Diseño y Análisis de Algoritmos

TEMA 2. ALGORITMOS EN GRAFOS

Universidad
Rey Juan Carlos

Tema 2. Exploración de grafos

Introducción

- Definiciones
- Recorrido en profundidad
- Recorrido en anchura
- Algoritmos sobre grafos
  - Recorridos de grafos infinitos
  - Ordenación topológica
  - Componentes fuertemente conexas
  - Puntos de articulación

Tema 2. Exploración de grafos

Definiciones

- Se suele **definir** un grafo $G=(V,E)$ como un conjunto de vértices V y de aristas $E \subseteq V \times V$.
- Usualmente, la **complejidad** de los algoritmos sobre grafos suele medirse en función del
  - número de vértices: $|V| = n$
  - número de aristas: $|E| = m$
- *Convenios de notación*: Cuando nos refiramos a estas cantidades en notación asintótica, sustituiremos $|V|$ y $|E|$ por V y E.

  Ejemplo: $O(VE)$ en lugar de $O(|V||E|)$

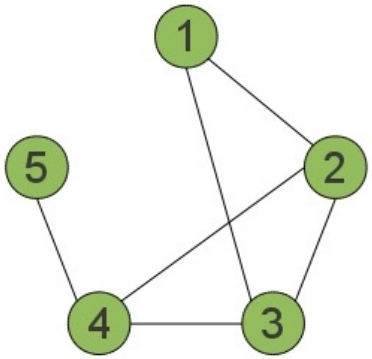
Tema 2. Exploración de grafos

Definiciones

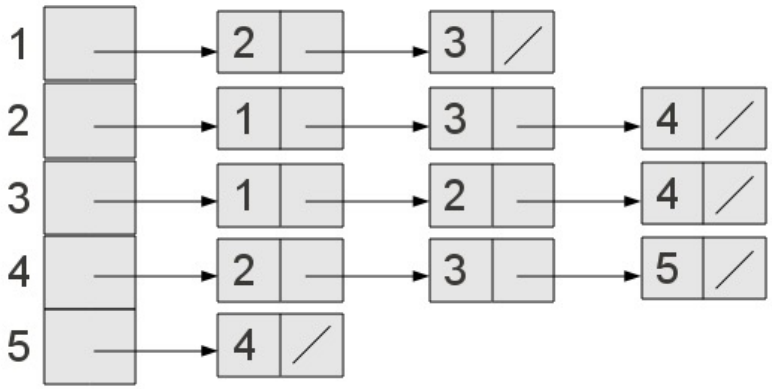
- Existen dos **representaciones** típicas de grafos:
  - Matriz de adyacencia
  - Lista de adyacencia

Grafo no dirigido:

Grafo



Lista de adyacencias



Matriz de adyacencias

	1	2	3	4	5
1	0	1	1	0	0
2	1	0	1	1	0
3	1	1	0	1	0
4	0	1	1	0	1
5	0	0	0	1	0

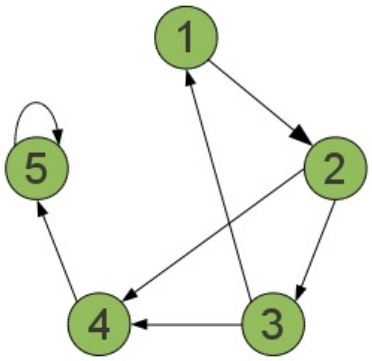
Tema 2. Exploración de grafos

Definiciones

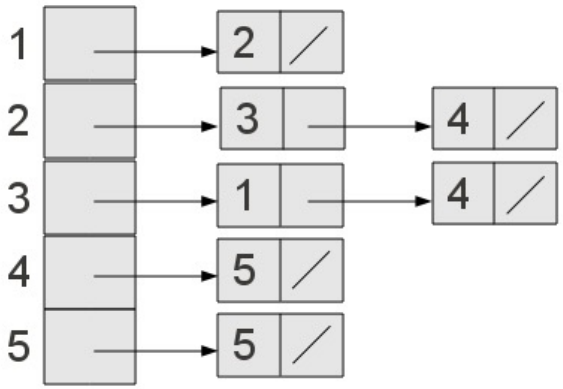
- Existen dos **representaciones** típicas de grafos:
  - Matriz de adyacencia
  - Lista de adyacencia

Grafo dirigido:

Grafo



Lista de adyacencias



Matriz de adyacencias

	1	2	3	4	5
1	0	1	0	0	0
2	0	0	1	1	0
3	1	0	0	1	0
4	0	0	0	0	1
5	0	0	0	0	1

Tema 2. Exploración de grafos

Introducción

Lista de adyacencia:

- Representación **compacta** para grafos dispersos ($|E| << |V|^2$)
- **No aseguran** un acceso rápido a la hora de comprobar si hay una arista entre dos vértices dados

Matriz de adyacencia:

- **Aseguran** un acceso rápido a la hora de comprobar si hay una arista entre dos vértices dados
- Se requiere una **memoria** de $\Theta(V^2)$, y no depende de la densidad del grafo
- Para grafos **densos** (aquéllos en los que $|E| \cong |V|^2$) ya no se desperdicia tanta memoria

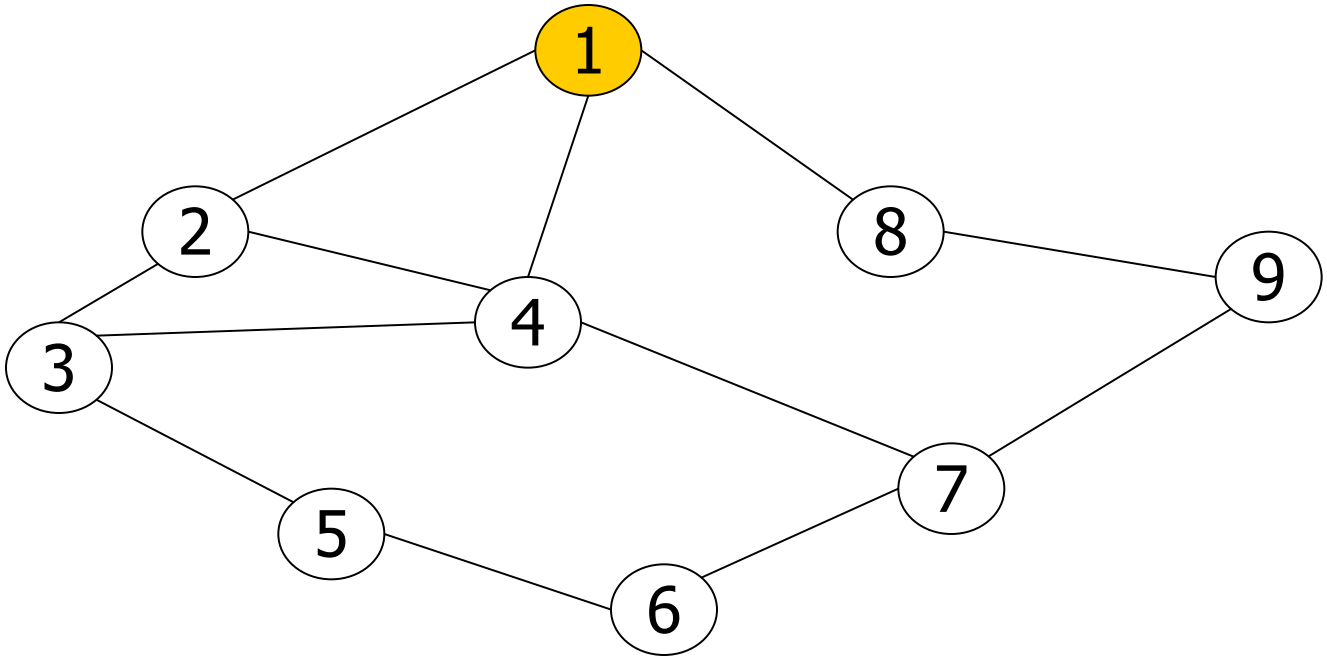
Tema 2. Exploración de grafos

Recorrido en profundidad

- El recorrido en profundidad (*depth-first search*) tiene como estrategia **profundizar** en el grafo siempre que sea posible
  - Dado un vértice, antes de **visitar** su *hermano* se visita a su *hijo* (equivalente a un recorrido preorden)
  - Suele implementarte se manera **recursiva**
  - Se tiene que incluir un **conjunto** de vértices visitados para evitar **ciclos** en la búsqueda

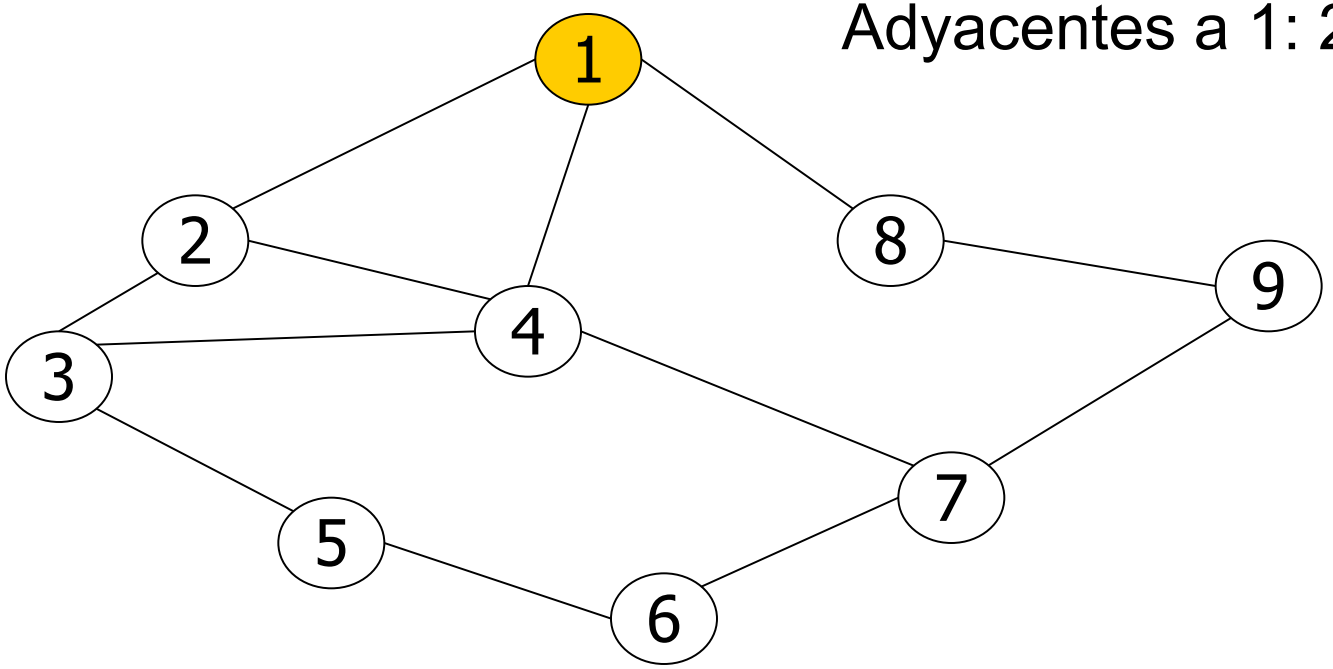
Tema 2. Exploración de grafos

Recorrido en profundidad



Tema 2. Exploración de grafos

Recorrido en profundidad

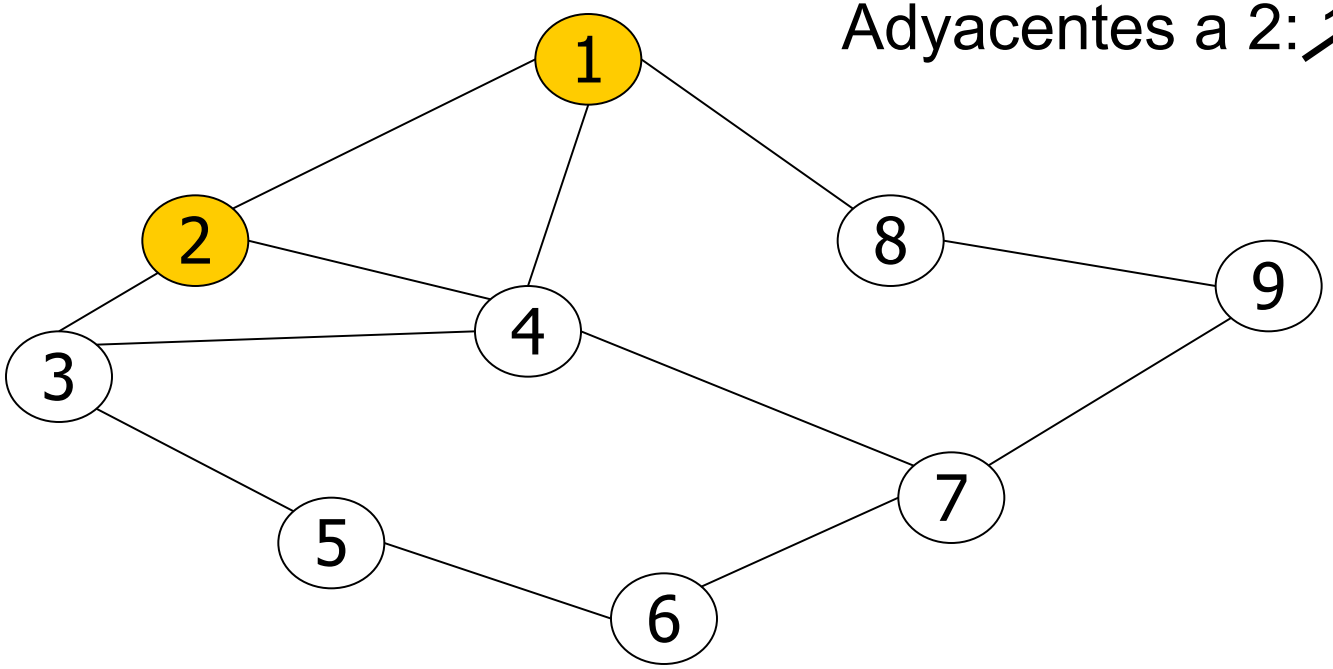


Adyacentes a 1: 2, 4, 8

Recorrido: 1, 2

Tema 2. Exploración de grafos

Recorrido en profundidad

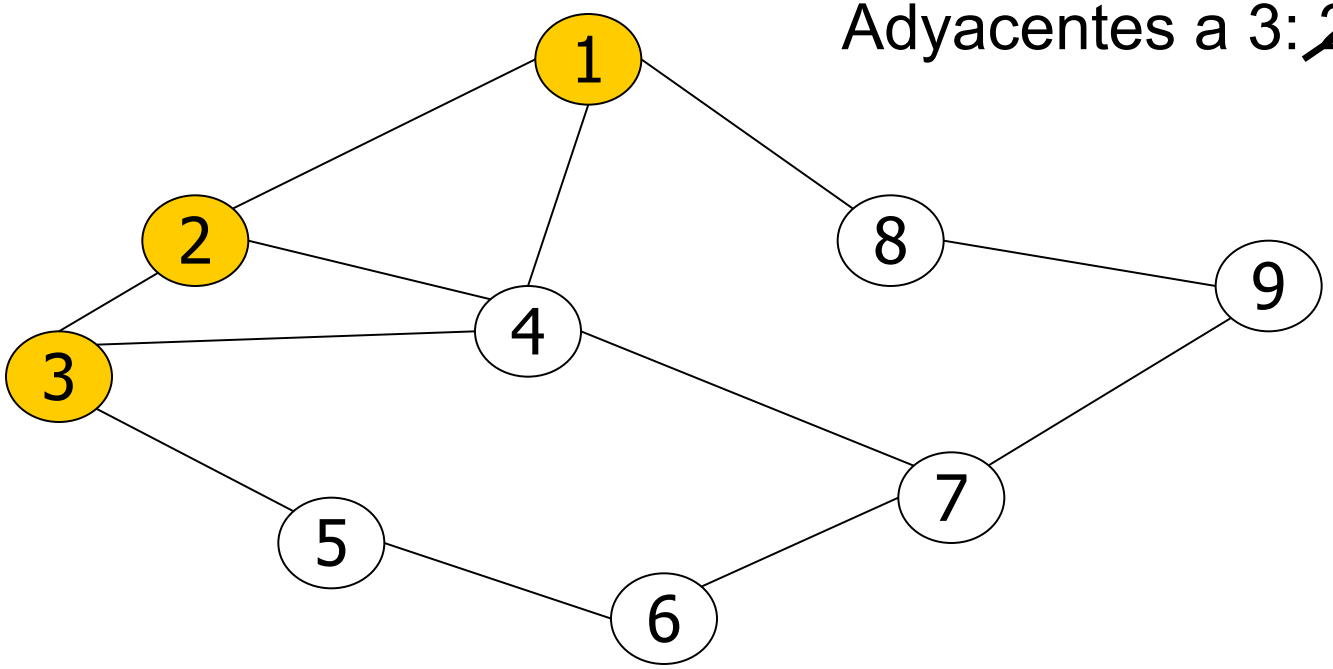


Adyacentes a 2: ~~1~~, 3, 4

Recorrido: 1, 2, 3

Tema 2. Exploración de grafos

Recorrido en profundidad

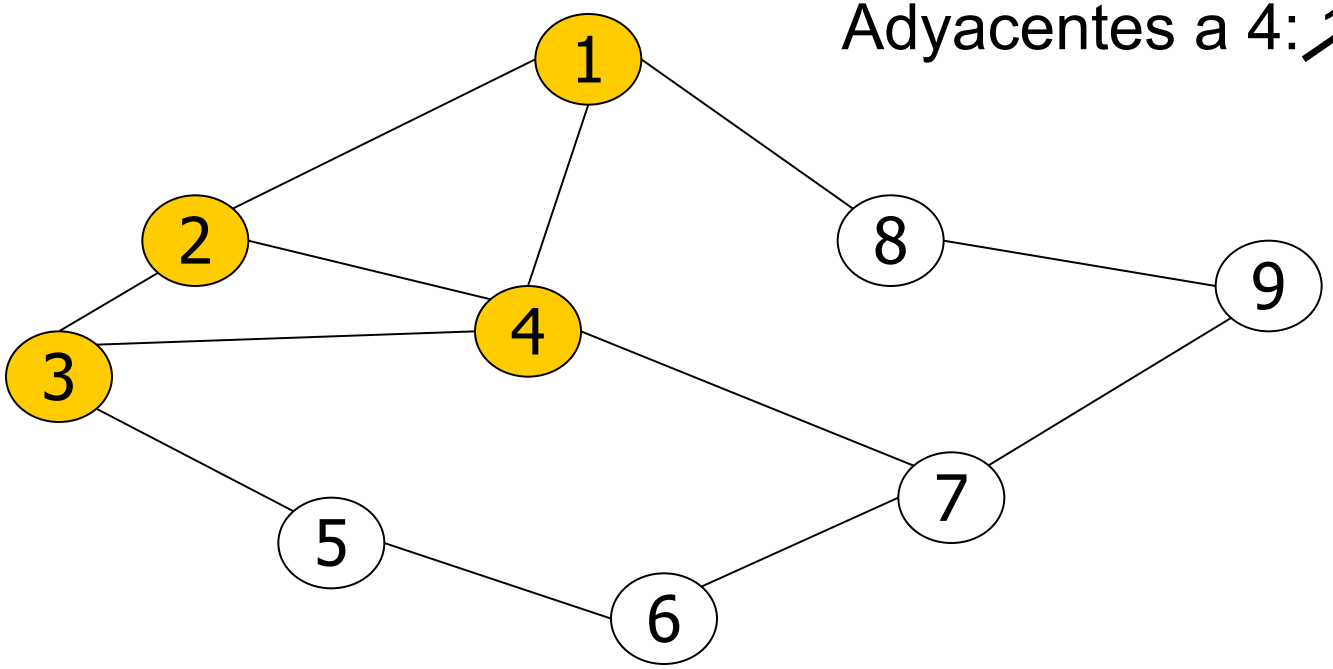


Adyacentes a 3: ~~2~~, 4, 5

Recorrido: 1, 2, 3 , 4

Tema 2. Exploración de grafos

Recorrido en profundidad

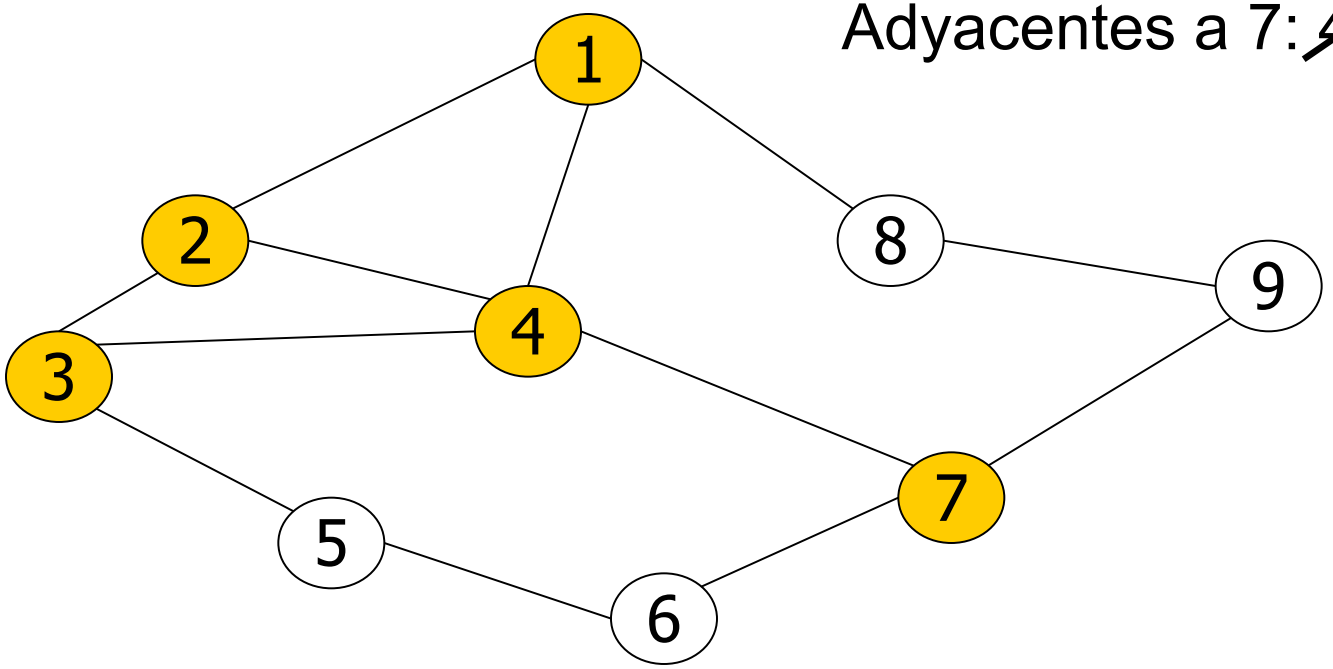


Adyacentes a 4: ~~1~~, ~~2~~, ~~3~~, 7

Recorrido: 1, 2, 3, 4 , 7

Tema 2. Exploración de grafos

Recorrido en profundidad

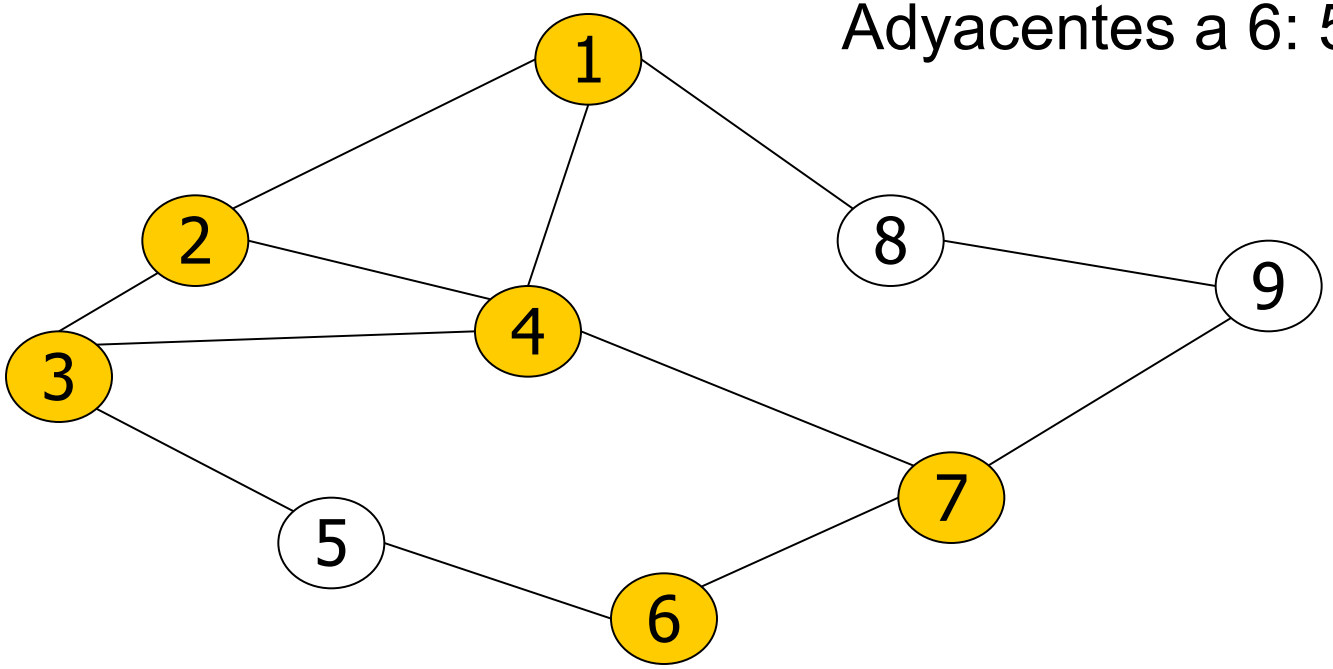


Adyacentes a 7: ~~4~~, 6, 9

Recorrido: 1, 2, 3, 4, 7, 6

Tema 2. Exploración de grafos

Recorrido en profundidad

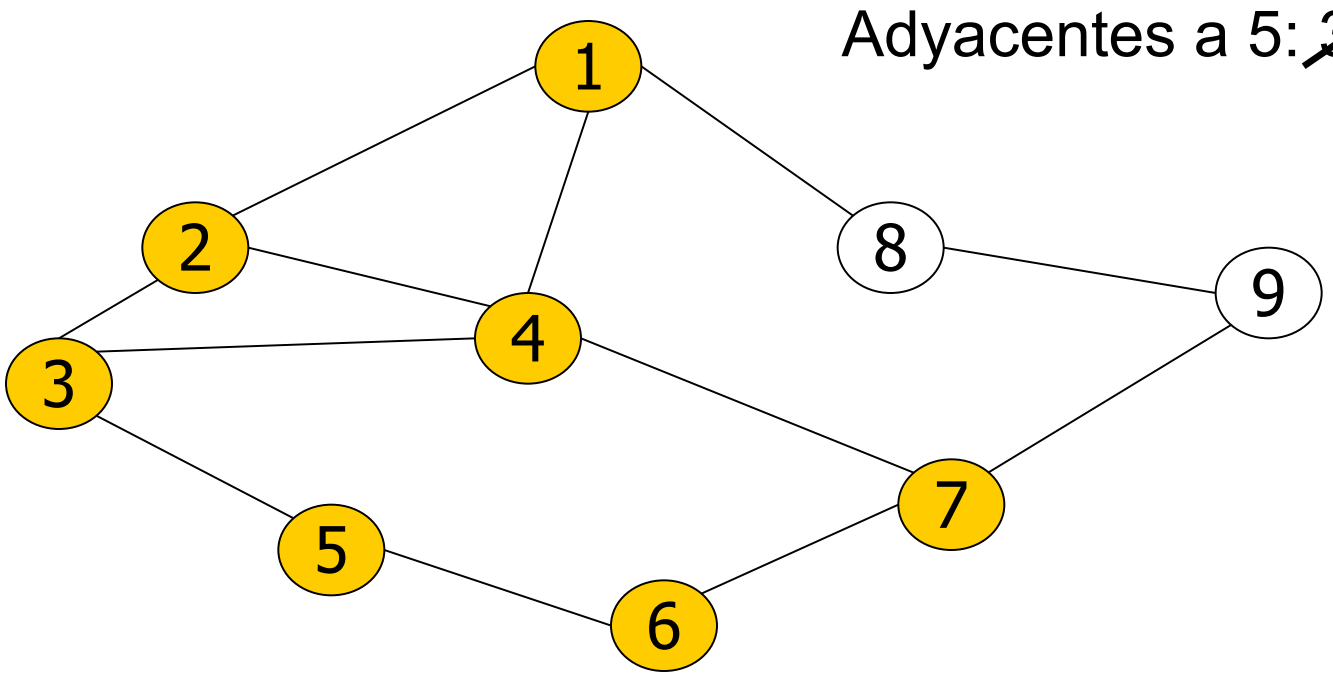


Adyacentes a 6: 5, 7

Recorrido: 1, 2, 3, 4, 7, 6 , 5

Tema 2. Exploración de grafos

Recorrido en profundidad

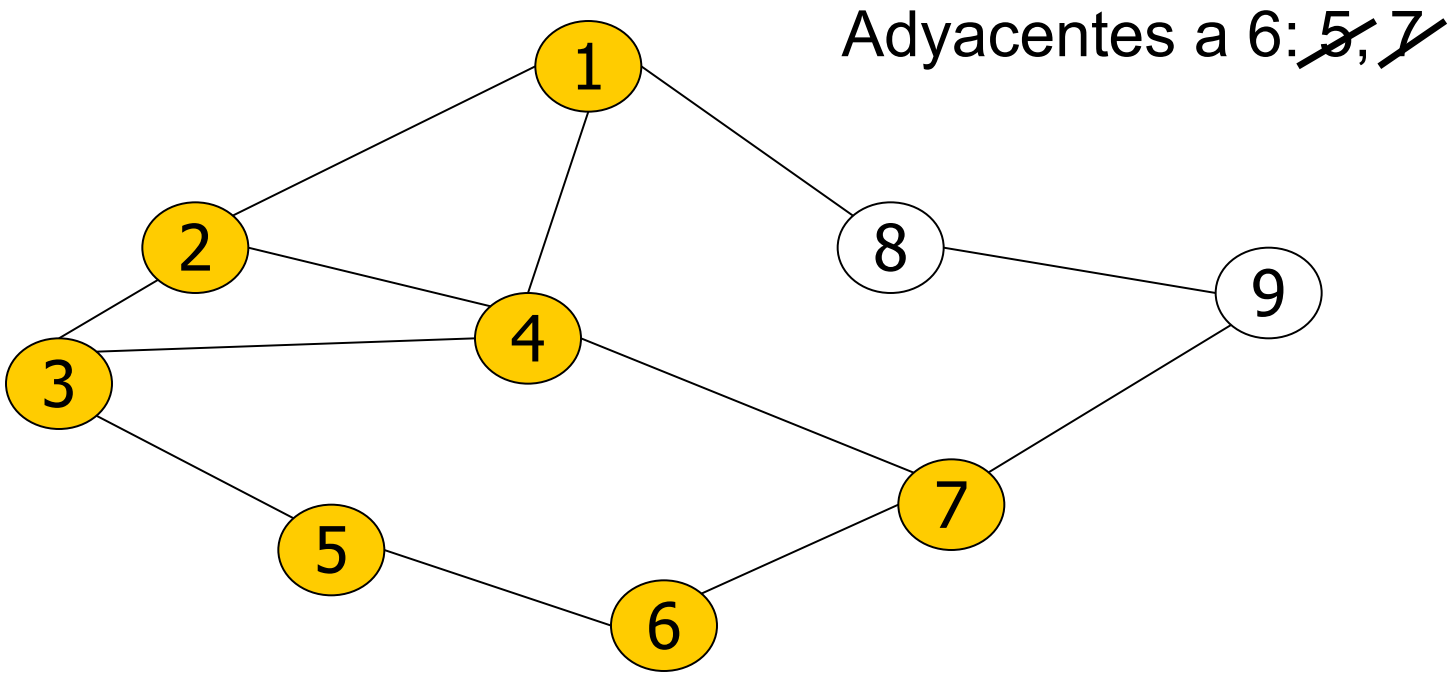


Adyacentes a 5: ~~3~~, ~~6~~

Recorrido: 1, 2, 3, 4, 7, 6, 5

Tema 2. Exploración de grafos

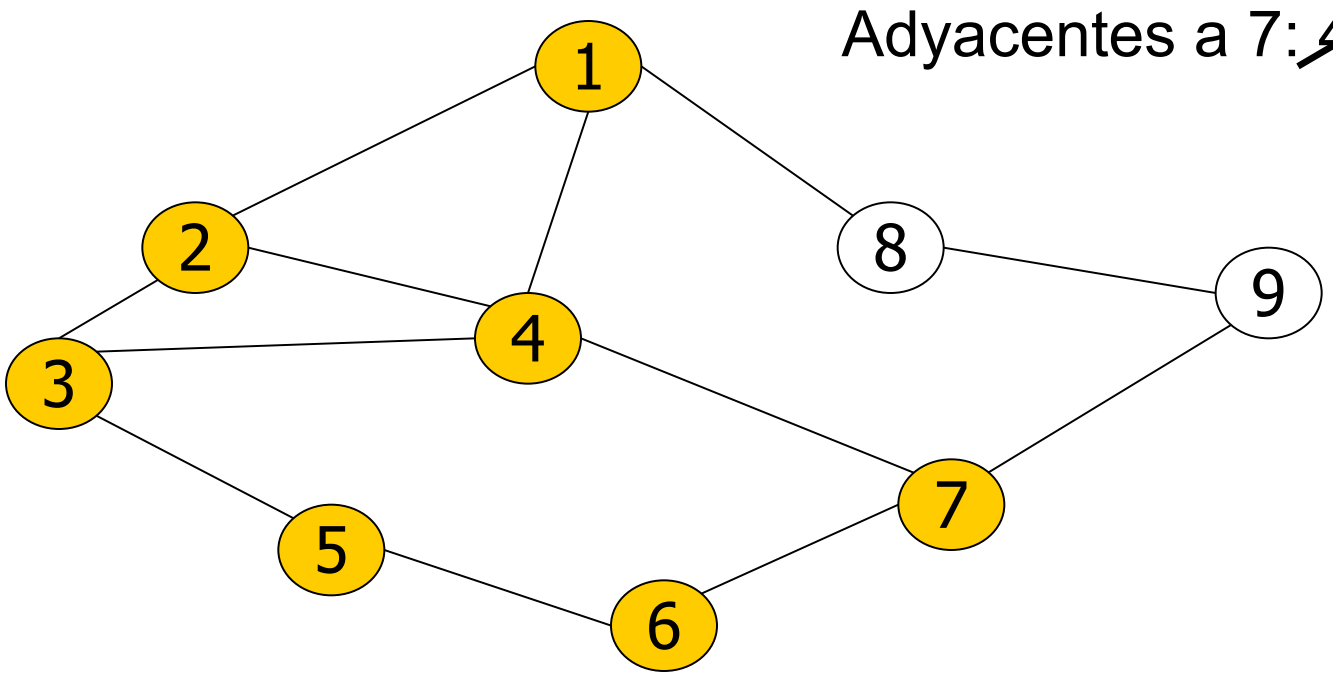
Recorrido en profundidad



Recorrido: 1, 2, 3, 4, 7, 6, 5

Tema 2. Exploración de grafos

Recorrido en profundidad

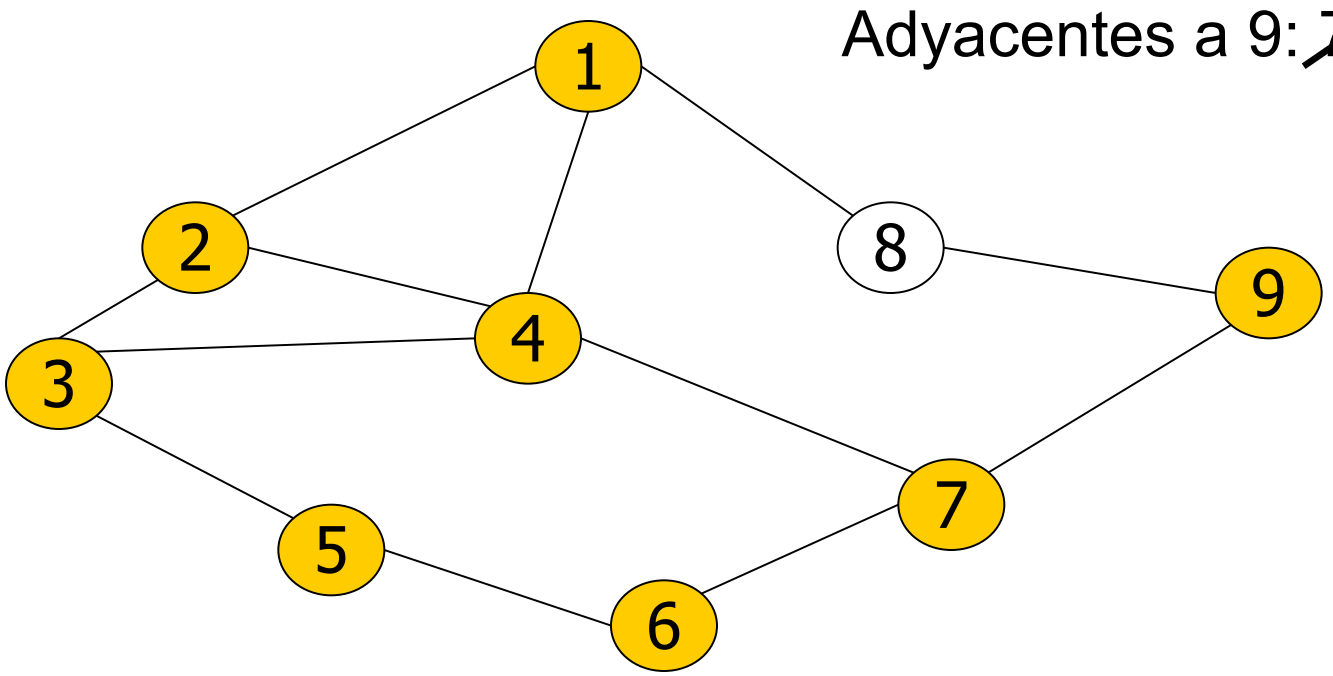


Adyacentes a 7: ~~4~~, ~~6~~, 9

Recorrido: 1, 2, 3, 4, 7, 6, 5 , 9

Tema 2. Exploración de grafos

Recorrido en profundidad



Recorrido: 1, 2, 3, 4, 7, 6, 5, 9 , 8

Tema 2. Exploración de grafos

Recorrido en profundidad

```
procedimiento recorrido_en_profundidad (G: grafo)
 para cada v ∈ N hacer
   marca[v] ← no visitado
 fpara
 para cada v ∈ N hacer
   si marca[v] ≠ visitado entonces
      rp(G, marca, v)
   fsi
 fpara
fprocedimiento
```

Tema 2. Exploración de grafos

Recorrido en profundidad

```
procedimiento rp (G: grafo;
 marca:vector[1..n]; v: nodo)
{El nodo v no ha sido visitado anteriormente}
 marca[v] ← visitado
 para cada nodo adyacente a v hacer
   si marca[w] ≠ visitado entonces
      rp(G, marca, w)
   fsi
 fpara
fprocedimiento
```

Análisis de la complejidad

- Cada **vértice** se visita **una única** vez $\Rightarrow n$ llamadas al procedimiento rp $\Rightarrow \Theta(V)$
- El algoritmo examina **todas** las **aristas** $\Rightarrow \Theta(E)$
- La **complejidad global** del algoritmo es

$$O(\max(V, E))$$

Tema 2. Exploración de grafos

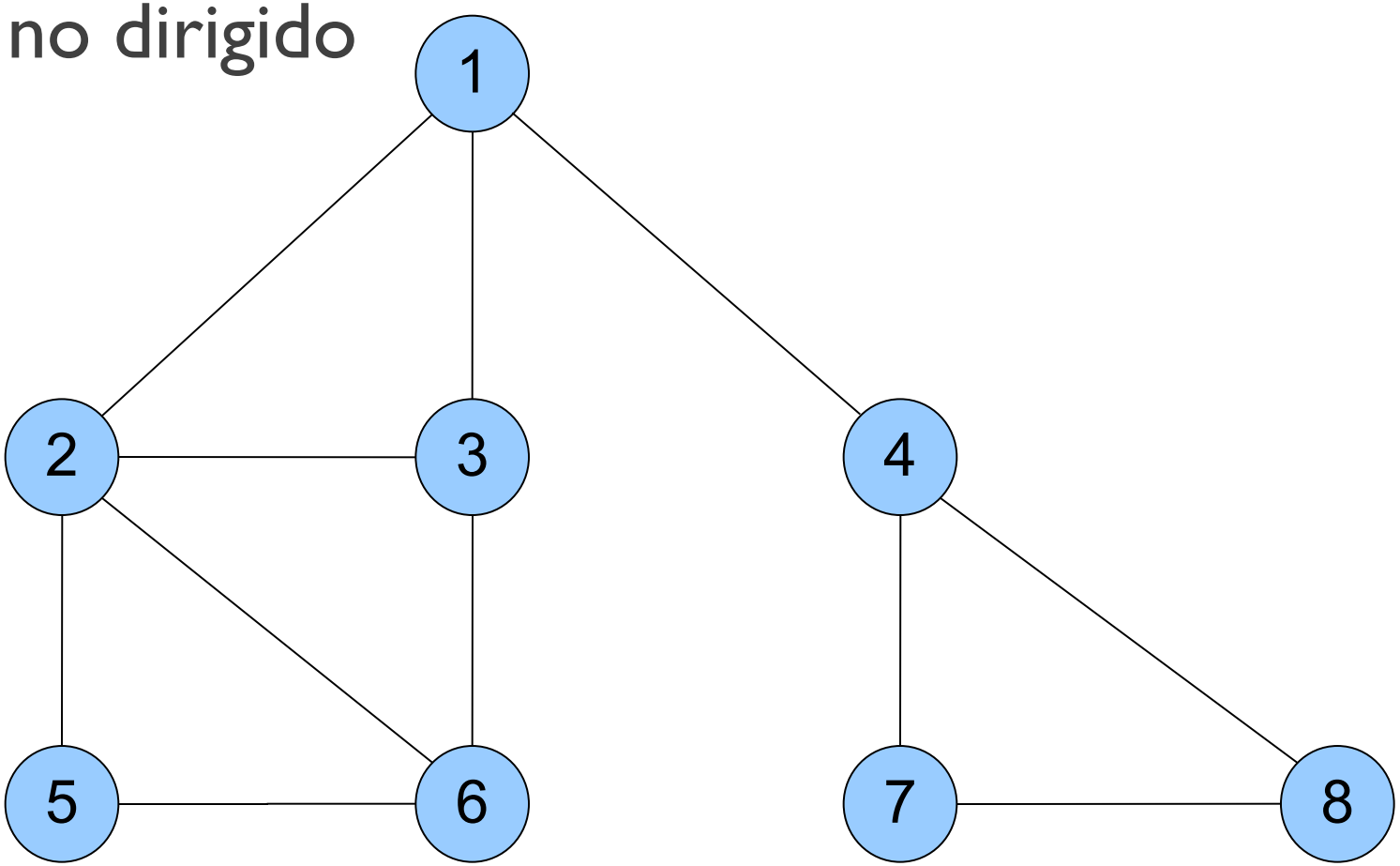
Recorrido en profundidad

- El recorrido en profundidad de un grafo conexo *G* crea un **árbol de recubrimiento** *T*
  - Las aristas de *T* son un **subcojunto** de las aristas de *G*
  - La **raíz** de *T* es el punto de partida de la exploración de *G*
- Si el grafo no es **conexo**, se obtiene un árbol por cada **componente conexa** (bosque)
- La exploración en profundidad de un grafo visita las nodos del grafo en "**preorden**"

Tema 2. Exploración de grafos

Recorrido en profundidad

- Ejemplo: recorrido en profundidad del siguiente grafo no dirigido

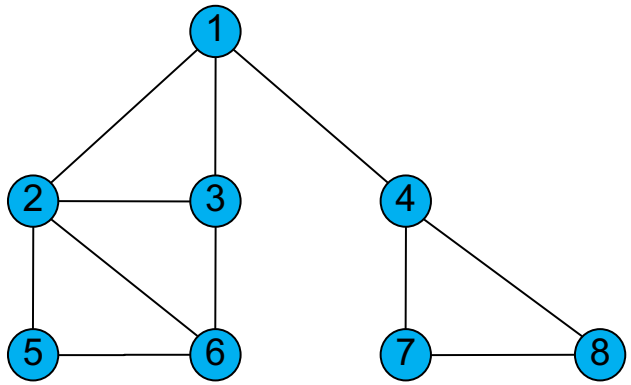


Tema 2. Exploración de grafos

Recorrido en profundidad

- Secuencia de llamadas a rp

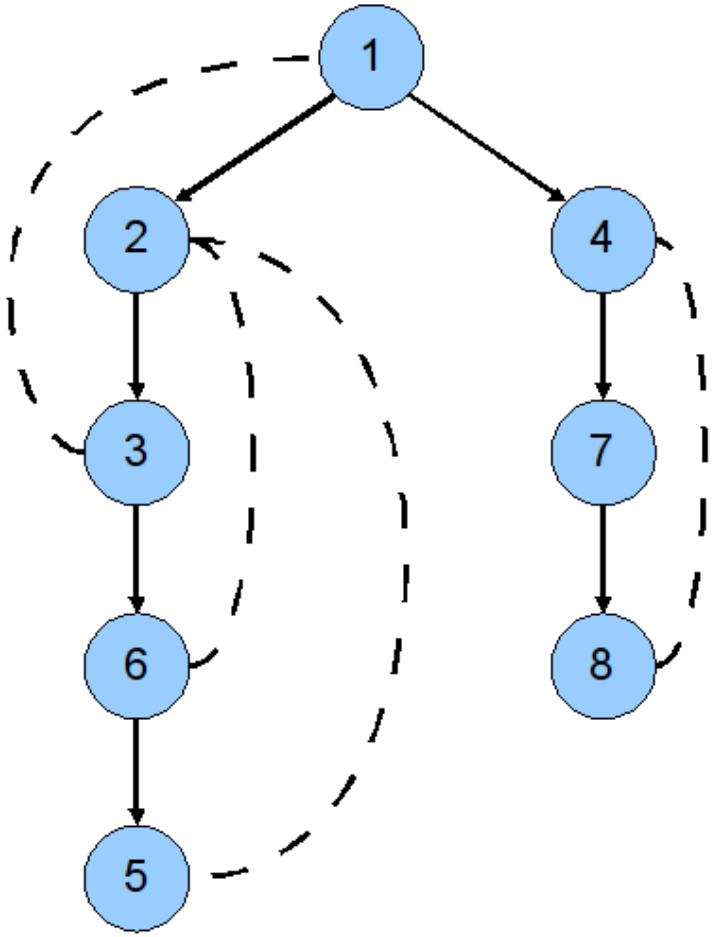
1. rp(1) {llamada inicial}
2. rp(2) {llam. rec.}
3. rp(3) {llam. rec.}
4. rp(6) {llam. rec}
5. rp(5) {llam. rec. no se puede continuar}
6. rp(4) {llam. adyacente al nodo 1}
7. rp(7) {llam. rec}
8. rp(8) {llam. rec. no se puede continuar

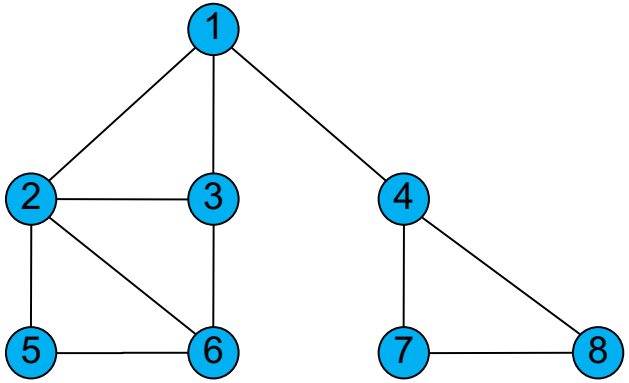


Tema 2. Exploración de grafos

Recorrido en profundidad

- Árbol de recorrido en profundidad (T)

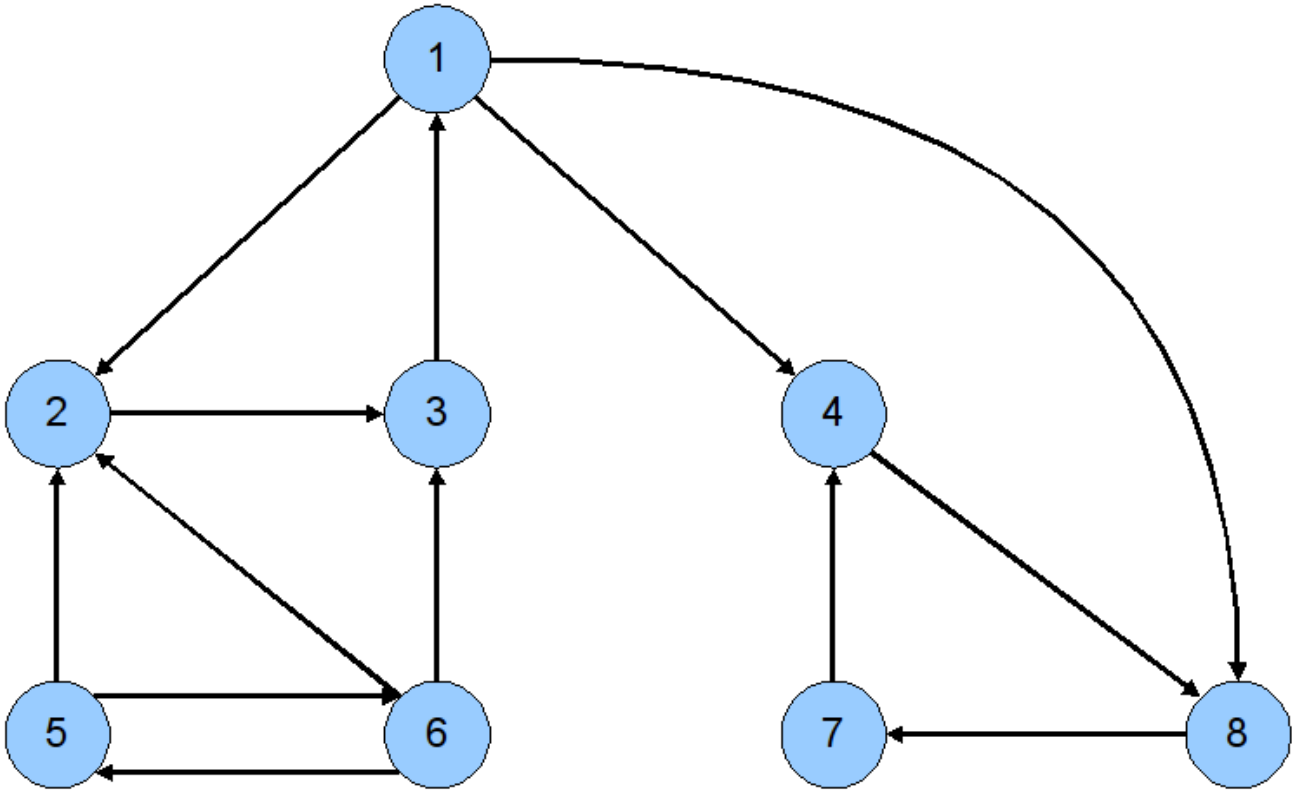




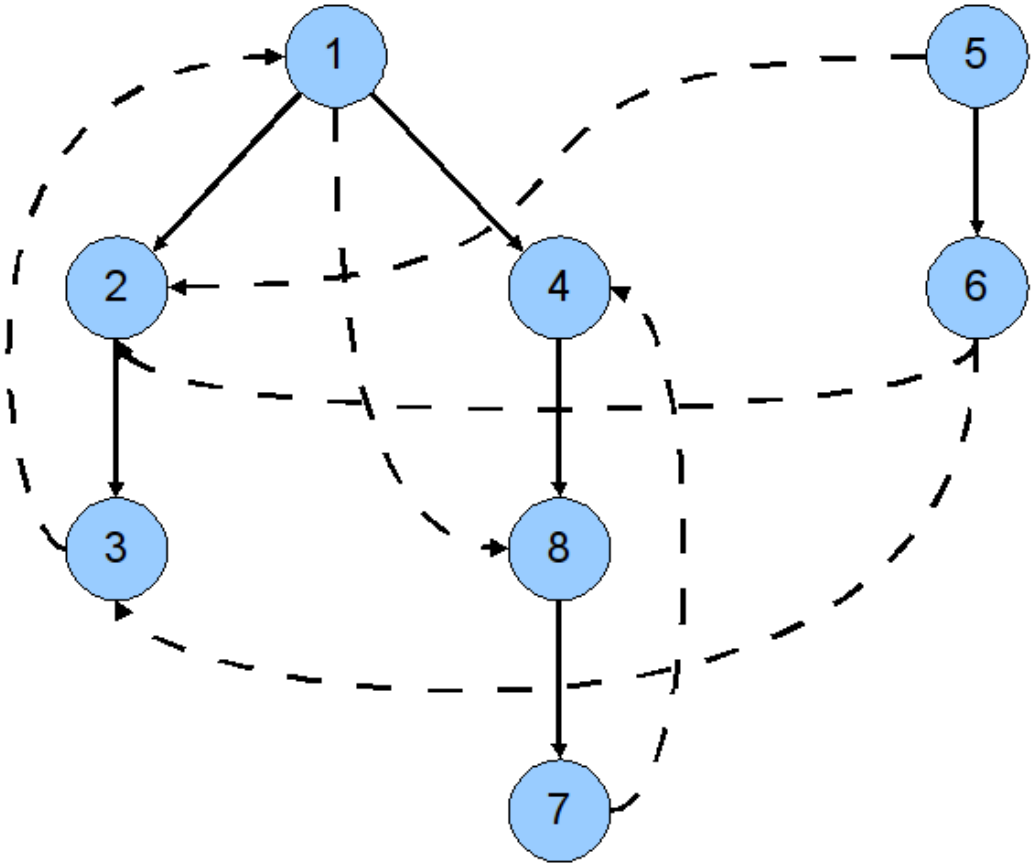
Tema 2. Exploración de grafos

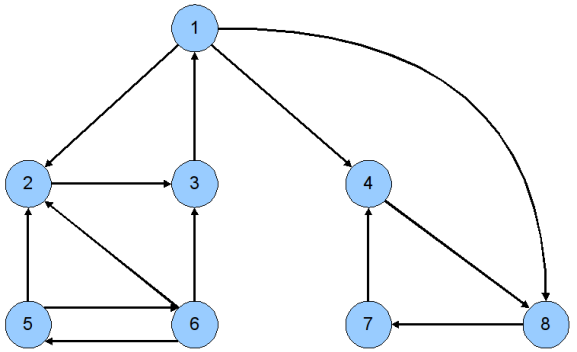
Recorrido en profundidad en grafos dirigidos

El procedimiento es esencialmente el mismo, salvo que se distingue entre adyacentes e incidentes



"Bosque" de recorrido en profundidad





Tema 2. Exploración de grafos

Recorrido en anchura

- El recorrido en anchura (*breath-first search*) se llama así porque recorre la frontera en anchura
  - Visita todos los vértices a una **distancia *k*** antes de descubrir el primer vértice a la **distancia *k*+1**
  - Suele implementarte se manera **iterativa**
  - Se tiene que incluir una **cola** con los vértices visitados para evitar **ciclos** y establecer el orden en la búsqueda

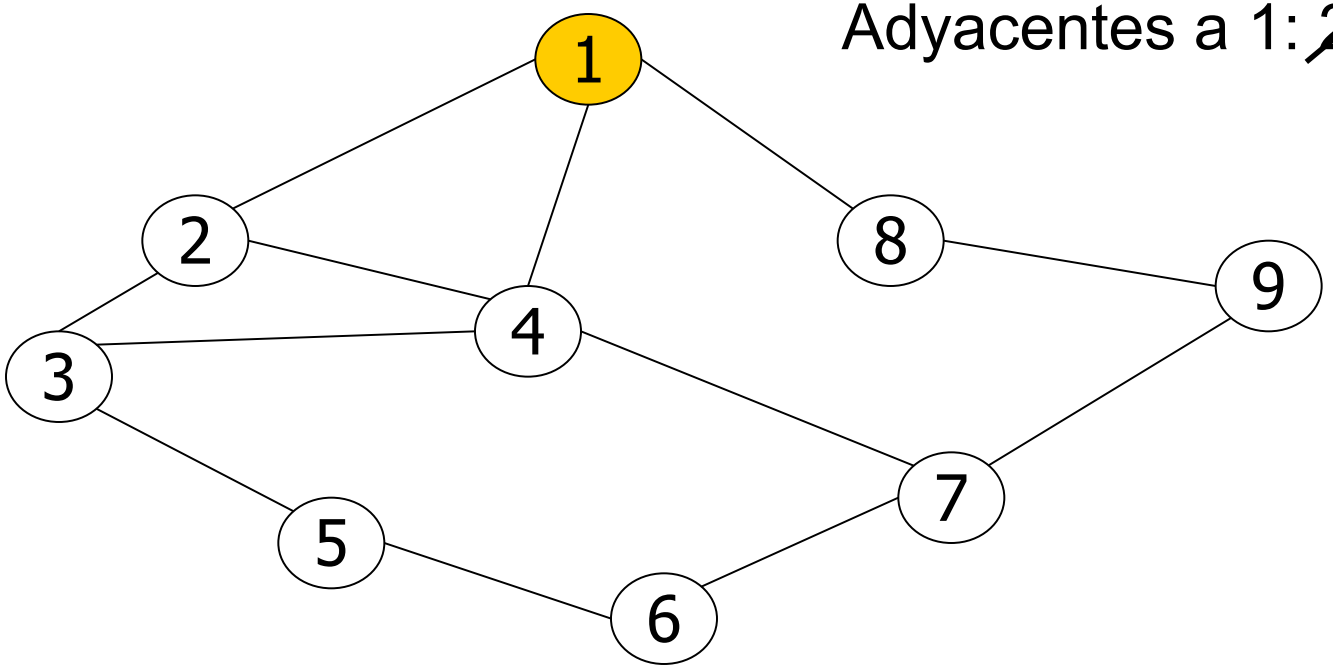
Tema 2. Exploración de grafos

Recorrido en anchura

- Dado un grafo $G=(V, E)$ y un vértice inicial s:
  - Calcula la **distancia** (menor número de vértices) desde s hasta los vértices alcanzables
  - Produce un **árbol** de recorrido en anchura donde la raíz es s
  - Para cualquier vértice v alcanzable desde s, la ruta en el árbol de recorrido en anchura desde s hasta v es el **camino más corto** entre esos dos vértices
  - Es un algoritmo adecuado para grafos dirigidos y no dirigidos

Tema 2. Exploración de grafos

Recorrido en anchura

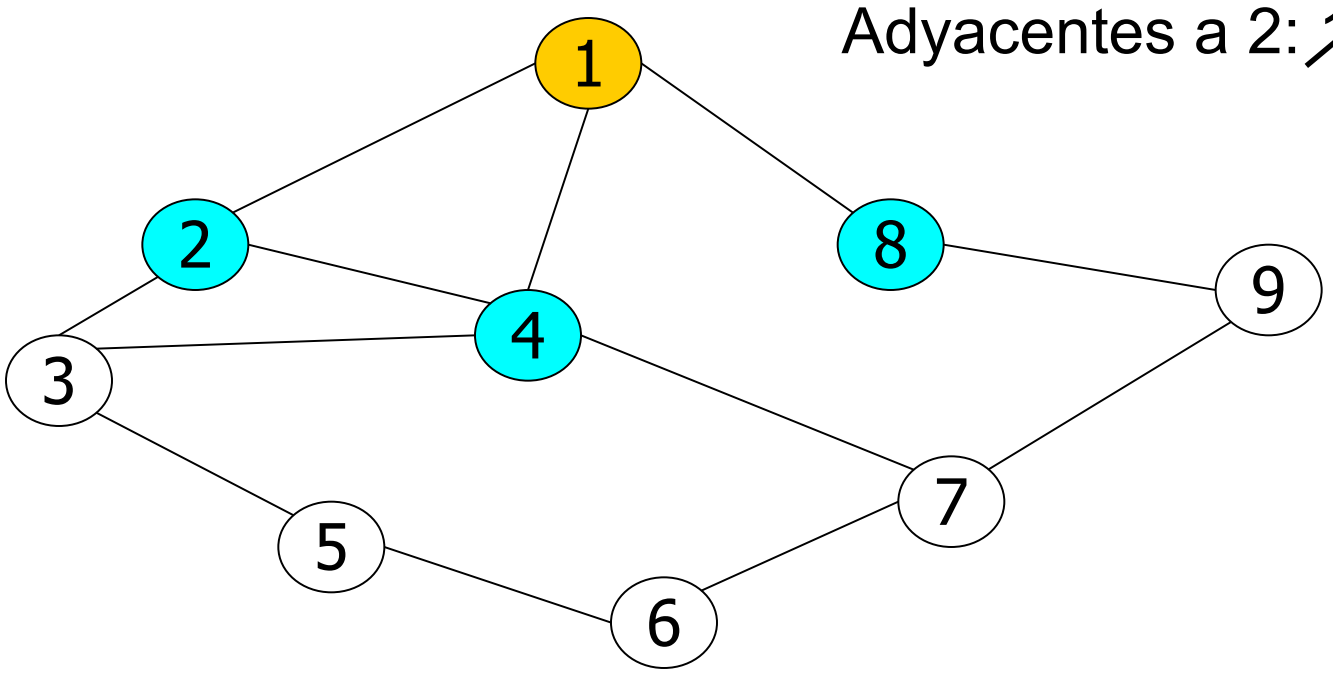


Adyacentes a 1: ~~2~~, ~~4~~, ~~8~~

Recorrido: 1, 2, 4, 8

Tema 2. Exploración de grafos

Recorrido en anchura

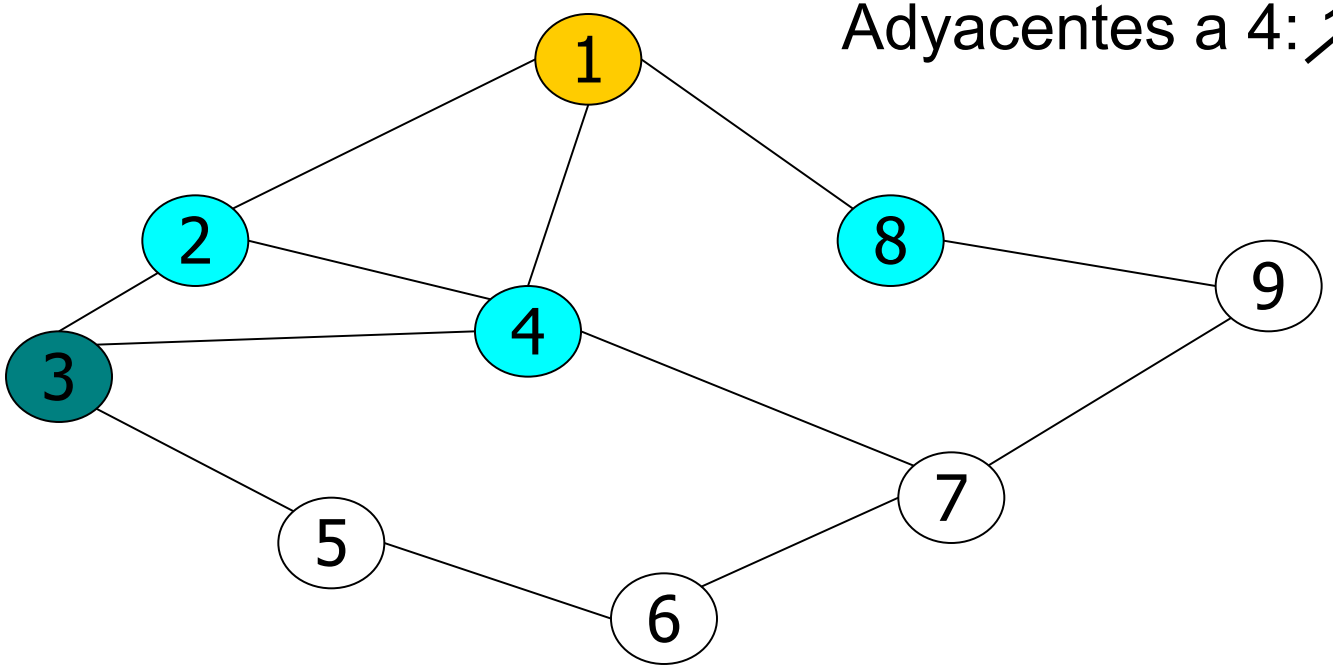


Adyacentes a 2: ~~1~~, ~~3~~, ~~4~~

Recorrido: 1, 2, 4, 8 , 3

Tema 2. Exploración de grafos

Recorrido en anchura

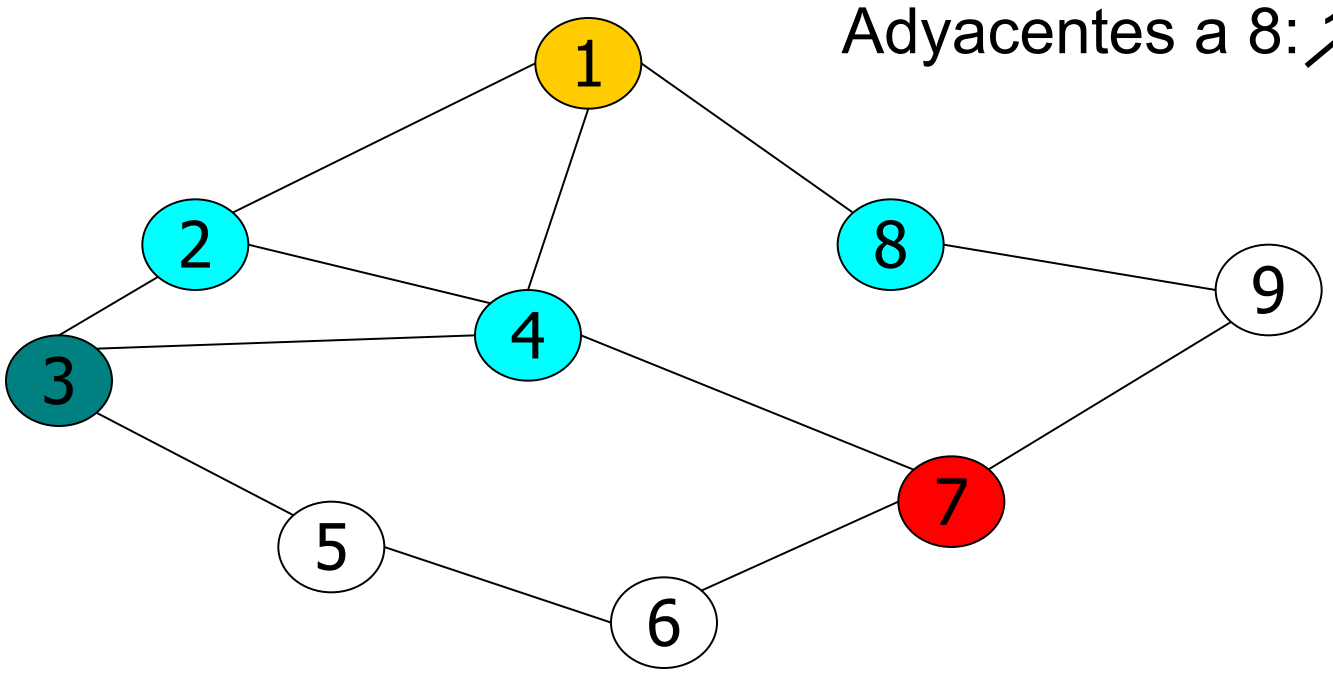


Adyacentes a 4: ~~1~~, ~~2~~, ~~3~~, ~~7~~

Recorrido: 1, 2, 4, 8, 3, 7

Tema 2. Exploración de grafos

Recorrido en anchura

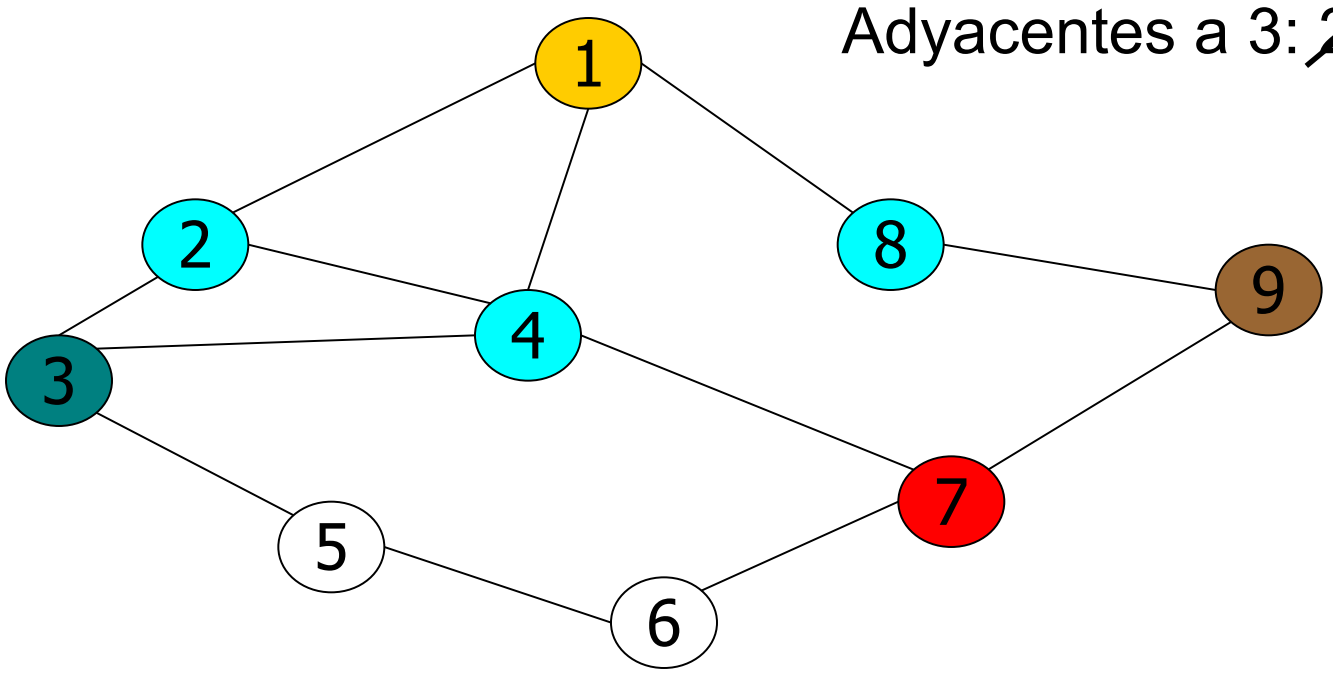


Adyacentes a 8: ~~1~~, ~~9~~

Recorrido: 1, 2, 4, 8, 3, 7 , 9

Tema 2. Exploración de grafos

Recorrido en anchura

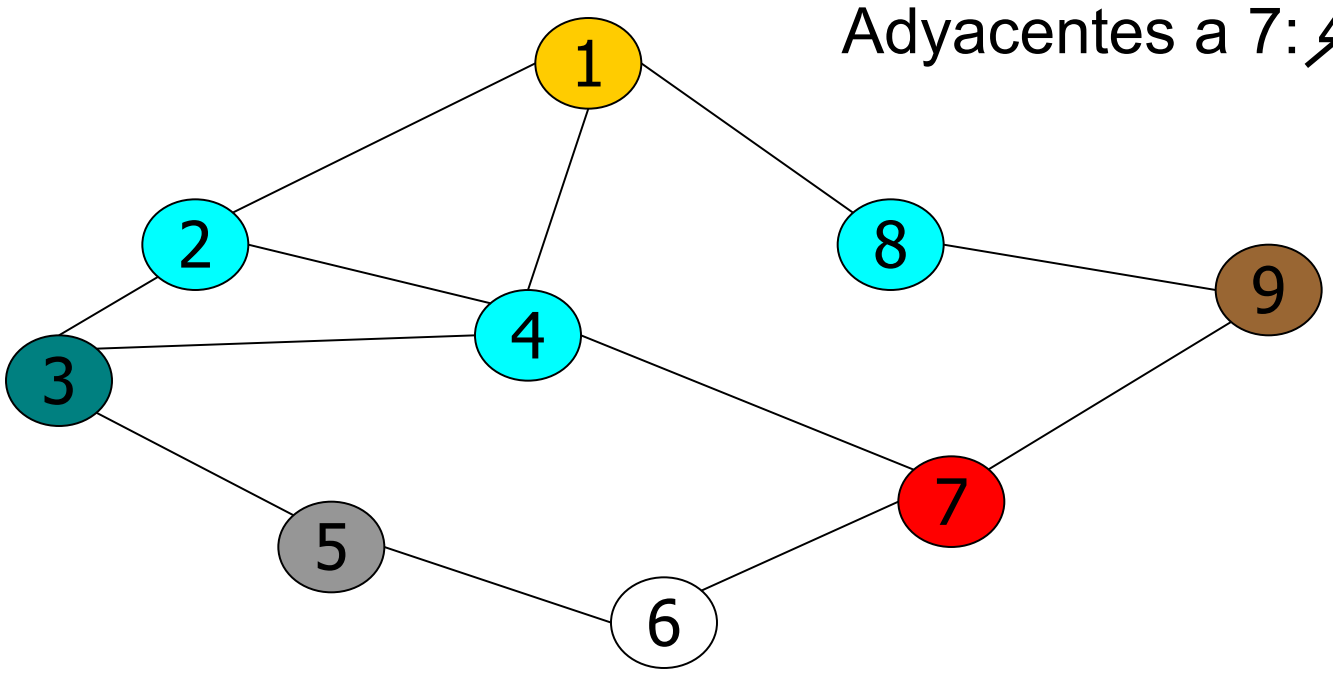


Adyacentes a 3: ~~2~~, ~~4~~, ~~5~~

Recorrido: 1, 2, 4, 8, 3, 7, 9 , 5

Tema 2. Exploración de grafos

Recorrido en anchura



Adyacentes a 7: ~~4~~, ~~6~~, ~~9~~

Recorrido: 1, 2, 4, 8, 3, 7, 9, 5 , 6

Tema 2. Exploración de grafos

Recorrido en anchura

```
procedimiento recorrido(G)
 para cada v ∈ N hacer
   marca[v] ← no visitado
 fpara
 para cada v ∈ N hacer
   si marca[v] ≠ visitado entonces
      ra(v)
   fsi
 fpara
fprocedimiento
```

Tema 2. Exploración de grafos

Recorrido en anchura

```
procedimiento ra(v)
 Q ← colavacia
 marca[v] ← visitado
 poner v en Q
 mientras Q no esté vacía hacer
   quitar aux de Q
   para todo w adyacente a aux hacer
      si marca[w] ≠ visitado entonces
         marca[w] ← visitado
         poner w en Q
      fsi
   fpara
 fmientras
fprocedimiento
```

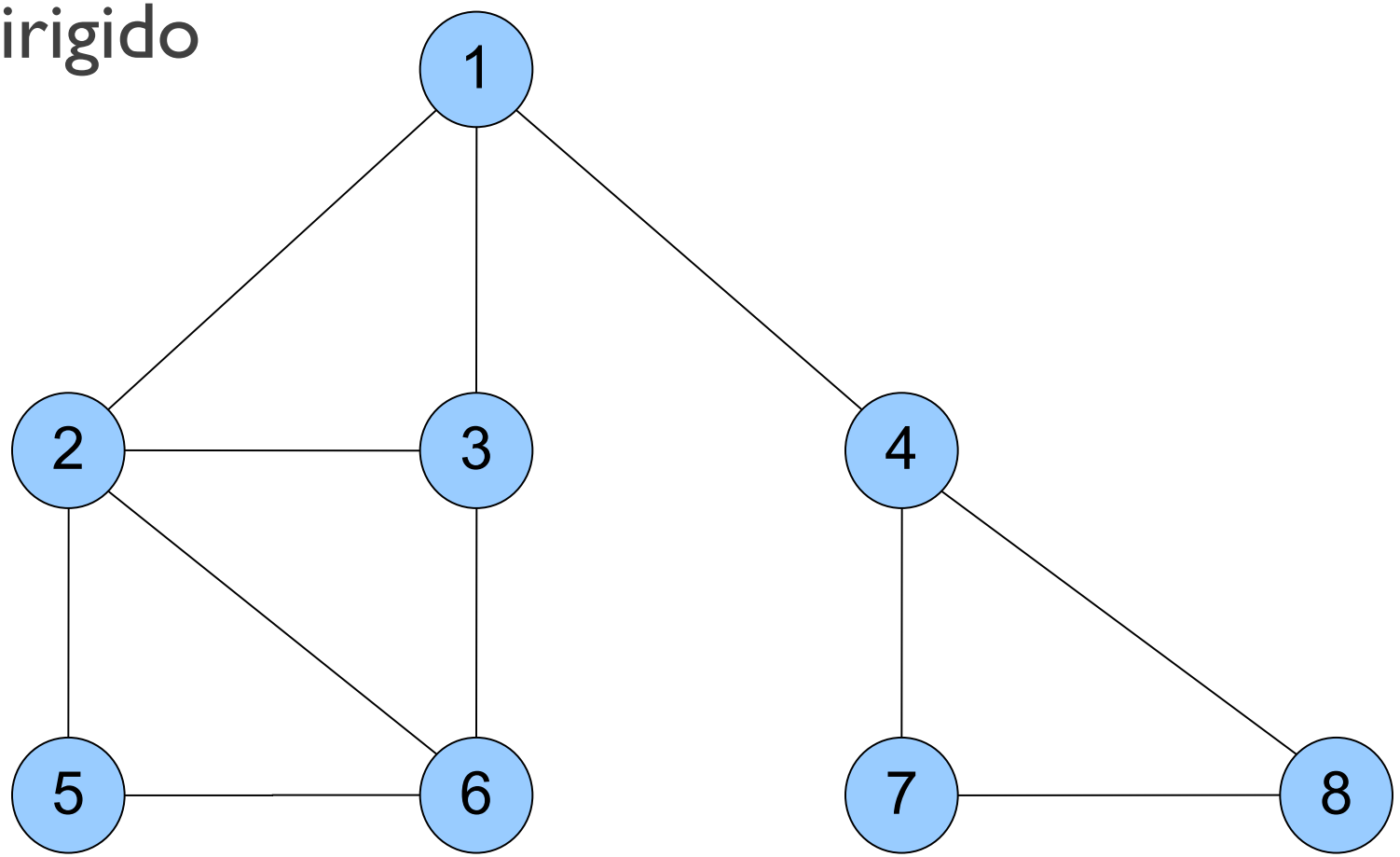
Análisis de complejidad

- El **tiempo requerido** para realizar un recorrido en **anchura** de un grafo es el mismo que para un recorrido en **profundidad** $\Rightarrow \Theta\ (max(V,E))$
- También genera **árboles de recubrimiento**. Si el grafo es conexo $\Rightarrow$ un único árbol.
- Se emplea en exploraciones parciales de grafos, para hallar el camino más corto entre dos puntos de un grafo, etc.

Tema 2. Exploración de grafos

Recorrido en anchura

- Ejemplo: recorrido en anchura del siguiente grafo no dirigido

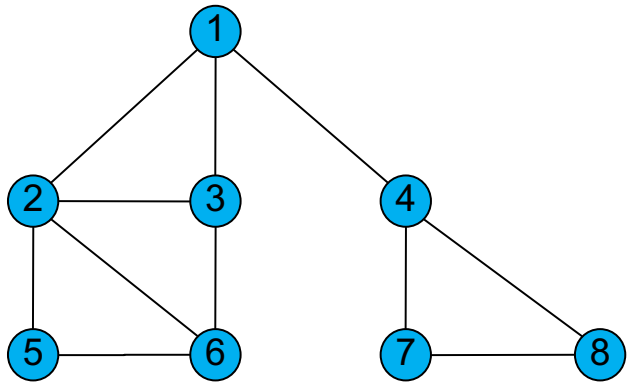


Tema 2. Exploración de grafos

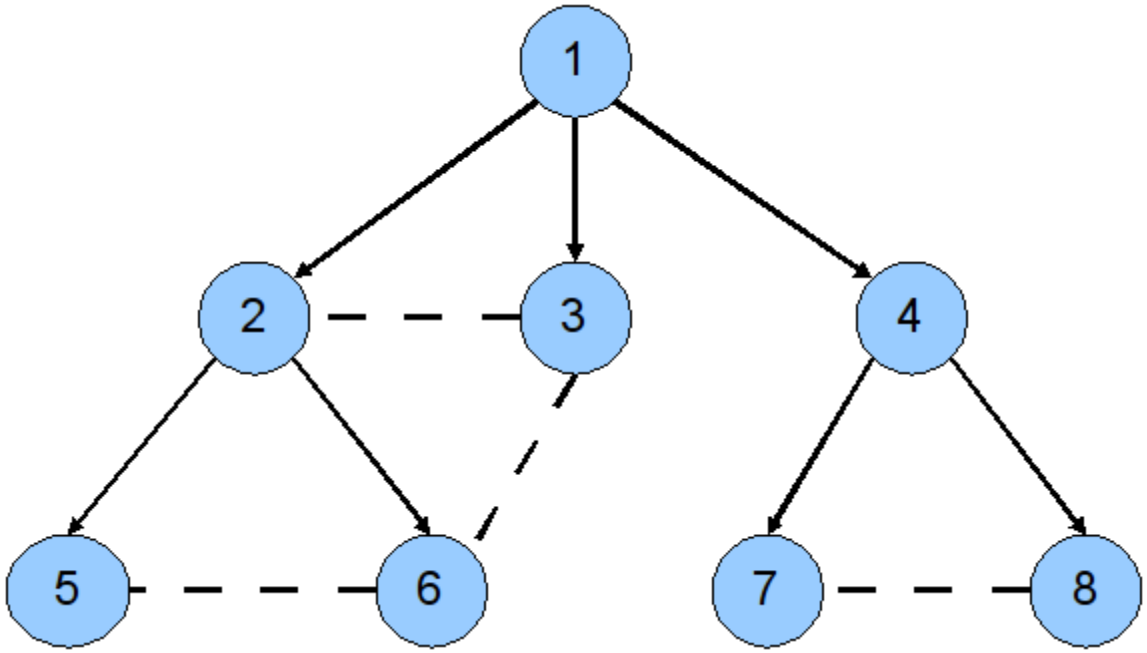
Recorrido en anchura

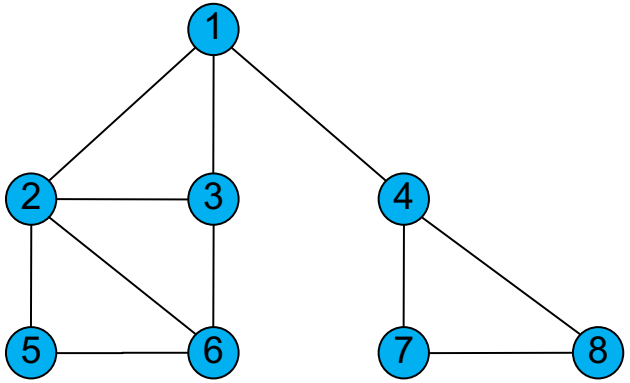
- Ejemplo: recorrido en anchura

Paso	Nodo Visitado	Q
1	1	2, 3, 4
2	2	3, 4, 5, 6
3	3	4, 5, 6
4	4	5, 6, 7, 8
5	5	6, 7, 8
6	6	7, 8
7	7	8
8	8	-



Árbol de recorrido en anchura:





Tema 2. Exploración de grafos

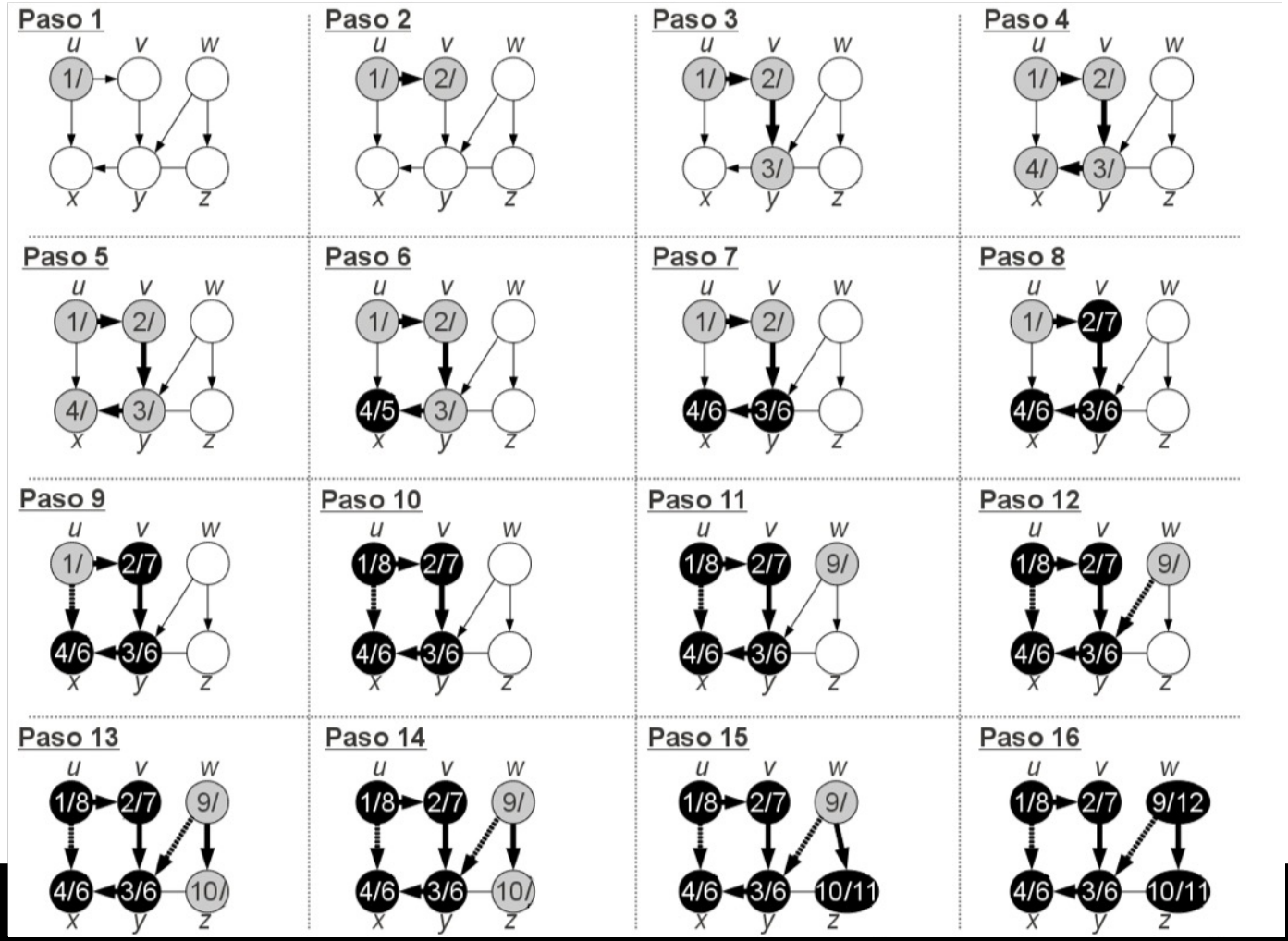
Algoritmos sobre grafos

- Dado un grafo dirigido y **acíclico**, se denomina **ordenación topológica** a una disposición lineal de los nodos tal que, dado un arco (*u*,*v*), el nodo *u* esté antes que *v* en la ordenación
  - Un vértice se visita sí y sólo sí se han visitados todos sus predecesores
  - En caso de grafos con ciclos, el algoritmo sigue siendo válido, pero la interpretación no es directa.
- Aplicaciones prácticas
  - Fases de un proyecto (PERT)
  - Evaluación de atributos en la fase semántica de un compilador

Recorrido en profundidad. Distancia y finalización

- Los vértices se colorean como sigue:
  - Inicialmente son todos **blancos**
  - Cuando se visitan, se colorean de **gris** (denotado por d[v]
  - Cuando toda la adyacencia se ha visitado se colorea de **negro** (denotado por f[v])
  - **Implementación:** Cuando el nodo se etiqueta como negro, almacenar en una lista
  - La ordenación topológica **no es única**

Ejemplo de recorrido



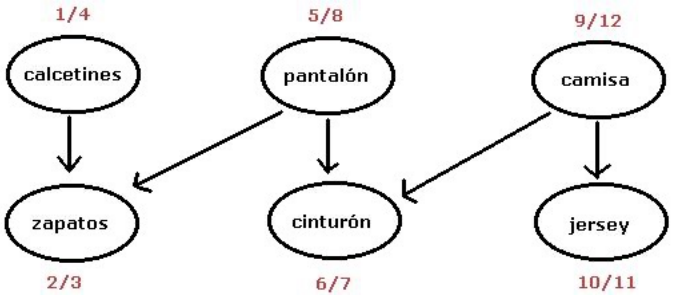
Algoritmos sobre grafos

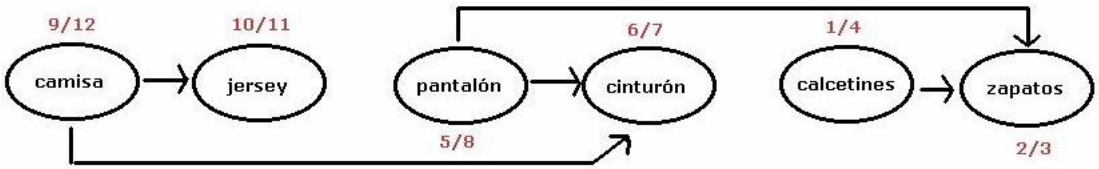
- Algoritmo

```
ORDENACIÓN_TOPOLÓGICA(grafo G)
for each vértice u ∈ V[G] do
  estado[u] = NO_VISITADO
  padre[u] = NULL
  tiempo =0
  for each vértice u ∈ V[G] do
    if estado[u] = NO_VISITADO then
      TOPOLÓGICO-Visitar(u)

TOPOLÓGICO-Visitar(nodo u)
estado[u]=VISITADO
tiempo = tiempo+1
d[u] = tiempo
for each v ∈ Adyacencia[u] do
  if estado[v]=NO_VISITADO then
    padre[v]=u
    TOPOLÓGICO-Visitar(v)
estado[u] = TERMINADO
tiempo = tiempo+1
f[u] = tiempo
insertar (lista, u)
```

- Ejemplo





- Ponerse la **camisa** antes que el **cinturón** y **jersey**
- Ponerse el **pantalón** antes que los **zapatos** y **cinturón**
- Ponerse los **calcetines** antes que los **zapatos**

Tema 2. Exploración de grafos

Algoritmos sobre grafos

- Un grafo se dice que es **fuertemente conexo** si existe un camino entre cada par de vértices
- Para **descomponer** un grafo *G* en sus componentes fuertemente conexas
  - Aplicar ordenación topológica sobre *G*
  - Calcular el grafo traspuesto G^T
  - Aplicar búsqueda en profundidad sobre G^T iniciando la búsqueda en los nodos de mayor a menor **tiempo de finalización** obtenidos en la primera ejecución de búsqueda en profundidad
  - El resultado será un bosque de árboles. Cada árbol es un componente fuertemente conexo.

Tema 2. Exploración de grafos

Algoritmos sobre grafos

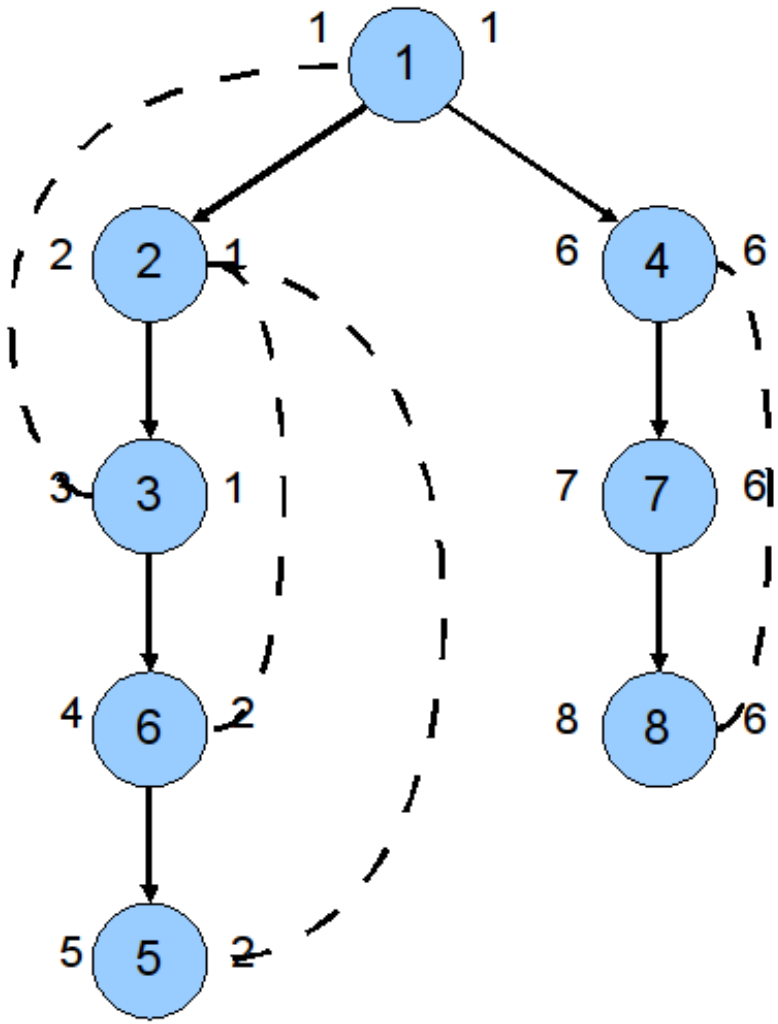
- **Puntos de articulación**: un vértice *v* de un grafo conexo es un punto de articulación si el subgrafo que se obtiene al eliminarlo (junto con sus aristas) es no conexo
- **Grafo biconexo** (o no articulado): aquél grafo que no tiene puntos de articulación
- **Grafo bicoherente** (o 2-arista conexo): aquél grafo cuyos puntos de articulación están unidos a cada componente del subgrafo restante por, al menos, dos aristas

Algoritmos sobre grafos

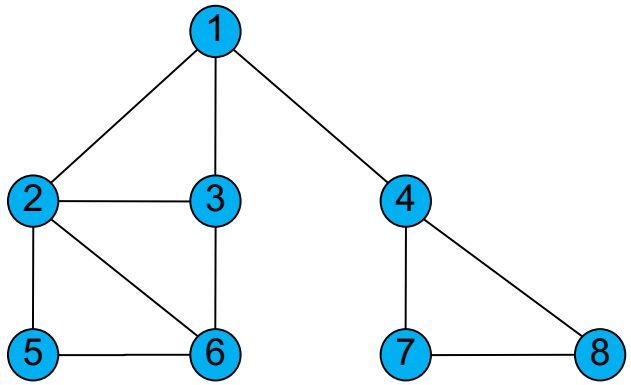
Si el grafo representa una red de comunicaciones

- Si es **biconexo** nos asegura el correcto funcionamiento de la red, aunque falle uno de los equipos
- Si es **bicoherente** nos asegura el correcto funcionamiento de la red, aunque falle una de las líneas de transmisión

Cálculo de los puntos de articulación



- A la izda de cada vértice se representa la it. de visita
- A la dcha de cada vértice se representa el nodo más alto alcanzable



Cálculo de los puntos de articulación

- Sea *v* cualquier vértice del árbol (excepto la raíz) y *x* un hijo de *v*
  - Si masAlto[x] < preOrden[v] implica que se puede llegar desde *x* a regiones más altas sin pasar por (*v*,*u*). Entonces *u* **no es** un **punto de articulación**
  - Si preOrden[x] >= masAlto[v] no se puede llegar desde *u* a regiones más altas del grafo sin pasar por (*v*,*u*). Entonces u **es un punto de articulación**
- Si *u* es la **raíz del árbol** y tiene más de un hijo, **es un punto de articulación**

Otras aplicaciones interesantes

- Comprobar que un grafo es bipartito
- Detectar ciclos en grafos
- Camino más largo en un DAG
- Detectar si dos nodos están conectados o no
- Caminos y ciclos eulerianos
- Cierre transitivo
- Caminos entre un origen y un destino con k aristas