Algoritmos sobre grafos

Marcos Kolodny

FAMAF-UNC

November 4, 2019

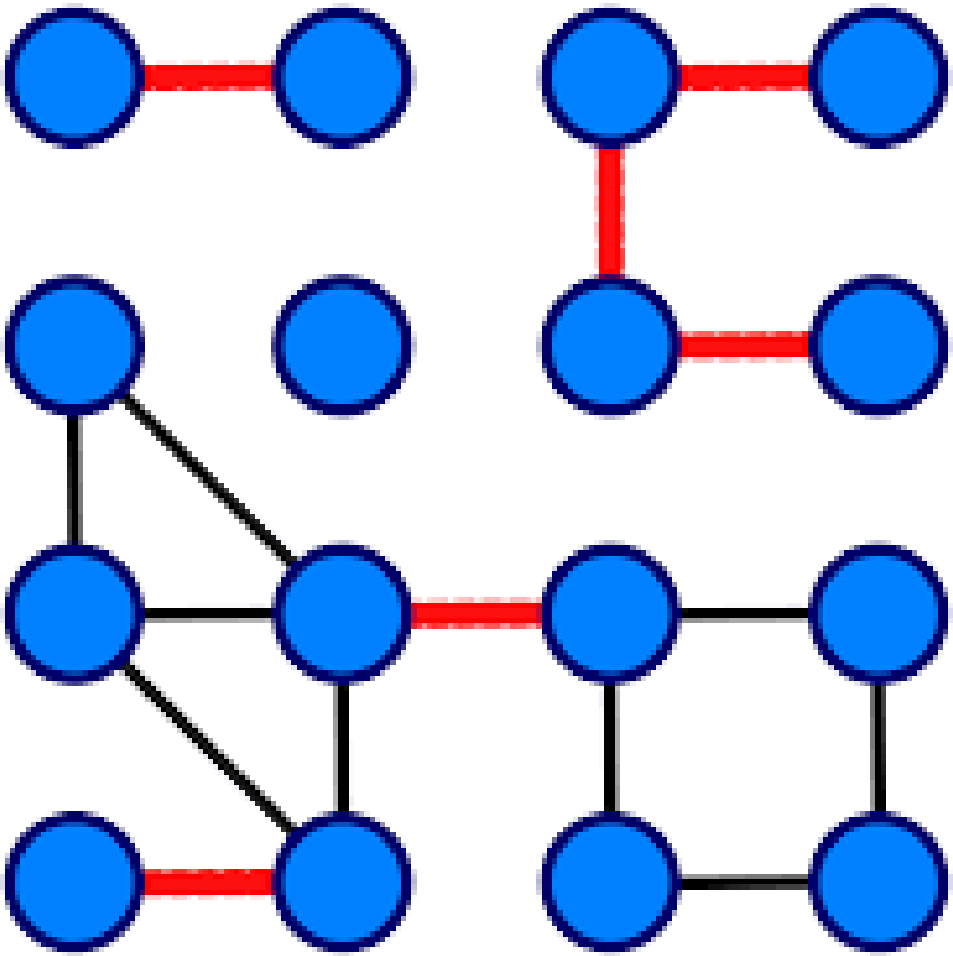
Algoritmos sobre grafos

En esta clase vamos a ver algunos de los algoritmos sobre grafos más clasicos. La idea es enfocarse en las aplicaciones y la utilidad de los mismos, y no tanto en la teoría detrás de cada uno.

Puentes

Dado un grafo no dirigido, decimos que un lado del mismo es un *puente* si, al borrarlo, aumenta la cantidad de componentes conexas (o en otras palabras, desconecta al menos un vértice del resto).

Puentes



Puentes

Una de las aplicaciones de los puentes es responder queries de "¿hay un único camino entre el nodo X y el nodo Y?"

Puentes

Otra cosa que suele ser muy útil es comprimir el grafo por puentes: comprimir en un único "nodo grande" a todos los nodos que se pueden alcanzar entre ellos sin pasar por puentes. Luego conectar estos nodos grandes mediante los puentes. Este nuevo grafo tiene propiedades interesantes:

- Todos los nodos dentro de una misma componente pertenecen a al menos un ciclo (o es un nodo aislado).
- El grafo comprimido es un árbol.
- Los lados del árbol generado son todos los puentes del grafo original.

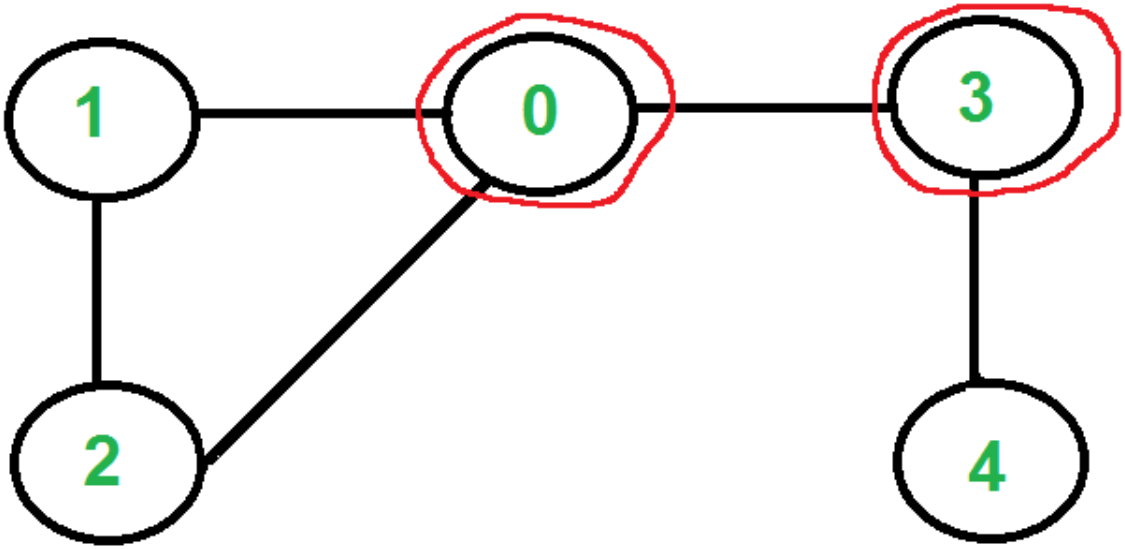
Puentes

Dado un grafo no dirigido y conexo, se permite agregar una arista entre dos nodos cualquiera. Dar la cantidad mínima de puentes que puede tener el grafo luego de agregar la arista óptima.

Puntos de articulación

Dado un grafo no dirigido, un punto de articulación es un vértice tal que, si se elimina del grafo junto a sus aristas, aumenta la cantidad de componentes conexas.

Puntos de articulación



Articulation points are 0 and 3

Componentes biconexas

Un grafo se dice biconexo si no contiene puntos de articulación.

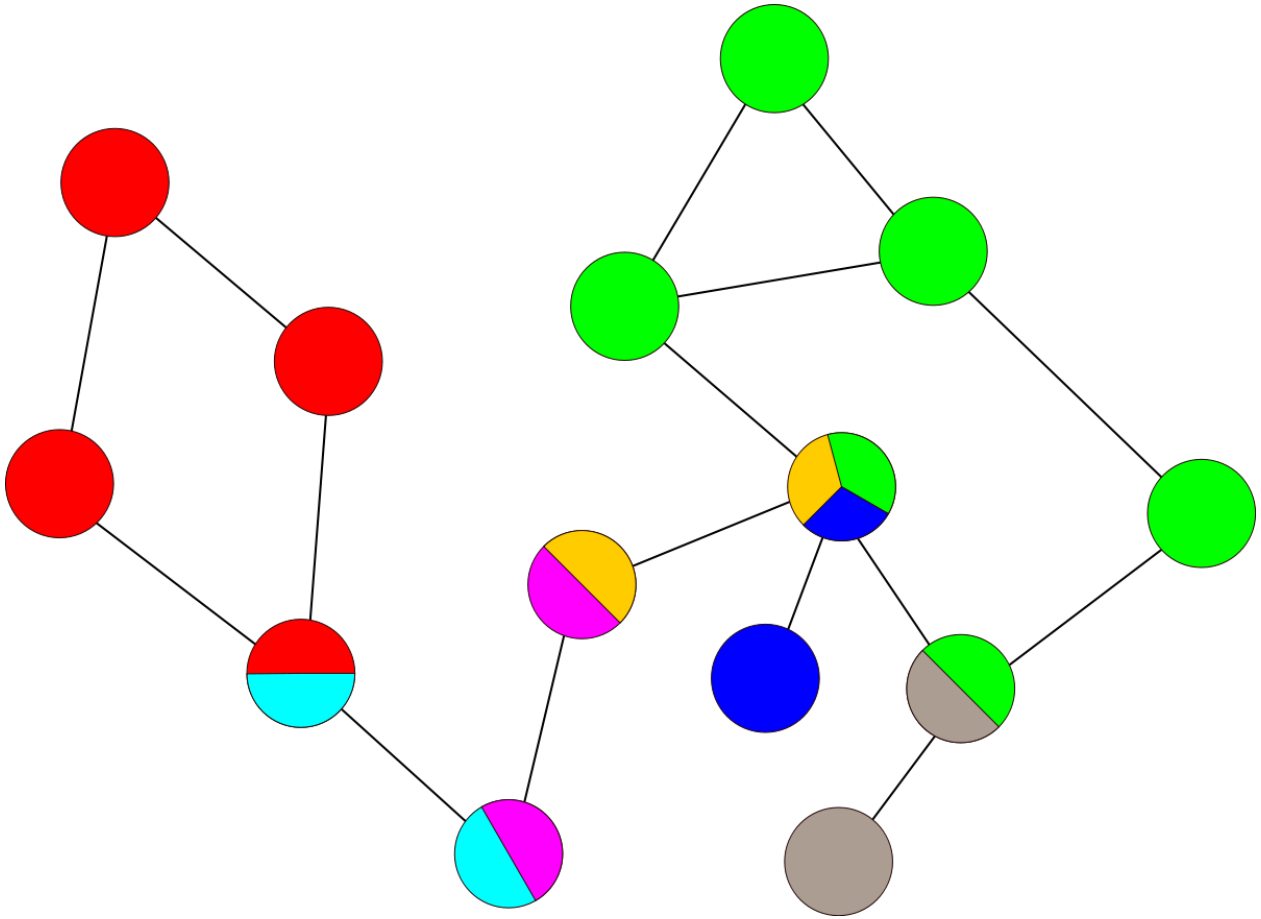
Una componente biconexa es un subconjunto de las **aristas** de un grafo tal que forman un subgrafo biconexo maximal. Existe una única descomposición de un grafo en componentes biconexas.

Componentes biconexas

Algunas propiedades interesantes son:

- Para todo par de lados en una componente, exite un ciclo que los contiene.
- Las componentes biconexas se encuentran separadas por los puntos de articulación. Esto implica que las articulaciones pueden pertenecer a más de una componente.
- Dados tres vertices A,B,C que posean lados en una misma componente, siempre existe un camino desde A hasta C que pasa por B.

Componentes biconexas



Componentes biconexas

Podríamos intentar comprimir el grafo en componentes biconexas, y unir los nodos comprimidos entre ellos si comparten un punto de articulación.

Pero, esto tiene un problema...

Componentes biconexas

Podríamos intentar comprimir el grafo en componentes biconexas, y unir los nodos comprimidos entre ellos si comparten un punto de articulación.

Pero, esto tiene un problema...

La cantidad de aristas del nuevo grafo es cuadrática.

Block cut tree

Pero no todo está perdido. Podemos solucionar el problema anterior agregando un vértice por cada punto de articulación además de uno por cada componente biconexa. De esa forma, logramos que la cantidad de aristas sea $O(n)$. Es más, el grafo generado es un árbol (y los árboles siempre traen buenas noticias en general).

Block cut tree

Dado un grafo no dirigido conexo, donde cada vértice tiene un valor, queremos poder responder queries del mínimo valor entre 2 vértices.

Algoritmo

Existe un algoritmo relativamente simple que, mediante un dfs, calcula los puentes, los puntos de articulación y las componentes biconexas de un grafo al mismo tiempo. Su complejidad es lineal en la cantidad de vértices del grafo.

Componentes fuertemente conexas

Dado un grafo **dirigido**, una componente fuertemente conexa es un subconjunto de vértices tal que para todo par de nodos (X, Y), existe al menos un camino desde X hasta Y.

Llamaremos a las componentes fuertemente conexas como SCC (por sus siglas en inglés).

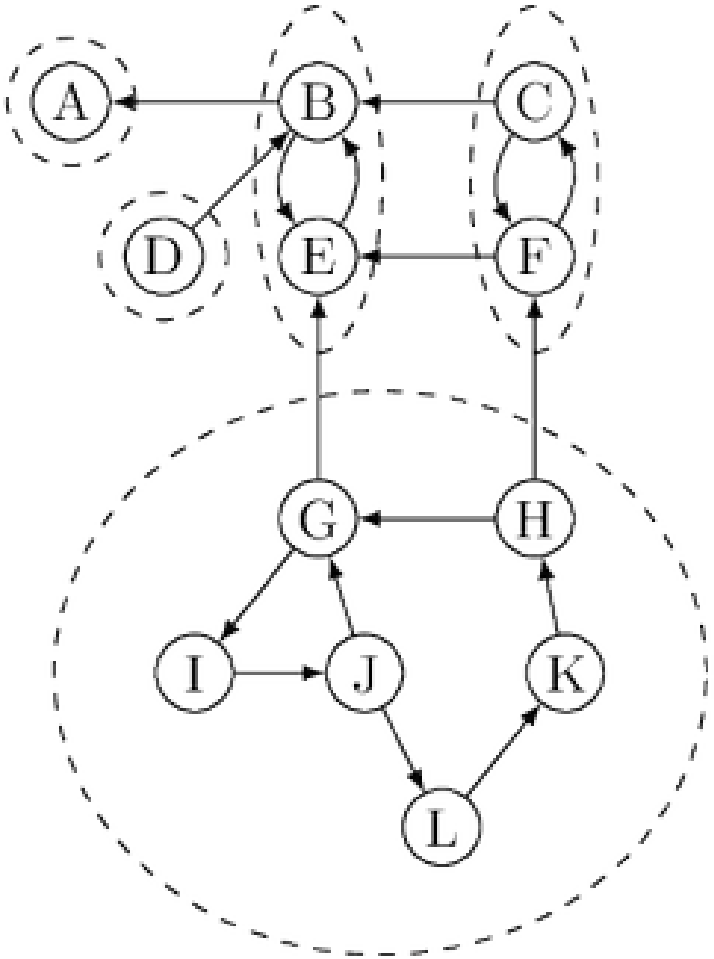
Existe una única partición de un grafo dirigido en SCC maximales, y el algoritmo de Tarjan calcula esto en $O(n)$.

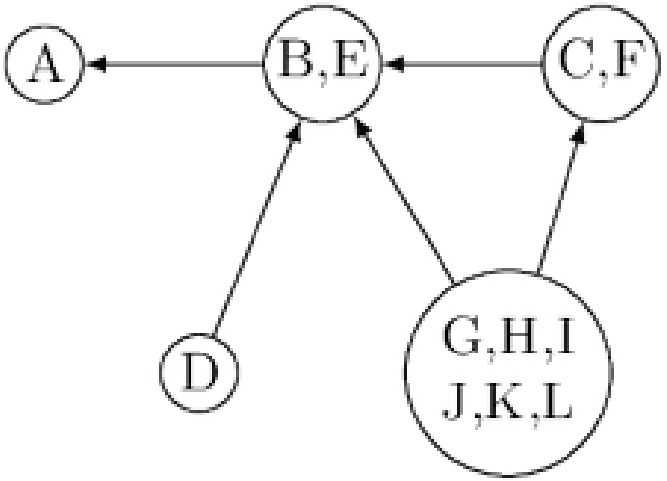
Componentes fuertemente conexas

Similar a la idea de los puentes, podemos comprimir un grafo por sus SCC, generando un nodo por cada SCC y uniendo dos de ellas mediante un lado (dirigido) si en el grafo original existía un lado entre vértices de esas componentes.

Resulta que esta compresión (que se llama grafo de condensación) genera un DAG (grafo dirigido **sin ciclos**). Como quizás saben, los DAG suelen ser muy útiles porque se puede aplicar en ellos técnicas como programación dinámica.

Componentes fuertemente conexas





2-SAT

Introducimos algunas definiciones:

- Un literal es una variable proposicional (variable booleana), o bien su negación.
- Una cláusula es una disyunción de literales. Algunos ejemplos son: $(A \vee B \vee \neg C)$ y $(P \vee \neg Q)$
- Una fórmula en forma normal conjuntiva es una conjunción de una o más cláusulas: $(A \vee B \vee \neg C) \wedge (P \vee \neg Q)$

2-SAT

2-SAT es un problema decisión donde se verifica si una fórmula, donde cada una de sus cláusulas es una disyunción de **dos** literales, es satisfactible.

Se conoce que si las cláusulas contienen más de 2 literales, el problema es NP-Completo.

2-SAT

2-SAT es un problema decisión donde se verifica si una fórmula, donde cada una de sus cláusulas es una disyunción de **dos** literales, es satisfactible.

Se conoce que si las cláusulas contienen más de 2 literales, el problema es NP-Completo.

Una buena noticia es que 2-SAT es resolvible en tiempo polinomial. Además, resulta que es una de las aplicaciones más importantes de componentes fuertemente conexas.

2-SAT

El algoritmo de SCC no solo nos permite decidir si una fórmula es satisfactible, si no que también podemos encontrar una asignación de valores a los literales tal que la fórmula evalua a *True*.

Recordemos que una implicación se puede ver como una disyunción: $(A \implies B) \equiv (\neg A \vee B)$.

La idea es modelar un grafo donde hay dos vértices por cada literal: uno para sí mismo, y otro para su negación. Luego, podemos ver cada cláusula como una implicación, y cada implica como un lado dirigido en el grafo.

Si luego de correr SCC vemos que un literal y su negación se encuentran en una misma componente, la fórmula no es satisfactible, porque (por transitividad del implica) tenemos que $(P \implies \neg P) \wedge (\neg P \implies P)$. Esto último claramente es falso, por lo que no hay asignación válida.

Caso contrario, basta con asignar valores de verdad distintos a la SCC de un literal y a la de su negación para lograr que la fórmula se evalúe a *True*.

Dudas/preguntas

¿Preguntas?