Representaciones y algoritmos de grafos

Algoritmos y Estructuras de Datos III (AED3)

Santiago Cifuentes

El problema

$\{P_4, C_4\}$-*free*

Dado un grafo G, decidir si contiene un P_4 o un C_4 inducido.

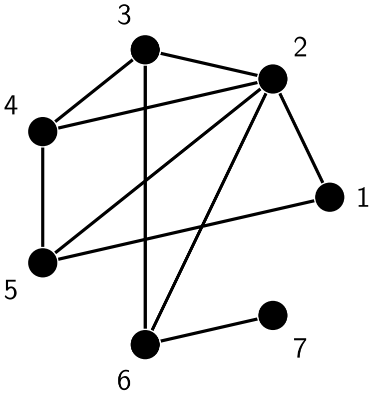
El problema

$\{P_4, C_4\}$-*free*

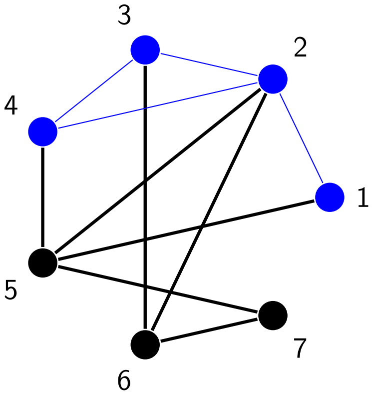
Dado un grafo G, decidir si contiene un P_4 o un C_4 inducido.

Un grafo $G = (V, E)$ contiene a otro grafo H como subgrafo inducido cuando existe un subconjunto de nodos $V' \subseteq V$ tal que el subgrafo de G obtenido al solo considerar los nodos V' y los ejes entre nodos de V' es isomorfo a H.

Ejemplos

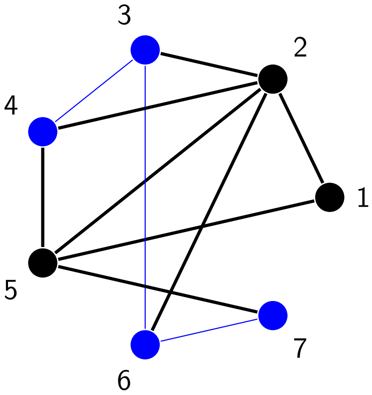


Ejemplos



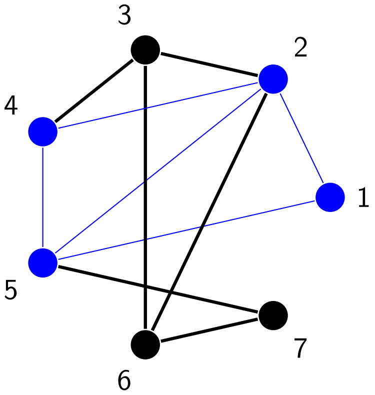
Subrafo inducido por $V' = \{1, 2, 3, 4\}$

Ejemplos



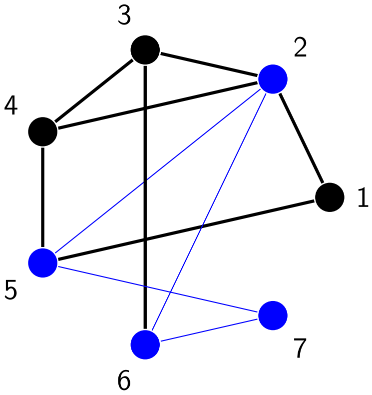
Subgrafo inducido por $V' = \{3, 4, 6, 7\}$.

Ejemplos



Subgrafo inducido por $V' = \{1, 2, 4, 5\}$

Ejemplos



Subgrafo inducido por $V' = \{2, 5, 6, 7\}$

Subgrafo prohibido

Idea más simple:

Subgrafo prohibido

Idea más simple: Alcanza con ver si existe alguna 4-upla w, x, y, z de nodos tal que:

Subgrafo prohibido

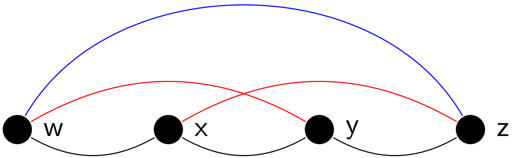
Idea más simple: Alcanza con ver si existe alguna 4-upla w, x, y, z de nodos tal que:

- $wxyz$ es un camino.
- Los ejes wy y xz no pertenecen al grafo.

Subgrafo prohibido

Idea más simple: Alcanza con ver si existe alguna 4-upla w, x, y, z de nodos tal que:

- $wxyz$ es un camino.
- Los ejes wy y xz no pertenecen al grafo.



Primer algoritmo

Iteramos sobre todas las $4-$ *uplas*, y para cada una checkeamos los ejes que queremos.

Algorithm 1 Primer algoritmo

function $\{P_4, C_4\}$-*free*(G)
    Armo una estructura para G
    for $w \in V$ **do**
        for $x \in V$ **do**
            for $y \in V$ **do**
                for $z \in V$ **do**
                    Verificar $wx, xy, yz \in E$, $wy, xz \notin E$.
                end for
            end for
        end for
    end for
end function

Primer algoritmo – Complejidad

- ¿Cuántas operaciones requiere el cuerpo del loop si usamos una matriz de adyacencias?.

Primer algoritmo – Complejidad

- ¿Cuántas operaciones requiere el cuerpo del loop si usamos una matriz de adyacencias?.
- La complejidad usando matriz de adyacencias queda $O(n^2 + n^4) = O(n^4)$.

Primer algoritmo – Complejidad

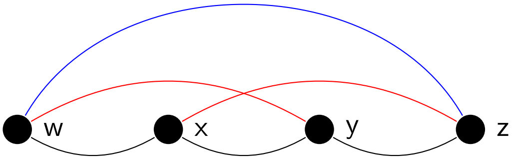
- ¿Cuántas operaciones requiere el cuerpo del loop si usamos una matriz de adyacencias?.
- La complejidad usando matriz de adyacencias queda $O(n^2 + n^4) = O(n^4)$.
- ¿Y si usamos la lista de adyacencias?

Primer algoritmo – Complejidad

- ¿Cuántas operaciones requiere el cuerpo del loop si usamos una matriz de adyacencias?.
- La complejidad usando matriz de adyacencias queda $O(n^2 + n^4) = O(n^4)$.
- ¿Y si usamos la lista de adyacencias?
- La complejidad usando lista de adyacencias queda $O((n + m) + n^5) = O(n^5)$.

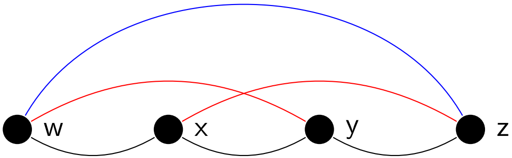
Segundo algoritmo – Idea

- En vez de iterar sobre todas las $4 - uplas$, iteremos sobre los ejes xy



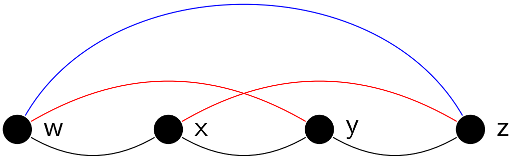
Segundo algoritmo – Idea

- En vez de iterar sobre todas las $4 - uplas$, iteremos sobre los ejes xy
- Queremos ver, para cada xy, si existe un nodo w vecino de x que no sea vecino de y, y otro z que sea vecino de y pero no de x.



Segundo algoritmo – Idea

- En vez de iterar sobre todas las $4 - uplas$, iteremos sobre los ejes xy
- Queremos ver, para cada xy, si existe un nodo w vecino de x que no sea vecino de y, y otro z que sea vecino de y pero no de x.
- Es decir, que ningún vecindario contenga al otro (i.e. que ningún nodo domine al otro).



Segundo algoritmo

- Traducimos el problema a **Decidir si existe un eje** xy **tal que** x **no domine a** y **y** y **no domine a** x

Segundo algoritmo

- Traducimos el problema a **Decidir si existe un eje xy tal que x no domine a y y y no domine a x**
- Es decir, un eje xy tal que $N(y) \not\subseteq N(x)$ y $N(x) \not\subseteq N(y)$.

Segundo algoritmo

- Traducimos el problema a **Decidir si existe un eje** xy **tal que** x **no domine a** y **y** y **no domine a** x
- Es decir, un eje xy tal que $N(y) \not\subseteq N(x)$ y $N(x) \not\subseteq N(y)$.
- A estos ejes los vamos a llamar **prohibidos**. A los que no son prohibidos los llamamos **válidos**.

Segundo algoritmo

Algorithm 2 Segundo algoritmo

```
function tieneEjeProhibido(G)
    Armo alguna estructura para G
    for xy ∈ E do
        Si N(x) ⊈ N(y) y N(y) ⊈ N(x) devolver true
    end for
end function
```

Segundo algoritmo – Complejidad

Podemos iterar sobre E en $O(m)$ porque nos dan la lista de aristas como input ¿Cuánto cuesta el algoritmo en cada estructura?

Segundo algoritmo – Complejidad

Podemos iterar sobre E en $O(m)$ porque nos dan la lista de aristas como input ¿Cuánto cuesta el algoritmo en cada estructura?

- **Para la matriz**:

Segundo algoritmo – Complejidad

Podemos iterar sobre E en $O(m)$ porque nos dan la lista de aristas como input ¿Cuánto cuesta el algoritmo en cada estructura?

- **Para la matriz**: tenemos que construir la matriz en $O(n^2)$. Luego, el cuerpo del for toma $O(n)$. En total, $O(n^2 + nm)$.

Segundo algoritmo – Complejidad

Podemos iterar sobre E en $O(m)$ porque nos dan la lista de aristas como input ¿Cuánto cuesta el algoritmo en cada estructura?

- **Para la matriz**: tenemos que construir la matriz en $O(n^2)$. Luego, el cuerpo del for toma $O(n)$. En total, $O(n^2 + nm)$.
- **Para la lista de adyacencias**:

Segundo algoritmo – Complejidad

Podemos iterar sobre E en $O(m)$ porque nos dan la lista de aristas como input ¿Cuánto cuesta el algoritmo en cada estructura?

- **Para la matriz**: tenemos que construir la matriz en $O(n^2)$. Luego, el cuerpo del for toma $O(n)$. En total, $O(n^2 + nm)$.
- **Para la lista de adyacencias**: construir la representación toma $O(n + m)$. Luego podemos revisar cada intersección en $O(d(x) + d(y))$. El orden de la cantidad de operaciones queda como:

$$\sum_{xy \in E} d(x) + d(y)$$

Segundo algoritmo – Complejidad

Acotemos el caso de la lista de adyacencias:

Segundo algoritmo – Complejidad

Acotemos el caso de la lista de adyacencias:

$$\sum_{xy \in E} d(x) + d(y) \leq \sum_{xy \in E} n + n = 2n \times \sum_{xy \in E} 1 = 2nm = O(nm)$$

Segundo algoritmo – Complejidad

Acotemos el caso de la lista de adyacencias:

$$\sum_{xy \in E} d(x) + d(y) \leq \sum_{xy \in E} n + n = 2n \times \sum_{xy \in E} 1 = 2nm = O(nm)$$

Esta cota es fina. Lo pueden verificar considerando grafos estrella grafos completos.

Segundo algoritmo – Complejidad

Acotemos el caso de la lista de adyacencias:

$$\sum_{xy \in E} d(x) + d(y) \leq \sum_{xy \in E} n + n = 2n \times \sum_{xy \in E} 1 = 2nm = O(nm)$$

Esta cota es fina. Lo pueden verificar considerando grafos estrella grafos completos.
¿Cuál algoritmo es mejor?

- **Con matriz**:

Segundo algoritmo – Complejidad

Acotemos el caso de la lista de adyacencias:

$$\sum_{xy \in E} d(x) + d(y) \leq \sum_{xy \in E} n + n = 2n \times \sum_{xy \in E} 1 = 2nm = O(nm)$$

Esta cota es fina. Lo pueden verificar considerando grafos estrella grafos completos.
¿Cuál algoritmo es mejor?

- **Con matriz**: $O(n^2 + nm)$
- **Con lista de adyacencias**: pause $O(n + m + nm) = O(nm)$

Algoritmo final – Idea

Algunas observaciones:

Algoritmo final – Idea

Algunas observaciones:

- Si $d(x) > d(y)$, ¿Hace falta checkear si y domina a x? ¿Y si $d(x) = d(y)$?

Algoritmo final – Idea

Algunas observaciones:

- Si $d(x) > d(y)$, ¿Hace falta checkear si y domina a x? ¿Y si $d(x) = d(y)$?
- Entonces podemos pensar que, si tenemos los nodos ordenados por grado de menor a mayor como $v_1, \ldots, v_n$, queremos ver para cada nodo v_i si este domina a todos los nodos adyacentes **a su izquierda** en el orden $v_1, \ldots, v_n$.

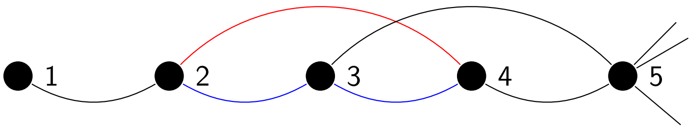
Algoritmo final – Idea

Algunas observaciones:

- Si $d(x) > d(y)$, ¿Hace falta checkear si y domina a x? ¿Y si $d(x) = d(y)$?
- Entonces podemos pensar que, si tenemos los nodos ordenados por grado de menor a mayor como $v_1, \ldots, v_n$, queremos ver para cada nodo v_i si este domina a todos los nodos adyacentes **a su izquierda** en el orden $v_1, \ldots, v_n$.
- Por otro lado, si x domina a y, y y domina a z, entonces x domina a z ¿Por qué?

Algoritmo final

Podemos usar la transitividad de la dominación para "verificar" menos ejes.



$$d(1) \leq d(2) \leq d(3) \leq \dots$$

Algoritmo final - Recapitulando

- Empezamos con el problema de decidir si un grafo contiene a P_4 o a C_4 como subgrafo inducido.
- Vimos que ese problema era equivalente al de decidir si el grafo contiene un eje **prohibido**: un eje xy es prohibido si x no domina a y y y no domina a x.
- Si tengo un eje xy con $d(x) > d(y)$, entonces y nunca va a dominar a x. Y si $d(x) = d(y)$ entonces alcanza con revisar una sola de las dominaciones. Por lo tanto, si ordeno los nodos por grado como $v_1, \ldots, v_n$ entonces para cada nodo v_i solo tengo que revisar si domina a sus vecinos a su izquierda en el orden.
- Usando la transitividad de la relación de dominación puedo ahorrarme revisar algunos ejes.
- Ahora vamos a definir un conjunto de ejes que me permite ahorrar una parte importante de los ejes del grafo.

Algoritmo final - Recapitulando

- Empezamos con el problema de decidir si un grafo contiene a P_4 o a C_4 como subgrafo inducido.
- Vimos que ese problema era equivalente al de decidir si el grafo contiene un eje **prohibido**: un eje xy es prohibido si x no domina a y y y no domina a x.
- Si tengo un eje xy con $d(x) > d(y)$, entonces y nunca va a dominar a x. Y si $d(x) = d(y)$ entonces alcanza con revisar una sola de las dominaciones. Por lo tanto, si ordeno los nodos por grado como $v_1, \ldots, v_n$ entonces para cada nodo v_i solo tengo que revisar si domina a sus vecinos a su izquierda en el orden.
- Usando la transitividad de la relación de dominación puedo ahorrarme revisar algunos ejes.
- Ahora vamos a definir un conjunto de ejes que me permite ahorrar una parte importante de los ejes del grafo.

Algoritmo final - Recapitulando

- Empezamos con el problema de decidir si un grafo contiene a P_4 o a C_4 como subgrafo inducido.
- Vimos que ese problema era equivalente al de decidir si el grafo contiene un eje **prohibido**: un eje xy es prohibido si x no domina a y y y no domina a x.
- Si tengo un eje xy con $d(x) > d(y)$, entonces y nunca va a dominar a x. Y si $d(x) = d(y)$ entonces alcanza con revisar una sola de las dominaciones. Por lo tanto, si ordeno los nodos por grado como $v_1, \ldots, v_n$ entonces para cada nodo v_i solo tengo que revisar si domina a sus vecinos a su izquierda en el orden.
- Usando la transitividad de la relación de dominación puedo ahorrarme revisar algunos ejes.
- Ahora vamos a definir un conjunto de ejes que me permite ahorrar una parte importante de los ejes del grafo.

Algoritmo final - Recapitulando

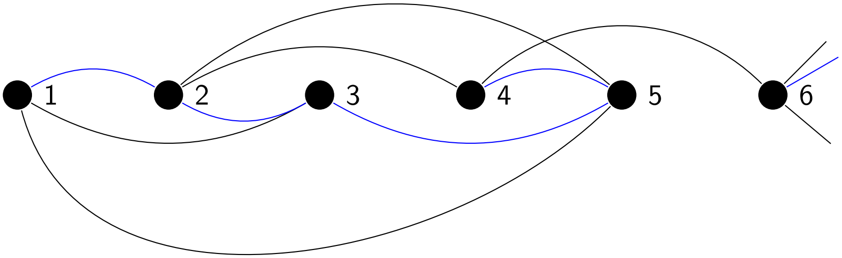
- Empezamos con el problema de decidir si un grafo contiene a P_4 o a C_4 como subgrafo inducido.
- Vimos que ese problema era equivalente al de decidir si el grafo contiene un eje **prohibido**: un eje xy es prohibido si x no domina a y y y no domina a x.
- Si tengo un eje xy con $d(x) > d(y)$, entonces y nunca va a dominar a x. Y si $d(x) = d(y)$ entonces alcanza con revisar una sola de las dominaciones. Por lo tanto, si ordeno los nodos por grado como $v_1, \ldots, v_n$ entonces para cada nodo v_i solo tengo que revisar si domina a sus vecinos a su izquierda en el orden.
- Usando la transitividad de la relación de dominación puedo ahorrarme revisar algunos ejes.
- Ahora vamos a definir un conjunto de ejes que me permite ahorrar una parte importante de los ejes del grafo.

Algoritmo final - Recapitulando

- Empezamos con el problema de decidir si un grafo contiene a P_4 o a C_4 como subgrafo inducido.
- Vimos que ese problema era equivalente al de decidir si el grafo contiene un eje **prohibido**: un eje xy es prohibido si x no domina a y y y no domina a x.
- Si tengo un eje xy con $d(x) > d(y)$, entonces y nunca va a dominar a x. Y si $d(x) = d(y)$ entonces alcanza con revisar una sola de las dominaciones. Por lo tanto, si ordeno los nodos por grado como $v_1, \ldots, v_n$ entonces para cada nodo v_i solo tengo que revisar si domina a sus vecinos a su izquierda en el orden.
- Usando la transitividad de la relación de dominación puedo ahorrarme revisar algunos ejes.
- Ahora vamos a definir un conjunto de ejes que me permite ahorrar una parte importante de los ejes del grafo.

Algoritmo final - Idea

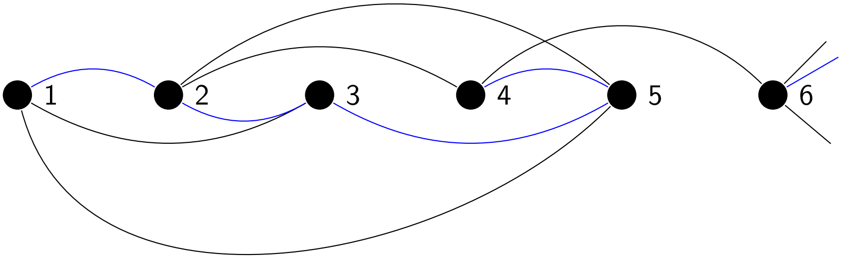
- Para cada nodo, llamamos *siguiente* de v al nodo adyacente a v que está más cerca por la derecha. Lo denotamos como $siguiente(v)$ ¿Todos los nodos tienen un siguiente?



$$siguiente(1) = 2, siguiente(2) = 3, siguiente(3) = 5, \ldots$$

Algoritmo final - Idea

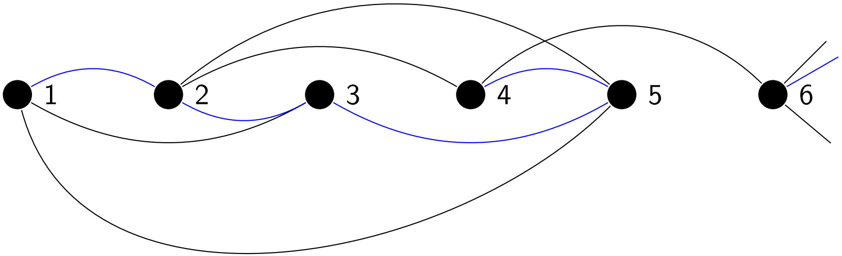
- Para cada nodo, llamamos *siguiente* de v al nodo adyacente a v que está más cerca por la derecha. Lo denotamos como $siguiente(v)$ ¿Todos los nodos tienen un siguiente?
- A los ejes que unen a un nodo con su siguiente los llamamos *importantes*. ¿Cuál es la máxima cantidad posible de ejes *importantes*?



$$siguiente(1) = 2, siguiente(2) = 3, siguiente(3) = 5, \ldots$$

Algoritmo final - Idea

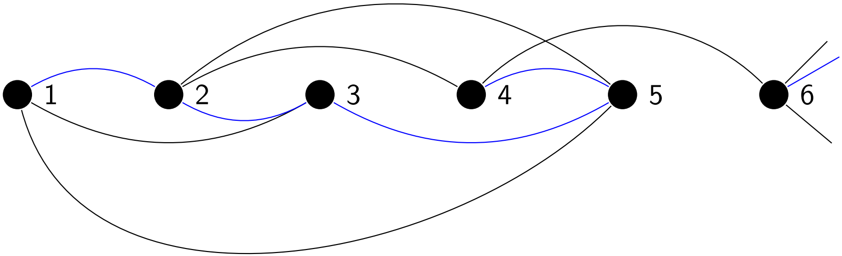
- Para cada nodo, llamamos *siguiente* de v al nodo adyacente a v que está más cerca por la derecha. Lo denotamos como $siguiente(v)$ ¿Todos los nodos tienen un siguiente?
- A los ejes que unen a un nodo con su siguiente los llamamos *importantes*. ¿Cuál es la máxima cantidad posible de ejes *importantes*?



$$siguiente(1) = 2, siguiente(2) = 3, siguiente(3) = 5, \ldots$$

Algoritmo final - Idea

- Para cada nodo, llamamos *siguiente* de v al nodo adyacente a v que está más cerca por la derecha. Lo denotamos como $siguiente(v)$ ¿Todos los nodos tienen un siguiente?
- A los ejes que unen a un nodo con su siguiente los llamamos *importantes*. ¿Cuál es la máxima cantidad posible de ejes *importantes*?



$$siguiente(1) = 2, siguiente(2) = 3, siguiente(3) = 5, \ldots$$

Algoritmo final - Idea

- Para cada nodo, llamamos *siguiente* de v al nodo adyacente a v que está más cerca por la derecha. Lo denotamos como $siguiente(v)$ ¿Todos los nodos tienen un siguiente?
- A los ejes que unen a un nodo con su siguiente los llamamos *importantes*. ¿Cuál es la máxima cantidad posible de ejes *importantes*?

Lema

Un grafo tiene algún eje *prohibido* si y solamente si alguno de los ejes *importantes* es *prohibido*.

Contrarecíproco de la ida

Si los ejes importantes no son prohibidos, entonces el grafo no tiene ejes prohibidos.

Algoritmo final - Idea

- Para cada nodo, llamamos *siguiente* de v al nodo adyacente a v que está más cerca por la derecha. Lo denotamos como $siguiente(v)$ ¿Todos los nodos tienen un siguiente?
- A los ejes que unen a un nodo con su siguiente los llamamos *importantes*. ¿Cuál es la máxima cantidad posible de ejes *importantes*?

Lema

Un grafo tiene algún eje *prohibido* si y solamente si alguno de los ejes *importantes* es *prohibido*.

Contrarecíproco de la ida

Si los ejes importantes no son prohibidos, entonces el grafo no tiene ejes prohibidos.

Algoritmo final

- Con todo esto, sabemos que alcanza con revisar los ejes importantes, que son menos que n.

Algoritmo final

- Con todo esto, sabemos que alcanza con revisar los ejes importantes, que son menos que n.
- Tenemos que ver que **cada nodo domine a aquellos de los que es siguiente**.

Algoritmo final

- Con todo esto, sabemos que alcanza con revisar los ejes importantes, que son menos que n.
- Tenemos que ver que **cada nodo domine a aquellos de los que es siguiente**.
- Si tenemos en un vector los *siguientes* de cada nodo, entonces podemos computar un vector *debeDominar* que dice, para cada v, los nodos que tiene que dominar v.

Algoritmo final

Algorithm 3 Algoritmo final

function *tieneEjesProhibidos*(G)
    Armo la lista de adyacencias de G.
    nodosOrdenados $\leftarrow$ ordeno a los nodos en función de su grado.
    siguientes $\leftarrow$ calculo los *siguientes* de cada nodo
    debeDominar $\leftarrow$ calculo, para cada v, los nodos que debe dominar
    for $v \in V$ **do**
        Marco los vecinos de v.
        Para cada $w \in \mathit{debeDominar}[v]$ reviso que su vecindario esté marcado.
        Desmarco los vecinos de v.
    end for
end function

Algoritmo final

Algorithm 4 Algoritmo final

function $\{P_4, C_4\}$-*free*(G)
  Armo la lista de adyacencias de G. $\triangleright$ $O(n+m)$
  nodosOrdenados $\leftarrow$ ordeno a los nodos en función de su grado. $\triangleright$ $O(n)$
  siguientes $\leftarrow$ calculo los *siguientes* de cada nodo $\triangleright$ $O(n+m)$
  debeDominar $\leftarrow$ calculo, para cada v, los nodos que debe dominar $\triangleright$ $O(n)$
  for $v \in V$ **do** $\triangleright$ $O(?)$
    Marco los vecinos de v.
    Para cada $w \in$ *debeDominar*$[v]$ reviso que su vecindario esté marcado.
    Desmarco los vecinos de v.
  end for
end function

Algoritmo final - Complejidad

Si solo consideramos el interior **for**, ¿Cuántas veces voy a recorrer un vecindario cualquiera $N(v)$?

Algoritmo final - Complejidad

Si solo consideramos el interior **for**, ¿Cuántas veces voy a recorrer un vecindario cualquiera $N(v)$?

- Dos veces cuando procese a v.

Algoritmo final - Complejidad

Si solo consideramos el interior **for**, ¿Cuántas veces voy a recorrer un vecindario cualquiera $N(v)$?

- Dos veces cuando procese a v.
- Una vez cuando procese a $siguiente(v)$.

Algoritmo final - Complejidad

Si solo consideramos el interior **for**, ¿Cuántas veces voy a recorrer un vecindario cualquiera $N(v)$?

- Dos veces cuando procese a v.
- Una vez cuando procese a $siguiente(v)$.

Algoritmo final - Complejidad

Si solo consideramos el interior **for**, ¿Cuántas veces voy a recorrer un vecindario cualquiera $N(v)$?

- Dos veces cuando procese a v.
- Una vez cuando procese a $\mathit{siguiente}(v)$.

Es decir, a lo sumo tres veces. Por lo que el orden de las operaciones es $\sum_{v \in V} 3d(v) = O(m)$

Algoritmo final - Complejidad

Si solo consideramos el interior **for**, ¿Cuántas veces voy a recorrer un vecindario cualquiera $N(v)$?

- Dos veces cuando procese a v.
- Una vez cuando procese a $siguiente(v)$.

Es decir, a lo sumo tres veces. Por lo que el orden de las operaciones es $\sum_{v \in V} 3d(v) = O(m)$

Por lo tanto, el algoritmo es $O(n + m)$.

Algoritmo final - Complejidad

Si solo consideramos el interior **for**, ¿Cuántas veces voy a recorrer un vecindario cualquiera $N(v)$?

- Dos veces cuando procese a v.
- Una vez cuando procese a $siguiente(v)$.

Es decir, a lo sumo tres veces. Por lo que el orden de las operaciones es $\sum_{v \in V} 3d(v) = O(m)$

Por lo tanto, el algoritmo es $O(n + m)$.

Luego, se puede decidir si un grafo contiene a P_4 o a C_4 como grafo inducido en tiempo lineal.