

RECORRIDO EN PROFUNDIDAD

Lo vemos sobre grafos NO dirigidos. De forma sistematica y eficiente.



UNIVERSIDAD
COMPLUTENSE
MADRID

ALBERTO VERDEJO

Recorridos

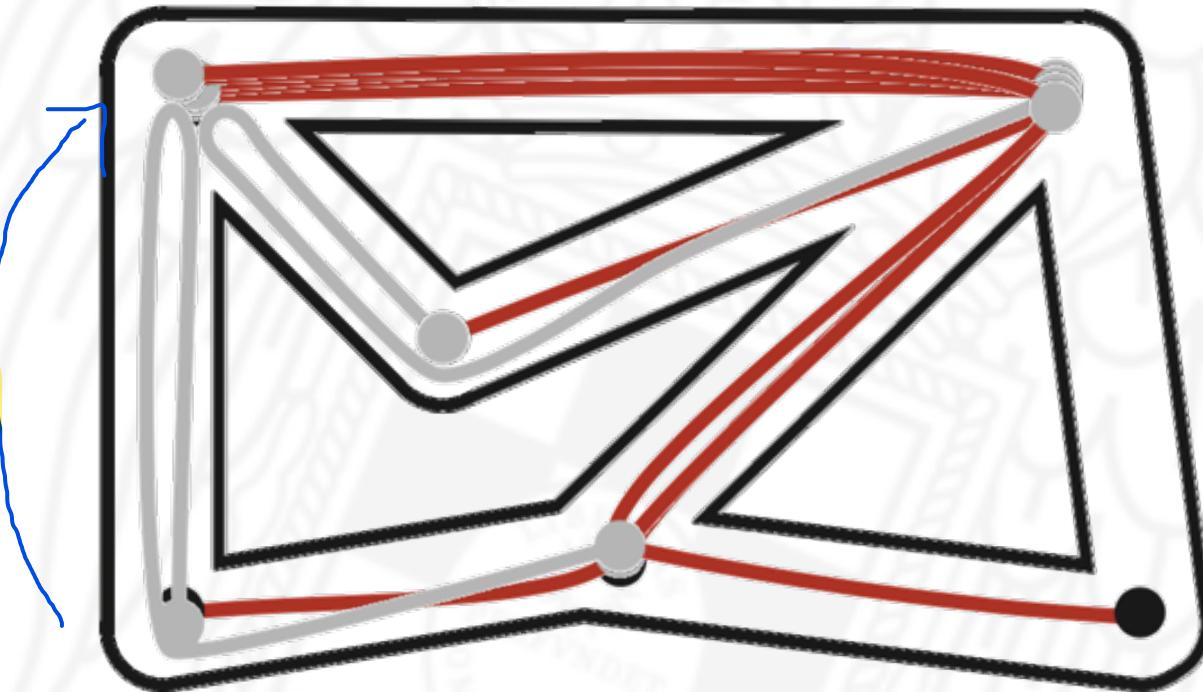
- ▶ Muchas propiedades de los grafos pueden conocerse explorando sistemáticamente sus vértices y aristas.
- ▶ Algunas propiedades pueden averiguarse simplemente analizando todas las aristas.
Saber el grado de un vértice o conocer si contiene un ciclo euleriano.
- ▶ Muchas otras propiedades están relacionadas con **caminos**, por lo que interesa explorar el grafo siguiendo aristas. Es lo que se conoce como **recorridos**.
Pueden ser recorridos en profundidad o en anchura como en el caso de los árboles.

Saber si dos vértices están conectados o si un grafo es conexo.

Recorrido en profundidad

DFS

- El recorrido o búsqueda en profundidad (en inglés, *depth-first search*) imita la resolución de un laberinto.

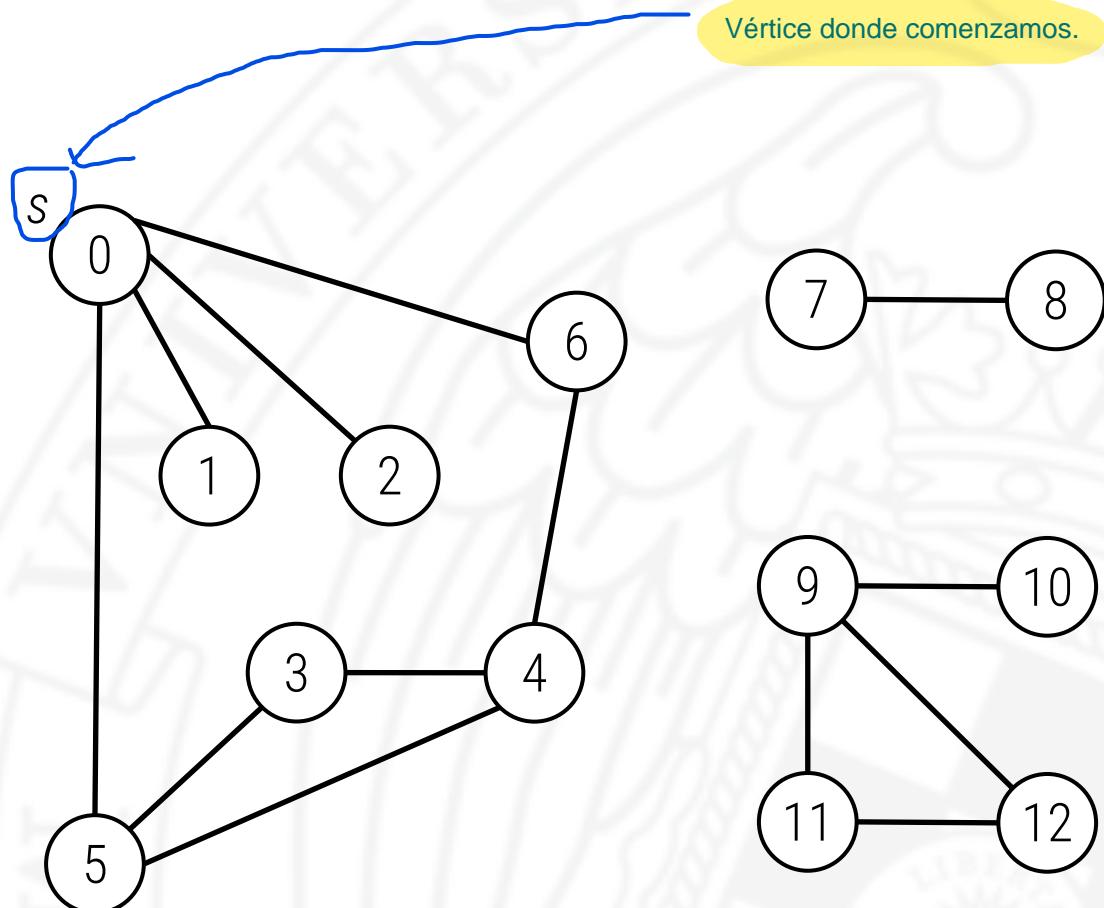


Si volvieramos ahí llegaríamos a un nodo YA visitado. Y nos conviene no volver a explorar los ya visitados

Recorrido en profundidad

- ▶ Para recorrer un grafo utilizamos un algoritmo recursivo que va visitando vértices.
- ▶ Visitar un vértice v consiste en:
 - ▶ marcarlo como visitado; ↗ *Para no volver a él y hacer un ciclo*
 - ▶ *hacer algo con él;* y ↗ *Será diferente dependiendo para qué algoritmo estamos haciendo el recorrido*
 - ▶ visitar (recursivamente) todos los vértices adyacentes a v aún no visitados.

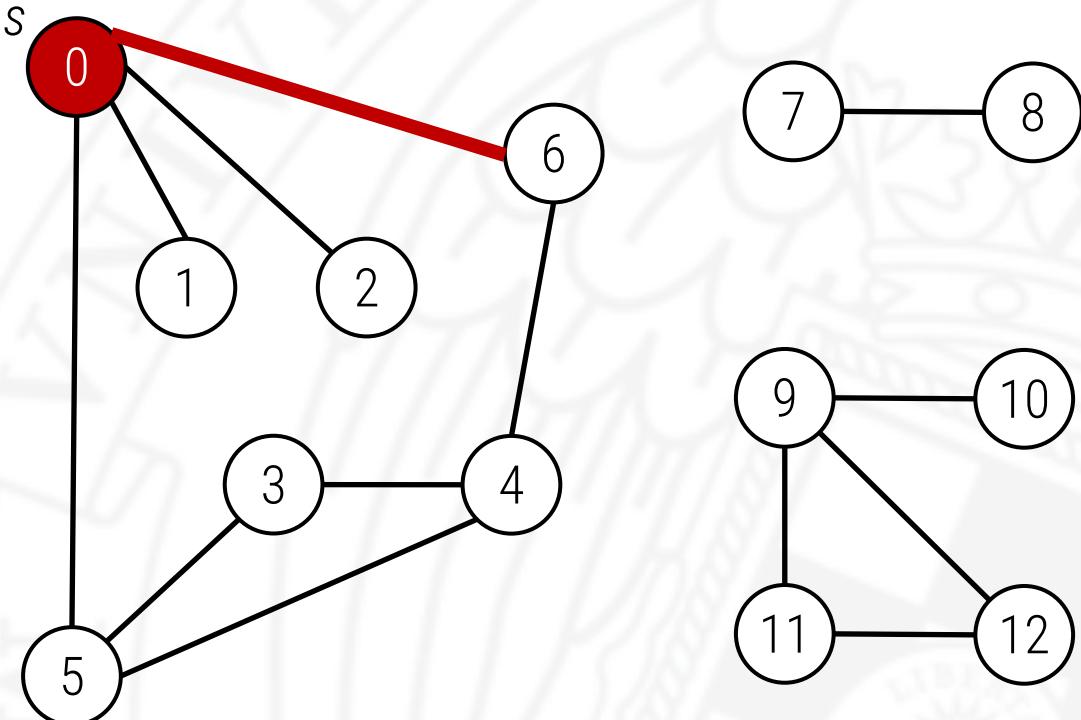
Recorrido en profundidad



Vector anterior para saber qué recorrido hacemos.

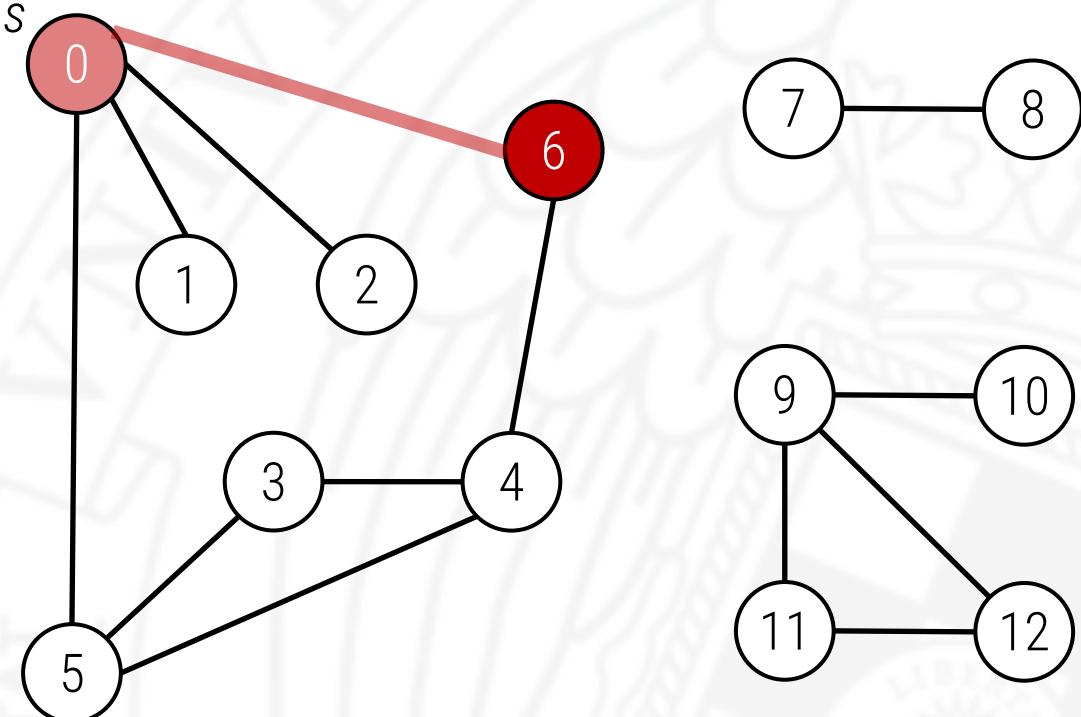
v	visitado	anterior
0	F	-
1	F	-
2	F	-
3	F	-
4	F	-
5	F	-
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Recorrido en profundidad



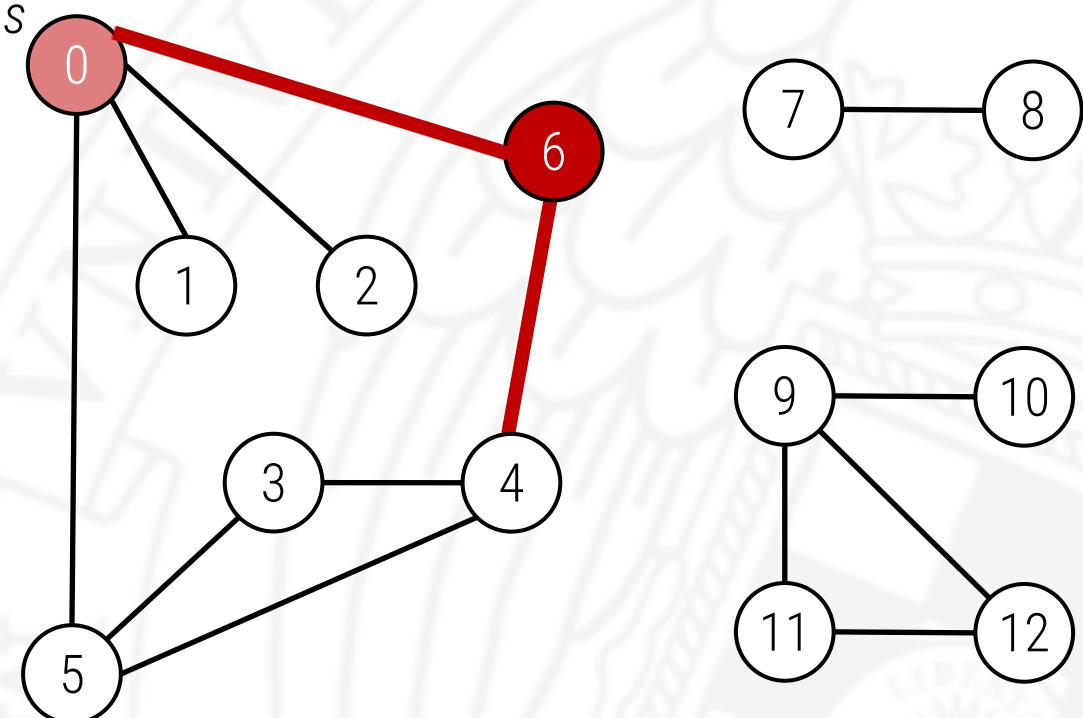
v	visitado	anterior
0	T	-
1	F	-
2	F	-
3	F	-
4	F	-
5	F	-
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Recorrido en profundidad



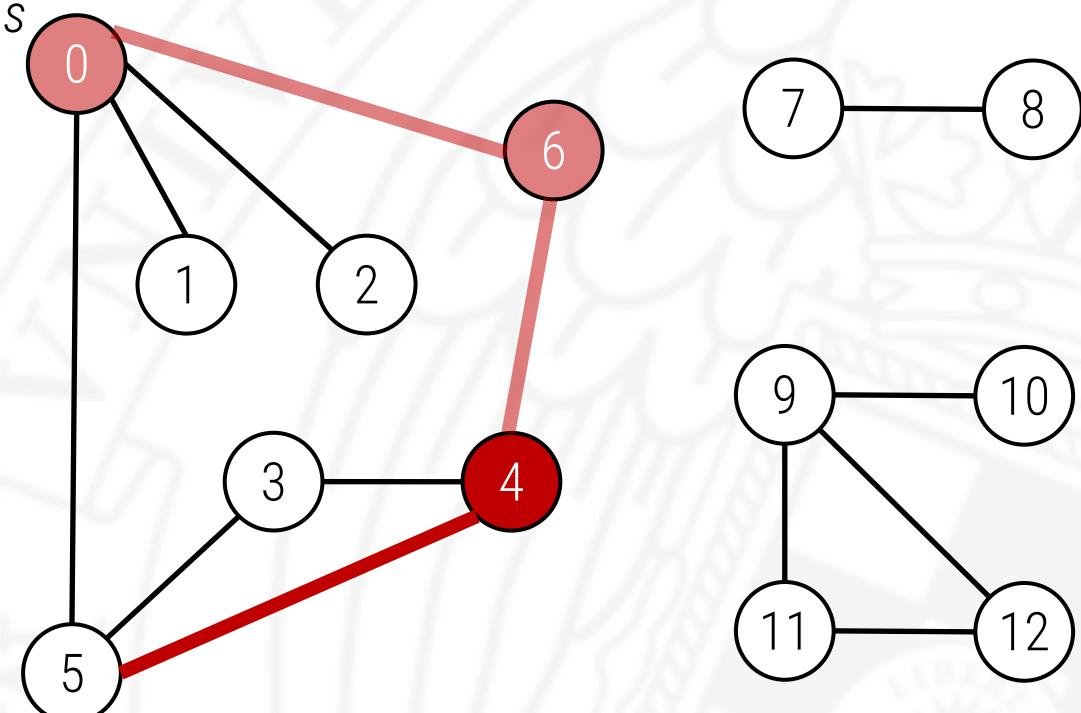
v	visitado	anterior
0	T	-
1	F	-
2	F	-
3	F	-
4	F	-
5	F	-
6	T	0
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Recorrido en profundidad



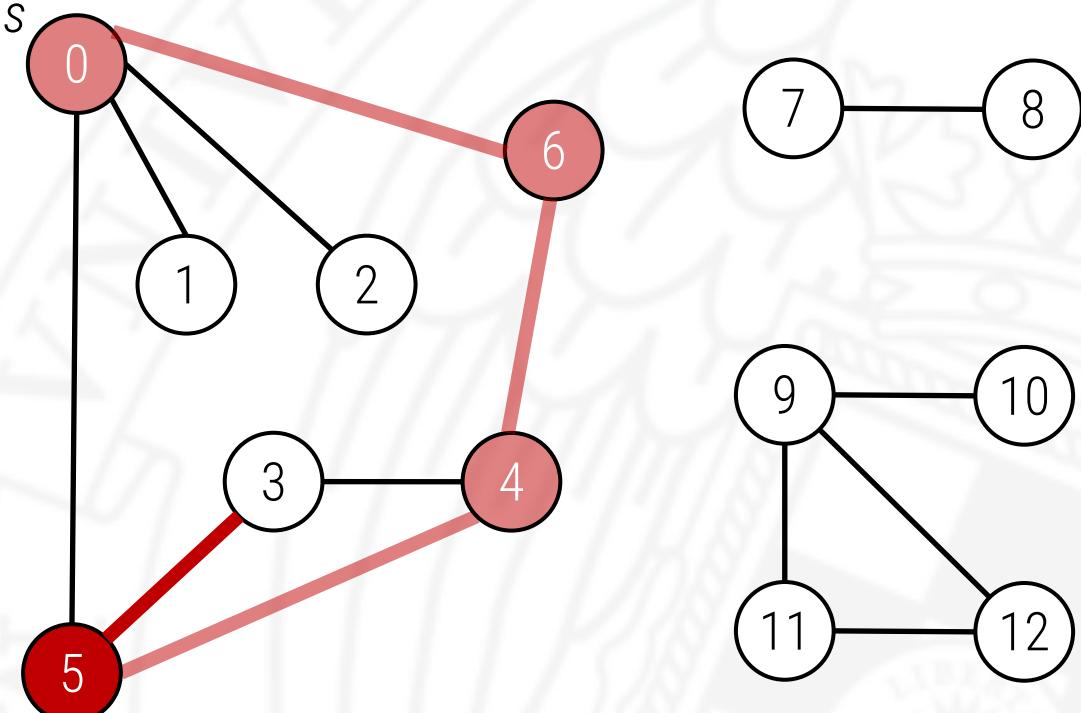
v	visitado	anterior
0	T	-
1	F	-
2	F	-
3	F	-
4	F	-
5	F	-
6	T	0
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Recorrido en profundidad



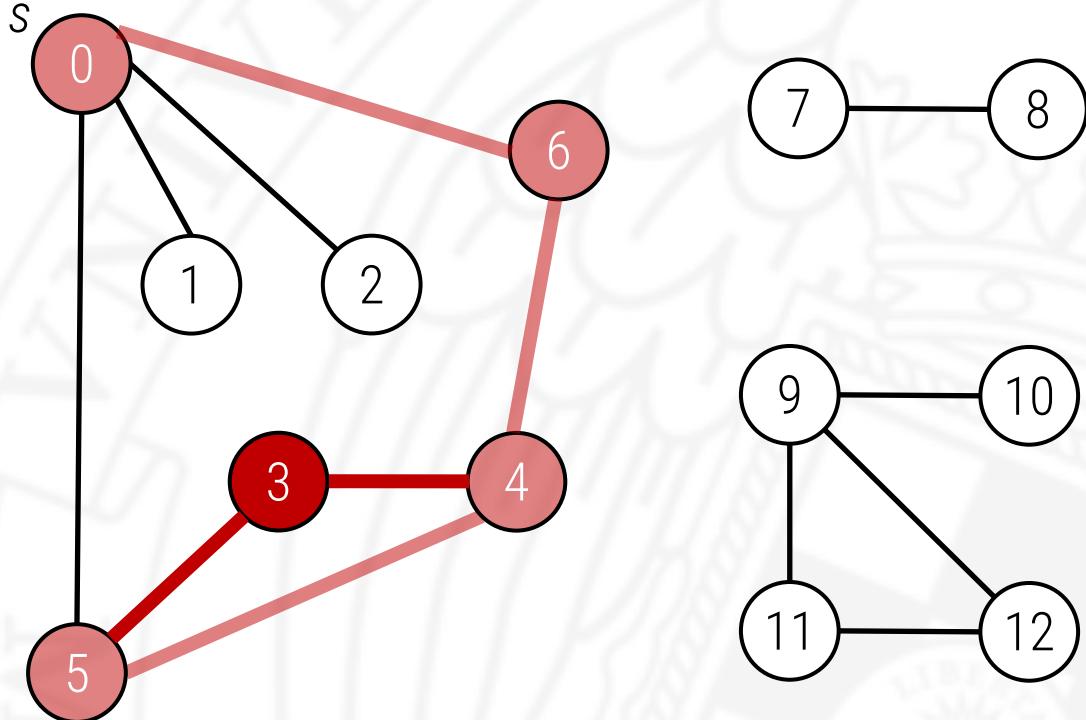
v	visitado	anterior
0	T	-
1	F	-
2	F	-
3	F	-
4	T	6
5	F	-
6	T	0
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Recorrido en profundidad



v	visitado	anterior
0	T	-
1	F	-
2	F	-
3	F	-
4	T	6
5	T	4
6	T	0
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

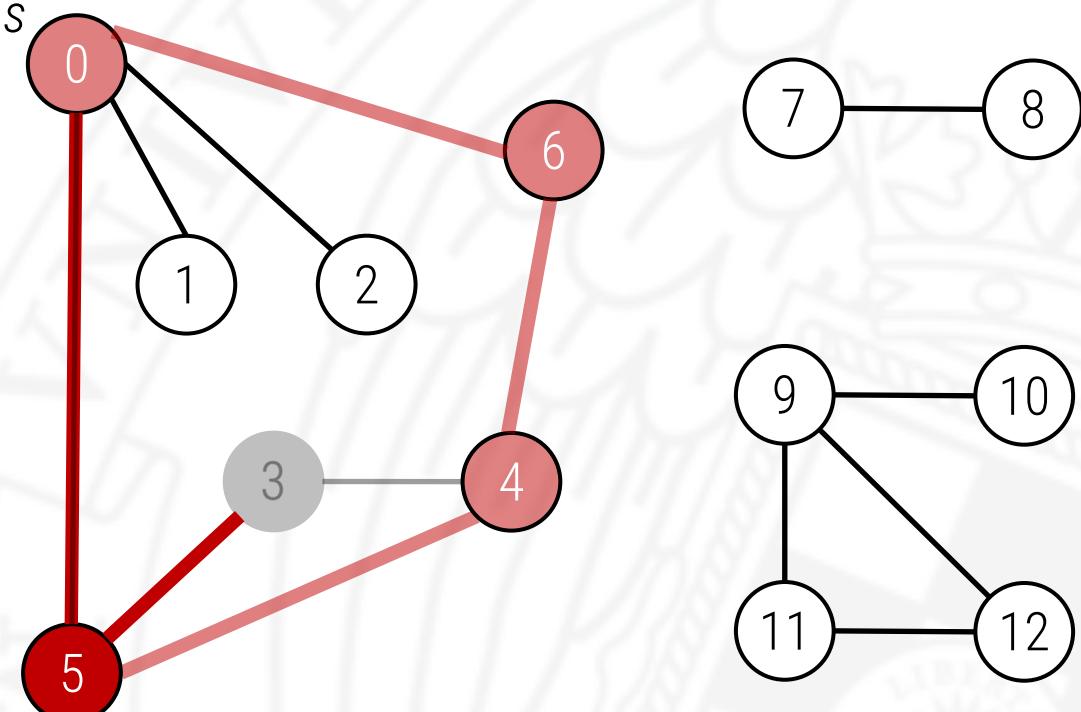
Recorrido en profundidad



EN ESTE CASO LLEGAMOS A QUE EL TODOS LOS VÉRTICES QUE CONECTAN CON EL 3 YA HAN SIDO VISITADOS, LA LLAMADA CON EL 3 TERMINA Y VUELVE A LA ANTERIOR LLAMADA RECURSIVA CON EL 5 QUE HABÍA QUEDADO PENDIENTE. CON EL 5 VA A OCURRIR LO MISMO, TODOS LOS VÉRTICES ADYACENTES AL 5 YA HAN SIDO VISITADOS, POR TANTO NOS OLVIDAMOS DEL 5 Y PASAMOS AL 4 QUE ES LA ANTERIOR.

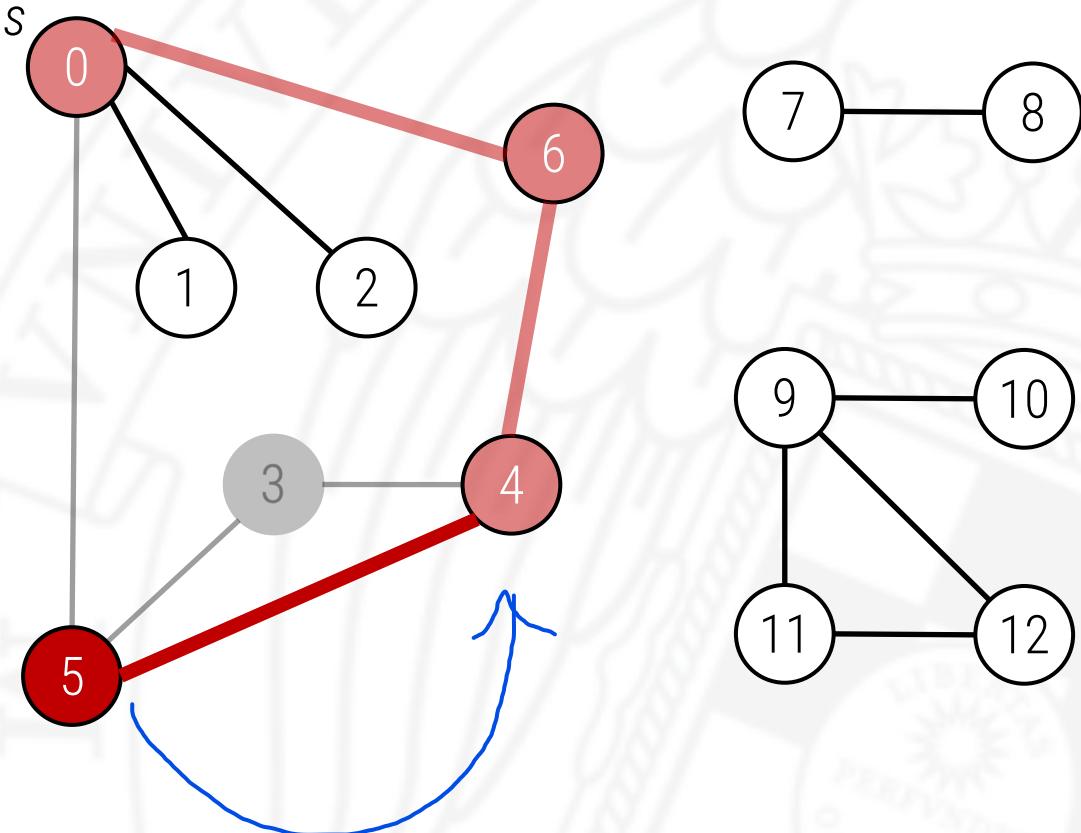
v	visitado	anterior
0	T	-
1	F	-
2	F	-
3	T	5
4	T	6
5	T	4
6	T	0
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Recorrido en profundidad



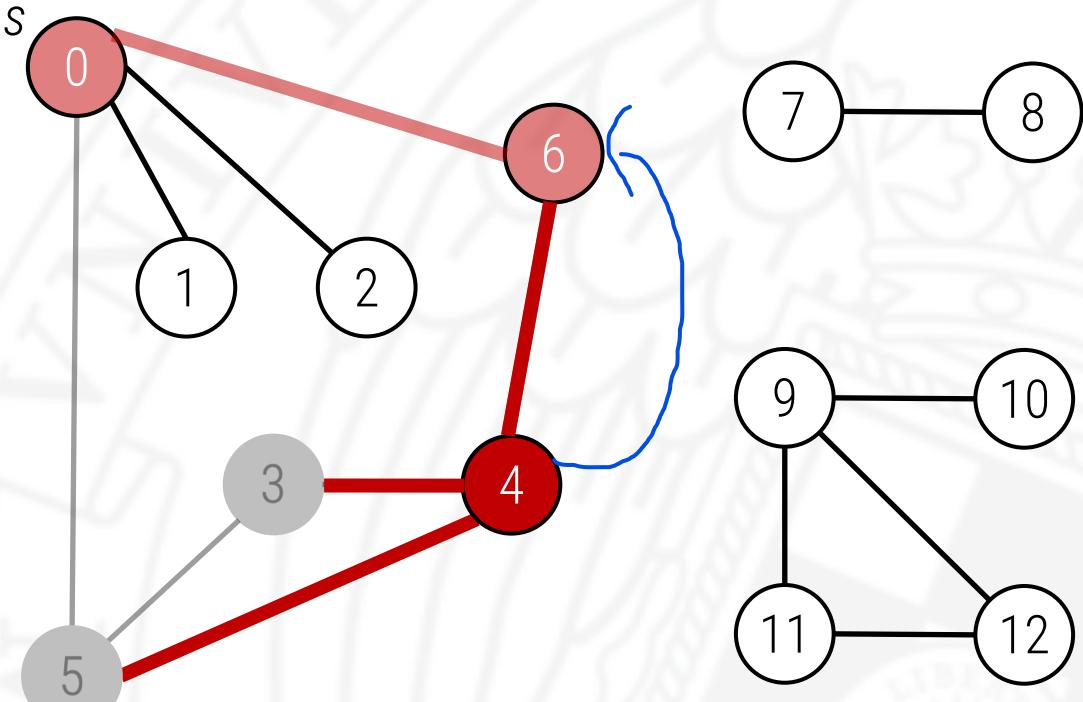
v	visitado	anterior
0	T	-
1	F	-
2	F	-
3	T	5
4	T	6
5	T	4
6	T	0
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Recorrido en profundidad



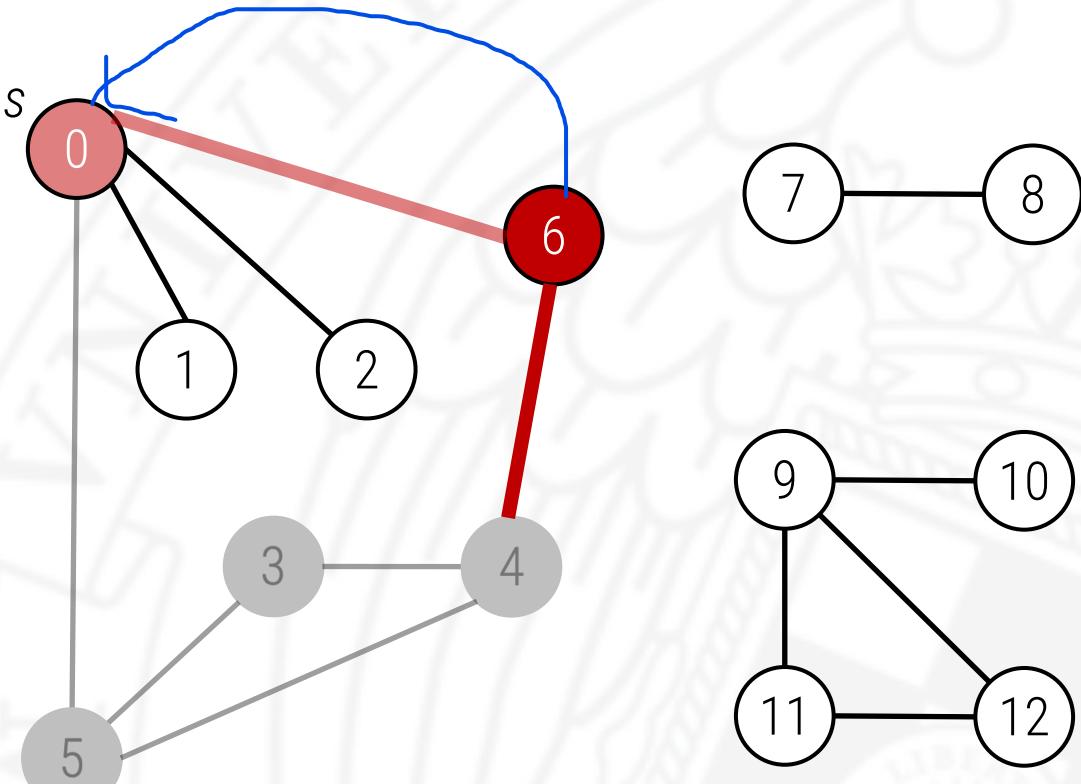
v	visitado	anterior
0	T	-
1	F	-
2	F	-
3	T	5
4	T	6
5	T	4
6	T	0
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Recorrido en profundidad



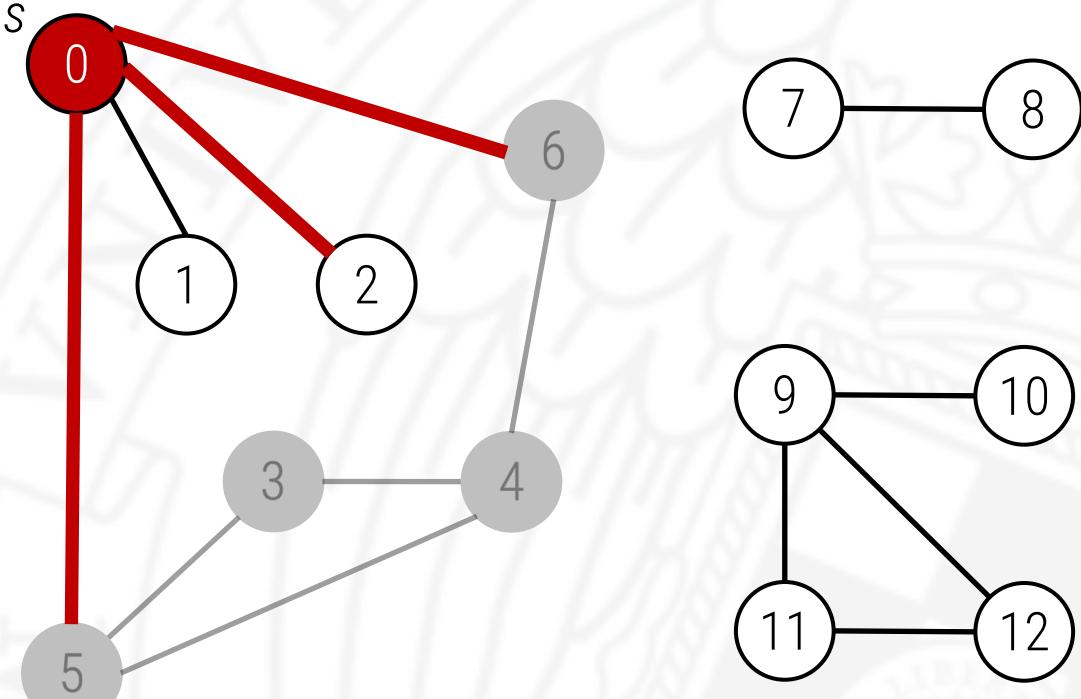
v	visitado	anterior
0	T	-
1	F	-
2	F	-
3	T	5
4	T	6
5	T	4
6	T	0
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Recorrido en profundidad



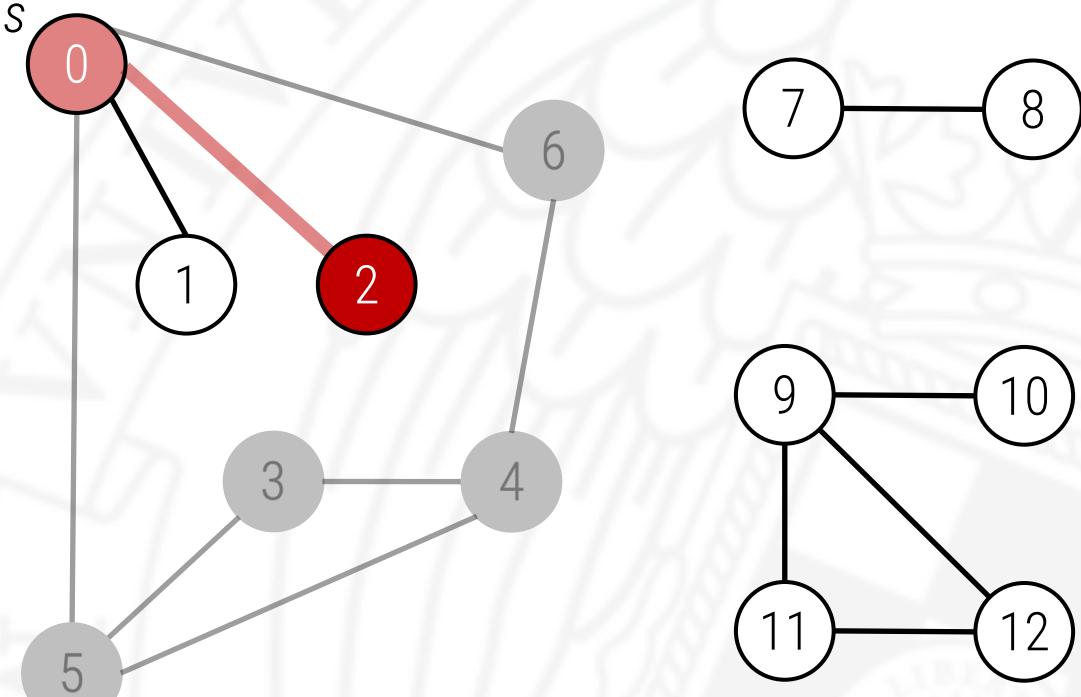
v	visitado	anterior
0	T	-
1	F	-
2	F	-
3	T	5
4	T	6
5	T	4
6	T	0
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Recorrido en profundidad



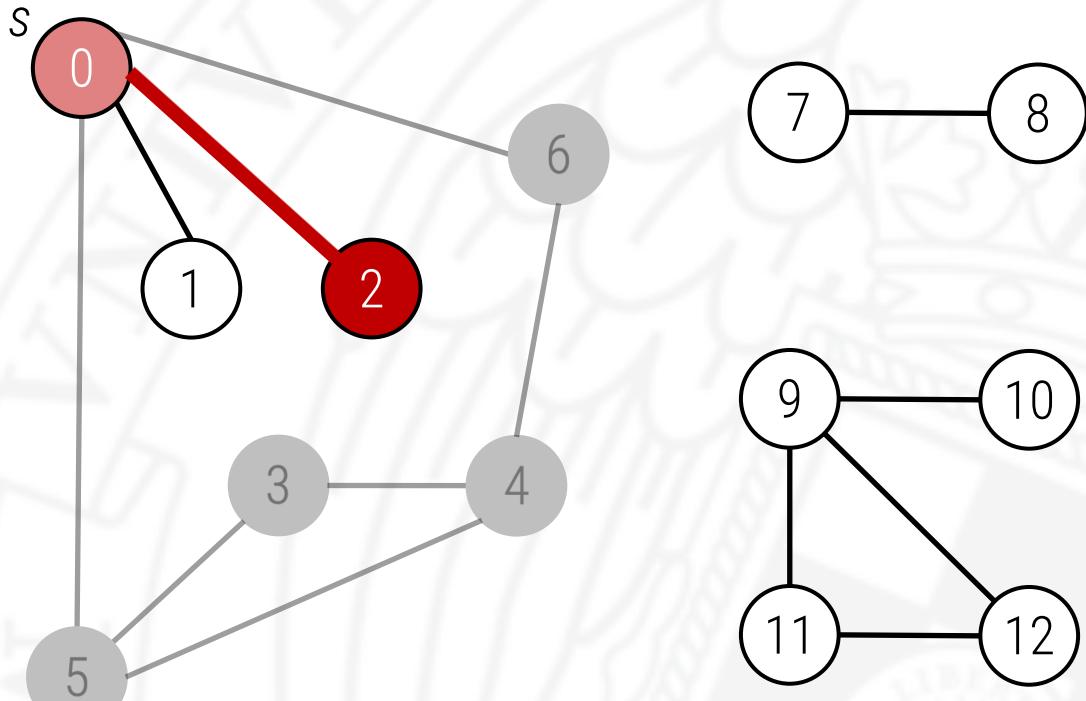
v	visitado	anterior
0	T	-
1	F	-
2	F	-
3	T	5
4	T	6
5	T	4
6	T	0
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Recorrido en profundidad



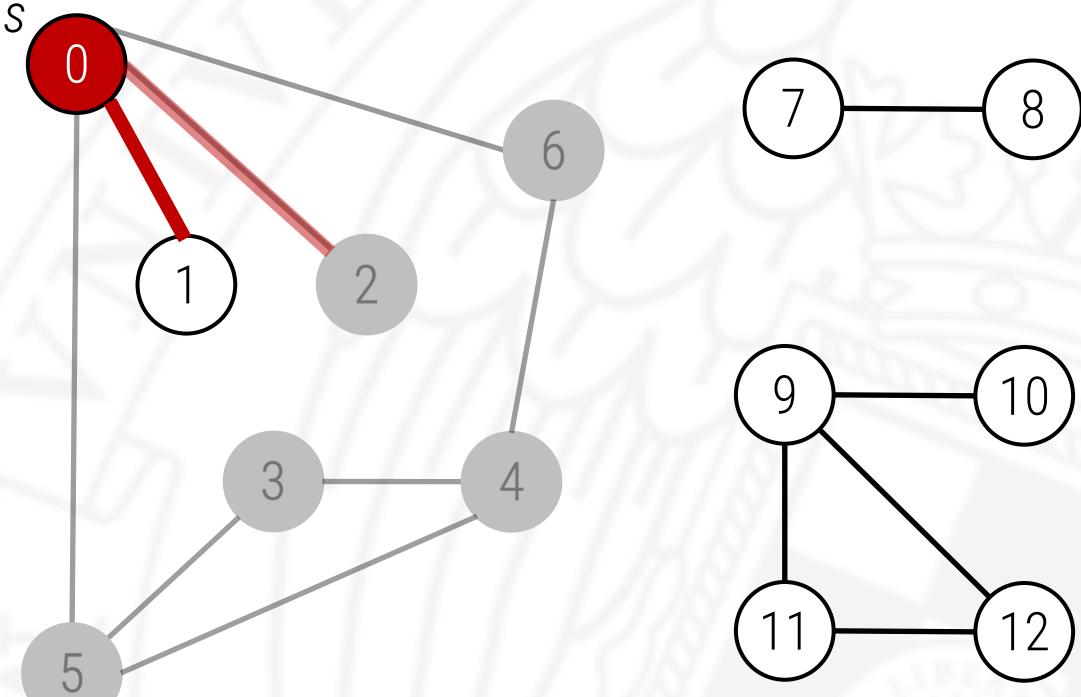
v	visitado	anterior
0	T	-
1	F	-
2	T	0
3	T	5
4	T	6
5	T	4
6	T	0
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Recorrido en profundidad



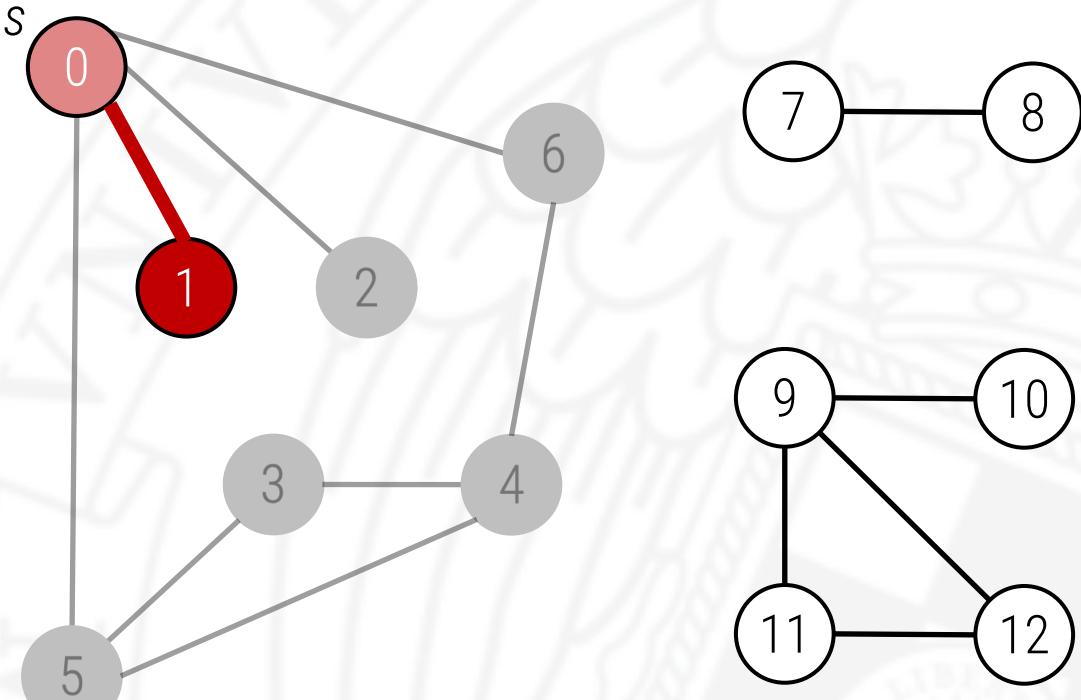
v	visitado	anterior
0	T	-
1	F	-
2	T	0
3	T	5
4	T	6
5	T	4
6	T	0
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Recorrido en profundidad



v	visitado	anterior
0	T	-
1	F	-
2	T	0
3	T	5
4	T	6
5	T	4
6	T	0
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

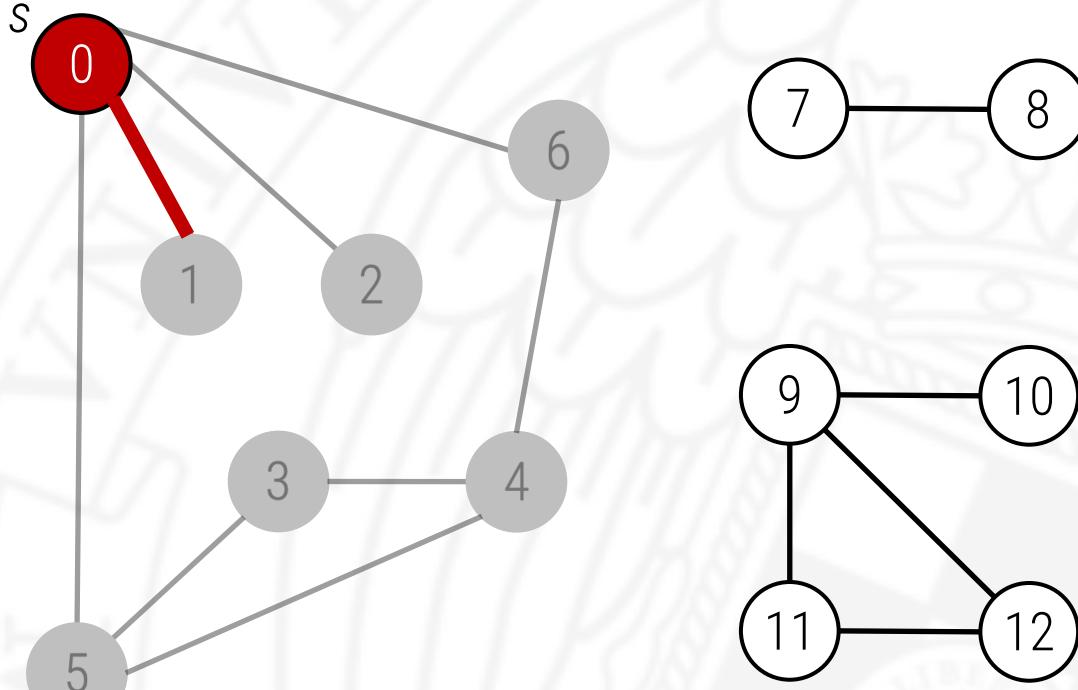
Recorrido en profundidad



v	visitado	anterior
0	T	-
1	T	0
2	T	0
3	T	5
4	T	6
5	T	4
6	T	0
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Recorrido en profundidad

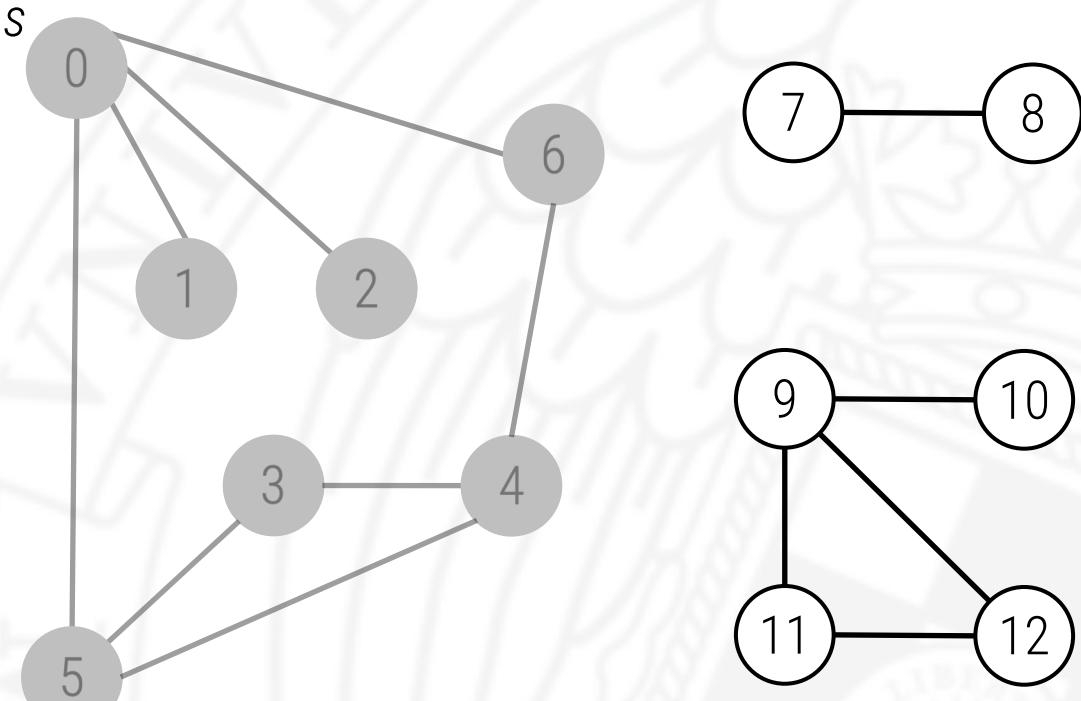
Que los vértices marcados sean consecutivos NO TIENE POR QUÉ OCURRIR es una mera casualidad.



v	visitado	anterior
0	T	-
1	T	0
2	T	0
3	T	5
4	T	6
5	T	4
6	T	0
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

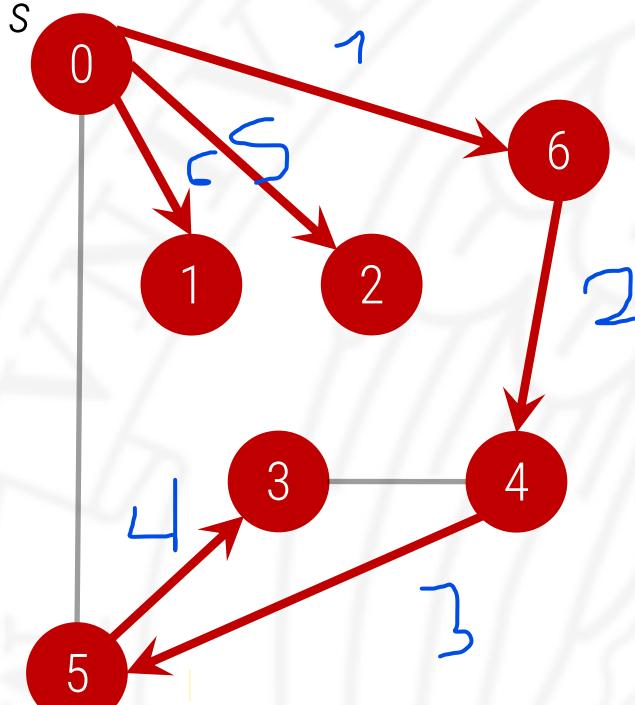
El resto de vértices NO ha sido marcado, lo que indica que el grafo NO es conexo.

Recorrido en profundidad



v	visitado	anterior
0	T	-
1	T	0
2	T	0
3	T	5
4	T	6
5	T	4
6	T	0
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Recorrido en profundidad



Si tuviéramos que recorrer el resto de grafos, tenemos que ver cuales es el primero NO visitado (F). En este caso es el 7.

v	visitado	anterior
0	T	-
1	T	0
2	T	0
3	T	5
4	T	6
5	T	4
6	T	0
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Caminos desde un origen

```
class CaminosDFS {  
private:  
    std::vector<bool> visit; // visit[v] = ¿hay camino de s a v?  
    std::vector<int> ant;    // ant[v] = último vértice antes de llegar a v  
    int s;                  // vértice origen  
  
    void dfs(Grafo const& G, int v) {  
        visit[v] = true;  
        for (int w : G.ady(v)) {  
            if (!visit[w]) {  
                ant[w] = v;  
                dfs(G, w);  
            }  
        }  
    }  
}
```

Recibe el grafo y el vértice en el que estamos (initialmente s), para hacer el recorrido en profundidad del grafo.

Se marca como visitado y se recorren los adyacentes.

Comprobamos si está o no marcado/visitado.

Solo puede ser parámetro como máximo una vez.

Caminos desde un origen

```
public:  
    CaminosDFS(Grafo const& g, int s) : visit(g.V(), false),  
        ant(g.V()), s(s) {  
        dfs(g, s);  
    }  
  
    // ¿hay camino del origen a v?  
    bool hayCamino(int v) const {  
        return visit[v];  
    }
```

Inicializamos todos a false.

vector de anteriores de tamaño v, y apuntamos que s es el origen.

Comprobar si v ha sido visitado.

Caminos desde un origen

Los vamos a querer recorrer de principio a fin y al ir añadiéndolo vamos a querer hacerlo por delante.

```
using Camino = std::deque<int>; // para representar caminos
```

```
// devuelve un camino desde el origen a v (debe existir)
```

```
Camino camino(int v) const {
    if (!hayCamino(v))
        throw std::domain_error("No existe camino");
```

```
Camino cam;
```

```
// recuperamos el camino retrocediendo
```

```
for (int x = v; x != s; x = ant[x])
```

```
    cam.push_front(x);
```

```
cam.push_front(s);
```

```
return cam;
```

```
}
```

```
};
```

va como retrocediendo devolviendo cuales son los vértices por los que ha ido pasando.

Si no hay camino al vértice v lanzamos error.

Pasamos al anterior mientras NO lleguemos al origen.

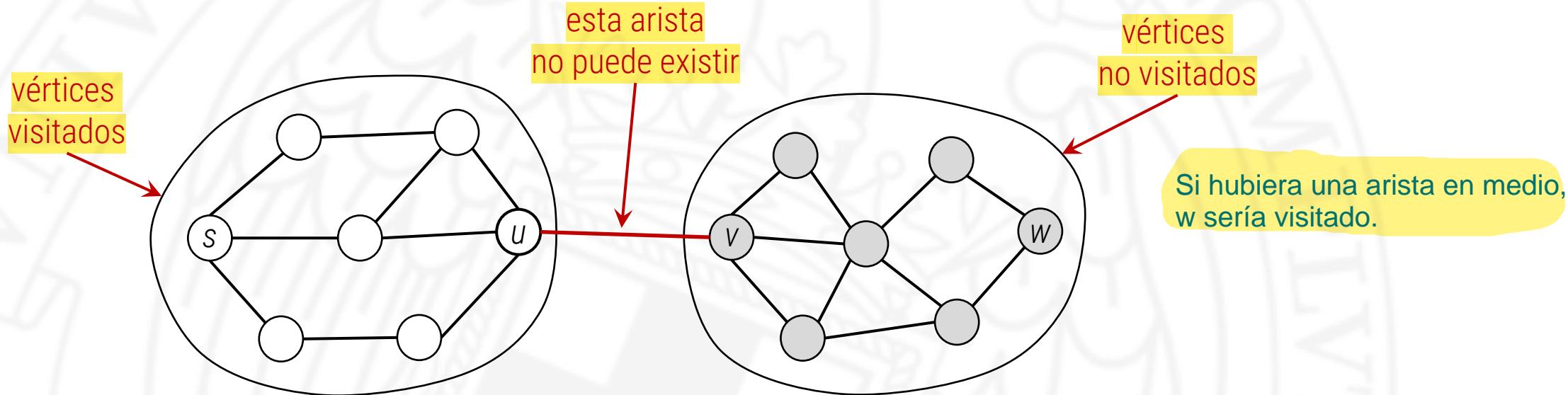
Al que queremos llegar

Recorrido en profundidad: corrección y coste

- ▶ El recorrido en profundidad visita todos los vértices alcanzables desde el origen s en un tiempo proporcional a la suma de sus grados (más la inicialización de los vectores). Que es lineal al número de vértices la inicialización del vector.
- ▶ Si w es visitado es porque está conectado a s (solo pasamos de un vértice a otro atravesando aristas).

Recorrido en profundidad: corrección y coste

- ▶ Si w está conectado a s , entonces termina siendo visitado.



- ▶ El coste está en $O(V + A)$ porque cada vértice se visita una única vez.

Lineal en la suma del número de vértices y el número de aristas.