

Cazador de Ciclos

14. ★En este ejercicio diseñamos un algoritmo para encontrar ciclos en un digrafo. Decimos que un digrafo es *acíclico* cuando no tiene ciclos dirigidos. Recordar que un (di)grafo es *trivial* cuando tiene un sólo vértice.

- a) Demostrar con un argumento constructivo que si todos los vértices de un digrafo D tienen grado de salida mayor a 0, entonces D tiene un ciclo.

Sea D un digrafo, $\forall v \in V(D)$ $d_{\text{out}}(v) > 0$

QED D tiene un ciclo

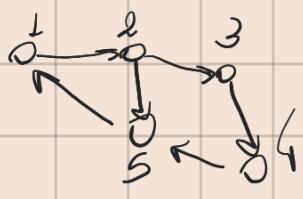
Sea $v \in V(G)$, como $d^{\text{out}}(v) > 0$ puedo moverme a otro vértice. Ahora como $d^{\text{out}}(w) > 0$ vale $\forall w \in V(G)$ puedo moverme infinitamente por D .

Si embargo, $\# V(G)$ es finita $\rightarrow D$ tiene que tener un ciclo

- b) Diseñar un algoritmo que permita encontrar un ciclo en un digrafo D cuyos vértices tengan todos grado de salida mayor a 0.



1. Dado un digrafo D sin vértice marcado y un vértice v , marco v .
2. Despues de marcar v me fijo anal es el $d^{\text{out}}(v)$ y lo recorro marcando el vértice.
3. Para el vértice del recorrido recorro también su vecindad de salida.
4. Cuando llegué a un nodo ya marcado significa que encontré un ciclo.
5. Termino el algoritmo y me fijo donde está el ciclo entre lo que marque.



Voy de 1 a 2. De 2 puedo ir a 3 y a 5. El algoritmo me dirá irte en 3 y seguir por 3, una vez hayas recorrido exhaustivamente el 3, pasa al 5.

Se podría hacer más simple? Si, yo ya sé que tiene un ciclo o bien. Como todo vértice tiene grado de salida mayor a 0 podía implementar simplemente considerando mis solo de los vértices de salida del $N^{\text{out}}(v)$. Total yo ya sé que ese vértice me va a llamar a otro y ese otro a otro y así. Como D va a ser finito la única forma que pase esto es que haya un ciclo.

1. Dado un digrafo D un vértice marcado y un vértice inicial v , marco v
2. Para ese v tomo un solo vértice de su N^{out} , lo marco y hago lo mismo.
3. Cuando ya haya llegado a un vértice marcado, paro y me fijo entre los marcados donde está el ciclo

Obs Si no le pongo un vértice inicial funciona para digrafos no conexos también

- c) Explicar detalladamente (sin usar código) cómo se implementa el algoritmo del inciso anterior. El algoritmo resultante tiene que tener complejidad temporal $O(n + m)$.

1. Pasar de lista de adyacencias a lista de aristas. En particular, queremos el vecindario de salida, con eso basta. $O(n+m)$
2. Almacenar mi vértice inicial en una variable v . $O(1)$ porque mi vértice es un int
3. Crear una pila vacía S . $O(1)$

En S vamos a guardar los nodos que fui viendo. Una vez termine la búsqueda de mi ciclo vamos a valernos de S para devolverlo.

4. Crear un vector M de n posiciones. $O(n)$

Para las marcas de los vértices, vamos a tener 2 colores: verde y rojo. Verde significa que estoy en un vértice que hasta ahora no había sido ni descubierto ni recorrido. Rojo significa que estoy en un vértice que ya descubrí.

Los voy a representar con los números 1 y 2.

5. Pongo todas las posiciones de mi vector en verde. $O(n)$
6. Armo una lista C .
7. Mientras mi vértice v no esté marcado
8. Lo marco con rojo en M $O(1)$
9. Lo agrego a mi pila S
10. Obtengo el vecindario de v $O(1)$
11. Ahora mi nuevo v pasa a ser un vértice cualquiera del vecindario $O(1)$

Como tengo un ciclo asegurado en mi digrafo, el while va a terminar siempre porque vamos a dar con un ciclo, es decir, con un vértice ya marcado.

El peor caso de este ciclo es el caso donde mi digrafo tenga un grafo subyacente C_n . En este caso voy a recorrer todos los nodos y todas las aristas. Entonces su complejidad termina siendo $O(n+m)$

12. Como el vértice v estaba marcado entonces lo agrego a C y desmarco el vértice v de M .

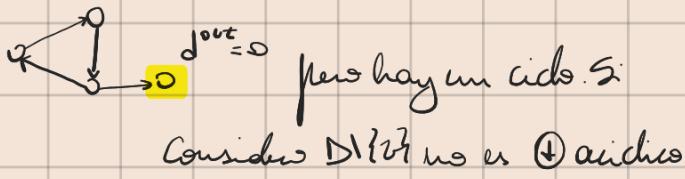
13. Mientras el tope de S esté marcado $O(n)$

14. Agrego el tope a C y lo quito de S $O(1)$

Complejidad total $O(n+m)$

d) Demostrar que un digrafo D es acíclico si y solo si D es trivial o D tiene un vértice con $d_{out}(v) = 0$ tal que $D \setminus \{v\}$ es acíclico.

Obs: decimos que tiene que tener un vértice con $d_{out}(v) = 0$ / $D \setminus \{v\}$ es acíclico porque puede dar un loop



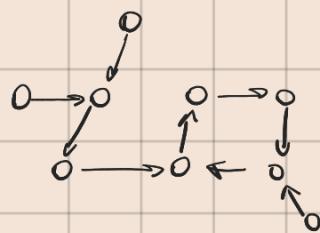
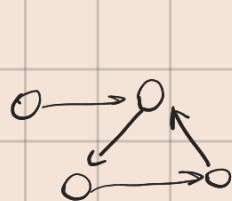
→) Sea D acíclico QVQ D es trivial o D tiene un vértice con $d_{out}(v) = 0$ / $D \setminus \{v\}$ es acíclico

Supongamos que D no es trivial y no tiene un vértice con $d_{out}(v) = 0$ / $D \setminus \{v\}$ sea acíclico → no es trivial y todos los vértices tienen $d_{out}(v) > 0$ o aquellos tq $d_{out}(v) = 0$, si hago $D \setminus \{v\}$ NO es acíclico → D tiene un ciclo → ABS!

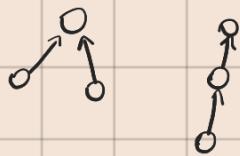
←) Sea D no trivial o con un vértice con $d_{out}(v) = 0$ / $D \setminus \{v\}$ es acíclico QVQ D es acíclico

Supongamos que D no es acíclico → D no es trivial y s $\exists v \in V(D) / d_{out}(v) = 0$ → $D \setminus \{v\}$ es acíclico → ABS!

- e) A partir del inciso anterior, diseñar un algoritmo que permita determinar si un grafo D tiene ciclos. En caso negativo, el algoritmo debe retornar una lista v_1, \dots, v_n de vértices tales que $d_{out}(v_i) = 0$ en $D \setminus \{v_1, \dots, v_{i-1}\}$ para todo i . En caso afirmativo, el algoritmo debe retornar un ciclo.



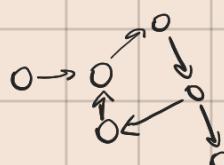
Estos eran los grafos que podían tener
antes.



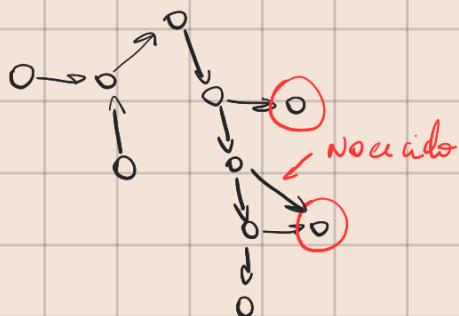
Estos son los grafos que puedo tener ahora

Si el grafo NO es trivial nevante un algoritmo. Puedo ir sacando vértice
hasta que el grafo quede con un ciclo o vacío

Supongamos



y



Puedo irme buscando las hojas del digrafo y ver qué pasa si las saco.

Si no aparecen nuevas \rightarrow tienen un ciclo

Si aparecen nuevas \rightarrow repito hasta quedarme sin vértice o tener un ciclo.

Algoritmo

1. Dado un digrafo D con vértice sin cuancas y un nodo inicial v , me fijo
en D es trivial. Si lo es devuelvo v .

2. Si no $\rightarrow D$ tiene hojas.

Si no tiene $\rightarrow D$ tiene un ciclo \rightarrow uso algoritmo anterior

Si tiene \rightarrow las guardo, las elimino y vuelvo a buscar si tiene hojas.

3. Si me quedo sin vértice \rightarrow ha acabado \rightarrow devuelvo donde sea que haya
guardado las hojas

- f) Explicar detalladamente (sin usar código) cómo se implementa el algoritmo del inciso anterior. El algoritmo resultante tiene que tener complejidad temporal $O(n + m)$.

1. Fijarme si la lista de aristas es vacía, si lo es entonces devuelvo el vértice v inicial. $O(1)$
2. Si no, paso de lista de adyacencias a lista de aristas. Voy a necesitar los vecindarios de salida N- y de entrada N+ $O(n+m)$
3. Armo una cola vacía H. $O(1)$

Acá voy a ir poniendo las hojas a medida que vaya encontrándolas. Uso una cola porque la prioridad siempre la van a tener las hojas que ya haya encontrado de antes y no las que vaya encontrando en el camino

4. Armo un vector G de n posiciones. $O(n)$

Voy a ir guardando los grados de salida de los vértices de D en este vector

5. En ese vector G para cada posición i pongo el grado del vértice i. $O(n)$

6. Inicializo una lista vacía R. $O(1)$

Voy a ir guardando las hojas acá.

7. Me fijo en G qué vértices tienen grado de salida 0 y los agrego a mi cola. $O(n)$

8. Mientras la cola no esté vacía

9. Busco el vecindario de entrada para el tope de mi cola $O(1)$

10. Resto a todos los grados de los vértices del vecindario de entrada 1 en mi vector G. Si alguno queda en 0, lo agrego a la cola. $O(d(v))$

11. Guardo el tope de mi cola en la lista R $O(1)$

12. Saco el tope $O(1)$

El peor caso del ciclo es que para cada vértice recorra todas las listas de vecinos entonces su complejidad es $O(n+m)$

13. Si la lista tiene tamaño n quiere decir que era acíclico, la devuelvo

14. Si no, quiere decir que hay un ciclo. Armo el grafo D\R y obtengo su lista de aristas para los vecindarios de salida. $O(n+m)$

15. Aplico el algoritmo del ejercicio más arriba sobre D\R. $O(n+m)$