Teoría de Grafos

Herramientas de programación para procesamiento de señales

Indice

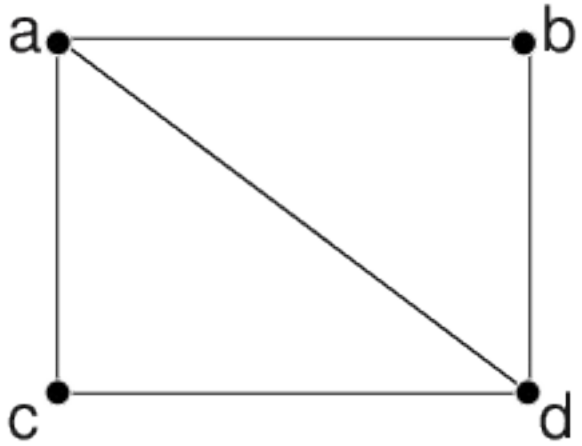
- Nociones básicas:
  - Definiciones
  - Ejemplos
  - Propiedades
- Nociones avanzadas:
  - Grafos planares
  - Árboles
  - Representación en computadora
- Algoritmos clásicos:
  - Búsqueda en profundidad
  - Búsqueda en anchura
  - Prim y Kruskal
- Algoritmos de ordenamiento:
  - Mergesort
  - Heapsort
  - Bubblesort
- Estructuras de datos:
  - Abstractas:
    - Lists
    - Maps
    - Queues
    - Arboles
  - Concretas:
    - Heap
- Complejidad
  - Estimación
  - NP-hard
- Problemas clásicos
  - Árboles generadores
  - Del viajante
  - De Euler
  - Caminos mínimos

Contenido

- Conceptos básicos
- Conceptos Avanzados
- Problemas Clásicos

Definiciones (1)

- Definición:
  - Conjunto no vacío V: vértices
  - Conjunto E: aristas
- Representación gráfica
- Definiciones:

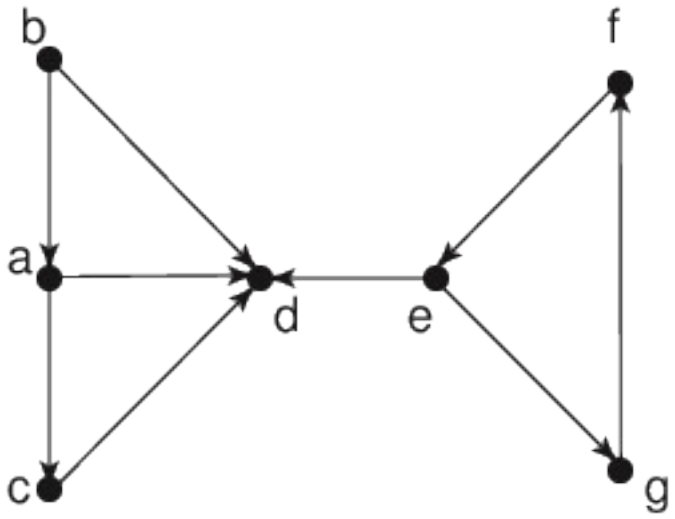


  - Vértices adyacentes: existe **e**, tal que **e=(a,b)**
  - **a** y **e** son incidentes
  - Vértices independientes: no existe **e**
  - Aristas adyacentes/independientes: vértice en común
  - Grado de un vértice: no. de aristas indicentes en **a**
  - Orden: |V|, grado de conexión |E|
  - No se permiten “aristas paralelas” o “autoconexiones”

Definiciones (2)

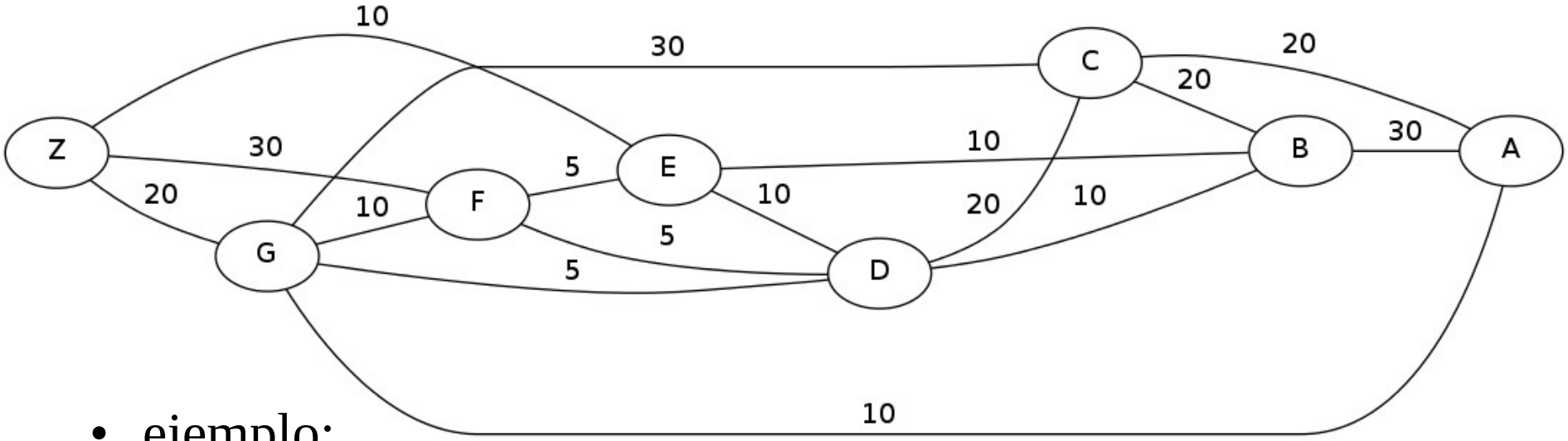
- Grafo dirigido:
  - par ordenado (a,b)!=(b,a)

- Ejemplos:
  - Calles (dirigido)
  - Red de computadoras (no-dirigido)



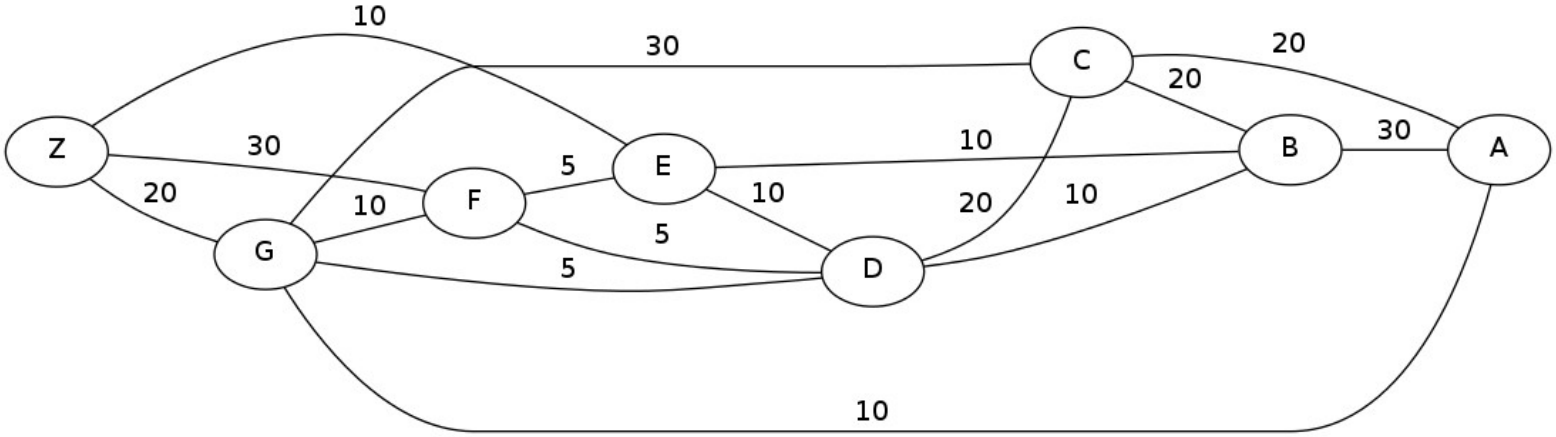
Definiciones (3)

- Grafo ponderado:
  - Valor asignado a rama: w(e)
  - O a un vértice: u(v)
  - Estático o dinámico

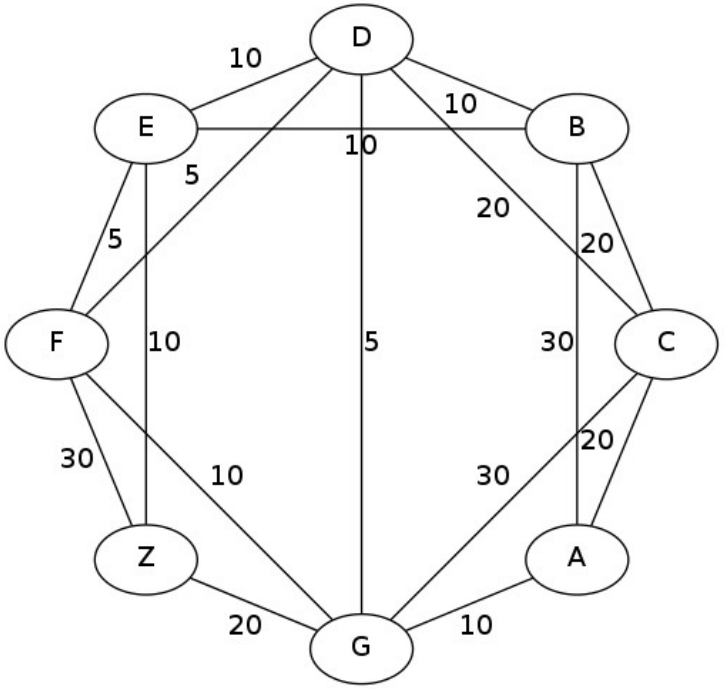


- ejemplo:
  - Distancias entre ciudades
  - Redes de computadoras (retardo)

Definiciones (4)

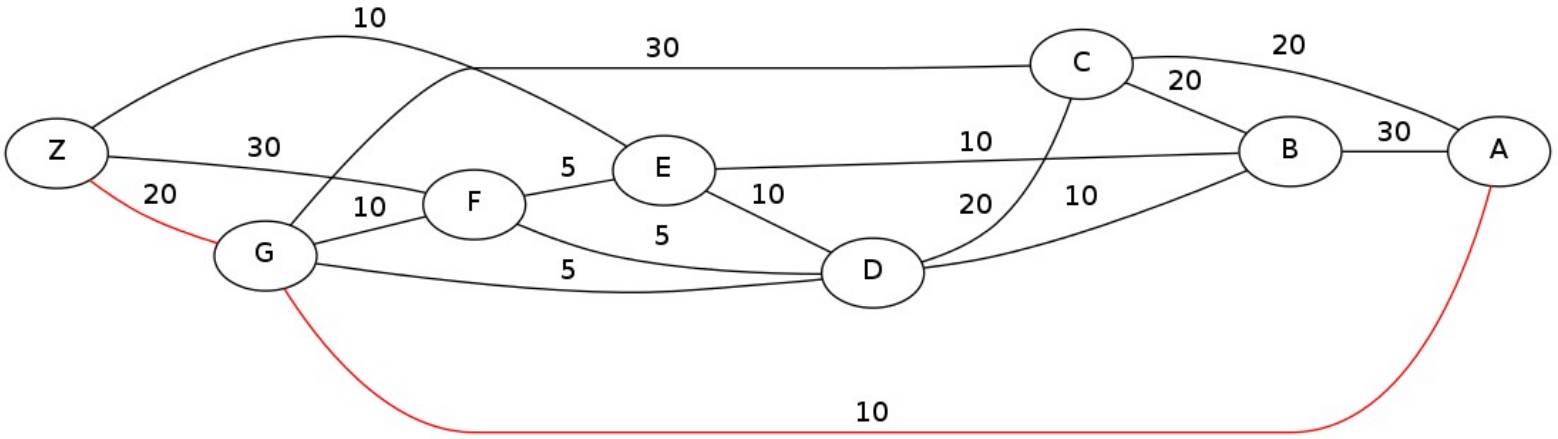


- Recordatorio:
  - Los grafos son abstractos
- Isomorfismo:
  - Relación 1 a 1 entre vértices



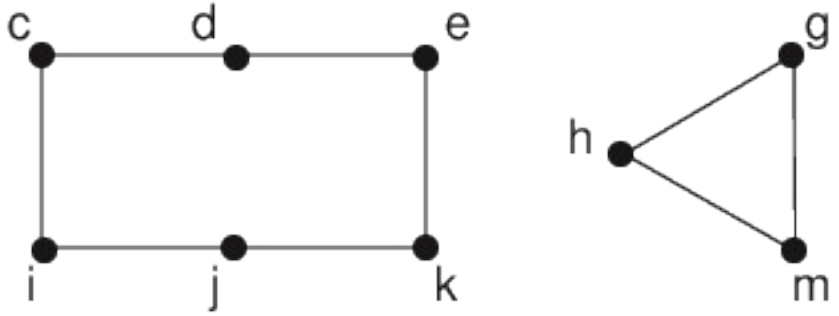
Definiciones (5)

- Camino:
  - Secuencia: $v_1 ... v_n$
  - v_i y v_{i+1} adyacentes
  - Cada nodo aparece una sola vez
- Ciclo:
  - Camino cerrado



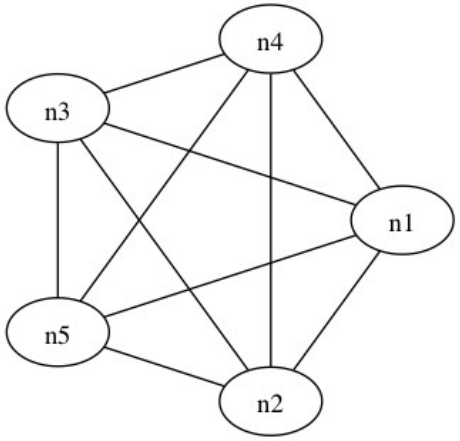
Definiciones (6)

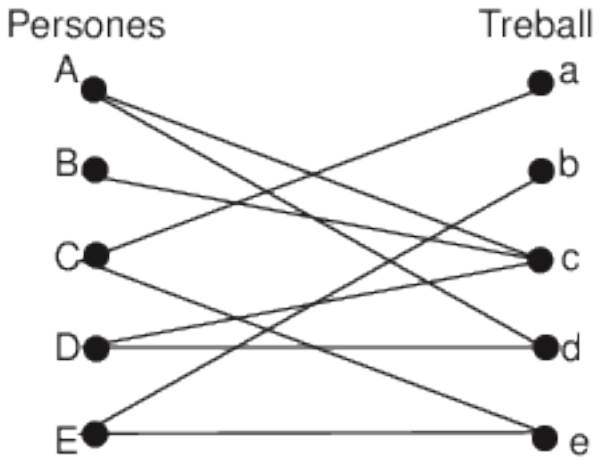
- Grafo conexos:
  - Para todo: $v_1 ... v_n$ existe un camino que los une
- Subgrafo:
  - Subconjunto de vértices y aristas
- Componente conexo:
  - Subgrafo conexo

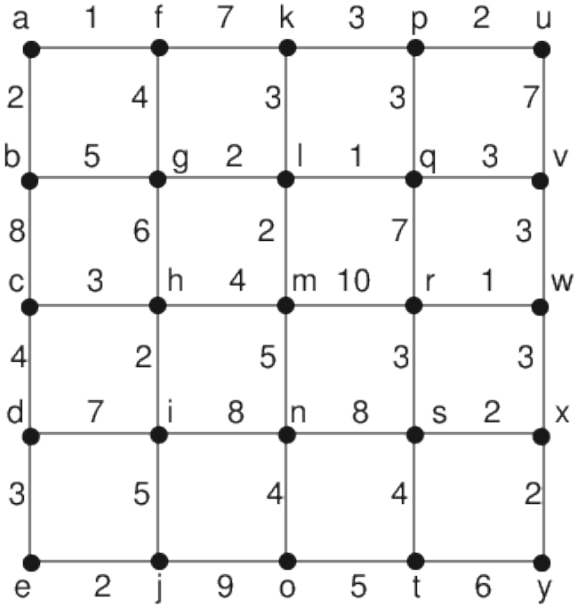


Definiciones (7)

- Tipos de grafos:
  - Completo: |E|=n(n-1)/2
  - Bipartito (completo): |E|=n
  - Planares: cota superior





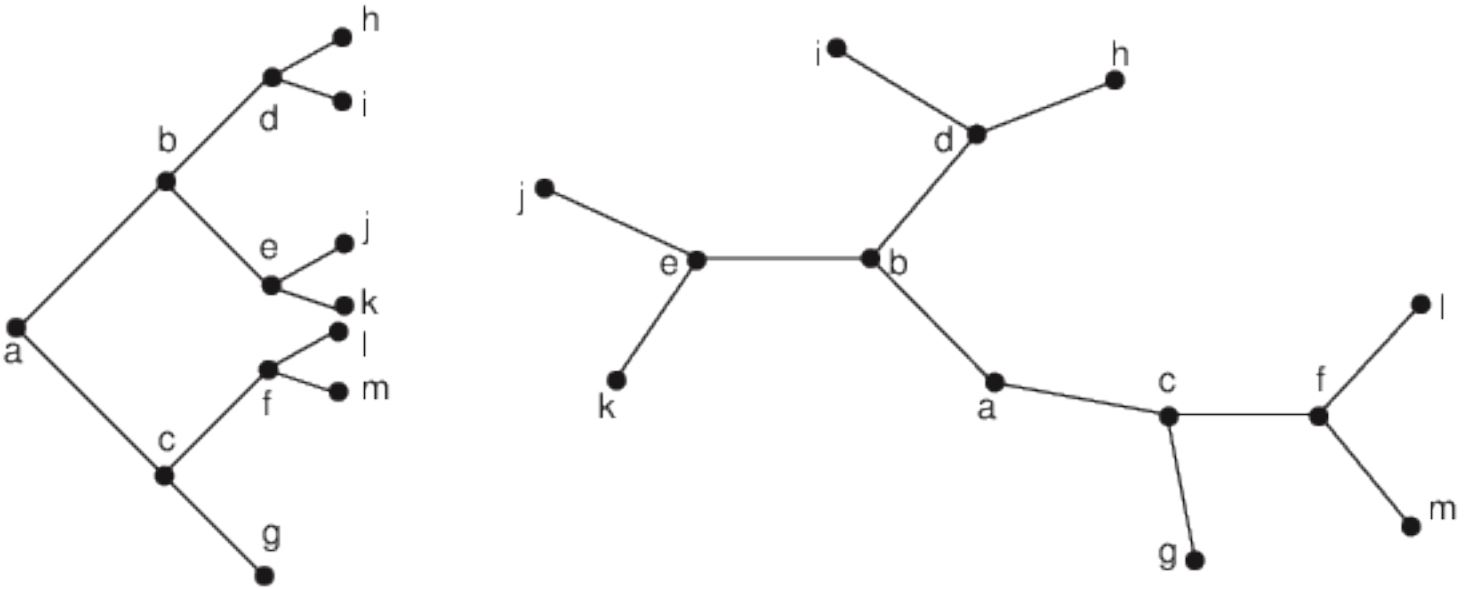


Contenido

- Conceptos básicos
- Conceptos Avanzados
- Problemas Clásicos

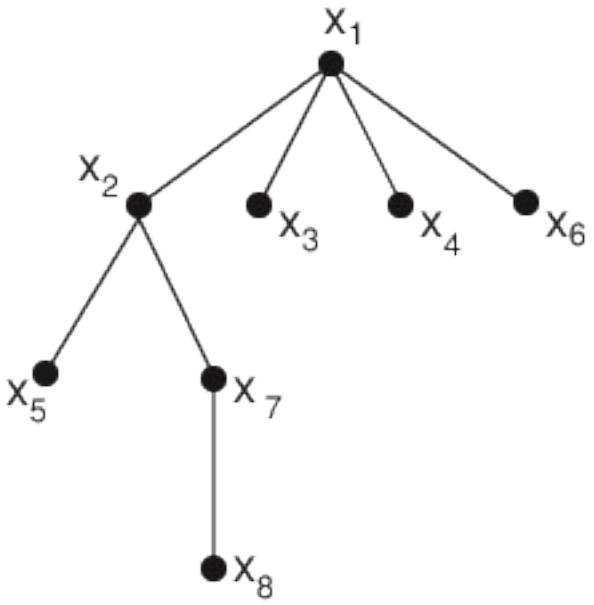
Árboles (1)

- Grafo conexo sin ciclos:
  - N nodos, N-1 aristas
  - Raíz: cualquiera

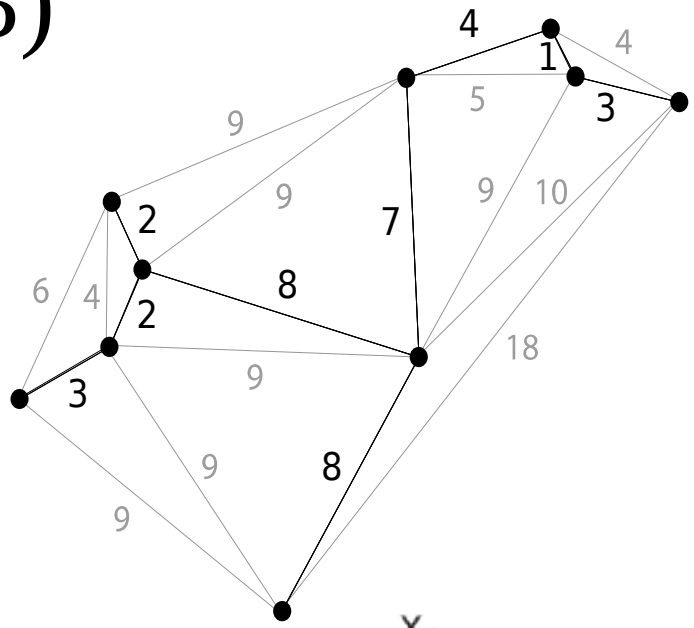


Árboles (2)

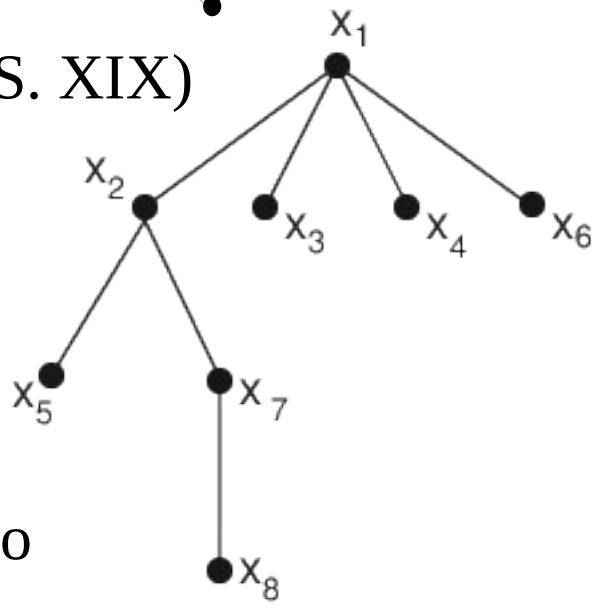
- dirigido:
  - Cada nodo tiene un único padre
  - Raíz: única



Árboles (3)

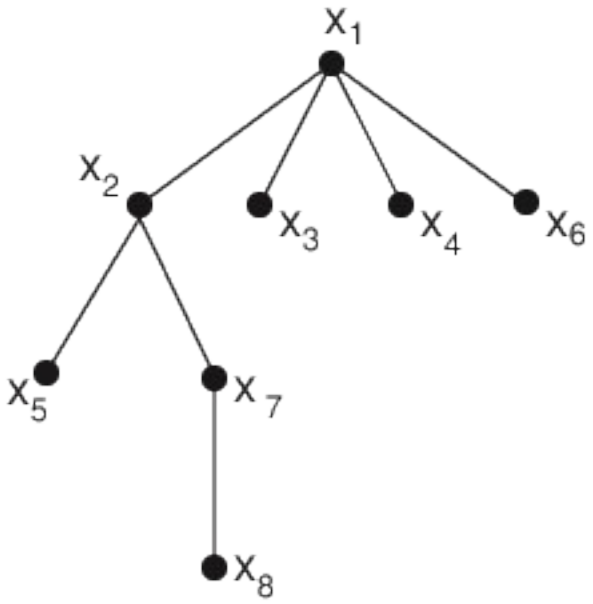


- Algoritmos básicos:
  - BFS:
  - DFS:
- Objetivo:
  - recorrer todos los nodos
  - Buscar uno en particular
- Motivación:
  - resolución de laberintos (C. Trémaux, S. XIX)
- Usos:
  - recorrer Árboles
  - Recorrer grafos
  - Árbol generador
- Derivados:
  - Prim y Kruskal: árbol generador mínimo



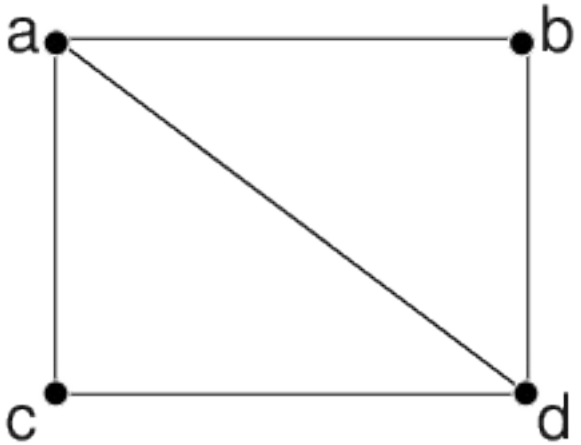
Árboles (4)

- Estado: conjunto de nodos “procesados”
- Ejecución:
  - DFS: 5, 8, 7, 2, 3, 4, 6, 1
  - BFS: 1, 2, 3, 4, 6, 5, 7, 8
- Complejidad: |E|
- Peor caso, denso: |E|~|V|^2



Representación en computadora (1)

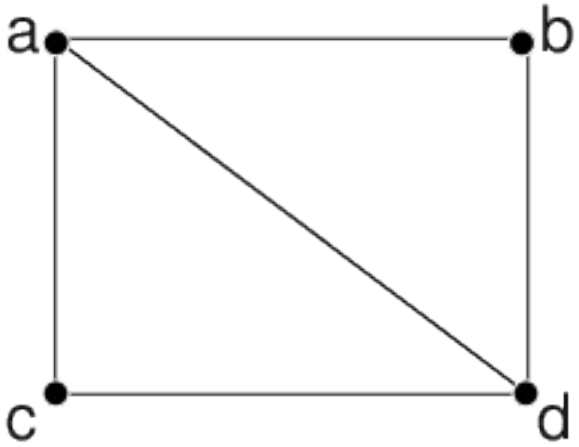
- Matriz de adyacencia:
  - Convención 1/0 conectado/no conectado
  - Simple
  - Útil para grafos densos
- Dirigido: arbitraria
- No dirigido: simétrica
- Almacenamiento: n^2



	a	b	c	d
a	0	1	1	1
b	1	0	0	1
c	1	0	0	1
d	1	1	1	0

Representación en computadora (2)

- Ejemplo:
  - Cálculo de grado

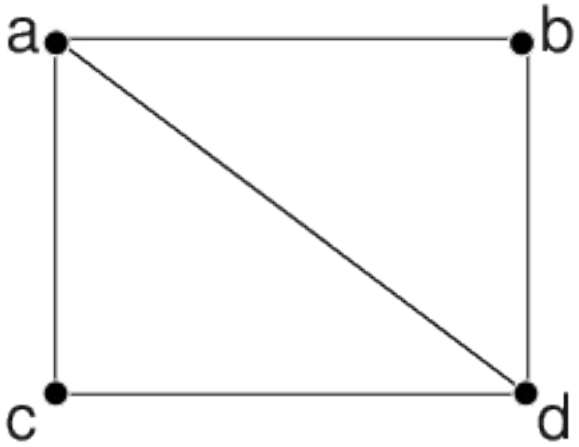


	a	b	c	d
a	0	1	1	1
b	1	0	0	1
c	1	0	0	1
d	1	1	1	0

=2

Representación en computadora (3)

- Lista de adyacencia
  - Cada nodo tiene su lista de vecinos
    - Almacenamiento: k.n (promedio)
    - Más complejo
    - Eficiente para grafos “ralos”



Abstracta

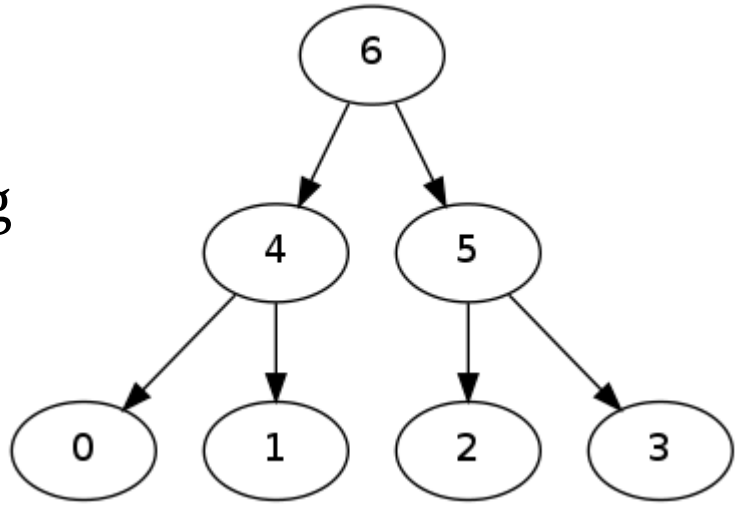
Vertice	Lista de adyacencia
a	3: b,c,d
b	2: a,d
c	2: a,d
d	3: a,b,c

Implementación

- Linked list:
  - eficiente memoria
  - Engorroso programación
- Array
  - Ineficiente memoria
  - Fácil programación
- C++: std::vector

Representación en computadora (4)

- Árbol:
  - Mínimo: parent
  - Comodidad: first_child, next_sibiling

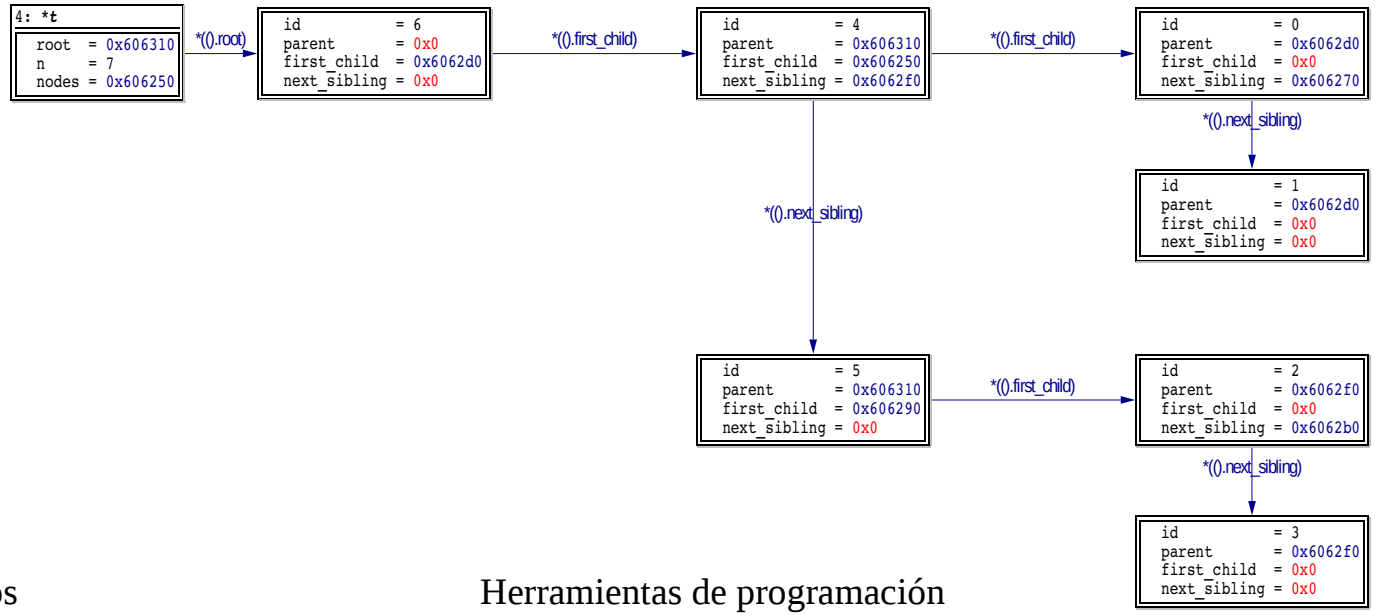


tree

```
int n
node* root
node[] nodes
```

node

```
int   id
node* parent
node* first_child
node* next_sibling
```

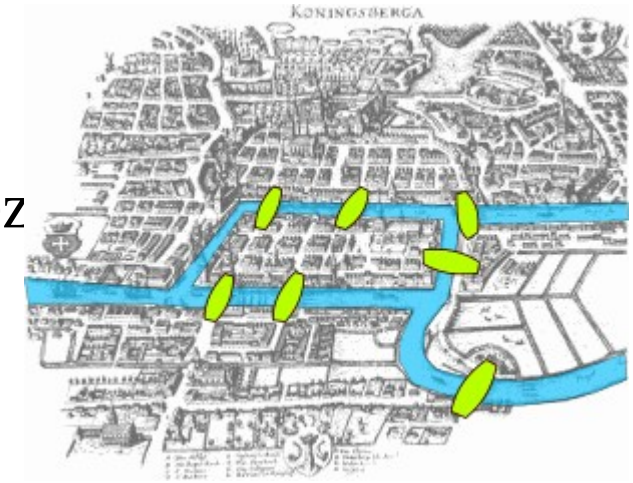


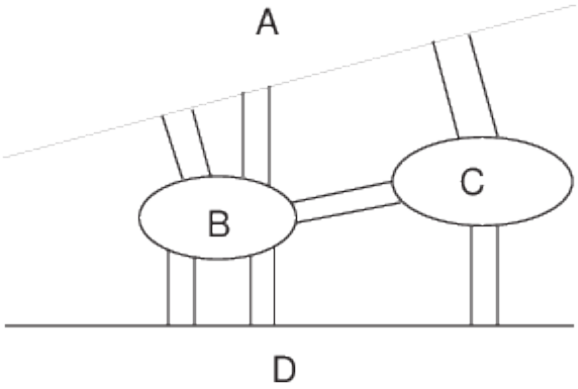
Contenido

- Conceptos básicos
- Conceptos Avanzados
- Problemas Clásicos

Problemas clásicos (1)

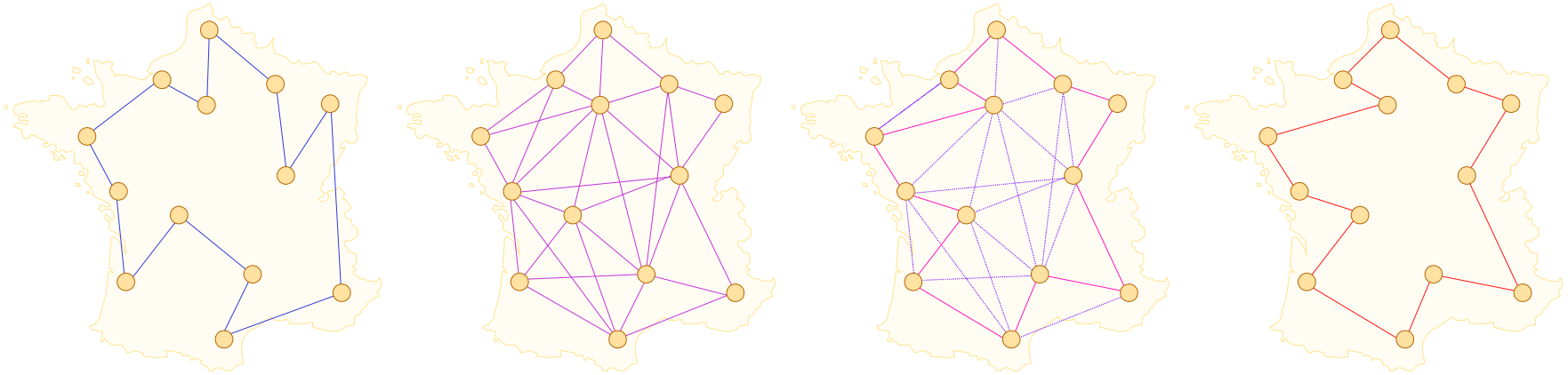
- Recorrido:
  - Secuencia de nodos
    - Cada nodo puede estar más de una vez
    - Cada rama aparece sólo una vez
- Circuito: recorrido cerrado
- Circuito euleriano
    - Todas las ramas de un grafo
    - Visita los nodos al menos una vez
- Los 7 puentes de Könisberg
  - Euler 1735
- Condición necesaria:
  - “Cada vez que entramos, salimos”
  - Grado par o nulo





Problemas clásicos (2)

- Circuito Hamiltoneano
  - Visita los nodos exactamente una vez
- Problema del viajante:
  - Travelling Salesman Problem (TSP): 1930
- NP-hard: grafo completo n vértices (n-1)! ciclos

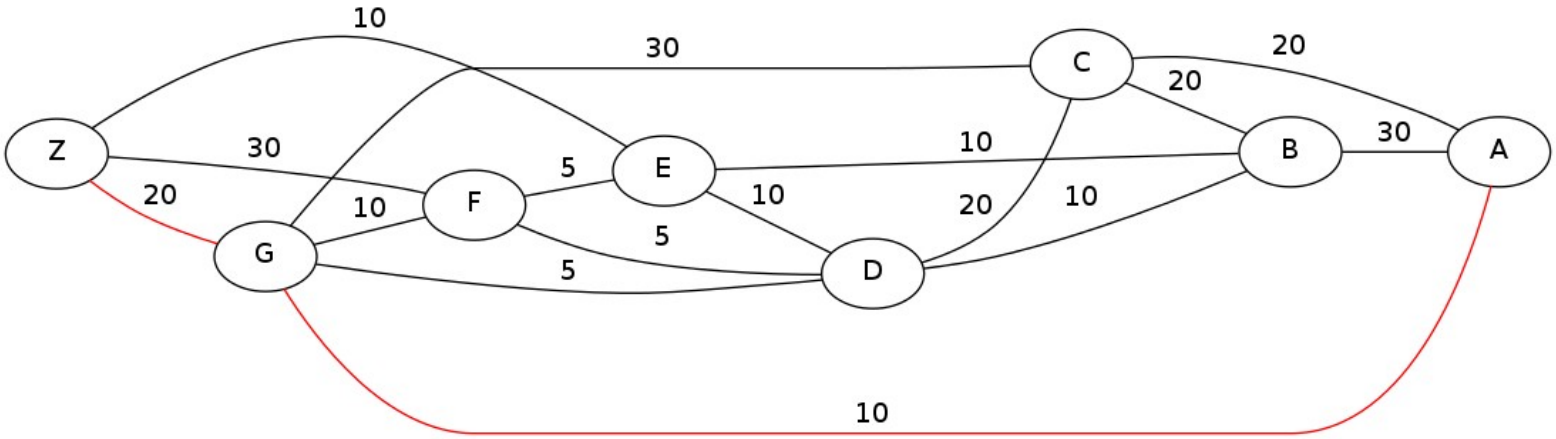


1 **2** **3** **4**

Mapa de ciudades importantes de Francia

Problemas clásicos (3)

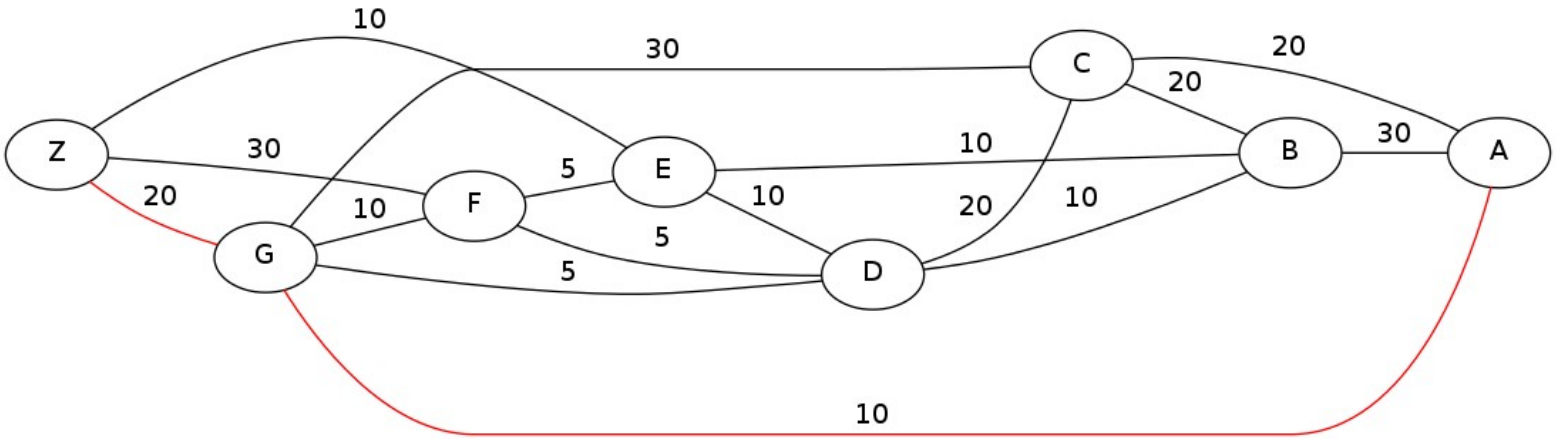
- Camínos mínimos
  - Camino de costo mínimo entre dos nodos
- E. Dijkstra 1959



Algoritmo de Dijsktra (1)

Ingredientes:

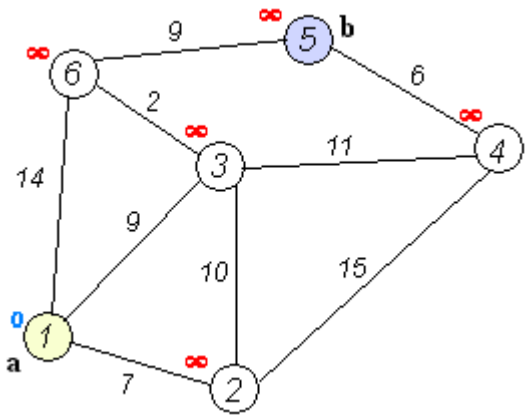
- Conjunto **T** de nodos para los cuales ya se calculó la distancia
- Lista **dist(v)** de distancias de cada nodo **v** al nodo inicial **a**
- Lista **prev(v)** de nodos previos: **prev(v)** es el nodo anterior a **v** en el camino (mínimo) que va de **a** a **v**



Algoritmo de Dijsktra (2)

Inicialización:

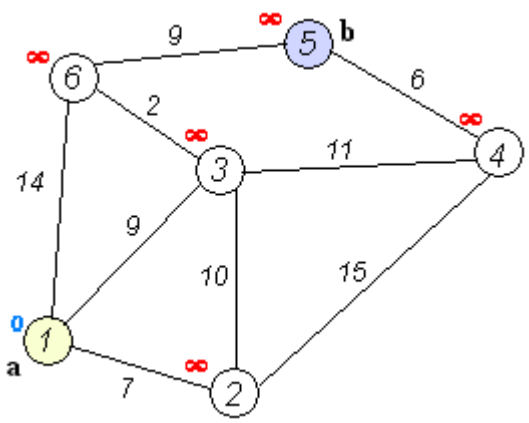
- **T** vacío
- **dist(v)=inf,** para todo **v!=a**
- **dist(a)=0**
- **prev(v)=NULL,** para todo **v**
- Nodo actual **u=a**



Iteración:

- Sea u de V-T tal que u=argmin(dist(v))
- Agrego u a T
- Sean t los vecinos de v:
  - Ver si u es un atajo para llegar a t: d=dist(u)+w(u,t)<dist(t)
  - Si es cierto, actualizo dist(t)=d, y prev(t)=u

Algoritmo de Dijsktra (3)



Propiedades:

- greedy
- Óptimo global
- Complejidad:
  - Ejecución:
    - Array: O(|V|^2+|E|) = O(|V|^2)
    - Heap: O((|E|+|V|) log |V|)
      - Óptimo: O(|V| log |V|), si |E|~|V|
      - Peor caso si es denso: 0(|V|)^2), si |E|~|V|^2
  - Almacenamiento:
    - Matriz: |V|^2
    - Heap: |V|
    - Aux: 3*N
    - Total: O(N^2)