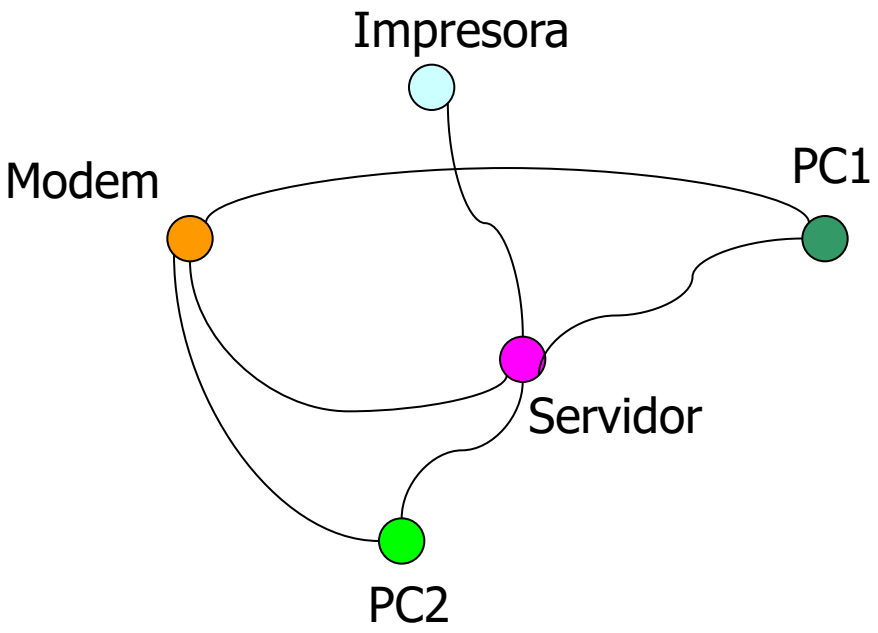
GRAFOS

ESTRUCTURA DE DATOS

INTRODUCCION

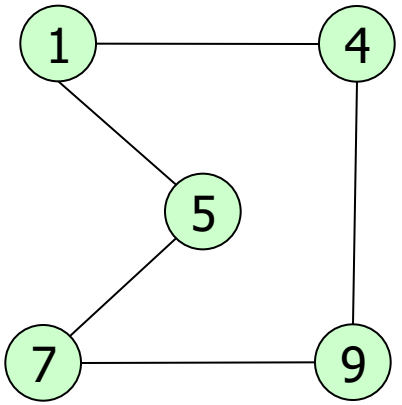
- Los grafos son estructuras de datos
- Representan relaciones entre objetos
  - Relaciones arbitrarias, es decir
  - No jerárquicas
- Son aplicables en
  - Química
  - Geografía
  - Ing. Eléctrica e Industrial, etc.
  - Modelado de Redes
    - De alcantarillado
    - Eléctricas
    - Etc.

Dado un escenario donde ciertos objetos se relacionan, se puede "modelar el grafo" y luego aplicar algoritmos para resolver diversos problemas



DEFINICION

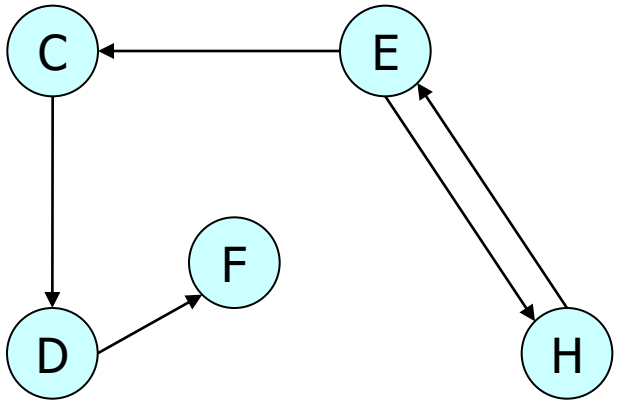
- Un grafo G = (V,A)
- V, el conjunto de vértices o nodos
  - Representan los objetos
- A, el conjunto de arcos
  - Representan las relaciones



V = {1, 4, 5, 7, 9}

A= {(1,4), (5,1), (7,9), (7,5), (4,9), (4,1), (1,5), (9,7), (5, 7), (9,4)}

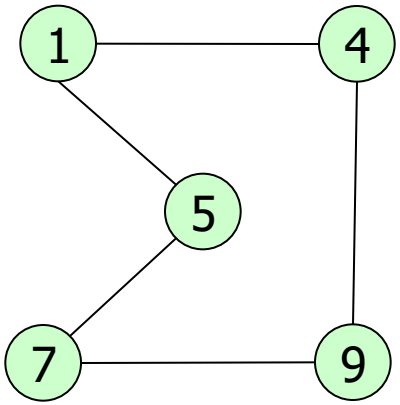
TIPOS DE GRAFOS



V = {C, D, E, F, H}

A= {(C,D), (D,F), (E,H), (H,E), (E,C)}

- Grafos dirigidos
  - Si los pares de nodos que forman arcos
  - Son ordenados. Ej.: (u->v)

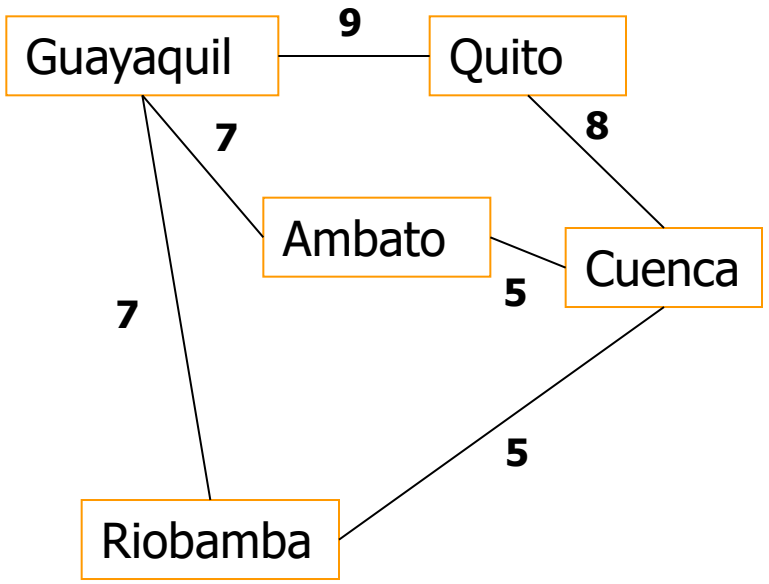


Grafo del ejemplo anterior

- Grafos no dirigidos
  - Si los pares de nodos de los arcos
  - No son ordenados Ej.: u-v

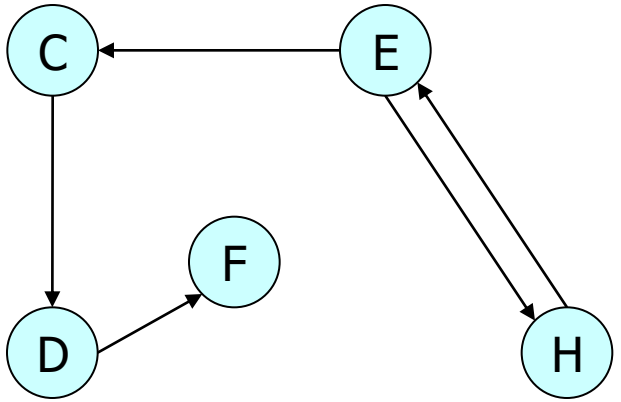
OTROS CONCEPTOS

- Arista
  - Es un arco de un grafo no dirigido
- Vertices adyacente
  - Vertices unidos por un arco
- Factor de Peso
  - Valor que se puede asociar con un arco
  - Depende de lo que el grafo represente
  - Si los arcos de un grafo tienen F.P.
    - Grafo valorado



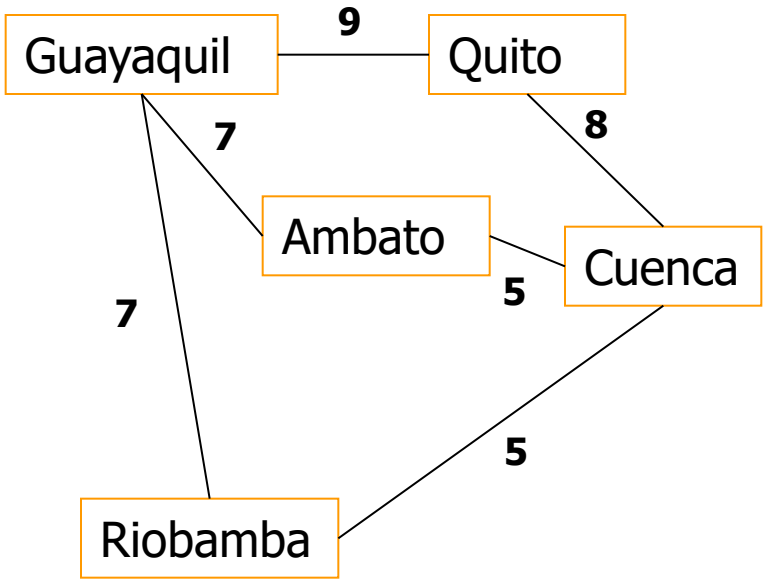
GRADOS DE UN NODO

- En Grafo No Dirigido
  - Grado(V)
    - Numero de aristas que contiene a V



Gradoent(D) = 1 y Gradsal(D) = 1

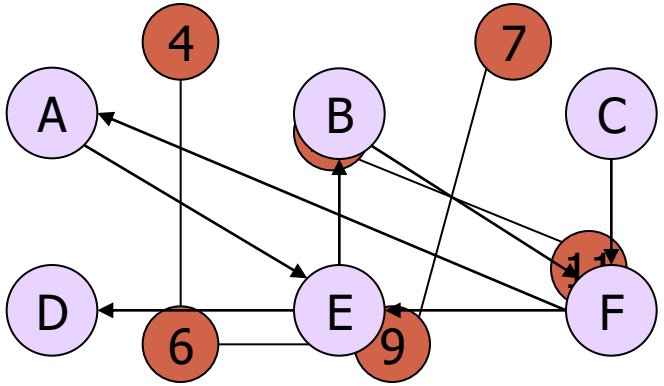
Grado(Guayaquil) = 3



- En Grafo Dirigido
  - Grado de entrada, Graden(V)
    - *Numero de arcos que llegan a V*
  - Grado de Salida, Gradsal(V)
    - *Numero de arcos que salen de V*

CAMINOS

- Definicion
  - Un camino P en un grafo G, desde V_0 a V_n
  - Es la secuencia de n+1 vertices
  - Tal que $(V_i, V_{i+1}) \in A$ para $0 \leq i \leq n$



Camino entre A y A

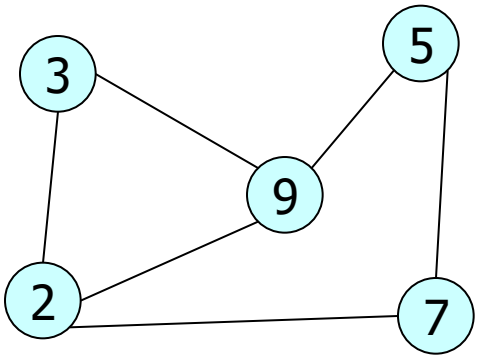
P = {A, E, B, F, A}

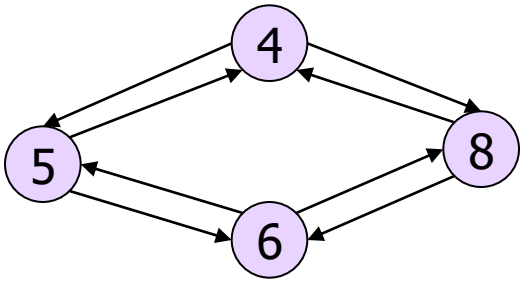
Longitud: 4 – 4ciclo

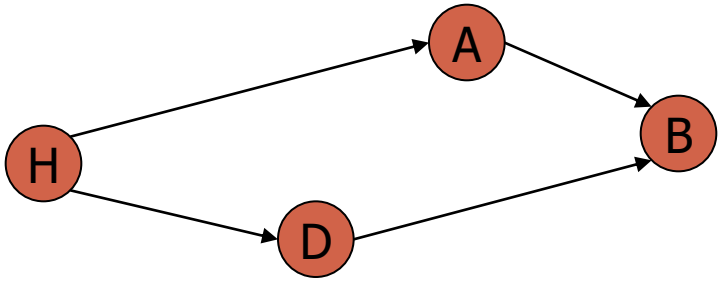
- Longitud de camino
  - El numero de arcos que lo forman
- Camino Simple
  - Todos los nodos que lo forman son distintos
- Ciclo
  - Camino simple cerrado de long. >= 2
  - Donde $V_0 = V_n$

CONECTIVIDAD

- Grafo No Dirigido
  - Conexo
    - Existe un camino entre cualquier par de nodos







- Grafo Dirigido
  - Fuertemente Conexo
    - *Existe un camino entre cualquier par de nodos*
  - Conexo
    - *Existe una cadena entre cualquier par de nodos*

TDA GRAFO

- Datos
  - Vertices y
  - Arcos(relacion entre vertices)
- Operaciones
  - void AñadirVertice(Grafo G, Vertice V)
    - Añadir un nuevo vertice
  - void BorrarVertice(Grafo G, Generico clave)
    - Eliminar un vertice existente
  - void Union(Grafo G, Vertice V1, Vertice V2)
    - Unir dos vertices
  - Void BorrarArco(Grafo G, Vertice V1, Vertice V2)
    - Eliminar un Arco
  - bool EsAdyacente(Grafo G, Vertice V1, Vertice V2)
    - Conocer si dos vertices son o no adyacentes

REPRESENTACION

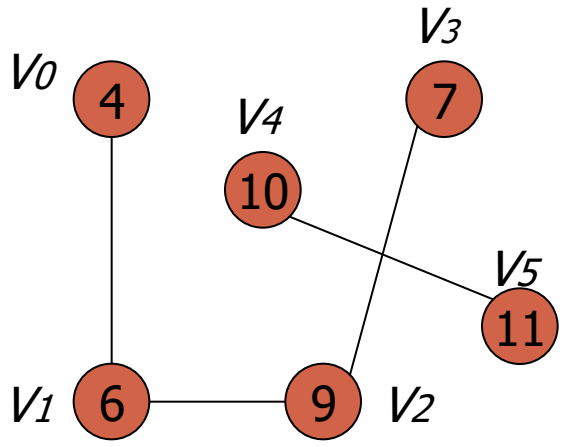
- Dos posibles representaciones
  - Estatica: Matriz de Adyacencia
    - Los vertices se representan por indices(0…n)
    - Las relaciones de los vertices se almacenan en una Matriz
  - Dinamica: Lista de Adyacencia
    - Los vertices forman una lista
    - Cada vertice tiene una lista para representar sus relaciones(arcos)

Si el grafo fuese valorado, en vez de 1, se coloca el factor de peso

MATRIZ DE ADYAC CIA

- Dado un Grafo G = (V, A)
- Sean los Vertices V = {V0, V1, … Vn}
  - Se pueden representar por ordinales 0,1,..n
- Como representar los Arcos?
  - Estos son enlaces entre vertices
- Puede usarse una matriz

$$a_{ij}\begin{cases}1, \text{si hay arco} & (Vi,Vj)\\ 0, \text{si no hay arco} & (Vi,Vj)\end{cases}$$



$$\begin{array}{c} \\ V_0 \\ V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \end{array}\begin{array}{c} \begin{array}{cccccc} V_0 & V_1 & V_2 & V_3 & V_4 & V_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

EL TIPO DE DATO

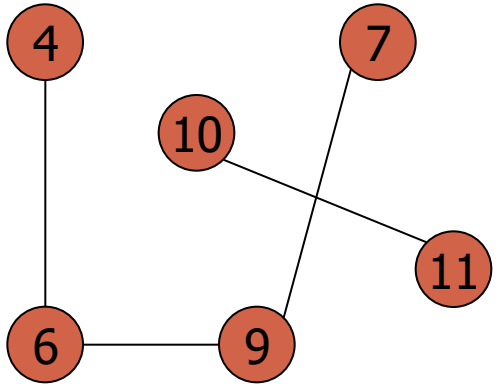
- Los Vertices
  - Se definen en un Arreglo
- Los Arcos
  - Se definen en una Matriz

```
#define MAX 20
typedef int [MAX][MAX] MatrizAdy;
typdef Generico[MAX] Vertices;
typedef struct Grafo{
        Vertices V;
        MatrizAdy A;
        int nvertices;
        int Dirigido;
};
```

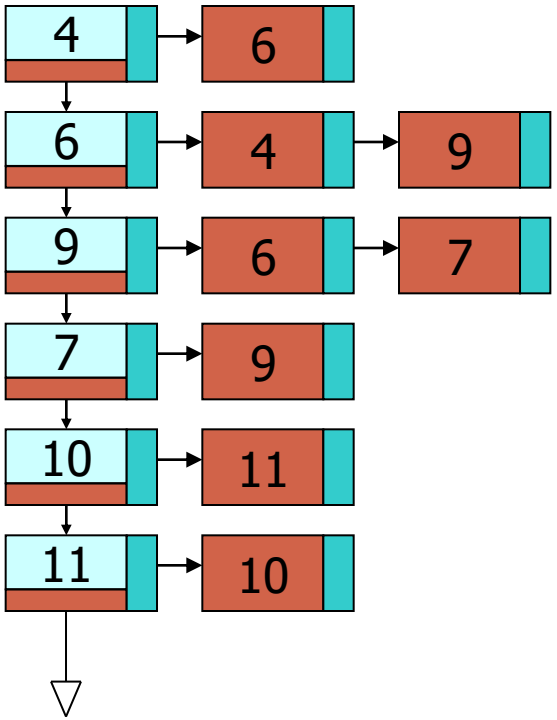
Eficiencia de la estructura Lista de Adyacencia

Operación	Tiempo
tamano, estaVacio, remplazarElemento, intercambiar	$O(1)$
numVertices, numAristas	$O(1)$
vertices	$O(n)$
aristas, aristasDirigidas, aristasNodirigidas	$O(m)$
elementos, posiciones	$O(n + m)$
verticesFinales, opuesto, origen, destino, esDirigida, grado, gradoEnt, gradoSalida	$O(1)$
aristasIncidentes, aristasIncidentesEnt, aristasIncidentesSal, verticesAdyacentes, verticesAdyacentesEnt, verticesdyacentesSal	$O(\text{grado}(v))$
esAdyacente	$O(\min(\text{grado}(u), \text{grado}(v))$
aristasIncidentes, aristasIncidentesEnt, aristasIncidentesSal, verticesAdyacentes, verticesAdyacentesEnt, verticesdyacentesSal	$O(1)$
insertaVertice	$O(1)$
eliminaVertice	$O(\text{grado}(v))$
Espacio requerido	$O(n + m)$

LISTA DE ADYACENCIA



- Si una matriz
  - Tiene muchos vertices y
  - Pocos arcos
  - La Matriz de Adyacencia
    - Tendra demasiados ceros
    - Ocupara mucho espacio
- Los vertices
  - Pueden formar una lista, no un vector
- Los arcos
  - Son relaciones entre vertices
  - Se pueden representar con una lista x cada vertice



EL TIPO DE DATO

- Cada vertice tiene
  - Contenido
  - Siguiente
  - Una lista de adyacencia
- Cada nodo en la lista de adyacencia
  - Peso del arco
  - Siguiente
  - Una referencia al vertice(arco)

```
typedef struct Vertice{
        Generico contenido;
        LSE *LA;
};
typedef Vertice *Arco;
typedef struct Grafo{
        LSE LVertices;
        int dirigido;
};
```

Eficiencia de la estructura Lista de Adyacencia

Operación	Tiempo
tamano, estaVacio, remplazarElemento, intercambiar	$O(1)$
numVertices, numAristas	$O(1)$
vertices	$O(n)$
aristas, aristasDirigidas, aristasNodirigidas	$O(m)$
elementos, posiciones	$O(n + m)$
verticesFinales, opuesto, origen, destino, esDirigida, grado, gradoEnt, gradoSalida	$O(1)$
aristasIncidentes, aristasIncidentesEnt, aristasIncidentesSal, verticesAdyacentes, verticesAdyacentesEnt, verticesdyacentesSal	$O(\text{grado}(v))$
esAdyacente	$O(\min(\text{grado}(u), \text{grado}(v))$
aristasIncidentes, aristasIncidentesEnt, aristasIncidentesSal, verticesAdyacentes, verticesAdyacentesEnt, verticesdyacentesSal	$O(1)$
insertaVertice	$O(1)$
eliminaVertice	$O(\text{grado}(v))$
Espacio requerido	$O(n + m)$

RECORRIDOS DEL GRAFO

- Se busca
  - Visitar todos los nodos posibles
  - Desde un vertice de partida D
  - Cualquiera
- Existe dos posibles recorridos
  - En Anchura y
  - En Profundidad

Recorridos sobre grafos.

- Idea similar al recorrido en un árbol.
- Se parte de un nodo dado y se visitan los vértices del grafo de manera ordenada y sistemática, *moviéndose* por las aristas.

- **Tipos de recorridos**:
  - **Búsqueda primero en profundidad.** Equivalente a un recorrido en preorden de un árbol.
  - **Búsqueda primero en amplitud o anchura.** Equivalente a recorrer un árbol por niveles.

- Los recorridos son una **herramienta** útil para resolver muchos problemas sobre grafos.

Recorridos sobre grafos.

- El recorrido puede ser tanto para grafos dirigidos como no dirigidos.
- Es necesario llevar una cuenta de los nodos visitados y no visitados.

```
var
        marca: array [1, ..., n] de (visitado, noVisitado)

operación BorraMarcas
        para i:= 1, ..., n hacer
                marca[i]:= noVisitado
```

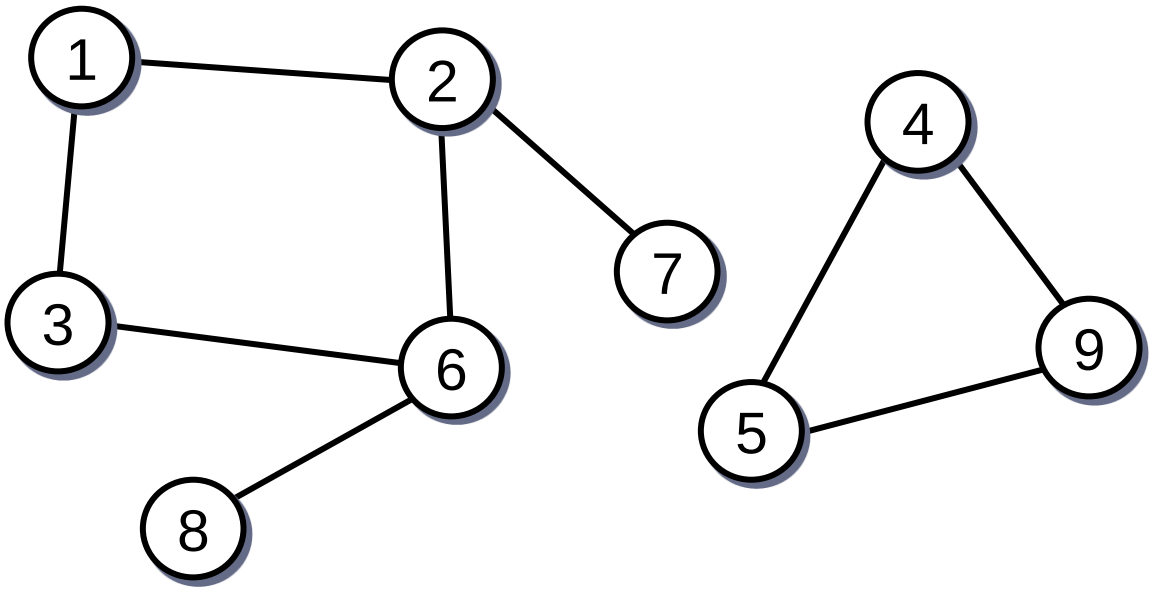
Búsqueda primero en profundidad.

```
operación bpp (v: nodo)
    marca[v]:= visitado
    para cada nodo w adyacente a v hacer
        si marca[w] == noVisitado entonces
            bpp(w)
    finpara

operación BúsquedaPrimeroEnProfundidad
    BorraMarcas
    para v:= 1, ..., n hacer
        si marca[v] == noVisitado entonces
            bpp(v)
    finpara
```

Búsqueda primero en profundidad.

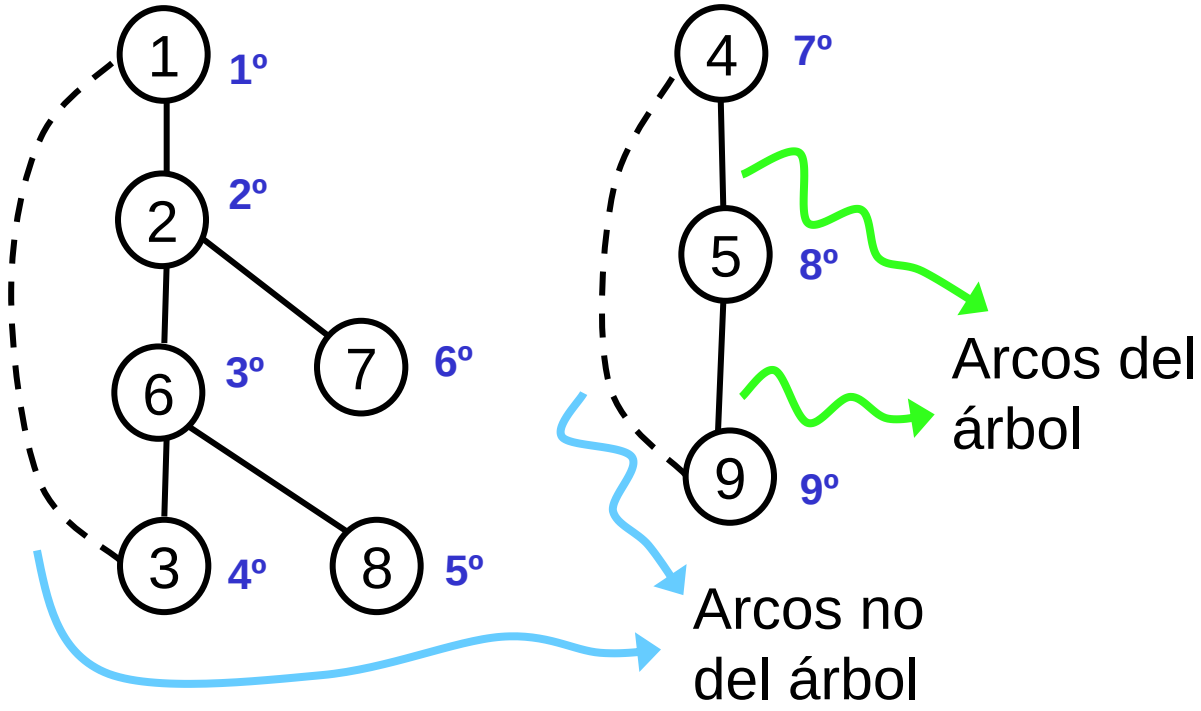
- El recorrido **no es único:** depende del nodo inicial y del orden de visita de los adyacentes.
- El orden de visita de unos nodos a partir de otros puede ser visto como un árbol: **árbol de expansión en profundidad asociado al grafo**.
- Si aparecen varios árboles: **bosque de expansión en profundidad**.

- **Ejemplo.**
  Grafo no dirigido.



Búsqueda primero en profundidad.

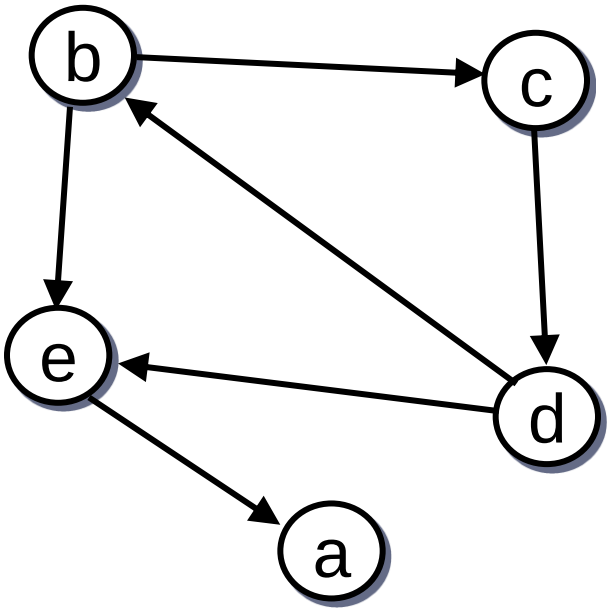
- **Bosque de expansión en profundidad**



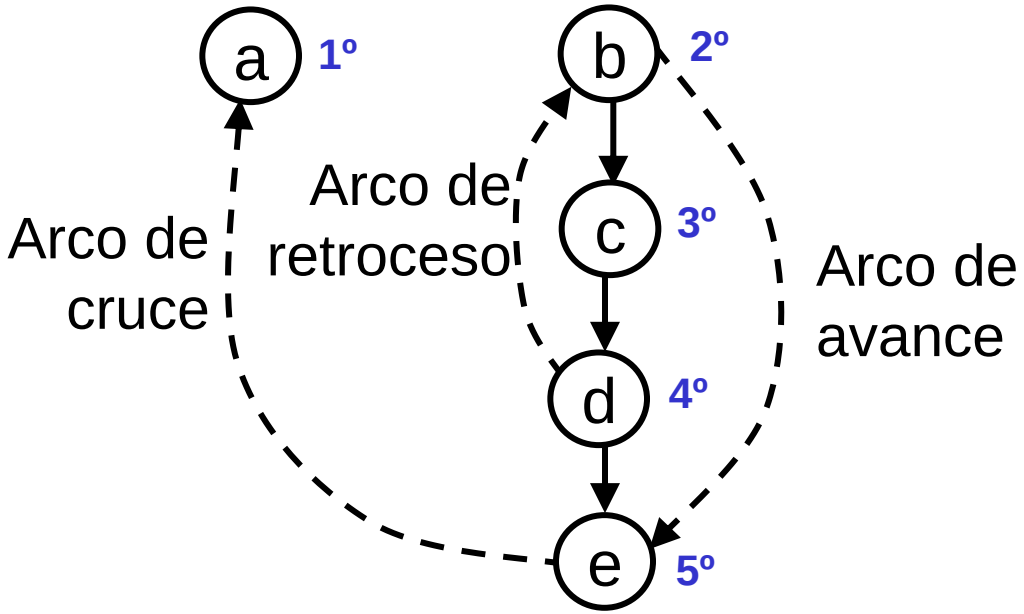
- **Arcos no del árbol: si** marca[v] == noVisitado ...
  → se detectan cuando la condición es falsa.

Búsqueda primero en profundidad.

- **Ejemplo:** Grafo dirigido.



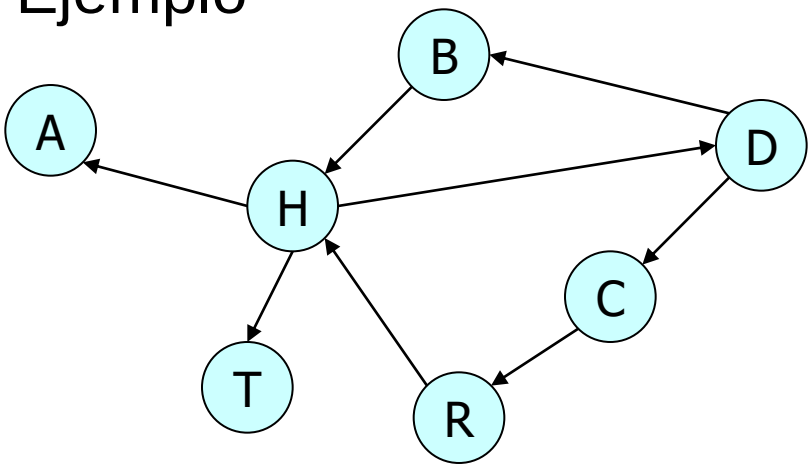
Bosque de expansión



- ¿Cuánto es el tiempo de ejecución de la BPP?
- Imposible predecir las llamadas en cada ejecución.
- **Solución:** medir el “trabajo total realizado”.

RECORRIDO EN PROFUNDIDAD

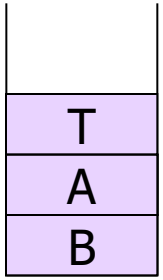
- Sin recursión utilizamos una pila
- Marcar vertice origen V como visitado
- Recorrer en Profundidad
  - Cada vertice adyacente de V
  - Que no haya sido visitado
- Ejemplo



Se Muestra:

D C R H T A B

Pila



Búsqueda primero en anchura (o amplitud).

- **Búsqueda en anchura** empezando en un nodo v:
  - Primero se visita v.
  - Luego se visitan todos sus adyacentes.
  - Luego los adyacentes de estos y así sucesivamente.
- El algoritmo utiliza una **cola de vértices**.
- Operaciones básicas:
  - Sacar un elemento de la cola.
  - Añadir a la cola sus adyacentes no visitados.

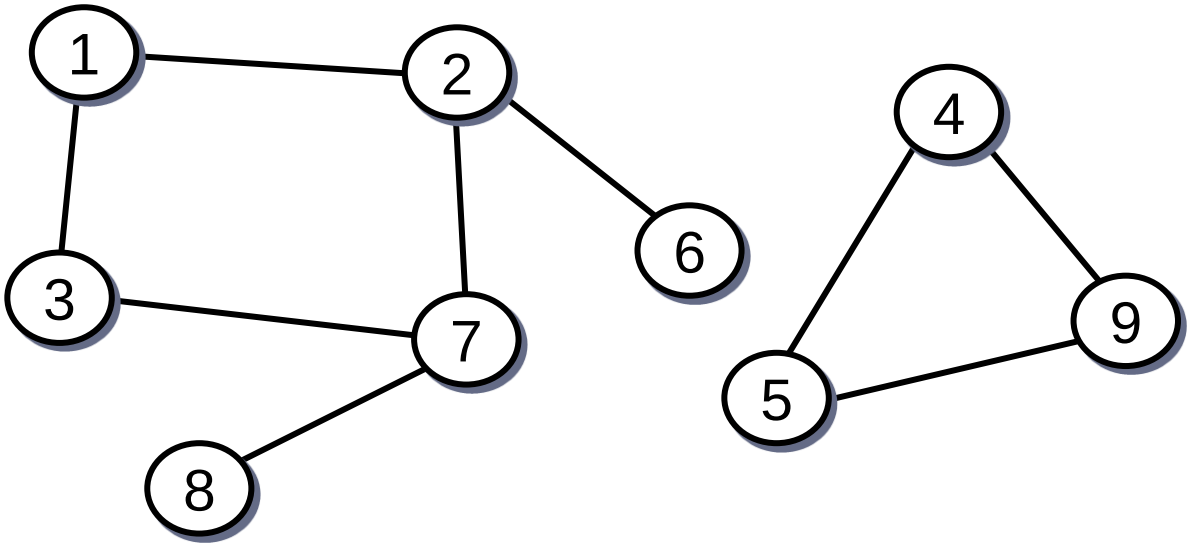
```
operación BúsquedaPrimeroEnAnchura
  BorraMarcas
  para v:= 1, ..., n hacer
          si marca[v] = noVisitado entonces
                bpa(v)
```

Búsqueda primero en anchura (o amplitud).

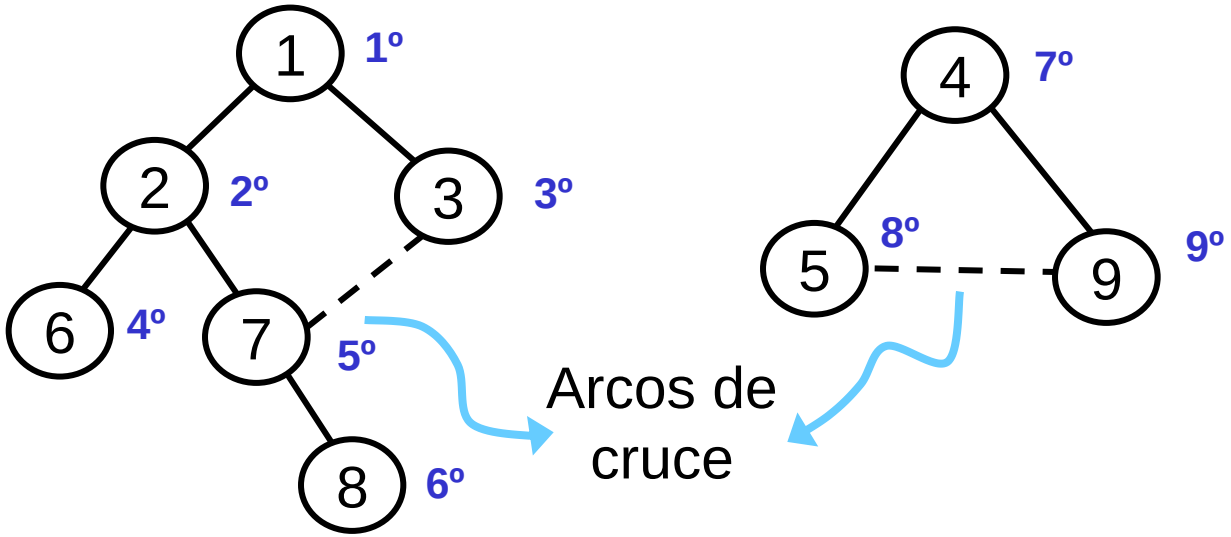
```
operación bpa (v: Nodo)
var  C: Cola[Nodo]
       x, y: Nodo

  marca[v]:= visitado
  InsertaCola (v, C)
  mientras NOT EsVacíaCola (C) hacer
       x:= FrenteCola (C)
       SuprimirCola (C)
       para cada nodo y adyacente a x hacer
            si marca[y] == noVisitado entonces
                 marca[y]:= visitado
                 InsertaCola (y, C)
             finsi
       finpara
  finmientras
```

Búsqueda primero en anchura (o amplitud).

- **Ejemplo.** Grafo no dirigido.

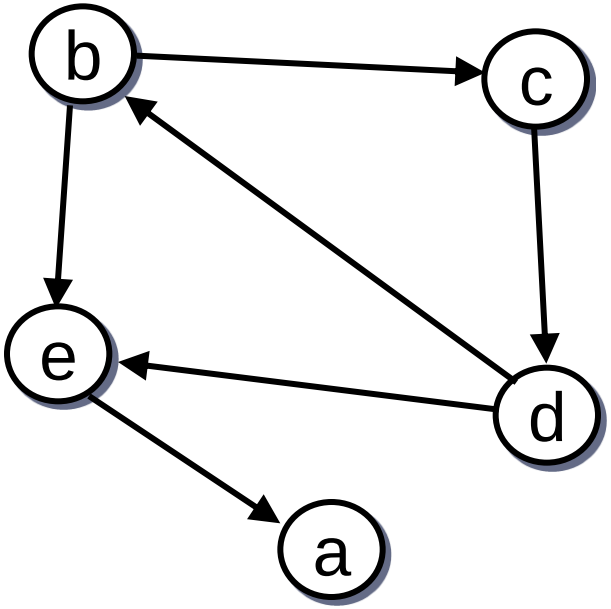


- **Bosque de expansión en anchura**.

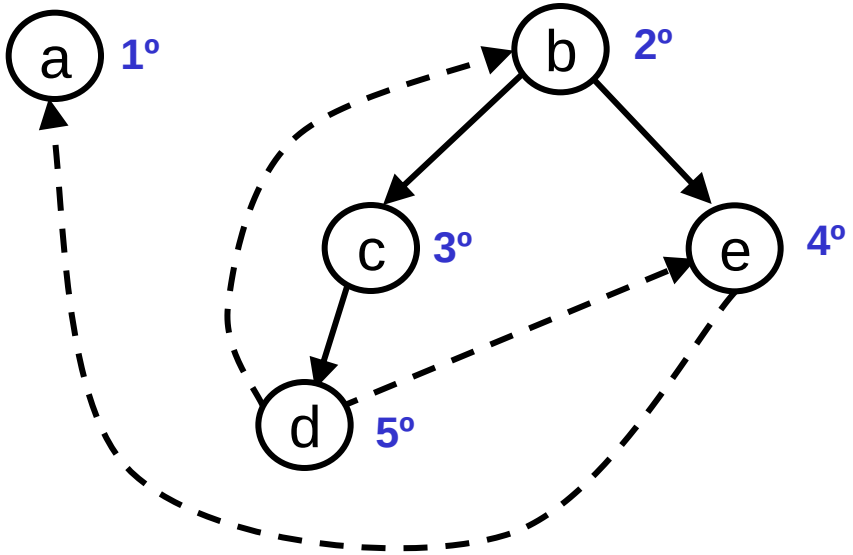


Búsqueda primero en anchura (o amplitud).

- **Ejemplo:** Grafo dirigido.

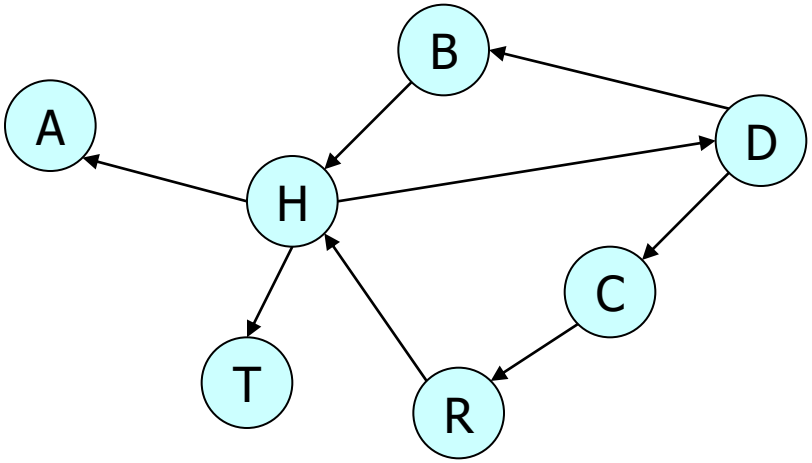


Bosque de expansión



- ¿Cuánto es el tiempo de ejecución de la BPA?
- ¿Cómo comprobar si un arco es de avance, cruce, etc.?
- **Solución:** Construir el bosque explícitamente.

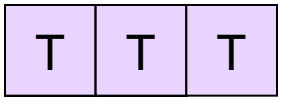
EJEMPLO



Se Muestra:

D B C H R A T

Cola



Recorridos sobre grafos.

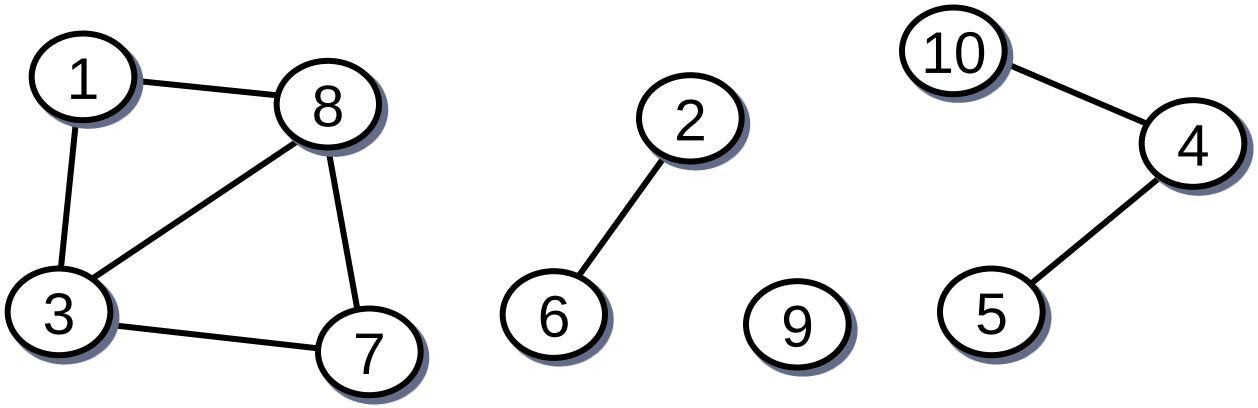
- Construcción explícita del bosque de expansión:
  Usamos una estructura de **punteros al padre**.

  marca: array [1, ..., n] **de** entero

- **marca[v]** vale: -1 si v no está visitado
  0 si está visitado y es raíz de un árbol
  En otro caso indicará cual es el padre de v
- Modificar BorraMarcas, bpp y bpa, para construir el bosque de expansión.
  - Arco de avance <v, w>: w es hijo de v en uno de los árboles del bosque.
  - Arco de retroceso <v, w>: v es hijo de w.
  - Arco de cruce <v, w>: si no se cumple ninguna de las anteriores.

Ejemplos de aplicación de los recorridos.

- **Problema 1:** Encontrar los componentes conexos de un grafo no dirigido.



- **Problema 2: Prueba de aciclicidad.** Dado un grafo (dirigido o no dirigido) comprobar si tiene algún ciclo o no.

Ejemplos de aplicación de los recorridos.

- **Prueba de aciclicidad.**
  - **Grafo no dirigido.** Hacer una BPP (o BPA). Existe algún ciclo si y sólo si aparece algún arco que no es del árbol de expansión.
  - **Grafo dirigido.** Hacer una BPP (o BPA). Existe un ciclo si y sólo si aparece algún arco de retroceso.

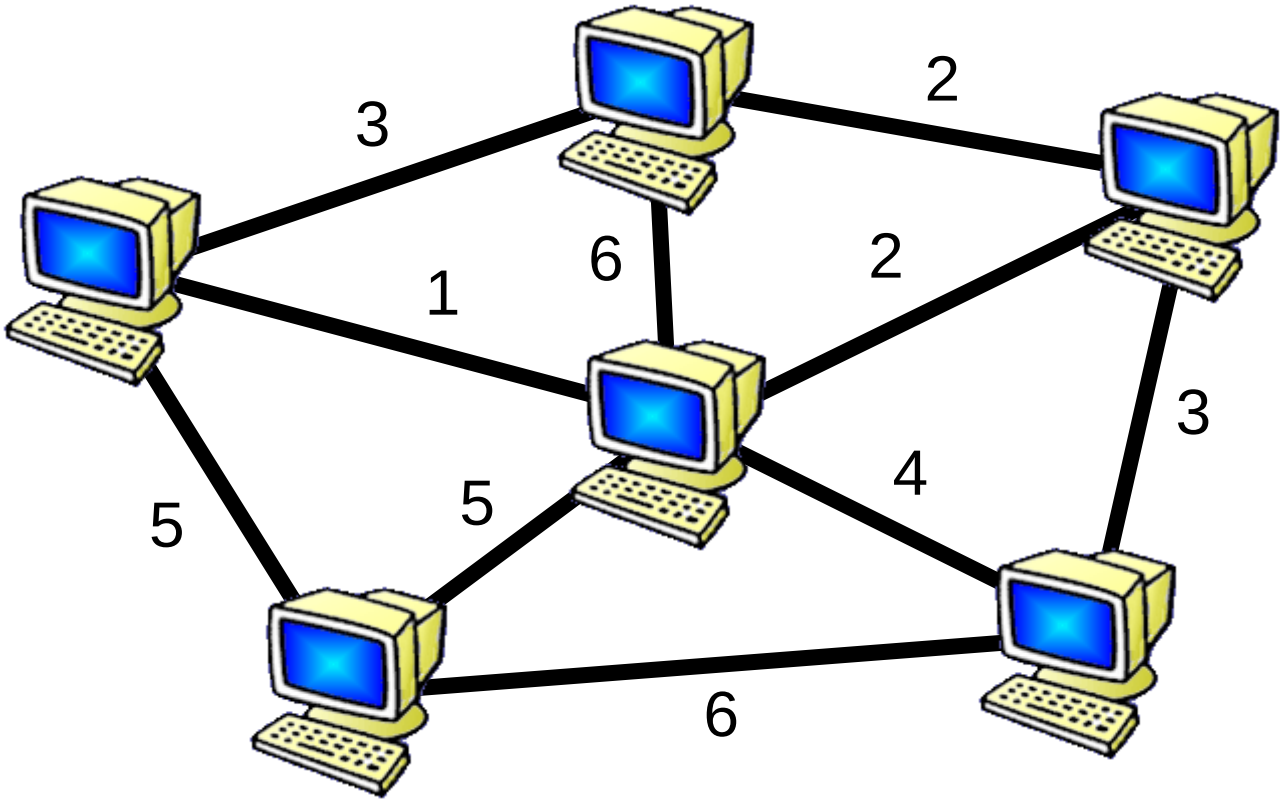
- Orden de complejidad de la prueba de aciclicidad: igual que los recorridos.
  - Con matrices de adyacencia: **$O(n^2)$**.
  - Con listas de adyacencia: **$O(a+n)$**.

Árboles de expansión mínimos.

- **Definición:** Un **árbol de expansión** de un grafo G=(V, A) no dirigido y conexo es un subgrafo G'=(V, A') conexo y sin ciclos.
- **Ejemplo:** los árboles de expansión en profundidad y en anchura de un grafo conexo.
- En grafos con pesos, el **coste del árbol de expansión** es la suma de los costes de las aristas.

- **Problema del árbol de expansión de coste mínimo:** Dado un grafo ponderado no dirigido, encontrar el árbol de expansión de menor coste.

Árboles de expansión mínimos.



- **Problema:** conectar todas las computadoras con el menor coste total.
- **Solución:** algoritmos clásicos de Prim y Kruskal.

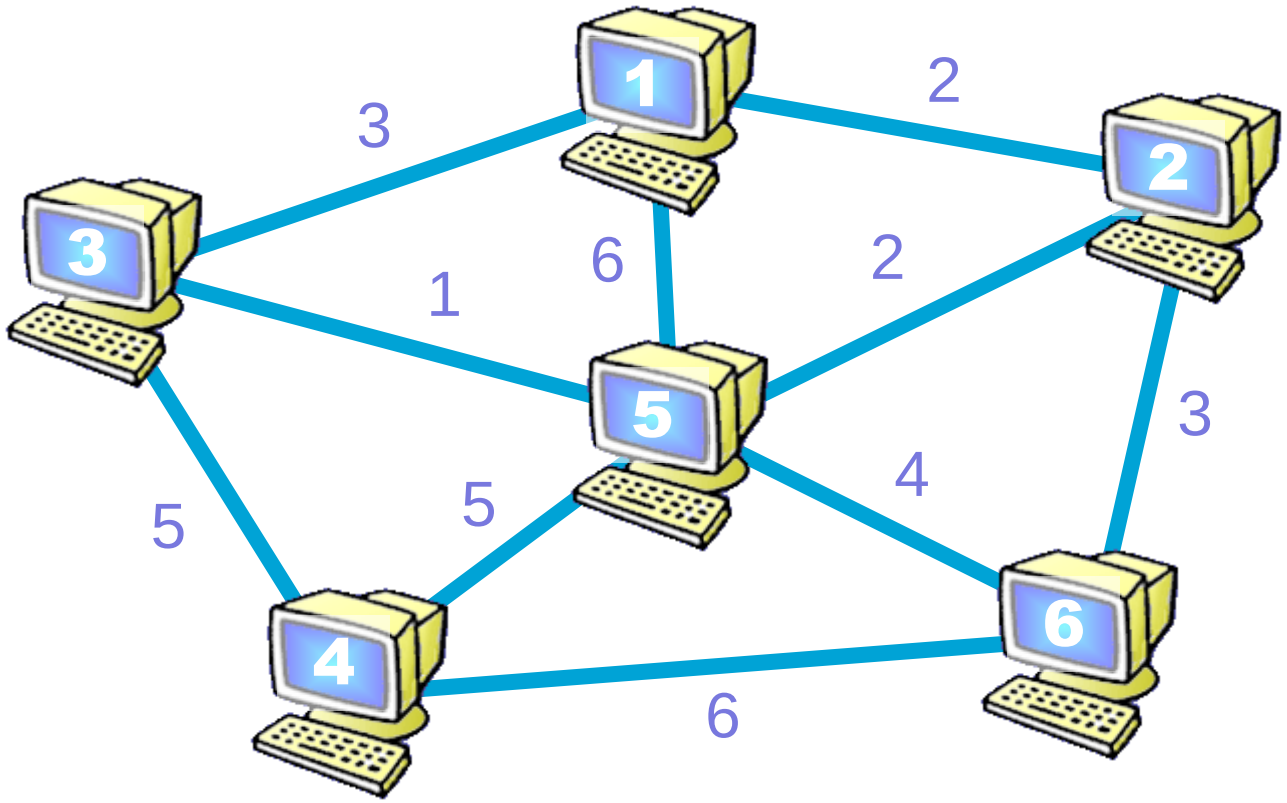
Algoritmo de Prim.

Esquema:

1. Empezar en un vértice cualquiera **v**. El árbol consta inicialmente sólo del nodo **v**.
2. Del resto de vértices, buscar el que esté más próximo a **v** (es decir, con la arista (**v**, **w**) de coste mínimo). Añadir **w** y la arista (**v**, **w**) al árbol.
3. Buscar el vértice más próximo a cualquiera de estos dos. Añadir ese vértice y la arista al árbol de expansión.
4. Repetir sucesivamente hasta añadir los **n** vértices.

Algoritmo de Prim.

- **Ejemplo de ejecución del algoritmo.**



Algoritmo de Prim.

- La solución se construye **poco a poco**, empezando con una solución “vacía”.

- Implícitamente, el algoritmo maneja los **conjuntos**:
  - **V**: Vértices del grafo.
  - **U**: Vértices añadidos a la solución.
  - **V-U**: Vértices que quedan por añadir.

- ¿Cómo implementar eficientemente la búsqueda: encontrar el vértice de **V-U** más próximo a alguno de los de **U**?

Algoritmo de Prim.

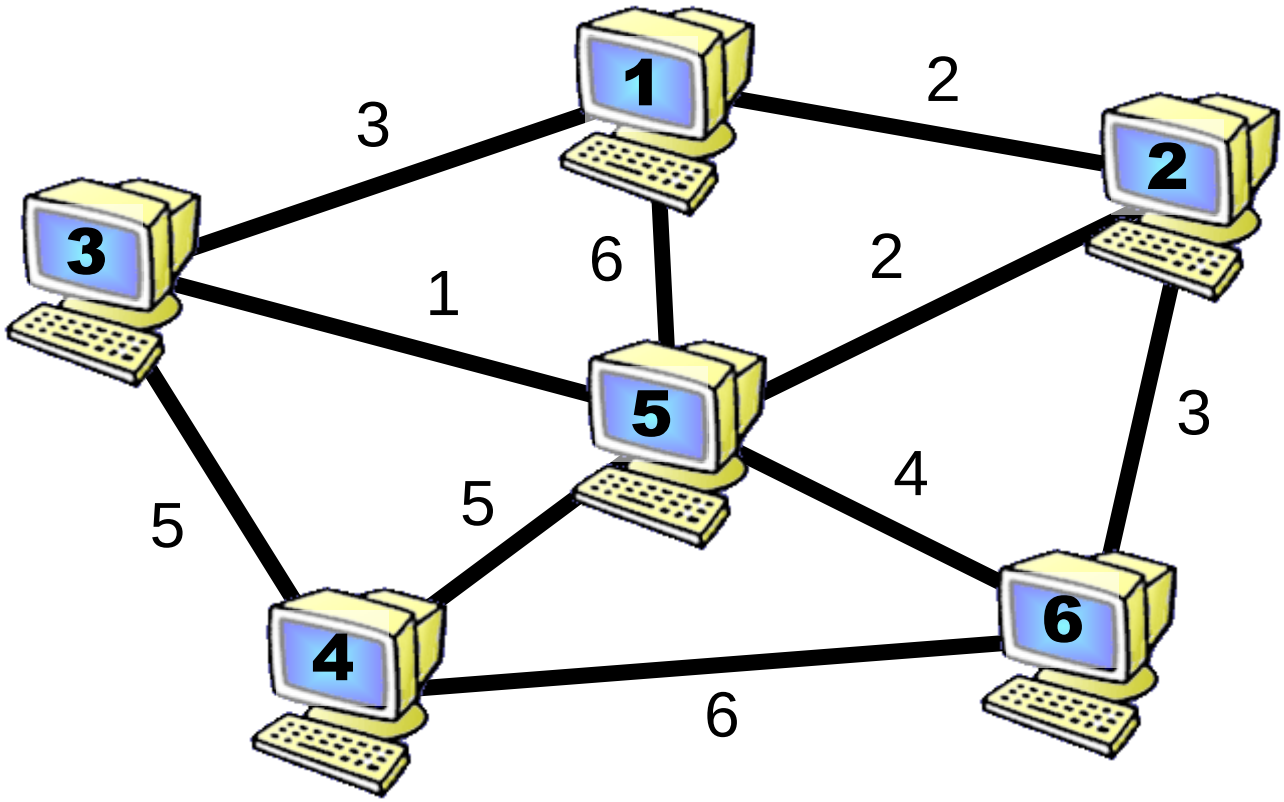
- Se usan dos arrays:
  - **MAS_CERCANO**: Para cada vértice de **V-U** indica el vértice de **U** que se encuentra más próximo.
  - **MENOR_COSTE**: Indica el coste de la arista más cercana.

Estructura del algoritmo de Prim: C[v, w] Matriz de costes

1. Inicialmente **U**= {1}. MAS_CERCANO[v]= 1.
   MENOR_COSTE[v]= C[1, v], para v= 2..n
2. Buscar el nodo **v**, con MENOR_COSTE mínimo.
   Asignarle un valor muy grande (para no volver a cogerlo).
3. Recalcular MAS_CERCANO y MENOR_COSTE de los nodos de **V-U**. Para cada w de **V-U**, comprobar si C[v, w] es menor que MENOR_COSTE[w].
4. Repetir los dos puntos anteriores hasta que se hayan añadido los **n** nodos.

Algoritmo de Prim.

- **Ejemplo:** Mostrar la ejecución del algoritmo sobre el grafo.



- ¿Dónde está almacenado el resultado del algoritmo?
- ¿Cuál es el orden de complejidad del algoritmo?

Algoritmo de Kruskal.

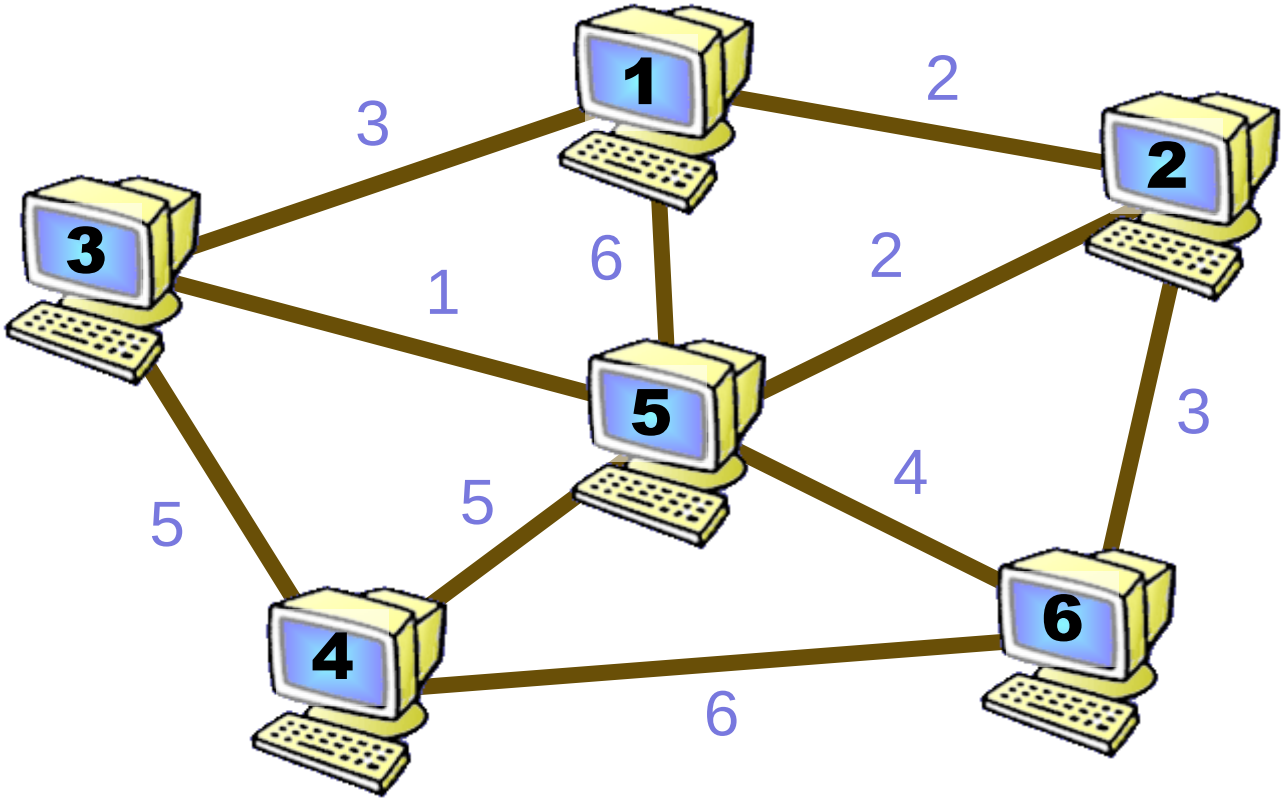
Esquema: G= (V, A)

1. Empezar con un grafo sin aristas: **G'**= (V, Ø)
2. Seleccionar la arista de menor coste de **A**.
3. Si la arista seleccionada forma un ciclo en **G'**, eliminarla. Si no, añadirla a **G'**.
4. Repetir los dos pasos anteriores hasta tener **n-1** aristas.

- ¿Cómo saber si una arista **(v, w)** provocará un ciclo en el grafo **G'**?

Algoritmo de Kruskal.

- **Ejemplo:** Mostrar la ejecución del algoritmo en el siguiente grafo.



Algoritmo de Kruskal.

Implementación del algoritmo

- **Necesitamos:**
  - Ordenar las aristas de **A**, de menor a mayor: **O(a log a).**
  - Saber si una arista dada **(v, w)** provocará un ciclo.
- ¿Cómo comprobar rápidamente si **(v, w)** forma un ciclo?
- Una arista **(v, w)** forma un ciclo si **v** y **w** están en el mismo componente conexo.
- La relación “estar en el mismo componente conexo” es una **relación de equivalencia**.

Algoritmo de Kruskal.

- Usamos la estructura de **relaciones de equivalencia** con punteros al padre:
  - Inicialización: crear una relación de equivalencia vacía (cada nodo es un componente conexo).
  - Seleccionar las aristas **(v, w)** de menor a mayor.
  - La arista forma ciclo si: **Encuentra(v)**=**Encuentra(w)**
  - Añadir una arista **(v, w)**: **Unión(v, w)** (juntar dos componentes conexos en uno).
- Mostrar la ejecución sobre el grafo de ejemplo.
- ¿Cuál es el orden de complejidad del algoritmo?

Otros problemas con grafos.

Problemas genéricos y clásicos sobre grafos:

- **Problemas de flujo en redes:** Los grafos representan canales de flujo de información, de líquidos, mercancías, coches, etc.
- **Problema del viajante:** Optimización de rutas en mapas de carreteras.
- **Coloración de grafos:** Los grafos representan relaciones de incompatibilidad.
- **Comparación, isomorfismo y subisomorfismo:** Representación de información “semántica”, búsqueda de patrones, inteligencia artificial.

Otros problemas con grafos.

Problemas de flujo en redes

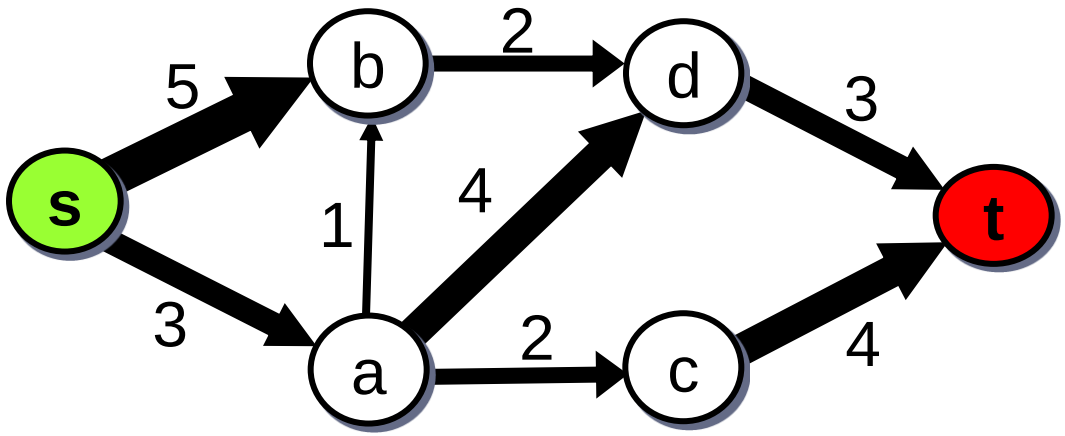
- Supongamos un grafo dirigido **G**= (V, A) con pesos.
  - Los nodos representan puntos de una red.
  - Las aristas representan canales de comunicación existentes entre dos puntos.
  - Los pesos de cada arista **C(v, w)** representan el número máximo de unidades que pueden “fluir” desde el nodo **v** al **w**.

- **Problema:** Encontrar el máximo volumen que se puede enviar entre dos puntos.

Otros problemas con grafos.

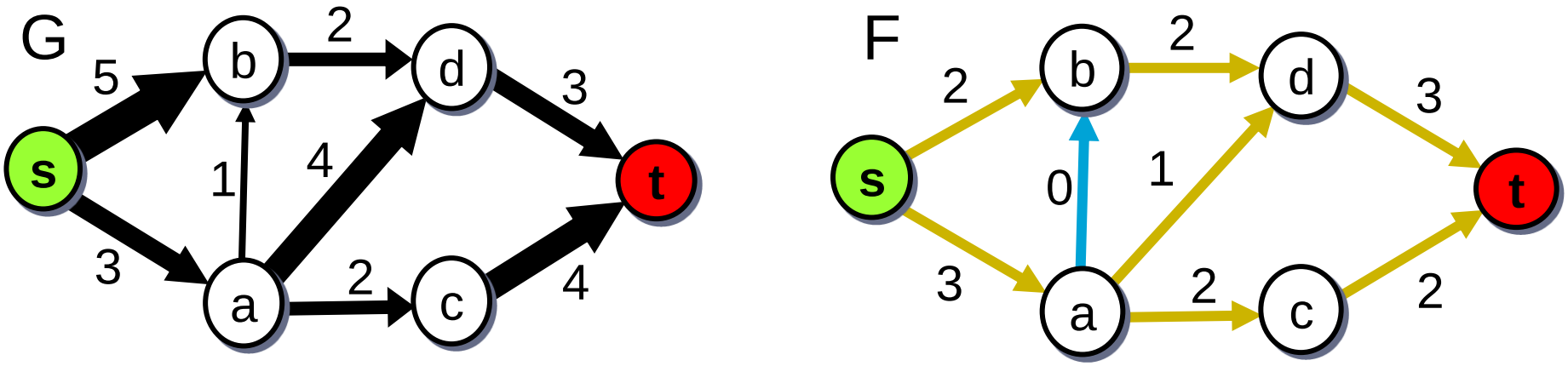
- **Problema del flujo máximo:**
  Dado un nodo origen **s** y un nodo destino **t** en un grafo dirigido con pesos, **G**, encontrar la cantidad máxima de flujo que puede pasar de **s** a **t**.
- **Restricciones:**
  - La suma de las entradas de cada nodo interior debe ser igual a la suma de sus salidas.
  - Los valores de flujo en cada arista no pueden superar los valores máximos.

G



Otros problemas con grafos.

- **Solución. G:** Grafo del problema. **F:** Grafo resultante.

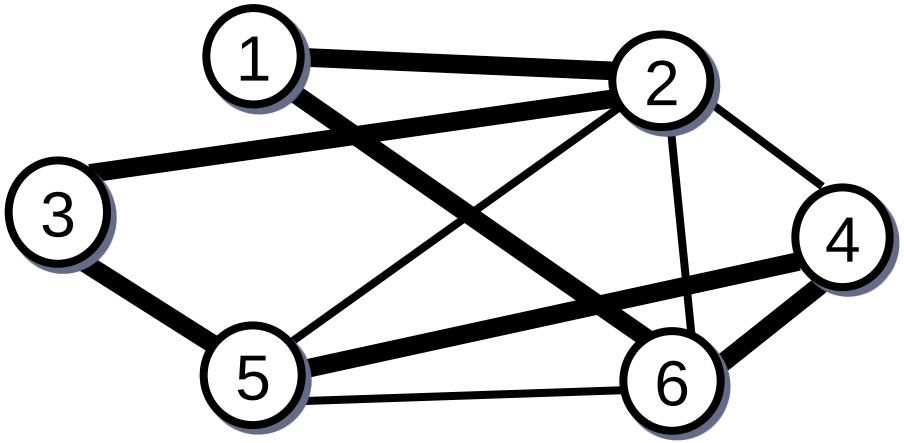


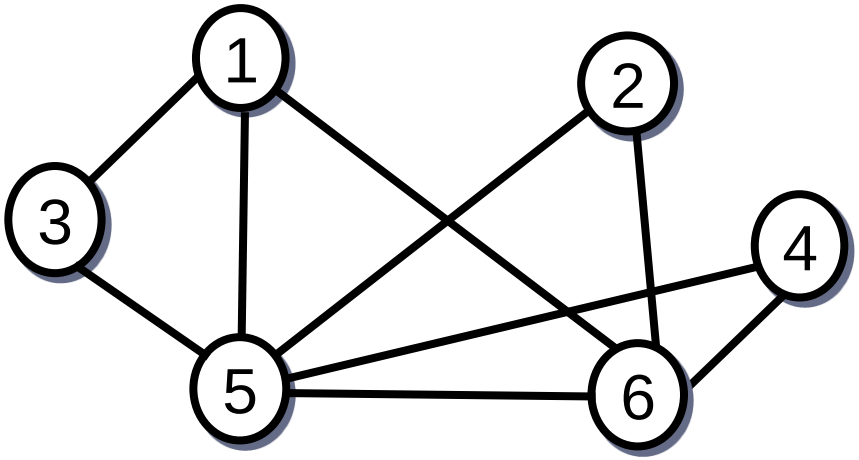
- El problema se puede resolver de forma eficiente.
- **Posible algoritmo:**
  - Encontrar un camino cualquiera desde **s** hasta **t**.
  - El máximo flujo que puede ir por ese camino es el mínimo coste de las aristas que lo forman, **m**.
  - Sumar **m** en el camino en **F**, y restarlo de **G**.
- **Ojo:** este algoritmo no garantiza solución óptima.

Otros problemas con grafos.

Problema del ciclo hamiltoniano

- **Definición:** Dado un grafo no dirigido **G**, un **ciclo de Hamilton (o hamiltoniano)** es un ciclo simple que visita todos los vértices. Es decir, pasa por todos los vértices exactamente una vez.
- **Problema del ciclo hamiltoniano.** Determinar si un grafo no dirigido dado tiene un ciclo hamiltoniano o no.





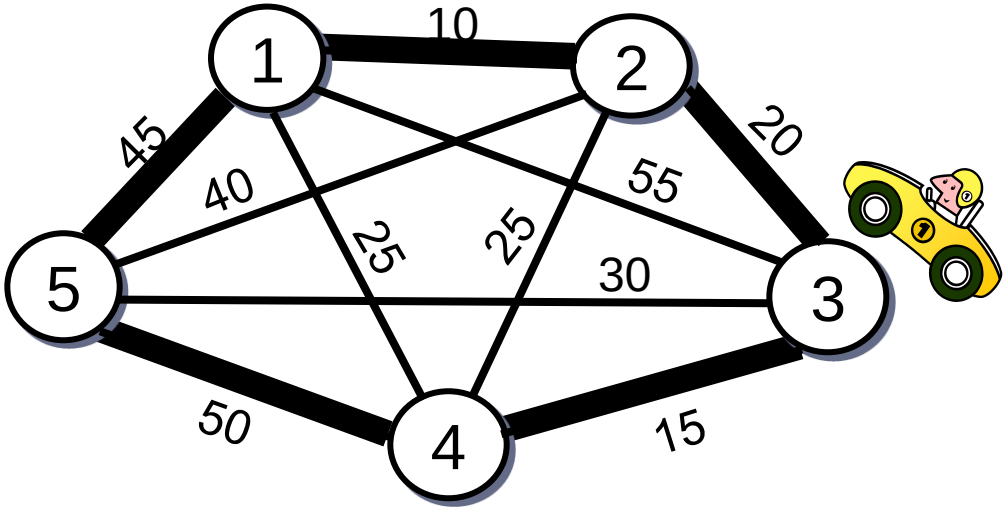
Otros problemas con grafos.

- Aunque el problema es muy parecido al del circuito de Euler, no se conoce ningún algoritmo eficiente para resolverlo, en tiempo polinomial.
- El problema del ciclo hamiltoniano pertenece a un conjunto de problemas de difícil solución, llamados **problemas NP-completos**.
- Las soluciones conocidas requieren básicamente "evaluar todas las posibilidades", dando lugar a órdenes de complejidad exponenciales o factoriales.
- Otra alternativa es usar métodos **heurísticos**: soluciones aproximadas que pueden funcionar en algunos casos y en otros no.

Otros problemas con grafos.

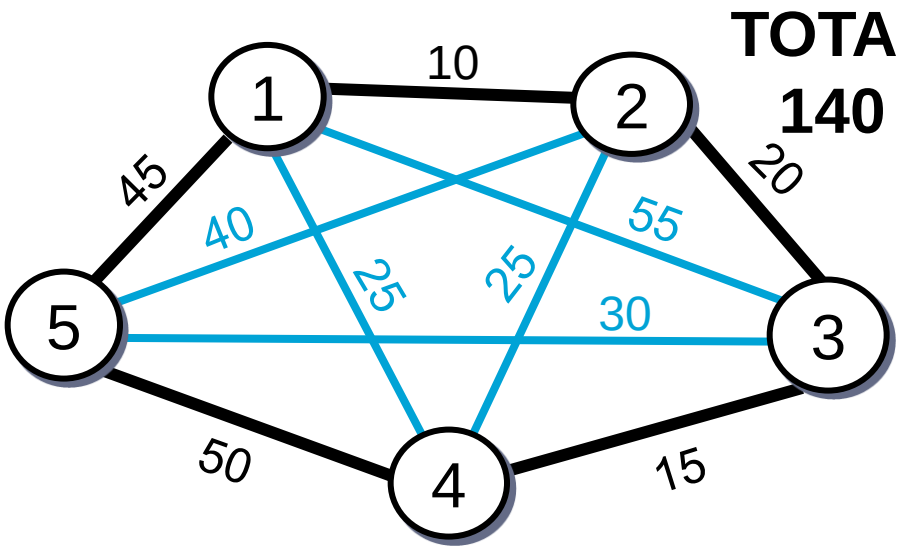
Problema del viajante (o del agente viajero)

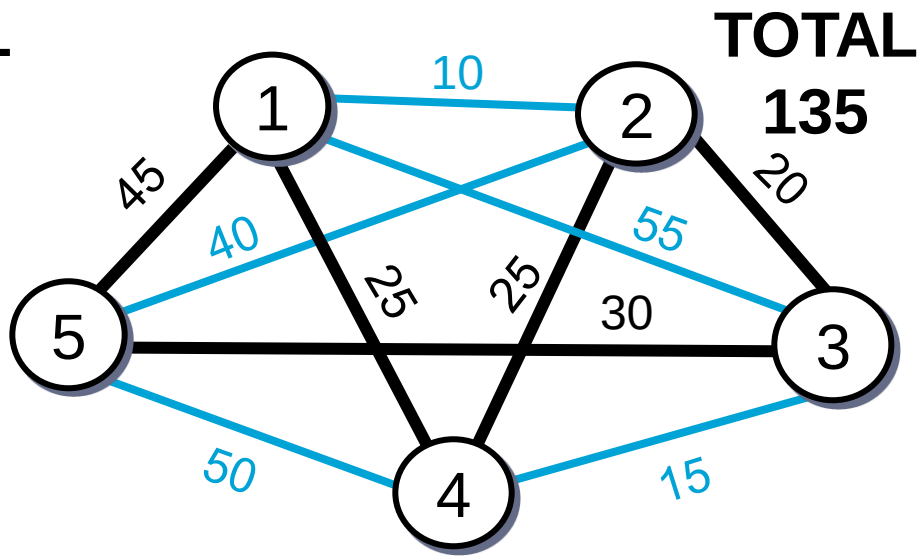
- Dado un grafo no dirigido, completo y con pesos, **G**, encontrar un ciclo simple de costo mínimo.



- **Ejemplo:** Un cartero tiene que repartir cartas por todo el pueblo. ¿Qué ruta debe seguir para que el coste de desplazamiento sea mínimo?

Otros problemas con grafos.





- El problema del viajante es un problema **NP-completo**, equivalente (reducible) al problema del ciclo hamiltoniano.
- No se conoce una solución con tiempo polinómico. Las soluciones conocidas tienen complejidad exponencial.
- Podemos aplicar heurísticas, técnicas probabilistas, algoritmos genéticos, computación con ADN, etc., obteniendo aproximaciones.

Otros problemas con grafos.

Coloración de grafos

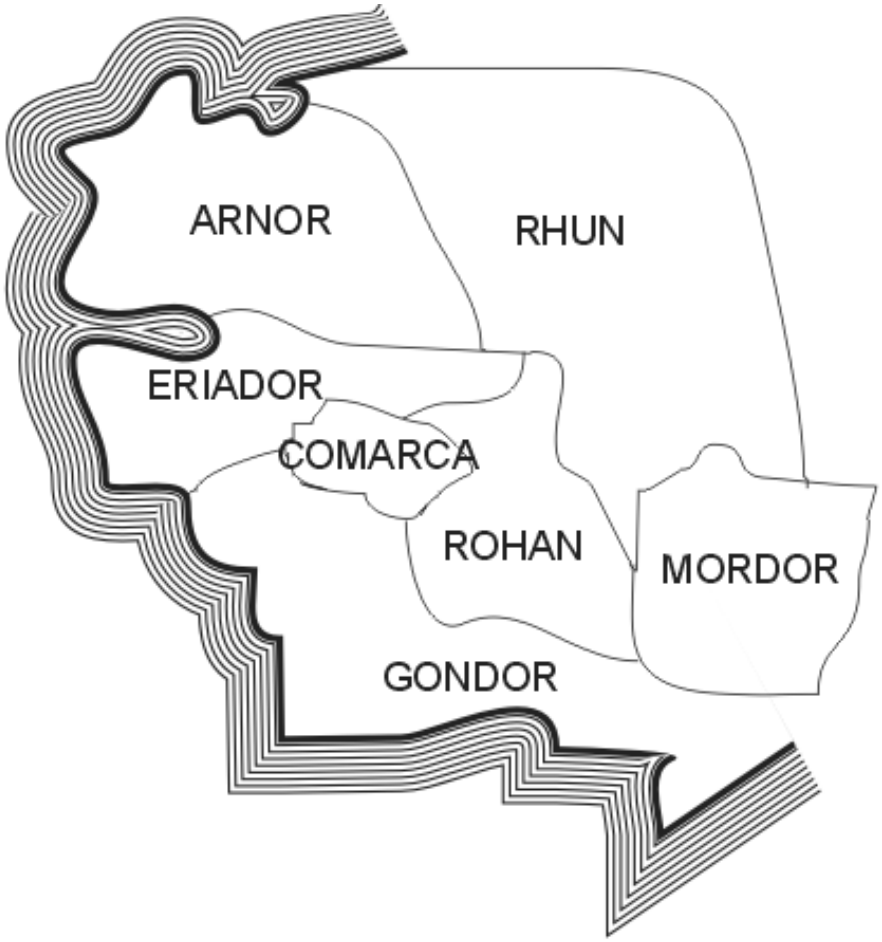
- Un grafo no dirigido **G** representa ciertos elementos.
- Una arista (**v**, **w**) representa una incompatibilidad entre los elementos **v** y **w**.

- La **coloración de un grafo** consiste en asignar un color (o etiqueta) a cada nodo, de forma que dos nodos incompatibles no tengan el mismo color.

- **Problema de coloración de grafos:**
  Realizar una coloración del grafo utilizando un número mínimo de colores.

Otros problemas con grafos.

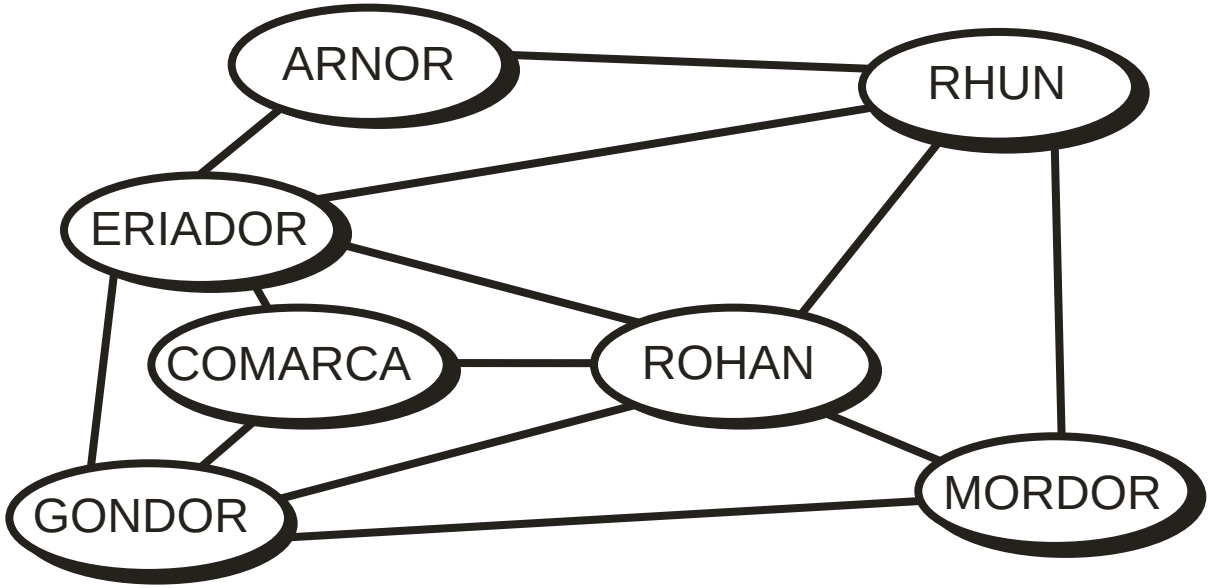
- **Ejemplo:** ¿Con cuántos colores, como mínimo, se puede pintar un mapa? Dos regiones adyacentes no pueden tener el mismo color.



- Modelamos el problema con una representación de grafos.

Otros problemas con grafos.

- **Modelado del problema:**
  - **Nodos** del grafo: Regiones del mapa.
  - **Aristas** del grafo: Hay una arista **(v, w)** si las regiones **v** y **w** tienen una frontera común.
  - **Solución:** Encontrar la coloración mínima del grafo.



- La coloración de grafos es un problema **NP-completo**.

Otros problemas con grafos.

Comparación e Isomorfismo de grafos

Igualdad

- **Definición:** Dados dos grafos **G**= (V_G, A_G) y **F**= (V_F, A_F), se dicen que son **iguales** si $V_G = V_F$ y $A_G = A_F$.

Isomorfismo

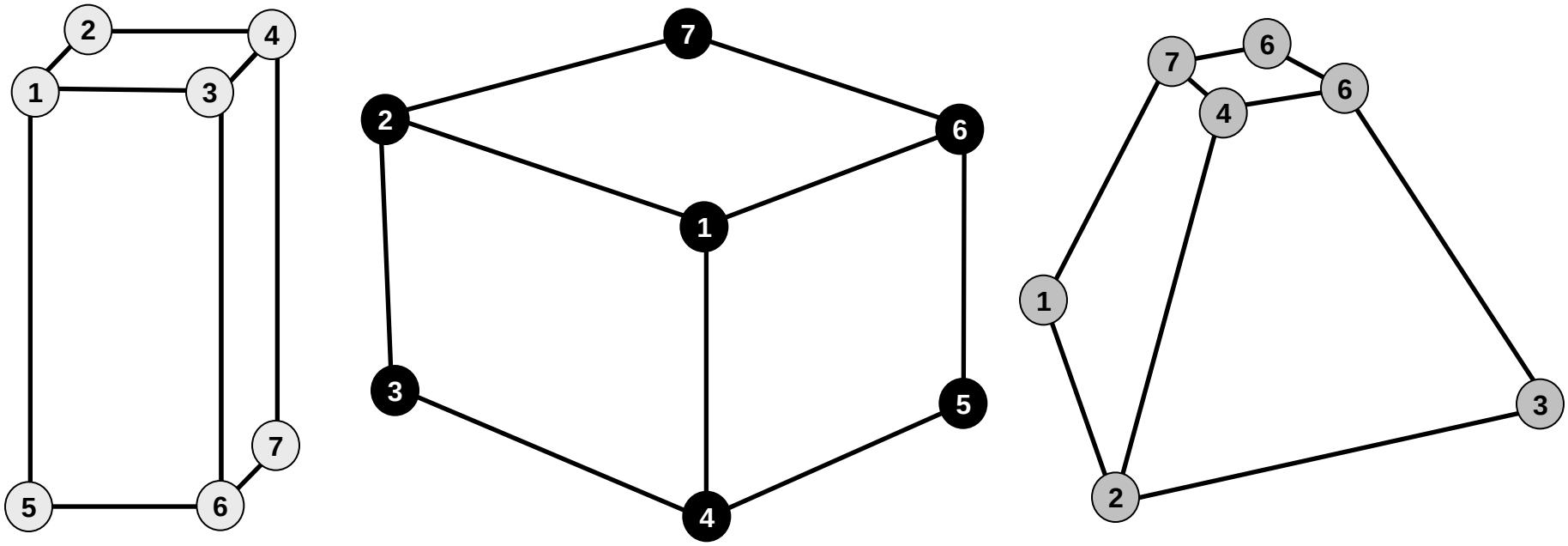
- **Definición:** Dos grafos **G**= (V_G, A_G) y **F**= (V_F, A_F) se dice que son **isomorfos** si existe una asignación de los nodos de V_G con los nodos de V_F tal que se respetan las aristas.
- **Isomorfismo entre grafos.** El isomorfismo es una función:

$$\mathbf{a} : V_G \rightarrow V_F, \text{ biyectiva tal que}$$

$$(v, w) \in A_G \Leftrightarrow (a(v), a(w)) \in A_F$$

Otros problemas con grafos.

- **Ejemplo: Reconocimiento de patrones.** Identificar las figuras isomorfas y los puntos “análogos” en ambas.



- El isomorfismo de grafos es también un problema **NP-completo**.
- La solución consistiría, básicamente, en comprobar todas las posibles asignaciones.