



Grafos p2

Programación 3 TUDAI
Cursada 2024

Recorridos sobre un Grafo

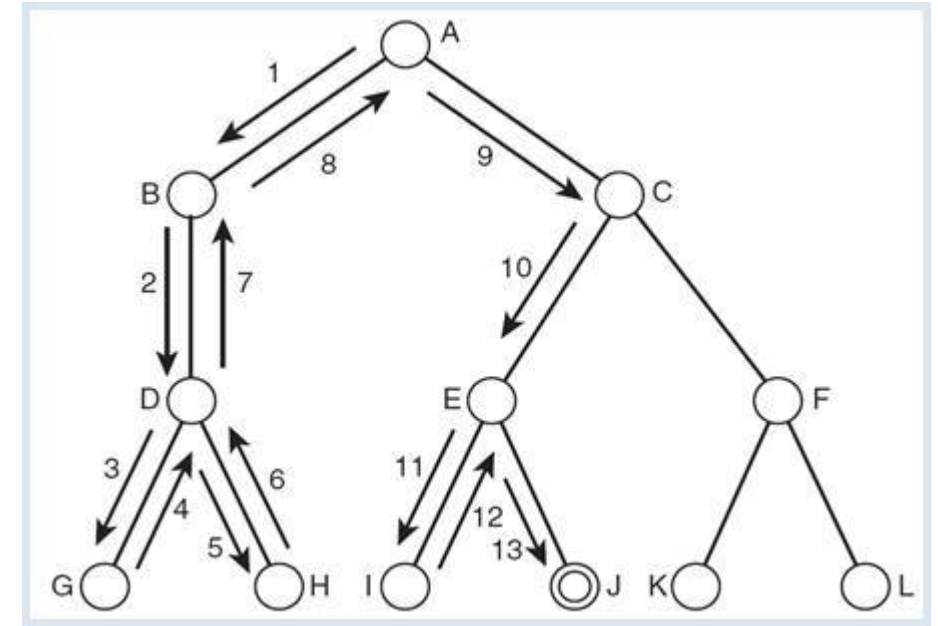
Un recorrido (o exploración) de un Grafo visita todos los vértices en una secuencia determinada.

Vamos a ver dos de los recorridos más comunes, el recorrido en profundidad (DFS) y el recorrido en amplitud (BFS).

DFS (Depth-First Search)

Recorrido en profundidad

- Es una generalización del recorrido preorden de un árbol.
- Al tratarse de Grafos, ya no tenemos hijos pero tenemos nodos adyacentes de un vértice.
- Recursivamente visita todos los adyacentes no visitados de un vértice. Hay que llevar registro de cuáles se visitaron para no volver a visitar.
- Se debe tener cuidado porque en un grafo se puede llegar a un vértice por mas de un camino, por eso hay que visitar los nodos que ya no fueron visitados por el recorrido.



RECORRIDO: A, B, D, G, H, C, E, I, J, F, K, L

DFS: Review del Código

```
DFS(G)
{
    for each vértice  $u \in G \rightarrow V$ 
    {
         $u \rightarrow \text{color} = \text{BLANCO};$ 
    }
    tiempo = 0;
    for each vértice  $u \in G \rightarrow V$ 
    {
        if ( $u \rightarrow \text{color} == \text{BLANCO}$ )
            DFS_Visit(u);
    }
}
```

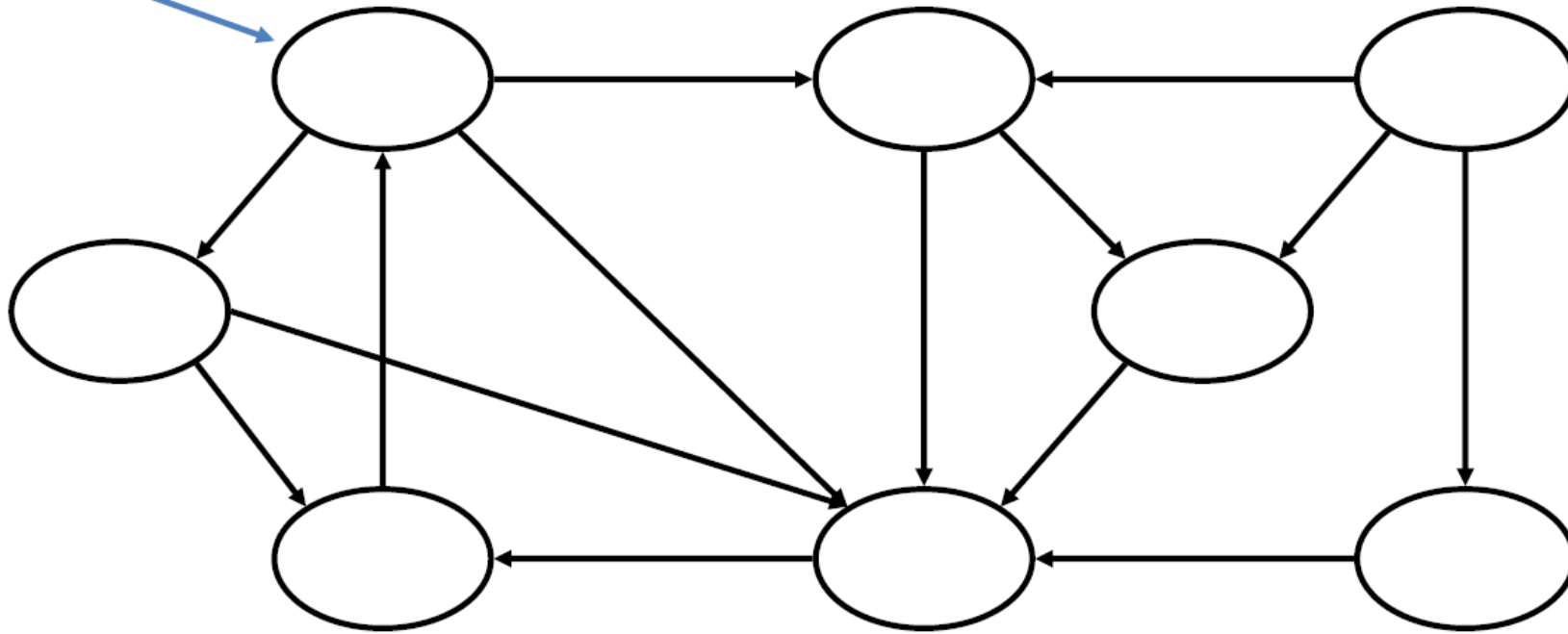
```
DFS_Visit(u)
{
     $u \rightarrow \text{color} = \text{AMARILLO};$ 
    tiempo = tiempo + 1;
     $u \rightarrow d = \text{tiempo};$ 
    for each  $v \in u \rightarrow \text{Adj}[]$ 
    {
        if ( $v \rightarrow \text{color} == \text{BLANCO}$ )
            DFS_Visit(v);
    }
     $u \rightarrow \text{color} = \text{NEGRO};$ 
    tiempo = tiempo + 1;
     $u \rightarrow f = \text{tiempo};$ 
}
```

Es un planteo recursivo.

Traza de Recorrido en Profundidad

DFS (1/17)

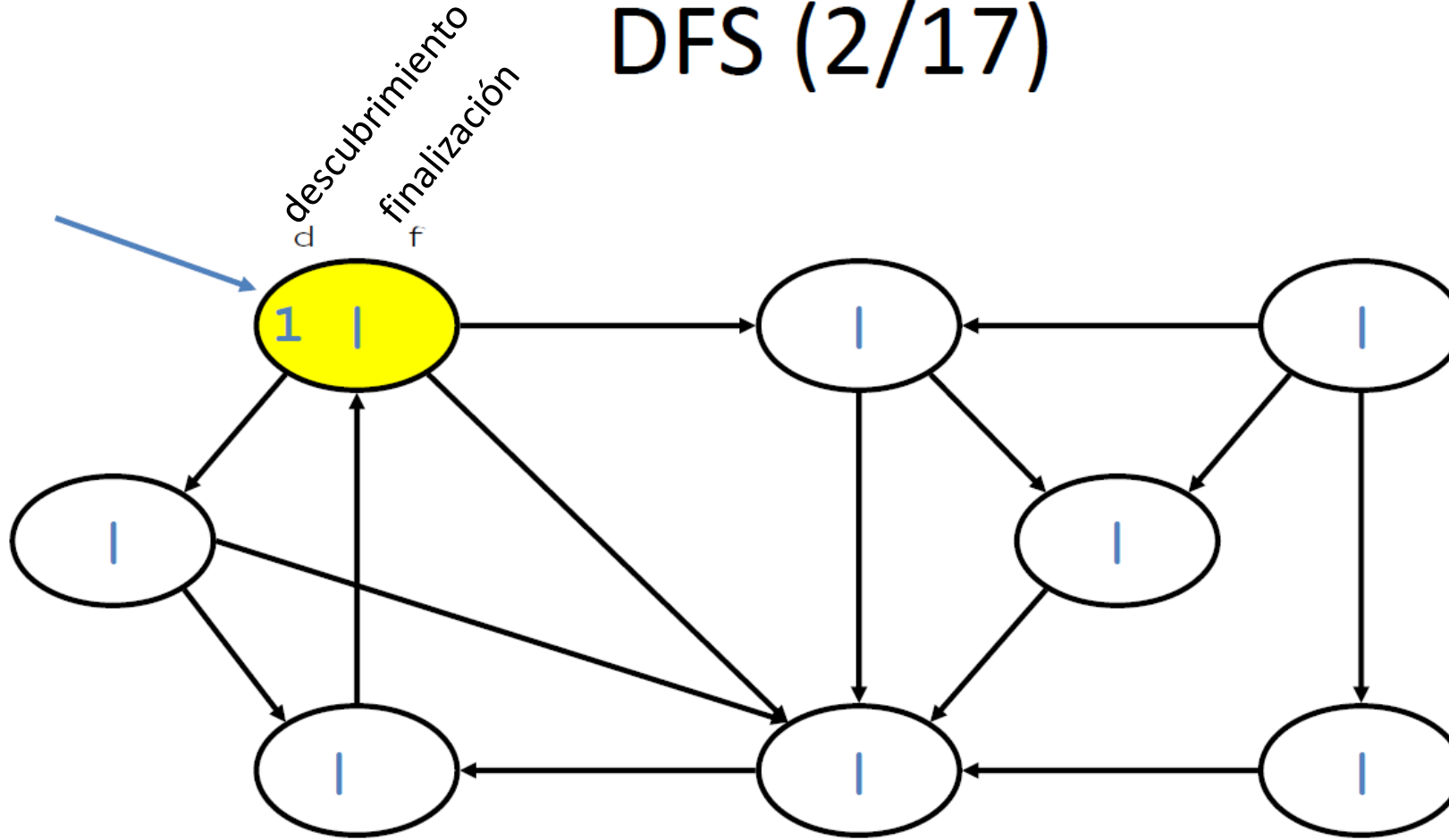
vértice
inicial



```
DFS_Visit(u)
{
    u->color = AMARILLO;
    tiempo = tiempo+1;
    u->d = tiempo;
    for each v ∈ u->Adj[]
    {
        if (v->color== BLANCO)
            DFS_Visit(v);
    }
    u->color = NEGRO;
    tiempo = tiempo +1;
    u->f = tiempo;
}
```

Traza de Recorrido en Profundidad

DFS (2/17)



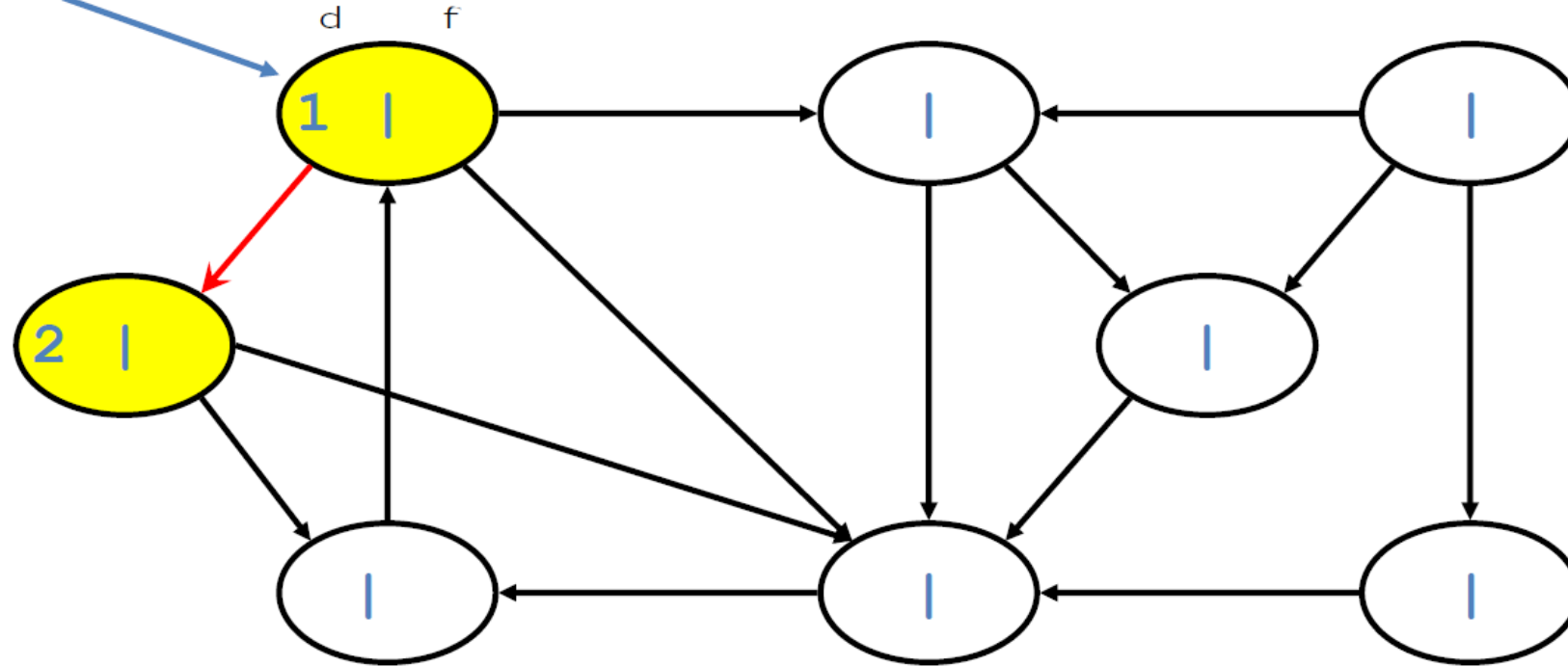
```
DFS_Visit(u)
{
    u->color = AMARILLO;
    tiempo = tiempo+1;
    u->d = tiempo;
    for each v ∈ u->Adj[]
    {
        if (v->color== BLANCO)
            DFS_Visit(v);
    }
    u->color = NEGRO;
    tiempo = tiempo +1;
    u->f = tiempo;
}
```

Traza de Recorrido en Profundidad

DFS (3/17)

vértice

inicial

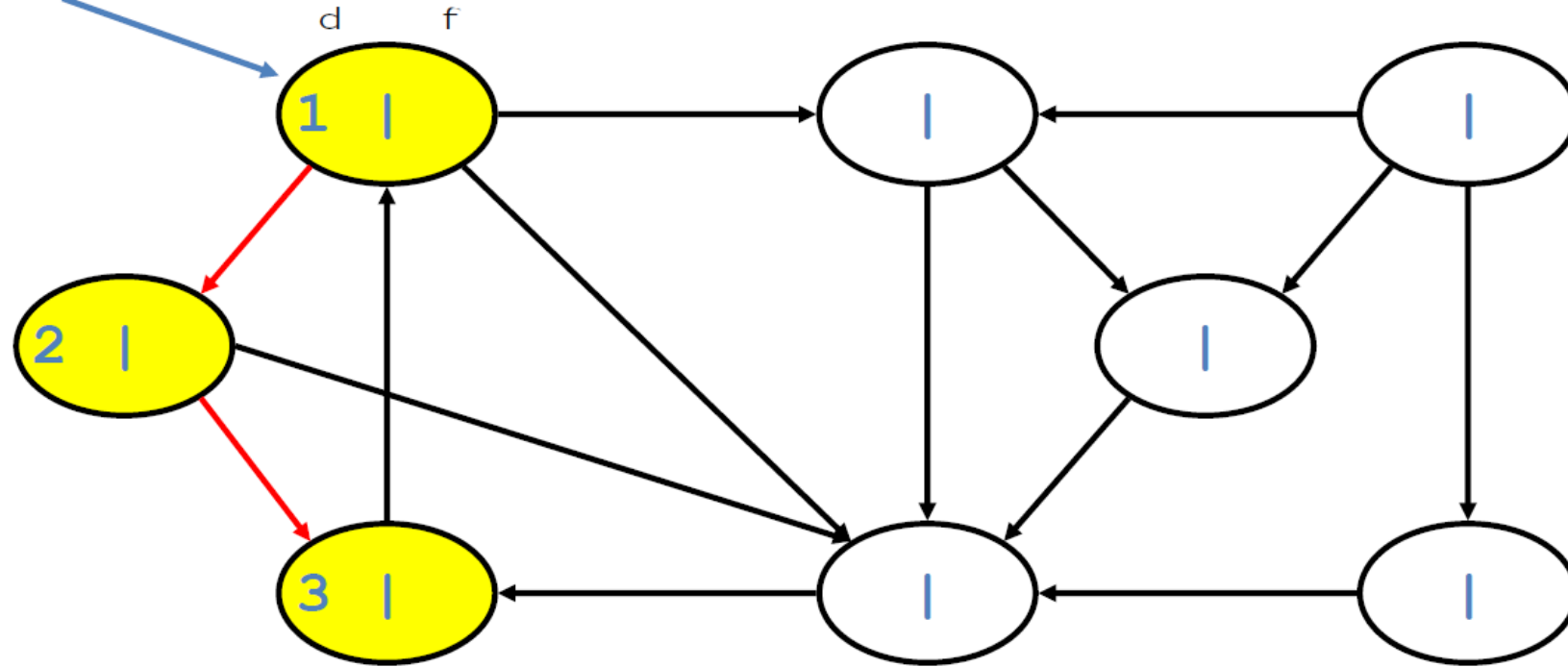


```
DFS_Visit(u)
{
    u->color = AMARILLO;
    tiempo = tiempo+1;
    u->d = tiempo;
    for each v ∈ u->Adj[]
    {
        if (v->color== BLANCO)
            DFS_Visit(v);
    }
    u->color = NEGRO;
    tiempo = tiempo +1;
    u->f = tiempo;
}
```

Traza de Recorrido en Profundidad

DFS (4/17)

vértice
inicial

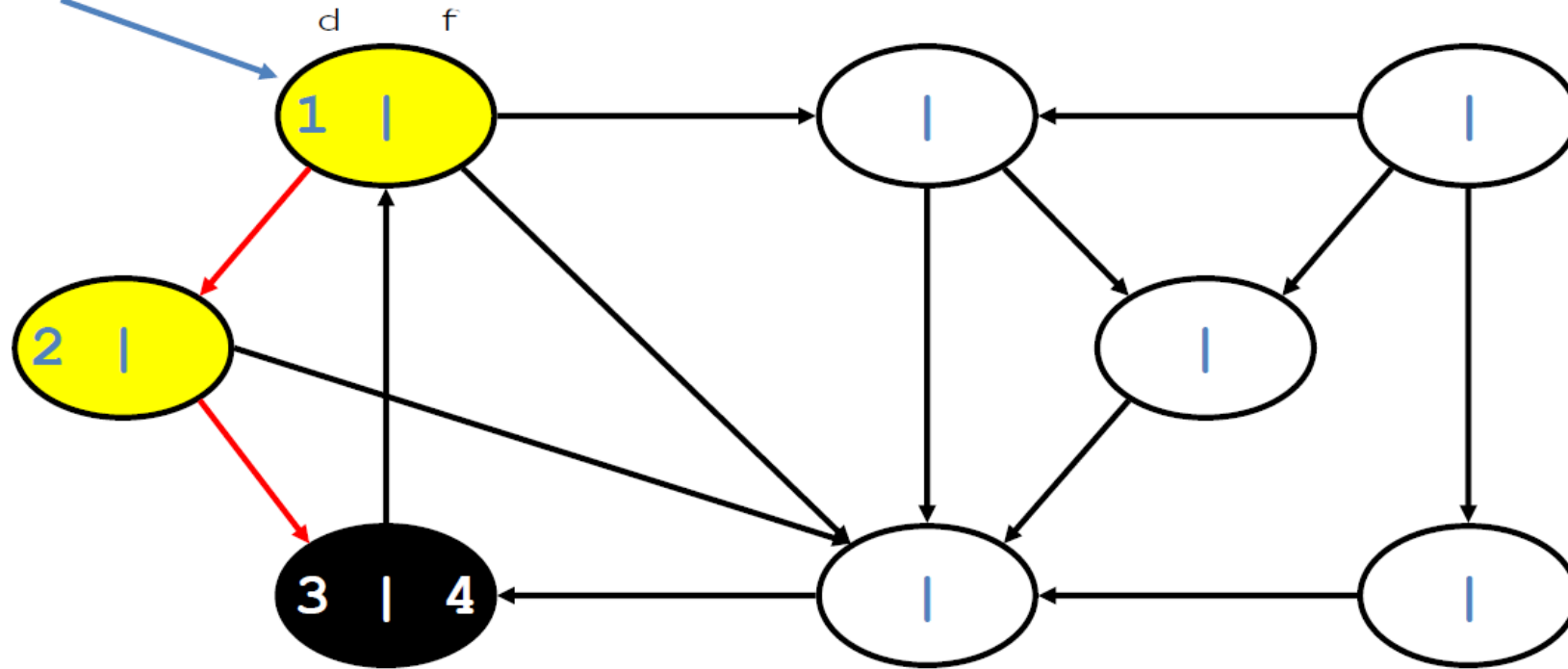


```
DFS_Visit(u)
{
    u->color = AMARILLO;
    tiempo = tiempo+1;
    u->d = tiempo;
    for each v ∈ u->Adj[]
    {
        if (v->color== BLANCO)
            DFS_Visit(v);
    }
    u->color = NEGRO;
    tiempo = tiempo +1;
    u->f = tiempo;
}
```


Traza de Recorrido en Profundidad

DFS (5/17)

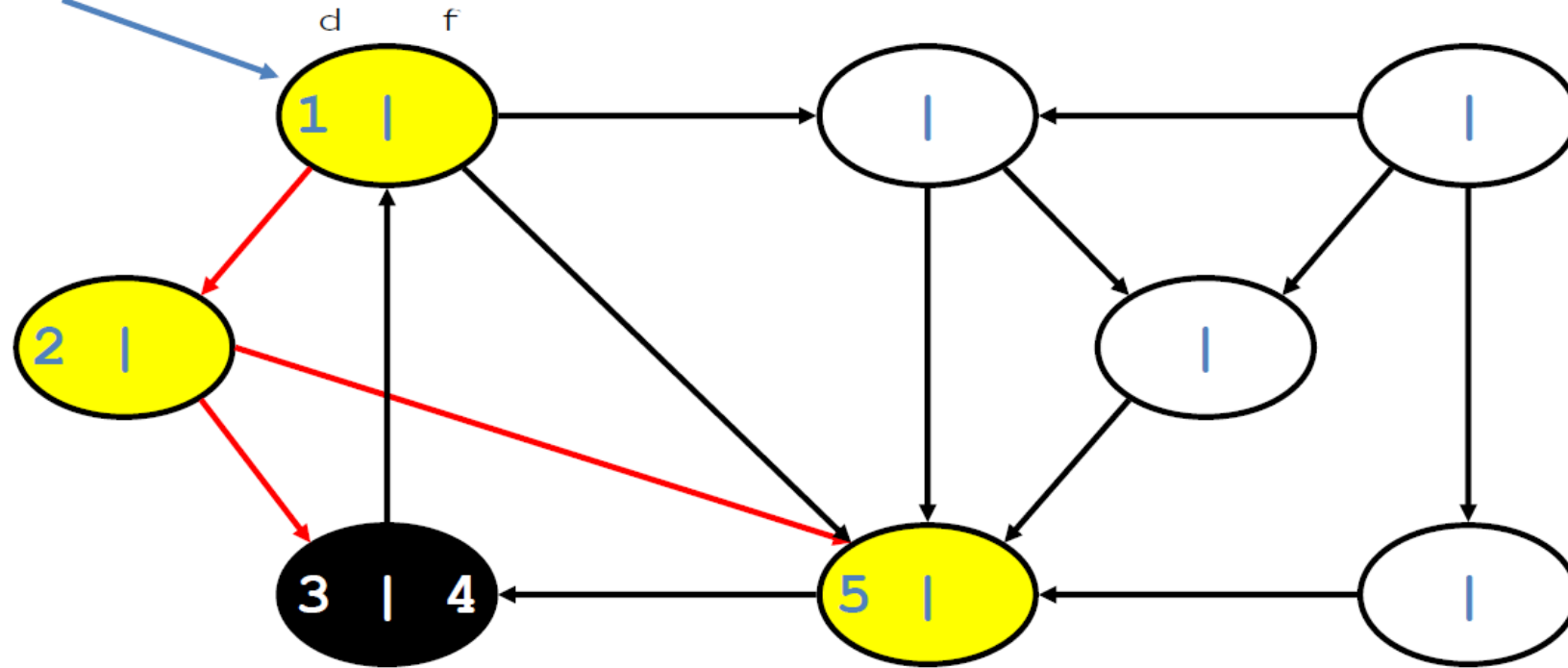
vértice
inicial



Traza de Recorrido en Profundidad

DFS (6/17)

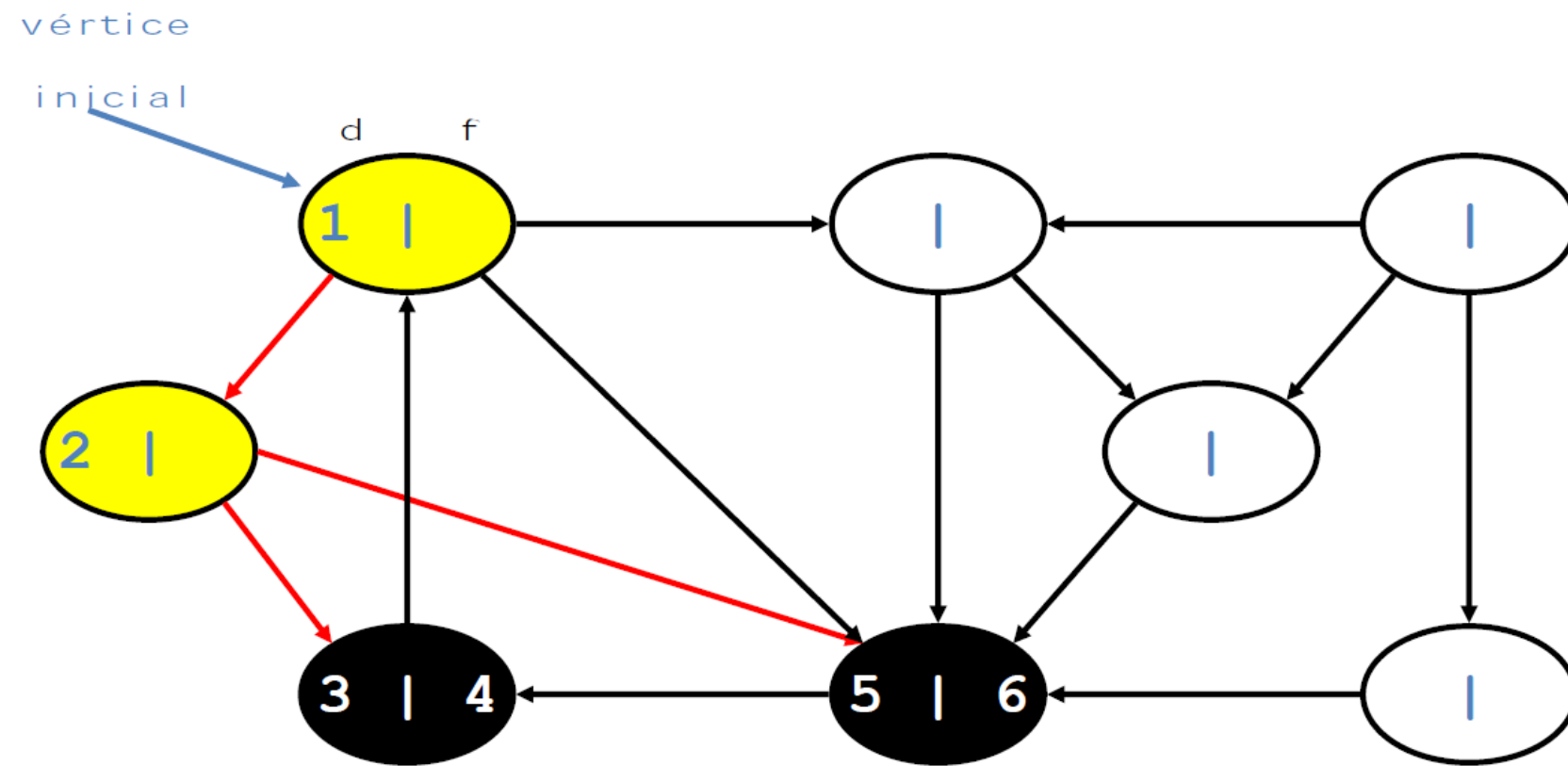
vértice
inicial



```
DFS_Visit(u)
{
    u->color = AMARILLO;
    tiempo = tiempo+1;
    u->d = tiempo;
    for each v ∈ u->Adj[]
    {
        if (v->color== BLANCO)
            DFS_Visit(v);
    }
    u->color = NEGRO;
    tiempo = tiempo +1;
    u->f = tiempo;
}
```

Traza de Recorrido en Profundidad

DFS (7/17)



```
DFS_Visit(u)
{
    u->color = AMARILLO;
    tiempo = tiempo+1;
    u->d = tiempo;
    for each v ∈ u->Adj[]
    {
        if (v->color== BLANCO)
            DFS_Visit(v);
    }
    u->color = NEGRO;
    tiempo = tiempo +1;
    u->f = tiempo;
}
```

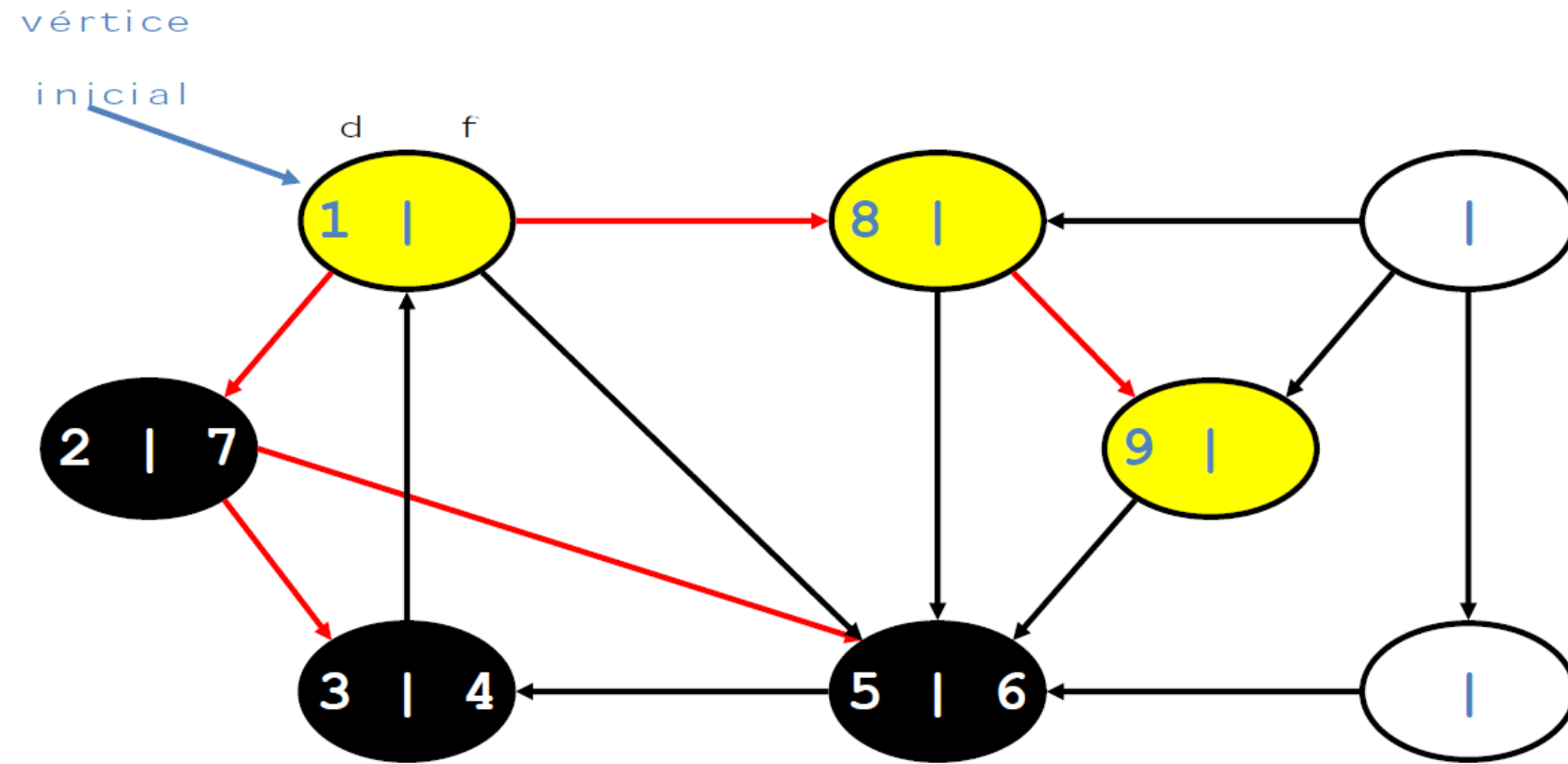
DFS (8/17)



```
DFS_Visit(u)
{
    u->color = AMARILLO;
    tiempo = tiempo+1;
    u->d = tiempo;
    for each v  $\in$  u->Adj[]
    {
        if (v->color== BLANCO)
            DFS_Visit(v);
    }
    u->color = NEGRO;
    tiempo = tiempo +1;
    u->f = tiempo;
}
```

Traza de Recorrido en Profundidad

DFS (10/17)



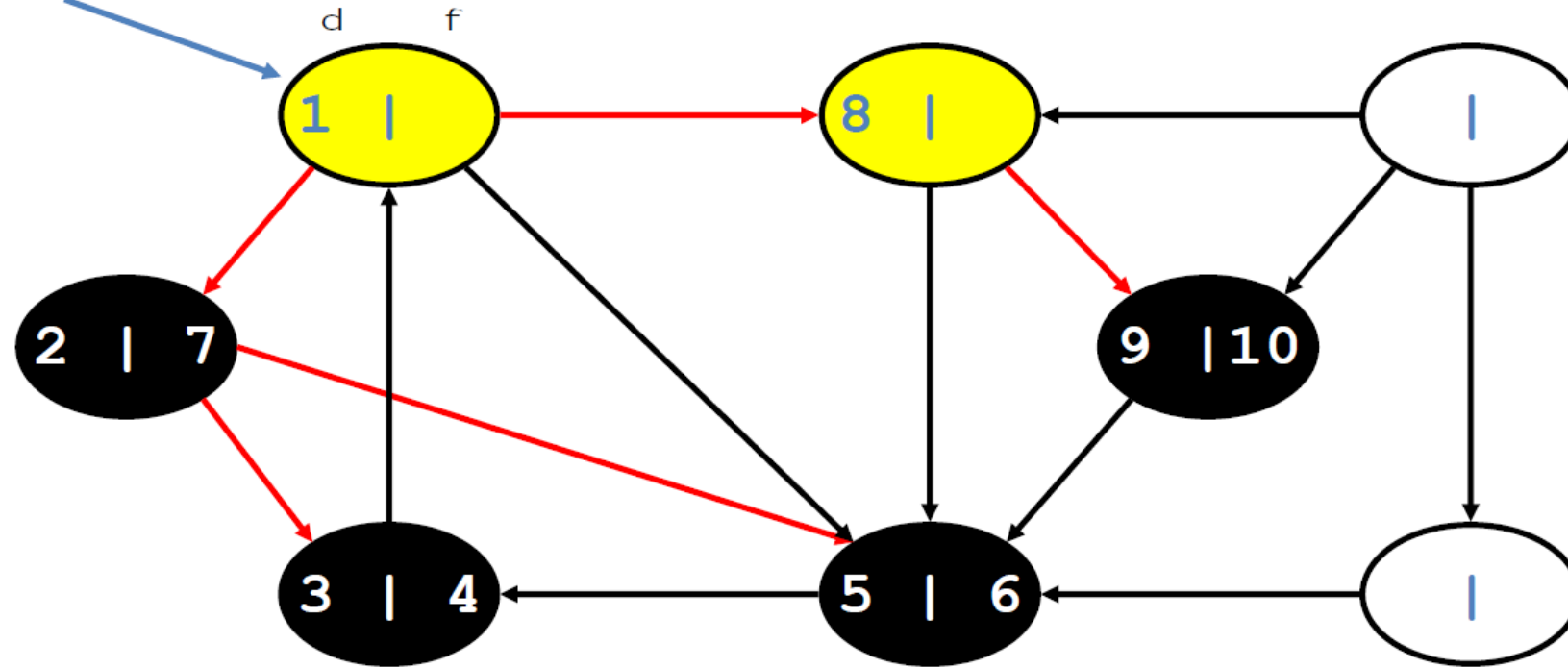
```
DFS_Visit(u)
{
    u->color = AMARILLO;
    tiempo = tiempo+1;
    u->d = tiempo;
    for each v ∈ u->Adj[]
    {
        if (v->color== BLANCO)
            DFS_Visit(v);
    }
    u->color = NEGRO;
    tiempo = tiempo +1;
    u->f = tiempo;
}
```


Traza de Recorrido en Profundidad

DFS (11/17)

vértice

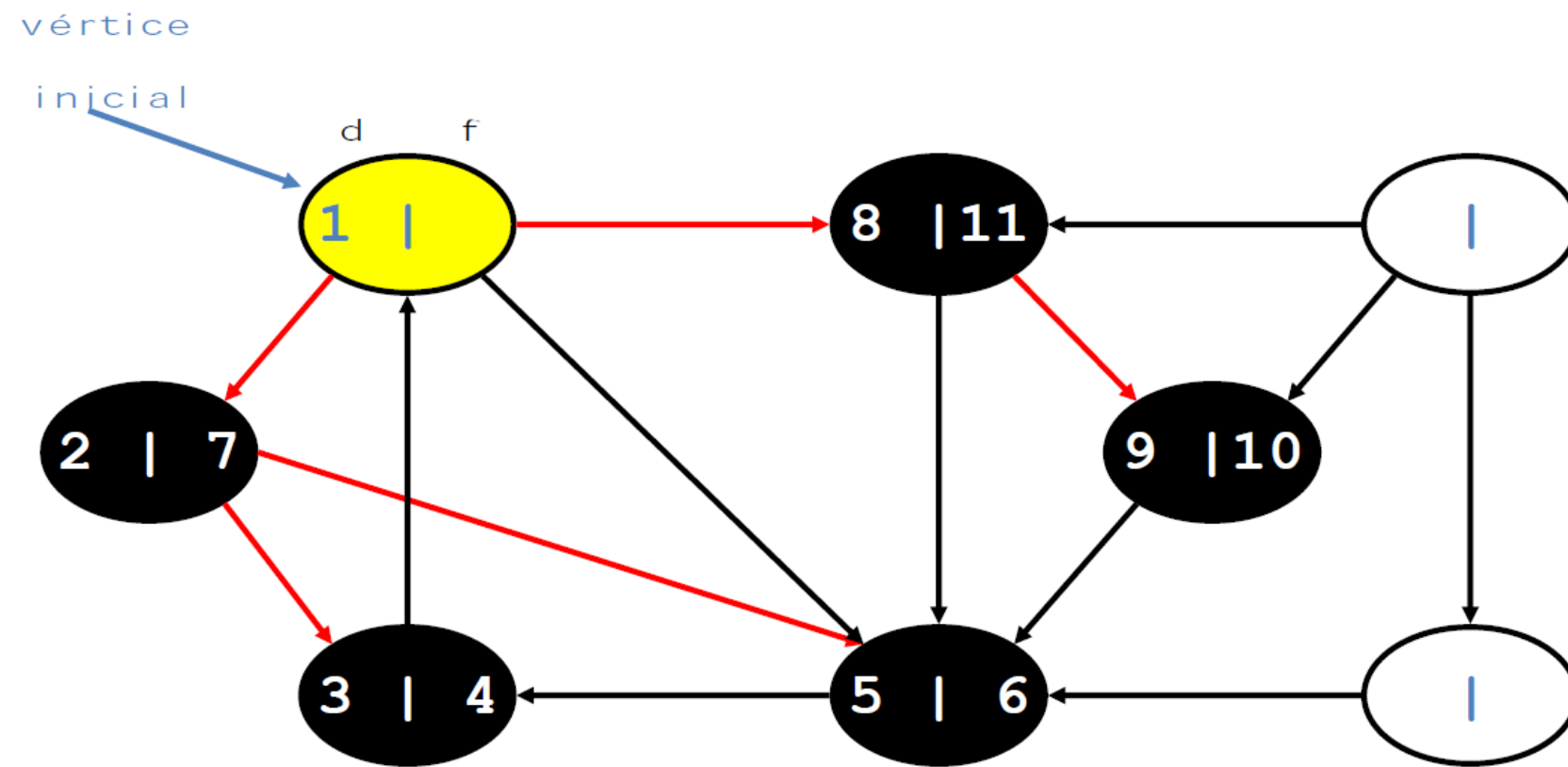
inicial



```
DFS_Visit(u)
{
    u->color = AMARILLO;
    tiempo = tiempo+1;
    u->d = tiempo;
    for each v ∈ u->Adj[]
    {
        if (v->color== BLANCO)
            DFS_Visit(v);
    }
    u->color = NEGRO;
    tiempo = tiempo +1;
    u->f = tiempo;
}
```

Traza de Recorrido en Profundidad

DFS (12/17)



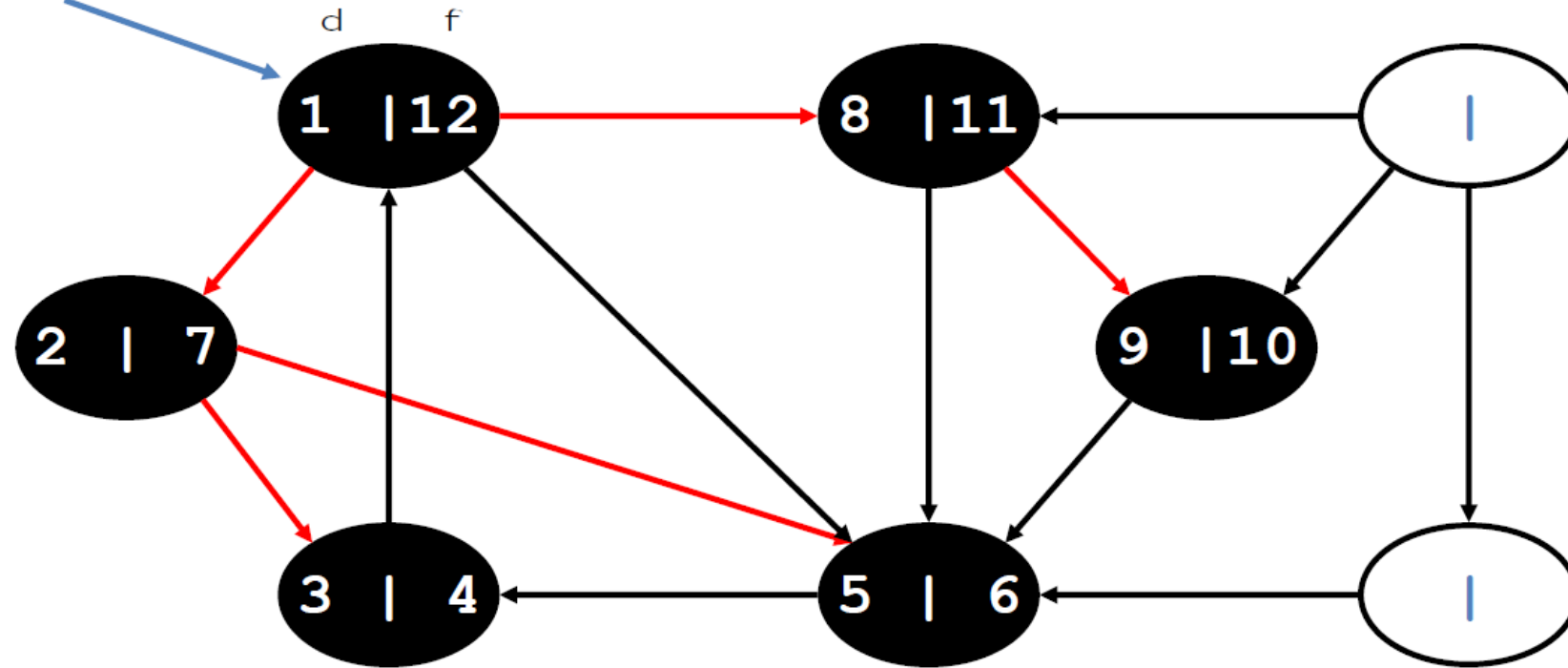
```
DFS_Visit(u)
{
    u->color = AMARILLO;
    tiempo = tiempo+1;
    u->d = tiempo;
    for each v ∈ u->Adj[]
    {
        if (v->color== BLANCO)
            DFS_Visit(v);
    }
    u->color = NEGRO;
    tiempo = tiempo +1;
    u->f = tiempo;
}
```

Traza de Recorrido en Profundidad

DFS (13/17)

vértice

inicial

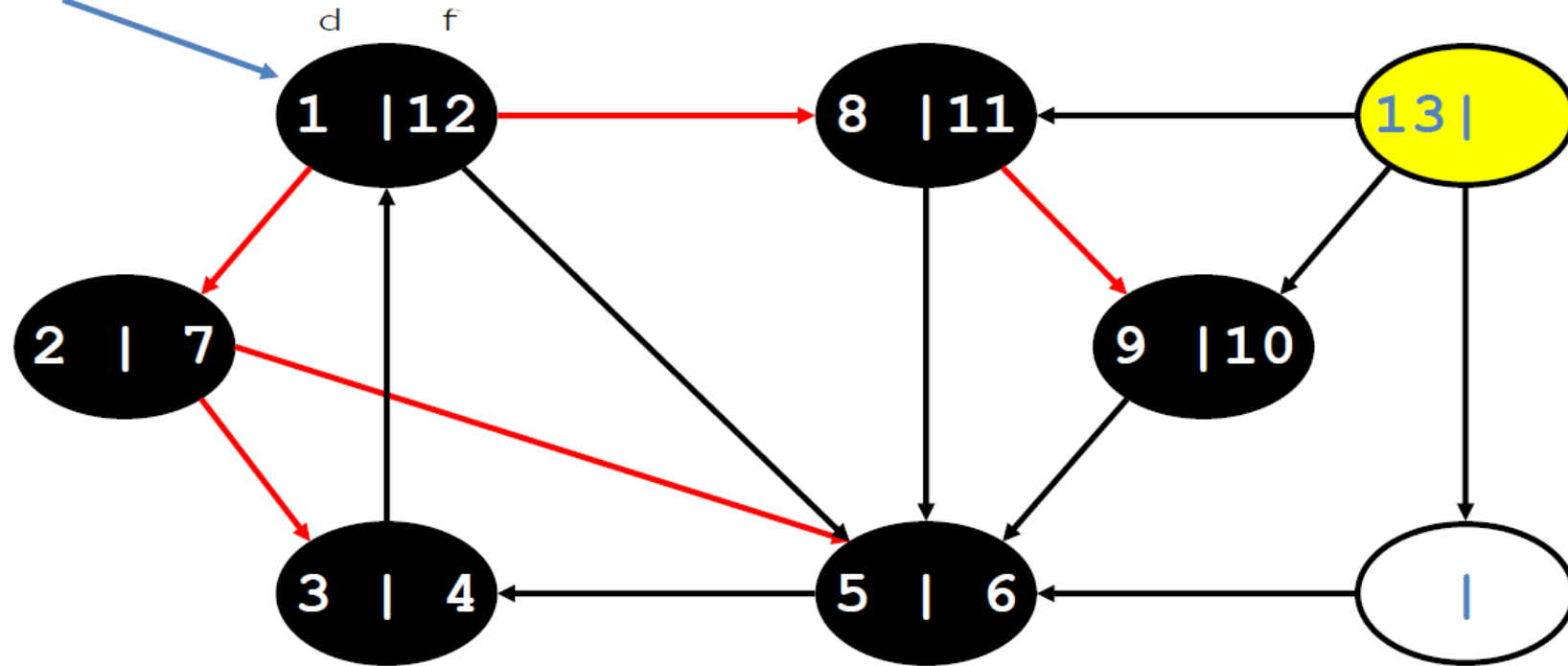


```
DFS_Visit(u)
{
    u->color = AMARILLO;
    tiempo = tiempo+1;
    u->d = tiempo;
    for each v ∈ u->Adj[]
    {
        if (v->color== BLANCO)
            DFS_Visit(v);
    }
    u->color = NEGRO;
    tiempo = tiempo +1;
    u->f = tiempo;
}
```

Traza de Recorrido en Profundidad

DFS (14/17)

vértice
inicial



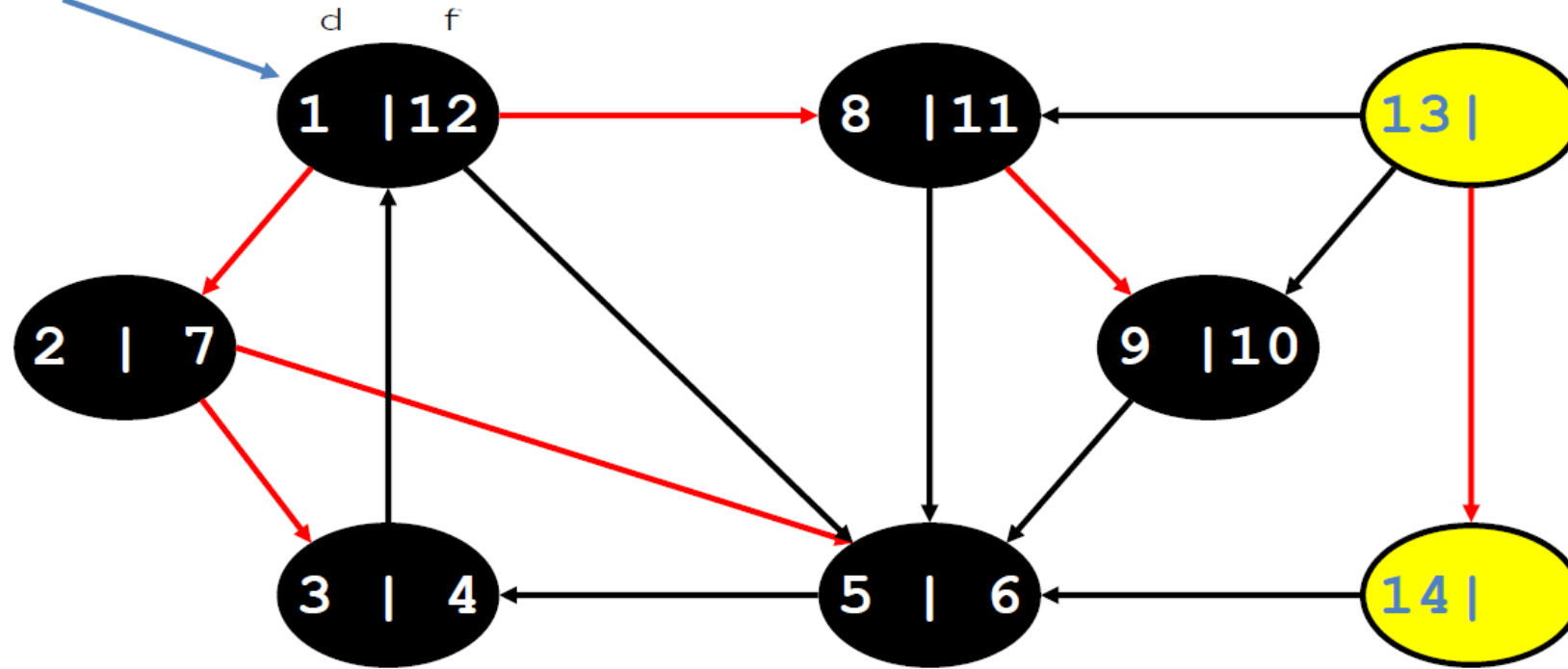
```
DFS(G)
{
    for each vértice u ∈ G->V
    {
        u->color = BLANCO;
    }
    tiempo = 0;
    for each vértice u ∈ G->V
    {
        if (u->color == BLANCO)
            DFS_Visit(u);
    }
}
```

```
DFS_Visit(u)
{
    u->color = AMARILLO;
    tiempo = tiempo + 1;
    u->d = tiempo;
    for each v ∈ u->Adj[]
    {
        if (v->color == BLANCO)
            DFS_Visit(v);
    }
    u->color = NEGRO;
    tiempo = tiempo + 1;
    u->f = tiempo;
}
```

Traza de Recorrido en Profundidad

DFS (15/17)

vértice
inicial



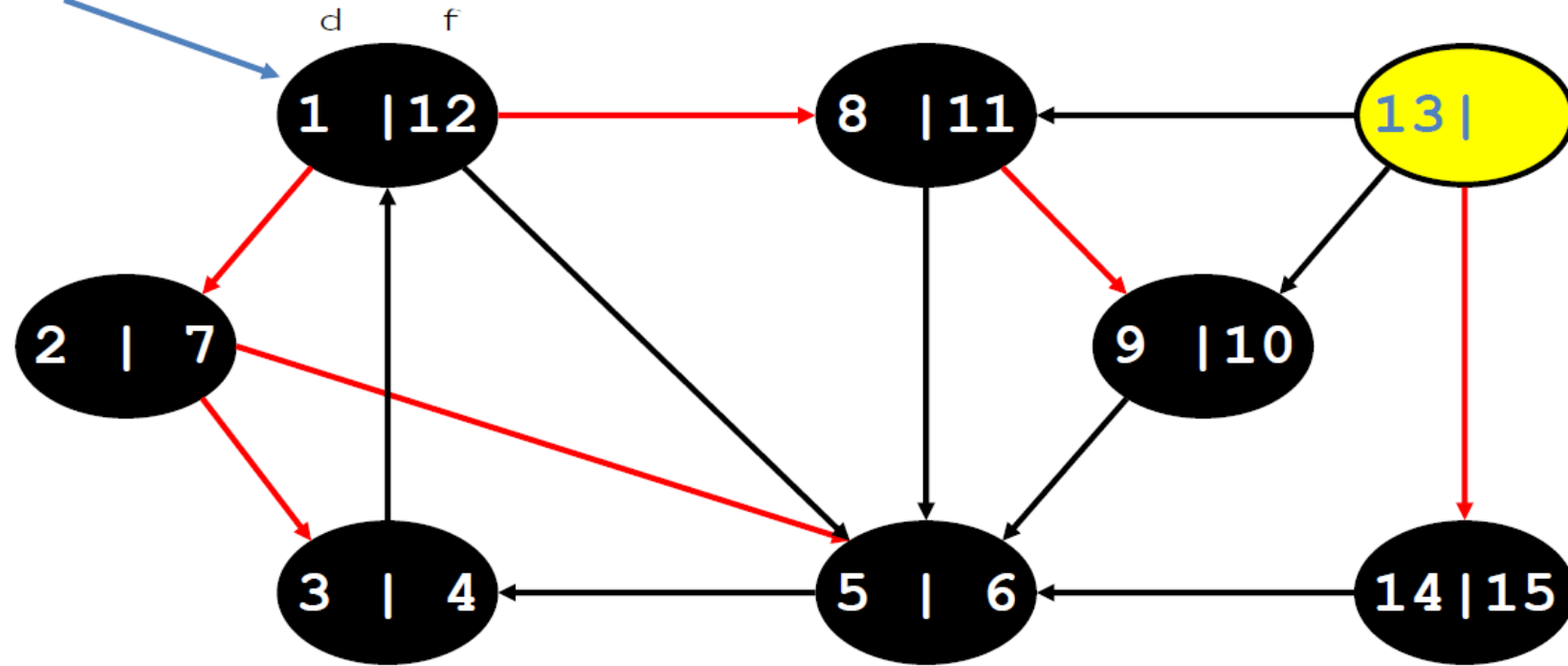
```
DFS_Visit(u)
{
    u->color = AMARILLO;
    tiempo = tiempo+1;
    u->d = tiempo;
    for each v ∈ u->Adj[]
    {
        if (v->color== BLANCO)
            DFS_Visit(v);
    }
    u->color = NEGRO;
    tiempo = tiempo +1;
    u->f = tiempo;
}
```


Traza de Recorrido en Profundidad

DFS (16/17)

vértice

inicial

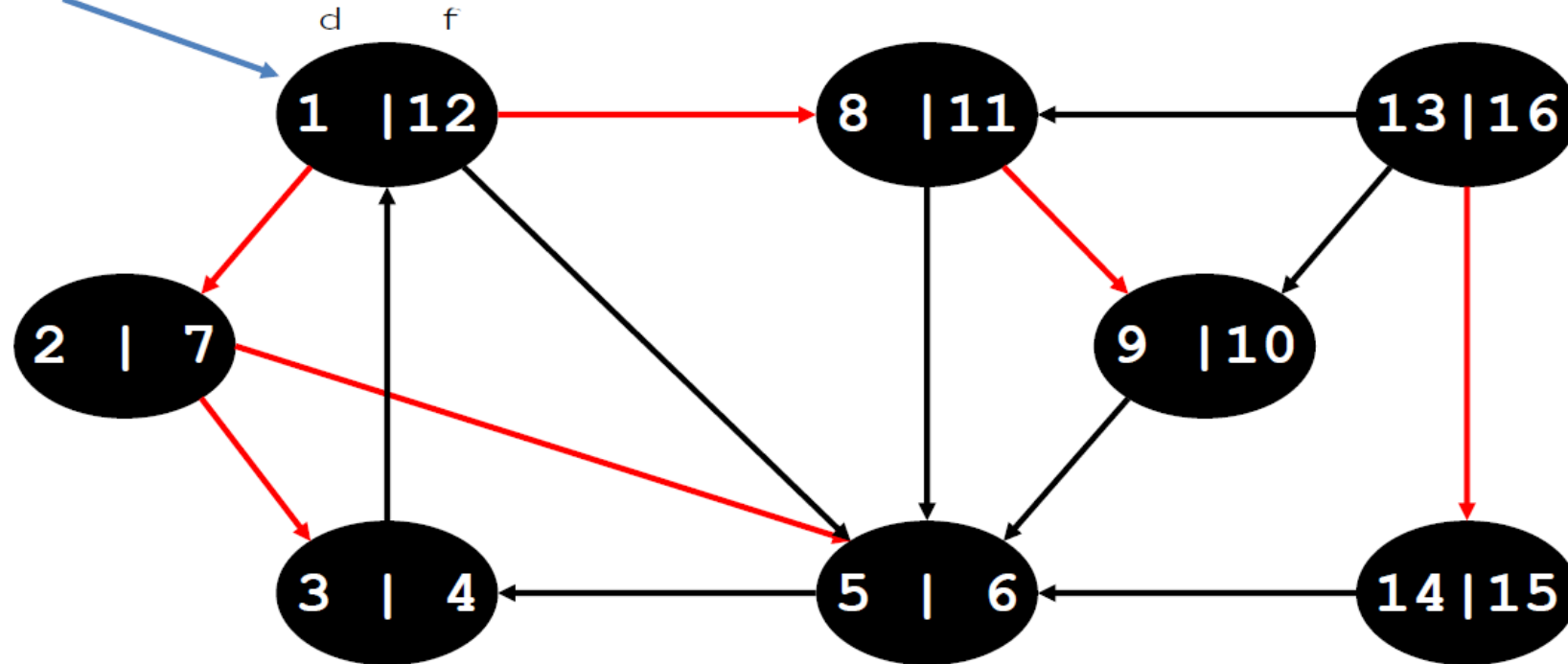


```
DFS_Visit(u)
{
    u->color = AMARILLO;
    tiempo = tiempo+1;
    u->d = tiempo;
    for each v ∈ u->Adj[]
    {
        if (v->color== BLANCO)
            DFS_Visit(v);
    }
    u->color = NEGRO;
    tiempo = tiempo +1;
    u->f = tiempo;
}
```

Traza de Recorrido en Profundidad

DFS (17/17)

vértice
inicial



```
DFS(G)
{
    for each vértice u ∈ G->V
    {
        u->color = BLANCO;
    }
    tiempo = 0;
    for each vértice u ∈ G->V
    {
        if (u->color == BLANCO)
            DFS_Visit(u);
    }
}

DFS_Visit(u)
{
    u->color = AMARILLO;
    tiempo = tiempo + 1;
    u->d = tiempo;
    for each v ∈ u->Adj[]
    {
        if (v->color == BLANCO)
            DFS_Visit(v);
    }
    u->color = NEGRO;
    tiempo = tiempo + 1;
    u->f = tiempo;
}
```

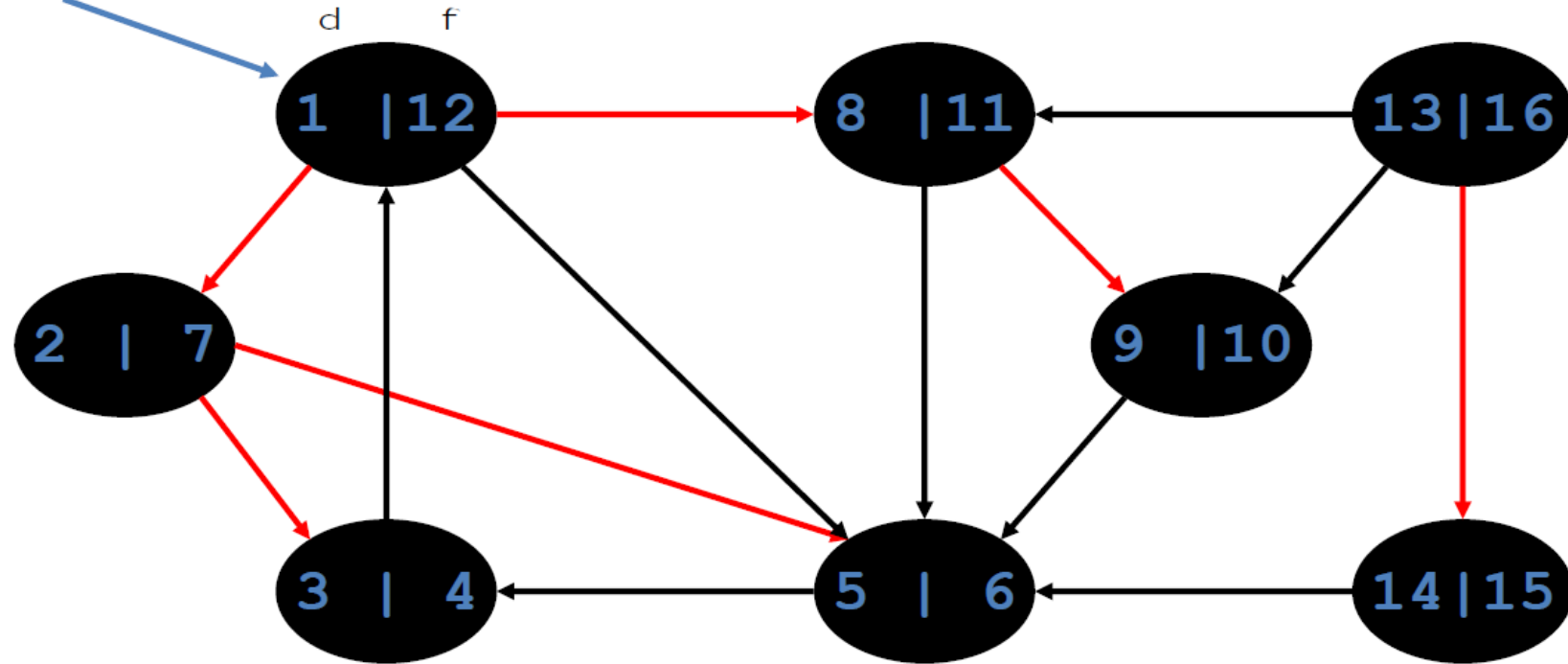
DFS: Clases de Arcos

DFS introduce una distinción importante entre arcos en el grafo original:

- *Arcos Tree* : encuentra vértices nuevos (blancos)
- Los arcos tree forman un bosque abarcador

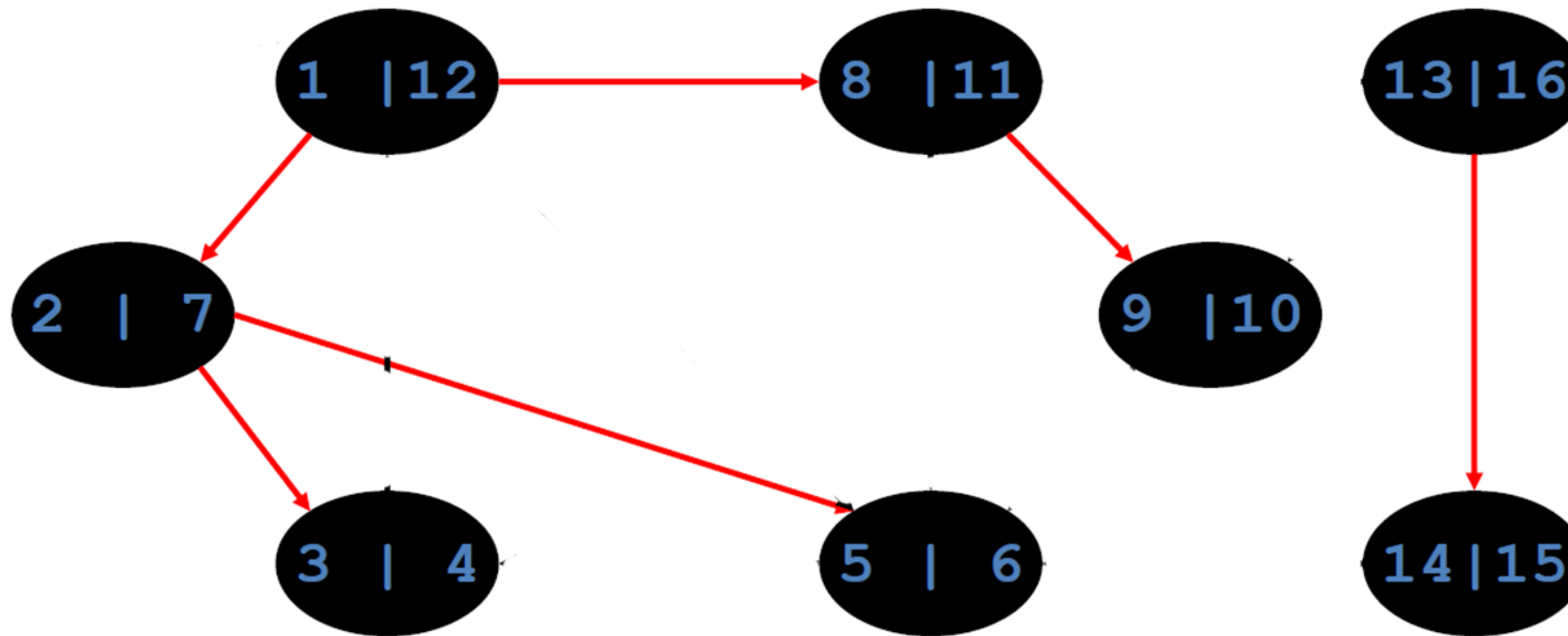
Clasificación de Arcos: TREE

vértice
inicial



Arco Tree

Clasificación de Arcos: TREE



Arco Tree

UN BOSQUE ABARCADOR EN PROFUNDIDAD.

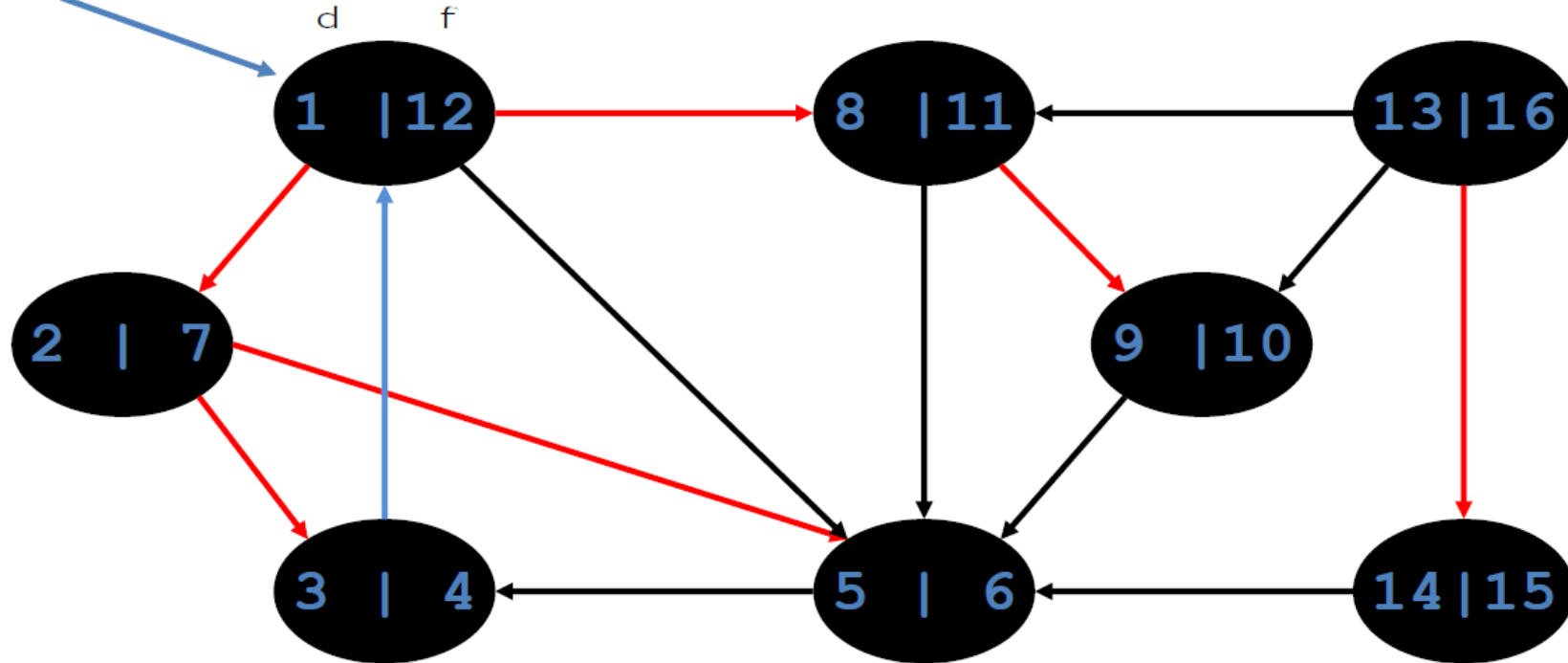
DFS: Clases de arcos

DFS introduce una distinción importante entre arcos en el grafo original:

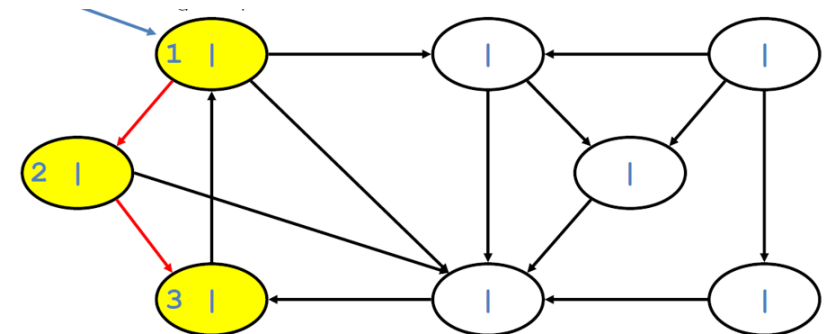
- *Arcos Back*: desde el descendiente al ancestro
- Encuentra un vértice amarillo (de amarillo a amarillo)

Clasificación de Arcos: Back

vértice
inicial



Arco Tree Arco Back



DFS y Ciclos en los grafos


Un Grafo es acíclico si y sólo si en el recorrido DFS no se encuentran arcos de tipo "back". O sea nunca se intenta ir de un vértice amarillo a otro amarillo.

Podemos usar el recorrido DFS entonces para hacer lo que se conoce como test de aciclicidad de un grafo, que es determinar si el grafo es cíclico o acíclico.

Cómo modificarías el pseudocódigo para detectar ciclos?

```
DFS(G)
{
    for each vértice  $u \in G \rightarrow V$ 
    {
         $u \rightarrow \text{color} = \text{BLANCO}$ ;
    }
    tiempo = 0;
    for each vértice  $u \in G \rightarrow V$ 
    {
        if ( $u \rightarrow \text{color} == \text{BLANCO}$ )
            DFS_Visit(u);
    }
}
```

```
DFS_Visit(u)
{
     $u \rightarrow \text{color} = \text{AMARILLO}$ ;
    tiempo = tiempo + 1;
     $u \rightarrow d = \text{tiempo}$ ;
    for each  $v \in u \rightarrow \text{Adj}[]$ 
    {
        if ( $v \rightarrow \text{color} == \text{BLANCO}$ )
            DFS_Visit(v);
    }
     $u \rightarrow \text{color} = \text{NEGRO}$ ;
    tiempo = tiempo + 1;
     $u \rightarrow f = \text{tiempo}$ ;
}
```

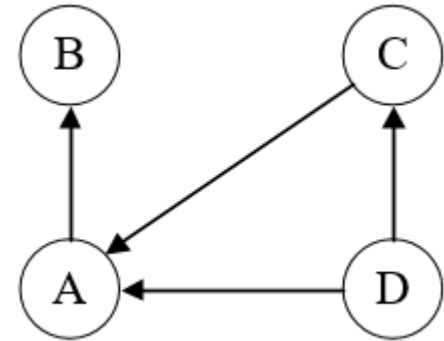
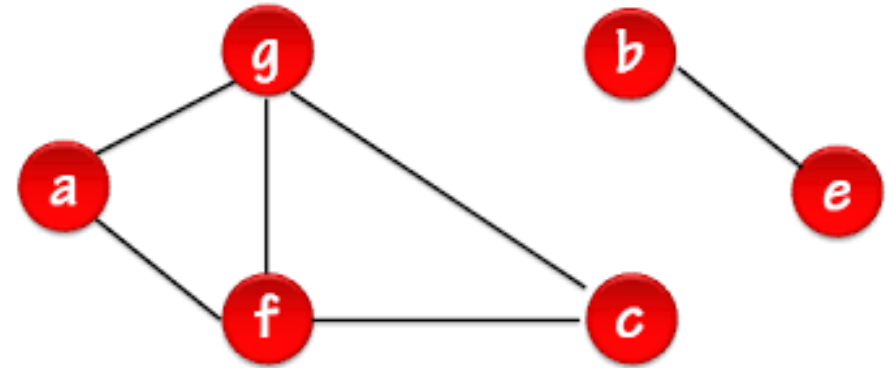


else
if ($v \rightarrow \text{color} == \text{AMARILLO}$)
print("Hay Ciclo!");

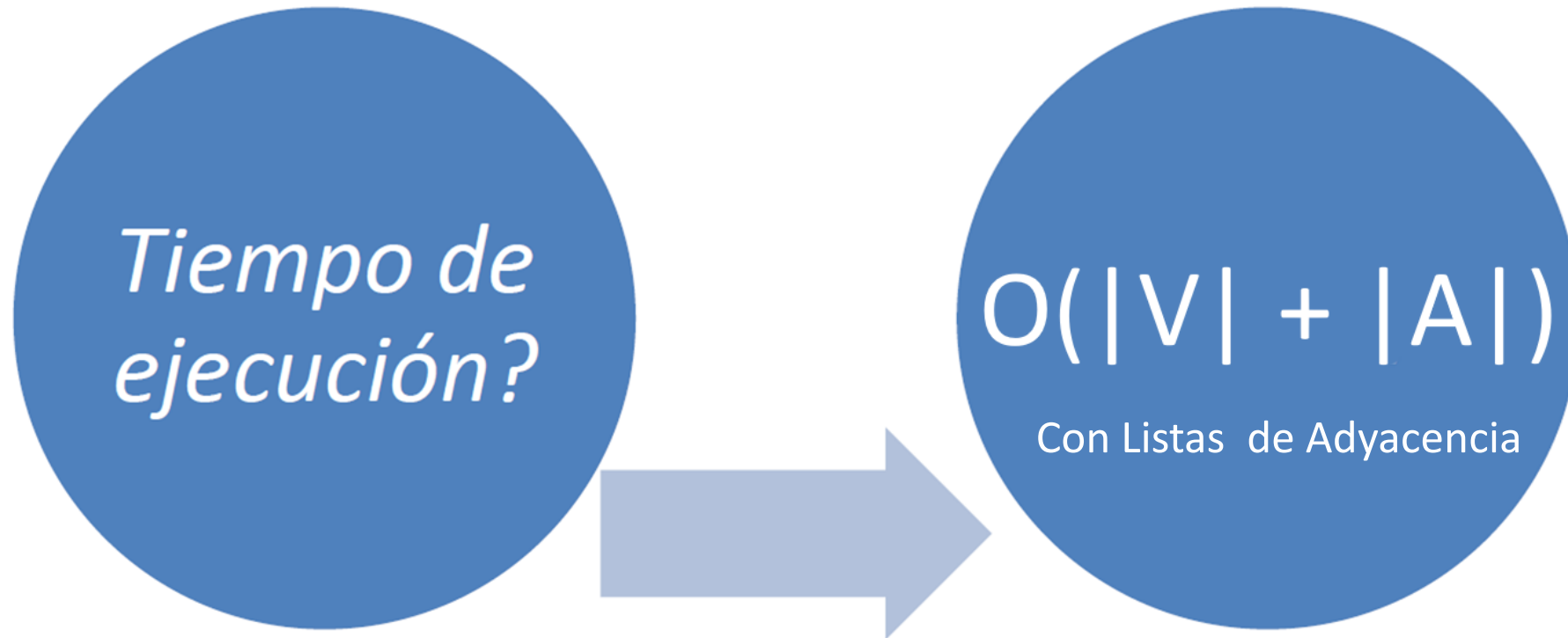
El DFS siempre visita todos los nodos sin importar la forma del grafo.

```
DFS(G)
{
    for each vértice u ∈ G->V
    {
        u->color = BLANCO;
    }
    tiempo = 0;
    for each vértice u ∈ G->V
    {
        if (u->color == BLANCO)
            DFS_Visit(u);
    }
}
```

```
DFS_Visit(u)
{
    u->color = AMARILLO;
    tiempo = tiempo + 1;
    u->d = tiempo;
    for each v ∈ u->Adj[]
    {
        if (v->color == BLANCO)
            DFS_Visit(v);
    }
    u->color = NEGRO;
    tiempo = tiempo + 1;
    u->f = tiempo;
}
```



DFS: Complejidad Computacional

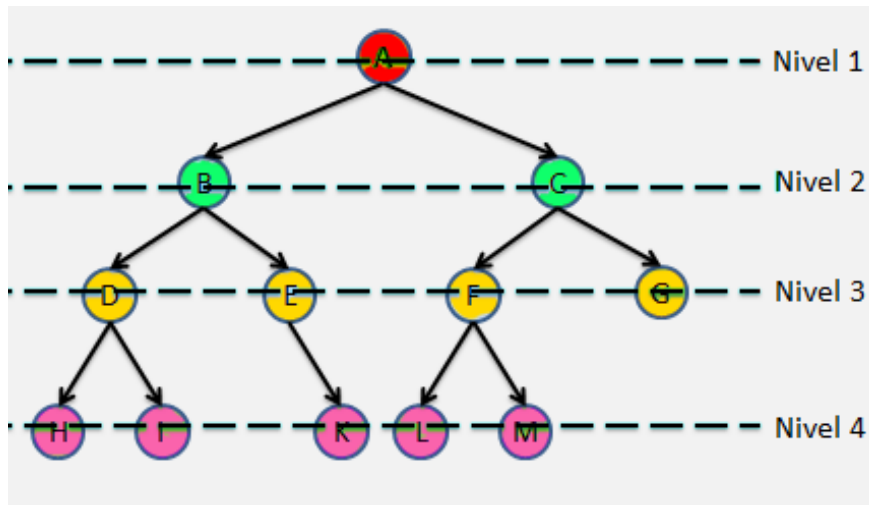


PASA UN VEZ POR CADA NODO, Y COMO MUCHO UNA VEZ POR CADA ARCO

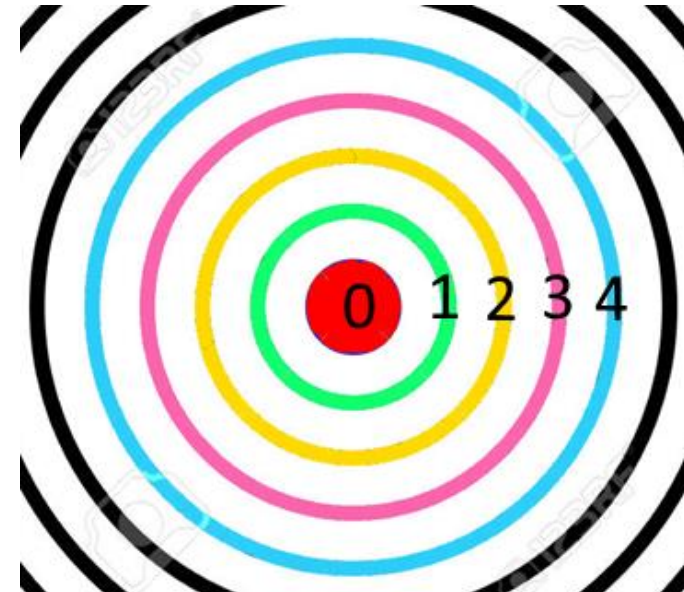
BFS (Breadth-First Search)

Recorrido en Amplitud

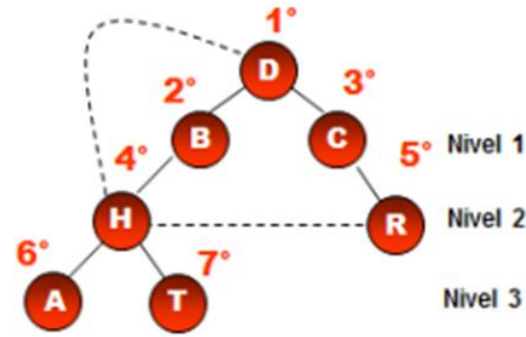
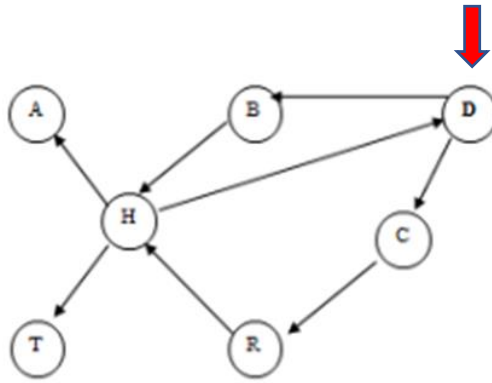
- Es una generalización de un recorrido por niveles de un árbol.



- Se realiza un recorrido por capas del grafo.
- Los nodos de una capa tienen la misma distancia al origen.



BFS



Recorrido desde Vertice $D = \{D, B, C, H, R, A, T\}$

Para hacerlo nos valemos de una estructura de fila (cola, queue):

BFS (Grafo G):

Vaciar la fila F .

Para cada vértices v de G

Marcar v como NO_VISITADO.

Para cada vértice v de G

Si v es NO_VISITADO:

BFS(G, v)

BFS (Grafo G , vértice s):

Marcar el vértice s como VISITADO.

Agregar s a la fila F .

Mientras la fila F no esté vacía

Tomamos vértice x de la fila,

Para cada vértice y adyacente a x :

Si y es NO_VISITADO :

Marcar el vértice y como VISITADO.

Agregar y a la fila F .

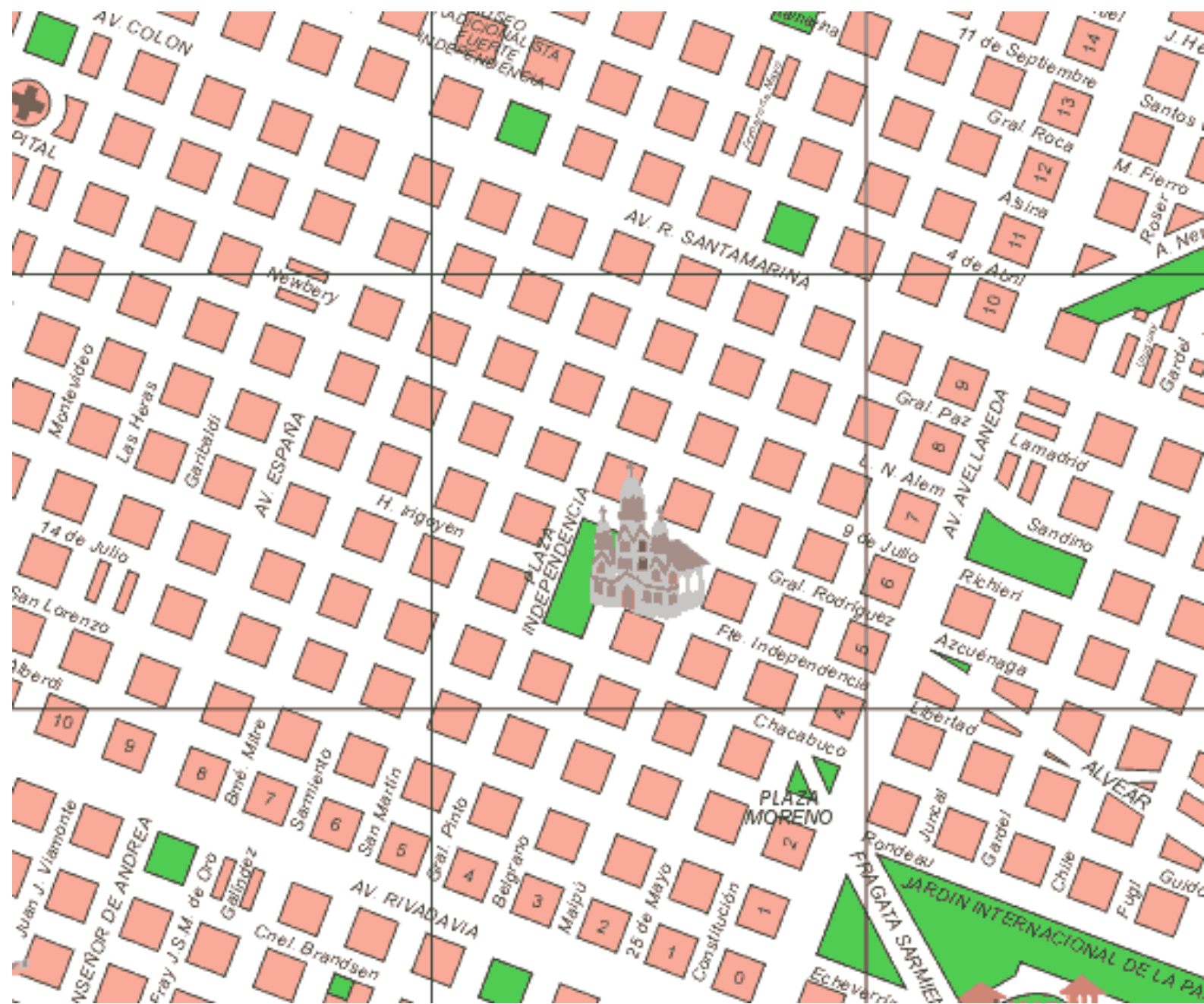
- 1) D
- 2) BC
- 3) CH
- 4) HR
- 5) RAT
- 6) AT
- 7) T
- 8)

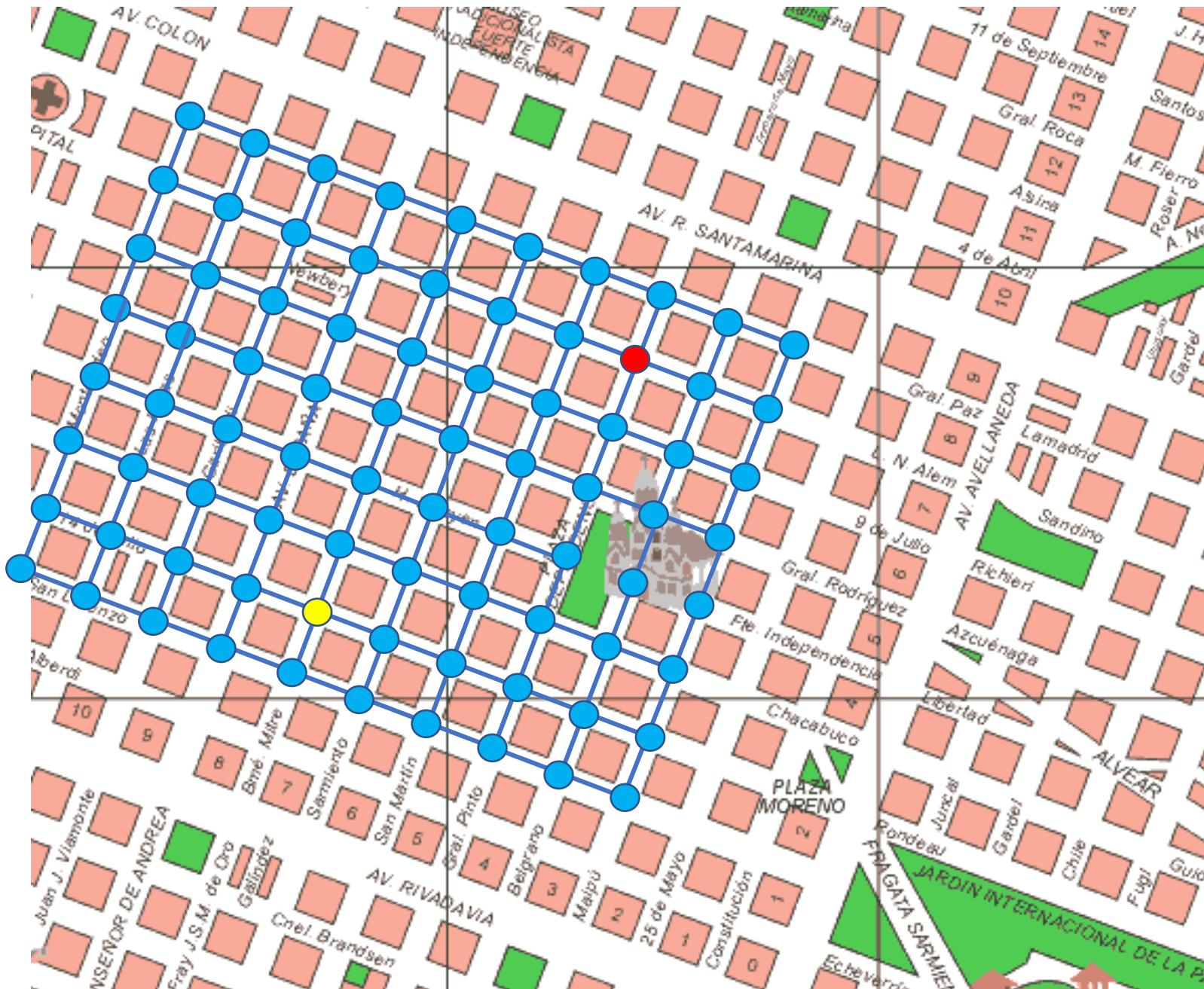
Es un planteo iterativo, no recursivo.



Problema clásico

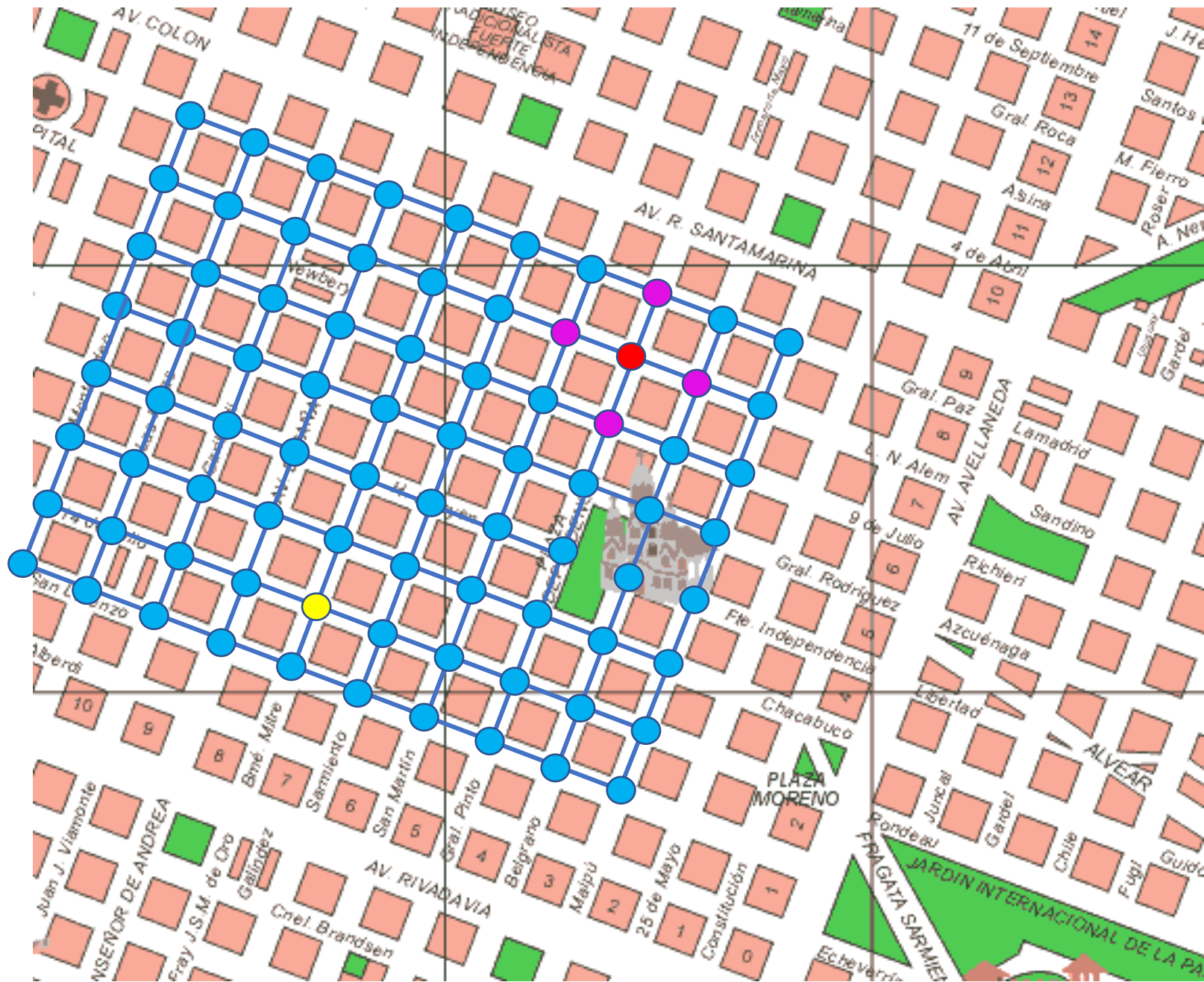
- Tenemos un Grafo no Pesado (los arcos no tienen peso asociado). ¿Cómo encontrar el camino con menor longitud (menor cantidad de arcos) entre un vértice **v** y otro vértice **w**?
- Estrategia: Visitar primero todos los vértices de distancia 1 desde **v**, luego los de distancia 2, luego los de 3, y así sucesivamente. A medida que visitamos los vértices verificamos si es el **w** que buscamos, la distancia será el nivel donde lo encontremos.
- Lo resolvemos haciendo un BFS desde **v** controlando cuando llegamos a **w**.

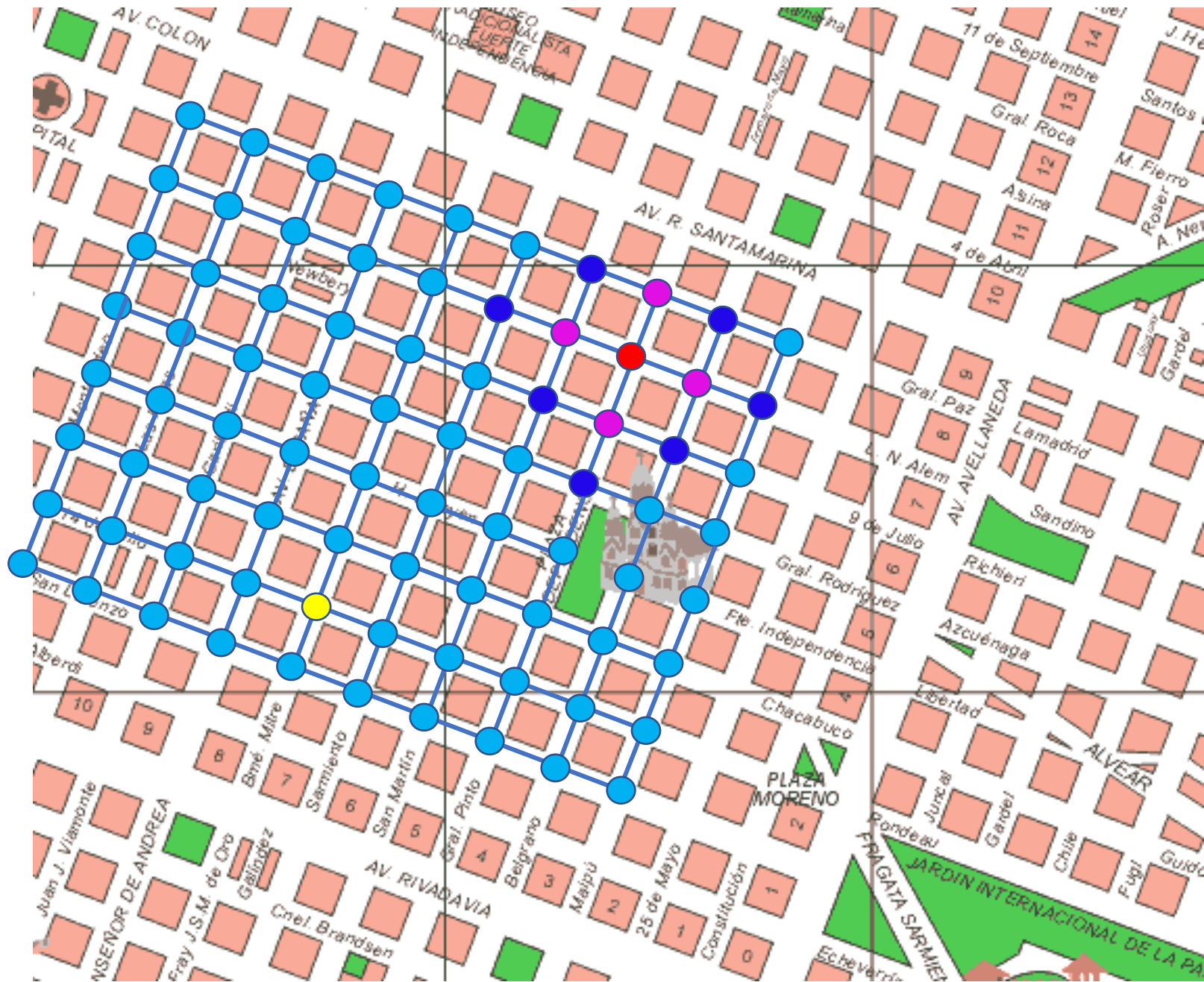


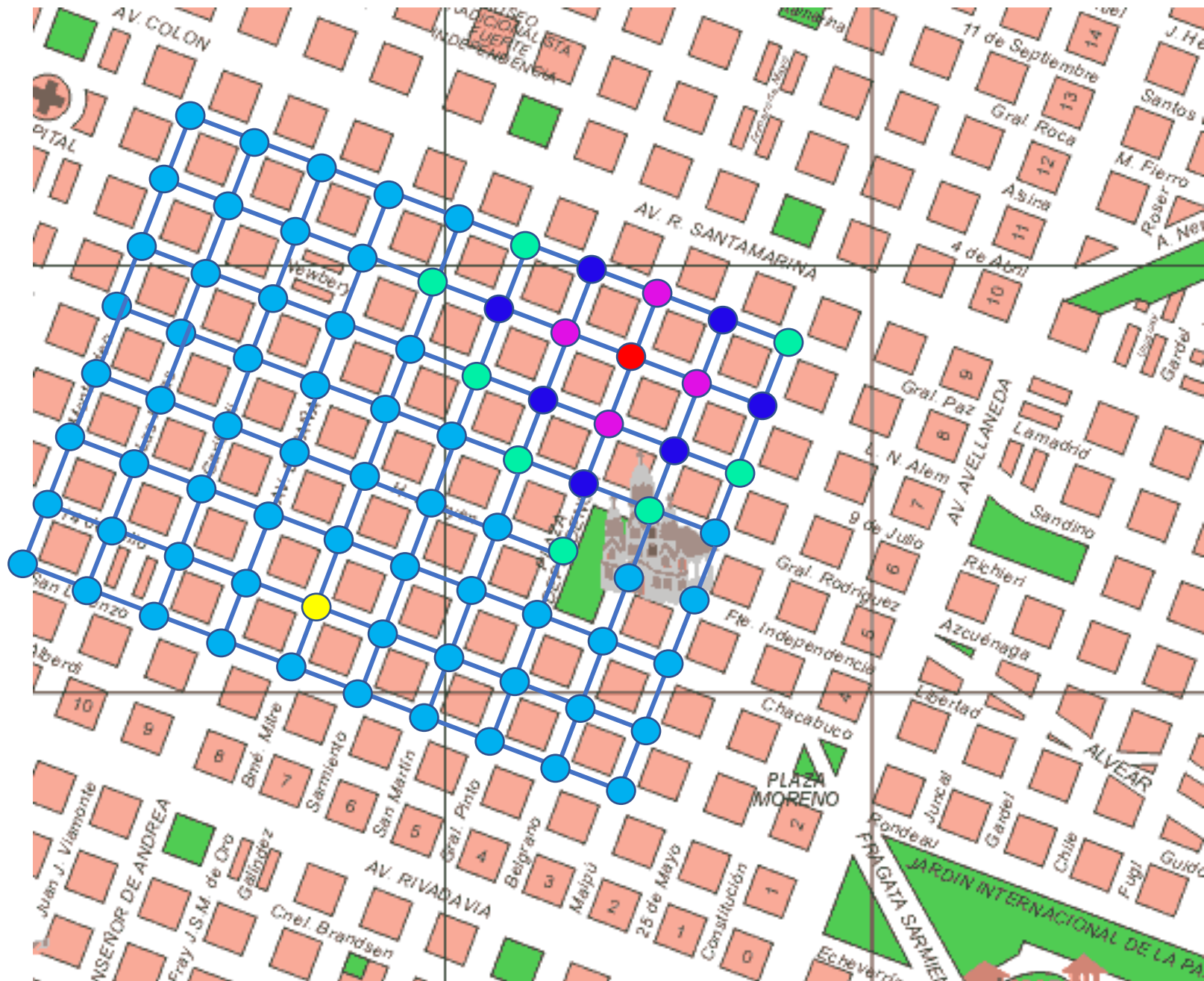


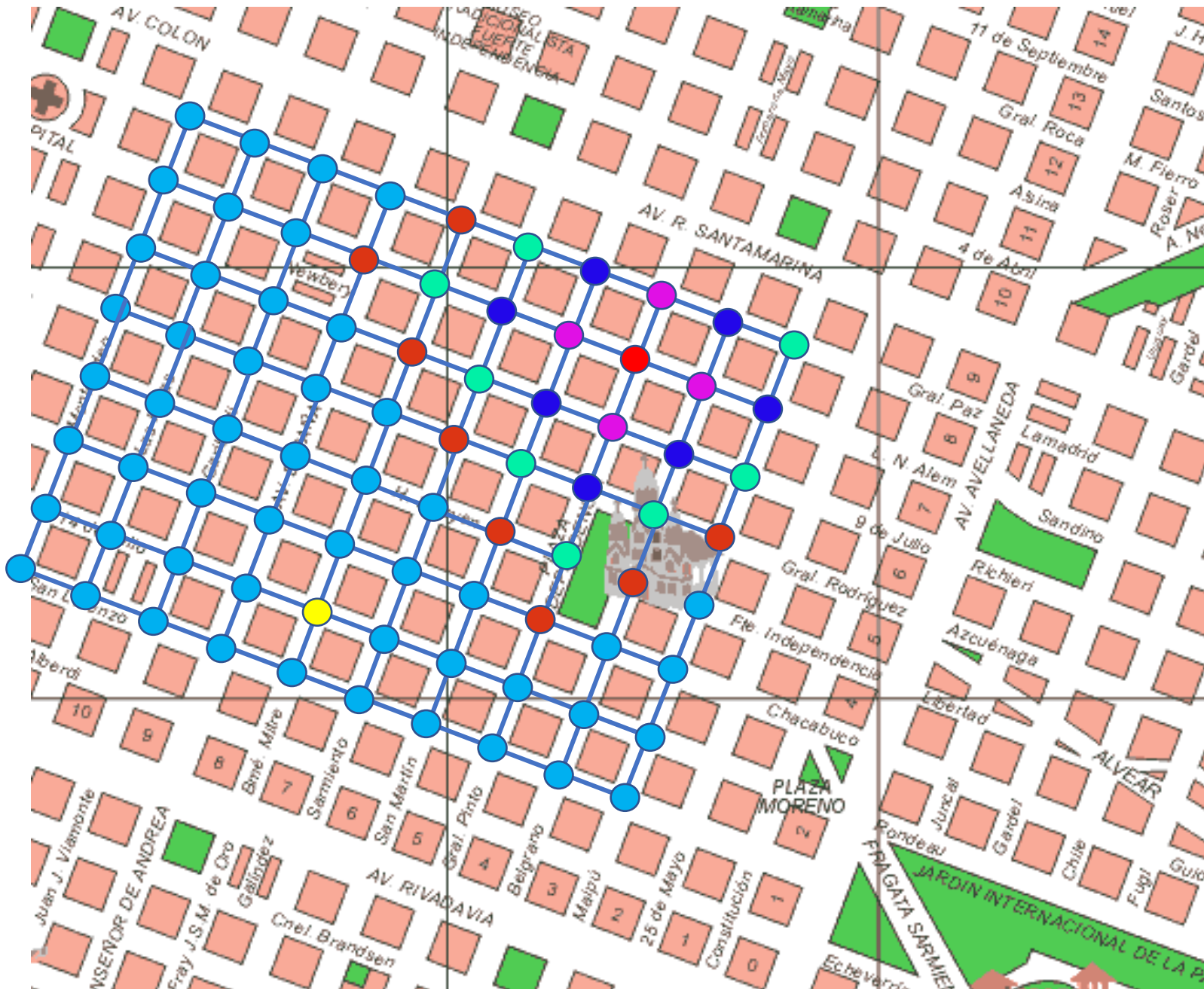
Cuál es la menor cantidad de cuadras a caminar para llegar desde rojo hasta amarillo?

Aplicamos BFS desde rojo hasta encontrar el amarillo.



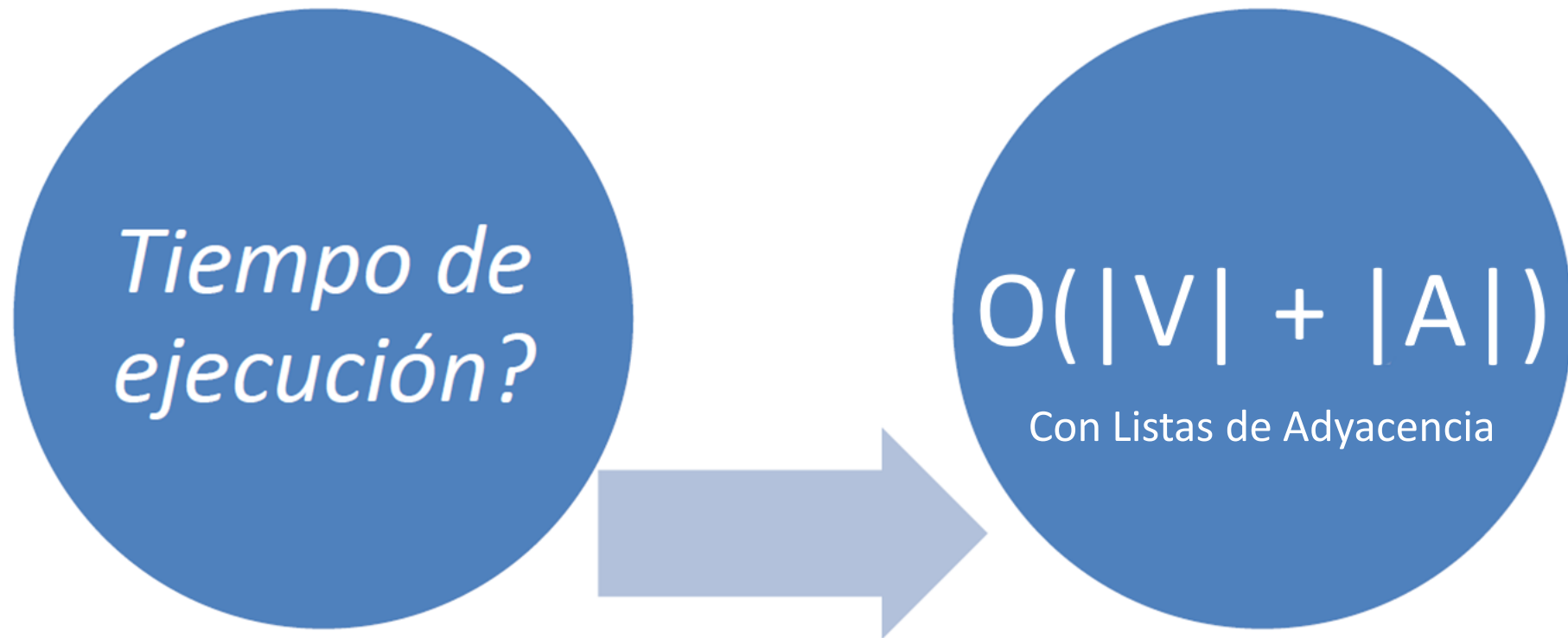




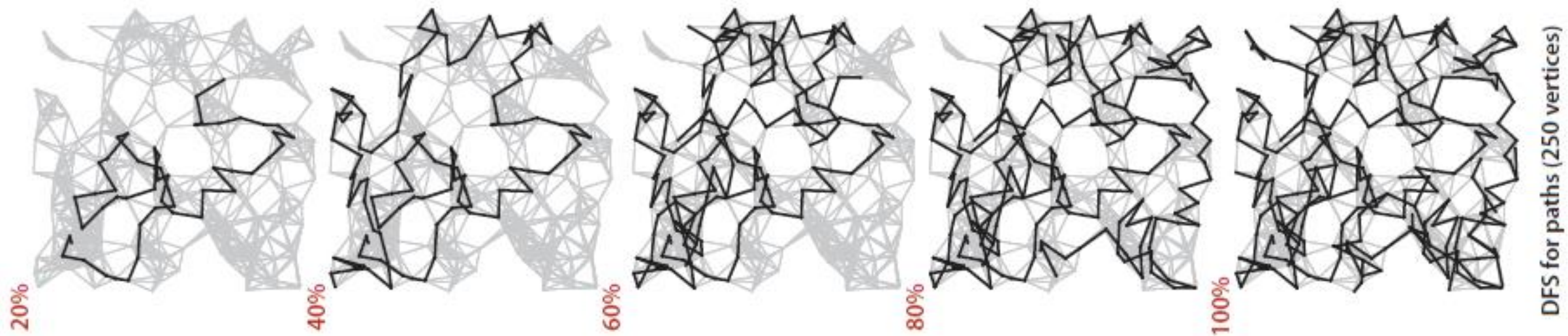


Encontramos al amarillo en el nivel 8 (distancia 8 desde el origen). O sea caminaremos como mínimo 8 cuadras.

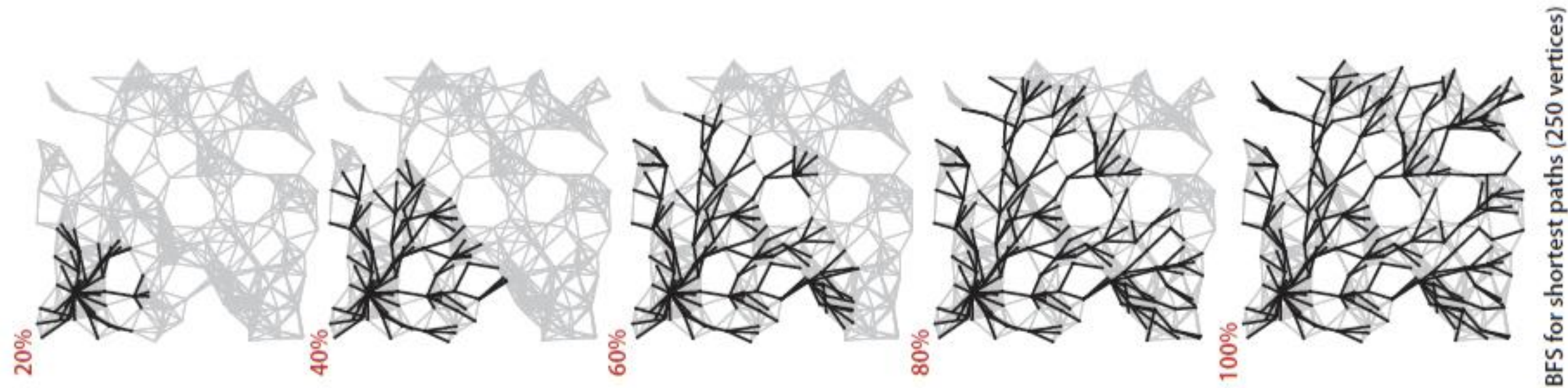
BFS: Complejidad Computacional



PASA UN VEZ POR CADA NODO, Y COMO MUCHO UNA VEZ POR CADA ARCO



DFS



BFS