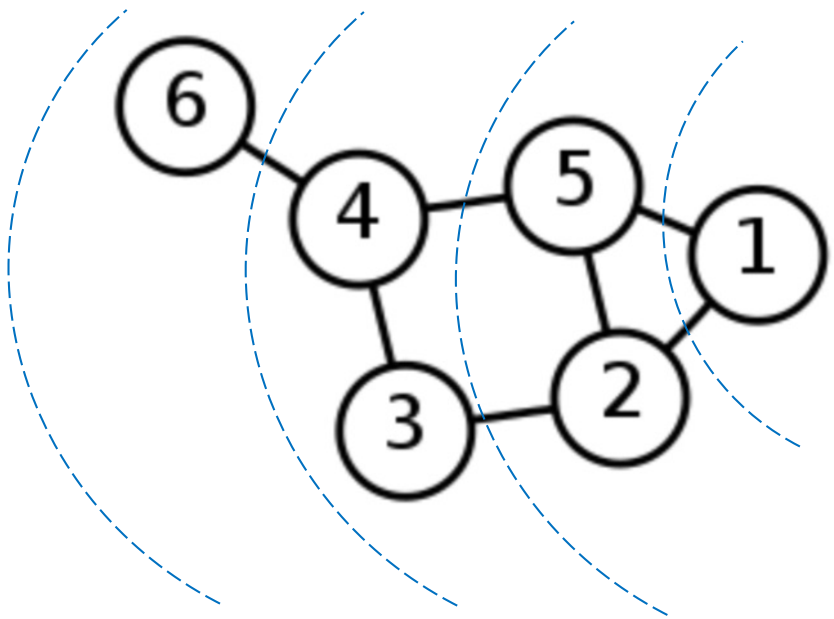
- <u>Elegir</u> una arista (U,V) con U visitado y V no visitado
- Procesar esta ruta
- Marcar el vértice V como visitado

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

Estrategias básicas de búsqueda

Recorrido a lo ancho

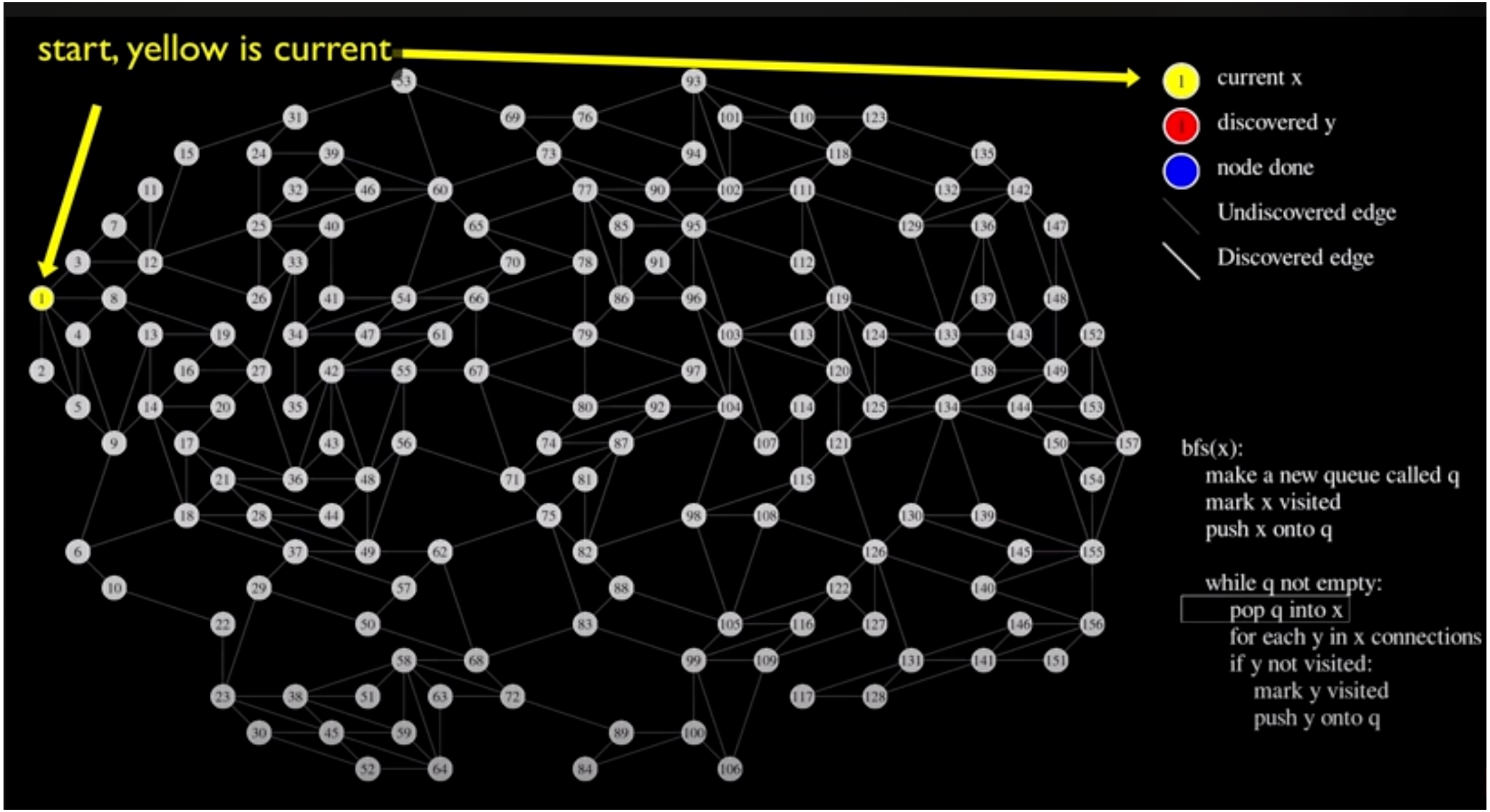
- *Breadth-First Search (BFS)*
- Exploración por niveles
- Se programa con una cola



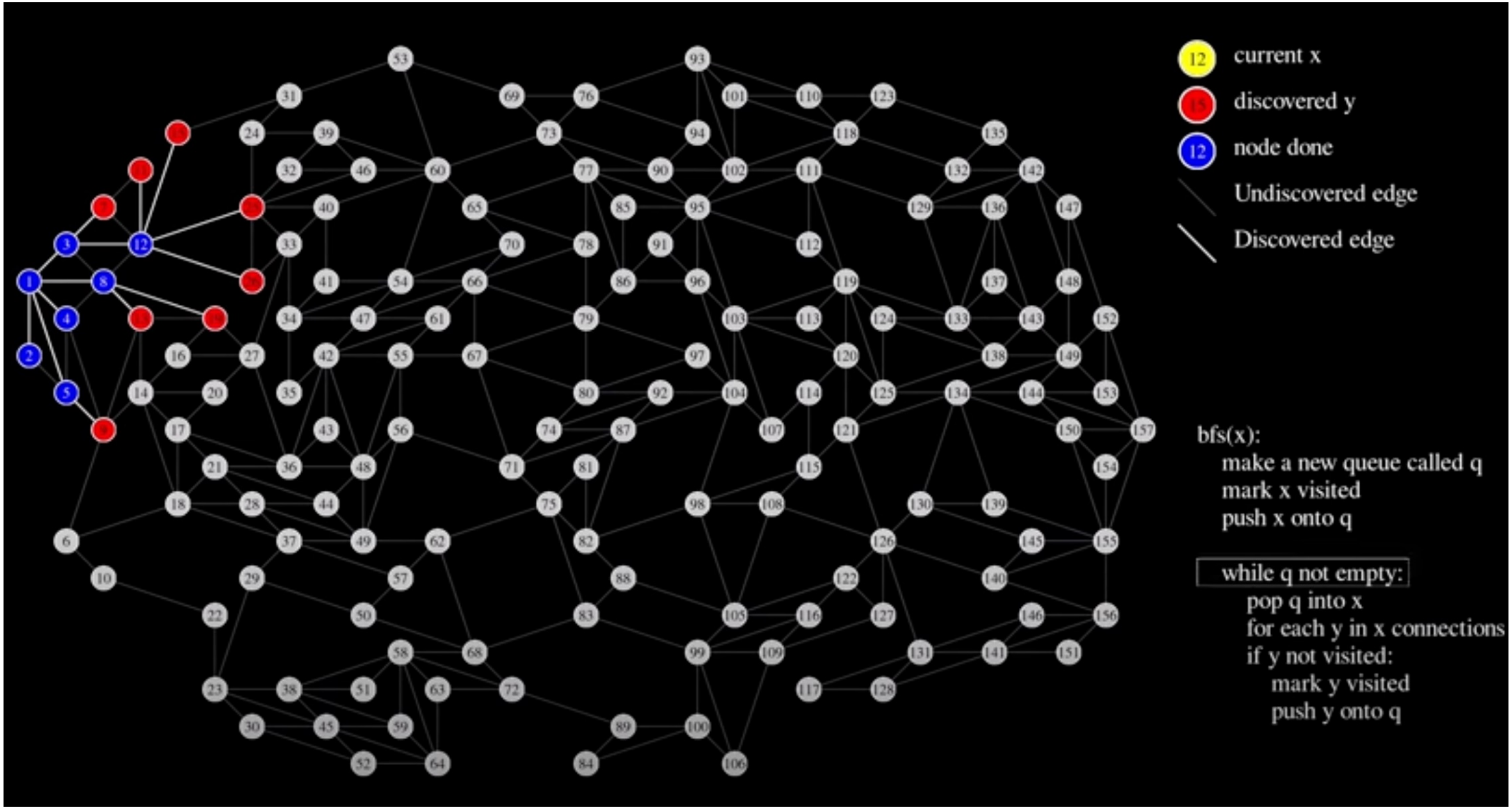
Aplicaciones básicas:

- Distancia mínima (en número de saltos)
- Componentes conectados

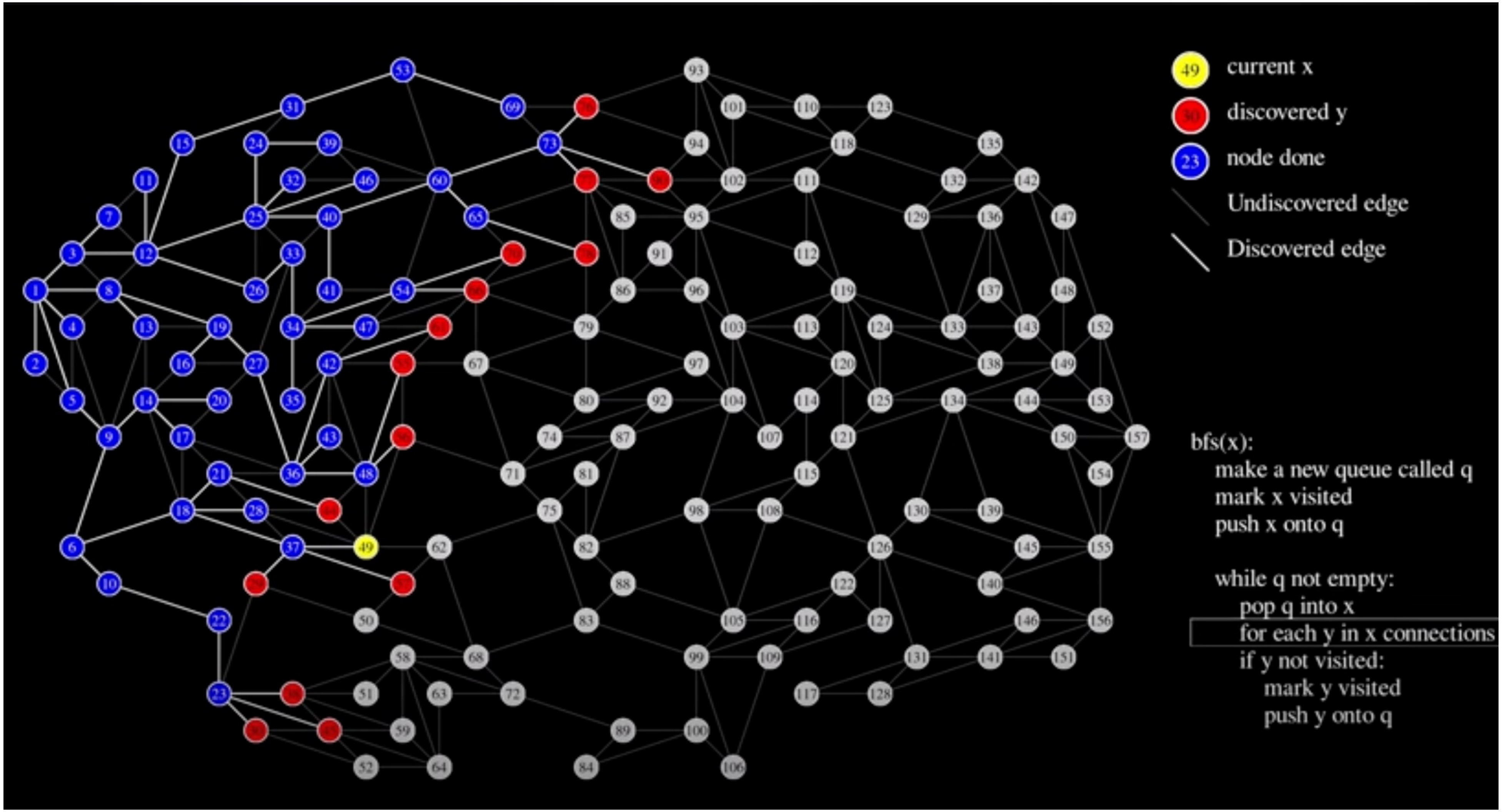
https://www.youtube.com/watch?v=x-VTfcmrLEQ



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



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



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

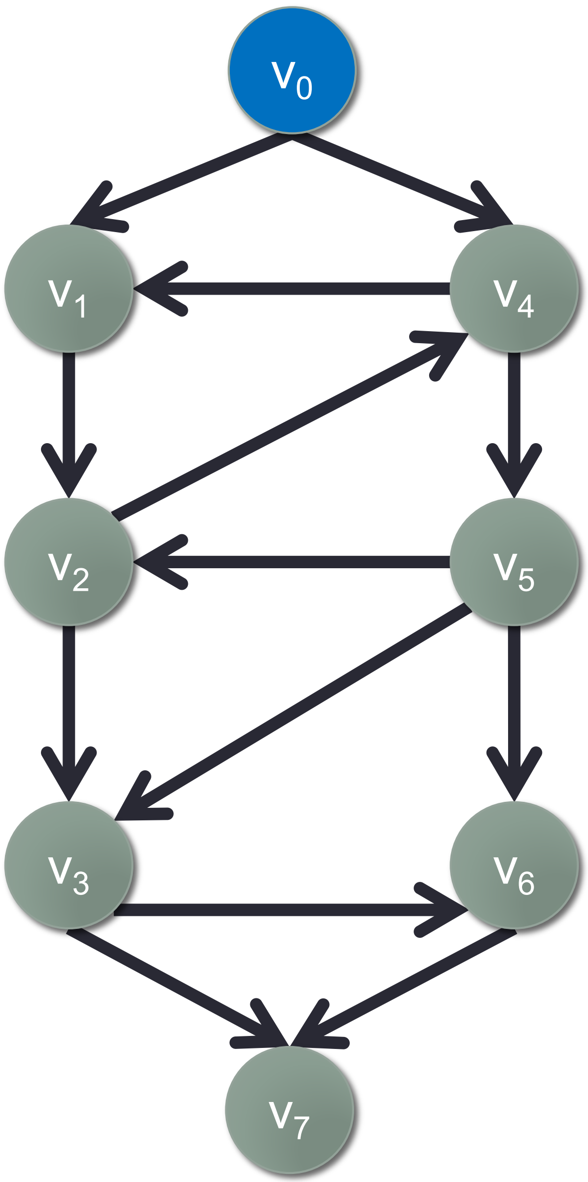
Recorrido a lo ancho (Cola)

```
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)

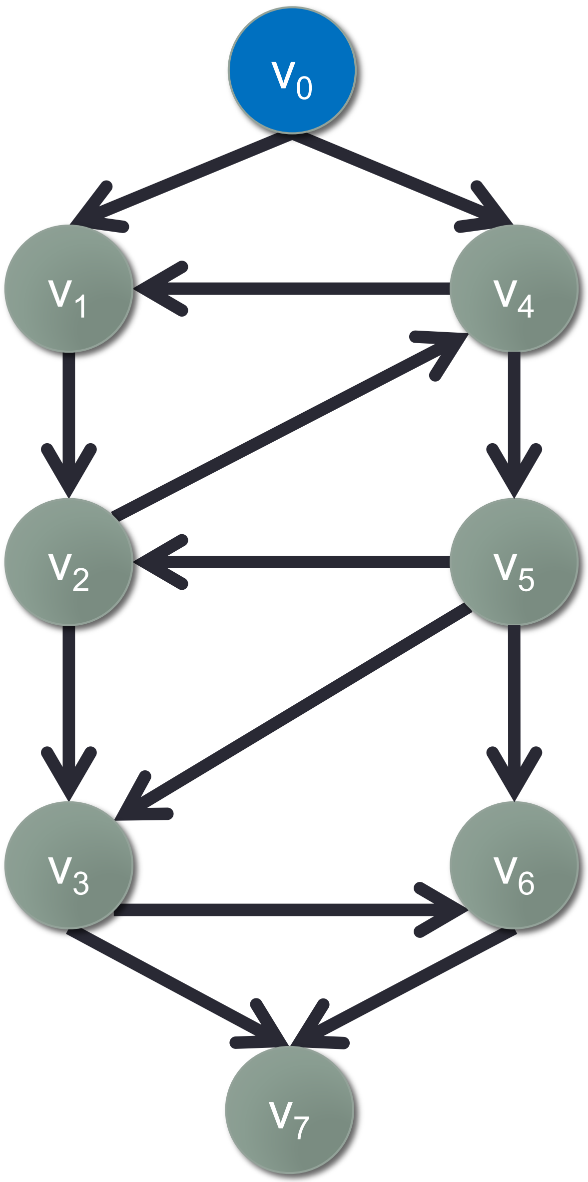
v_0							

Vértices visibles

v_0							

Recorrido a lo ancho (*BFS*)

Paso 1



V_0

Sacamos un elemento de la cola

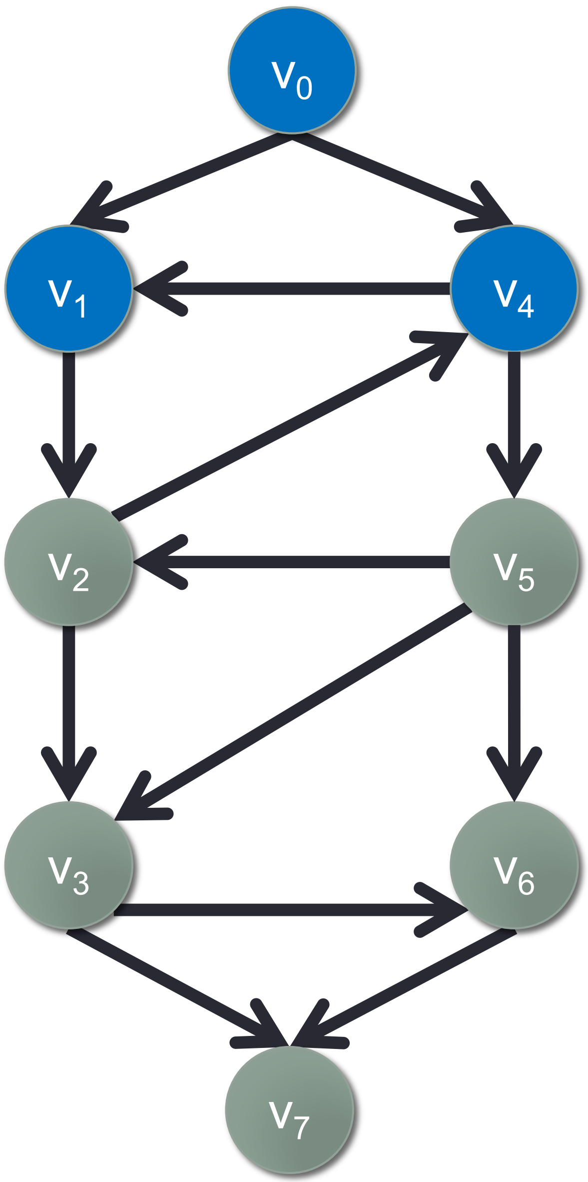
Cola (vértices pendientes de procesar)

Vértices visibles

v_0							

Recorrido a lo ancho (*BFS*)

Paso 2



V_0

Sacamos un elemento de la cola

… marcamos sus vecinos como visibles:

V_1 y V_4

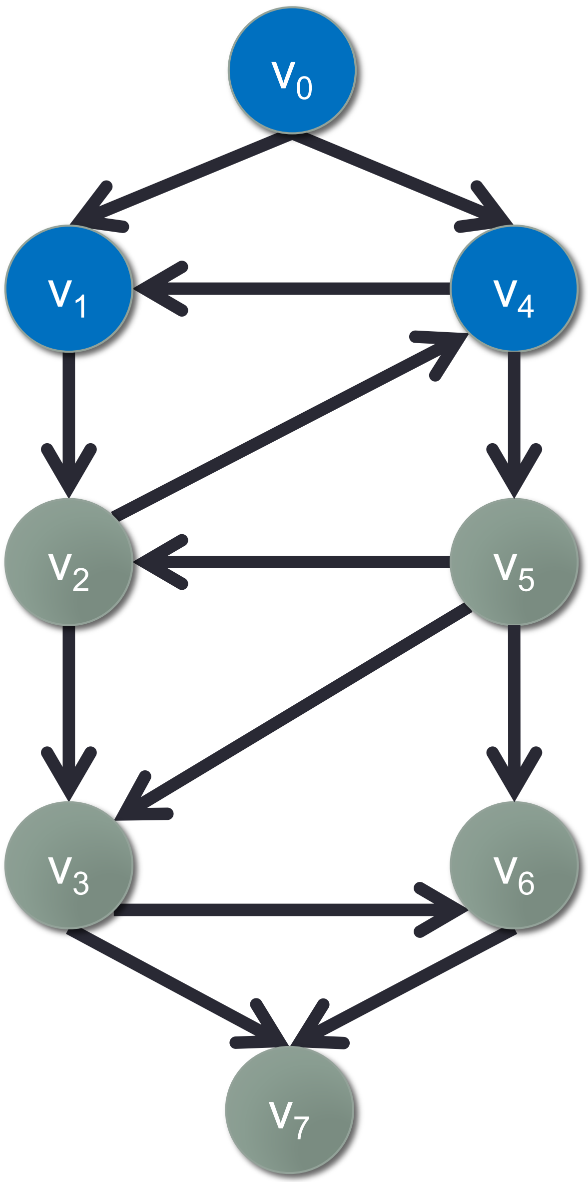
Cola (vértices pendientes de procesar)

Vértices visibles

v_0	v_1			v_4			

Recorrido a lo ancho (*BFS*)

Paso 3



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)

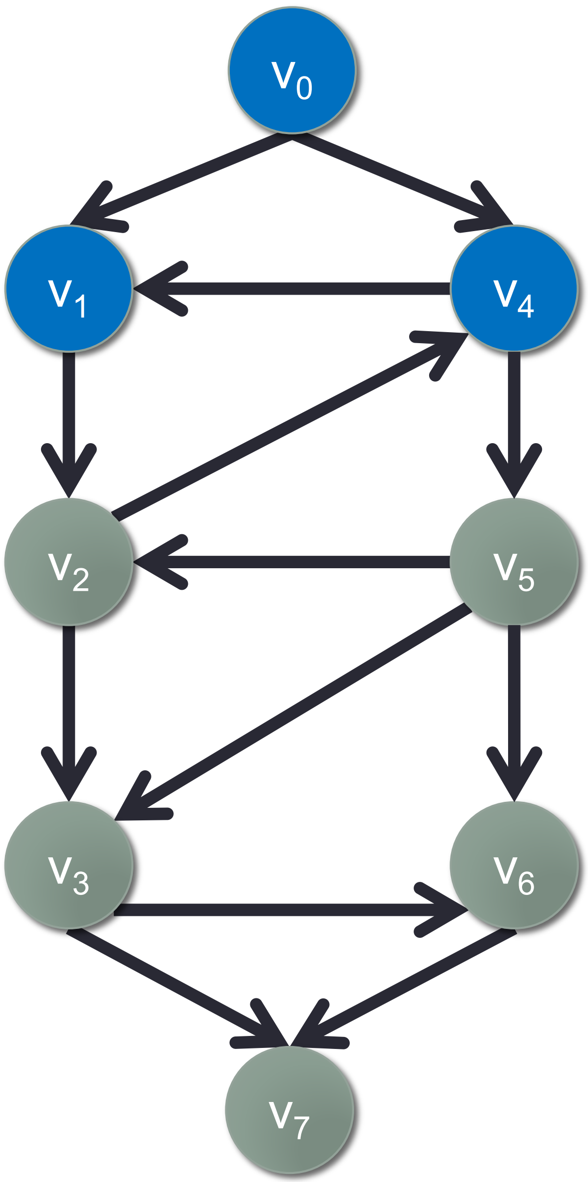
v_1	v_4					

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)

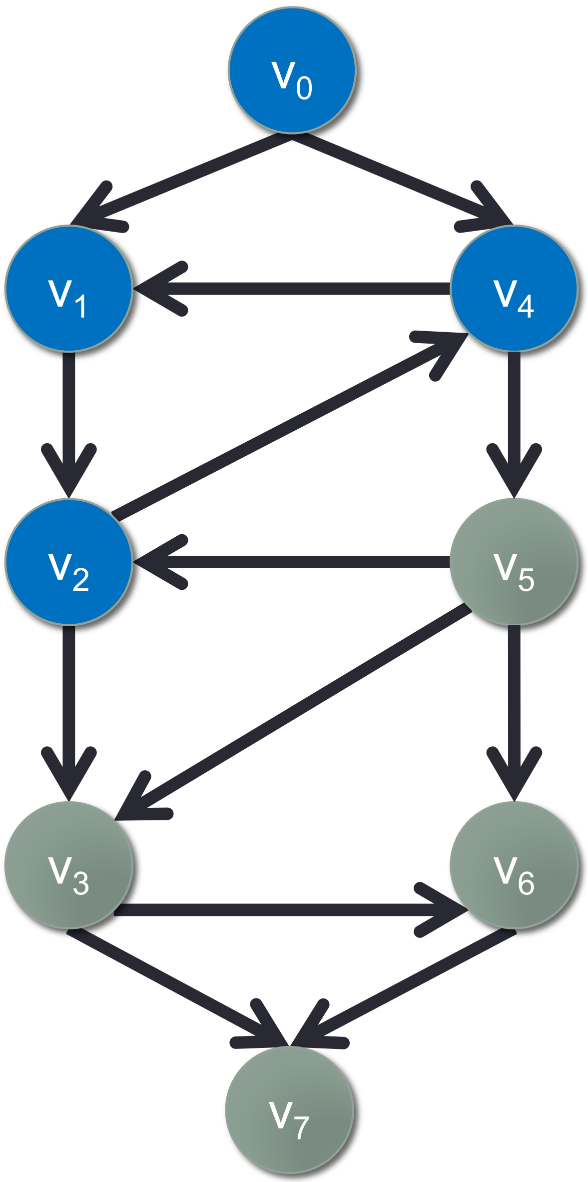
v_4					

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)

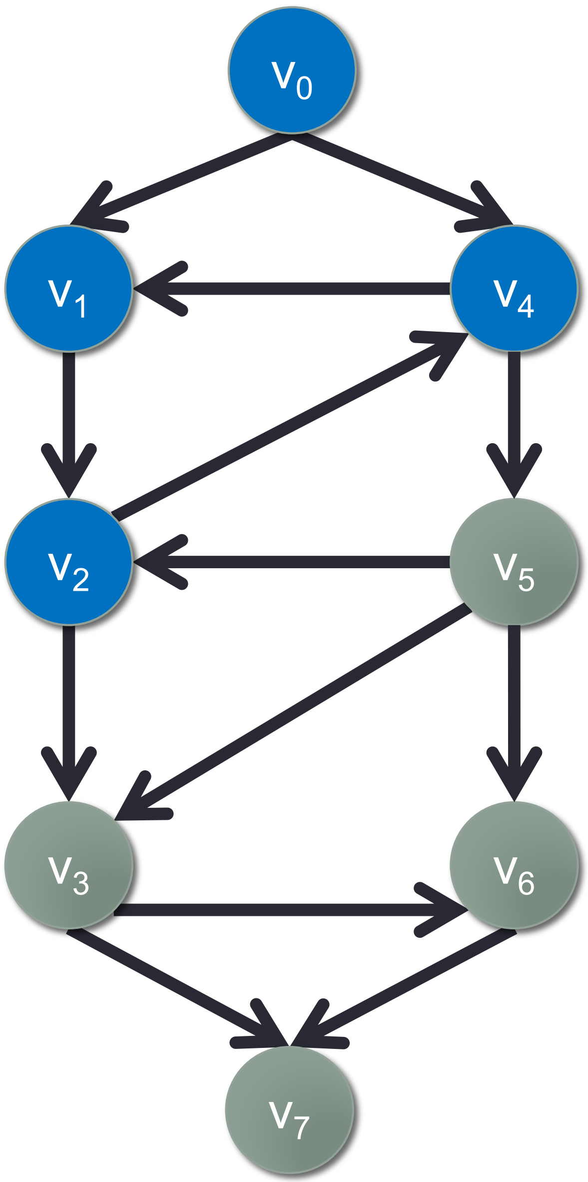
v_4					

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)

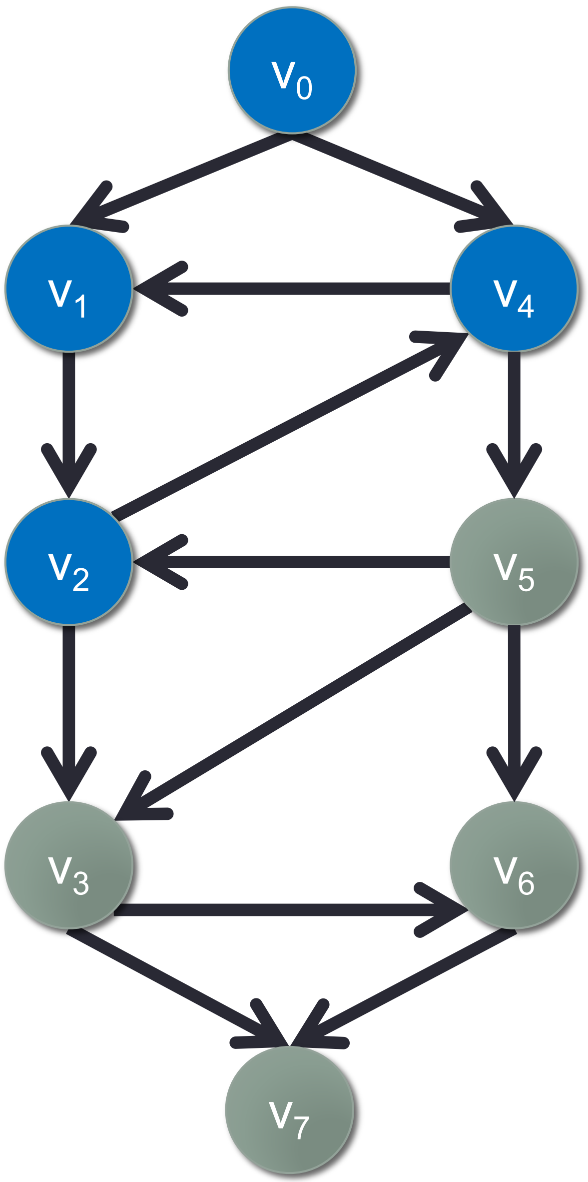
v_4	v_2				

Vértices visibles

v_0	v_1	v_2		v_4			

Recorrido a lo ancho (*BFS*)

Paso 1



V_0 V_1 V_4

Sacamos un elemento de la cola

Cola (vértices pendientes de procesar)

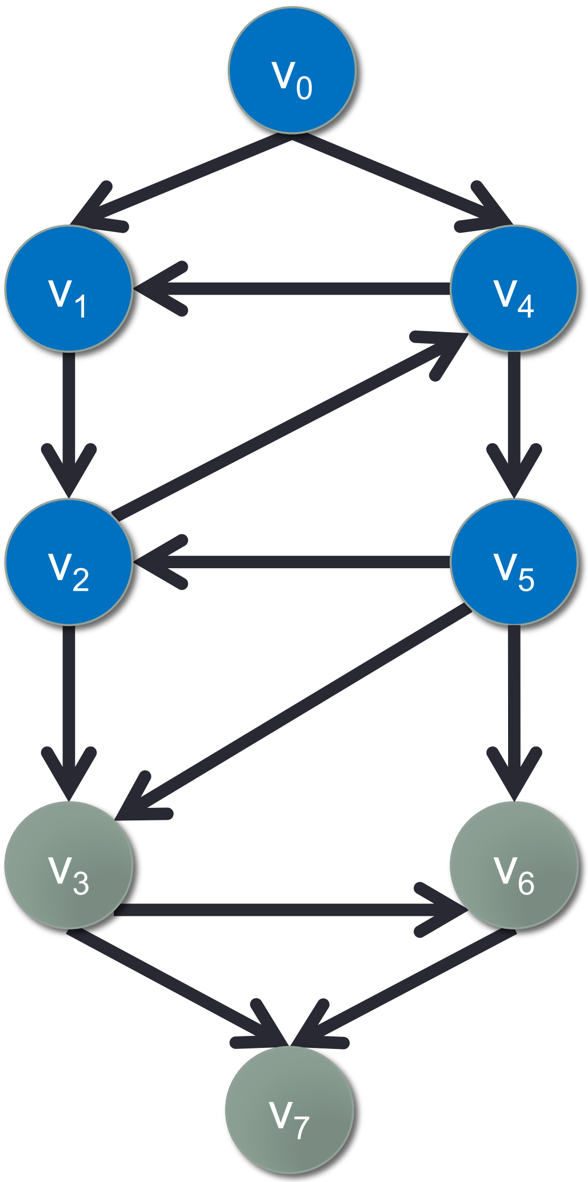
v_2				

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)

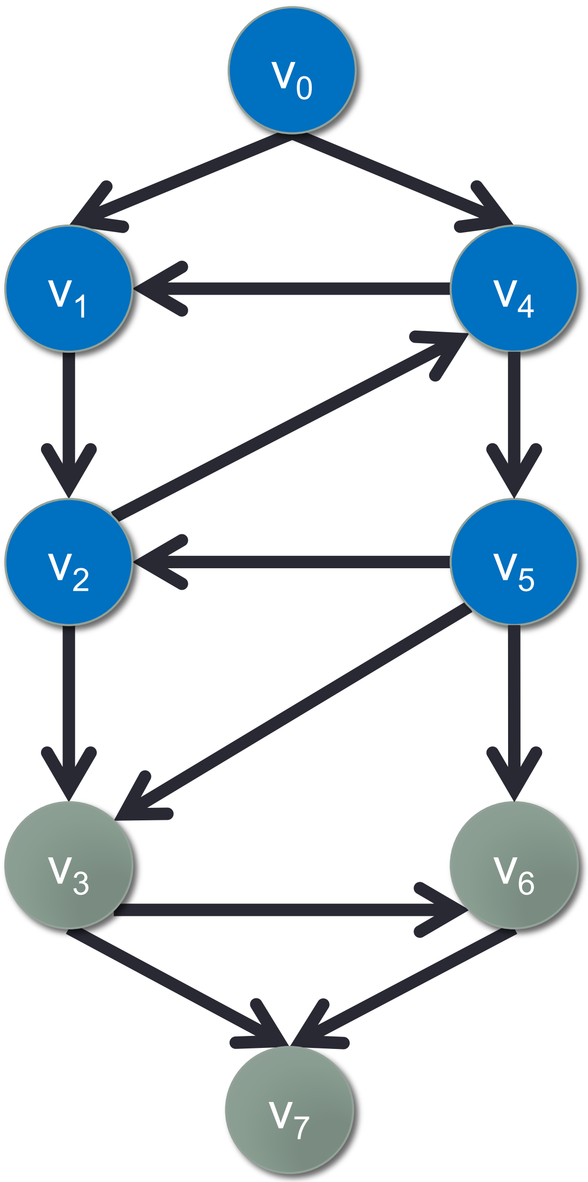
v_2				

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)

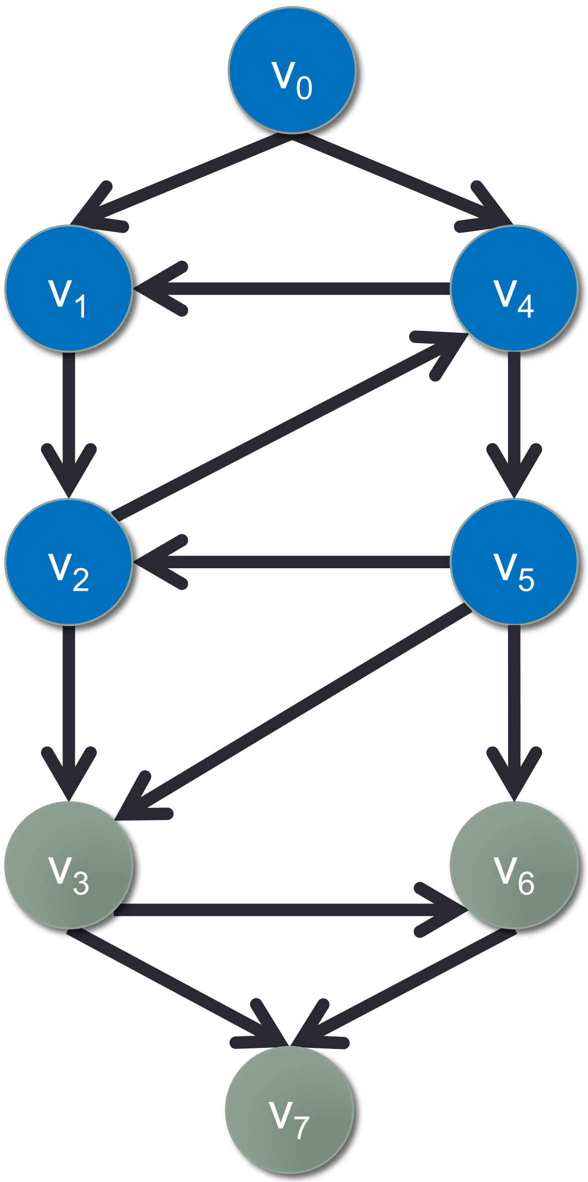
v_2	v_5			

Vértices visibles

v_0	v_1	v_2		v_4	v_5		

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)

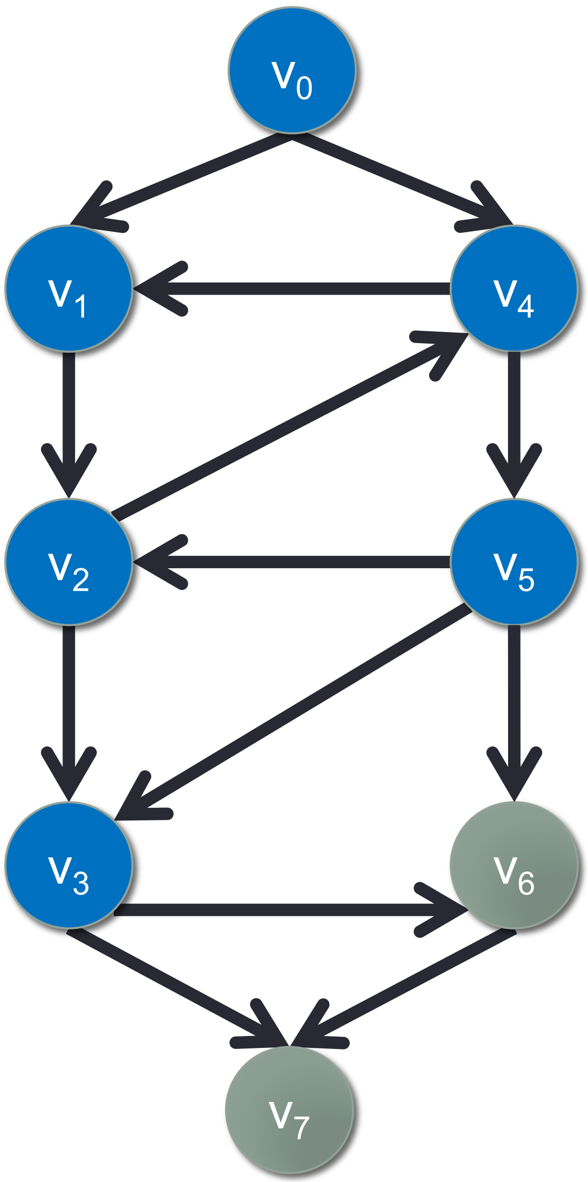
v_5			

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)

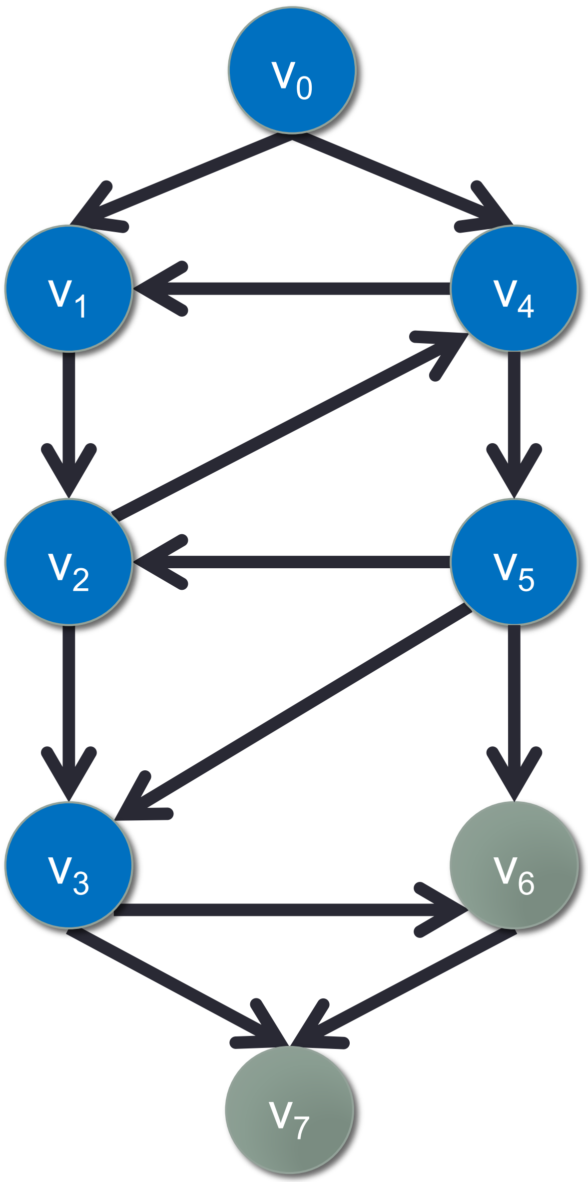
v_5			

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)

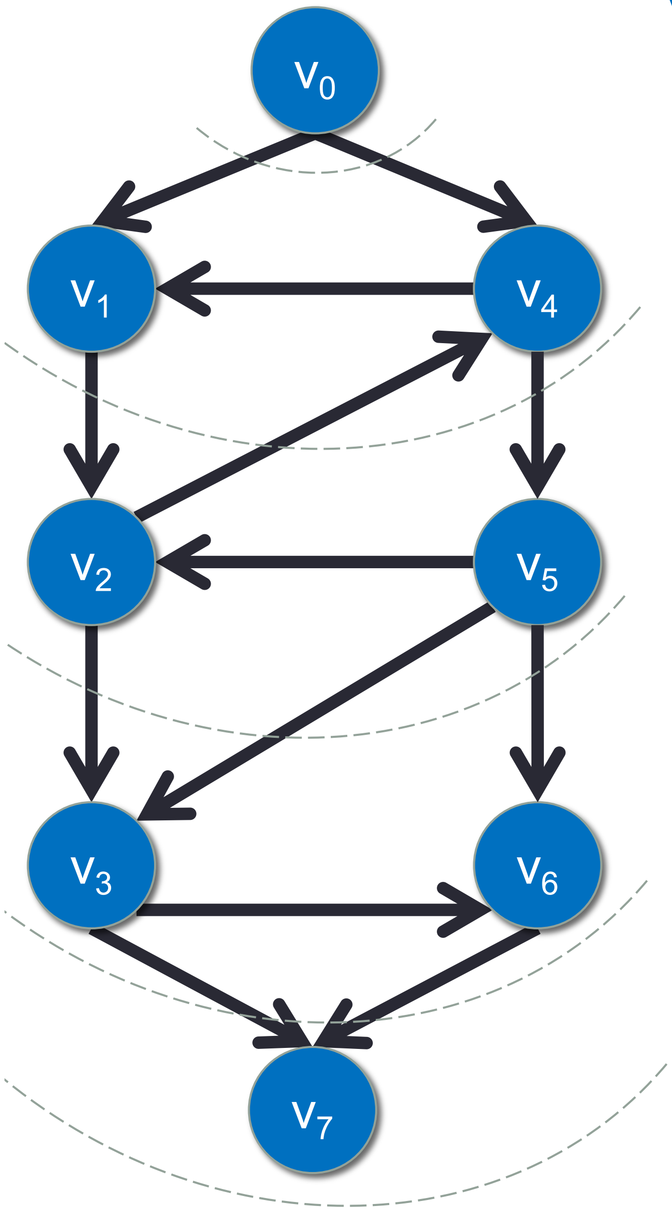
v_5	v_3		

Vértices visibles

v_0	v_1	v_2	v_3	v_4			

Recorrido a lo ancho (*BFS*)

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



… completando el algoritmo éste es el recorrido obtenido

Breadth_First_Search (grafo G, vertice inicial vi)

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

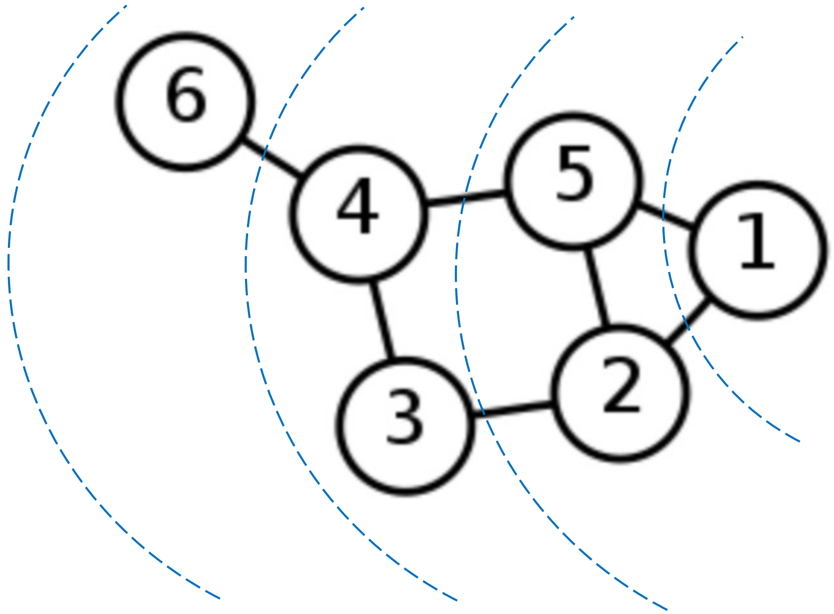
- Mientras not Q.is_empty():
  - v = Q.dequeue()
  - Para todas las aristas (v,w):
    - Si w no es visible:
      - Marcar w como visible
      - Q.enqueue(w)

Estrategias básicas de búsqueda

Recorrido a lo ancho

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

Aplicaciones:
- Distancia mínima (en número de saltos)
- Componentes conectados



Aplicación 1: Distancia Mínima

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

Shortest_Path_Length (grafo G, vertice inicial vi)

- Q = Queue();
- Marcar vi como visitado; 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)

Partimos del algoritmo BFS que acabamos de ver

Aplicación 1: Distancia Mínima

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

Shortest_Path_Length (grafo G, vertice inicial vi)

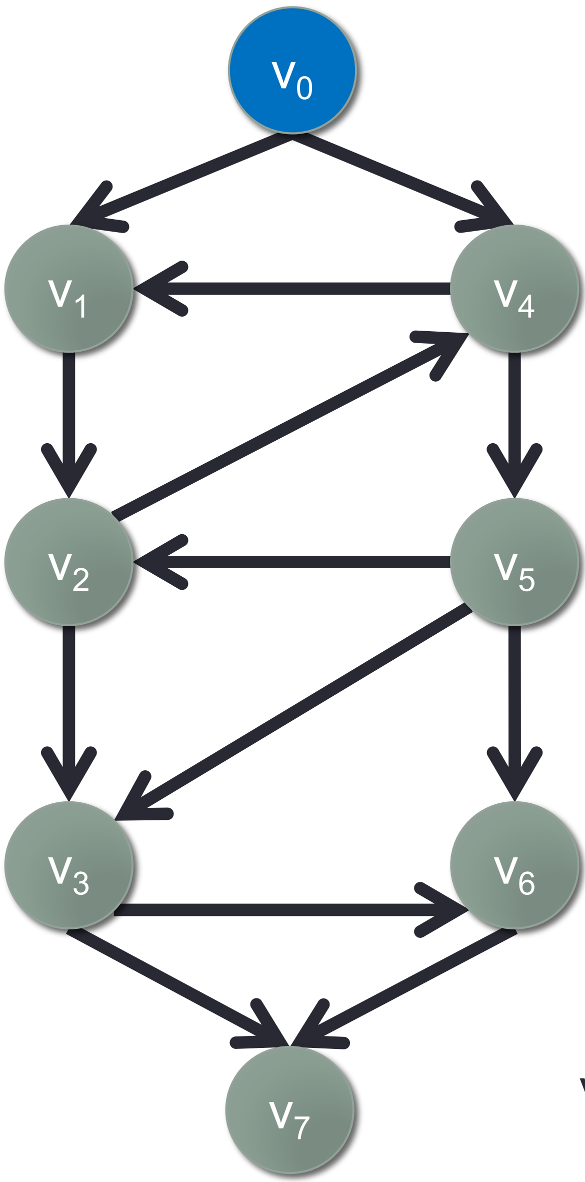
- Q = Queue();
- Marcar vi como visitado; Q.enqueue(vi)
- Dist = [0 si k = vi; infinito si k != vi]

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

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

… y le añadimos el cálculo de la distancia.

Distancia Mínima

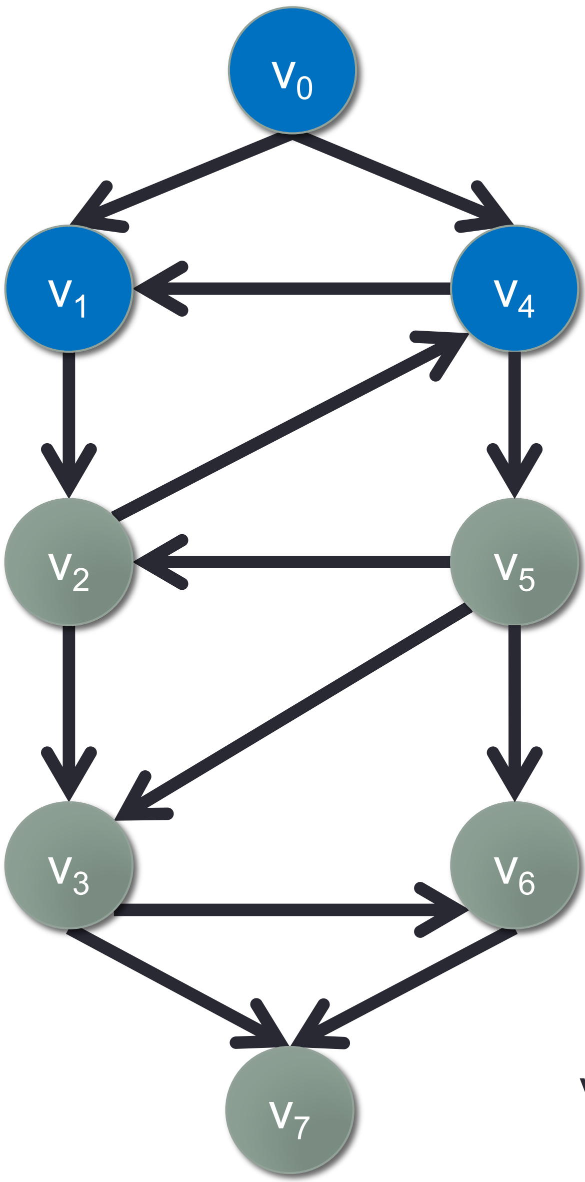


Marcamos el vértice origen como visible
… y lo añadimos a la cola

Distancia

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

Distancia Mínima



Sacamos un elemento de la cola: V_0

… marcamos sus vecinos visibles:

V_1 y V_4

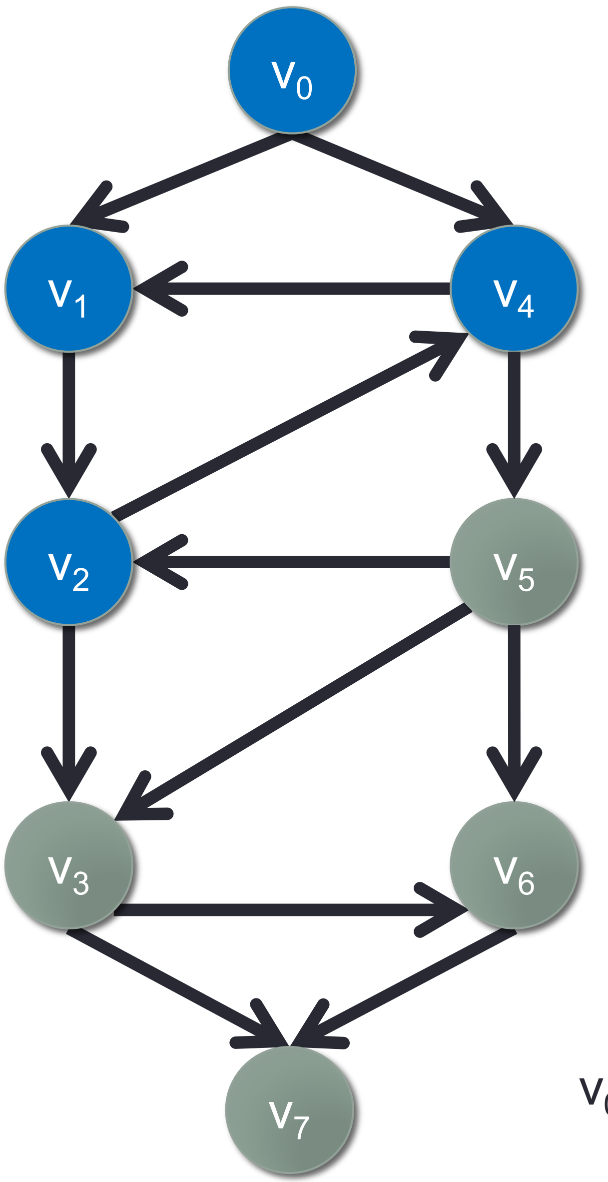
… y actualizamos $dist(V_1)$, $dist(V_4)$

$$dist(V_1) = dist(V_0) + 1$$
$$dist(V_4) = dist(V_0) + 1$$

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

Distancia Mínima



Sacamos un elemento de la cola: V_1

… marcamos sus vecinos como visibles:

V_2

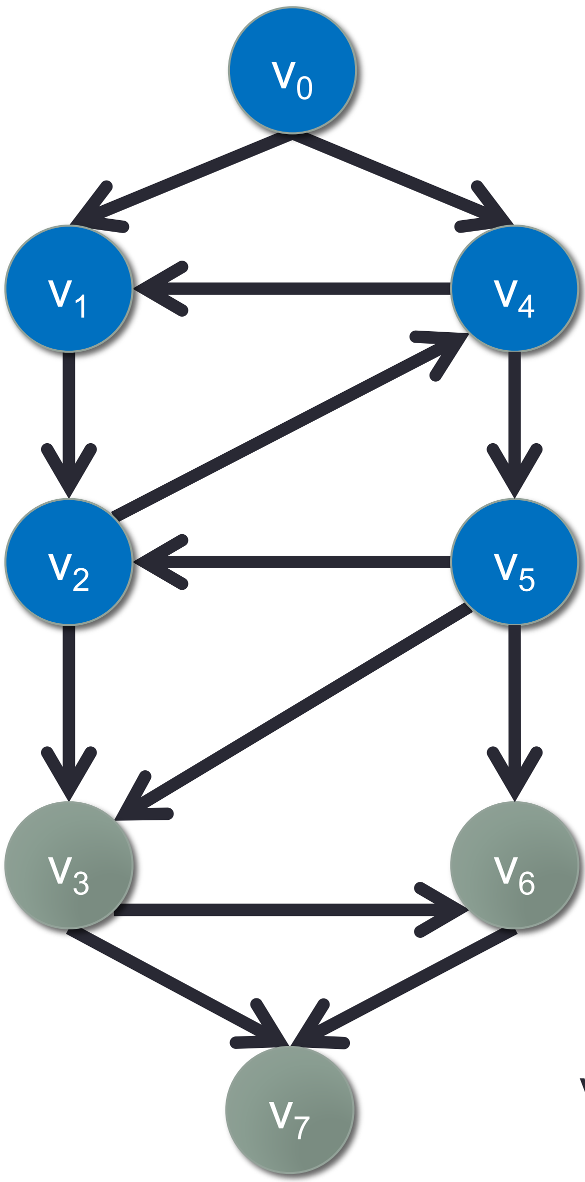
… y actualizamos $dist(V_2)$

$$dist(V_2) = dist(V_1) + 1$$

Distancia

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

Distancia Mínima



Sacamos un elemento de la cola: V_4

… marcamos sus vecinos como visibles:

V_5

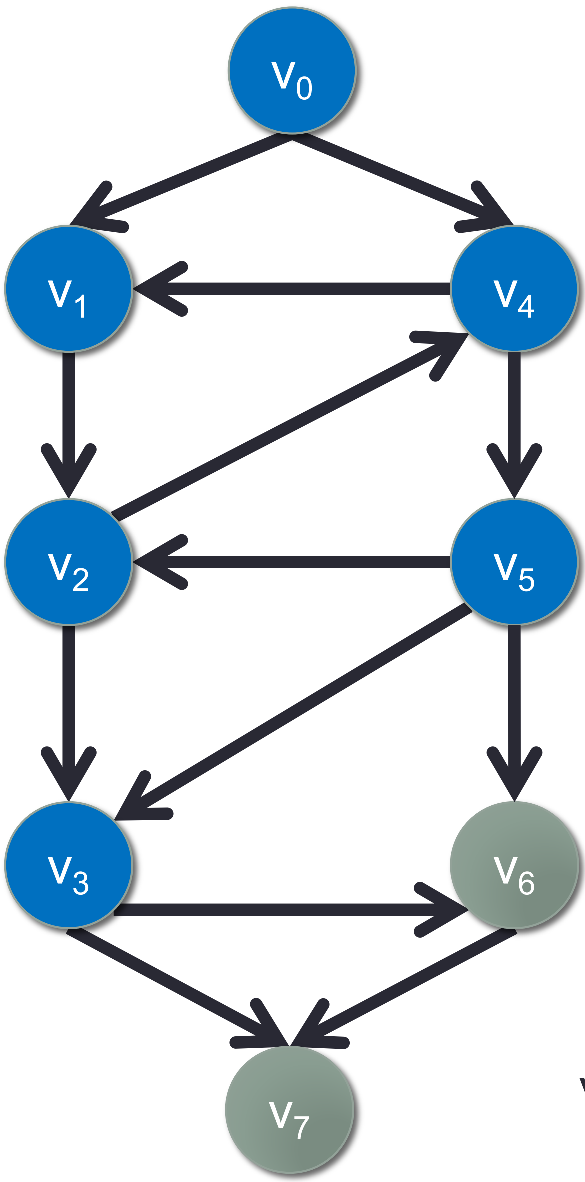
… y actualizamos dist(V_5)

$$dist(V_5) = dist(V_4) + 1$$

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

Distancia Mínima



Sacamos un elemento de la cola: V_2

… marcamos sus vecinos como visibles:

V_3

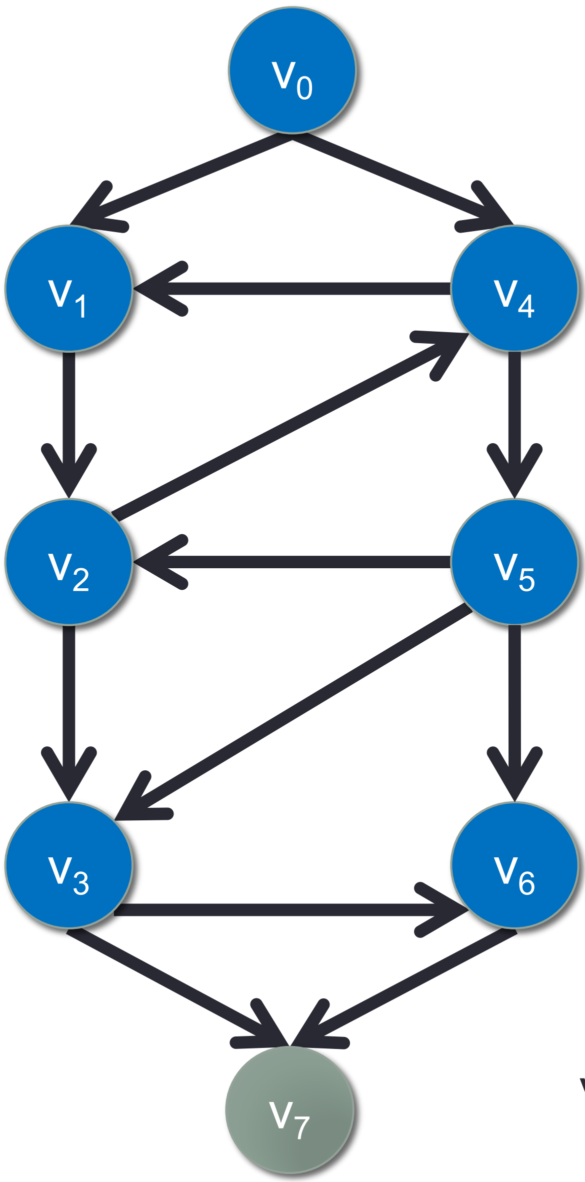
… y actualizamos dist(V_3)

$$dist(V_3) = dist(V_2) + 1$$

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

Distancia Mínima



Sacamos un elemento de la cola: V_5

… marcamos sus vecinos como visibles:

V_6

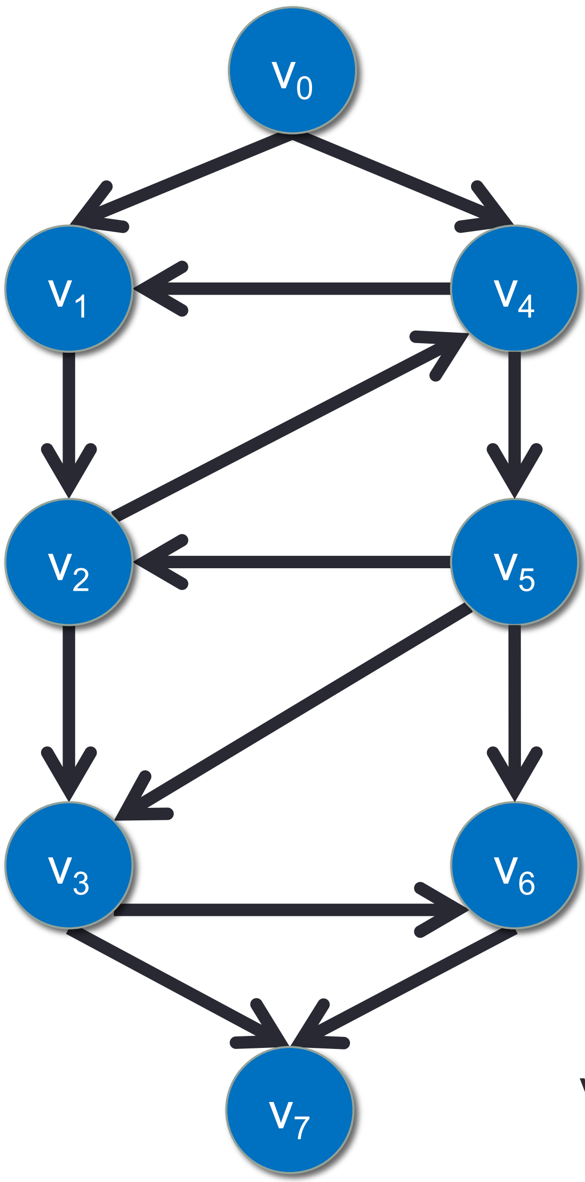
… y actualizamos dist(V_6)

$$dist(V_6) = dist(V_5) + 1$$

Distancia

	V_0	V_1	V_2	V_3	V_4	V_5	V_6	V_7
V_0	**0**	**1**	**2**	**3**	**1**	**2**	**3**	**inf**

Distancia Mínima



Sacamos un elemento de la cola: V_3

… marcamos sus vecinos como visibles:

V_7

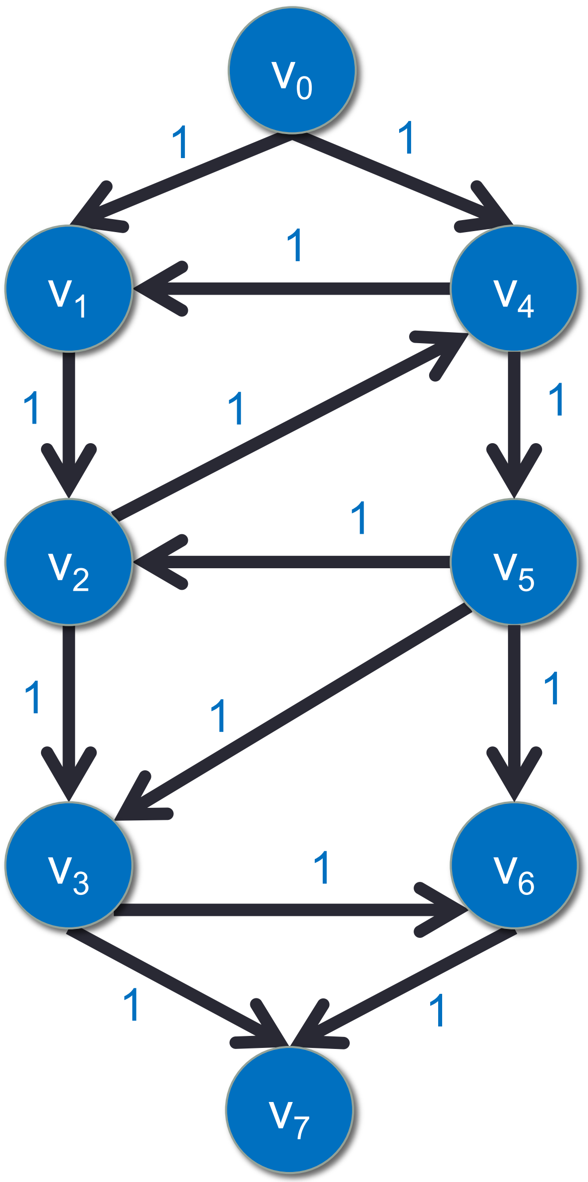
… y actualizamos dist(V_7)

$$dist(V_7) = dist(V_3) + 1$$

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

¿Distancia Mínima en un grafo con pesos?

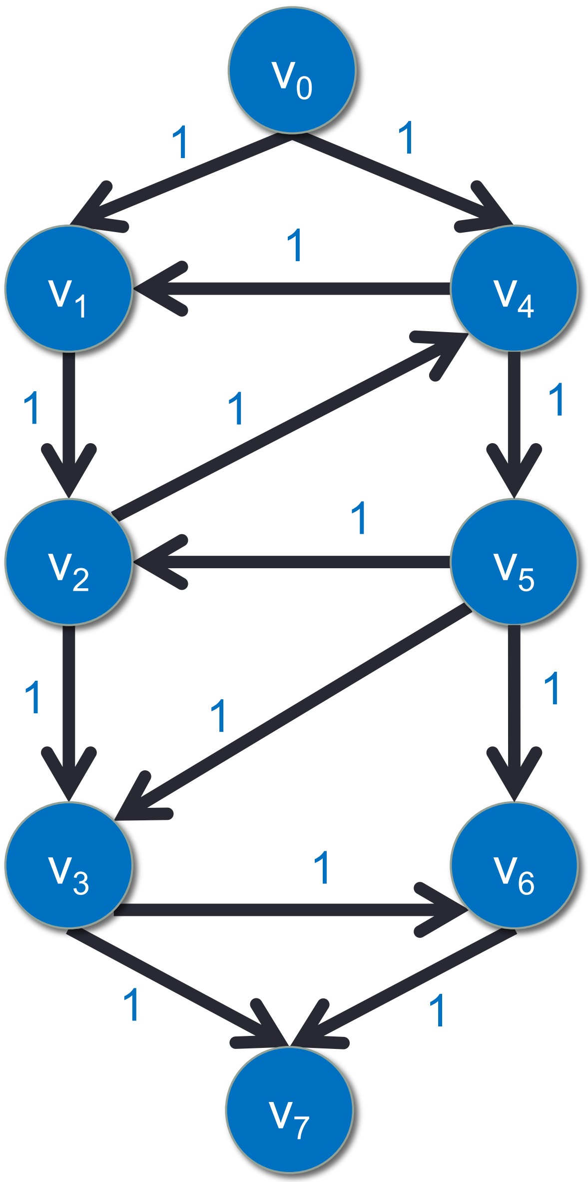


Limitación de este algoritmo

Sólo funciona bien cuando todas las aristas tienen peso 1

¿ Podríamos adaptarlo para calcular la distancia minima en un grafo con pesos ?

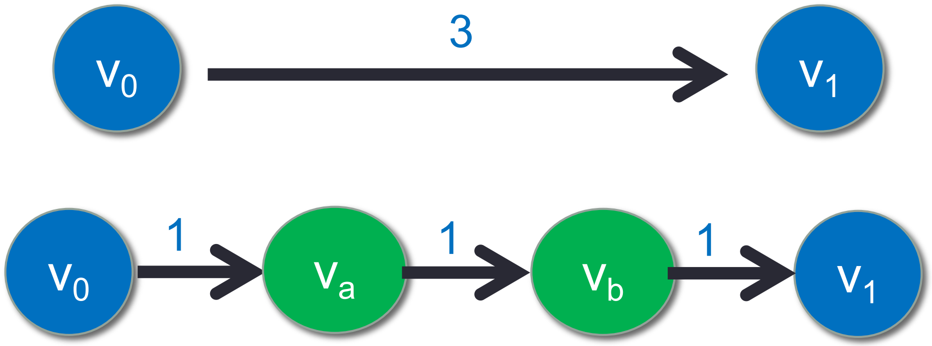
¿Distancia Mínima en un grafo con pesos?



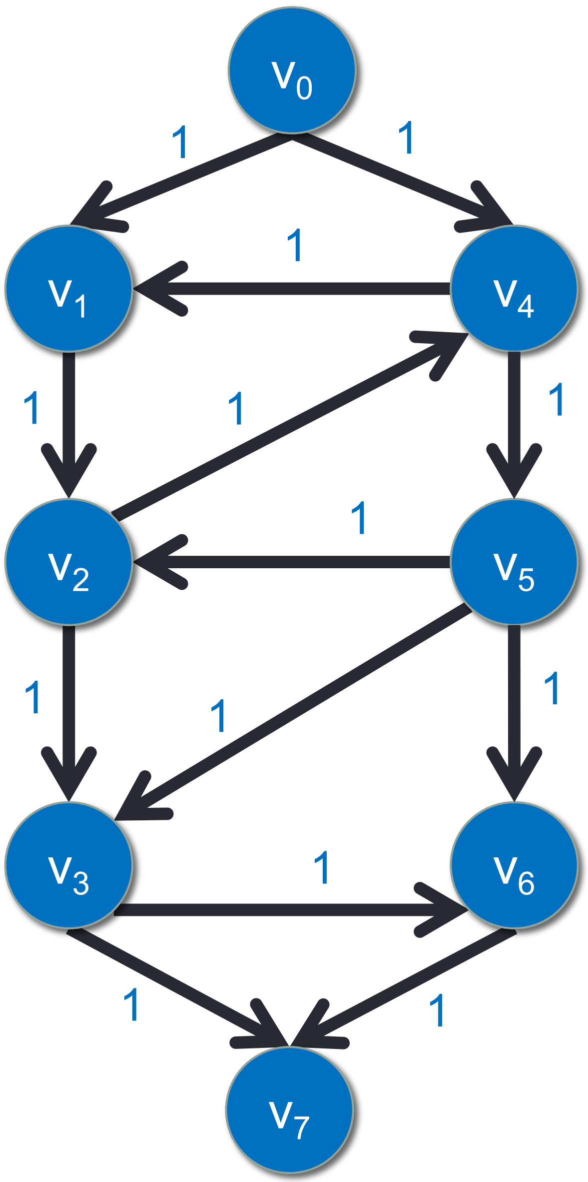
Limitación de este algoritmo

Sólo funciona bien cuando todas las aristas tienen peso 1

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



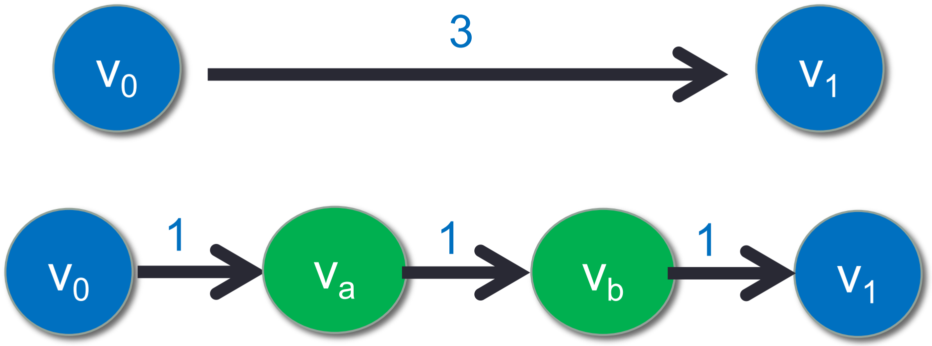
¿Distancia Mínima en un grafo con pesos?



Limitación de este algoritmo

Sólo funciona bien cuando todas las aristas tienen peso 1

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



¿ Conocemps alguna 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: $dt(v_i) = D_a + d(a,v_i)$. 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 $dt(vi) < Dvi \rightarrow Dvi = dt(vi)$

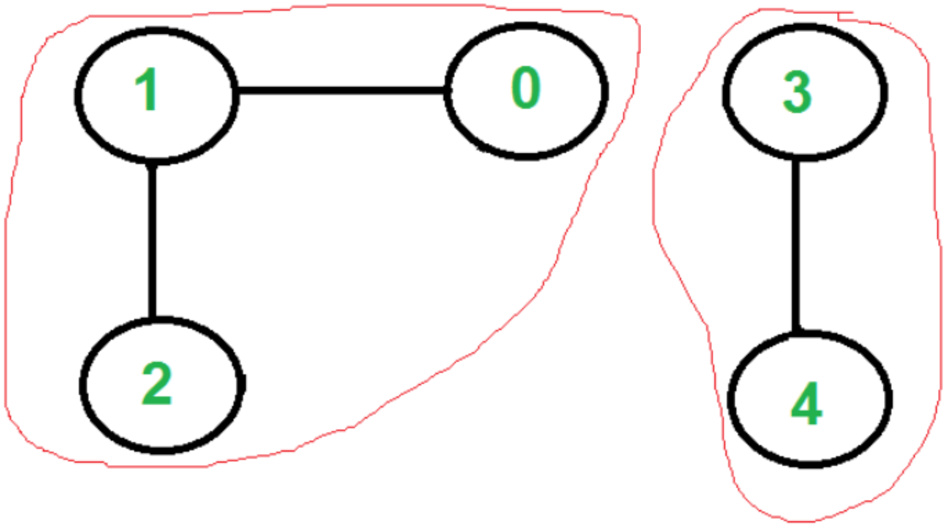
5) Se marca como completo el nodo a.

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.

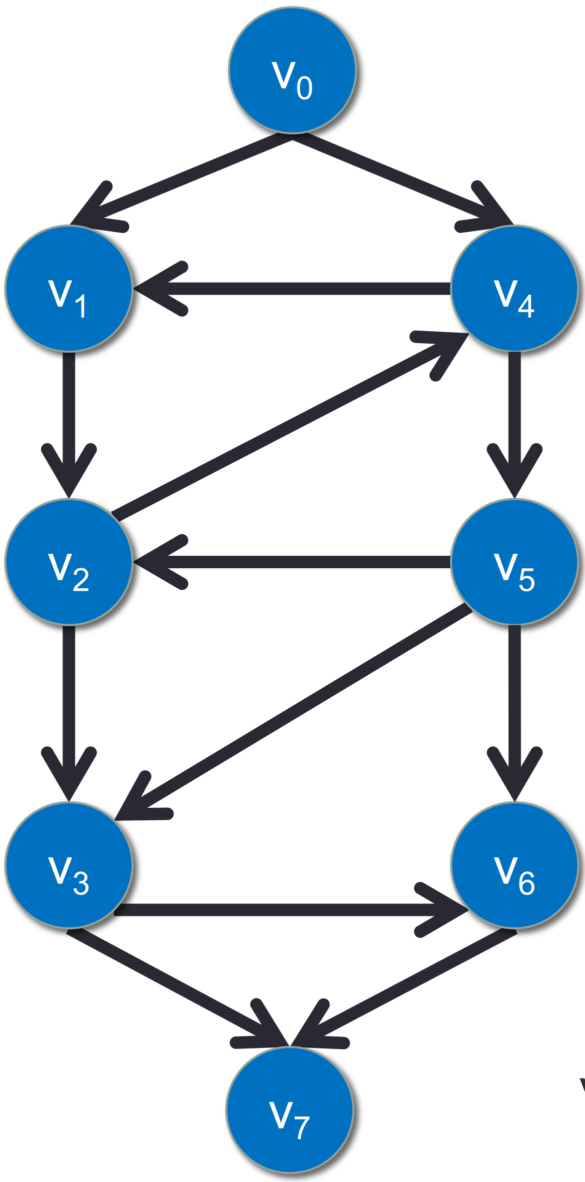
Aplicación 2: Conectividad

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



- Marcamos todos los vertices como no-visibles
- Recorremos todos los vertices J
  - Si J no está visible:
    - **Breadth_First_Search** (Grafo, J)

Práctica Semana 2



Utilizando Python y NetworkX, programa el algoritmo que calcula la distancia mínima (en número de saltos).

python™

NetworkX
Network Analysis in Python

Distancia

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