GRAFOS VALORADOS

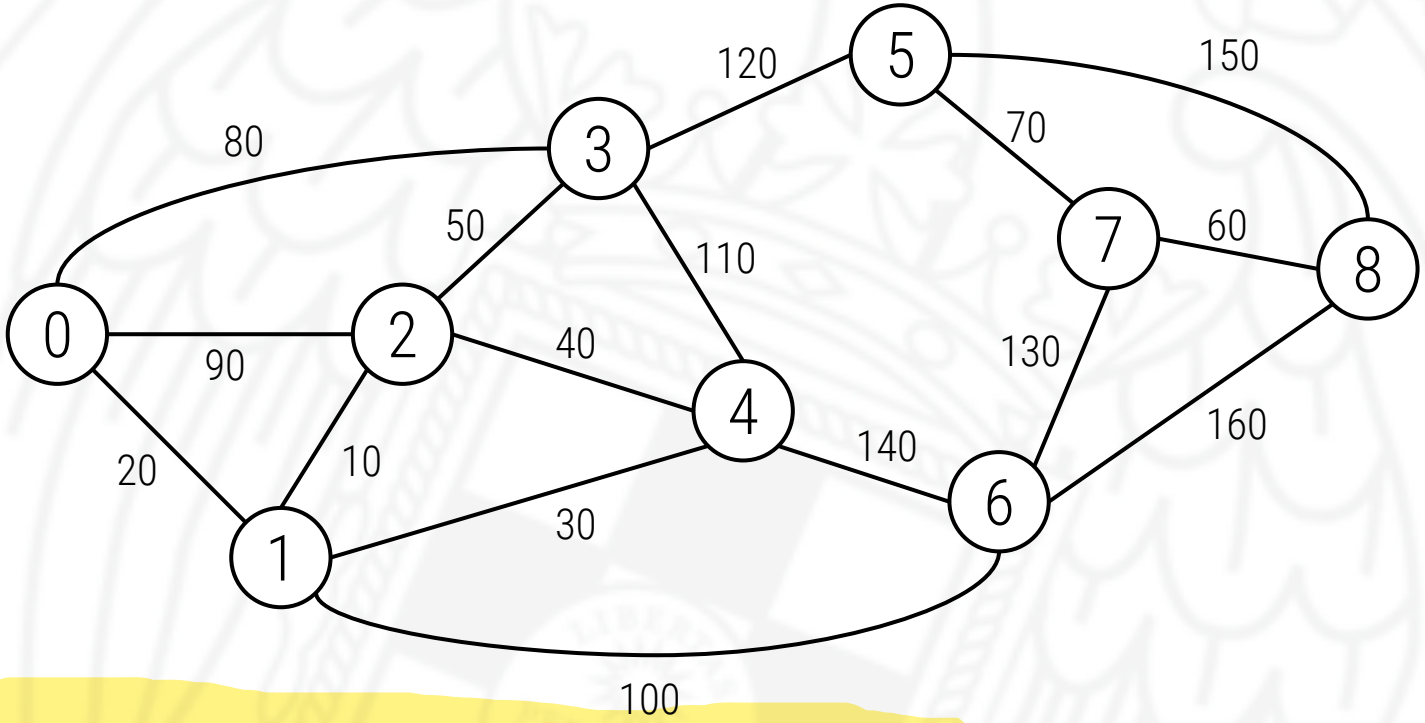
Grafos cuyas aristas tienen asignadas un valor, un peso o coste.

UNIVERSIDAD COMPLUTENSE MADRID

ALBERTO VERDEJO

Grafos valorados

- Grafos cuyas aristas tienen asociado un valor (peso, coste).



Si el problema que estamos modelando es un problema de ciudades, los valores de las aristas pueden ser la distancia.. la tarifa del viaje...

Los grafos valorados pueden ser dirigidos y no dirigidos. Aquí nos vamos a centrar en los NO dirigidos. Le dedicaremos otro vídeo a los dirigidos

Podemos hacer las mismas cosas que con un grafo no valorado. Contar el grado de un vértice, saber si el grafo es conexo o no lo es... Y simplemente ignoraríamos los valores de las aristas.

La mayoría de las veces, la representación más apropiada de estos grafos valorados es la lista de adyacencia. También se puede con la matriz de adyacencia. Grafos valorados suelen ser también dispersos.

Aristas

GrafoValorado.h

```
template <typename Valor>
class Arista {
public:
    Arista(int v, int w, Valor valor);
    int uno() const;
    int otro(int u) const;
    Valor valor() const;
    bool operator<(Arista<Valor> const& b) const;
    bool operator>(Arista<Valor> const& b) const;
};
```

Los extremos de la arista y el valor de la misma.

Devuelve uno de los extremos

Nos devuelve el otro extremo de la arista U debe ser uno de los extremos.

Valor de la arista.

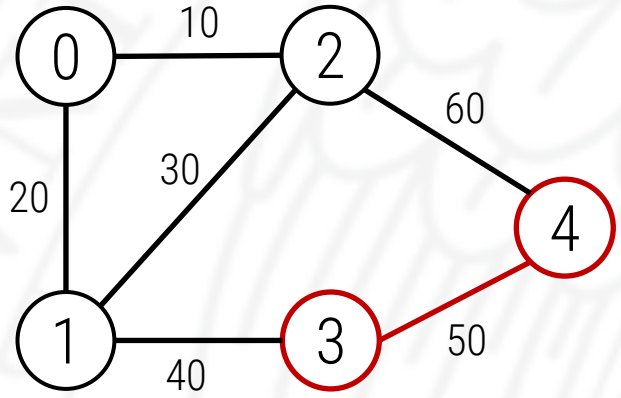
```
int v = arista.uno(), w = arista.otro(v);
```

Código para saber los extremos de una arista.

Listas de adyacencia

GrafoValorado.h

Listas son vectores de longitud variable



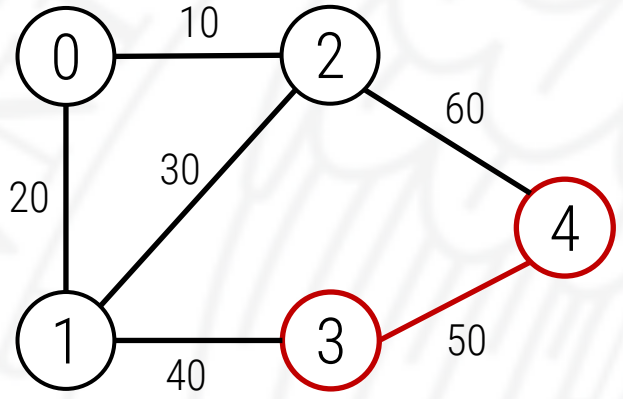
Aparecen los extremos y después el valor.

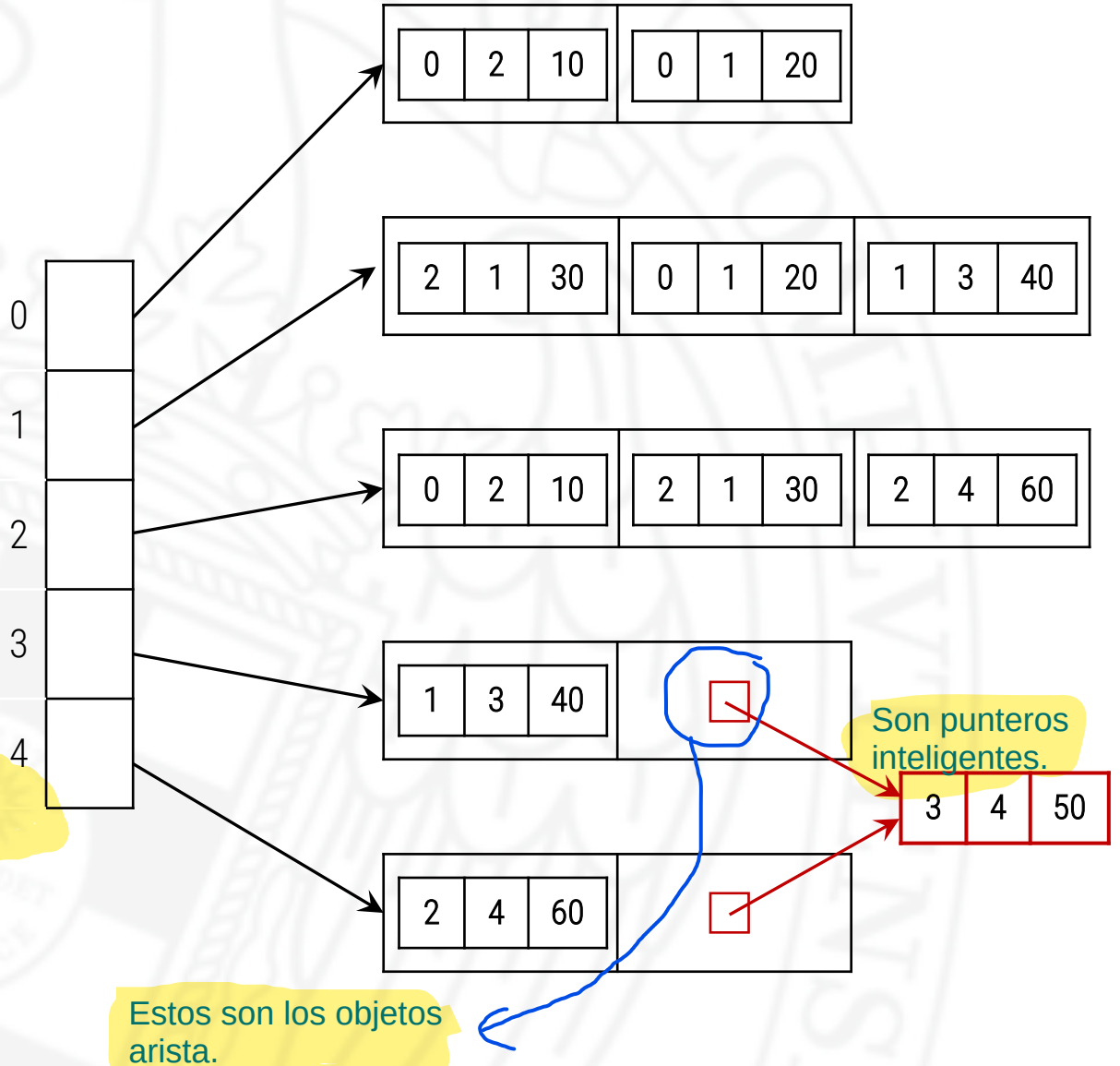
Vector indexado por los vértices de listas de aristas

Cada arista aparece dos veces en la lista de adyacencia, con los mismos extremos y mismo valor.

Listas de adyacencia

GrafoValorado.h





Cada elemento de la lista simplemente tendrá un puntero para evitar que haya tanta información repetida.

Son punteros inteligentes.

Estos son los objetos arista.

Grafo valorado

GrafoValorado.h

```
template <typename Valor>
class GrafoValorado {
public:
    GrafoValorado(int V);
    void ponArista(Arista<Valor> arista);
    int V() const;
    int A() const;
    AdysVal<Valor> const& ady(int v) const;
    std::vector<Arista<Valor>> aristas() const;
};
```

Recibe un objeto arista ya construido.

Devuelve el vector de aristas.

Útil para tratarlas por orden, menor a mayor valor.

Devuelve todas las aristas del grafo

Grafo valorado

GrafoValorado.h

```
void ponArista(Arista<Valor> arista) {
    int v = arista.uno(), w = arista.otro(v);
    if (v < 0 || v >= _V || w < 0 || w >= _V)
        throw std::invalid_argument("Vertice inexistente");
    ++_A;
    _ady[v].push_back(arista);
    _ady[w].push_back(arista);
}
```

Extrae los extremos

incrementa en 1 el número de aristas

Añade la arista en las dos listas de adyacentes.

Las dos aristas compartirán la información.

Grafo valorado

GrafoValorado.h

```
std::vector<Arista<Valor>> aristas() const {
    std::vector<Arista<Valor>> ars;
    for (int v = 0; v < V(); ++v)
        for (auto arista : ady(v))
            if (v < arista.otro(v))
                ars.push_back(arista);
    return ars;
}
```

Recorre las listas de adyacentes

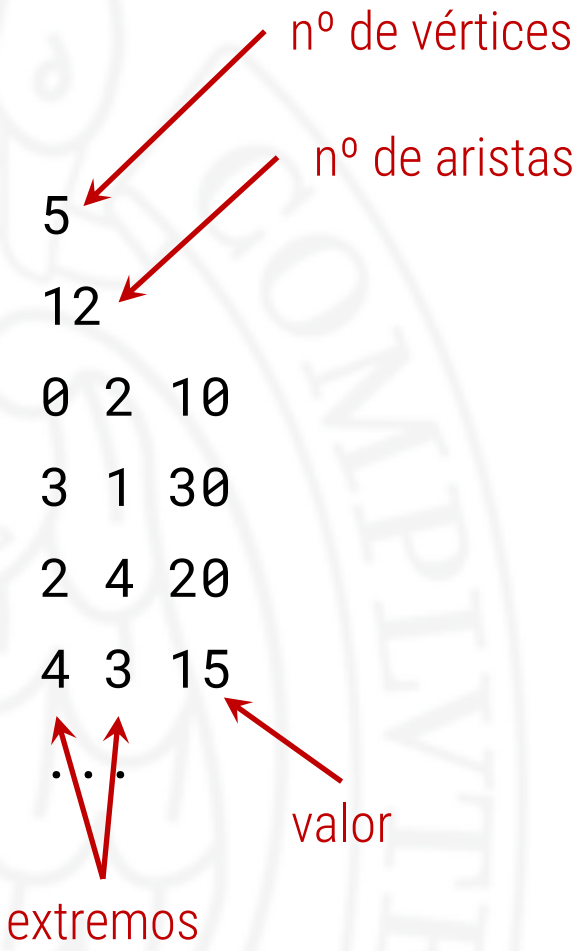
La añadimos la primera vez que la encontramos.

Añade las aristas a un vector que después se devuelve. Tener cuidado, las aristas aparecen 2 veces y debemos añadirlas una única vez

Con el ejemplo de la izquierda, el v<arista.otro(v) no implica que no se añade esta Arista, si no que se añadirá cuando el vértice que recorremos es el 0.

Construcción de un grafo valorado

```
bool resuelveCaso() {
    int V, A;
    cin >> V >> A;
    GrafoValorado<int> grafo(V);
    int u, v, valor;
    for (int i = 0; i < A; ++i) {
        cin >> u >> v >> valor;
        grafo.ponArista({u, v, valor});
    }
    ...
```



Recorrido en profundidad

```
// visita los nodos alcanzables desde v respetando el umbral
void dfs(GrafoValorado<int> const& G, int v, int umbral) {
    visit[v] = true;
    for (auto a : G.ady(v)) {
        if (a.valor() < umbral) {
            int w = a.otro(v);
            if (!visit[w])
                dfs(G, w, ancho);
        }
    }
}
```

Procesamos aristas en vez de vértices.

Aquí consideramos aristas que tenga asociado un valor menor que cierto umbral que dfs recibe como parametro.

$O(V+2A) = O(V+A)$