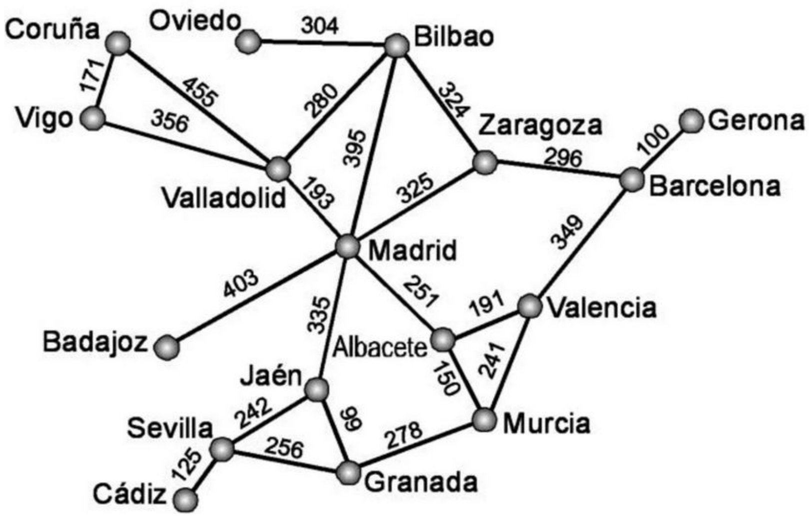
Universidad de Jaén

Estructuras de Datos

Grafos



Lección 16:

- Representación interna
- Recorridos sobre grafos

Parte I: Representación de un grafo

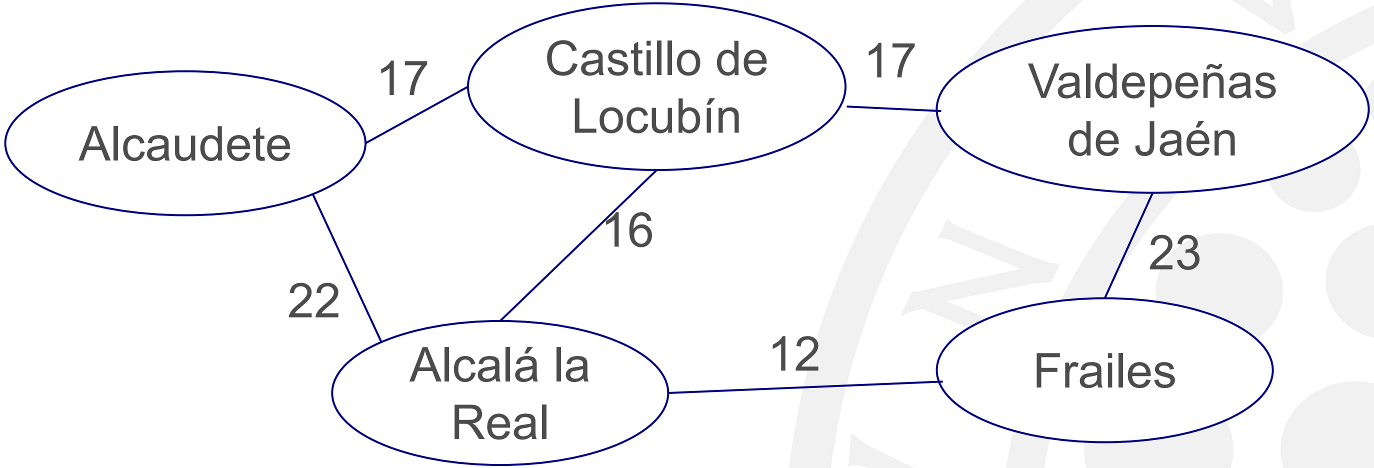
- Motivación
- Definiciones
- Representación interna

Motivación

- A un repartidor de paquetería a domicilio le han adjudicado el reparto en los pueblos de la Sierra Sur de Jaén.
- No siempre lleva paquetes a todos los pueblos, por lo que el itinerario puede variar
- ¿Cual es el recorrido que tendría que hacer para recorrer todos los pueblos?

Motivación

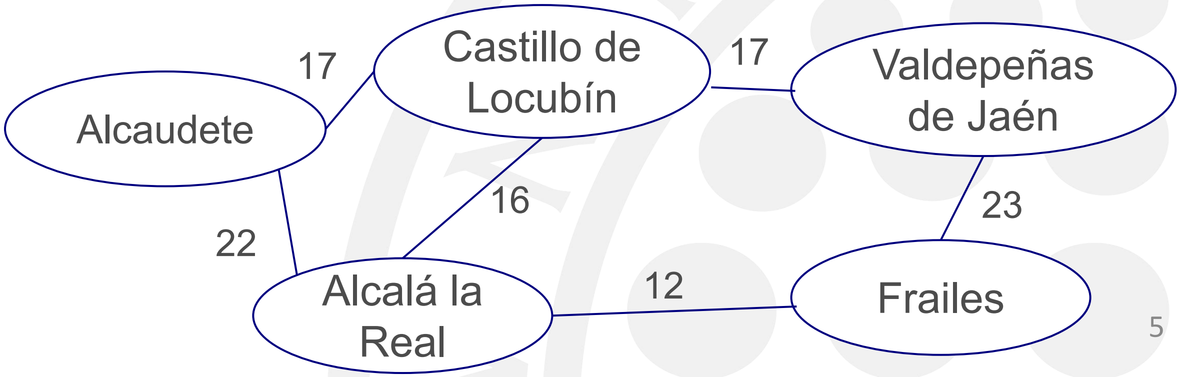
- No todos los pueblos tienen carreteras directas, por ejemplo para ir de Alcalá a Valdepeñas tiene que pasar por Frailes o el Castillo.
- Conoce los kilómetros existentes entre los distintos pueblos.
- Este problema es el típico que se soluciona mediante **grafos.**



Definiciones

- Un **grafo** es una representación abstracta de un conjunto de objetos entre los cuales puede haber parejas conectadas entre sí mediante enlaces
- Los objetos se representan mediante **vértices** o **nodos** y los enlaces mediante **ejes** o **aristas**
- El grafo G=(V,E);
  - V: conjunto de vértices
  - E: conjunto de ejes
  - El eje e=(u,v)
  - u, v son vértices

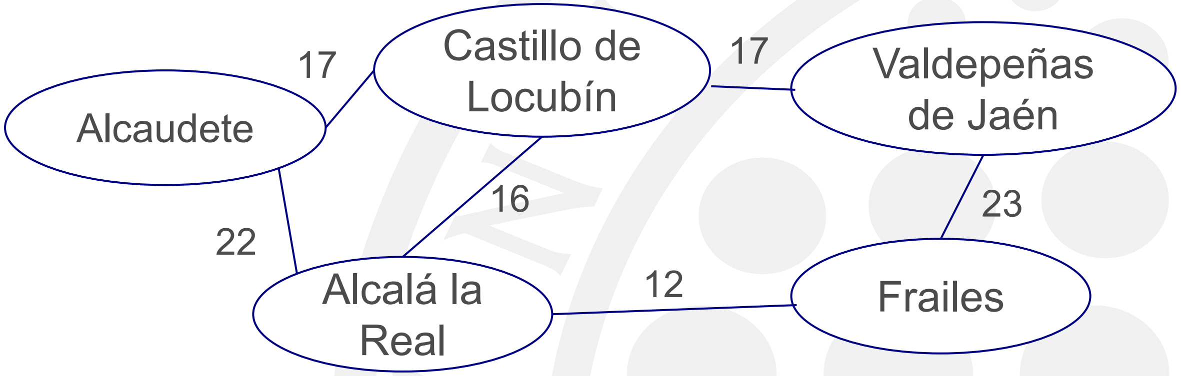
- nodos: pueblos
- ejes: conexión por carretera



Definiciones

- Un **camino** es una sucesión de vértices unidos mediante aristas
- La **longitud** de un camino es su número de aristas
- El **peso** de un camino en un grafo con pesos es la suma de los pesos de todas las aristas atravesadas.
- Un **grafo ponderado** asocia un valor o peso a cada arista en el grafo.

- El peso = kilómetros
- Ejemplo de camino: {Alc, Cas, Val, Fra}
- La longitud es 3



Definiciones

- En un **grafo dirigido** o **digrafo** cualquier eje *e=(u,v)* define un orden: si *e'=(v,u)*, e≠e'
- En un digrafo, en el eje *e=(u,v)*, *u* es el **origen,** y *v* el **destino** y se representa mediante una flecha.
- Esta flecha representa la dirección de un camino posible. En dirección contraria no es posible
- Un **grafo no dirigido** no impone ningún orden: *e=(u,v)* y *e'=(v,u)*, *e=e'*.

El grafo de las rutas de Jaén es no dirigido

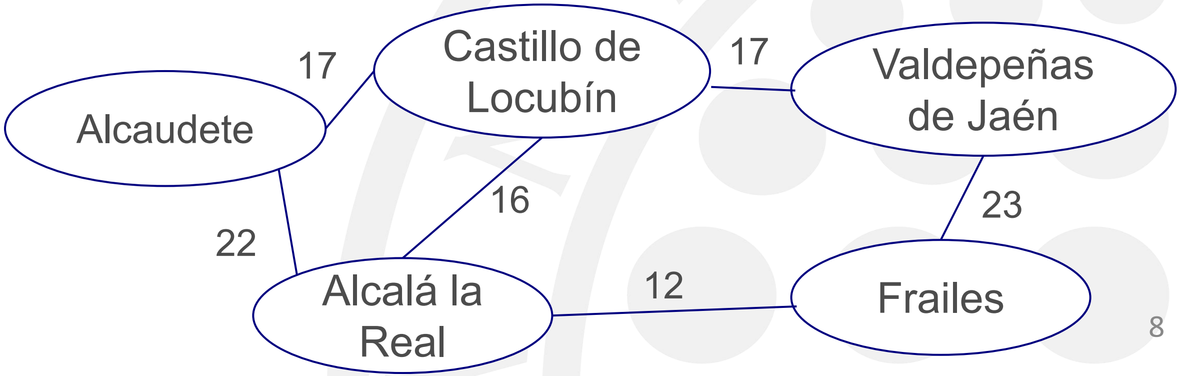
grafo dirigido

grafo no dirigido

Definiciones

- Los nodos de la arista *(u,v)* de un grafo no dirigido son **adyacentes** porque comparten arista.
- En la arista *e=(u,v)* de un grafo dirigido, **u es adyacente hacia v** y **v es adyacente desde u.**
- La arista *e=(u,v)* es **incidente** en los nodos u y v.
- En un grafo no dirigido, el **grado** de un vértice es el número de aristas incidentes.

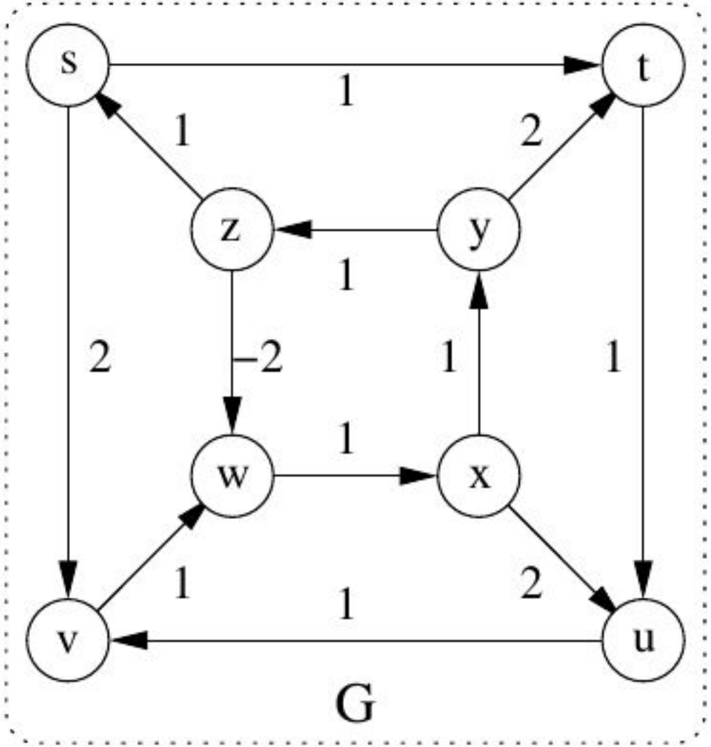
A Castillo de Locubín llegan tres carreteras, el grado del nodo es 3, Frailes tiene grado 2



Definiciones

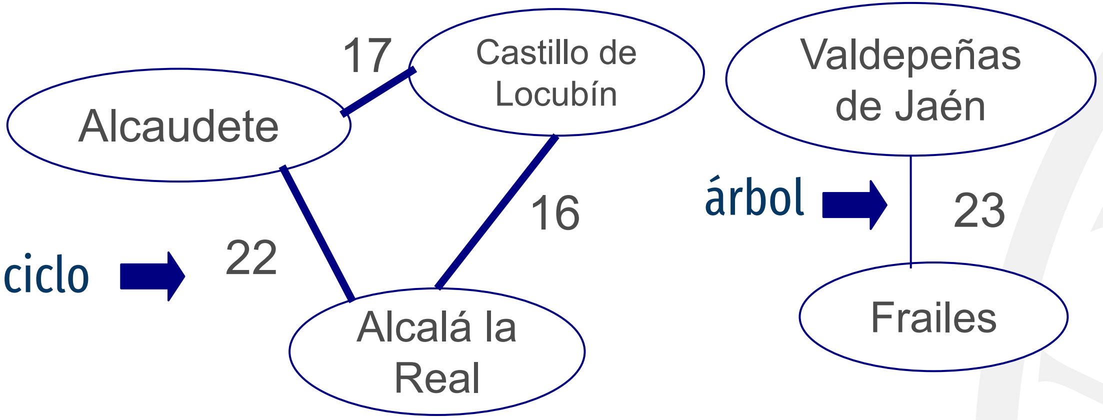
- En un grafo dirigido, el **grado de entrada** a un vértice *v* es el número de ejes que tienen a *v* como destino
- El **grado de salida** del vértice *v* es el número de ejes que tienen a *v* como origen

El grado de entrada es 2
y el de salida 1



Definiciones

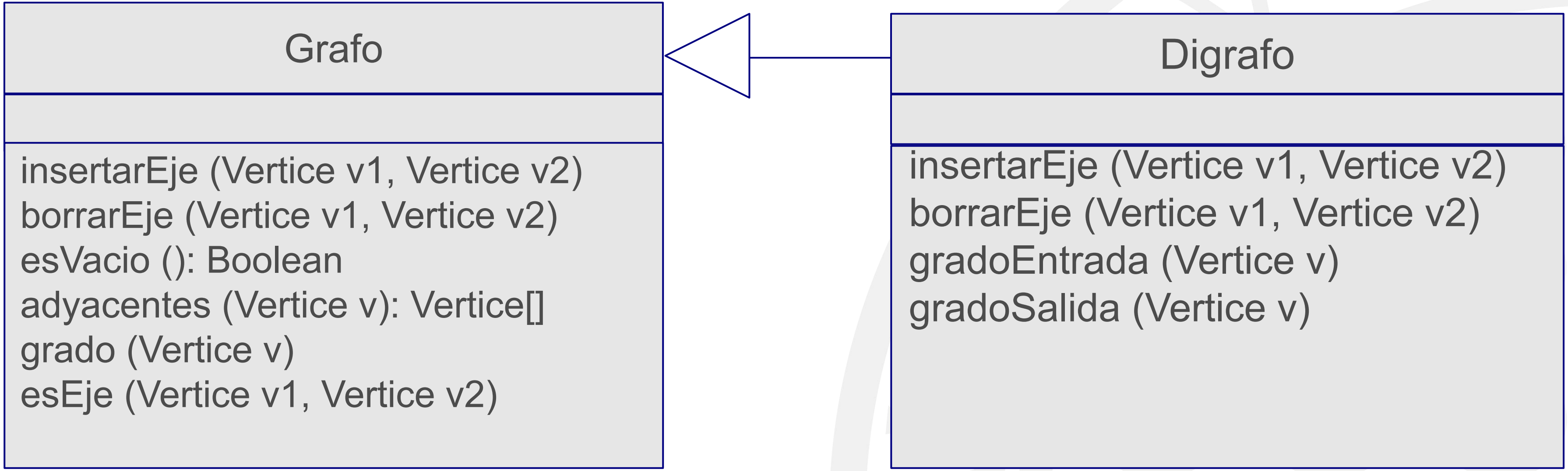
- Un **camino simple** es aquel que no repite vértices
- Un **ciclo** es un camino simple excepto porque el primer vértice y el último coinciden
- Un **grafo es conexo** cuando existe un camino entre cualquier pareja de vértices
- Un **árbol** es un grafo conexo y libre de ciclos



- Ciclo:{Al,AR,CL,Al}
- Las inundaciones han corta
dos carreteras y el grafo que
con dos componentes conexa

Representación interna

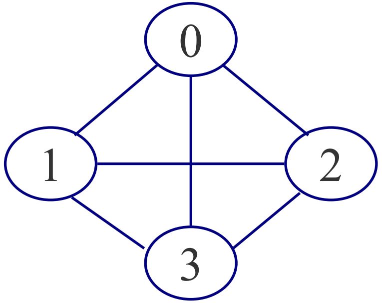
- Existen dos modos representaciones típicas de grafos: la **matrices de adyacencia** y las **listas de adyacencia.**
- Ambas representaciones son válidas para cualquier grafo, la mejor opción depende del caso.



Matriz de adyacencia

- La matriz de adyacencia del grafo *G=(V,E)* con n vértices es un array bidimensional *matAd(nxn)* de modo que el eje $e=(v_i,v_j)$ está representado en *matAd[i,j]=1*
- Si no existe un eje que conecte v_i con v_j entonces matAd[i,j]=0
- Si el grafo es no dirigido, entonces la matriz es simétrica, *matAd[i,j]=matAd[j,i]*, $\forall$ *i,j*
- Si el grafo es dirigido, *matAd[i,j]≠matAd[j,i]*, $\forall$ i,j
- Si el grafo es ponderado, los ejes *matAd[i,j]≠0* (puede tener cualquier valor positivo)

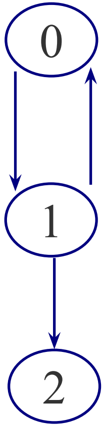
Matriz de adyacencia



matAd(1,3)=
=matAd(3,1)=1

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

G1



matAd(0,2)=0
matAd(0,1)=1

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

G2

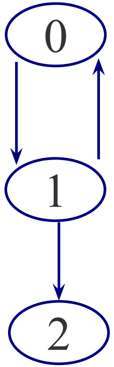
Matriz de adyacencia

- El grado del vértice v_i (no digrafo) se obtiene sumando la fila (o columna) :

$$\sum_{j=0}^{n-1} adj_{mat}[i][j]$$

- En un digrafo, el grado de salida de un vértice v_i se obtiene sumando la fila, y el grado de entrada se obtiene sumando la columna.

gradoE(vi)=$\sum_{j=0}^{n-1} A[j,i]$ gradoS(vi)=$\sum_{j=0}^{n-1} A[j,i]$



$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

gradoS(v1)=2
gradoE(v1)=1
gradoE(v0)=1

Implementación: Matriz de adyacencia

```
typedef int Vertice;
class Grafo{
protected:
   int n;
   vector<vector <int> > matAd;
public:
    Grafo(int tam);
    void insertarEje(Vertice v1, Vertice v2, int peso = 1);
    void borrarEje(Vertice v1, Vertice v2);
    bool esVacio();
    int grado(Vertice v);
    vector<int> adyacentes(Vertice v);
    int operator() (Vertice x, Vertice y) { return matAd[x][y];}
    bool esEje(Vertice v1, Vertice v2);
};
```

- grado() queda inaccessible,
- la inserción y borrado hay que redefinirlo

```
class Digrafo: public Grafo{
private:
    int grado(Vertice v);
public:
    Digrafo(int tam): Grafo(tam){}
    void insertarEje(Vertice v1, Vertice v2, int peso = 1);
    void borrarEje(Vertice v1, Vertice v2);
    int gradoEntrada(Vertice v);
    int gradoSalida(Vertice v);
};
```

Implementación: Matriz de adyacencia

```
void Grafo::insertarEje(Vertice v1, Vertice v2, int peso){
    if (v1 < 0 || v1 >= n) throw ErrorVerticeNoExiste();
    if (v2 < 0 || v2 >= n) throw ErrorVerticeNoExiste();
    matAd[v1][v2] = peso;
    matAd[v2][v1] = peso;
}
int Grafo::grado(Vertice v){
    if (v < 0 || v >= n) throw ErrorVerticeNoExiste();
    int sum=0;
    for (int j = 0; j < n; j++)
        sum += matAd[v][j];
    return sum;
}
```

inserta en posiciones simétricas

recorre filas

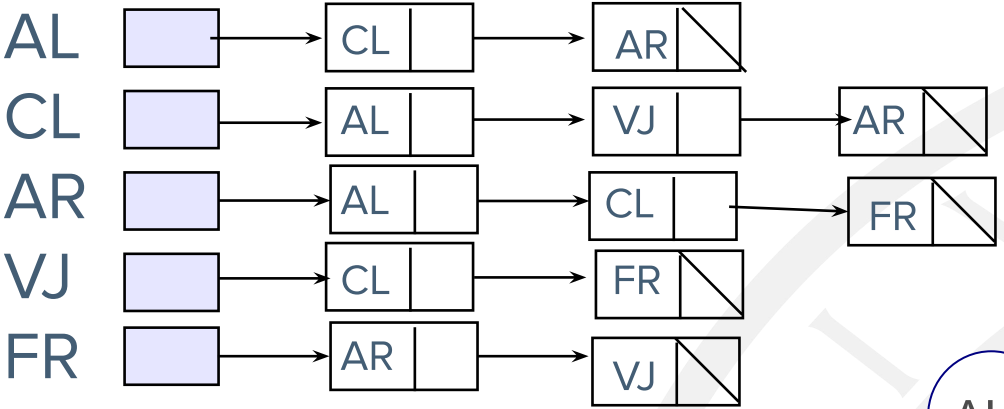
```
void Digrafo::insertarEje(Vertice v1, Vertice v2, int peso){
    if (v1 < 0 || v1 >= n) throw ErrorVerticeNoExiste();
    if (v2 < 0 || v2 >= n) throw ErrorVerticeNoExiste();
    matAd[v1][v2] = peso;
}
int Digrafo::gradoEntrada(Vertice v){
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += matAd[i][v];
    return sum;
}
```

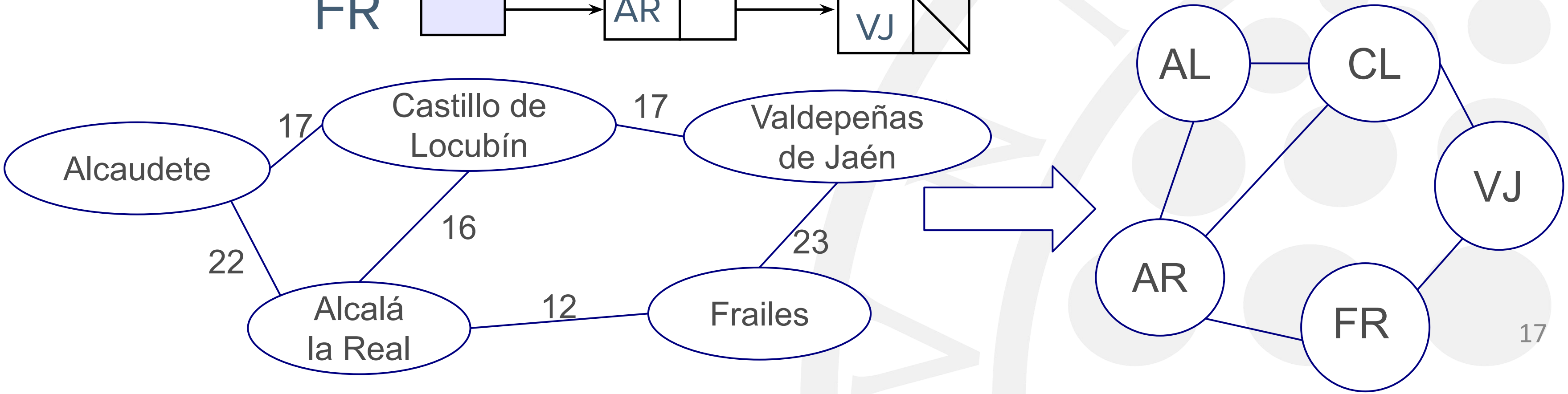
inserta en una posición

recorre columnas

Lista de adyacencia

- La lista de adyacencia genera, para cada vértice *v*, una lista con los vértices que son adyacentes a *v*





Lista de adyacencia

- Si el grafo es no dirigido una inserción implica dos entradas, en el digrafo sólo una.
- La lista de adyacencia resulta más compacta y necesita poco espacio para grafos poco densos.
- Insertar un nuevo eje es *O(1)*, (precondición: no se comprueban duplicados).
- Grado de un vértice en grafos no dirigidos: el número de elementos de su lista de adyacencia.
- Conocer el número de ejes del grafo: *O(n+ne)*, *ne:#ejes*.
- Ejes de salida en un digrafo: *O(k)*, el tamaño de la lista de adyacencia.
- Ejes de entrada: *O(n+ne)*, hay que recorrer toda la eedd.

Implementación: Lista de adyacencia

- Código del grafo no dirigido
  - Para el digrafo se siguen las pautas sugeridas con la matriz de adyacencia

```
class GrafoLa {
    vector <list<int> > listAd;
    int n;
public:
    GrafoLa(int nn): n(nn), listAd(n){}
    void insertarEje(Vertice v1, Vertice v2, int peso);
    ...
};
```

Precondición:
No existe e=(v1,v2)

```
void GrafoLa::insertarEje(Vertice v1, Vertice v2, int peso){
    if (v1 <0 || v1>= n) throw ErrorVerticeNoExiste();
    if (v2 <0 || v2>= n) throw ErrorVerticeNoExiste();
    listAd[v1].push_back(v2);
    listAd[v2].push_back(v1);
}
```

Parte II: Recorridos sobre grafos

- Recorrido en profundidad
- Recorrido en anchura

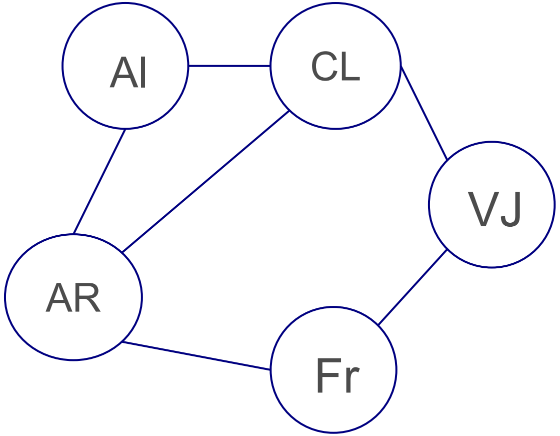
Recorridos en grafos

¿Cómo encuentra un navegador GPS una ruta? ¿Cómo se puede encontrar el conductor una ruta que le lleve a todos los pueblos?

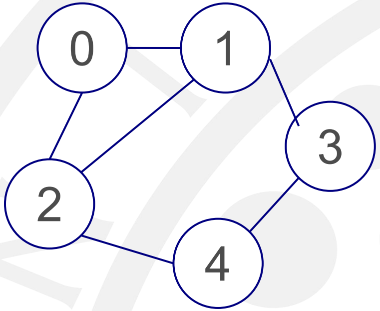
- Existen dos recorridos típicos de los nodos de un grafos:
  - recorrido en profundidad (**DFS**: Depth First Search)
  - recorrido en anchura (**BFS**: Breadth First Search)
- El recorrido de un grafo implica:
  - visitar todos los nodos para numerarlos o procesarlos
  - sólo hay que indicar el nodo de comienzo

Recorrido en profundidad

- Es una versión generalizada del recorrido en preorden de un árbol
- Funcionamiento:
  a. Se comienza por un vértice cualquiera (o sugerido)
  b. Se elige el primer vértice adyacente (de salida en un digrafo) no visitado. Si no quedan volver al vértice anterior
  c. Se repite el proceso anterior hasta el final del camino

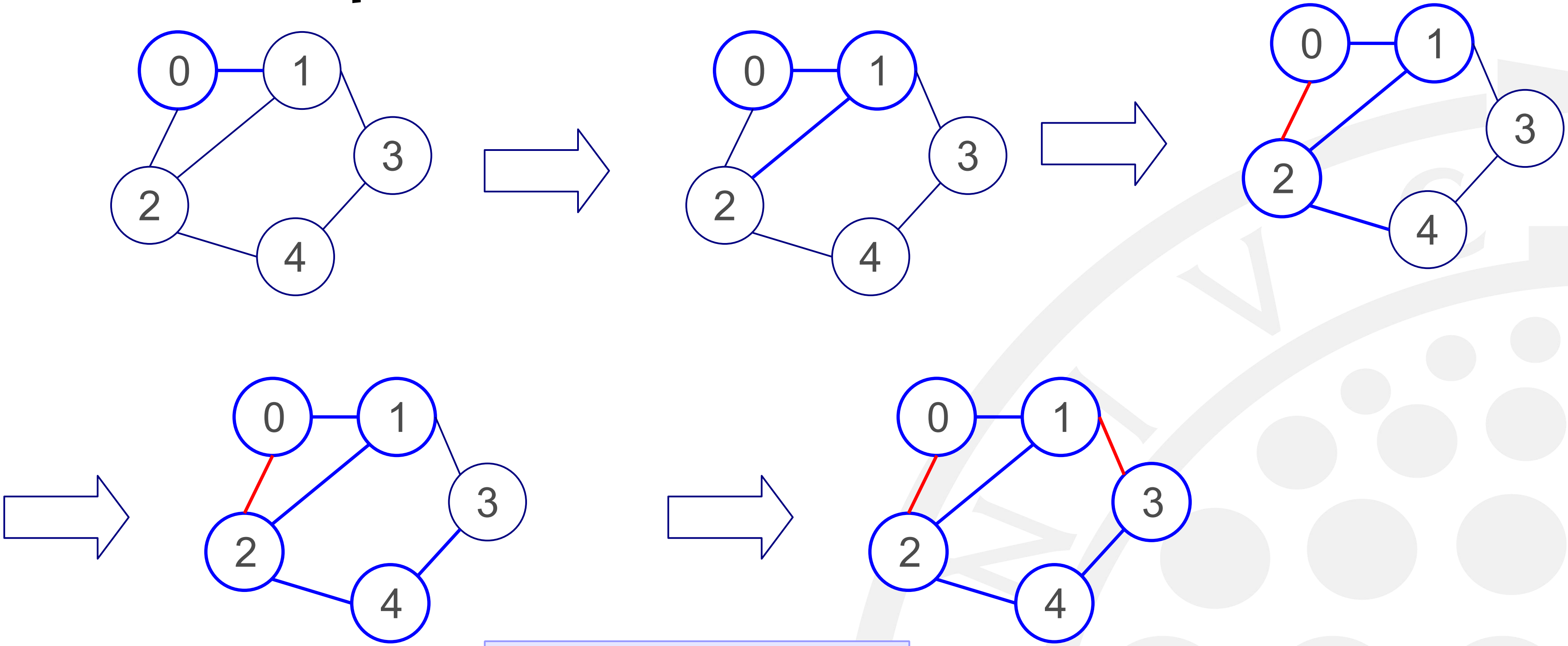


Al: 0
CL: 1
AR: 2
VJ: 3
Fr: 4



Solución: 0, 1, 2, 4, 3

Recorrido en profundidad



Solución: 0,1,2,4,3

Recorrido en profundidad

- Se etiqueta al inicio cada nodo como blanco, y gris cuando ya ha sido visitado (o usar booleanos)
- Se añade el vector de estado y funciones privadas
- Implementamos el método sobre matrices de ady.

```
enum EstadoVertice {blanco, gris};
class Grafo{
    protected:
         ...
       vector<estadoVertice> estado;
    private:
       void iniDFS();
       void DFSrec(Vertice v, vector<int> &r);
    public:
         ...
        vector<Vertice> DFS(Vertice v);
};
```

Funciones privadas:
- inicializar a blancos
- proceso recursivo

Recorrido en profundidad

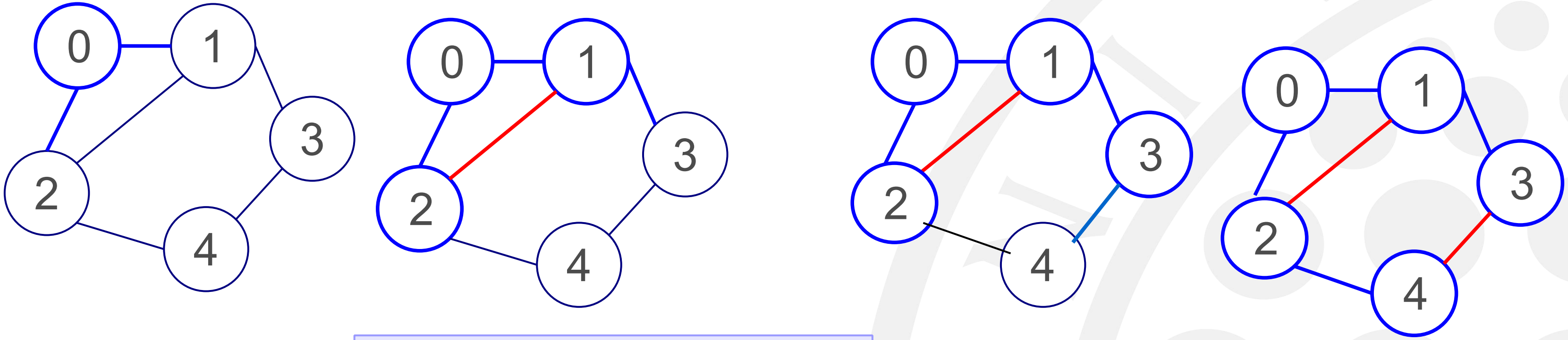
```
void Grafo::iniDFS(){
     for (int i = 0; i < n; i++)
     estado.push_back(blanco);
}
void Grafo::DFSrec(Vertice v, vector<Vertice> &r) {
     estado[v] = gris;
     r.push_back(v);
     for (int j = 0; j < n; j++)
          if (esEje(v, j) && estado[j] == blanco)
               DFSrec(j, r);
}
vector<Vertice> Grafo::DFS(Vertice v){
     vector<Vertice> recorrido;
     iniDFS();
     DFSrec(v, recorrido);
     return recorrido;
}
```

inicializar a blanco

vector del recorrido

Recorrido en anchura

- Es una versión generalizada del recorrido por niveles de un árbol
- Funcionamiento:
  1. Primero se visitan los vecinos adyacentes al nodo
  2. En la siguiente iteración se visitan a los adyacentes de estos adyacentes, y así hasta finalizar los recorridos de búsqueda generados
  3. Suele trabajarse con una versión no recursiva



Resultado: 0, 1, 2, 3, 4

Recorrido en anchura

```
vector<Vertice> Grafo::BFS (Vertice v){
    deque<Vertice> q;
    vector<bool> visitados(n,false);
    vector<Vertice> recorrido;
    q.push_back(v);
    visitados[v]=true;
     recorrido.push_back(v);
    while (!q.empty()){
        Vertice k = q.front();
        q.pop_front();
        for (int i=0; i<n; ++i)
            if (esEje(k, i) && !visitados[i]) {
                q.push_back(i);
                visitados[i] = true;
                recorrido.push_back(i);
            }
    }
    return recorrido;
}
```

- q: cola de vértices sin procesar
- visitados: true si visitado
- recorrido: devuelve el recorrido

Resultado: 0, 1, 2, 3, 4

Conclusiones

- Los grafos tienen numerosas aplicaciones en muchos campos: planificación de trayectorias, teoría de circuitos, juegos de estrategia, etc.
- Hemos definido algunos conceptos y definido la estructura de datos que manejan.
- En esta lección sólo hemos estudiado métodos para su recorrido exhaustivo.
- Los métodos del cálculo de caminos mínimos será objeto de estudio en Diseño de Algoritmos.

De ahora en adelante…

- Los grafos son una estructura de datos que se utiliza para resolver problemas concretos de caminos.
- Para trabajar con posiciones, como las coordenadas de un GPS, utilizaremos las denominadas estructuras de datos espaciales.

Lección 17: Mallas regulares y quadtrees