Estructuras de datos

1. Tipos abstractos de datos y algoritmia
2. Listas, pilas, colas y conjuntos
3. Árboles
4. **Grafos** ←

Universidad Rey Juan Carlos
www.urjc.es

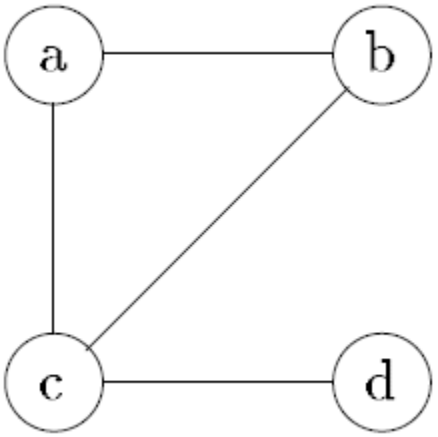
Grado en Ingeniería de Computadores
Curso 2020/2021

Prof. Dr. Carlos Grima Izquierdo
www.carlosgrima.com

v1

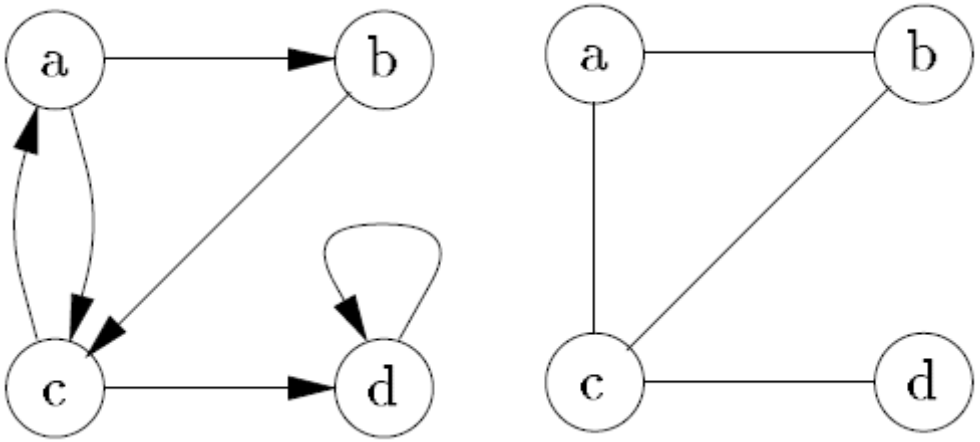
Definiciones de grafo I

- Un grafo es (definición informal):
  - Una colección de elementos interconectados entre sí con cualquier organización imaginable
    - Una lista es una colección de elementos interconectados entre sí con organización secuencial
    - Un árbol es una colección de elementos interconectados entre sí con organización jerárquica



Definiciones de grafo II

- Un grafo es (definición formal):
  - Un par (N, A) donde N es un conjunto de elementos llamados nodos o vértices y A es un conjunto de pares de nodos, denominados arcos o aristas
  - Si los pares son ordenados (importa qué nodo pones en primer lugar en el par), el grafo será dirigido. En caso contrario será no dirigido
    - Si el grafo es dirigido (ej: primera figura), los arcos se pintan con flechas. Si no (ej: segunda figura), se pintan sólo como líneas sin flechas
    - Si no decimos nada, asumimos que un grafo es no dirigido
    - Un arco no dirigido equivale a dos arcos dirigidos, uno por cada sentido



Definiciones de grafo III

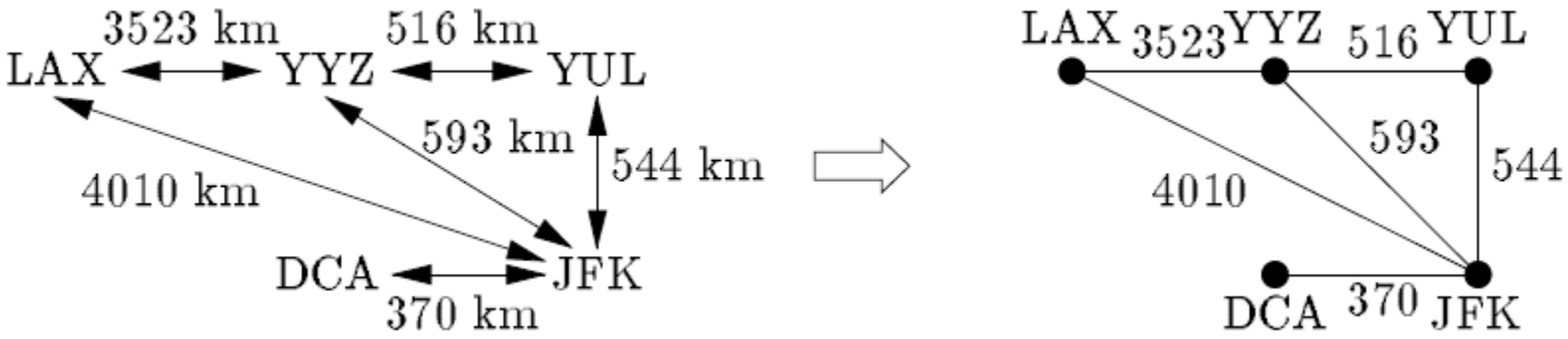
- Posible definición recursiva de grafo:
  - Un sólo nodo (o ninguno) es un grafo
  - El resultado de introducir un nuevo nodo en un grafo es un grafo
  - El resultado de conectar dos nodos de un grafo mediante un nuevo arco es un grafo

Definiciones de grafo IV

- Pseudografo: grafo en el que hay arcos en los que los dos extremos se corresponden con el mismo nodo
- Multigrafo: grafo en el que puede haber más de un arco entre un par de nodos
- Camino: lista de nodos en la que cada par de elementos sucesivos es un arco
- Ciclo: camino en el que el primer y el último nodo son iguales, y sin arcos repetidos
- Grafo acíclico: grafo sin ciclos
- Grafo conexo: grafo en el que ningún nodo está desconectado del resto
- Árbol: grafo acíclico, conexo, no dirigido, no pseudografo, no multigrafo, y en el que uno de los nodos se marca como raíz
- Grafo etiquetado: grafo en el que se asocia una etiqueta (una información) a cada arco. Si la etiqueta es numérica se llama grafo ponderado.

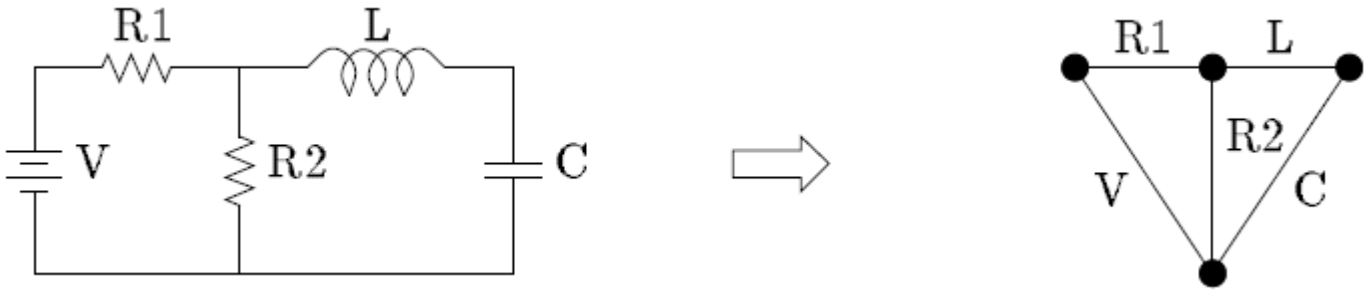
Ejemplos de grafos I

- Aeropuertos internacionales (nodos) y sus rutas (arcos)
  - Grafo dirigido, no pseudografo, no multigrafo (en este ejemplo), no acíclico, conexo, no árbol, etiquetado con las distancias de las rutas
  - Como todos los arcos cubren los dos sentidos, esto es equivalente a que el grafo sea no dirigido
    - Recordemos que un arco no dirigido equivale a dos dirigidos entre el mismo par de nodos, uno por cada sentido



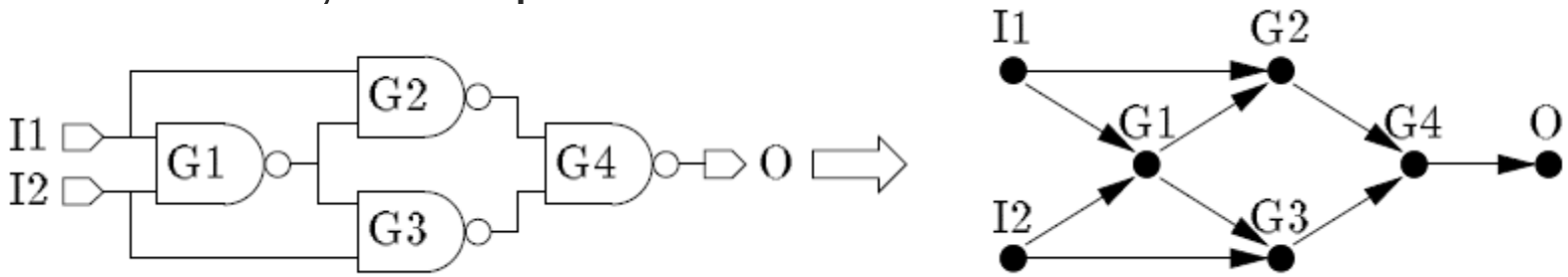
Ejemplos de grafos II

- Un circuito eléctrico puede ser visto como un grafo en el cual cada nodo de cables es un nodo, y cada elemento del circuito (resistencia, condensadores, pilas, etc.) es una etiqueta del arco
  - En este ejemplo, el grafo sería no dirigido, no pseudografo, no multigrafo, no acíclico, conexo, no árbol, etiquetado con los elementos del circuito



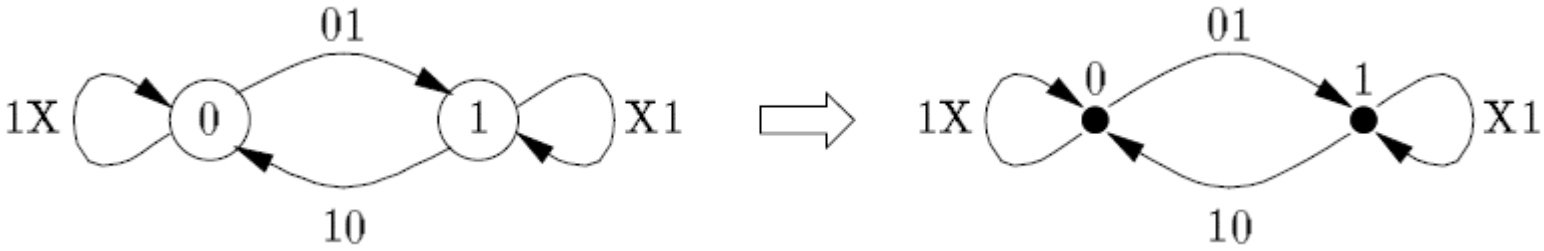
Ejemplos de grafos III

- Un circuito de puertas lógicas con dos entradas y una salida
  - Las entradas, salidas y puertas lógicas pueden ser vistas como nodos
  - Los cables que conectan los nodos son los arcos
  - En este ejemplo, el grafo sería dirigido, no pseudografo, no multigrafo, acíclico, conexo, no árbol, no etiquetado



Ejemplos de grafos IV

- Un diagrama de estados, en donde los nodos son los estados y los arcos son las transiciones entre estados
  - En este ejemplo, el grafo sería dirigido, pseudografo, no multigrafo, no acíclico, conexo, no árbol, etiquetado con los eventos que producen los cambios de estado



Listas de adyacencia I

- Las dos cuestiones principales para implementar un grafo es plantearnos:
  - ¿Cómo almacenamos los nodos?
  - ¿Cómo almacenamos los arcos?
- Los nodos son un conjunto, por lo que podemos utilizar cualquier estructura que nos permita almacenar un conjunto
  - Ej: una lista o una tabla hash
- Lo problemático es almacenar los arcos. Tenemos dos opciones principales:
  - Listas de adyacencia (lo vemos ahora)
  - Matriz de adyacencia (lo vemos luego)

Listas de adyacencia II

- Las “listas de adyacencia” consiste en plantearnos implementar un grafo de forma similar a un árbol:
  - Una colección de nodos interconectados entre sí por punteros
    - Cada nodo tendría una lista de punteros
    - Cada puntero apunta a otro nodo (puede ser él mismo si estamos en un pseudografo)
    - Si un nodo A apunta a otro B, significa que hay un arco de A a B

Listas de adyacencia III

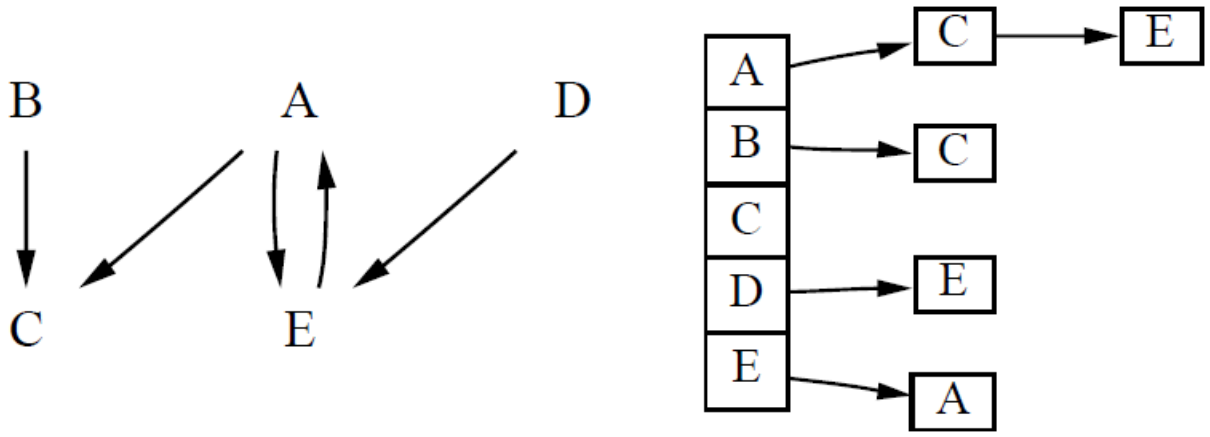
- Exactamente, lo que vamos a hacer en “listas de adyacencia” es lo siguiente:
  - Los nodos los almacenamos en una lista, generalmente contigua
  - Por cada nodo, le asociamos una lista (generalmente enlazada) con sus arcos salientes
    - Visto de otra manera: por cada nodo se guarda una lista enlazada con los arcos salientes, los cuales nos llevan inmediatamente a sus nodos “adyacentes”
      - De ahí el nombre “Lista de adyacencia”

Listas de adyacencia IV

- Cada elemento de dicha lista enlazada contendrá la información de un arco:
  - El nodo origen del arco no hace falta almacenarlo porque es el nodo en donde está la lista enlazada de arcos
  - El arco tendrá que contener el puntero, clave o índice del nodo destino
    - ¡OJO! No contiene el nodo destino en sí mismo, sino sólo su puntero, clave o índice
    - La información del nodo destino ya está almacenada en la lista contigua de nodos…
    - … y por lo tanto si lo almacenáramos de nuevo en el arco, sería información redundante
  - Si el grafo es etiquetado, el arco tendrá también que contener la información asociada al arco

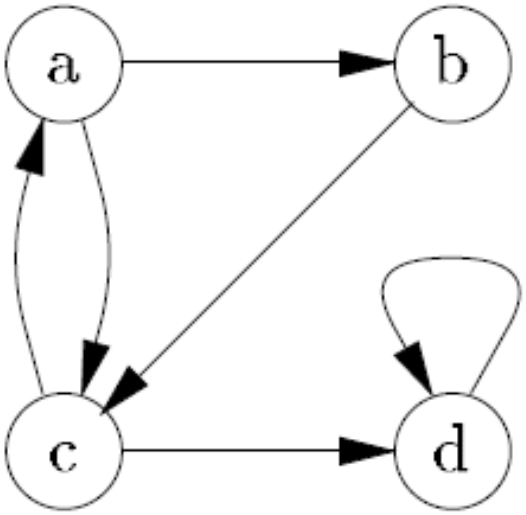
Listas de adyacencia V

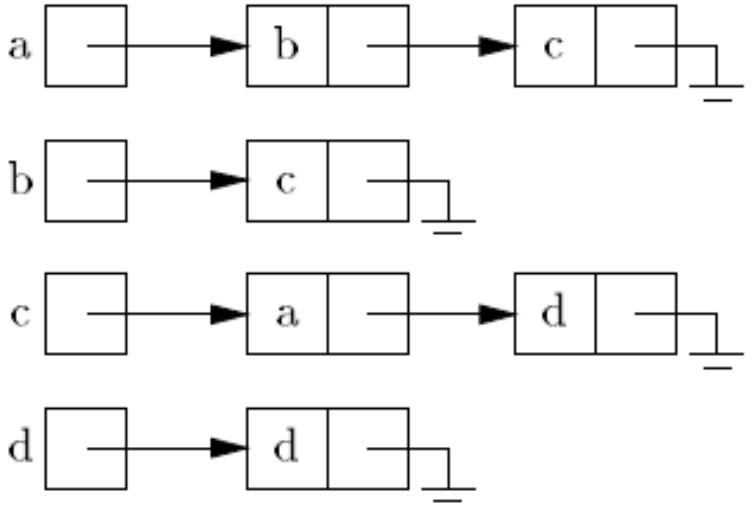
- Ejemplo con grafo no etiquetado (y por lo tanto cada arco sólo contiene la clave del nodo destino, que en este caso es una letra):



Listas de adyacencia VI

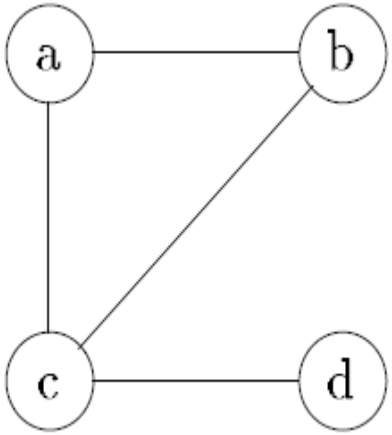
- Otro ejemplo, esta vez con un pseudografo no etiquetado, en el cual la clave de cada nodo es de nuevo una letra:

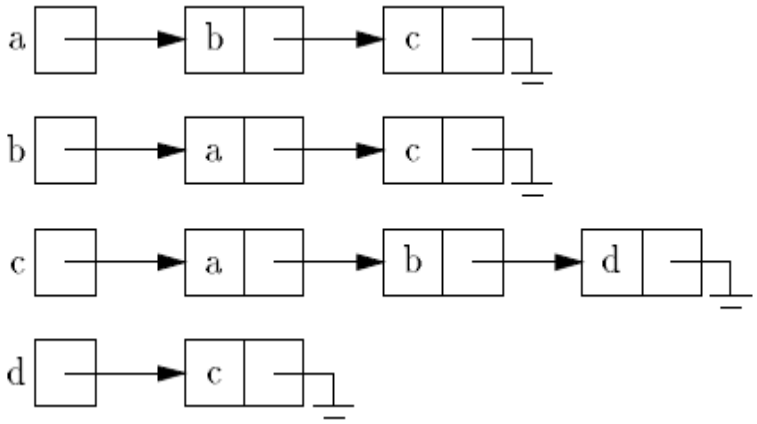




Listas de adyacencia VII

- Si el grafo es no dirigido, tendremos información redundante, salvo que sólo almacenemos el arco en uno de los sentidos.
  - Ejemplo, almacenando cada arco en ambos sentidos:



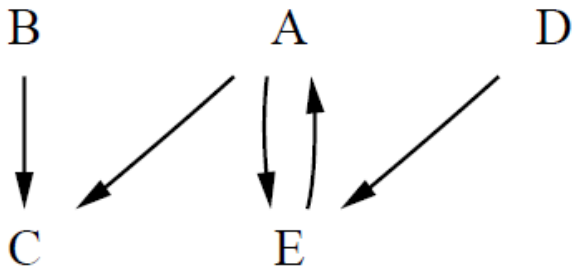


Listas de adyacencia VIII

- Si el grafo es dirigido, podría ser conveniente tener dos listas asociadas a cada nodo:
  - Una con los nodos a los que se puede llegar
  - Y otra con los nodos desde los que se puede llegar
- Aunque sea información redundante, podría facilitar mucho los algoritmos
- Practica ahora programando un grafo que contenga aeropuertos y rutas entre ellos, usando listas de adyacencia para implementar el grafo (EjercicioListasAdyacencia)

Matriz de adyacencia I

- En la “matriz de adyacencia” se guardan los arcos como casillas de una matriz bidimensional, ya que un arco es un par de nodos
  - Cada casilla es un valor booleano que indica si ese posible arco existe o no.
  - La fila indica el nodo de origen, y la columna indica el nodo destino



	A	B	C	D	E
A	0	0	1	0	1
B	0	0	1	0	0
C	0	0	0	0	0
D	0	0	0	0	1
E	1	0	0	0	0

Matriz de adyacencia II

- Si el grafo es etiquetado, cada casilla guardaría también la etiqueta, además del valor booleano de antes
- Si tenemos un multigrafo, en cada casilla tendríamos que tener un conjunto de arcos
- Si la matriz se almacena como un vector de vectores, podemos añadir, modificar y borrar arcos en O(1)
  - Suponiendo que sabemos de antemano la posición de la fila/columna en donde está el nodo
  - Sólo tenemos que ir a la casilla adecuada (es O(1) porque estamos haciendo la matriz con memoria contigua) y modificar el dato que hay allí (O(1)).
- Para encontrar todos los vecinos de un nodo necesitamos O(n)
  - Es recorrer los n elementos de la fila y ver cuáles tienen el bit a verdadero
  - Si queremos saber todos los nodos desde los que podemos llegar directamente a nuestro nodo, tendríamos que recorrer la columna en vez de la fila

Matriz de adyacencia III

- Insertar un nuevo nodo:
  - Como los nodos no mantienen ningún orden entre sí, podemos insertar la fila y columna asociadas al nuevo nodo en cualquier posición de la matriz
    - Lo más sencillo, al igual que la inserción en una lista contigua, es insertar la fila y la columna al final de la matriz
    - Ampliar la capacidad de la matriz es $O(n^2)$, pues, al igual que ampliar la capacidad en una lista contigua, en el peor caso implica copiar todos los datos (n^2 datos) a una nueva dirección de memoria más grande
  - Una vez insertada la fila y columna y ampliada la capacidad de la matriz, es necesario rellenar con falsos todas las casillas de la nueva columna y la nueva fila
    - Son 2n-1 casillas en total, así pues el relleno es O(n)
  - Por lo tanto, el algoritmo entero para insertar un nuevo nodo es $O(n^2)$
    - Es porque la T(n) total es un polinomio de grado 2, resultado de sumar dos polinomios: uno de grado 2 para la parte de ampliar la capacidad, y otro de grado 1 para la parte de inicializar las nuevas casillas

Matriz de adyacencia IV

- Eliminar un nodo:
  - Es también costoso, pues implica añadir/quitar una fila y una columna enteras, desplazando una posición hacia arriba las filas siguientes y una posición hacia la izquierda las columnas siguientes
    - $O(n^2)$ en el peor caso (eliminar la primera fila y columna)
  - Para mejorar el rendimiento y borrar en O(1), podríamos seguir la estrategia perezosa (“lazy”):
    - Nos limitados a marcar su fila y columna como borradas, en vez de eliminarlo realmente y tener que desplazar todo lo que hay después
    - De vez en cuando (por la noche, una vez a la semana, etc.) habrá que reestructurar para eliminar realmente todas las columnas y filas marcadas como borradas

Matriz de adyacencia V

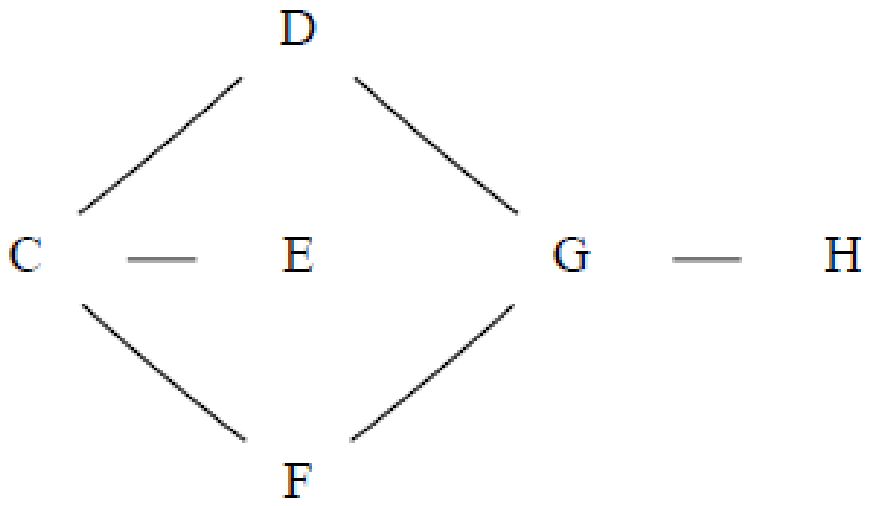
- El principal inconveniente de las matrices de adyacencia es el desperdicio de memoria...
  - ... ya que necesitamos una matriz con tantas casillas como posibles arcos podrían existir
  - Podrían existir n^2 arcos en un grafo dirigido, pseudografo y no multigrafo con n nodos
  - Normalmente los grafos no son tan “densos”
    - Es decir el número de arcos es muy inferior al número máximo posible de arcos que podría haber
      - En este caso, tenemos una “matriz dispersa”, en la cual la mayoría de sus casillas están a falso
    - Usar estrategias, vistas en el tema de listas, para almacenar matrices dispersas (ej: una lista de las casillas que están a verdadero)
  - Si el grafo es no dirigido, la matriz es simétrica y por lo tanto podríamos ahorrar memoria guardando sólo una de las dos mitades de la matriz
- Practica ahora con el EjercicioMatrizAdyacencia

Recorrido en profundidad I

- Vamos a ver a continuación varios tipos de recorridos en un grafo. Al igual que en árboles, el recorrido de un grafo es ordenar los nodos linealmente en una lista.
- Algoritmo del recorrido en profundidad (naturalmente recursivo):
  - Se selecciona cualquier nodo “v” como punto de partida, se marca como visitado, y se copia en la lista de salida
  - Para cada nodo adyacente a “v” no visitado, se toma como nuevo punto de partida y comenzamos de nuevo recursivamente
  - Si queda algún nodo sin visitar cuando termine el algoritmo, el grafo no es conexo
- La idea del recorrido en profundidad es: profundizar todo lo posible por un camino, regresar hasta el principio, coger otro camino y repetir sucesivamente hasta haber visitado todos los nodos conectados.

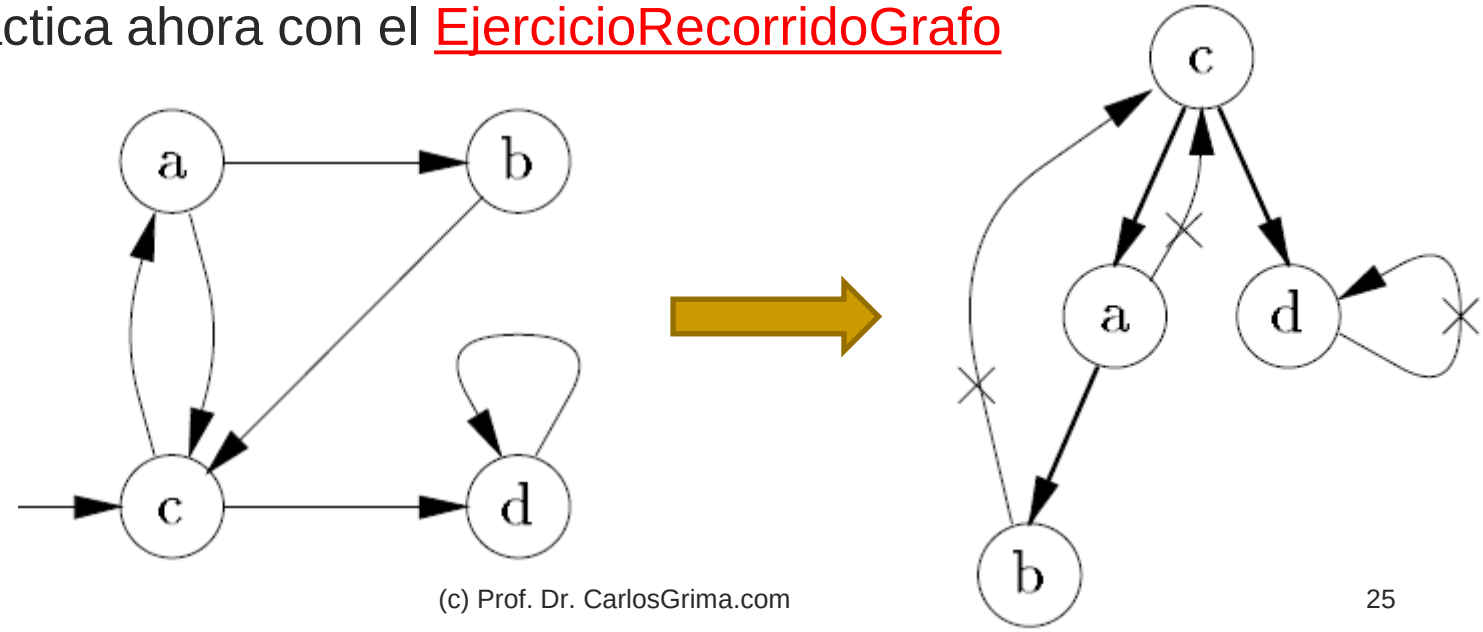
Recorrido en profundidad II

- En el siguiente grafo de ejemplo (no dirigido, por lo tanto cada arco tiene los dos sentidos) tenemos muchos posibles recorridos en profundidad. Algunos de ellos son:
  - C, D, G, H, F, E
  - C, E, D, G, F, H
  - E, C, D, G, H, F
  - H, G, D, C, E, F
  - H, G, D, C, F, E
  - Y muchos más…



Recorrido en profundidad III

- El recorrido en profundidad también sirve para convertir un grafo en un árbol
  - La raíz del nuevo árbol será el nodo por el que empecemos el recorrido.
    - En el ejemplo de abajo, empezamos por “C”
  - Los arcos por los que no pasamos los eliminamos del grafo.
- Practica ahora con el EjercicioRecorridoGrafo

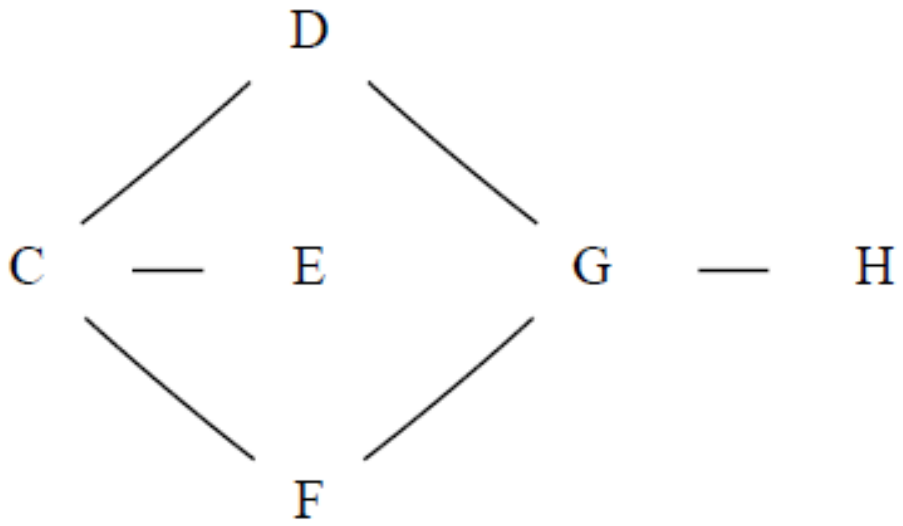


Recorrido en anchura I

- Es el mismo recorrido en anchura de un árbol, pero generalizado a un grafo
- Al igual que en profundidad, empezamos por cualquier nodo “v” y se saca a la lista de salida
- Cuando se llega a un nodo “v”, primero se visitan (y se sacan a la salida) todos los vecinos inmediatos de “v” que no hayan sido aún visitados
- Una vez visitados todos los vecinos inmediatos, se visitan y se sacan a la salida todos los adyacentes no visitados de cada uno de esos vecinos inmediatos, y así sucesivamente
- No es un algoritmo naturalmente recursivo

Recorrido en anchura II

- Algunos recorridos en anchura de este grafo de ejemplo:
  - C, D, E, F, G, H
  - C, E, F, D, G, H
  - E, C, F, D, G, H
  - H, G, D, F, C, E
  - Y muchos más…

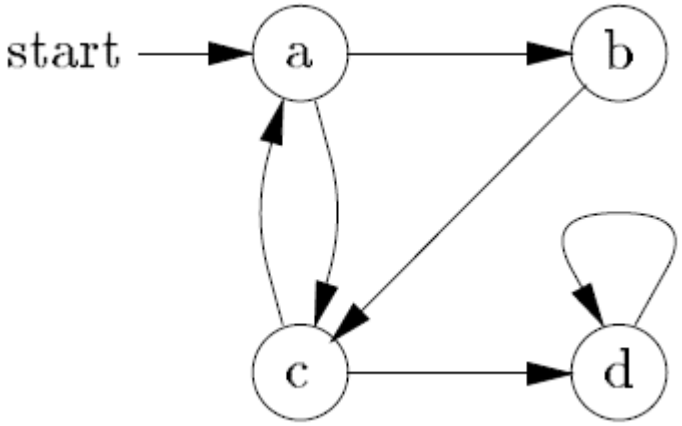


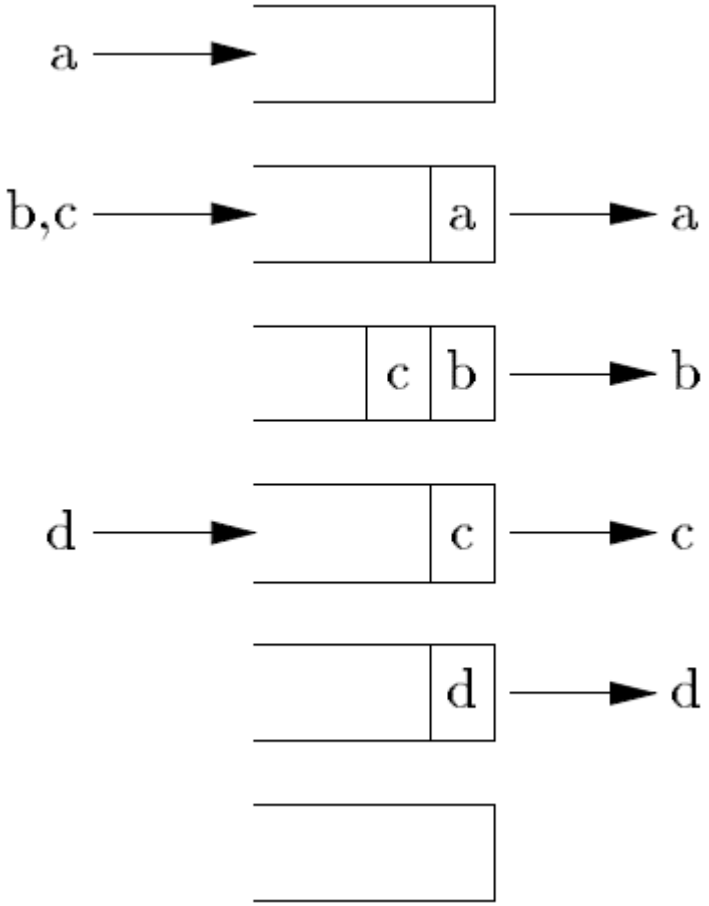
Recorrido en anchura III

- Podemos usar una cola para programar iterativamente el recorrido en anchura:
  - Se mete el nodo de origen en la cola y se marca como visitado
  - En cada paso iterativo:
    - Se saca de la cola un nodo (y se copia en la lista de salida)
    - Se meten en la cola todos los vecinos inmediatos no visitados del nodo que se acaba de sacar
    - Se marcan como visitados los nodos que acabamos de meter
  - Terminamos cuando la cola queda vacía

Recorrido en anchura IV

- Ejemplo de ejecución iterativa de recorrido en anchura, implementado con una cola:



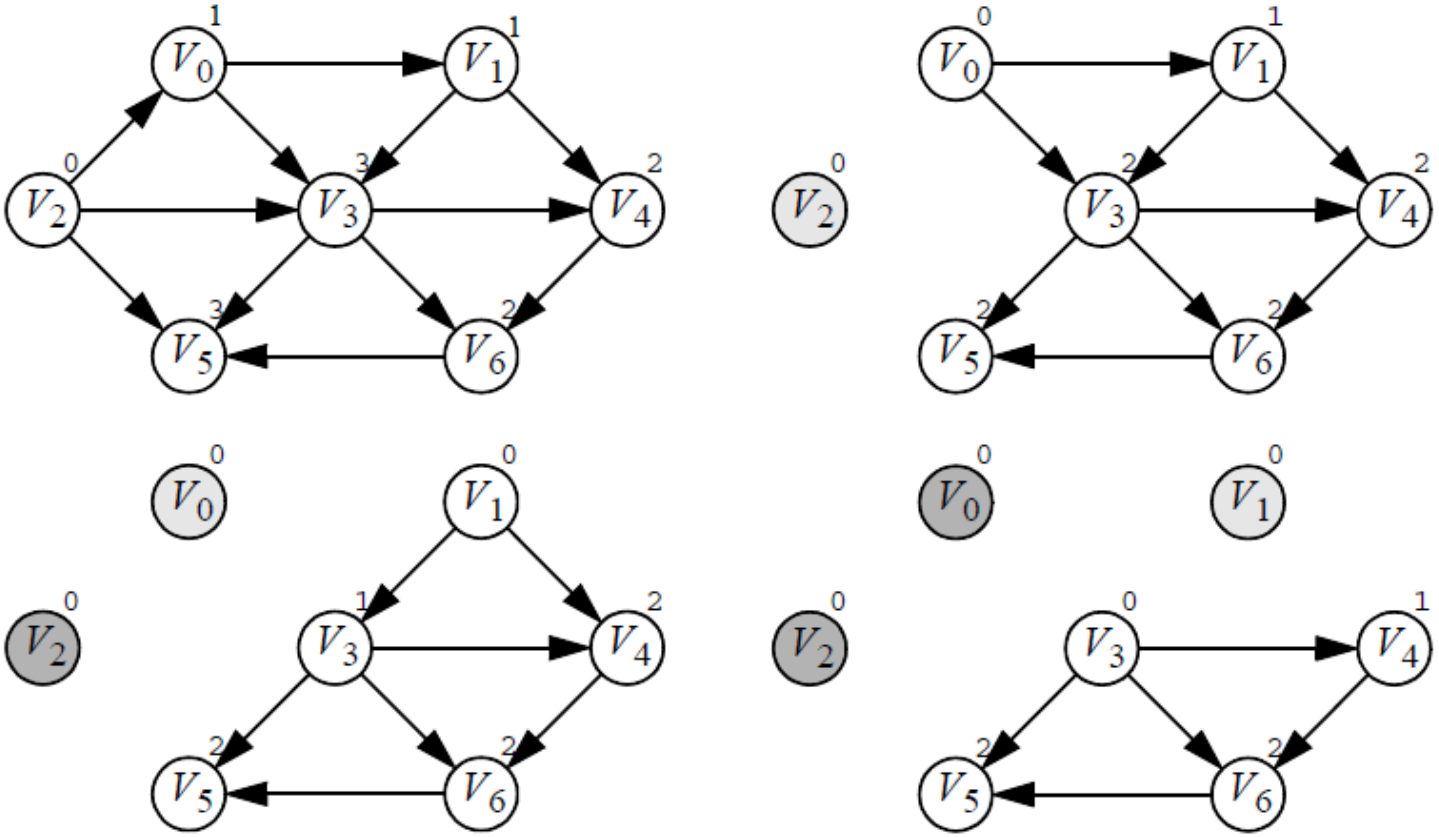


Recorrido topológico I

- Recorrido o clasificación topológica:
  - Ordenar los nodos en un GDA (grafo dirigido acíclico) de forma que si hay un camino de “u” a “v”, entonces “v” aparece después de “u” en la lista resultado
  - Algoritmo (naturalmente recursivo):
    - Encontrar un nodo sin arcos de entrada
    - Se envía a la lista de salida y se marca como borrado del grafo
    - Repetir

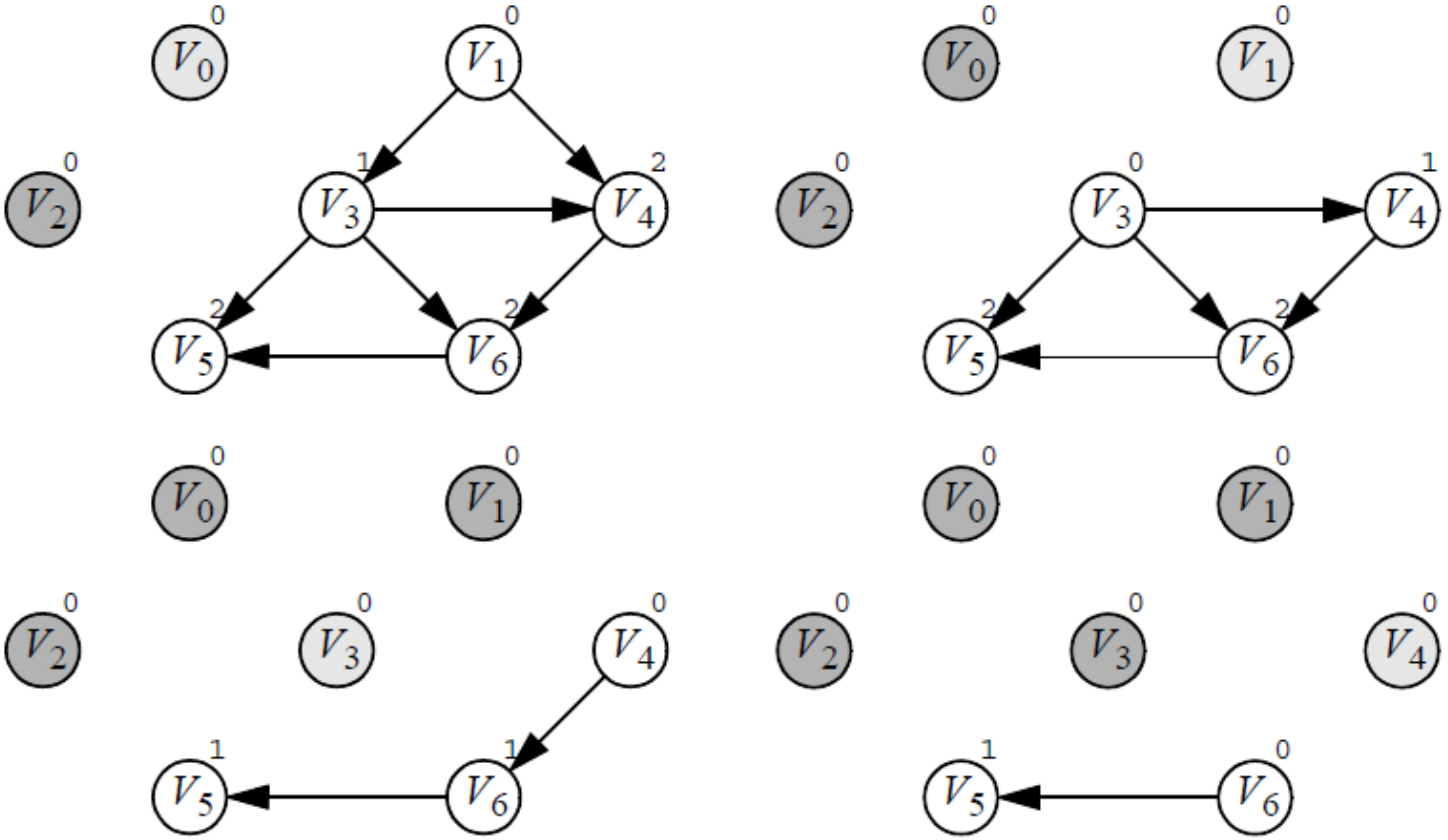
Recorrido topológico II

- Ejemplo de clasificación topológica:



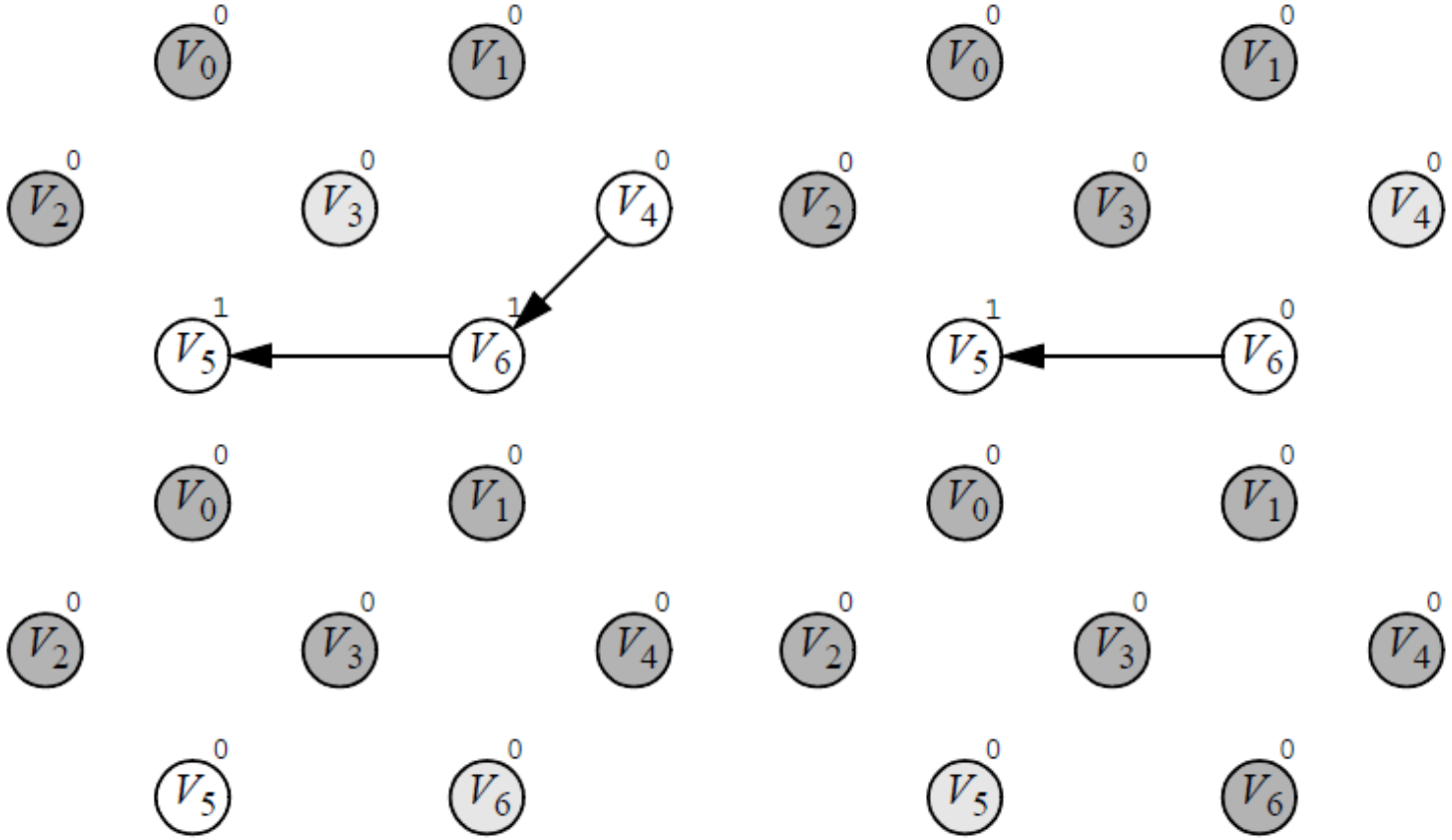
Recorrido topológico III

- Ejemplo de clasificación topológica (cont.):



Recorrido topológico IV

- Ejemplo de clasificación topológica (cont.):



Recorrido topológico V

- Implementación iterativa de la clasificación topológica, utilizando una cola. Por cada nodo, guardamos el número de arcos que llegan a él (lo llamamos “grado de entrada” del nodo)
  - Metemos en la cola los nodos de grado de entrada 0
  - Se saca de la cola un nodo, se marca como “borrado” y se envía a la lista de salida
    - Al marcarlo como borrado, eliminamos los arcos que salen de él
    - Al borrar los arcos que salen de él, tenemos que reducir el grado de entrada de todos sus vecinos inmediatos
    - Se meten en la cola los vecinos inmediatos que se queden con grado de entrada 0
  - Terminamos cuando la cola se queda vacía
    - Si la cola queda vacía y quedan nodos sin procesar, el grafo tiene algún ciclo
- Esta implementación es O(arcos), pues al final hay que eliminar todos los arcos del grafo