Grado en Ingeniería Información

Estructura de Datos y Algoritmos

Sesión 15

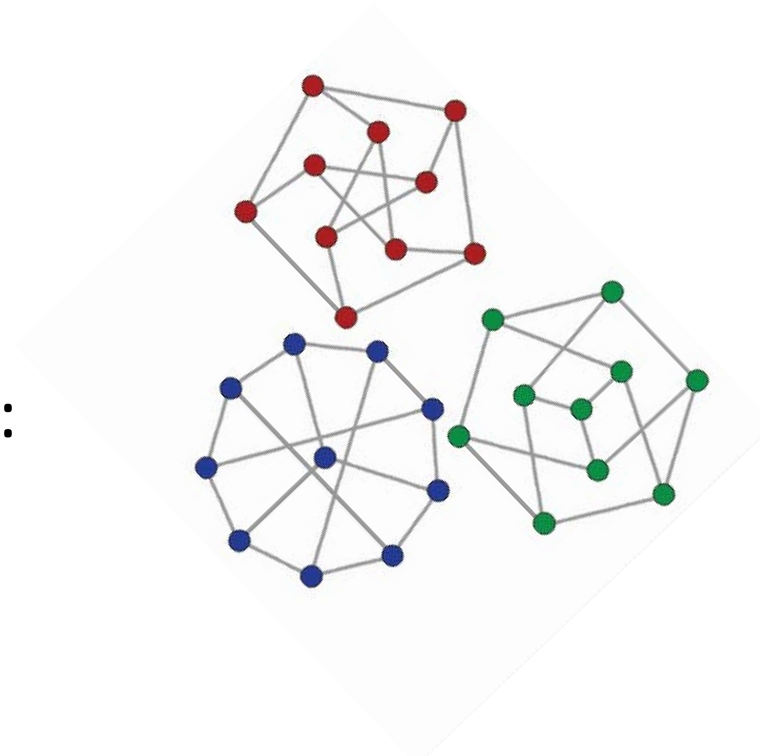
Curso 2023-2024

Marta N. Gómez

UNIVERSIDAD NEBRIJA

T3. Tipos Abstractos de Datos (TAD)

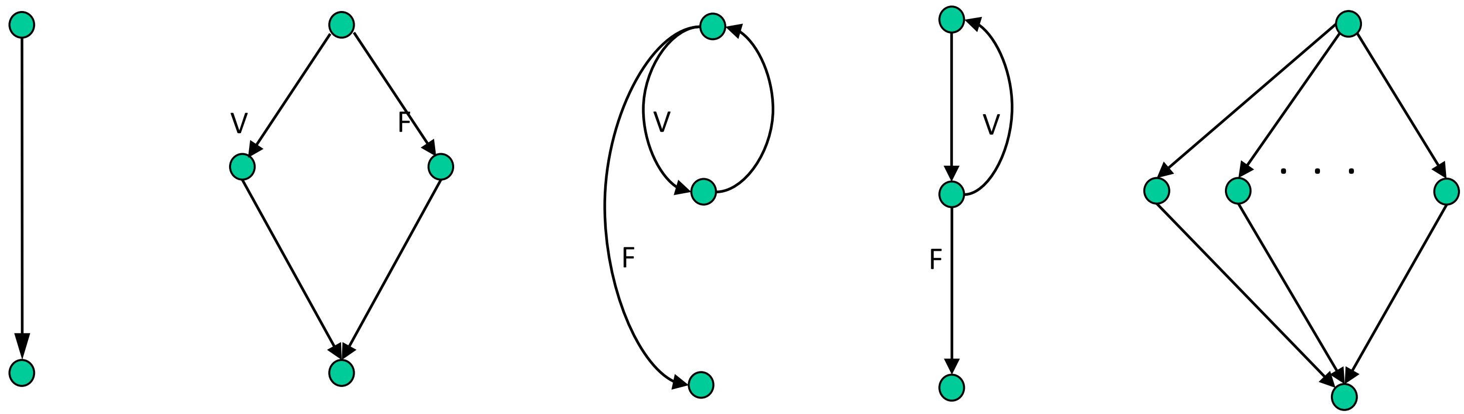
- **Grafos.**
  - Introducción
  - Definiciones básicas
  - Implementación de grafos:
    - Matrices de Adyacencia
    - Listas de Adyacencia



UNIVERSIDAD NEBRIJA

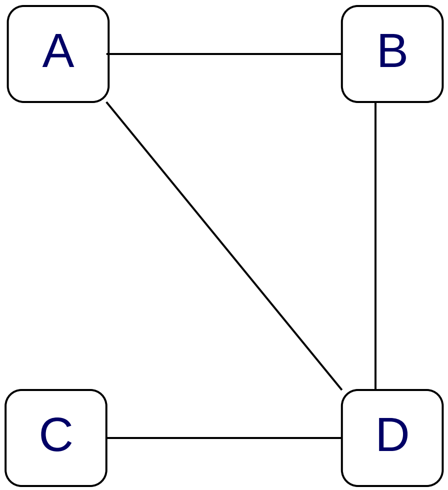
ÍNDICE

El **grafo G** se representa por G = (V, E), donde **V** es el *conjunto de vértices o nodos* y **E** el conjunto de *arcos o aristas*

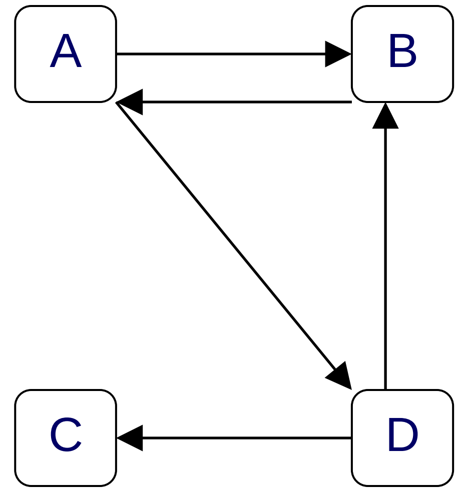


secuencia *if-else* *while* *do-while* *switch-case*

Grafo no dirigido



Grafo dirigido



E(G no dirigido) = {(A,B), (B,A), (B,D), (D,B), (A,D), (D,A), (C,D), (D,C)}

E(G dirigido) = {(A,B), (B,A), (A,D), (D,B), (D,C)}

Grafo Completo

Un grafo se dice **completo** si pertenecen al conjunto de arcos **E** **todos los arcos posibles**.

Gráficamente, el número de arcos será:

- ✓ ***Grafos no dirigidos***: $n (n - 1) / 2$
- ✓ ***Grafos dirigidos***: $n (n - 1)$

Adyacencia e Incidencia

Si existe un arco (v1, v2) $\in$ **E**(G), se dice que el vértice v2 es **adyacente** al vértice v1, además se dice que dicho arco es ***incidente*** en el vértice v2.

Subgrafo Se dice que G1 es un **subgrafo** del grafo G si se cumple que: $V(G1) \subseteq V(G)$ $E(G1) \subseteq E(G)$

Camino Un **camino** desde el vértice $\mathbf{v_i}$ al vértice $\mathbf{v_j}$ en un grafo G es una **secuencia** de vértices: $v_i, v_{k1}, v_{k2}, \ldots, v_{kn-1}, v_{kn}, v_j$, tal que:

$(v_i, v_{k1}), (v_{k1}, v_{k2}), \ldots, (v_{kn-1}, v_{kn}), (v_{kn}, v_j) \in E(G)$.

Longitud de un camino Es el **número de arcos** que forman el **camino**.

Conexión entre vértices Se dice que V_i y V_j están conectados si existe un camino en el grafo G desde V_i hasta V_j. Cuando el grafo NO es dirigido, también existirá un camino desde V_j hasta V_i.

Grafo conexo Un grafo NO dirigido se dice que es **conexo** si para cada par de vértices V_i y V_j distintos, **existe un camino en G**. Es decir, **si cada par de vértices están conectados**.

Grado

- ✓ *Grafos No Dirigidos:*

  Grado de un vértice es el **número de arcos que tienen dicho vértice como extremo**.

- ✓ *Grafos Dirigidos:*

  Grado de entrada de un vértice es el **número de arcos que llegan** o **inciden** en dicho vértice.

  Grado de salida de un vértice es el **número de arcos que salen** de dicho vértice.

Implementación de grafos dependiendo de los Conjuntos

Realizaciones del **conjunto de vertices**:

1. Un array lógico, donde cada posición del vector indica con un 1 ó 0 la existencia o ausencia del elemento en el conjunto.
2. Un array de los elementos del conjunto uno detrás de otro.
3. Una lista simplemente enlazada de los elementos que forman parte del conjunto.

El **conjunto de arcos** es un conjunto de **pares de vértices** y su **<u>implementación determinará</u>** la **representación del grafo**.

Un **recorrido de un grafo** consiste en dado un determinado vértice, **visitar todos aquellos otros vértices del grafo que son accesibles desde el vértice de partida**, en un determinado orden.

1. Recorrido en **profundidad** (DFS: Depth First Search).
2. Recorrido en **anchura** (BFS: Breadth First Search).

I M P O R T A N T E

1. Un recorrido proporciona el conjunto de vértices accesibles desde un determinado nodo.

2. Dichos vértices accesibles quedan almacenados en un conjunto de vértices, **VISITADOS**.

Algoritmo:

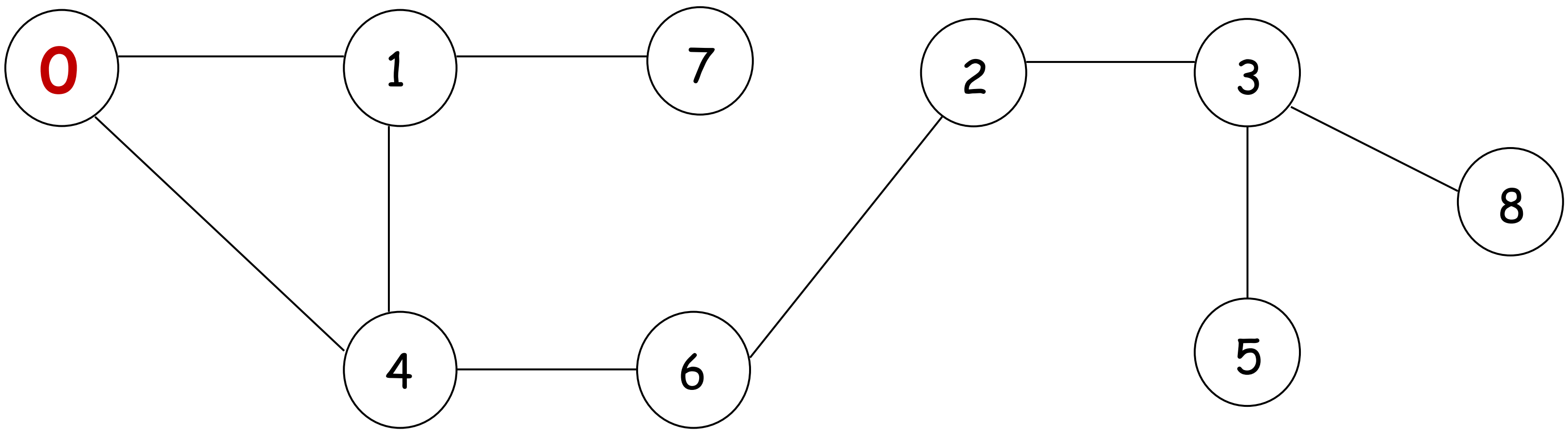
1. Se visita el **vértice de partida**, V.

2. Se selecciona un vértice, W, **adyacente a V** y que aun **no haya sido visitado**.

3. Se realiza el <u>recorrido en profundidad partiendo de dicho vértice W</u>.

4. Cuando se encuentra un vértice cuyo conjunto de adyacentes ya han sido visitados en su totalidad, se **retrocede hasta el último vértice visitado que tenga más vértices adyacentes no visitados** y <u>se ejecuta desde él el paso 2</u>.

Operaciones con Grafos – Recorrido en Profundidad

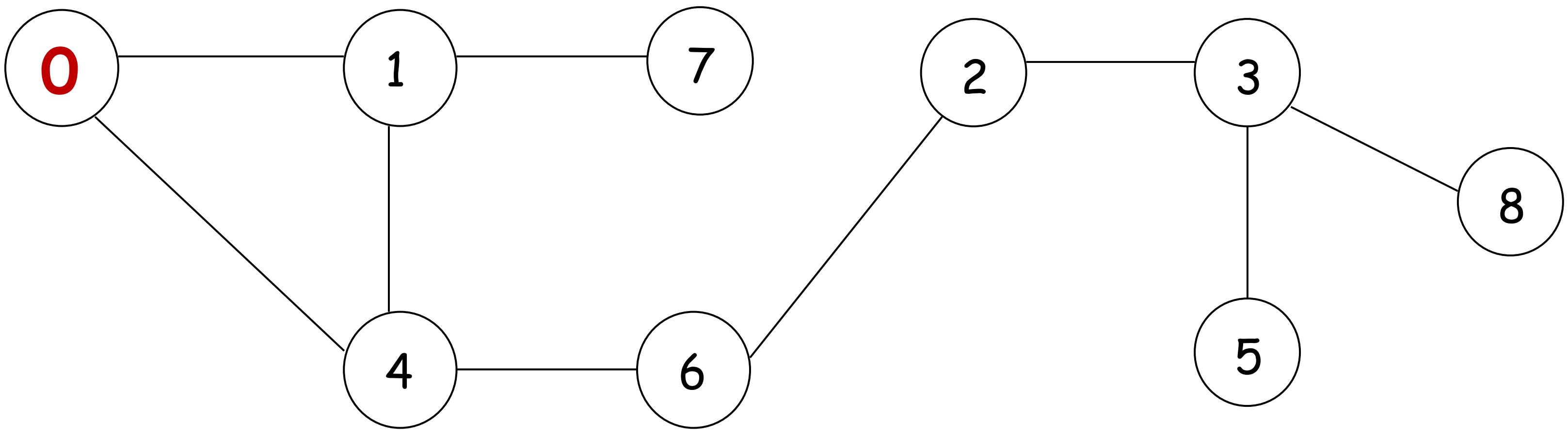


0	1	4	
1	0	4	7
2	3	6	
3	2	5	8
4	0	1	6
5	3		
6	2	4	
7	1		
8	3		

Algoritmo:

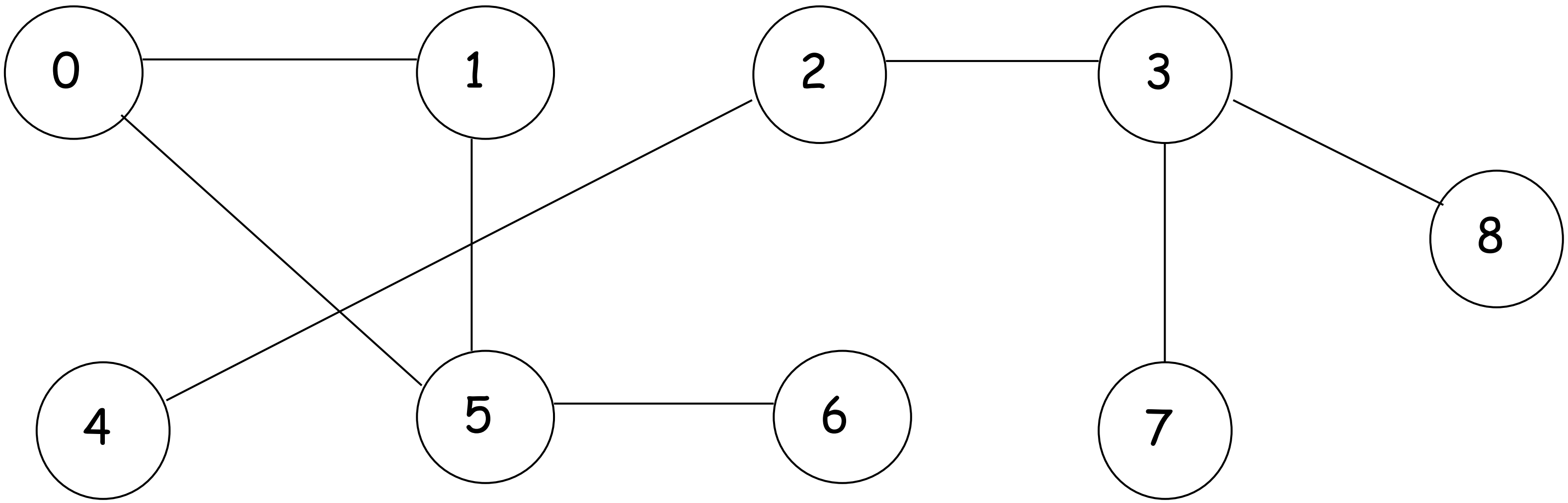
1. Se **visita el vértice de partida** del recorrido, V.
2. Se **visitan todos sus vértices adyacentes que no hayan sido visitados**. Se continúa así sucesivamente hasta terminar con todos los vértices del grafo.

Operaciones con Grafos – Recorrido en Anchura

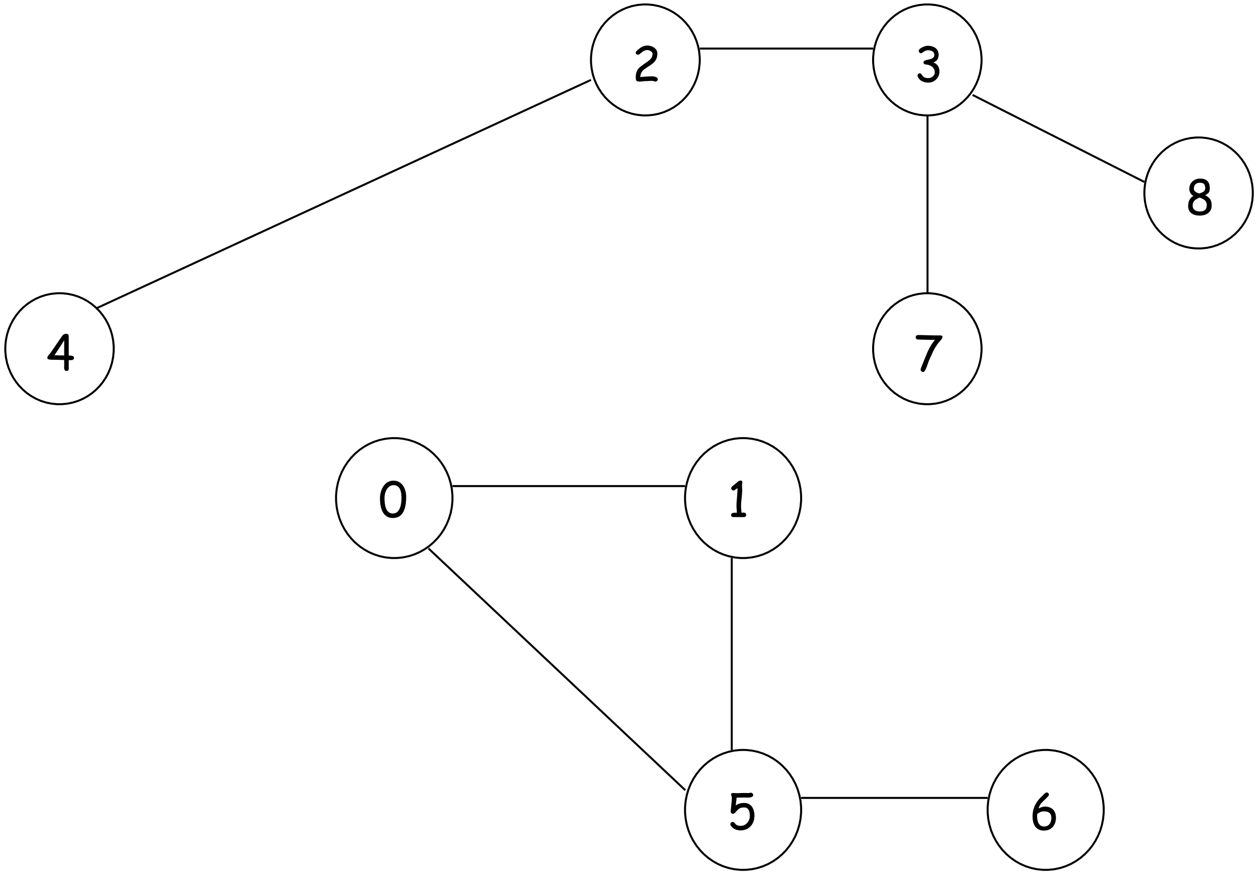


0	1	4	
1	0	4	7
2	3	6	
3	2	5	8
4	0	1	6
5	3		
6	2	4	
7	1		
8	3		

Un ejemplo de grafo **NO CONEXO** sería el siguiente:



Las componentes conexas del grafo anterior son las siguientes:

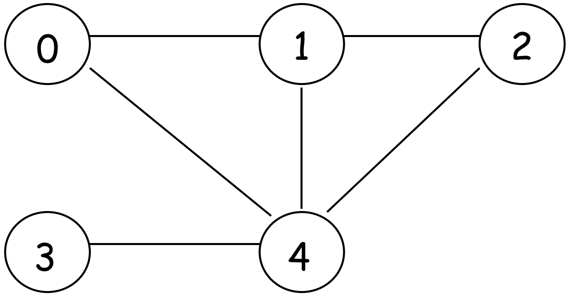


Implementación de grafos mediante Matrices de Adyacencia

Sea el grafo G = (V, E) donde V = { V0, V1, V2, . . ., Vn-1}.

La **matriz de adyacencia** de G será una matriz A de n x n elementos, cada uno de los cuales toma valores lógicos:

$$A(i,j) = \begin{cases} 1 & \text{si } [V_i, V_j] \in E \\ 0 & \text{en caso contrario} \end{cases}$$



	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	0	1
2	0	1	0	0	1
3	0	0	0	0	1
4	1	1	1	1	0

La **matriz de adyacencia** es una **matriz en la que se indican las conexiones existentes entre los vértices del grafo**. Para grafos no dirigidos, la matriz será simétrica.

UNIVERSIDAD NEBRIJA

Implementación de grafos mediante **Matrices de Adyacencia**

Los nodos se identifican por su posición (índice en un array o vector)

```
class Grafo  {
public:
      Grafo();
      void incluir_arco(int origen, int final) {
            cArcos[origen][final] = true;
      }
      void borrar_arco(int origen, int final) {
            cArcos[origen][final] = false;
      }
private:
      array<array<bool, N>, N> cArcos;
      array<bool, N> cVertices;     // set también valdría
};
```

Implementación de grafos mediante Matrices de Adyacencia

Ventajas sobre la **matriz de adyacencia** son:

- El **orden de eficiencia** de las <u>operaciones de obtención de un arco y del coste asociado</u>, cuando existe.
- La <u>comprobación de conexión entre dos vértices cualesquiera</u> es independiente del número de vértices y de arcos del grafo.

Hay dos grandes inconvenientes, que justifican la existencia de otra representación, :

- **Representación orientada a grafos que no modifican el número de sus vértices**. Una matriz no permite que se supriman filas o columnas.
- Puede producir un gran **derroche de memoria en grafos poco densos**, con <u>gran número de vértices y escaso número de arcos</u>.

Implementación de grafos mediante Listas de Adyacencia

```
class datoV  {
    public:
        datoV();
        // gets y sets necesarios
    private:
        int w;
};
class nodoV {
    private:
        datoV   w;
        shared_ptr<nodoV> next = nullptr;
    public:
        nodoV();
        // métodos necesarios
};
```

UNIVERSIDAD NEBRIJA

Implementación de grafos mediante Listas de Adyacencia

```
class listaVertices {
    private:
        shared_ptr<nodoV> ppio;
    public:
        listaVertices():ppio(nullptr){}
        // métodos necesarios
};
class datoA {
    private:
        datoV   w;
        listaVertices  cAdy;
    public:
        datoA();
        // métodos necesarios
};
```

Implementación de grafos mediante Listas de Adyacencia

```
class nodoA {
    private:
        datoA   a;
        shared_ptr<nodoA> next = nullptr;
    public:
        nodoA();
        // métodos necesarios
};
class listaArcos {
    private:
        shared_ptr<nodoA> ppio;
    public:
        listaArcos():ppio(nullptr){}
        // métodos necesarios
};
```

Implementación de grafos mediante Listas de Adyacencia

```
class Grafo  {
   public:
      Grafo();
      // métodos necesarios
   private:
      listaArcos        cArcos;
      listaVertices    cVertices;
};
```