Orden topológico

Fuentes, una cola y el algoritmo de Kahn

Carlos A Delgado S

Pontificia Universidad Javeriana Cali

Septiembre de 2026

Objetivos de la sesión

Al terminar la clase, usted podrá

- Decir qué es un orden topológico de un grafo dirigido y reconocer cuándo un orden dado lo es.
- Explicar por qué un grafo con ciclos no tiene ninguno y por qué un grafo sin ciclos tiene al menos uno.
- Ejecutar a mano el algoritmo de Kahn, con su cola de fuentes, sobre un grafo pequeño.
- Implementarlo en $\Theta(V + E)$ y hacer que reporte cuando el grafo tiene ciclos.
- Llevar un problema de precedencias a un grafo y resolverlo con el algoritmo.

Contenido

1 Antes de empezar

2 Recuento

3 Orden topológico

4 El algoritmo de Kahn

5 Paso a paso sobre el plan de estudios

6 Un problema de juez

7 Errores comunes

8 Ejercicios

9 Referencias

Contenido

1 Antes de empezar

2 Recuento

3 Orden topológico

4 El algoritmo de Kahn

5 Paso a paso sobre el plan de estudios

6 Un problema de juez

7 Errores comunes

8 Ejercicios

9 Referencias

Preguntas sobre la tarea 3

Lo que hay que tener claro en cada problema

- Qué es un vértice y qué es una arista: en varios de los problemas el grafo no viene dibujado, hay que decidirlo.
- Qué recorrido contesta la pregunta: alcance, distancia en aristas, componentes, o algo que se calcula sobre el recorrido.
- Cómo se lee la entrada completa y cómo se detecta el final.

Lo de hoy

El orden topológico sirve en uno de los problemas de la tarea; cuál, se nota al leer qué pide.

Contenido

1 Antes de empezar

2 Recuento

3 Orden topológico

4 El algoritmo de Kahn

5 Paso a paso sobre el plan de estudios

6 Un problema de juez

7 Errores comunes

8 Ejercicios

9 Referencias

Lo que se usa hoy de un grafo dirigido

Grados

En un grafo dirigido $G = (V, E)$ cada vértice tiene dos grados: de **salida**, cuántas aristas salen de él, que es el largo de su lista de adyacencia; y de **entrada**, cuántas aristas llegan a él, que es cuántas veces aparece en las listas de los demás.

Los recorridos

Profundidad y amplitud visitan lo alcanzable desde el arranque, cada vértice una vez, y miran cada arista una vez por cada extremo. Sobre lista de adyacencia cuestan $\Theta(V + E)$. Hoy se usa esa misma cuenta.

Grados de entrada, en Python

Un recorrido por todas las listas: cada vez que v aparece en la lista de alguien, su contador sube.

Lo que se usa hoy de un grafo dirigido

```
def grados_de_entrada(G):
    # entrada[v]: cuantas aristas llegan a v
    n = len(G)
    entrada = [0] * n
    u = 0
    while u < n:
        for v in G[u]:
            entrada[v] = entrada[v] + 1
        u = u + 1
    return entrada
```

Contenido

1 Antes de empezar

2 Recuento

3 Orden topológico

4 El algoritmo de Kahn

5 Paso a paso sobre el plan de estudios

6 Un problema de juez

7 Errores comunes

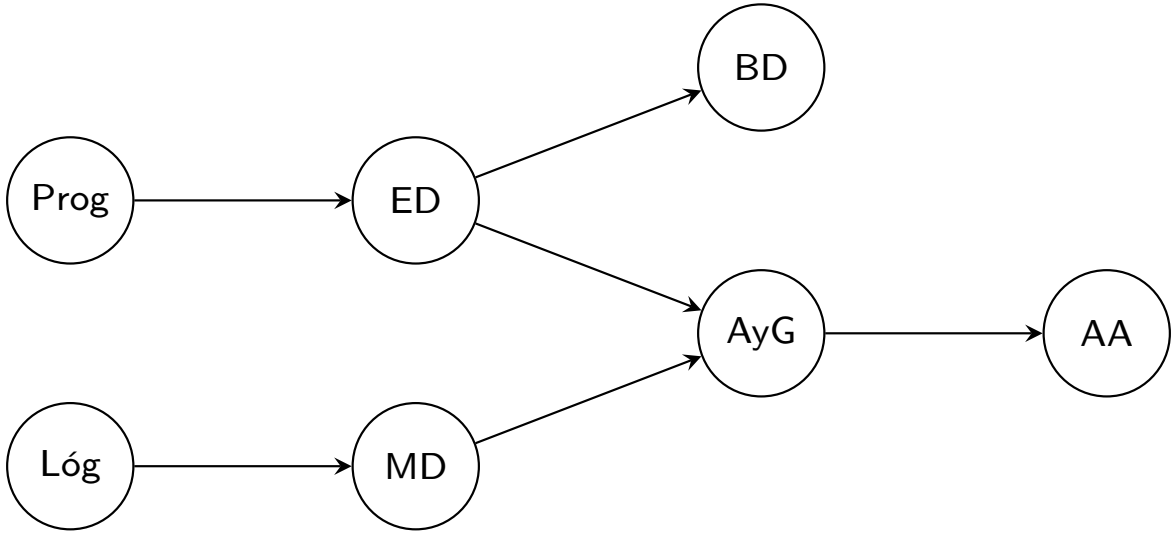
8 Ejercicios

9 Referencias

Un plan de estudios

El problema

Siete materias, y cada flecha es un prerrequisito: la de la cola se cursa antes que la de la punta. Hay que cursarlas de a una, en algún orden que no viole ningún prerrequisito. ¿Cuál?



Un plan de estudios

Las materias

Prog Programación, **Lóg** Lógica, **ED** Estructuras de datos, **MD** Matemáticas discretas, **BD** Bases de datos, **AyG** Árboles y grafos, **AA** Análisis de algoritmos.

Antes de cualquier algoritmo

¿Cuál puede ir de primera? ¿Puede Árboles y grafos ir de tercera? ¿Y de segunda? ¿Qué tienen en común las materias que pueden ir de primeras?

Por tanteo

Tres intentos

- Prog, ED, Lóg, MD, AyG, AA, BD: sirve. Cada materia aparece después de sus prerrequisitos.
- Lóg, MD, Prog, ED, BD, AyG, AA: también sirve, y es distinto.
- Prog, AyG, ED, . . . : no sirve desde la segunda posición. AyG tiene dos flechas que le llegan y ninguna de las dos materias ha pasado.

Por tanteo

Lo que se ve en los intentos

- Hay varios órdenes que sirven, no uno solo.
- Los que sirven empiezan por una materia a la que no le llega ninguna flecha.
- Una materia se puede poner cuando todas las que le llegan ya están puestas, y no antes.

La definición

Definición (Orden topológico, CLRS Sección 22.4, p. 612)

*Un **orden topológico** de un grafo dirigido $G = (V, E)$ es un orden lineal de todos sus vértices tal que, si G contiene la arista (u, v), entonces u aparece antes que v en el orden.*

Definición (Grafo dirigido acíclico)

*Un grafo dirigido es **acíclico** cuando no contiene ningún ciclo dirigido. En CLRS se abrevia dag.*

¿Entendimos el problema?

Entrada: un grafo dirigido G en lista de adyacencia. **Salida:** una lista con los n vértices, cada uno una sola vez, donde para toda arista (u, v) el vértice u está antes que v; o el aviso de que no existe tal lista.

La definición

El plan de estudios, con números

Los vértices se numeran 0 AyG, 1 Prog, 2 MD, 3 ED, 4 AA, 5 Lóg, 6 BD. Las listas de adyacencia: `[[4], [3], [0], [0, 6], [], [2], []]`. El primer intento de la lámina anterior es la salida `[1, 3, 5, 2, 0, 4, 6]`.

Cuándo no existe

Teorema

Si un grafo dirigido G contiene un ciclo, entonces G no tiene orden topológico.

Demostración

Se procede por contradicción. Supóngase que G contiene el ciclo $v_1 \to v_2 \to \cdots \to v_k \to v_1$ y que existe un orden topológico; sea $p(v)$ la posición de v en ese orden.

Desarrollo

Cada arista del ciclo obliga a que su cola vaya antes que su punta: $p(v_1) < p(v_2) < \cdots < p(v_k)$ por las aristas $(v_1, v_2), \ldots, (v_{k-1}, v_k)$, y $p(v_k) < p(v_1)$ por la arista (v_k, v_1). Encadenando, $p(v_1) < p(v_1)$, que es imposible.

Cuándo no existe

Conclusión

Por lo tanto, se puede concluir que un grafo con un ciclo dirigido no tiene orden topológico. Lo que sigue muestra el recíproco: sin ciclos siempre hay uno, y un algoritmo lo encuentra.

No es único

El plan de estudios tiene 22 órdenes válidos

Algunos: `1 3 5 2 0 4 6`, `1 5 3 2 6 0 4`, `5 2 1 3 6 0 4`, `5 1 2 3 0 6 4`, `1 3 6 5 2 0 4`. En todos, 1 y 5 van antes que 3 y 2; 3 y 2 antes que 0; 0 antes que 4; 3 antes que 6. Lo demás es libre.

Qué fija el grafo y qué no

El orden topológico ordena las parejas que una arista, o una cadena de aristas, conecta. Dos vértices sin camino entre ellos, como BD y MD, pueden ir en cualquier orden relativo. Cuando el enunciado de un problema dice *cualquier orden válido*, se refiere a esto; cuando pide uno en particular, hay que leer cuál.

Contenido

1 Antes de empezar

2 Recuento

3 Orden topológico

4 El algoritmo de Kahn

5 Paso a paso sobre el plan de estudios

6 Un problema de juez

7 Errores comunes

8 Ejercicios

9 Referencias

Por dónde se empieza

Definición (Fuente)

Un vértice con grado de entrada 0*: no le llega ninguna arista.*

La idea

Una fuente puede ir de primera, porque nada tiene que ir antes que ella. Y una vez puesta, se retira del grafo con sus aristas de salida; en lo que queda puede haber fuentes nuevas, y se repite. Lo que hace falta es que siempre haya una fuente que sacar mientras queden vértices.

Lema

Todo grafo dirigido acíclico con al menos un vértice tiene al menos una fuente.

Por dónde se empieza

Demostración

Se procede por contradicción: supóngase que G es acíclico, tiene $n \geq 1$ vértices y ninguno es fuente.

Desarrollo

Entonces a todo vértice le llega al menos una arista. Se toma cualquier w_0 y se camina hacia atrás: w_1 es un vértice con arista (w_1, w_0), w_2 uno con arista (w_2, w_1), y así. La caminata nunca se detiene, porque cada vértice tiene un predecesor. Tras n pasos se han nombrado $n+1$ vértices $w_0, \ldots, w_n$ y el grafo solo tiene n: alguno se repite, $w_i = w_j$ con $i < j$. El tramo $w_j \to w_{j-1} \to \cdots \to w_i$ es un ciclo dirigido, contra la hipótesis.

Por dónde se empieza

Conclusión

Por lo tanto, se puede concluir que todo grafo dirigido acíclico no vacío tiene una fuente. Y como quitar vértices no crea ciclos, lo que queda después de retirar una fuente sigue siendo acíclico y vuelve a tener una: el procedimiento no se atasca hasta vaciar el grafo.

La versión que uno escribe primero

KahnIngenuo(G)

1. Calcular $v.entrada$ para cada vértice v
2. Marcar todos los vértices como no emitidos; *orden* vacío
3. Buscar, recorriendo los vértices desde el primero, uno no emitido con $v.entrada = 0$
4. Mientras la búsqueda encuentre un vértice u:
   - 4.1 Agregar u al final de *orden* y marcarlo emitido
   - 4.2 Para cada v adyacente a u: $v.entrada = v.entrada - 1$
   - 4.3 Repetir la búsqueda del paso 3
5. Retornar *orden*

La versión que uno escribe primero

Lo que cuesta

Cada búsqueda recorre hasta n vértices y hay n búsquedas: $\Theta(n^2)$. Los decrementos suman m en total. En conjunto, $\Theta(n^2 + m)$: con 10^5 vértices son 10^{10} pasos.

La versión que uno escribe primero

```
def fuente_pendiente(entrada, emitido):
    # El primer vertice sin emitir con contador 0, o -1
    n = len(entrada)
    resultado = -1
    u = 0
    while u < n and resultado == -1:
        if not emitido[u] and entrada[u] == 0:
            resultado = u
        u = u + 1
    return resultado
```

La versión que uno escribe primero

```
def kahn_ingenuo(G):
    # Cada vuelta busca una fuente desde el principio de la lista
    n = len(G)
    entrada = grados_de_entrada(G)
    emitido = [False] * n
    orden = []
    u = fuente_pendiente(entrada, emitido)
    while u != -1:
        orden.append(u)
        emitido[u] = True
        for v in G[u]:
            entrada[v] = entrada[v] - 1
        u = fuente_pendiente(entrada, emitido)
    return orden
```

La versión que uno escribe primero

Sobre el plan de estudios

Devuelve `[1, 3, 5, 2, 0, 4, 6]`. Como la búsqueda arranca siempre desde el vértice 0, sale la fuente de número más bajo que haya en ese momento.

Dónde se pierde el tiempo

La búsqueda sobra

La búsqueda del paso 3 recorre la lista entera para encontrar una fuente que el algoritmo ya sabía dónde estaba: un vértice se vuelve fuente en el instante exacto en que su contador llega a 0, y eso pasa dentro del paso 4.2, con v en la mano.

El arreglo

En vez de buscarlas, las fuentes se guardan a medida que aparecen. Al arrancar, las que ya tienen contador 0; después, cada v cuyo contador llegue a 0 al restar. Una cola sirve: se saca por el frente y se agrega por el final, y cada vértice entra una sola vez, porque su contador llega a 0 una sola vez.

Dónde se pierde el tiempo

Kahn(G)

1. Calcular $v.entrada$ para cada vértice v
2. Para cada vértice v: si $v.entrada = 0$, agregar v a la cola P
3. *orden* vacío
4. Mientras la cola P no esté vacía:
    4.1 Retirar el frente u de P y agregarlo al final de *orden*
    4.2 Para cada v adyacente a u:
        4.2.1 $v.entrada = v.entrada - 1$
        4.2.2 Si $v.entrada = 0$: agregar v a P
5. Retornar *orden*

La implementación

```
from collections import deque

def kahn(G):
    # Las fuentes esperan en una cola
    n = len(G)
    entrada = grados_de_entrada(G)
    cola = deque()
    u = 0
    while u < n:
        if entrada[u] == 0:
            cola.append(u)
        u = u + 1
    orden = []
    while len(cola) > 0:
        u = cola.popleft()
        orden.append(u)
        for v in G[u]:
            entrada[v] = entrada[v] - 1
            if entrada[v] == 0:
                cola.append(v)
    return orden
```

La implementación

Dos detalles

Los vértices aislados tienen contador 0 desde el principio y entran a la cola en el paso 2: salen en el orden sin que nadie los mencione. Y el contador se resta *antes* de preguntar si llegó a 0; al revés, la fuente nueva se pierde.

Los colores de la ejecución

Qué dice cada color

- **Blanco:** vértice pendiente, con predecesores sin emitir. El número de al lado es su contador de entrada.
- **Naranja:** vértice en la cola. Su contador es 0: nada le falta, solo el turno.
- **Verde:** vértice ya emitido, que está en *orden*.

Un vértice pasa de blanco a naranja en el instante en que su contador llega a 0, y de naranja a verde cuando sale de la cola. Ninguno vuelve atrás.

Contenido

1 Antes de empezar

2 Recuento

3 Orden topológico

4 El algoritmo de Kahn

5 Paso a paso sobre el plan de estudios

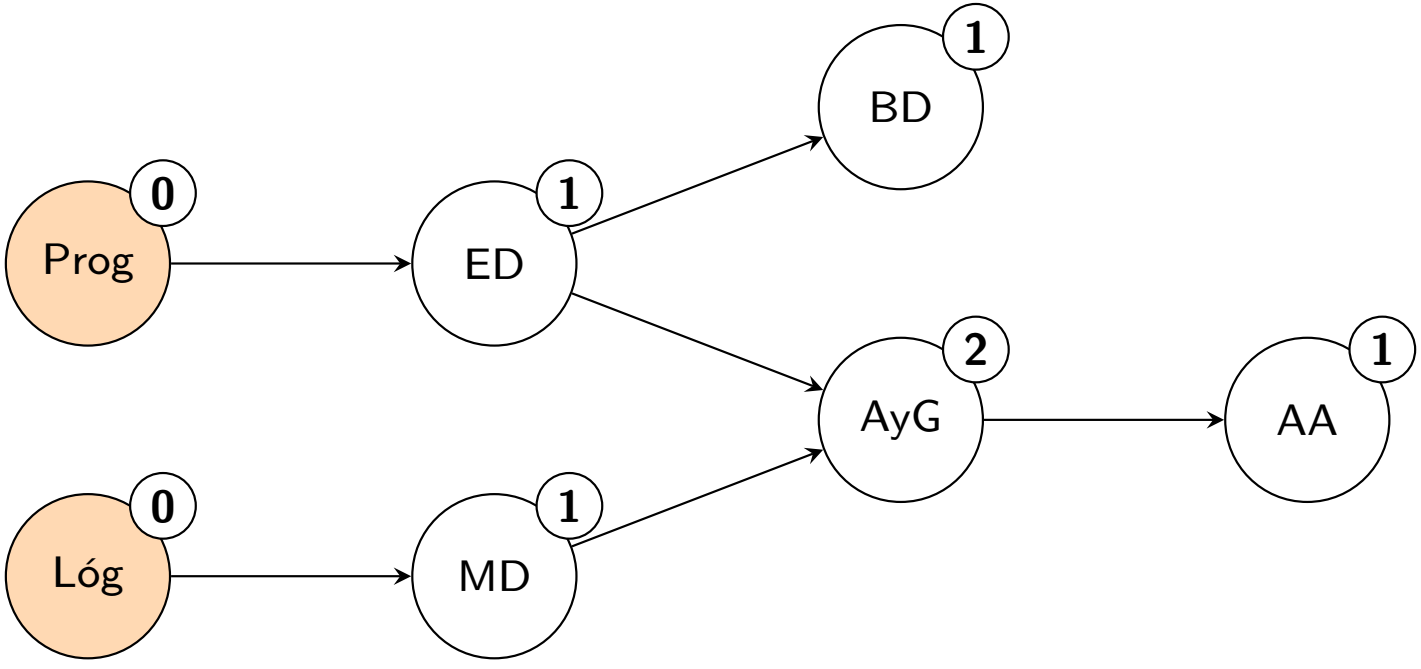
6 Un problema de juez

7 Errores comunes

8 Ejercicios

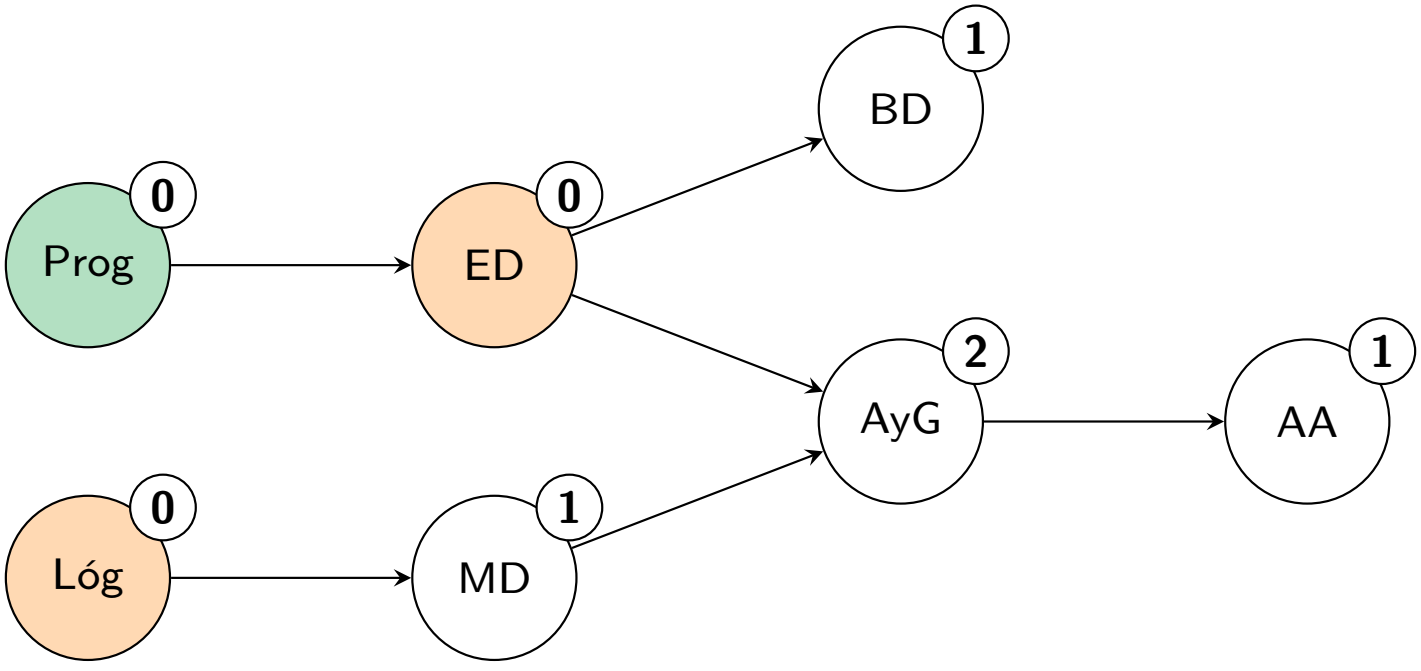
9 Referencias

El arranque



Contadores de entrada. Las dos fuentes, Prog y Lóg, entran a la cola.
Cola: Prog, Lóg *orden*: vacío

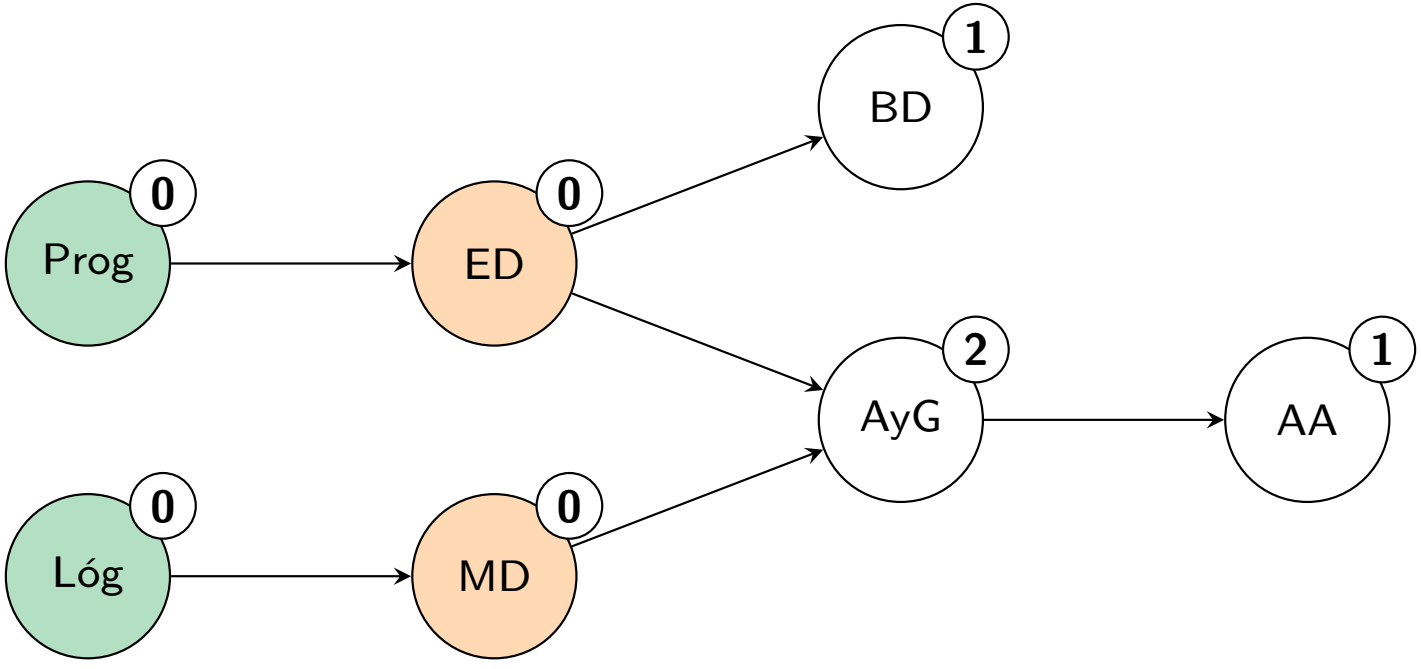
El arranque



Sale Prog. ED baja a 0 y entra a la cola.

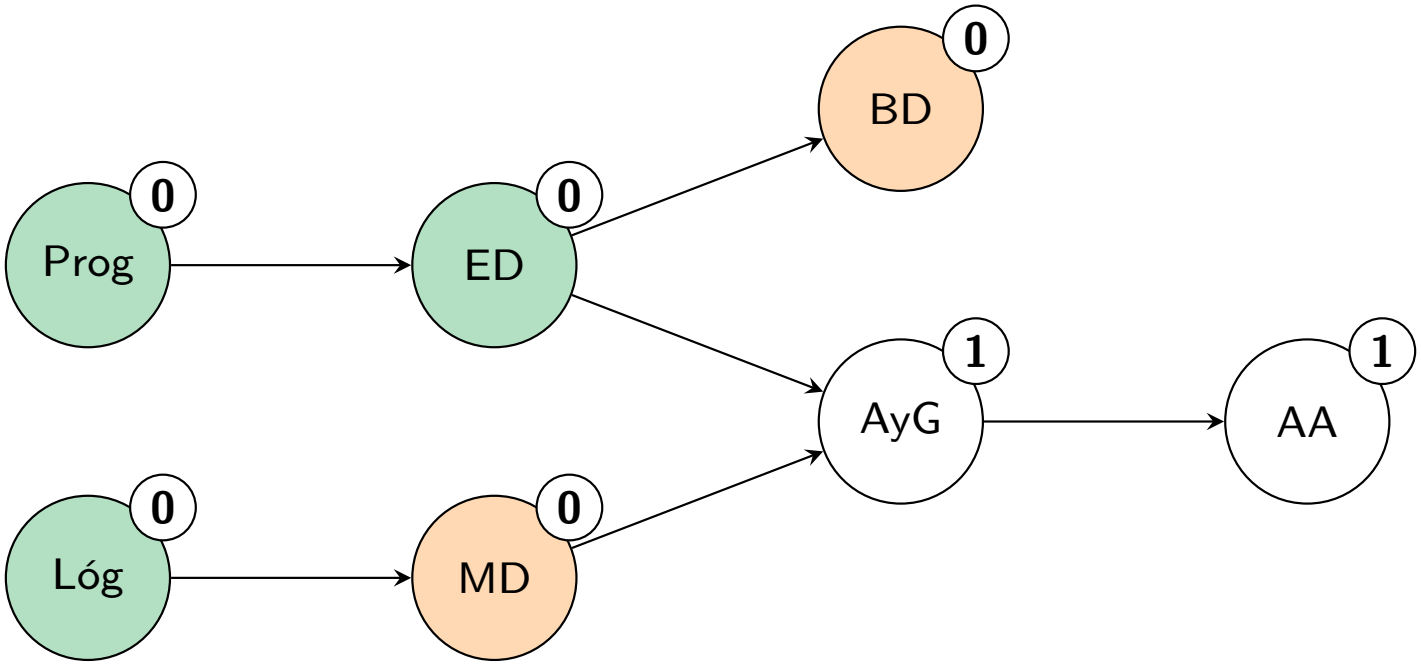
Cola: Lóg, ED *orden*: Prog

Salen las dos fuentes



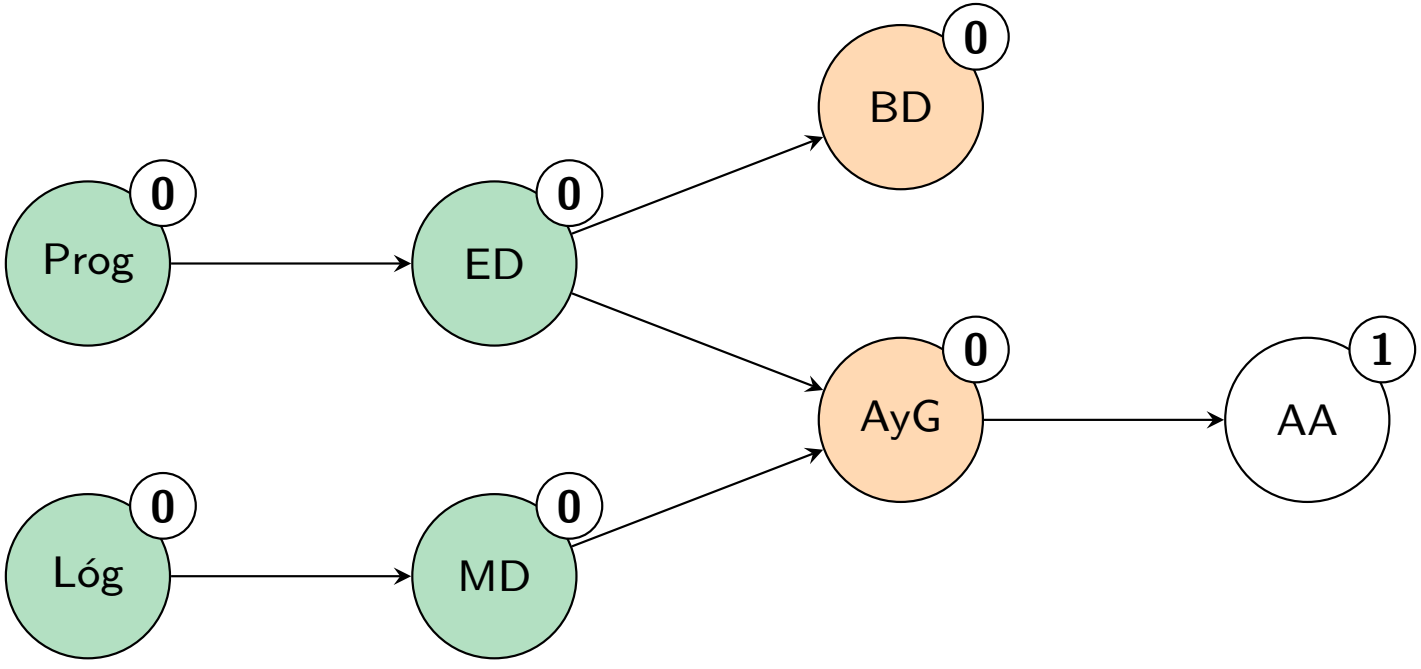
Sale Lóg. MD baja a 0 y entra a la cola.
Cola: ED, MD *orden*: Prog, Lóg

Salen las dos fuentes



Sale ED. AyG baja a 1; BD baja a 0 y entra a la cola.
Cola: MD, BD *orden*: Prog, Lóg, ED

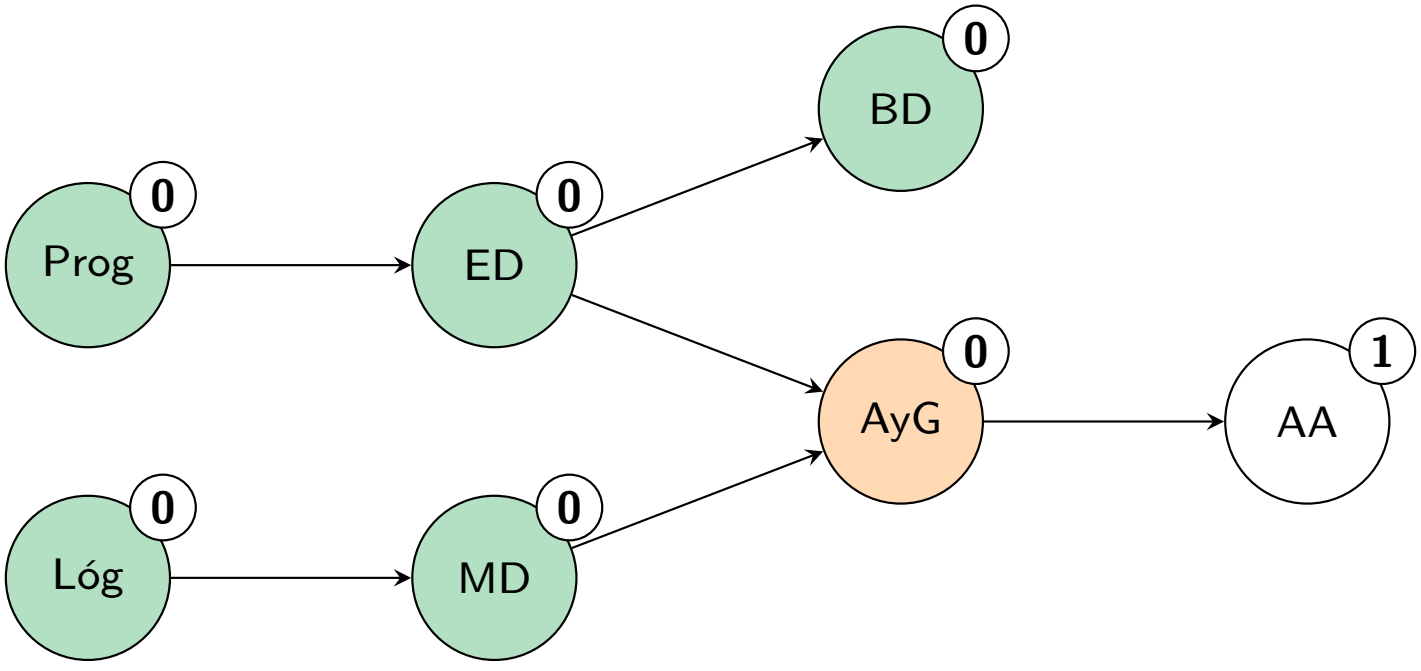
Se destapan ED y MD



Sale MD. AyG baja a 0 y entra a la cola.

Cola: BD, AyG *orden*: Prog, Lóg, ED, MD

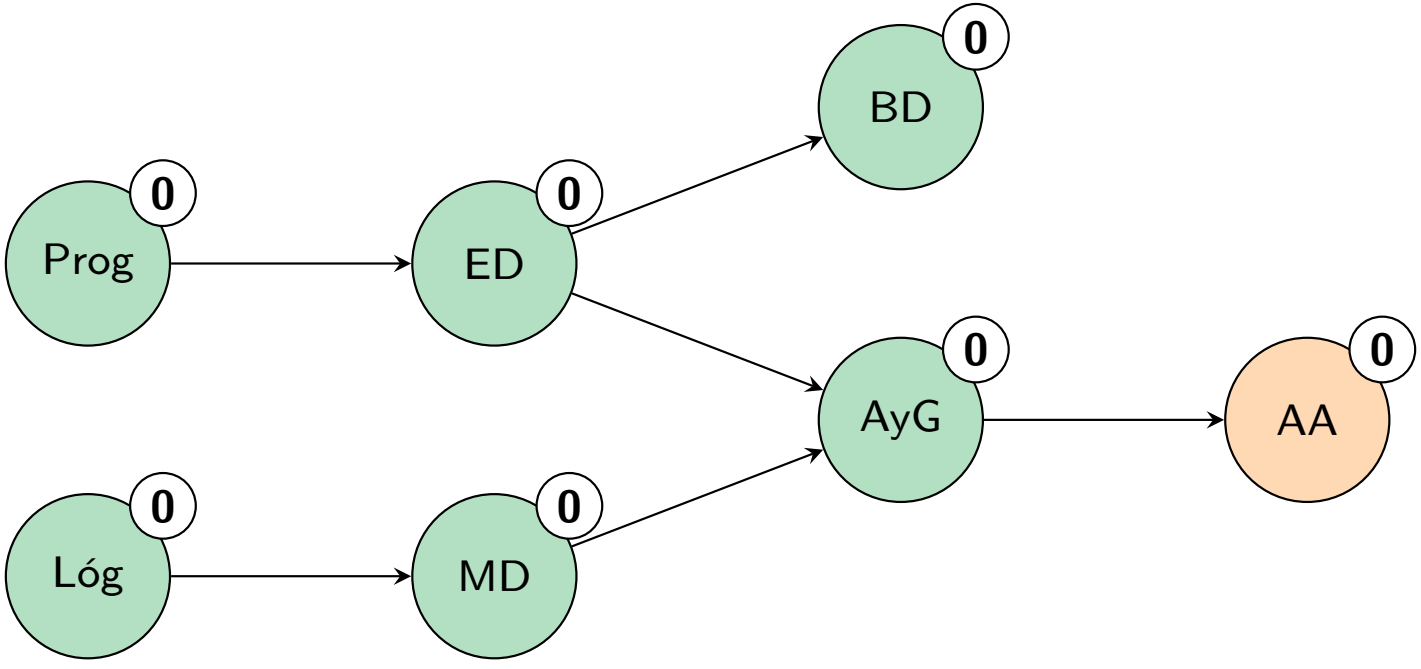
Se destapan ED y MD



Sale BD. No tiene sucesores.

Cola: AyG *orden*: Prog, Lóg, ED, MD, BD

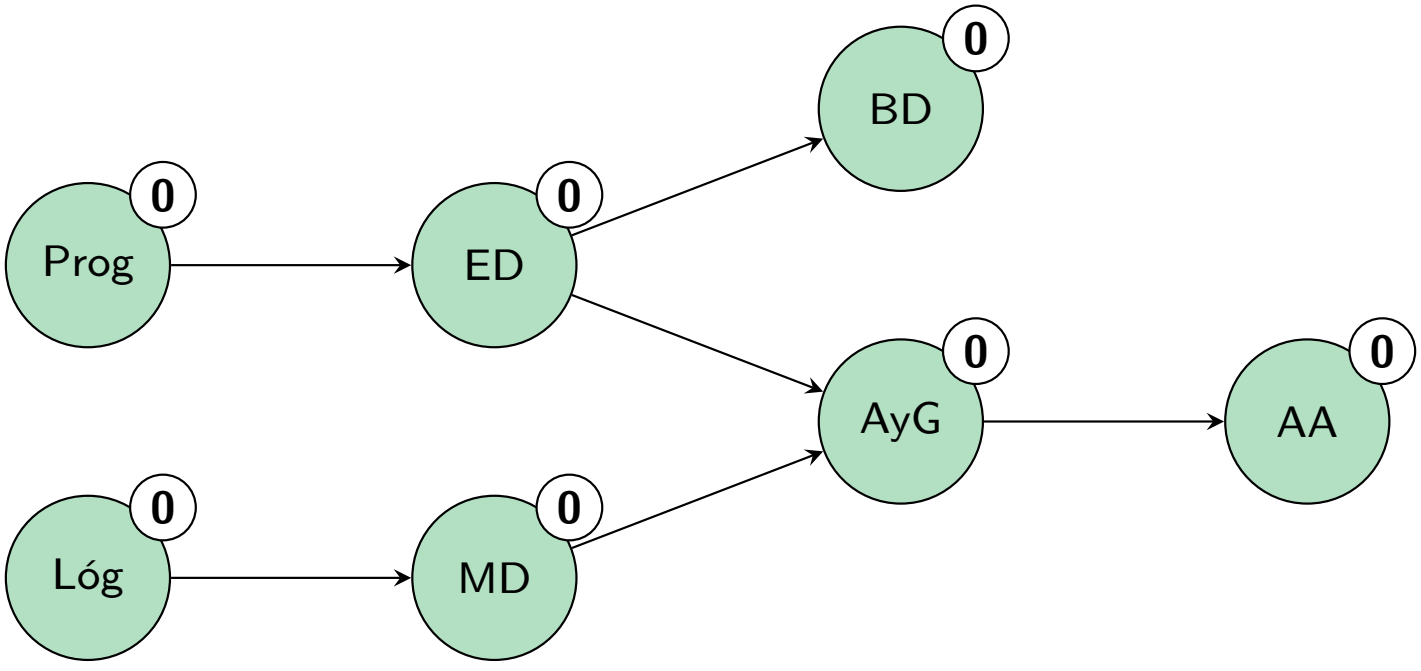
BD y AyG llegan a cero



Sale AyG. AA baja a 0 y entra a la cola.

Cola: AA *orden*: Prog, Lóg, ED, MD, BD, AyG

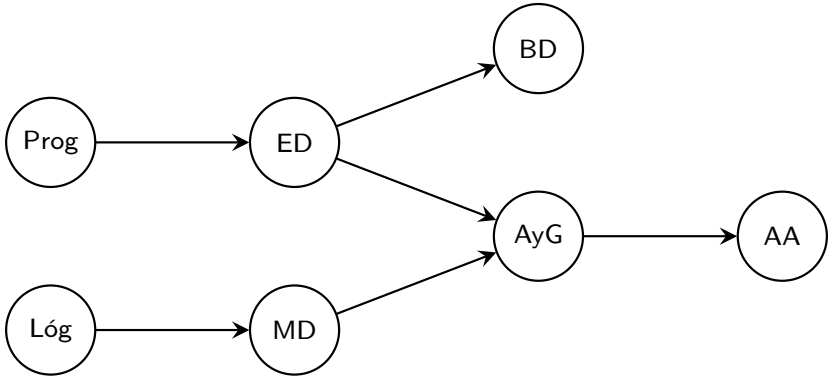
BD y AyG llegan a cero



Sale AA. No tiene sucesores.

Cola: vacía *orden*: Prog, Lóg, ED, MD, BD, AyG, AA

A mano sobre el plan de estudios



Contadores al arrancar

AyG 2, Prog 0, MD 1, ED 1, AA 1, Lóg 0, BD 1. Fuentes: Prog y Lóg, y entran a la cola en orden de número: $P = [\text{Prog}, \text{Lóg}]$.

A mano sobre el plan de estudios

La ejecución

Paso	Sale u	Contadores que bajan	Cola P después
1	Prog	ED $1 \to 0$, entra	Lóg, ED
2	Lóg	MD $1 \to 0$, entra	ED, MD
3	ED	AyG $2 \to 1$; BD $1 \to 0$, entra	MD, BD
4	MD	AyG $1 \to 0$, entra	BD, AyG
5	BD	ninguno	AyG
6	AyG	AA $1 \to 0$, entra	AA
7	AA	ninguno	vacía

orden = Prog, Lóg, ED, MD, BD, AyG, AA, que en números es `[1, 5, 3, 2, 6, 0, 4]`. Siete pasos para siete vértices.

A mano sobre el plan de estudios

Preguntas sobre la traza

¿Por qué AyG no entró a la cola en el paso 3, si ya había salido una de sus dos materias previas? ¿Qué habría pasado si los vértices se hubieran numerado al revés?

Por qué el orden que sale es topológico

Lo que sostiene el algoritmo

Para todo vértice v en la cola o ya emitido, todos los predecesores de v están en *orden*. Vale al arrancar, porque en la cola solo hay vértices sin predecesores. Y se conserva, porque v entra a la cola en el instante en que su contador llega a 0, y el contador bajó una vez por cada predecesor que salió de la cola, es decir, que ya está en *orden*.

Teorema

Si G es un grafo dirigido acíclico, Kahn(G) devuelve un orden topológico de G.

Demostración

Se procede de forma directa, a partir de lo que sostiene el algoritmo y del lema de la fuente.

Por qué el orden que sale es topológico

Desarrollo

Sea (u, v) una arista. Cuando v sale de la cola, todos sus predecesores, u entre ellos, ya están en *orden*: u aparece antes que v. Falta que salgan todos: mientras quede algún vértice sin emitir, los no emitidos forman un subgrafo acíclico no vacío, que por el lema tiene una fuente; su contador es 0 y está en la cola, así que la cola no se vacía antes de emitir los n.

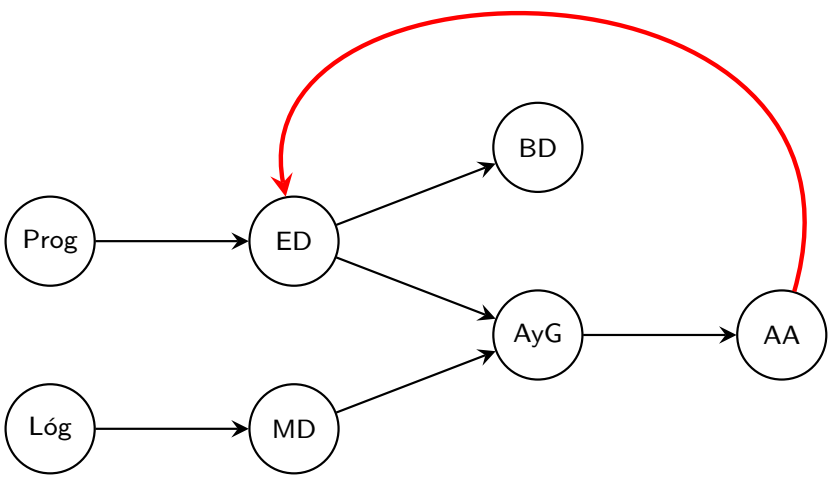
Conclusión

Por lo tanto, se puede concluir que *orden* contiene los n vértices y respeta cada arista: es un orden topológico de G.

Cuando hay un ciclo

Un prerrequisito de más

Al plan se le agrega la flecha AA $\rightarrow$ ED: Análisis de algoritmos antes que Estructuras de datos. Ahora ED $\rightarrow$ AyG $\rightarrow$ AA $\rightarrow$ ED es un ciclo. Las listas quedan `[[4], [3], [0], [0, 6], [3], [2], []]`.



Cuando hay un ciclo

La ejecución

Contadores: AyG 2, Prog 0, MD 1, ED 2, AA 1, Lóg 0, BD 1.

Paso	Sale u	Contadores que bajan	Cola P después
1	Prog	ED $2 \to 1$	Lóg
2	Lóg	MD $1 \to 0$, entra	MD
3	MD	AyG $2 \to 1$	vacía

La cola se vacía con *orden* = Prog, Lóg, MD: tres de siete. Quedan por fuera ED, AyG y AA, que forman el ciclo, y BD, que no está en el ciclo pero depende de él.

Teorema

Kahn(G) emite los n vértices de G si y solo si G no tiene ciclos dirigidos.

Cuando hay un ciclo

Demostración

Se procede en las dos direcciones. La vuelta, que sin ciclos salen los n, ya está: es la segunda parte del desarrollo anterior.

Desarrollo de la ida

Supóngase que G tiene el ciclo $v_1 \to \cdots \to v_k \to v_1$ y que algún vértice del ciclo sale de la cola; sea w el primero en salir. Su predecesor en el ciclo no había salido cuando w entró a la cola, luego el contador de w seguía en al menos 1 y w no pudo entrar: contradicción. Ningún vértice del ciclo sale, y *orden* queda con menos de n.

Cuando hay un ciclo

Conclusión

Por lo tanto, se puede concluir que `len(orden) < n` es exactamente la señal de que G tiene un ciclo, y el mismo algoritmo contesta las dos preguntas: el orden cuando existe, y que no existe cuando no.

Reportar el ciclo

```
def tiene_ciclo(G):
    # Kahn emite todos los vertices exactamente cuando no hay ciclos
    return len(kahn(G)) < len(G)
```

Y los que quedan por fuera

Los vértices que no están en `orden` son los de los ciclos y los que tienen algún camino desde un ciclo. En un sistema de módulos serían los que no se pueden compilar; en el plan de estudios, las materias que nadie podría cursar nunca.

Lo que cuesta

Teorema

Kahn(G) *sobre un grafo* $G = (V, E)$ *en lista de adyacencia cuesta* $\Theta(V + E)$.

Demostración

Se procede contando por separado lo que aporta cada vértice y lo que aporta cada arista, como en el costo de los recorridos.

Desarrollo

Los contadores de entrada se calculan recorriendo todas las listas una vez: $\Theta(V + E)$. Cada vértice entra a la cola a lo sumo una vez, porque su contador llega a 0 una sola vez, y sale una vez: $\Theta(V)$. Al salir u se recorre su lista, de $\delta^+(u)$ entradas, y la suma de los grados de salida es $|E|$: $\Theta(E)$ en decrementos. Sumando, $\Theta(V + E)$.

Lo que cuesta

Conclusión

Por lo tanto, se puede concluir que el algoritmo cuesta $\Theta(V + E)$, lo mismo que un recorrido, y que la cola es lo que separa esa cuenta de la $\Theta(V^2 + E)$ de la versión que busca la fuente cada vez.

Contenido

1 Antes de empezar

2 Recuento

3 Orden topológico

4 El algoritmo de Kahn

5 Paso a paso sobre el plan de estudios

6 Un problema de juez

7 Errores comunes

8 Ejercicios

9 Referencias

Ordering Tasks

UVa 10305 — Ordering Tasks

Enunciado: `https://vjudge.net/problem/UVA-10305`
Envío: `https://vjudge.net/problem/UVA-10305`

El problema

John tiene n tareas, numeradas de 1 a n, y m relaciones de precedencia: la tarea i debe ejecutarse antes que la j. Hay que imprimir un orden posible de ejecución.

Entrada y salida

Varias instancias; cada una empieza con n y m ($1 \leq n \leq 100$), seguidos de m líneas con i y j. Termina con $n = m = 0$. Por instancia, una línea con los n números. Con $n = 5$, $m = 4$ y las parejas $(1, 2)$, $(2, 3)$, $(1, 3)$, $(1, 5)$, la salida de la muestra es `1 4 2 5 3`.

Ordering Tasks

¿Entendimos el problema?

Vértices: las tareas. **Aristas:** (i, j) por cada precedencia. **Pregunta:** un orden topológico, cualquiera. El enunciado garantiza que existe, así que no hace falta reportar ciclos. Los números vienen desde 1; el programa los pasa a desde 0 al leer y los devuelve al imprimir.

Ordering Tasks

```
def leer_grafo(datos, p, n, m):
    # Las m parejas desde datos[p]; devuelve (G, posicion siguiente)
    G = []
    u = 0
    while u < n:
        G.append([])
        u = u + 1
    k = 0
    while k < m:
        G[int(datos[p]) - 1].append(int(datos[p + 1]) - 1)
        p = p + 2
        k = k + 1
    return (G, p)
```

Ordering Tasks

```
def leer_y_resolver(entrada):
    datos = entrada.read().split()
    p = 0
    salida = []
    n = int(datos[p])
    m = int(datos[p + 1])
    p = p + 2
    while n != 0 or m != 0:
        leido = leer_grafo(datos, p, n, m)
        G = leido[0]
        p = leido[1]
        linea = []
        for u in kahn(G):
            linea.append(str(u + 1))
        salida.append(" ".join(linea))
        n = int(datos[p])
        m = int(datos[p + 1])
        p = p + 2
    print("\n".join(salida))
```

Ordering Tasks

Sobre la muestra

Contadores: 1:0, 2:1, 3:2, 4:0, 5:1. Cola inicial $[1, 4]$; sale 1 y entran 2 y 5; sale 4; sale 2 y entra 3; sale 5; sale 3. Imprime `1 4 2 5 3`, que es la salida de la muestra. El juez acepta cualquier orden válido; que coincida es solo porque la cola saca las fuentes en el orden en que aparecen.

Contenido

1 Antes de empezar

2 Recuento

3 Orden topológico

4 El algoritmo de Kahn

5 Paso a paso sobre el plan de estudios

6 Un problema de juez

7 Errores comunes

8 Ejercicios

9 Referencias

Los que más cuestan

Sobre el algoritmo

- Preguntar si el contador es 0 antes de restar: la fuente nueva nunca entra a la cola.
- Restar el contador de v cada vez que se *ve* a v en vez de una vez por arista: con aristas repetidas en la entrada, el contador se calcula con las repeticiones y se resta con ellas, y cuadra; si se descartan al leer, hay que descartarlas en las dos partes.
- Marcar visitados en lugar de contar entradas: un vértice con dos predecesores saldría apenas sale el primero.
- Olvidar que un vértice sin aristas también va en el orden.

Los que más cuestan

Sobre el problema

- Dar por hecho que el orden es único, o que el juez espera el mismo que dio la muestra. Se lee qué pide: cualquiera, o uno en particular.
- Dar por hecho que no hay ciclos cuando el enunciado no lo garantiza: `len(orden) < n` es la comprobación y cuesta una línea.
- Poner la arista al revés: *i antes que j* es (i, j), de i hacia j.

Contenido

1 Antes de empezar

2 Recuento

3 Orden topológico

4 El algoritmo de Kahn

5 Paso a paso sobre el plan de estudios

6 Un problema de juez

7 Errores comunes

8 Ejercicios

9 Referencias

Tres que pueden aparecer en el parcial

1. ¿Tiene ciclos?

Escriba `tiene_ciclo(G)` usando búsqueda en profundidad, sin el algoritmo de Kahn. Entrada: un grafo dirigido G en lista de adyacencia. Salida: `True` si G tiene algún ciclo dirigido, `False` si no. Sobre `[[4], [3], [0], [0, 6], [], [2], []]` responde `False`; sobre `[[4], [3], [0], [0, 6], [3], [2], []]` responde `True`.

Cómo atacarlo: la marca de visitado no alcanza, porque llegar a un vértice ya visitado por otro camino no es un ciclo. Cada vértice pasa por tres estados: sin visitar, en proceso (su llamada no ha terminado) y terminado. Piense qué significa toparse con uno en proceso.

Tres que pueden aparecer en el parcial

2. Kahn con una pila

En $Kahn(G)$ se cambia la cola por una pila: las fuentes se apilan y sale la última que entró. ¿Sigue devolviendo un orden topológico? Justifíquelo con lo que sostiene el algoritmo. ¿Cuál orden devuelve sobre el plan de estudios? Ejecútelo a mano y compárelo con `[1, 5, 3, 2, 6, 0, 4]`.

Tres que pueden aparecer en el parcial

3. El menor de todos

Modifique *Kahn*(G) para que, entre todos los órdenes topológicos, devuelva el menor en orden lexicográfico: el que tiene el menor primer vértice; entre los que empatan, el menor segundo, y así. Sobre el plan de estudios debe devolver `[1, 3, 5, 2, 0, 4, 6]`. Diga qué estructura reemplaza a la cola y cuánto cuesta el algoritmo con ella.
Cómo atacarlo: la versión ingenua ya lo devuelve, porque busca desde el vértice 0. La pregunta es cómo sacar siempre la fuente de número más bajo sin recorrer la lista entera.

Dos más para el juez

UVa 11686 — Pick up sticks

Enunciado: `https://vjudge.net/problem/UVA-11686`
Envío: `https://vjudge.net/problem/UVA-11686`
n palitos y m parejas a, b que dicen que el palito a está encima del b, con n, m hasta un millón; termina con `0 0`. Hay que listar un orden para recogerlos sin levantar nunca uno que tenga otro encima, o imprimir `IMPOSSIBLE`.
Cómo atacarlo: es la segunda variante de hoy, con `len(orden) < n` decidiendo la palabra. Con un millón de líneas se lee todo de una vez con `sys.stdin.read()`.

Dos más para el juez

UVa 11060 — Beverages

Enunciado: `https://vjudge.net/problem/UVA-11060`
Envío: `https://vjudge.net/problem/UVA-11060`
N bebidas con nombre y M parejas B_1 *antes que* B_2; hay que imprimir un orden para beberlas, y cuando dos no están relacionadas va primero la que apareció primero en la entrada.
Cómo atacarlo: es el ejercicio 3 con los nombres numerados por orden de aparición. La salida tiene un formato con `Case #` y una línea en blanco después de cada caso; se copia del enunciado letra por letra.

Lo que sigue

El árbol de la búsqueda en profundidad

Kahn no es la única forma de ordenar un grafo acíclico. La búsqueda en profundidad deja en cada vértice dos marcas de tiempo, la de descubrimiento y la de finalización, y con ellas clasifica las aristas del grafo en cuatro tipos. Una de esas clases, la arista hacia un ancestro, es exactamente un ciclo, y el orden de finalización, leído al revés, es un orden topológico.

Para investigar

CLRS, Sección 22.3: qué son $v.d$ y $v.f$, qué dice el teorema del paréntesis y cuáles son los cuatro tipos de arista. Traiga una ejecución de la búsqueda en profundidad sobre el plan de estudios con los dos tiempos anotados en cada vértice.

Contenido

1 Antes de empezar

2 Recuento

3 Orden topológico

4 El algoritmo de Kahn

5 Paso a paso sobre el plan de estudios

6 Un problema de juez

7 Errores comunes

8 Ejercicios

9 Referencias

Referencias

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. *Introduction to Algorithms*, 3rd ed. MIT Press, 2009. Sección 22.4 (pp. 612–615) y Ejercicio 22.4-5 (p. 615), que es el algoritmo de hoy.
- Kahn, A. B. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, 1962.
- Halim, S., Halim, F., Effendy, S. *Competitive Programming 4*. Lulu, 2020. Sección 4.2.5, orden topológico.
- Universidad de Valladolid Online Judge: problemas 10305, 11686 y 11060. `https://vjudge.net`