Programación II
Grafos

2C 2023 TN – Ing. Elizabeth Barrera

Temas

- *TDA*
- *Grafos*
- *Especificación*
- *Ejemplos*
- *Implementación estática*
- *Implementación dinámica*

TDA

- *Es una **abstracción**, ignoramos algunos detalles y nos concentramos en los que nos interesan.*
- *A la definición del TDA la llamamos **especificación** y a la forma de llevar a cabo lo definido lo denominamos **implementación**.*

Recordar que:
*Existen siempre **2 visiones** diferentes en el TDA: usuario e implementador.*
Son separadas, y una oculta a la otra.


Grafos

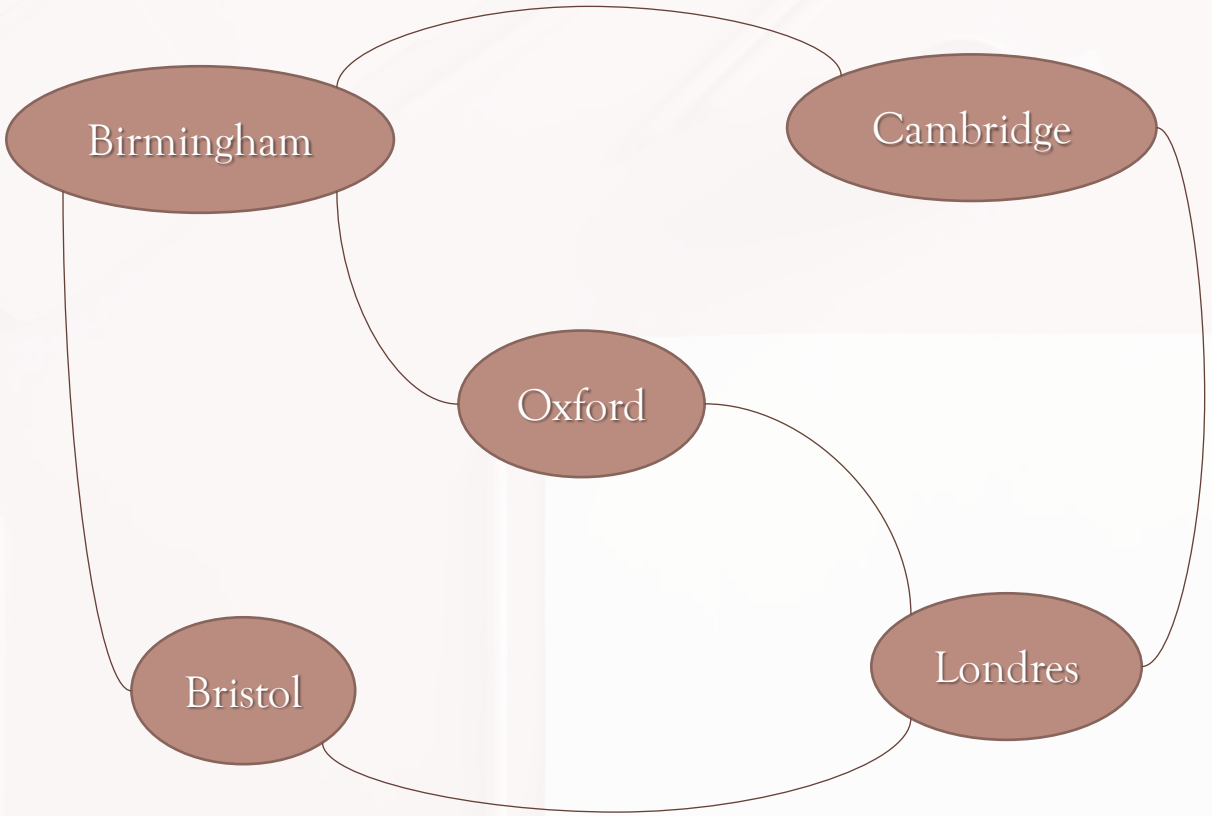
Grafos

Un grafo es un conjunto de elementos llamados nodos o ***vértices*** *y un conjunto de pares de vértices llamados* ***aristas****.*

Cada arista tiene un valor entero asociado al que llamaremos ***peso****.*

Un grafo sirve para representar una ***red****.*

Grafos

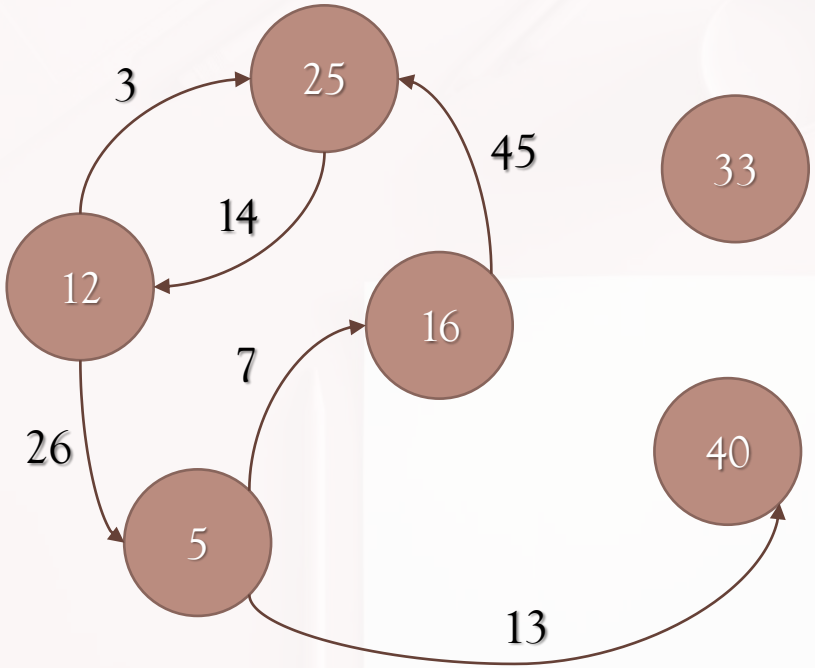


Grafos dirigidos

Un caso particular de grafos, son aquellos en los que las aristas tienen un sentido, y se denominan ***grafos dirigidos****. La arista tiene un vértice origen y un vértice destino.*

Un vértice tiene ***aristas entrantes*** *si existe al menos una arista que tiene al vértice como destino. Un vértice tiene* ***aristas salientes*** *si existe al menos una arista que tiene al vértice como origen.*

Grafos dirigidos



Grafos dirigidos - definiciones

- Un vértice que no tiene aristas entrantes ni salientes es un ***vértice aislado***.
- La cantidad de aristas salientes se llama ***grado positivo*** de un vértice. La cantidad de aristas entrantes se llama ***grado negativo*** de un vértice.
- Un vértice v_2 es ***adyacente*** de v_1, si existe una arista de v_1 a v_2. En el caso de grafos dirigidos, puede suceder que uno sea adyacente de otro, pero no recíprocamente.

Grafos dirigidos - definiciones

- Un ***camino***, es una sucesión ordenada de vértices, en donde existe una arista entre el primero y el segundo, el segundo y el tercero, y así sucesivamente. El ***costo del camino*** es la suma de los pesos de cada arista.
- Un ***ciclo*** es un camino que comienza y termina en un mismo vértice.
- Un vértice es ***alcanzable*** desde otro si existe un camino que una al segundo con el primero.

Grafos dirigidos - definiciones

- Las ***aristas paralelas*** son un par de aristas con mismo origen y mismo destino.
- Un ***bucle*** es una arista que tiene igual origen y destino.
- Aclaración: vamos a trabajar con grafos dirigidos, sin aristas paralelas ni bucles, donde los vértices son números enteros y los pesos son enteros positivos.

Especificación - Operaciones

- ***inicializarGrafo***: *inicializa el grafo.*
- ***agregarVertice***: *agrega un nuevo vértice al grafo, siempre que el grafo esté inicializado y el vértice no exista.*
- ***eliminarVertice***: *elimina el vértice dado, siempre que el vértice exista.*
- ***vertices***: *devuelve el conjunto de vértices de un grafo, siempre que el grafo esté inicializado.*

Recordar que:
*Las **precondiciones**, son condiciones que deben cumplirse antes de la ejecución de la operación.*

Especificación - Operaciones

- ***agregarArista***: *agrega una arista al grafo entre los vértices y con el peso dados, siempre que existan ambos vértices pero no exista una arista entre ambos.*
- ***eliminarArista***: *elimina la arista entre los vértices dados, siempre que esta arista exista.*
- ***existeArista***: *indica si el grafo contiene una arista entre los vértices dados, siempre que los vértices existan.*
- ***pesoArista***: *devuelve el peso de la arista entre los vértices dados, siempre que la arista exista.*

Recordar que:
*Las **precondiciones**, son condiciones que deben cumplirse antes de la ejecución de la operación.*

Especificación - Interfaz

```
public interface GrafoTDA {
  void inicializarGrafo( );
  void agregarVertice(int v); //grafo inicializado y ∄ vértice
  void eliminarVertice(int v); //grafo inicializado y ∃ vértice
  ConjuntoTDA vertices(); //grafo inicializado
  void agregarArista(int v1, int v2, int peso); //grafo
       inicializado, ∄ arista y ∃ ambos vértice
  void eliminarArista(int v1, int v2); //grafo inicializado y
       ∃ arista
  boolean existeArista(int v1, int v2); //grafo inicializado y
       ∃ ambos vértices
  int pesoArista(int v1, int v2); //grafo inicializado y ∃ arista
}
```

Uso

Uso - Ejemplos

- *Calcular el conjunto de los predecesores de un vértice v.*
- *Aclaración: un vértice "origen" es predecesor de otro vértice "destino", si hay una arista que comienza en "origen" y termina en "destino".*

Implementación

Implementación estática

- Se utilizan ***matrices de adyacencia***.
- Una matriz de adyacencia es una ***matriz cuadrada*** donde la cantidad de filas y columnas está dada por la cantidad de vértices del grafo.
- El contenido de cada celda es el ***peso*** de la arista con origen en el número de fila y destino en el número de columna.

Implementación estática

- Si no existe ninguna arista que vaya de un vértice i a un vértice j, el contenido de la posición i-j es 0.
- Como los índices van de cero en adelante en forma consecutiva, y los vértices pueden tener valores no continuos, se utiliza un ***mapeo*** que indique en qué índice de la matriz está el vértice que se busca.
- Este mapeo se realiza con un ***arreglo de etiquetas***.

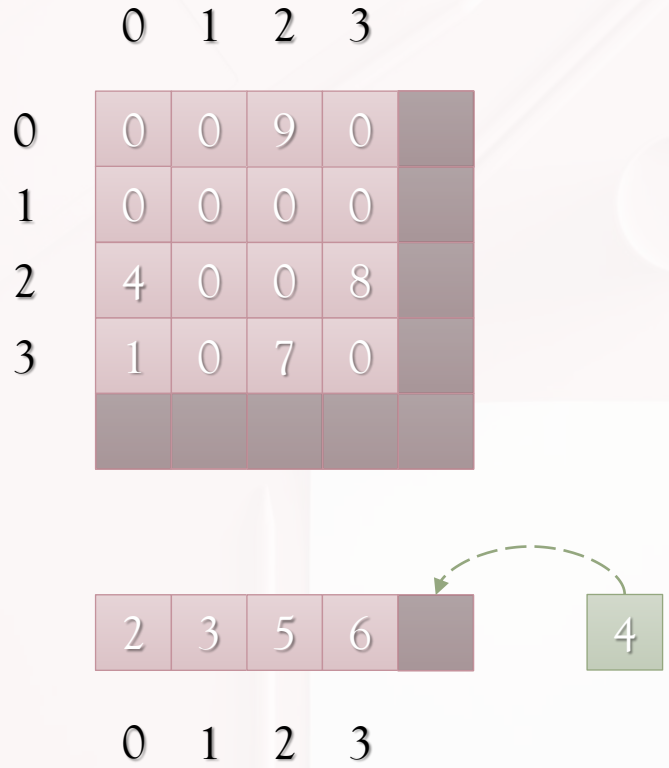
Implementación estática

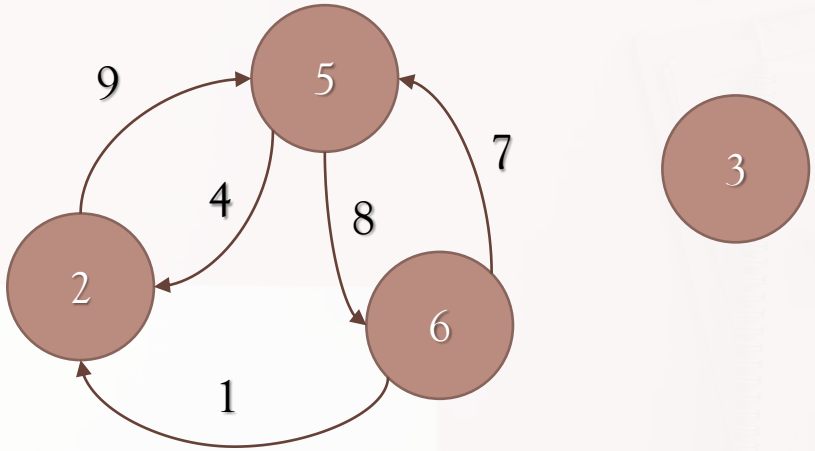
- Para **agregar** un **vértice** se completan con 0s nuevas filas y columnas en las primeras posiciones libres de la matriz y el arreglo de etiquetas (cantVertices), y se incrementa cantVertices.
- Para **eliminarlo**, se pisa la fila y columna correspondientes, con la última (de índice cantVertices-1), y se decrementa cantVertices.

Implementación estática

- *Para **agregar** una **arista** nueva, se ubican los índices de los vértices origen y destino, y se completa dicha celda con el peso correspondiente.*

Grafos - Implementación estática





Implementación dinámica

- Utilizaremos ***listas de adyacencia***.
- Se tiene una lista encadenada de vértices ***NodoVertice***. Cada vértice tiene su valor, una referencia al próximo y una referencia a la lista de aristas que tienen a ese vértice como origen.
- Las aristas están representadas con los vértices ***NodoArista***, que tiene el peso, una referencia al vértice destino y una referencia a la próxima arista.

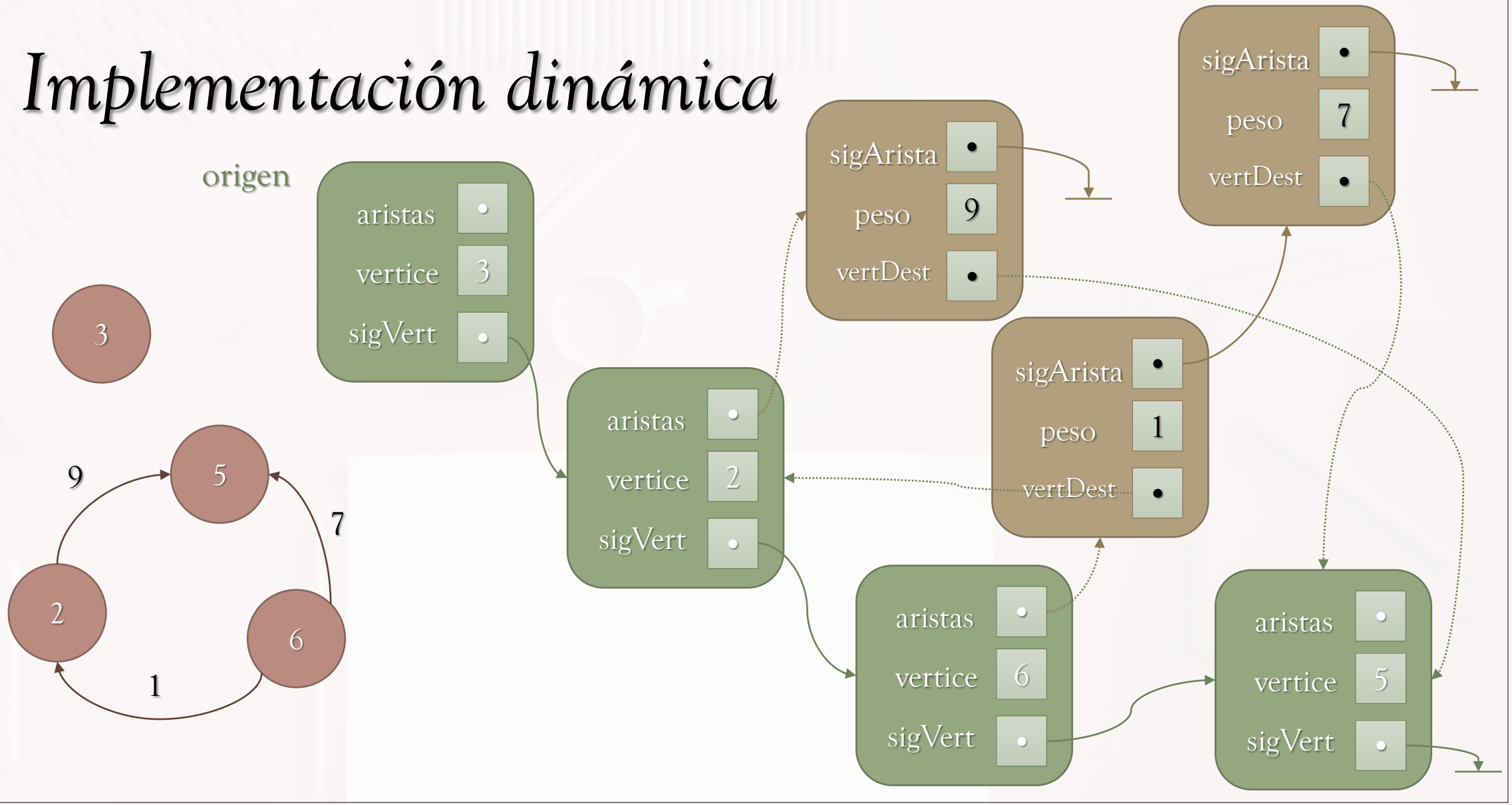
NodoVertice

```
class NodoVertice {
    int vertice;
    NodoArista aristas;
    NodoVertice sigVertice;
}
```

NodoArista

```
class NodoArista {
    int peso;
    NodoVertice verticeDestino;
    NodoArista sigArista;
}
```

Implementación dinámica




¡Muchas Gracias!

Bibliografía

- *Programación II – Apuntes de Cátedra – V1.3 – Cuadrado Trutner – UADE*
- *Programación II – Apuntes de Cátedra – Wehbe – UADE*