

Grafos

Recorridos

Recorridos de grafos

- Recorrido en profundidad: DFS (Depth First Search)
 - Repaso del algoritmo DFS
 - Ejemplo: ¿cómo funciona la recursión?
- Recorrido en amplitud: BFS (Breadth First Search)

Recorrido en profundidad de Grafos

Depth First Search (DFS)

El **DFS** es **equivalente** al recorrido **preorden de un árbol**.

El **DFS** explora las aristas del grafo de manera que **se visitan los vértices adyacentes al recién visitado**, con lo que se consigue **profundizar** en las ramas del grafo.

Es un recorrido recursivo.

Dado $G = (V, A)$

1. Marcar todos los vértices como no visitados.
2. Elegir vértice **u** (no visitado) como **punto de partida**.
3. Marcar **u** como visitado.
4. Para todo **v** adyacente a **u**, $(u,v) \in A$, si **v** no ha sido visitado, repetir recursivamente (3) y (4) para **v**.

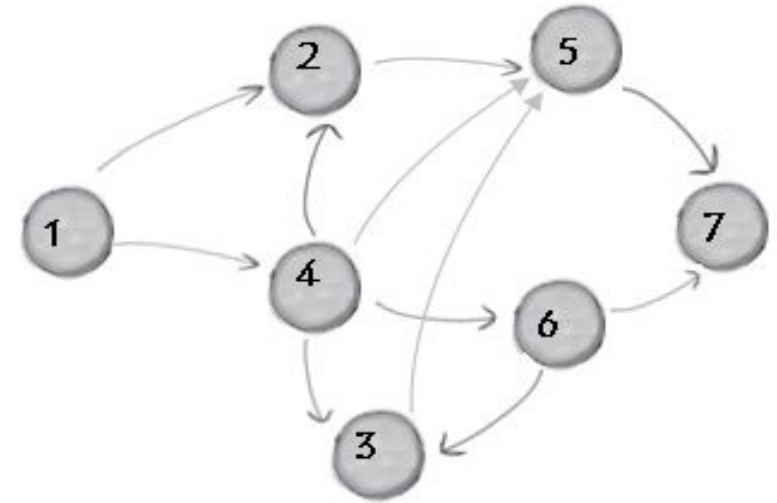
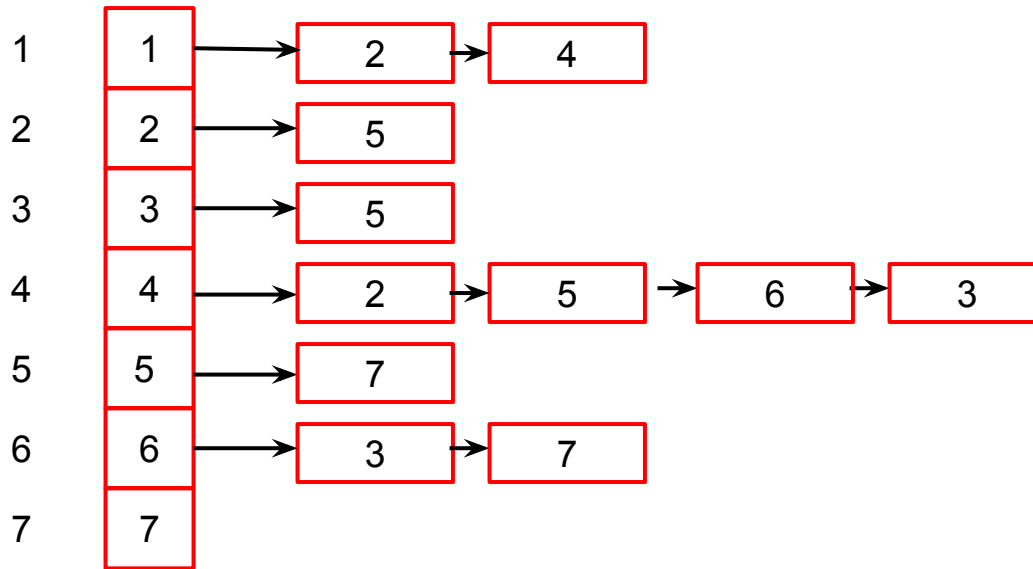
¿Cuándo finaliza el recorrido?

Finaliza cuando se **visitaron todos los nodos alcanzables** desde **u**.

Si desde **u** no fueran alcanzables todos los nodos del grafo: volver a (2), **elegir un nuevo vértice de partida u** no visitado, y repetir el proceso hasta que se hayan **recorrido todos los vértices**.

Recorrido en profundidad de Grafos

Depth First Search (DFS)



Es necesario **registrar los nodos visitados** para evitar recorrerlos varias veces.

El **recorrido no es único**: depende del vértice inicial y del orden de visita de los vértices adyacentes.

```

public class Recorridos<T> {

    public void dfs(Grafo<T> grafo){
        boolean[] marca = new boolean[grafo.listaDeVertices().tamanio()];
        for(int i=0; i<grafo.listaDeVertices().tamanio();i++){
            if (!marca[i]) // si no está marcado
                this.dfs(i, grafo, marca);
        }
    }

    private void dfs(int i,Grafo<T> grafo, boolean[] marca){
        marca[i] = true;
        Arista <T> arista=null;
        int j=0;
        Vertice<T> v = grafo.listaDeVertices().elemento(i);
        System.out.println(v);
        ListaGenerica<Arista<T>> ady = grafo.listaDeAdyacentes(v);
        ady.comenzar();
        while(!ady.fin()){
            arista=ady.proximo();
            j = arista.getVerticeDestino().getPosicion();
            if(!marca[j])
                this.dfs(j, grafo, marca);
        }
    }
}

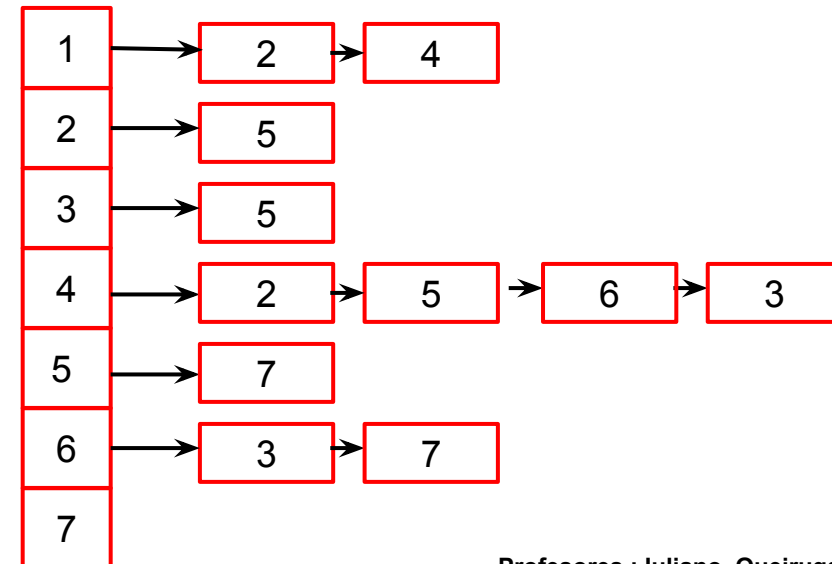
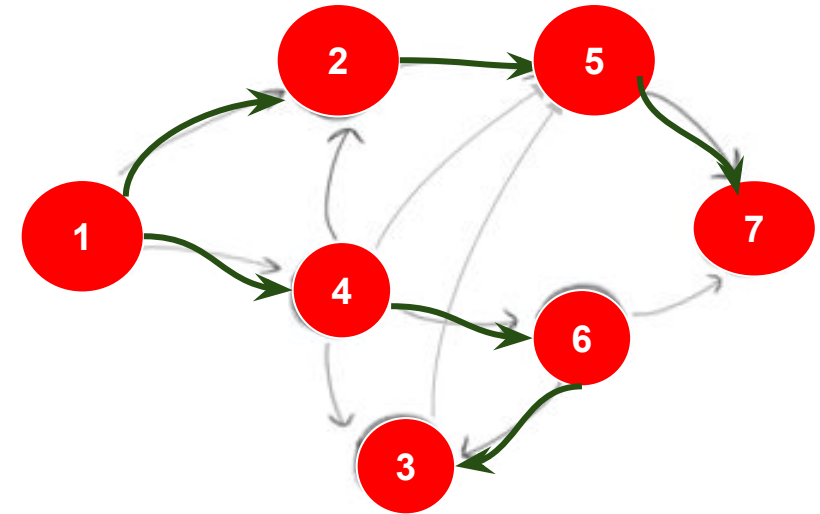
```

Veamos cómo funciona el DFS en el ejemplo

Vértice de partida: 1

Vértice - Lista de Adyacentes	Marca							
	1	2	3	4	5	6	7	
	F	F	F	F	F	F	F	
1 2 4	T	F	F	F	F	F	F	
2 5	T	T	F	F	F	F	F	
5 7	T	T	F	F	T	F	F	
7	T	T	F	F	T	F	T	
4 2 5 6 3	T	T	F	T	T	F	T	
6 3 7	T	T	F	T	T	T	T	
3 5	T	T	T	T	T	T	T	

Recorrido DFS: 1 2 5 7 4 6 3



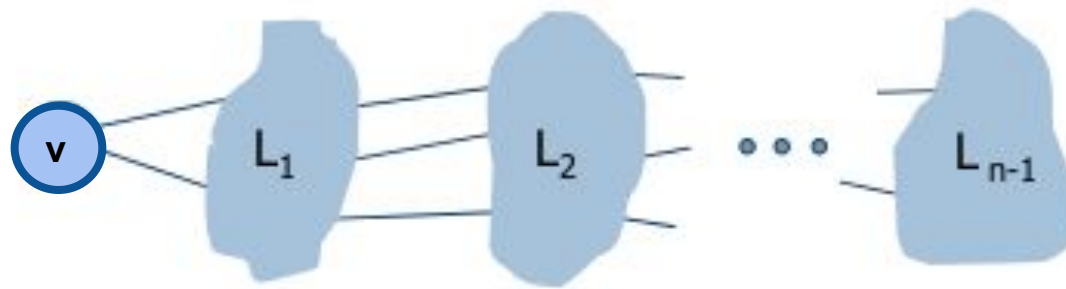
Recorrido en amplitud de Grafos

Breadth First Search (BFS)

Este recorrido es **equivalente al recorrido por niveles de un árbol**.

La estrategia es la siguiente:

- Partir de algún **vértice v** , **visitar v** , después **visitar cada uno de los vértices adyacentes a v** .
- **Repetir** el proceso para **cada nodo adyacente a v** , siguiendo el orden en que fueron visitados.
- **Si desde v no fueran alcanzables todos los nodos del grafo**: elegir un **nuevo vértice de partida no visitado**, y repetir el proceso hasta que se hayan **recorrido todos los vértices**.



Exploración desde v por niveles:

$L_0 = \{v\}$

$L_1 =$ Nodos adyacentes a L_0 .

$L_2 =$ Nodos adyacentes a los vértices de L_1 que no están en L_0 ni L_1 .

$L_{i+1} =$ Nodos adyacentes a los vértices de L_i que no ningún nivel anterior.

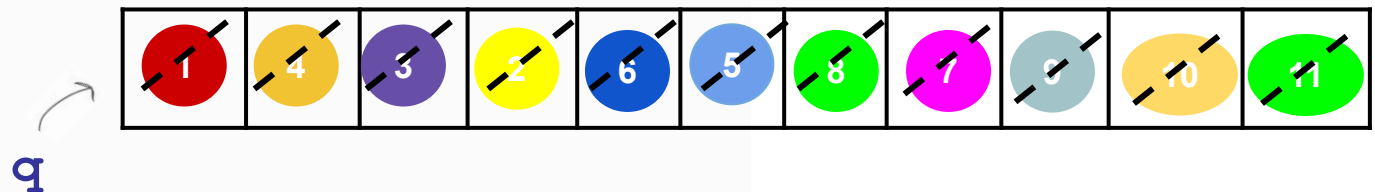
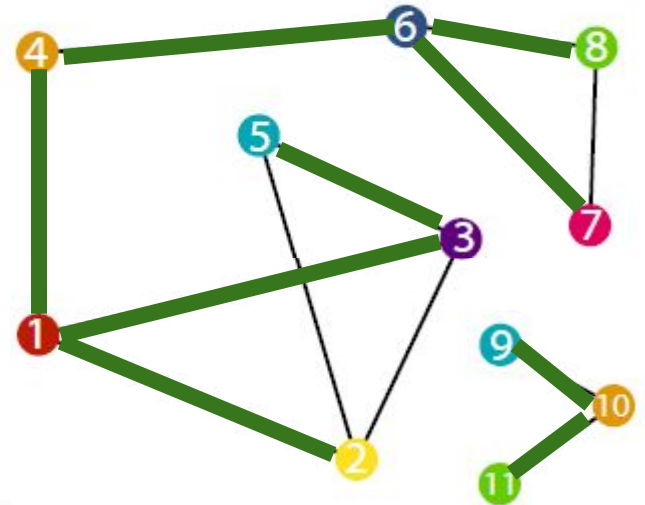
```
public class Recorridos {  
    public void bfs(Grafo<T> grafo) {  
        boolean[] marca = new boolean[grafo.listaDeVertices().tamano()];  
        for (int i = 0; i < marca.length; i++) {  
            if (!marca[i])  
                this.bfs(i+1, grafo, marca); //las listas empiezan en la pos 1  
        }  
    }  
  
    private void bfs (int i, Grafo<T> grafo, boolean[] marca) {  
        //siguiente diapo  
    }  
}
```

```

public class Recorridos {

    private void bfs(int i, Grafo<T> grafo, boolean[] marca) {
        ListaGenerica<Arista<T>> ady = null;
        ColaGenerica<Vertice<T>> q = new ColaGenerica<Vertice<T>>();
        q.encolar(grafo.listaDeVertices().elemento(i));
        marca[i] = true;
        while (!q.esVacia()) {
            Vertice<T> v = q.desencolar();
            System.out.println(v);
            ady = grafo.listaDeAdyacentes(v);
            ady.comenzar();
            while (!ady.fin()) {
                Arista<T> arista = ady.proximo();
                int j = arista.getDestino().posicion();
                if (!marca[j]) {
                    Vertice<T> w = arista.getDestino();
                    marca[j] = true;
                    q.encolar(w);
                }
            }
        }
    }
}

```

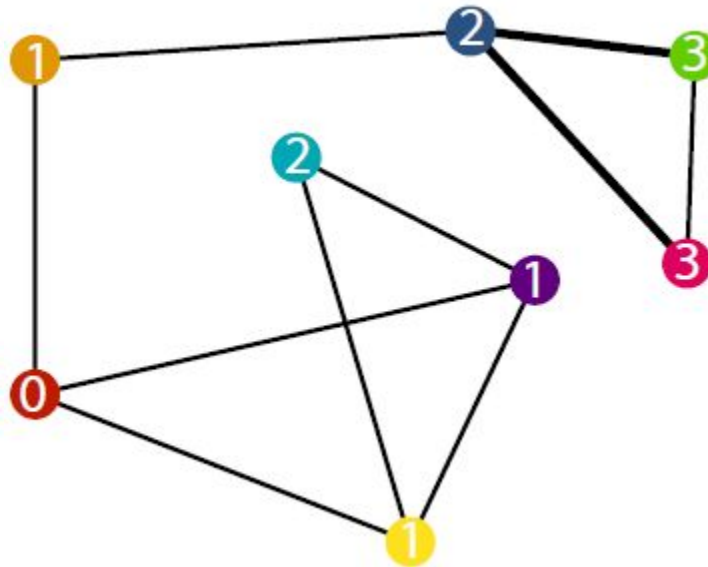


Ejemplo: Virus de computadoras

Un poderoso e inteligente **virus de computadora** infecta cualquier computadora en **1 minuto**, logrando infectar toda la red de una empresa con cientos de computadoras.

Dado un **grafo que representa las conexiones entre las computadoras de la empresa**, y una **computadora infectada**, escriba un programa en Java que permita **determinar el tiempo que demora el virus en infectar el resto de las computadoras**.

Todas las computadoras pueden ser infectadas, no todas las computadoras tienen conexión directa entre sí, y un mismo virus puede infectar un grupo de computadoras al mismo tiempo sin importar la cantidad.



```

public class BFSVirus {
    public int calcularTiempoInfeccion(Grafo<String> g, Vertice<String> inicial) {
        int n = g.listaDeVertices().tamano();
        ColaGenerica<Vertice<String>> cola = new ColaGenerica<Vertice<String>>();
        int distancias[] = new int[n+1];    //no se usa la posicion 0
        int maxDist = 0; int nuevaDist = 0;
        for (int i = 0; i<=n; ++i) {
            distancias[i] = Integer.MAX_VALUE;
        }
        distancias[inicial.posicion()] = 0;
        cola.encolar(inicial);
        while (!cola.esVacia()) {
            Vertice<String> v = cola.desencolar();
            nuevaDist = distancias[v.posicion()] + 1;
            ListaGenerica<Arista<String>> adyacentes = v.obtenerAdyacentes();
            adyacentes.comenzar();
            while (!adyacentes.fin()) {
                Arista<String> a = adyacentes.proximo();
                Vertice<String> w = a.getDestino();
                int pos = w.posicion();
                if (distancias[pos]== Integer.MAX_VALUE) {
                    distancias[pos] = nuevaDist;
                    if (nuevaDist > maxDist)
                        maxDist = nuevaDist;
                    cola.encolar(w);
                }
            }
        }
        return maxDist;
    }
}

```

