Algoritmos codiciosos

Clase 17

IIC 2133 - Sección 2

Prof. Mario Droguett

Sumario

Componentes fuertemente conectadas

Definición

Sea G un grafo dirigido. Una **componente fuertemente conectada (CFC)** es un conjunto maximal de nodos $C \subseteq V(G)$ tal que dados $u, v \in C$, existe un camino dirigido desde u hasta v

Proposición

Si G es cíclico y los nodos de $B \subseteq V(G)$ forman un ciclo, entonces existe una componente fuertemente conectada C tal que $B \subseteq C$

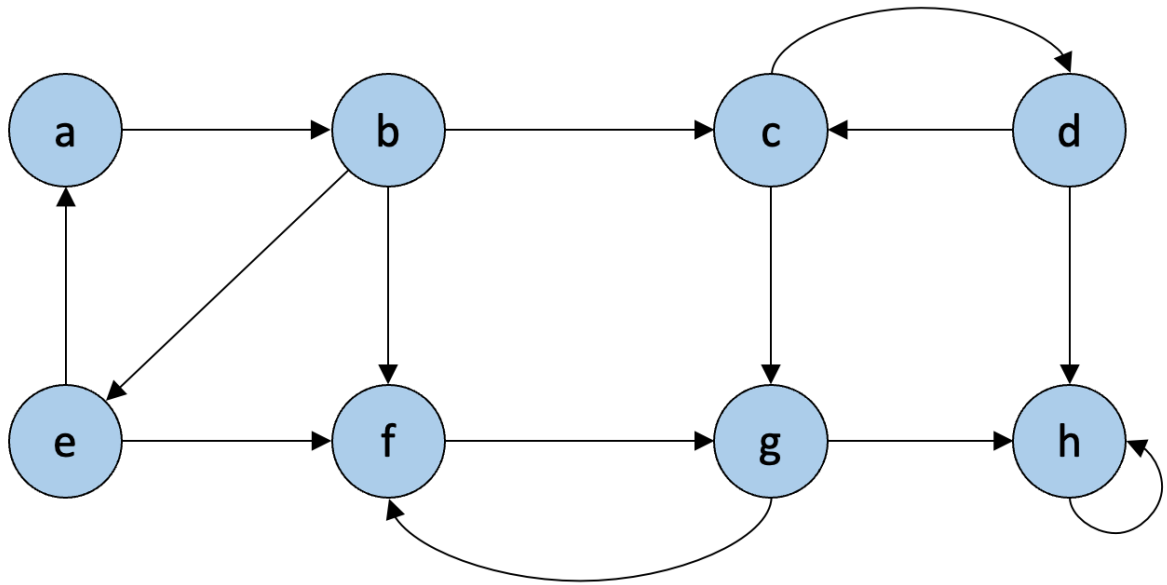
Los nodos de un ciclo pertenecen a la misma CFC

Proposición

Un grafo G acíclico tiene 0 componentes fuertemente conectadas

Componentes fuertemente conectadas

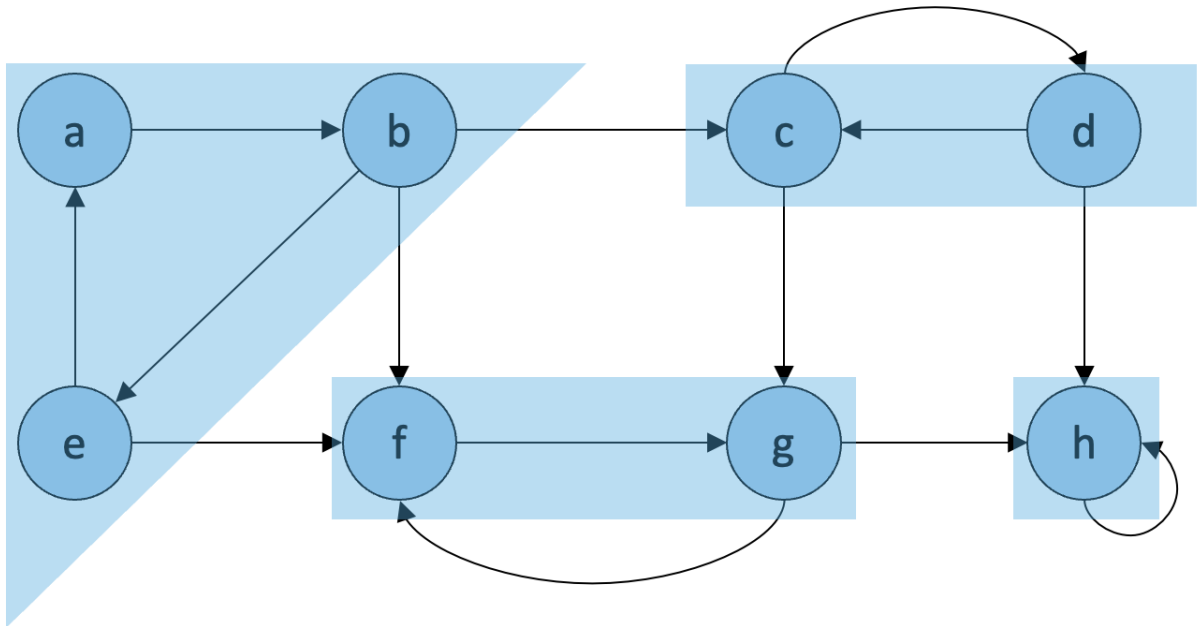
Consideremos el siguiente grafo dirigido cíclico



¿Cuáles son las componentes fuertemente conectadas de G?

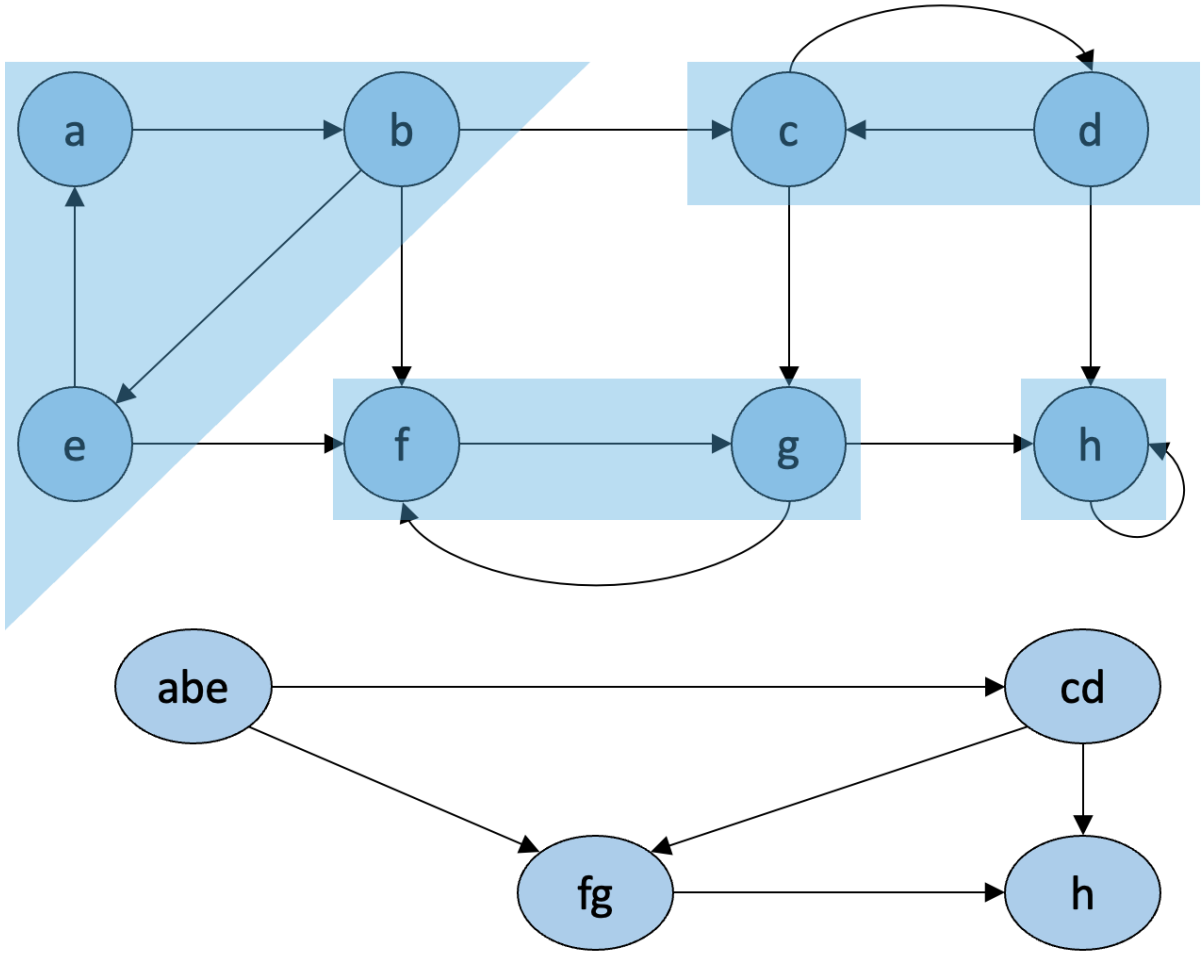
Componentes fuertemente conectadas

Existen 4 CFC's en el grafo anterior



Notemos que es necesario poder *ir y volver* dentro de una CFC

Componentes fuertemente conectadas



Cada componente tiene un **representante** que combina sus nodos

Grafo transpuesto

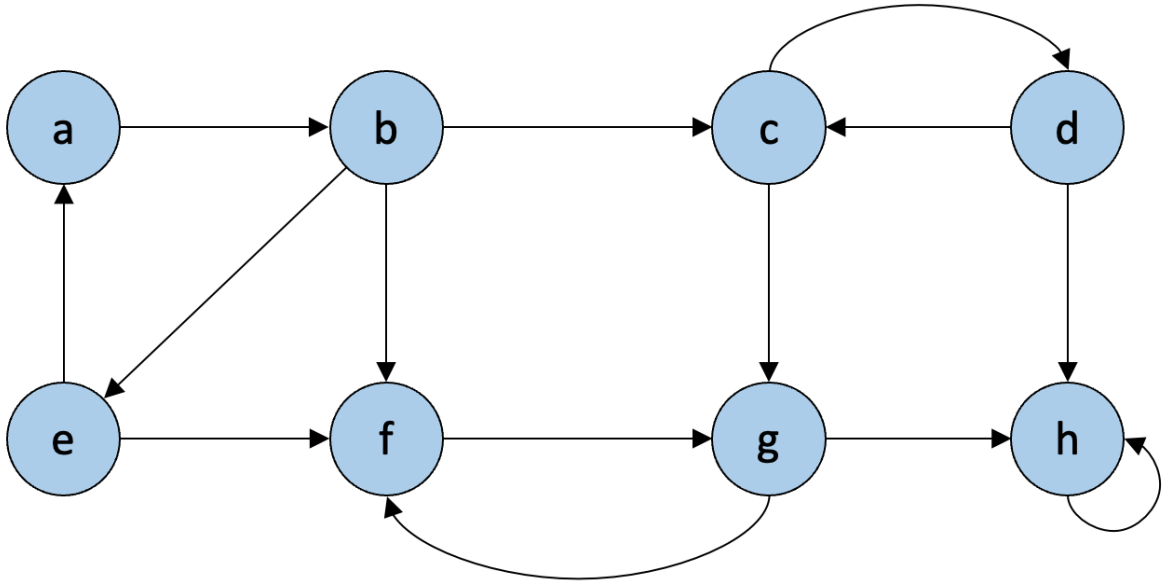
Para proponer un algoritmo, necesitamos un grafo nuevo

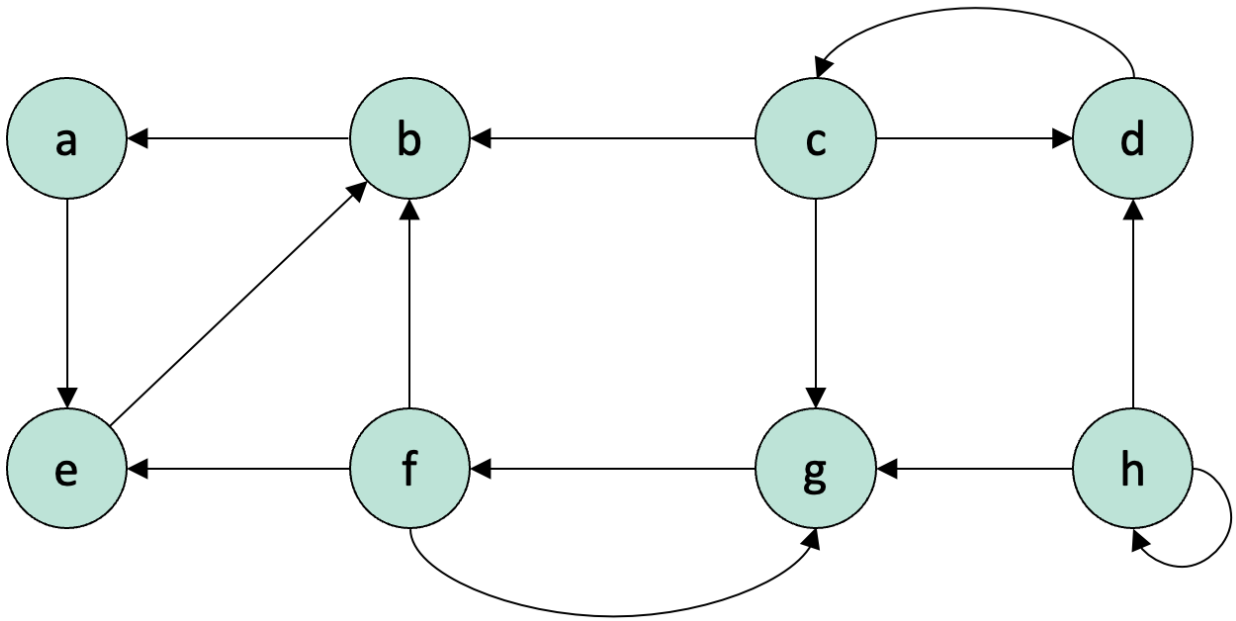
Definición

Sea G un grafo dirigido. Decimos que G^T es el grafo **transpuesto** de G si

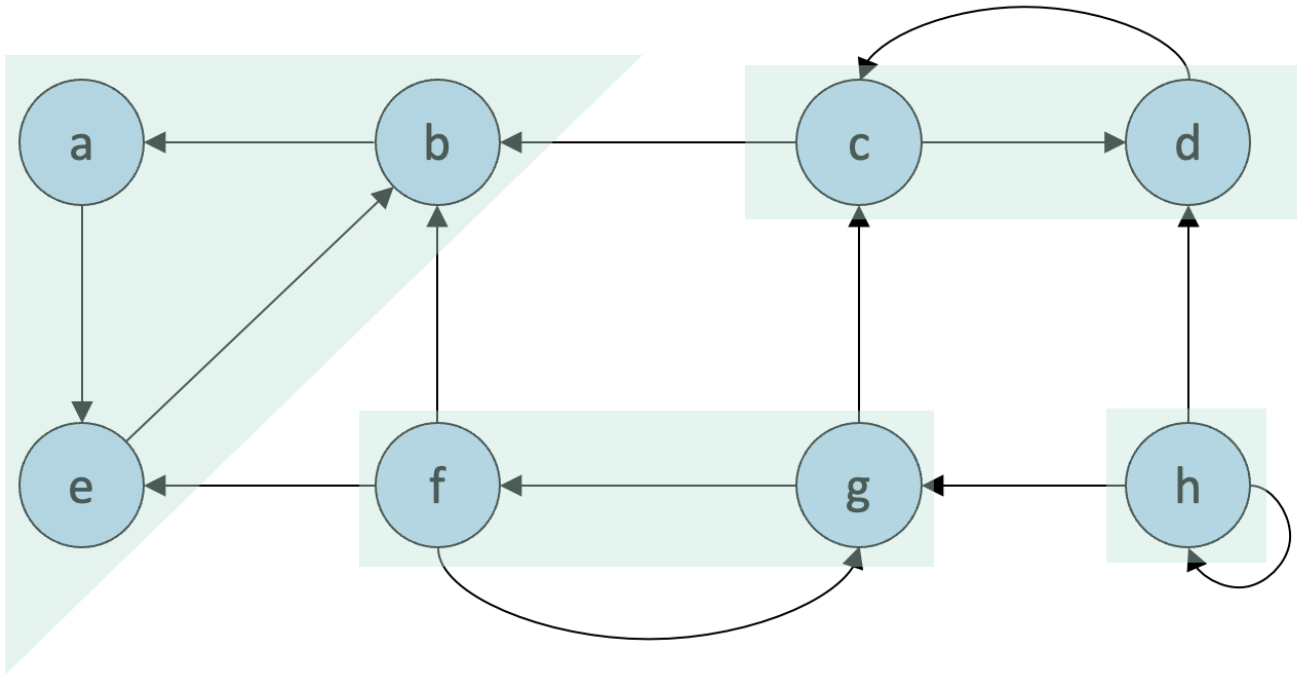
- $V(G) = V(G^T)$
- $\forall u, v \in V(G).\ (u, v) \in E(G) \rightarrow (v, u) \in E(G^T)$

El transpuesto se obtiene invirtiendo todas las aristas de G


a
b
c
d
e
f
g
h


a
b
c
d
e
f
g
h

Grafo transpuesto

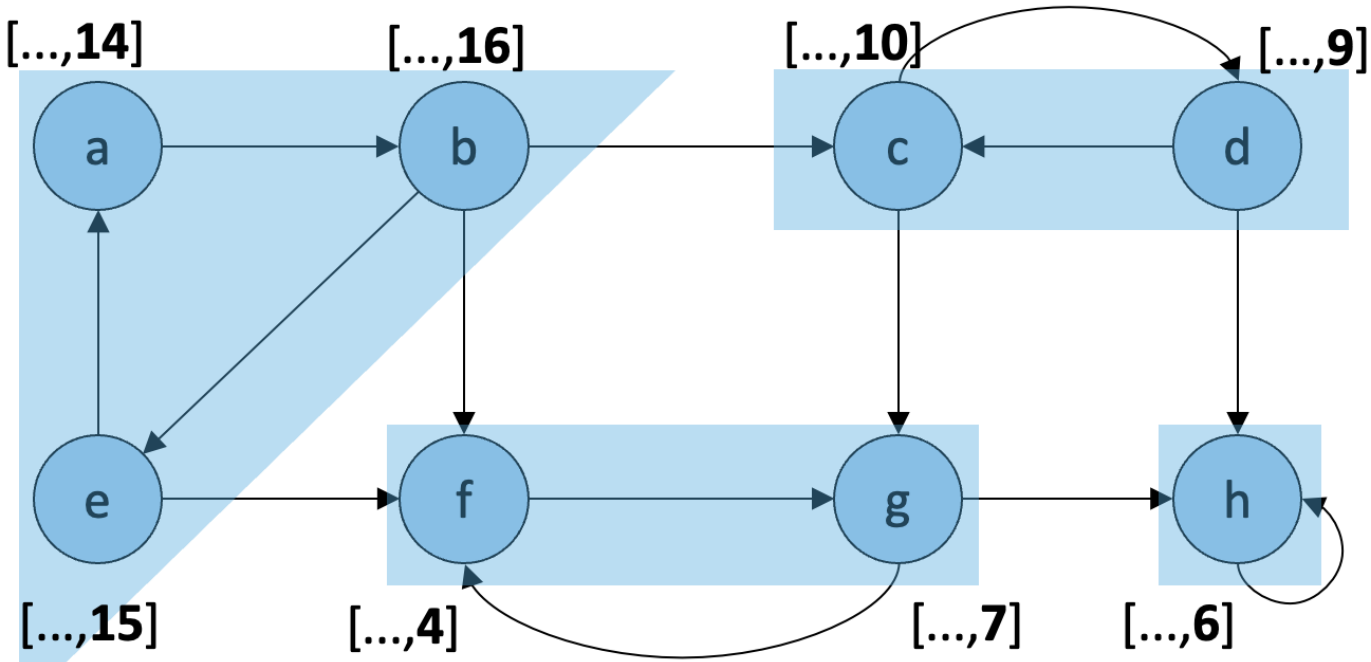


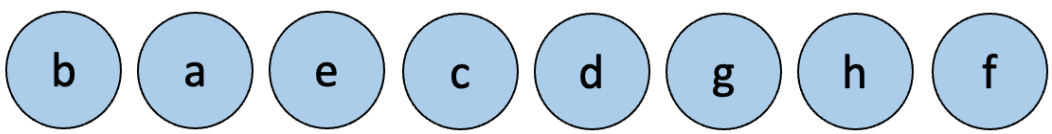
Proposición

Los grafos G y G^T tienen las mismas componentes fuertemente conectadas

Hacia un algoritmo para determinar las CFC

Construimos un orden de los nodos de G según tiempos de término

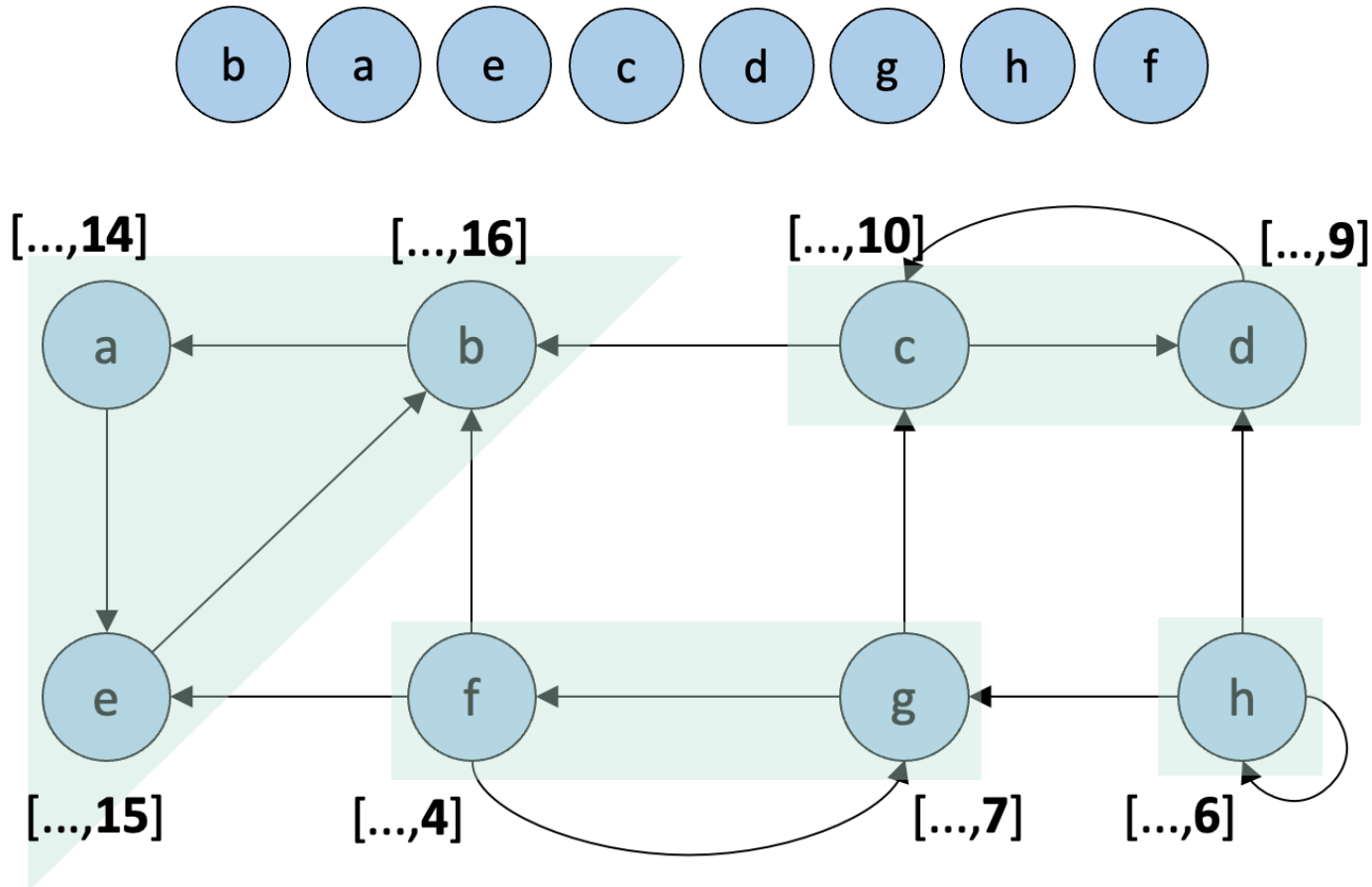




¡Ojo! Esto no es un orden topológico porque G es cíclico

Hacia un algoritmo para determinar las CFC

Recorremos el grafo transpuesto partiendo según el orden anterior



Al transponer, no es posible ir de b a c
porque están en componentes diferentes

Algoritmo de Kosaraju

input : grafo G

`Kosaraju`(G):

```
1    L ← TopSort(G)
2    for u ∈ L :
3        Assign(G, u, u)
```

input : grafo G, nodo $u \in V(G)$, nodo representante r

`Assign`(G, u, r):

```
1    if u.rep = ∅ :
2        u.rep ← r
3        for v ∈ N_{G^T}(u) :
4            Assign(G, v, r)
```

No olvidar: no podemos interpretar L como orden topológico.
Es un orden que se construye de la misma forma

Algoritmo de Kosaraju

El algoritmo de Kosaraju se basa en las propiedades del siguiente grafo

Definición

Dado un grafo G dirigido, sean $C_1, \ldots, C_k$ sus componentes fuertemente conectadas. Se define el **grafo de componentes** G^{CFC} según

- $V(G^{CFC}) = \{C_1, \ldots, C_k\}$
- Si $(u, v) \in E(G)$ y $u \in C_i, v \in C_j$, entonces $(C_i, C_j) \in E(G^{CFC})$

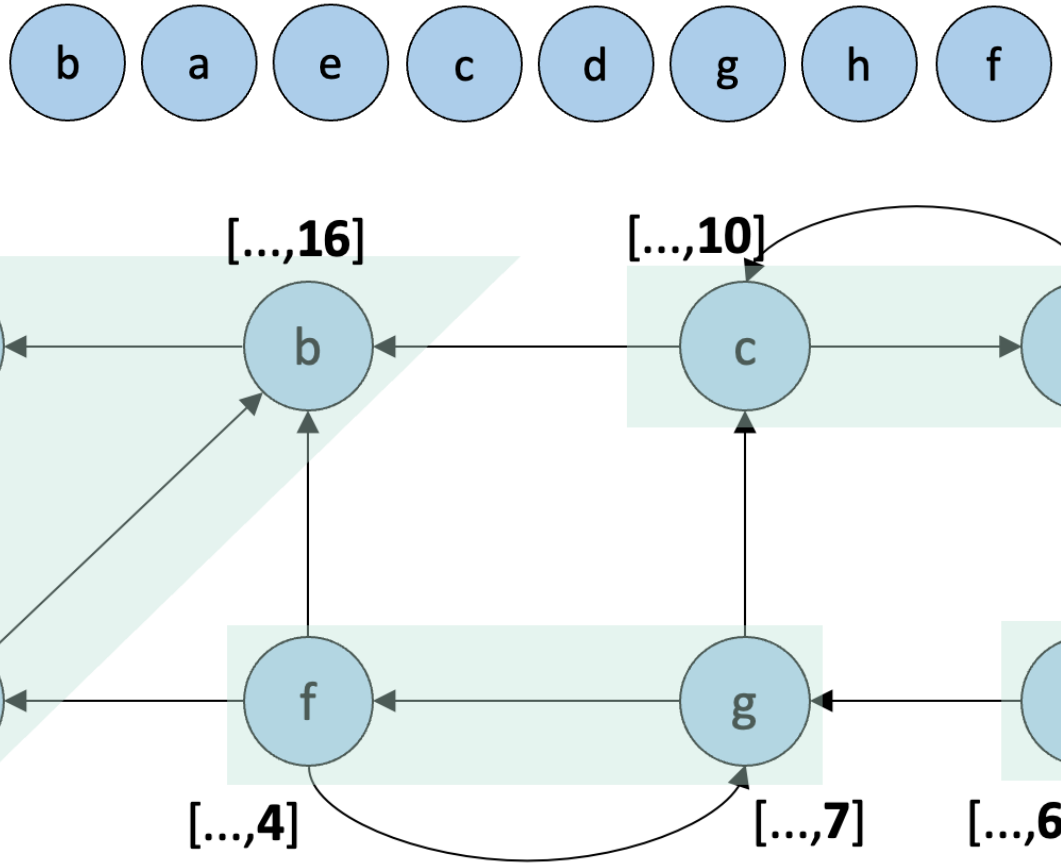
Teorema

El grafo de componentes G^{CFC} es un grafo dirigido acíclico

Corolario

El grafo de componentes G^{CFC} tiene un orden topológico

Hacia un algoritmo para determinar las CFC



La forma en que recorremos las componentes nos da su orden topológico

$(bae)(cd)(gf)(h)$

Sumario

Algoritmos codiciosos

Los **algoritmos codiciosos** plantean una estrategia algorítmica basada en el paradigma de subconjuntos

1. Tenemos conjunto $S = \{s_1, \ldots, s_n\}$ con n inputs
2. Queremos un subconjunto $S' \subseteq S$ que satisfaga restricciones
3. Queremos solución factible que maximice o minimice una **función objetivo**

Un subconjunto S' que cumple las restricciones se llama **factible**. Una solución que maximiza/minimiza se llama **óptima**

Esta es una concepción teórica
Veremos su aplicación a problemas concretos

Ejemplo

Considere el **problema de la mochila con objetos fraccionables**.

Tenemos n objetos y una mochila

- Los objetos tienen pesos $\{w_1, \ldots, w_n\}$
- Los objetos tienen ganancias por unidad de peso $\{p_1, \ldots, p_n\}$
- La mochila tiene una capacidad m, en peso
- Incluir una fracción x_k del objeto k proporciona ganancia $p_k x_k$

Ejemplo

Interesa llenar la mochila cumpliendo tres condiciones

- Queremos maximizar la ganancia total

$$\sum_{k=1}^{n} p_k x_k$$

  Esta es la **función objetivo.**

- No podemos exceder la capacidad de la mochila

$$\sum_{k=1}^{n} w_k x_k \leq m$$

- Las fracciones deben cumplir

$$0 \leq x_k \leq 1, \quad 1 \leq k \leq n$$

Ejemplo

Interesa llenar la mochila cumpliendo tres condiciones

- Maximizar función objetivo $\sum_{k=1}^{n} p_k x_k$
- No podemos exceder la capacidad $\sum_{k=1}^{n} w_k x_k \leq m$
- Las fracciones deben cumplir $0 \leq x_k \leq 1$ para $1 \leq k \leq n$

Una **solución factible** es $\{x_1, \ldots, x_n\}$ que cumple las dos últimas condiciones. Una **solución óptima** es una solución factible que maximiza la función objetivo.

¿Cómo escoger los valores x_k adecuados
para encontrar una solución óptima?

Algoritmos codiciosos

Los **algoritmos codiciosos** trabajan en etapas

- Consideran un input a la vez
- Una vez que se decide sobre un input, la decisión **es final**
- Ninguna decisión posterior cambia la actual

En el caso del problema de la mochila

- Se trata de seleccionar un subconjunto de objetos
- Y determinar la fracción x_k

Algoritmos codiciosos

Para lograr este objetivo, debemos considerar los input en cierto orden

- Se usa un **procedimiento de selección**
- Si la inclusión del próximo input en la solución óptima parcial produce solución infactible, no lo consideramos
- En otro caso, el input se agrega a la solución

¿Qué diferencia hay con backtracking?

Algoritmos codiciosos

El procedimiento de selección se basa en una **medida de optimización** o **estrategia codiciosa**

- Seleccionamos un input de forma **localmente óptima**
- Esperamos que esa selección nos lleve a una solución globalmente óptima

¿Qué estrategias codiciosas podemos usar?

Algoritmos codiciosos

Ejemplo

Para el problema de la mochila, podemos considerar las siguientes estrategias codiciosas *para la etapa actual del algoritmo*

- Incluir el objeto con mayor ganancia
- Incluir el con menor peso
- Incluir el que tenga mayor cuociente ganancia/peso

Normalmente, la mayoría de las estrategias no producen soluciones óptimas

Ejemplo

Considere la siguiente instancia del problema de la mochila con $m = 20$ y $n = 3$

- pesos $w_1 = 18,\ w_2 = 15,\ w_3 = 10$
- ganancias $p_1 = 25,\ p_2 = 24,\ p_3 = 15$

Algunas soluciones **factibles**

estrategia	x_1	x_2	x_3	$\sum_{k=1}^{n} w_k x_k$	$\sum_{k=1}^{n} p_k x_k$
mayor ganancia	1	2/15	0	20	28.2
menor peso	0	2/3	1	20	31
mayor ganancia/peso	0	1	1/2	20	31.5

En este problema, el óptimo se encuentra privilegiando ganancia/peso

Algoritmos codiciosos

Dado un problema

- Varias estrategias codiciosas pueden ser plausibles
- La mayoría produce soluciones **subóptimas**

Para garantizar que una estrategia produce soluciones óptimas es necesario **demostrarlo**

Algoritmos codiciosos

input : Arreglo de inputs $A[0 \ldots n-1]$, cantidad de inputs n

`Greedy`(A, n)**:**

```
1    S ← ∅
2    for i = 1 ... n :
3        x ← Select(A)
4        if Feasible(S, x) :
5            S ← Union(S, x)
6    return S
```

Tal como en backtracking, esta es una idea abstracta...
Su implementación dependerá de cada problema

Sumario

Selección de tareas

Consideremos el problema de escoger tareas

- La tarea i tiene un plazo d_i (día del mes)
- Además tiene una ganancia p_i que se obtiene si la tarea se hace a tiempo (antes del plazo)

Una tarea toma **un día en completarse** y solo se puede realizar **una tarea al día** y

Objetivo: maximizar la ganancia total

Selección de tareas

Una solución factible será un subconjunto T se tareas que se pueden realizar en algún orden

- El **valor** de T será

$$p(T) = \sum_{k \in T} p_k$$

Una solución factible T es óptima si su valor $p(T)$ es máximo

Selección de tareas

Ejemplo

Suponiendo que hoy es el día 0, sean las tareas $\{1, 2, 3, 4\}$ tales que

- Sus plazos son $[d_1, d_2, d_3, d_4] = [2, 1, 2, 1]$
- Sus ganancias son $[p_1, p_2, p_3, p_4] = [100, 10, 15, 27]$

Consideremos la estrategia: escoger la tarea que entrega más ganancia cada día

Es decir, estamos usando la función objetivo como estrategia codiciosa

Selección de tareas

Ejemplo

Suponiendo que hoy es el día 0, sean las tareas $\{1, 2, 3, 4\}$ tales que

- Sus plazos son $[d_1, d_2, d_3, d_4] = [2, 1, 2, 1]$
- Sus ganancias son $[p_1, p_2, p_3, p_4] = [100, 10, 15, 27]$

Tenemos entonces

Solución factible T	Orden de procesam.	Valor $p(T)$
$\{1, 2\}$	2,1	110
$\{1, 3\}$	1,3 o 3,1	115
$\{1, 4\}$	4,1	127
$\{2, 3\}$	2,3	25
$\{3, 4\}$	4,3	42
$\{1\}$	1	100
$\{2\}$	2	10
$\{3\}$	3	15
$\{4\}$	4	27

Programación de charlas

Consideremos ahora el problema de asignar charlas en una misma sala

- Tenemos n charlas por asignar
- La charla i tiene hora de inicio s_i y de término f_i
- Es decir, se define el intervalo de tiempo $[s_i, f_i)$

Solo se puede realizar **una charla a la vez**

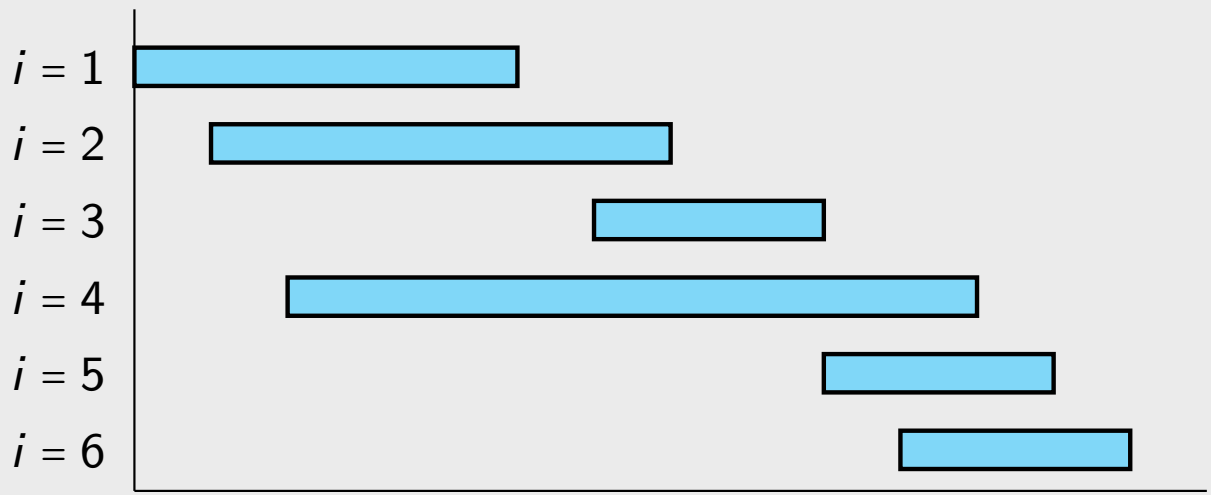
Objetivo: maximizar el número de charlas ofrecidas en la sala

Programación de charlas

Ejemplo

Sean las siguientes charlas con sus intervalos

- $i = 1$, $[0, 5)$
- $i = 2$, $[1, 7)$
- $i = 3$, $[6, 9)$
- $i = 4$, $[2, 11)$
- $i = 5$, $[9, 12)$
- $i = 6$, $[10, 13)$



Programación de charlas

Ejemplo

Posibles estrategias codiciosas: elegir primero la charla...

- que empieza más temprano
- más corta
- tiene menos incompatibilidades con otras charlas

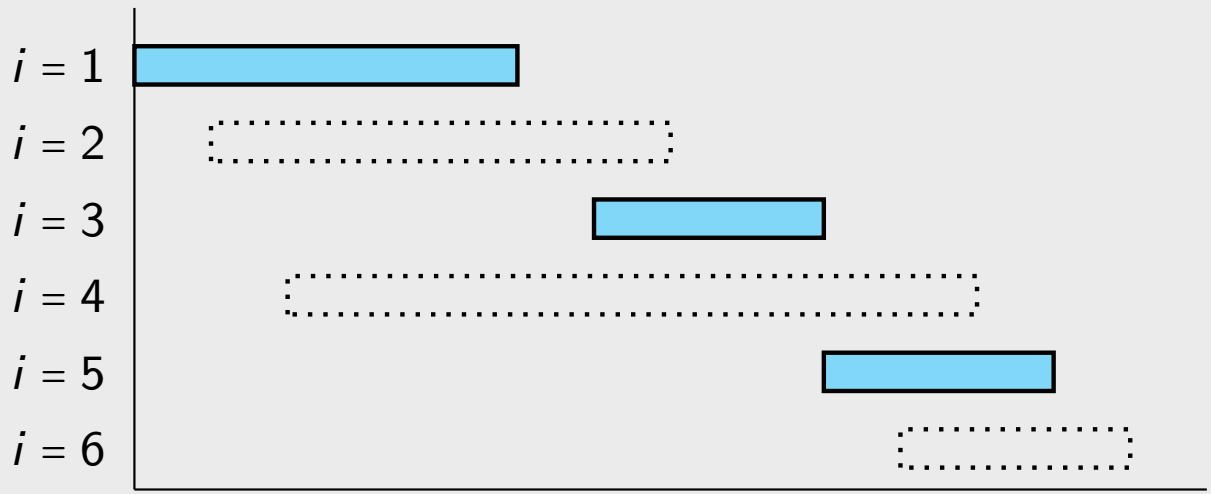
En general, ninguna de estas estrategias produce una solución óptima

Programación de charlas

Ejemplo

Escojamos según la charla **que termina más temprano**

- $i = 1,\ [0, 5)$
- $i = 2,\ [1, 7)$
- $i = 3,\ [6, 9)$
- $i = 4,\ [2, 11)$
- $i = 5,\ [9, 12)$
- $i = 6,\ [10, 13)$



¿Cómo demostramos que en el caso general, esta estrategia es óptima?

Programación de charlas

Para demostrar que la estrategia codiciosa es óptima seguiremos el siguiente procedimiento

1. Definimos $A = \{i_1, \ldots, i_k\}$ como la solución producida por nuestro algoritmo codicioso
2. Definimos $B = \{j_1, \ldots, j_m\}$ como una solución óptima en algún orden
3. Definimos una métrica $g(S)$ que entrega un valor para un subconjunto S de inputs escogidos
4. Usamos la métrica para comparar soluciones parciales y demostramos por inducción esa comparación
5. Finalmente argumentamos que la solución es óptima

Programación de charlas

Ejemplo

Para el problema de programación de charlas, sean

- $A = \{i_1, \ldots, i_k\}$ las k charlas escogidas por el codicioso, en el orden en que fueron agregadas
- $B = \{j_1, \ldots, j_m\}$ una solución óptima ordenada por tiempo de término
- $g(S)$ entrega el último tiempo de término del conjunto S de charlas, i.e.

$$g(\{i_1, \ldots, i_r\}) = f_{i_r}$$

Nuestro objetivo es probar que para todo $r \leq k$,

$$g(\{i_1, \ldots, i_r\}) \leq g(\{j_1, \ldots, j_r\})$$

Programación de charlas

Ejemplo

Probamos por inducción

- **C.B.** Para $r = 1$, dado que i_1 es la charla con el menor tiempo de término, para cualquier otra j_1 se cumple $f_{i_1} \leq f_{j_1}$. Luego

$$g(\{i_1\}) \leq g(\{j_1\})$$

- **H.I.** Suponemos que se cumple para $t \geq 1$

$$g(\{i_1, \ldots, i_t\}) \leq g(\{j_1, \ldots, j_t\})$$

Ejemplo

- Consideramos el conjunto de charlas óptimas $\{j_1, \ldots, j_t\}$. Cualquier charla que sea agregada como j_{t+1} a dicha solución óptima cumple por factibilidad que

$$f_{j_t} \leq s_{j_{t+1}} \leq t_{j_{t+1}}$$

Ahora consideramos la siguiente charla codiciosa i_{t+1}. Esta cumple con ser la primera que termina y comienza después de i_t. Por **H.I.** sabemos que i_t termina antes que j_t y esto abre dos opciones para i_{t+1} gracias al criterio de selección por tiempo de término

  - i_{t+1} es la misma charla j_{t+1}
  - i_{t+1} termina antes que j_{t+1}

En ambos casos, se obtiene que $f_{i_{t+1}} \leq f_{j_{t+1}}$ y con ello

$$g(\{i_1, \ldots, i_{t+1}\}) \leq g(\{j_1, \ldots, j_{t+1}\})$$

Ejemplo

Una consecuencia de este resultado demostrado es que las últimas charlas de las soluciones cumplen

$$f_{i_k} \leq f_{j_k}$$

Probaremos que A es óptima por contradicción.

Supongamos que A no es óptima. En tal caso, $m > k$ pues se podrían haber programado más charlas, i.e. hay una charla j_{k+1} en B que no está en A. Dado que

$$f_{i_k} \leq f_{j_k} \leq s_{j_{k+1}} \leq f_{j_{k+1}}$$

esta charla es compatible con A y el algoritmo debió haberla escogido. Esto incumple el criterio de selección y se concluye que hay una contradicción. □