Grafos

Mario Macías Lloret

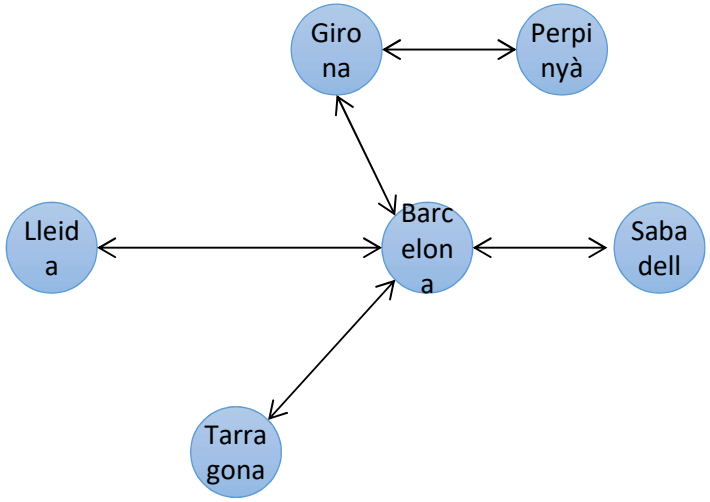
Programació Orientada a Objectes

Universitat Politècnica de Catalunya

Concepto de grafo

- Un grafo es una estructura de datos
  - Formada por **Nodos** (o **vértices**)
  - Nodos interconectados por **aristas** (o **arcos**)
    - Dirección: sentido en el que se puede navegar de un nodo a otro
    - Peso: "coste" de recorrer esa arista
    - Etiqueta: identificador de esa etiqueta.
- Representa información de relaciones diversos elementos de un sistema
  - Circuitos eléctricos
  - Jerarquía en una empresa
  - Redes sociales
  - Conexiones de vuelos
  - etc...

Ejemplo: mapa de autopistas

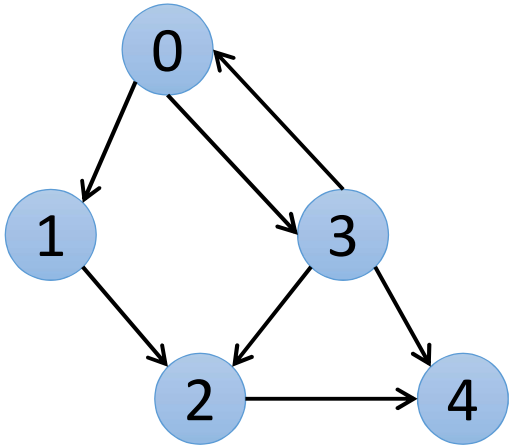


- Datos de la arista:
  - Peso: distancia entre ciudades
  - Dirección: en este caso, sería “bi-direccional”
  - Etiqueta: nombre de la carretera y tramo

Representando un grafo (1)

- Matriz de adyacencia

```
     [0][1][2][3][4]
[0] {0, 1, 0, 1, 0},
[1] {0, 0, 1, 0, 0},
[2] {0, 0, 0, 0, 1},
[3] {1, 0, 1, 0, 1},
[4] {0, 0, 0, 0, 0},
```

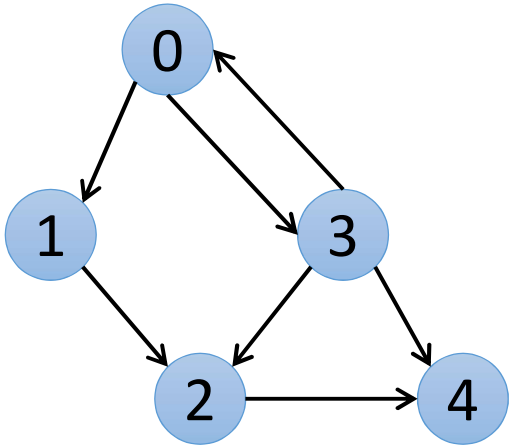


- Inconveniente: desperdicia mucha memoria

Representando un grafo (2)

- Lista de arcos

```
{{0,1},
{0,3},
{1,2},
{2,4},
{3,0},
{3,2},
{3,4}}
```

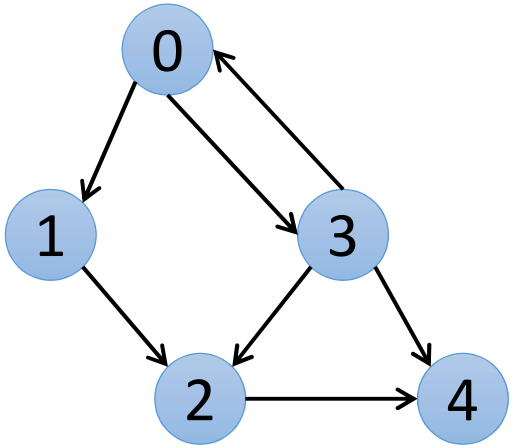


- Inconveniente: buscar una arista supone recorrer una lista entera (complejidad **O(n)**)

Representando un grafo (y 3)

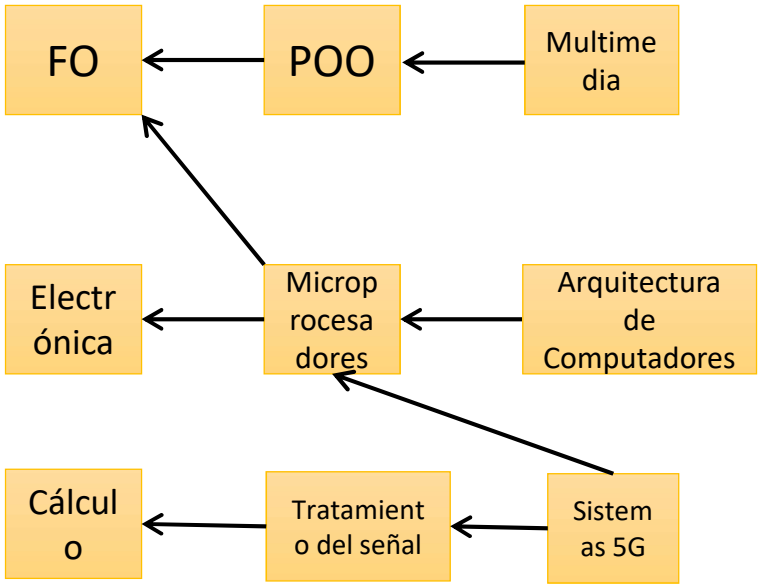
- Lista de adyacencias

```
[0] {1, 3},
[1] {2},
[2] {4},
[3] {0, 2, 4},
[4] {},
```



- Formas de representación:
  - Si el identificador del nodo es un entero: un array de listas
  - En cualquier caso: un mapa (clave: identificador del nodo, valor: lista de adyacencias).

Ejemplo: ¿Qué asignaturas necesito cursar antes?



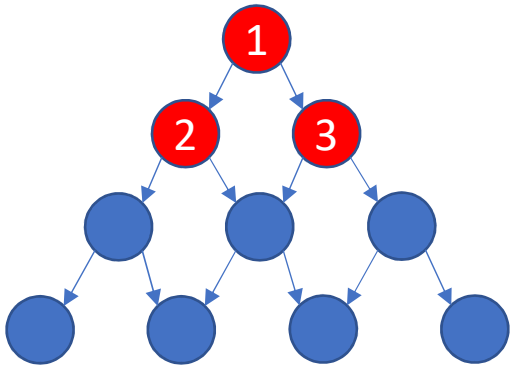
- ¿Qué debo haber cursado para poder estudiar “Sistemas 5G”?
  - Tratamiento del Señal
  - Cálculo
  - Microprocesadores
  - Electrónica
  - Fonaments Ordinadors

Ejemplo Java: ¿Qué asignaturas necesito cursar antes?

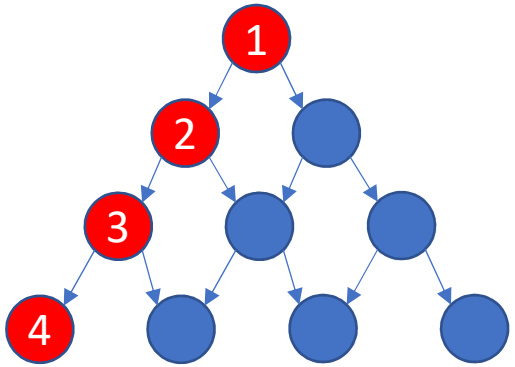
```
public class Asignatura {
  private String nombre;
  // constructores, getters, setters…
  public int hashCode() { ... }
  public boolean equals(Object o) { ... }
}

public class PlanCarrera {
  // clave: nodo, valor: nodos adyacentes
  private Map<Asignatura,Set<Asignatura>> grafo;

  // Busca asignaturas que debo estudiar antes de
  // poder estudiar la asignatura A
  public List<Asignatura> queEstudiar(Asignatura a) {

    // ¿cómo lo hago para recorrer el grafo tal que
    // me retorne TODAS y ÚNICAMENTE las asignaturas
    // antecesoras a "a"?

  }
}
```

Búsqueda en grafos

- Búsqueda en amplitud
  - ***Breadth First Search (BFS)***
  - A partir de un nodo, busca primero en todos sus nodos adyacentes, y luego en los adyacentes de éstos

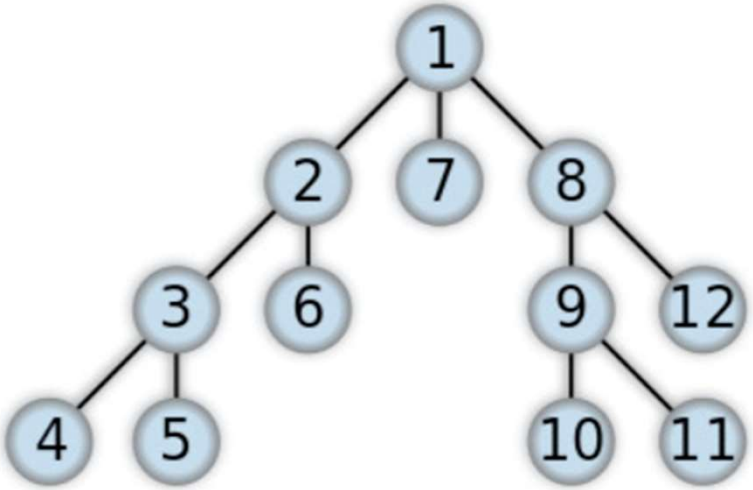


- Búsqueda en profundidad
  - ***Depht First Search (DFS)***
  - A partir de un nodo, va buscando en los adyacentes de los adyacentes….

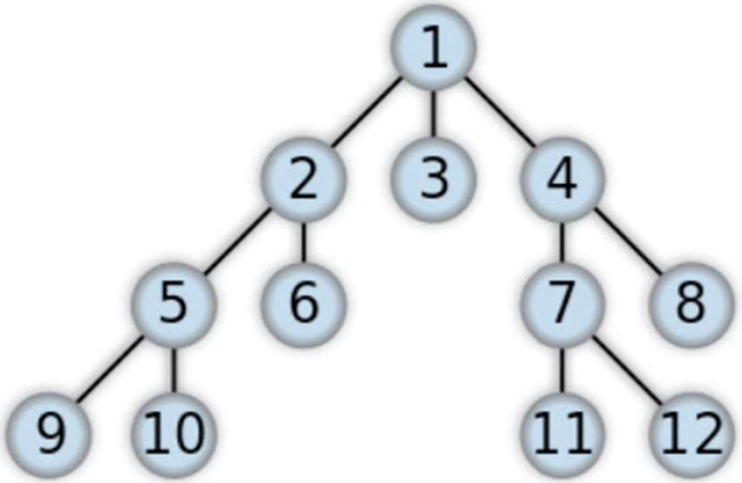


Orden de recorrido de nodos

DFS



BFS

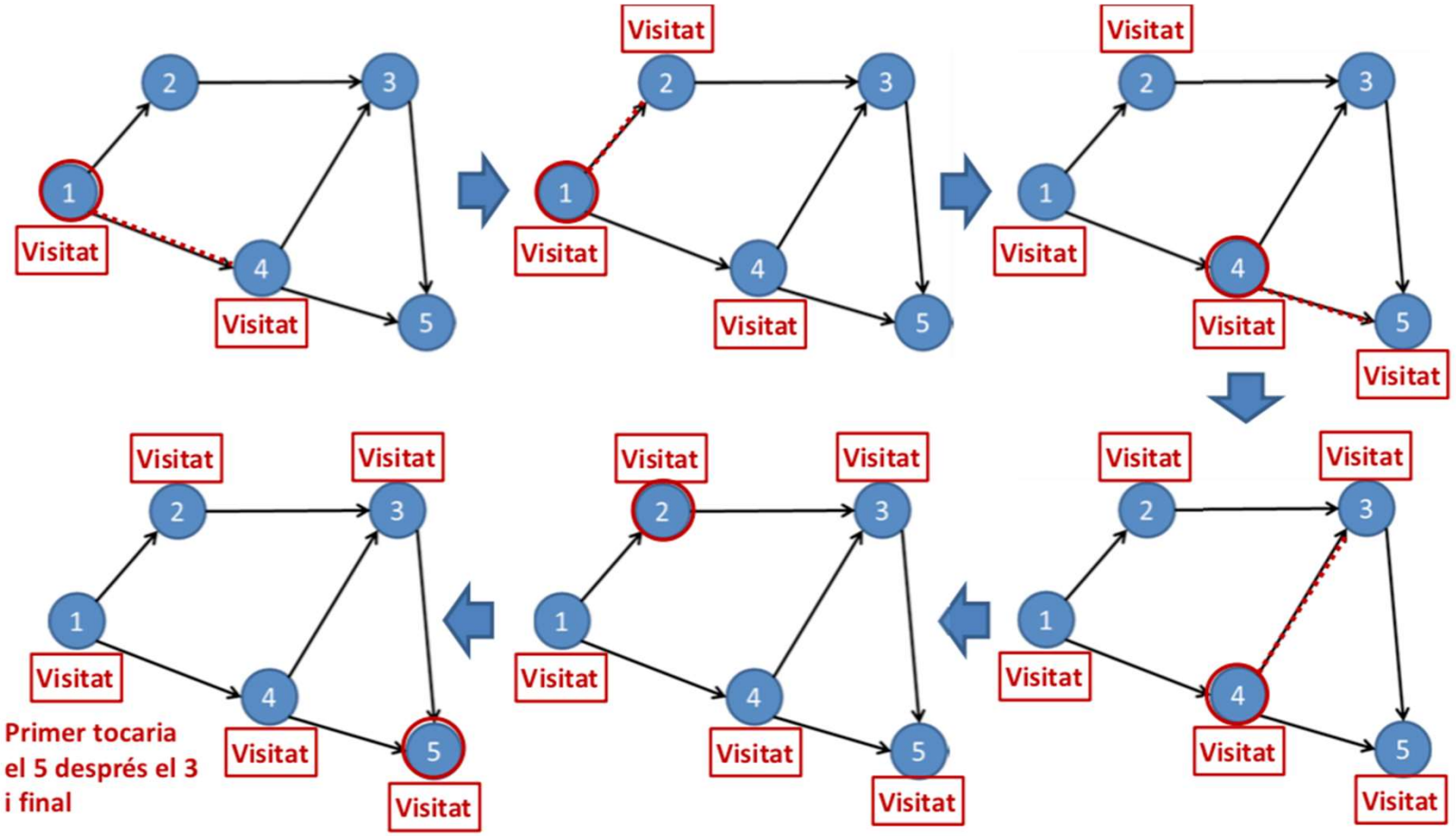


Estrategia BFS

- Necesitamos una estructura de datos del tipo “cola” (First-In First-Out)
  - Añadimos al final, sacamos del principio → LinkedList

1. Seleccionar un nodo de origen
2. Añadirlo a la cola
3. Mientras la cola no esté vacía:
   4.1 Extraer elemento de la cola → “actual”
   4.2 Si el nodo "actual" no se ha visitado ya
      4.2.1 Marcar el nodo como visitado
      4.2.2 Añadir nodos adyacentes a la cola
4. Fin

Recorriendo Nodos con BFS

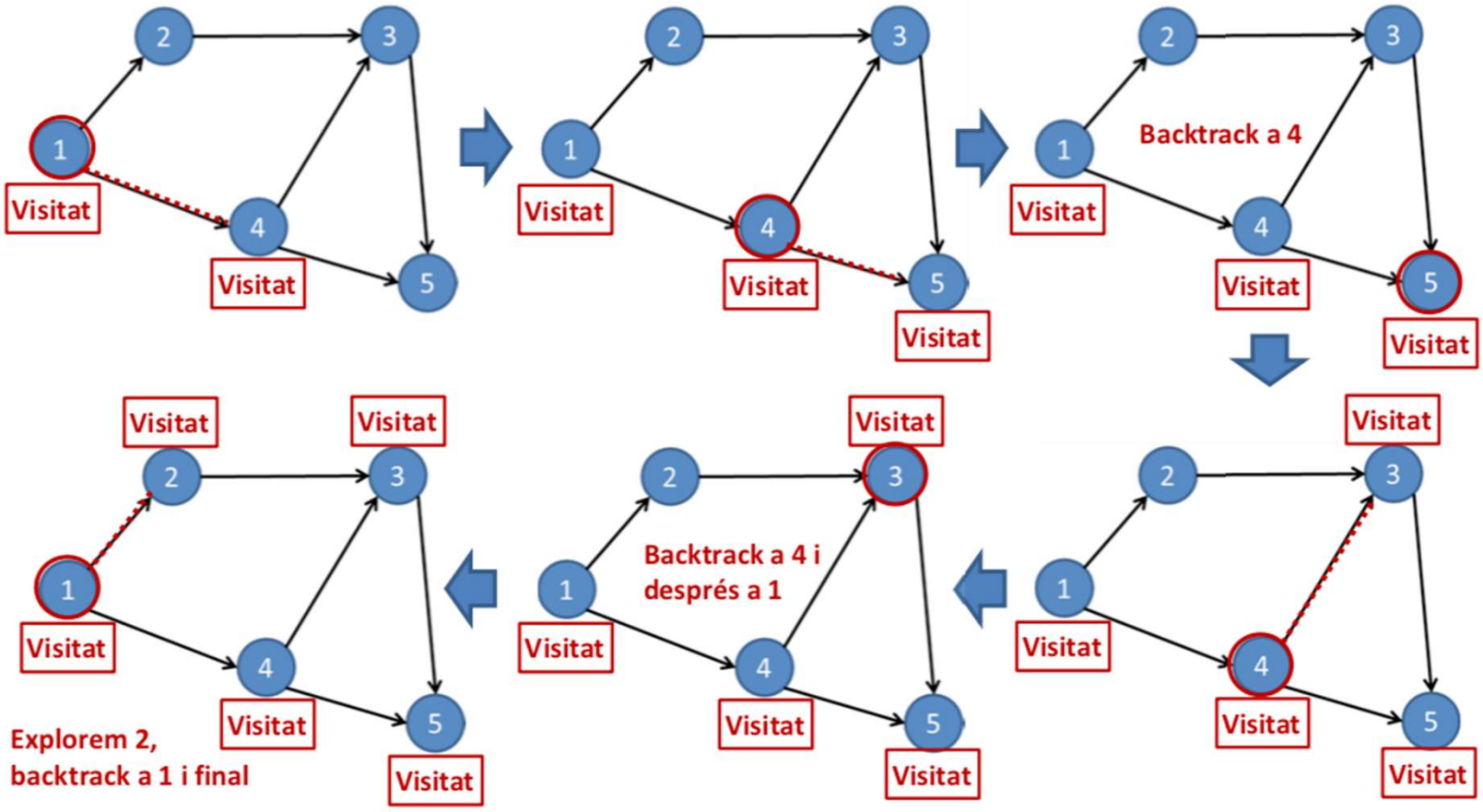


Estrategia de recorrido DFS

- Necesitamos una estructura de datos del tipo “pila” (Last-In First-Out)
  - Añadimos al final, sacamos del final

1. Seleccionar un nodo de origen
2. Añadirlo a la pila
3. Mientras la cola no esté vacía:
   4.1 Extraer elemento de la pila → “actual”
   4.2 Si el nodo "actual" no se ha visitado ya
      4.2.1 Marcar el nodo como visitado
      4.2.2 Añadir nodos adyacentes a la pila
4. Fin

Recorriendo nodos con DFS



DFS recursivo

- La recursividad se da cuando un método o función se llama a sí mismo.
- Condición indispensable:
  - En algún momento debe darse una condición en la que la función no se llama a sí misma (por ejemplo, ante un parámetro trivial)
- Ejemplo. Función factorial:

```
long factorial(long n) {
  if (n == 0) { // Caso trivial
    return 1;
  }
  return n * factorial(n - 1);
}
```

Cuándo usar recursividad

- Esta implementación sería mucho más rápida

```
long factorial(long n) {
  long f = 1;
  for (; n > 1 ; n--) {
    f *= n;
  }
  return f;
}
```

- La recursividad tiene un coste computacional.
  - Afecta a la memoria caché del procesador
  - Cada llamada a función implica:
    - Reservar espacio para variables locales
    - Guardar registros del procesador
    - Restaurar registros cuando se retorna de la llamada
- Sin embargo, la recursividad nos permite definir algunos algoritmos de manera mucho más sencilla.
  - A menudo, el mejor código es el más sencillo y legible, no el más rápido.

Ejemplo de DFS recursivo

```
void dfs(grafo, nodoOrigen) {
   1. marcar nodoOrigen como visitado
   2. para cada nodoAdyacente:
      2.1 si nodoAdyacente no ha sido
          visitado aún
        2.1.1 dfs(grafo,nodoAbyacente)
}
```