# Tema 14 – Grafos y su Implementación en Java Parte II

Germán Moltó
Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia



## Tema 14 – Grafos y su Implementación en Java

Índice general:





## Recorrido de un Grafo: Ampliación de la clase Grafo

- Los recorridos permiten obtener una secuencia de sus elementos.
- **Recorrido en Profundidad** o Depth First Search (DFS)
  - Generalización del Recorrido Pre-Orden de un Árbol
  - 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.
- **Recorrido en Anchura** o Breadth First Search (BFS)
  - Generalización del Recorrido por Niveles de un Árbol.
  - Explora las Aristas del Grafo de manera que se visitan los vértices adyacentes al explorado actualmente. Para cada uno de esos vértices, se exploran sus vecinos, provocando un recorrido en anchura.



## Recorrido en Profundidad (DFS) de un Grafo

- En Árboles Binarios conseguíamos hacer un recorrido en profundidad de la siguiente manera:

```
PreOrden (Árbol) {
   PreOrden(RaízÁrbol);
}
```

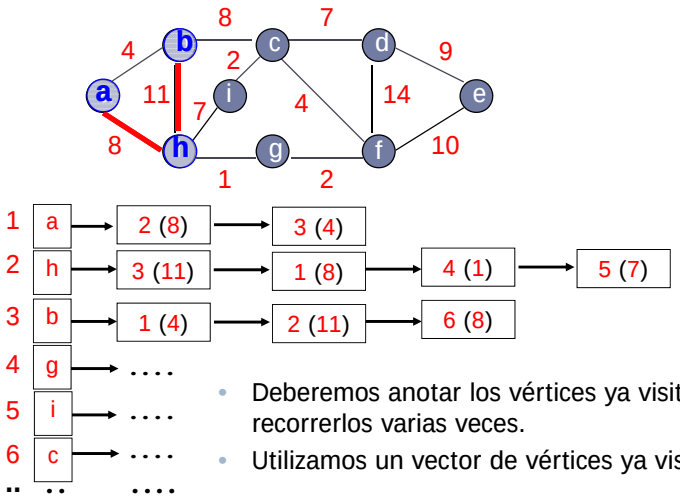
```
tratar(RaízÁrbol);
∀Hijo(RaízÁrbol)
    PreOrden(Hijo(RaízÁrbol));
```

- En Grafos lo haremos de manera similar:

```
DFS(Grafo) {
∀ Vertice_i ∈ Grafo
   DFS(Vertice_i);
}
```

```
tratar(Vértice_i);
∀ adyacente(Vértice_i)
   DFS(adyacente(Vértice_i));
```

## Ejemplo de Recorrido en Profundidad




- Deberemos anotar los vértices ya visitados para no recorrerlos varias veces.
- Utilizamos un vector de vértices ya visitados.

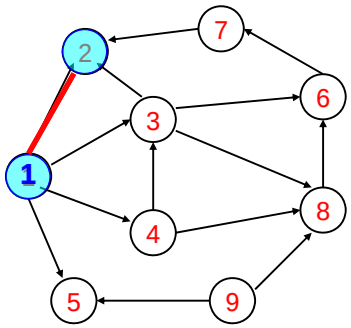
## Gestión de Visitados para DFS

- Utilizar atributos para marcar los Vértices ya visitados:
  - si visitados[i] == 0, el Vértice i NO se ha visitado aún
  - si visitados[i] > 0, el Vértice i SÍ se ha visitado.
- El valor de visitados[i] indica el orden en el que se ha visitado el Vértice:
- De esta manera se interpretará:
  - visitados[3] = 5 → El Vértice 3 se ha visitado en 5° posición, es decir, antes se han visitado otros 4 vértices.



## Traza de Recorrido en Profundidad: DFS (1/9)

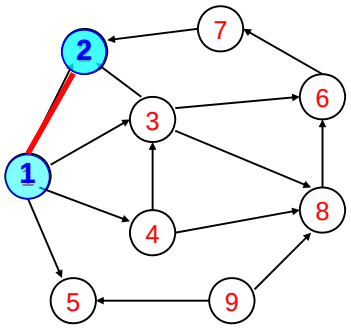
- Vértice origen: 1




| Vertice's | visitados | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ~~X~~ 2,3,4,5 | 1 | | | | | | | | |

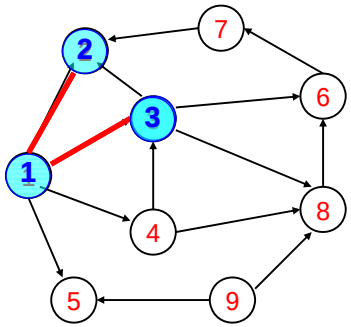


## Traza de Recorrido en Profundidad: DFS (2/9)




| Vertice's | visitados | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ~~X~~ 2,3,4,5 | 1 | - | - | - | - | - | - | - | - |
| ~~X~~ | | 2 | | | | | | | |

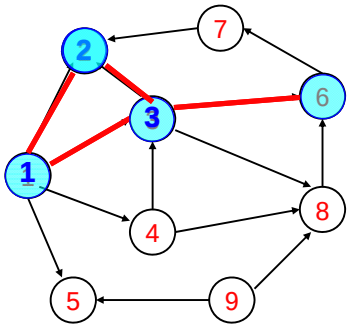
## Traza de Recorrido en Profundidad: DFS (3/9)




| Vertice's | visitados | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ~~1~~:2,3,4,5 | 1 | - | - | - | - | - | - | - | - |
| ~~2~~:2,6,8 | - | 2 | - | - | - | - | - | - | - |
| | - | - | **3** | - | - | - | - | - | - |



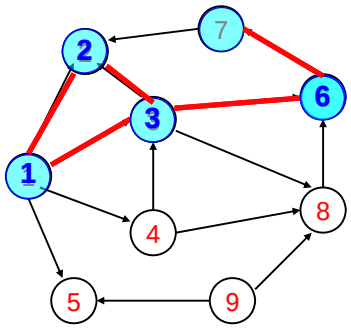
## Traza de Recorrido en Profundidad: DFS (4/9)




| Vertice's | visitados | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ~~1~~:2,3,4,5 | 1 | - | - | - | - | - | - | - | - |
| ~~2~~:2,6,8 | - | 2 | - | - | - | - | - | - | - |
| | - | - | 3 | - | - | - | - | - | - |



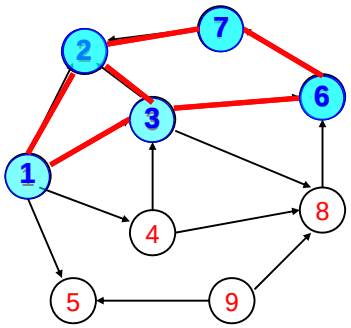
## Traza de Recorrido en Profundidad: DFS (5/9)




| Vertice's | visitados | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ~~1~~:2,3,4,5 | 1 | - | - | - | - | - | - | - | - |
| ~~2~~:2,6,8 | - | 2 | - | - | - | - | - | - | - |
| ~~3~~:7 | - | - | 3 | - | - | - | - | - | - |
| | | | | **4** | | | | | |

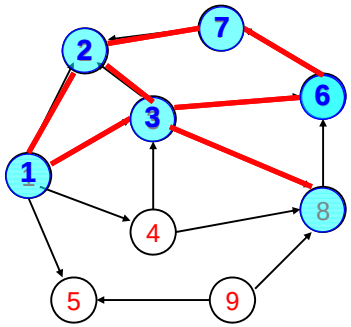


## Traza de Recorrido en Profundidad: DFS (6/9)




| Vertice's | visitados | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ~~1~~:2,3,4,5 | 1 | - | - | - | - | - | - | - | - |
| ~~2~~:2,6,8 | - | 2 | - | - | - | - | - | - | - |
| ~~3~~:7 | - | - | 3 | - | - | - | - | - | - |
| ~~7~~:2 | - | - | - | - | 4 | - | - | - | - |
| | | | | | | **5** | | | |

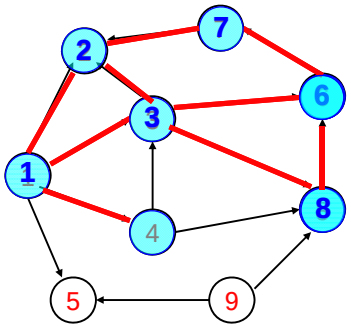
## Traza de Recorrido en Profundidad: DFS (7/9)




| Vertice's | visitados | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ~~1~~:2,3,4,5 | 1 | - | - | - | - | - | - | - | - |
| ~~2~~ | - | 2 | - | - | - | - | - | - | - |
| ~~3~~:2,6,8 | - | - | 3 | - | - | - | - | - | - |
| ~~6~~:7 | - | - | - | - | - | 4 | - | - | - |
| ~~7~~:2 | - | - | - | - | - | - | 5 | - | - |



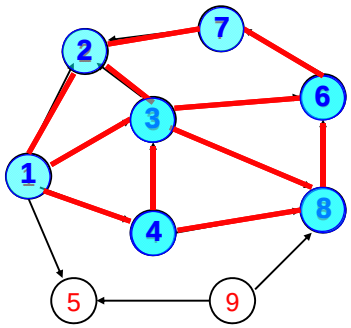
## Traza de Recorrido en Profundidad: DFS (8/9)




| Vertice's | visitados | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ~~1~~:2,3,4,5 | 1 | - | - | - | - | - | - | - | - |
| ~~2~~ | - | 2 | - | - | - | - | - | - | - |
| ~~3~~:2,6,8 | - | - | 3 | - | - | - | - | - | - |
| ~~6~~:7 | - | - | - | - | - | 4 | - | - | - |
| ~~7~~:2 | - | - | - | - | - | - | 5 | - | - |
| ~~8~~:6 | | | | | | | | | **6** |



## Traza de Recorrido en Profundidad: DFS (9/9)

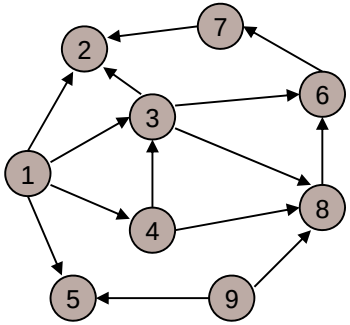



| Vertice's | visitados | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ~~1~~:2,3,4,5 | 1 | - | - | - | - | - | - | - | - |
| ~~2~~ | - | 2 | - | - | - | - | - | - | - |
| ~~3~~:2,6,8 | - | - | 3 | - | - | - | - | - | - |
| ~~6~~:7 | - | - | - | - | - | 4 | - | - | - |
| ~~7~~:2 | - | - | - | - | - | - | 5 | - | - |
| ~~8~~:6 | - | - | - | - | - | - | - | 6 | - |
| ~~4~~:3,8 | | | | | | | **7** | | |

. . . . .



## Traza de Recorrido en Profundidad: DFS (Final)




| Vertice's | visitados | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1:2,3,4,5 | 1 | - | - | - | - | - | - | - | - |
| 2 | - | 2 | - | - | - | - | - | - | - |
| 3:2,6,8 | - | - | 3 | - | - | - | - | - | - |
| 6:7 | - | - | - | - | - | 4 | - | - | - |
| 7:2 | - | - | - | - | - | - | 5 | - | - |
| 8:6 | - | - | - | - | - | - | - | 6 | - |
| 4:3,8 | - | - | - | 7 | - | - | - | - | - |
| 5 | - | - | - | - | 8 | - | - | - | - |
| 9:5,8 | - | - | - | - | - | - | - | - | 9 |
| | **1** | **2** | **3** | **7** | **8** | **4** | **5** | **6** | **9** |

- La secuencia de vértices visitados (DFS) es: 1 2 3 6 7 8 4 5 9

## Ampliación de la clase Grafo para Soporte de DFS

- Ampliamos la clase Grafo para soportar la implementación del recorrido DFS.

```
public abstract class Grafo<E>{
   protected int visitados[]; //Para el recorrido DFS
   protected int ordenVisita; //Orden de visita de los vértices

   public String toStringDFS() {...}
   protected String arrayToString(int v[]){ … }
   public int[] toArrayDFS() { … }
   protected void toArrayDFS(int origen, int res[]) { … }
} /* Fin de la clase Grafo */
```



## Métodos de Soporte para DFS (I)

```
public int[] toArrayDFS() {
      int res[] = new int[numVertices() + 1];
      visitados = new int[numVertices() + 1];
      ordenVisita = 1;
      for (int  i = 1; i <= numVertices(); i++)
         if ( visitados[i] == 0 ) toArrayDFS(i, res);
      return res;
  }
```

**Es necesario realizar el bucle *for* por si el grafo tiene varias componentes conexas (en ese caso no sería posible alcanzar todos los vértices desde el vértice origen).**



## Métodos de Soporte para DFS (II)

```
protected void toArrayDFS(int origen, int res[]) {
      res[ordenVisita] = origen;
      visitados[origen] = ordenVisita++;
      ListaConPI<Adyacente> l = adyacentesDe(origen);
      for (l.inicio(); !l.esFin(); l.siguiente()) {
         Adyacente a = l.recuperar();
         if (visitados[a.destino] == 0) toArrayDFS(a.destino, res);
      }
  }
```

**El array *res* contiene directamente la secuencia de vértices en el orden en el que son recorridos mediante DFS.**



## Métodos de Soporte para DFS (III)

```
public String toStringDFS() {
      return arrayToString(toArrayDFS());
   }

protected String arrayToString(int v[]){
      StringBuilder sb = new StringBuilder();
      for (int i = 1; i < v.length; i++)
         sb.append(v[i] + "\n");
      return sb.toString();
  }
```

# Recorrido en Anchura (BFS) de un Grafo

- En Árboles Binarios conseguíamos hacer un recorrido por niveles (o en anchura) de la siguiente manera:

```
PorNiveles (Árbol) {
    PorNiveles(RaízÁrbol);
}
```

```
Cola<NodoABB<E>> q = new
    ArrayCola<NodoABB<E>>();
q.encolar(RaízÁrbol);
mientras ( !q.esVacia() ){
    r = q.desencolar();
    tratar(r);
    ∀ Hijo(r): q.encolar(Hijo(r));
}
```

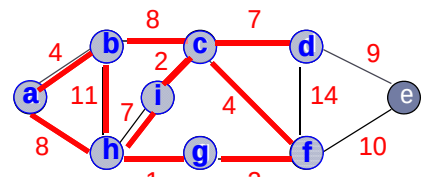
- En Grafos se hace:

```
BFS(Grafo) {
Cola<Integer> q = new
    ArrayCola<Integer>();
∀ Vertice_i ∈ Grafo
    BFS(Vertice_i);
}
```

```
tratar(Vértice_i); q.encolar(Vértice_i);
mientras ( !q.esVacia() ) {
    v = q.desencolar();
    ∀Adyacente(v):
        tratar(Adyacente(v) );
        q.encolar(Adyacente(v));
}
```



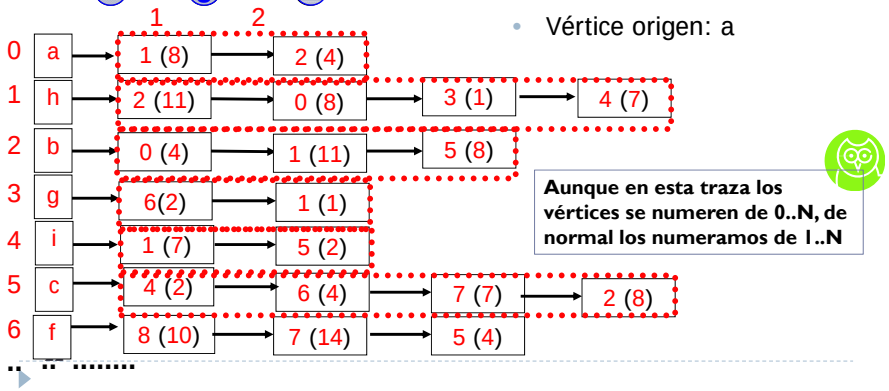
# Traza de Recorrido en Anchura: BFS




- Estado de la Cola de Vértices por procesar:

| 6| 7|

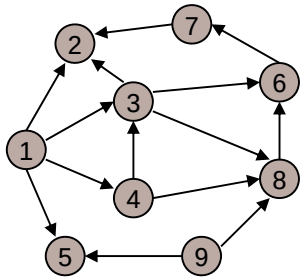
- Vértice origen: a




**Aunque en esta traza los vértices se numeren de 0..N, de normal los numeramos de 1..N**

# Traza de Recorrido en Anchura: BFS (Final)

- Vértice origen: 1




| Vértice | Visitados | Cola |
|---|---|---|
| | 1 2 3 4 5 6 7 8 9<br>0 0 0 0 0 0 0 0 0 | |
| | 1 0 0 0 0 0 0 0 0 | 1 |
| 1: 2 3 4 5 | 1 2 3 4 5 0 0 0 0 | 2 3 4 5 |
| 2: | 1 2 3 4 5 0 0 0 0 | 3 4 5 |
| 3: 2 6 8 | 1 2 3 4 5 6 0 7 0 | 4 5 6 8 |
| 4: 3 8 | 1 2 3 4 5 6 0 7 0 | 5 6 8 |
| 5: | 1 2 3 4 5 6 0 7 0 | 6 8 |
| 6: 7 | 1 2 3 4 5 6 8 7 0 | 8 7 |
| 8: 6 | 1 2 3 4 5 6 8 7 0 | 7 |
| 7: 2 | 1 2 3 4 5 6 8 7 0 | |
| 9: 5 8 | 1 2 3 4 5 6 8 7 9 | |

- La secuencia de vértices visitados (BFS) es: 1 2 3 4 5 6 8 7 9



# Ampliación de la clase Grafo para Soporte de BFS

```
public abstract class Grafo<E>{
    …
   /** Para el recorrido DFS y BFS */
   protected int visitados[];
    protected int ordenVisita;
    /** para el Recorrido en Anchura (BFS)  */
    protected Cola<Integer> q;

    public String toStringBFS() { … }
    public int[] toArrayBFS() { … }
    protected void toArrayBFS(int origen, int res[]) {  … }

} /* Fin de la clase Grafo */
```

## El método toArrayBFS (1/2)

```
public String toStringBFS() {
        return arrayToString(toArrayBFS());
    }
}
public int[] toArrayBFS() {
        int res[] = new int[numVertices() + 1];
        visitados = new int[numVertices() + 1];
        ordenVisita = 1;
        q = new ArrayCola<Integer>();
        for ( int  i = 1; i <= numVertices(); i++ )
            if (visitados[i] == 0) toArrayBFS(i, res);
        return res;
    }
```



## El método toArrayBFS (2/2)

```
protected void toArrayBFS(int origen, int res[]) {
        res[ordenVisita] = origen;
        visitados[origen] = ordenVisita++;
        q.encolar(new Integer(origen));
        while (!q.esVacia()) {
            int u = q.desencolar().intValue();
            ListaConPI<Adyacente> l = adyacentesDe(u);
            for (l.inicio(); !l.esFin(); l.siguiente()) {
                Adyacente a = l.recuperar();
                if (visitados[a.destino] == 0) {
                    res[ordenVisita] = a.destino;
                    visitados[a.destino] = ordenVisita++;
                    q.encolar(new Integer(a.destino));
                }
            }
        }
    }
```



## Aplicaciones de las Estrategias de Recorrido

- El recorrido en profundidad de un Grafo permite:
  - Determinar el número de componentes conexas, a partir del número de invocaciones al método *toArrayDFS* desde el bucle principal.
  - Determinar si un Grafo es acíclico. Si se trata de visitar un nodo ya visitado entonces existe un ciclo en el Grafo.
- El recorrido en anchura de un Grafo permite:
  - Exploración parcial de un grafo de tamaño elevado.
  - Encontrar el camino más corto entre dos nodos.