DFS Avanzado: Componentes biconexas y fuertemente conexas

Agustín Santiago Gutiérrez

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Training Camp Argentina 2017

Contenidos

Contenidos

DFS

- "*Depth-first search yields valuable information about the structure of a graph.*"

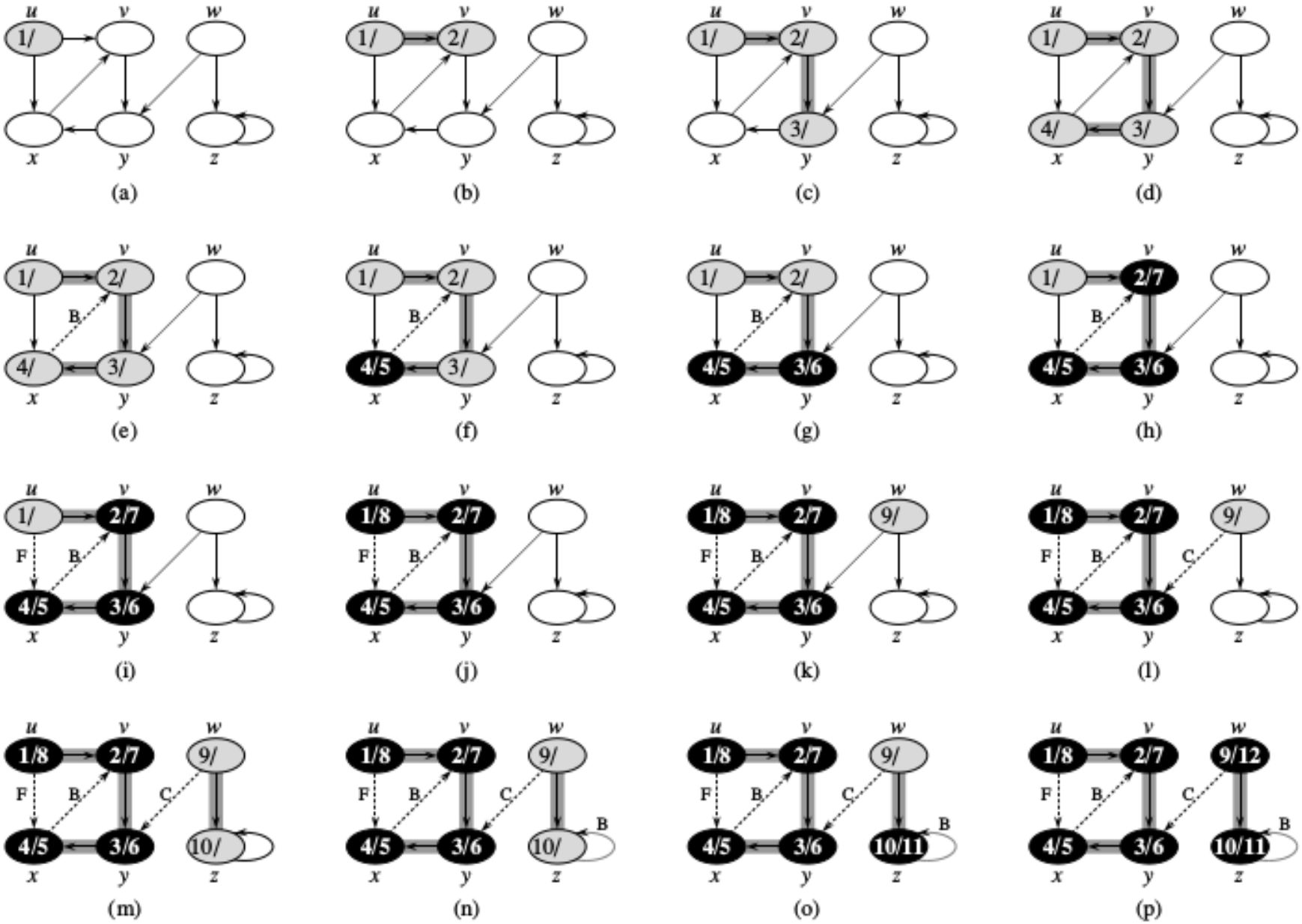
  Introduction to Algorithms, Cormen et al.

- A diferencia del BFS, que suele utilizarse con la misión específica de resolver el problema de caminos mínimos desde un origen s, el algoritmo de DFS suele usarse para obtener información útil sobre el grafo en sí, que puede ser luego utilizada por algoritmos posteriores.

Idea general

- La idea del DFS consiste en recorrer el grafo caminando por las aristas, utilizando siempre las aristas disponibles (aún no utilizadas) del último nodo descubierto.
- Para visualizar más fácil la ejecución del algoritmo, es conveniente pensar en que los nodos se pintan de tres colores: **Blanco, Gris y Negro**.
- Los nodos inician todos pintados de blanco.
- Al descubrir un nodo y comenzar a procesarlo, se lo pinta de **gris**. Mientras haya nodos **blancos**, se hace una exploración de DFS desde cualquier nodo **blanco**.
- Desde el nodo actual en procesamiento, se exploran las aristas salientes, y si alguna llega a un nodo **blanco** nuevo, se pinta de **gris** y se sigue explorando desde allí recursivamente.
- Luego de considerar y eventualmente recorrer todas las aristas de un nodo, se completa su procesamiento y se pinta de **negro**, volviendo la recursión a su **padre**.

Ejemplo


u v w
1/
x y z
(a)
1/ 2/
(b)
1/ 2/ 3/
(c)
1/ 2/ 4/ 3/
(d)
B
(e)
4/5
(f)
4/5 3/6
(g)
2/7
(h)
F
(i)
1/8
(j)
9/
(k)
C
(l)
10/
(m)
(n)
10/11
(o)
9/12
(p)

Paréntesis

- Además, es muy útil para razonar sobre el algoritmo, y para utilizar en algoritmos posteriores, asociar a cada nodo dos "timestamps":
- Un primer timestamp $d[i]$, que para cada nodo i, indica el tiempo en que fue **descubierto** (se pintó de gris).
- Un segundo timestamp $f[i]$, que para cada nodo i, indica el tiempo en que fue **terminado** (se pintó de negro).
- El timestamp es simplemente una variable global, por ejemplo que comienza en 0 y se incrementa hasta $2n$ durante la ejecución del algoritmo.
- Nos permite ordenar los $2n$ eventos de descubrimiento y finalización de un nodo de manera total en el tiempo.

Paréntesis (continuado)

- Una propiedad muy importante y útil para razonar, es que los tiempos de finalización y terminación de los distintos nodos presentan **estructura de paréntesis**.
- Es decir, si armamos un string de longitud $2n$ con paréntesis que abren en las posiciones $d[i]$ y paréntesis que cierran en las $f[i]$, tendremos una cadena bien parenteseada.
- Por ejemplo, una propiedad muy importante es que un descendiente de un nodo en un arbol de dfs, necesariamente tiene sus parentesis contenidos en el de su ancestro.

Clasificación de las aristas

El algoritmo de DFS nos induce una clasificacion de aristas en 4 tipos:

- **Tree edge**
  - Viaja a un nodo blanco
  - Es con la que se descubre un nodo por primera vez
  - Viaja al **hijo** del nodo actual en un árbol de DFS
- **Back edge**
  - Viaja a un nodo gris
  - Viaja a un **ancestro** del nodo actual
- **Forward edge**
  - Viaja a un nodo negro con descubrimiento posterior al nodo actual
  - Viaja a un **descendiente** (no necesariamente hijo) del nodo actual
  - *Solo aparece en grafos dirigidos*
- **Cross edge**
  - Viaja a un nodo negro que finaliza antes que el descubrimiento del nodo actual
  - No viaja ni a un ancestro ni a un descendiente
  - *Solo aparece en dirigidos*

Observaciones útiles

- Los descendientes de un nodo x serán exactamente los nodos blancos alcanzables por un camino de nodos blancos con origen en x, en el instante en que se descubre (*white-path-theorem*).
- Un grafo es acíclico (dirigido o no) si y solo si un recorrido de DFS no encuentra ninguna back-edge.
- El dfs genera un spanning forest (dfs-forest), donde cada árbol está formado por las tree-edges.
- Si el grafo es no dirigido, el DFS construye un dfs-forest con un spanning tree de cada componente conexa.
- Si el grafo es dirigido, no hay una relación clara entre los árboles del dfs-forest y las “componentes” del grafo original.

Observaciones útiles (¡Más!)

- Las back-edges siempre están involucradas en un ciclo (concretamente, el que se completa con las tree edges que bajan en sentido inverso)
- Las demás (cross-edges, tree-edges y forward-edges) forman un subgrafo acíclico (ya que por definición, siempre van de un nodo a otro con un tiempo de finalización menor)
- Topological sort: Es tomar los vertices en orden inverso de finalizacion, gracias a que (en un DAG):

$$a \to b \Rightarrow f[a] > f[b]$$

- Notar que no hace falta correr un sort: si cada vez que finalizamos un nodo lo agregamos a una lista, quedan al final en orden de finalización.

Problema atacable con DFS

- Un grafo dirigido es *singly-connected* si para cada par de nodos (a, b) existe a lo sumo un camino simple desde a hacia b. Dar un algoritmo $O(n(n+m))$ para decidir si un grafo es *singly-connected*.

Problema atacable con DFS

- Un grafo dirigido es *singly-connected* si para cada par de nodos (a, b) existe a lo sumo un camino simple desde a hacia b. Dar un algoritmo $O(n(n+m))$ para decidir si un grafo es *singly-connected*.
- Solución: Un grafo es *singly-connected* si y solo sí al correr un dfs desde cada nodo, nunca aparecen **forward edges** ni **cross edges** (dentro de un mismo DFS-tree).

Tarea

Para pensar en el hogar:

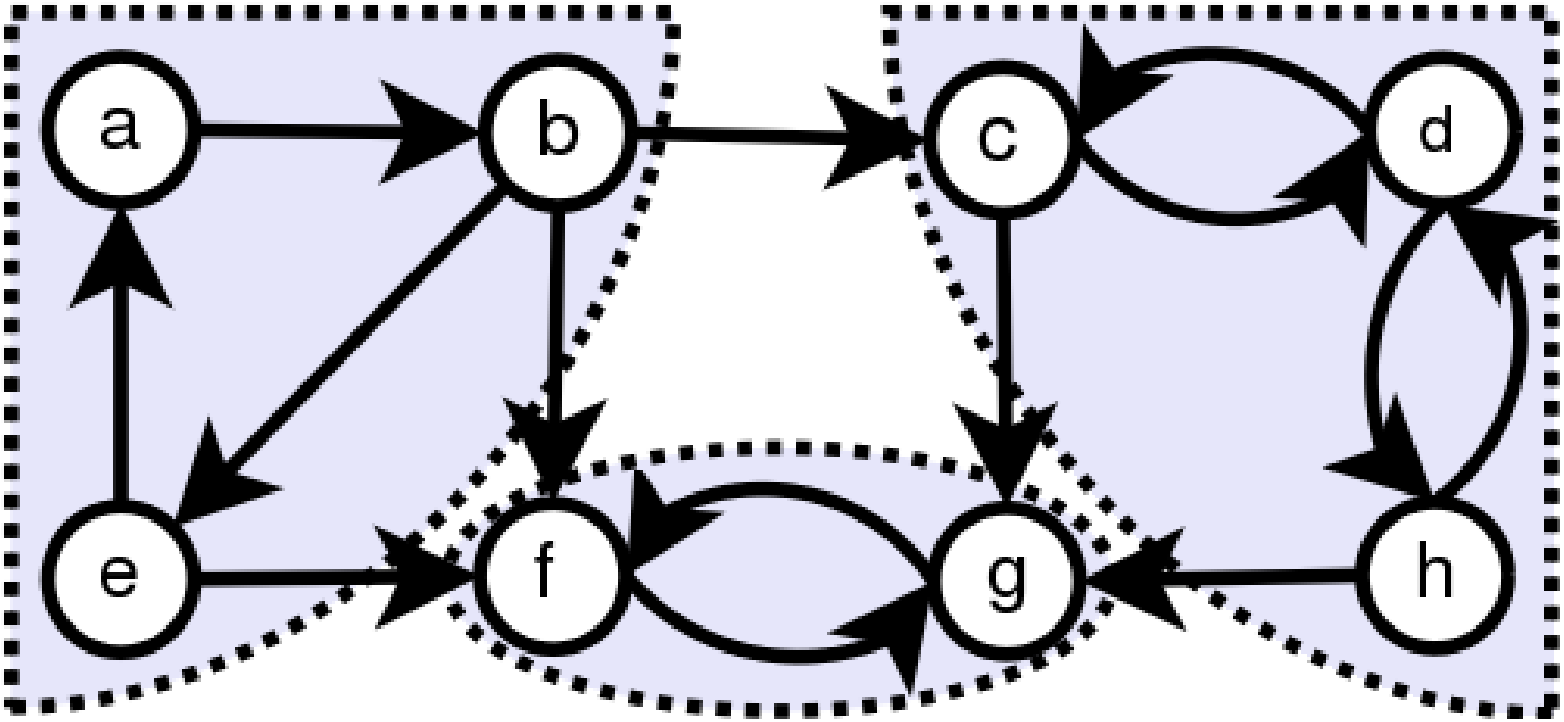
- En un grafo dirigido, calcular para cada vértice v, el mínimo índice posible de un vértice w que sea alcanzable desde v, en tiempo lineal, es decir, $O(|V| + |E|)$.

Contenidos

Definiciones

- Un grafo dirigido es **fuertemente conexo** si para todo par de nodos *a*,*b* existe un camino (dirigido) desde *a* hasta *b*.
- En un grafo no dirigido, una **componente fuertemente conexa** es un subgrafo fuertemente conexo maximal.

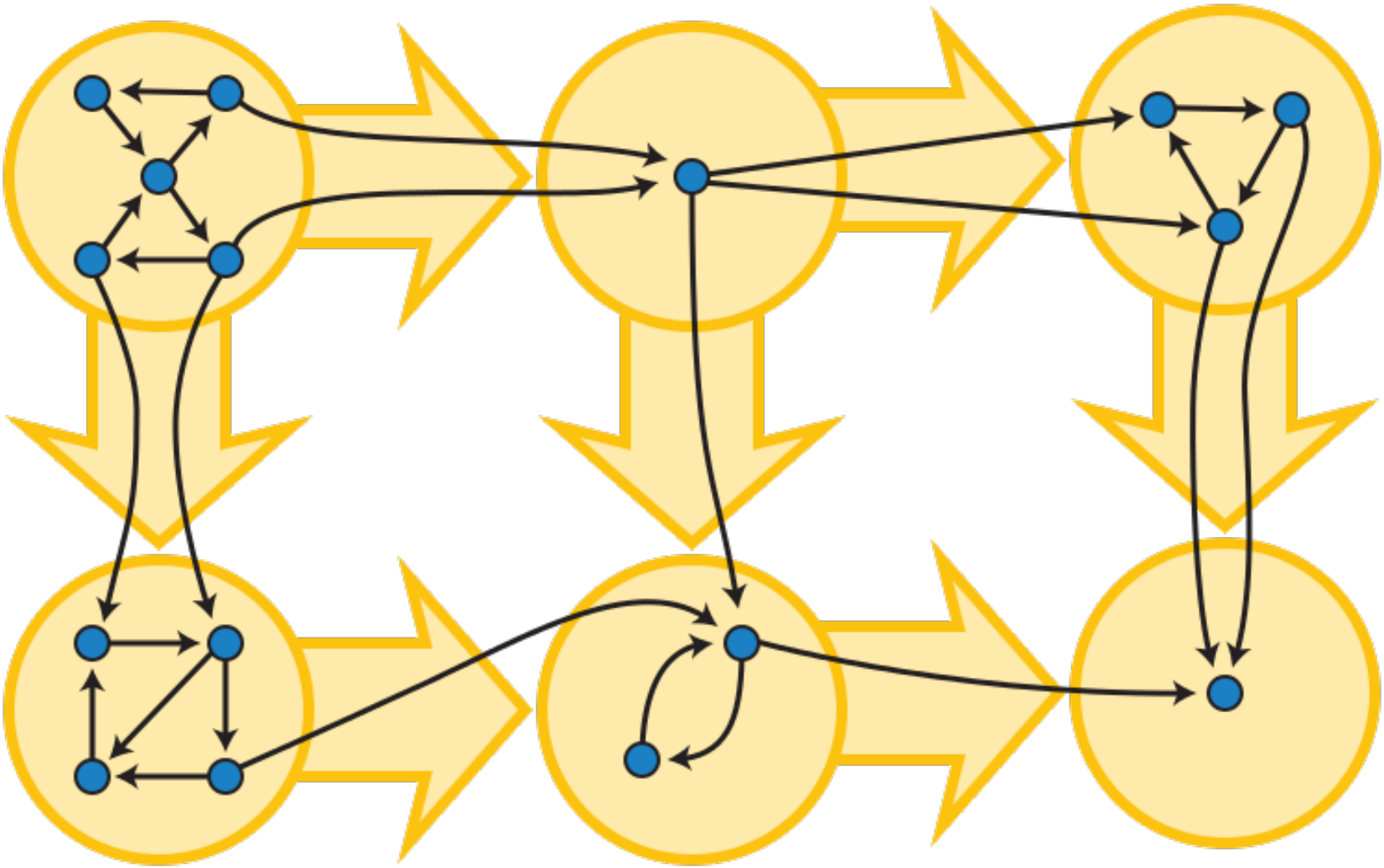
Dibujito



Observaciones

- Una componente fuertemente conexa es un subgrafo inducido del grafo original.
- Las componentes fuertemente conexas particionan los nodos (según la relación de equivalencia de alcanzabilidad mutua).
- Si compactamos cada componente fuertemente conexa a un nodo (manteniendo las aristas que cruzan entre componentes), obtenemos un DAG llamado la *condensación* del grafo original.

Condensación



Cálculo de las componentes fuertemente conexas

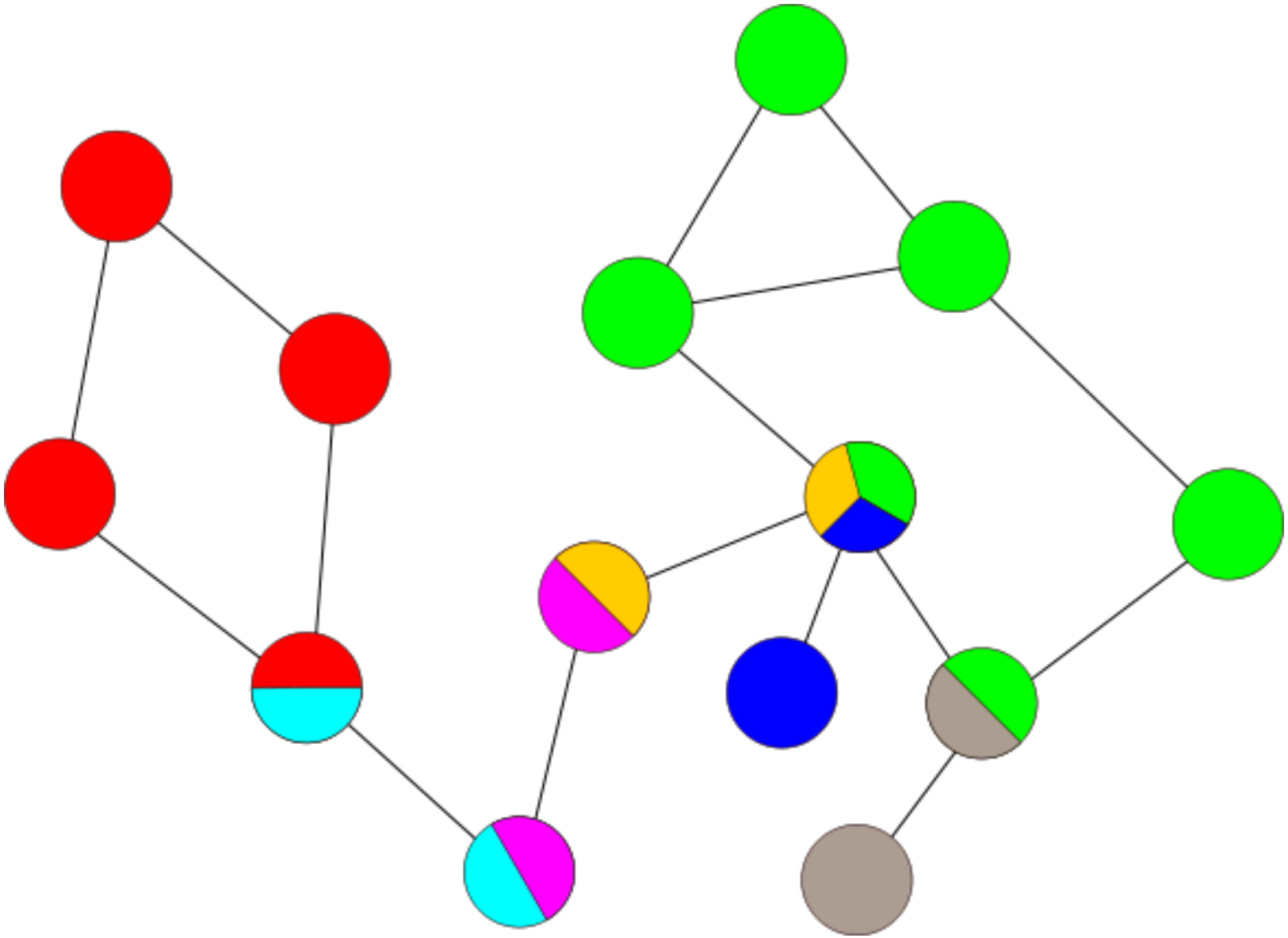
- Aunque hay otros mejores, el algoritmo eficiente más sencillo es el de Kosaraju.
- Con un primer DFS, obtenemos la lista de nodos en orden inverso de finalización (sería el orden topológico si fuera un DAG).
- Luego, damos vuelta todas las aristas de G (obteniendo G^t). Esto no cambia las componentes fuertemente conexas, e invierte todas las aristas de la condensación de G.
- Notar que el primer nodo de la lista debe pertecer a una componente sin aristas salientes en la condensación de G^t. Luego un DFS desde él alcanza a exactamente a los elementos de esa componente.
- De la misma manera, si ahora continuamos realizando un recorrido cada vez desde el siguiente nodo sin recorrer, los alcanzables forman una nueva componente fuertemente conexa.

Contenidos

Definiciones

- En un grafo no dirigido, un **punto de articulación** es un **nodo** tal que al removerlo del grafo, la cantidad de componentes conexas aumenta.
- En un grafo no dirigido, un **puente** es un **eje** tal que al removerlo del grafo, la cantidad de componentes conexas aumenta.
- Un grafo no dirigido es **biconexo** si es conexo y no tiene puntos de articulación.
- En un grafo no dirigido, una **componente biconexa** es un subgrafo biconexo maximal.

Biconexas (Dibujito)



Observaciones

- Los nodos aislados son un caso especial. Conviene separarlos, y contarlos o no como componentes según convenga.
- Los puentes son exactamente las componentes biconexas de 2 nodos (y necesariamente, una arista).
- Las componentes biconexas **no particionan los nodos** (a diferencia de las componentes conexas o fuertemente conexas).
- Las componentes biconexas **particionan las aristas** del grafo (ignorando los nodos aislados), de acuerdo a la relación de equivalencia de cociclidad (pertenencia a un mismo ciclo simple).

Observaciones (más)

- Podríamos obtener algo parecido a la condensación de un grafo dirigido para este caso.
- Si miramos el dibujo y nos imaginamos que cada componente biconexa se transforma en un nodo, lo que queda tiene “Pinta de árbol”.
- Sin embargo, simplemente contraer cada componente a un único nodo y unir componentes que se tocan no funciona, pues el resultado no es árbol.

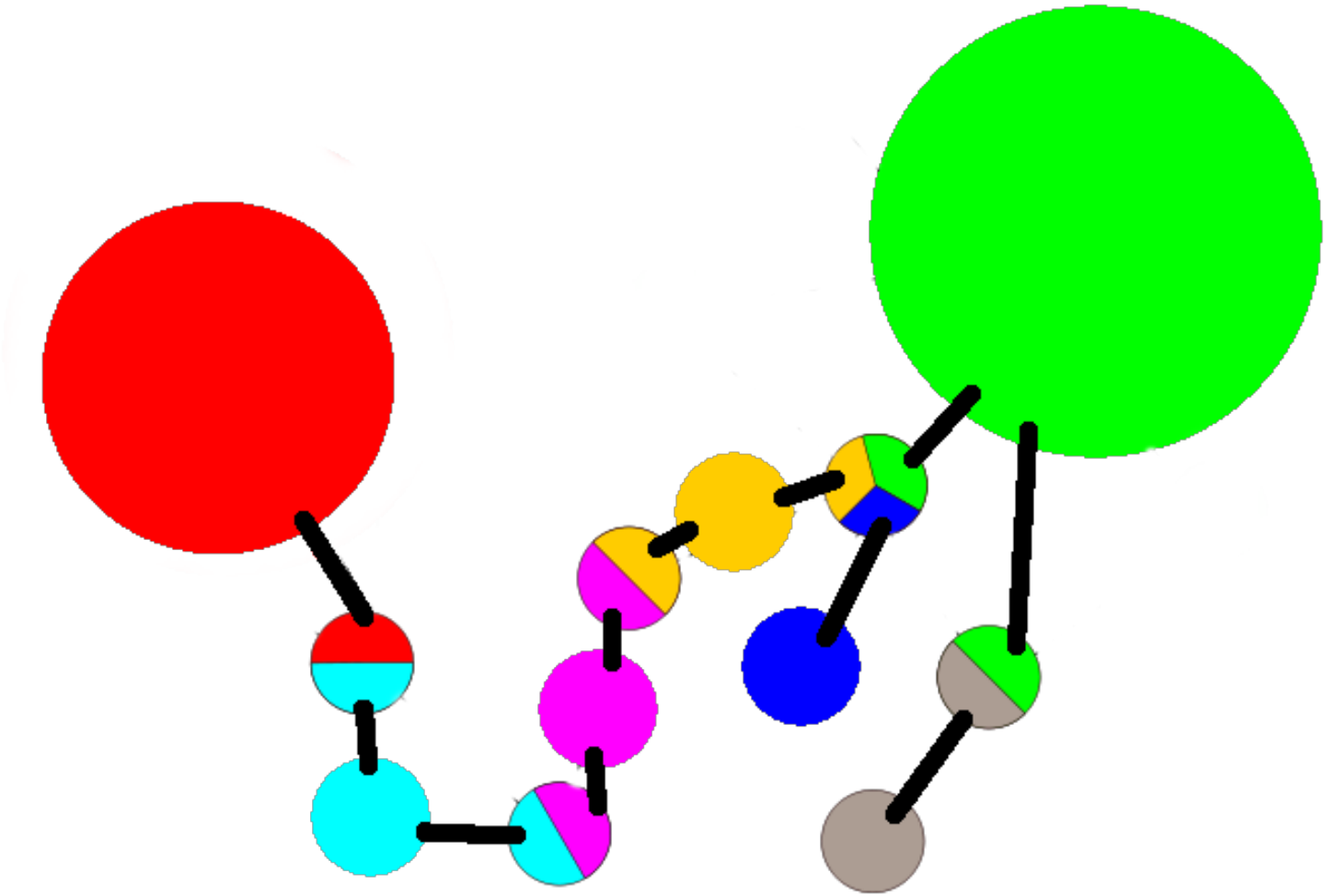
Observaciones (más)

- Podríamos obtener algo parecido a la condensación de un grafo dirigido para este caso.
- Si miramos el dibujo y nos imaginamos que cada componente biconexa se transforma en un nodo, lo que queda tiene “Pinta de árbol”.
- Sin embargo, simplemente contraer cada componente a un único nodo y unir componentes que se tocan no funciona, pues el resultado no es árbol.
- Solución: utilizar un nodo por cada **componente biconexa**, y también un nodo por cada **punto de articulación**.

Block-cut tree

- Conectamos un punto de articulación a las componentes biconexas que lo contienen (siempre serán al menos 2). El resultado es un árbol.
- Notar que si bien todo árbol es bipartito, aquí la bipartición tiene un significado claro en relación al problema: por un lado puntos de articulación, por otro componentes.
- Además, todas las hojas están en el mismo conjunto de la bipartición, lo cual no ocurre en cualquier árbol, ya que en nuestro caso los puntos de articulación nunca son hoja.
- A este árbol se lo conoce como el *block-cut tree* del grafo.
- En realidad lo anterior es un árbol si el grafo original es conexo. Sino, se obtiene un bosque, con un árbol de estos por componente conexa.

Block-cut tree (Dibujito)



Cálculo mediante DFS

- La idea central es computar mediante recursión durante el recorrido del DFS, un valor $low[i]$ que para cada nodo, indique la menor distancia de la raíz del árbol de DFS actual a la que es posible saltar, desde alguna parte del sub-arbol de DFS con raíz en i.
- Este valor puede computarse simplemente como el mínimo entre el *low* de los hijos del nodo actual, y la profundidad mínima a la que llega una back-edge que sale del nodo actual.

Cálculo mediante DFS (puentes)

- Observación: Un tree-edge del nodo k a uno de sus hijos x es puente si y solo sí, $low[x] >= depth[x]$, es decir, no existe ningún back-edge que permita salir del subarbol con raíz en x.
- Observación: Un back-edge nunca puede ser puente.

Cálculo mediante DFS (nodos)

- Observación: La raíz de un árbol de DFS es un punto de articulación si y solo sí tiene más de un hijo.
- Observación: Un nodo x distinto de la raíz es punto de articulación, si y solo sí, para alguno de sus hijos y, se cumple $low[y] >= depth[x]$.

Cálculo mediante DFS (componentes biconexas)

- Para calcular las componentes biconexas, basta agregar una pila al DFS anterior:
- Cada vez que recorremos una arista, agregarla a la pila.
- Cada vez que volvemos de una llamada recursiva por un eje x, y, con x padre de y, chequeamos si ahí termina una componente biconexa, es decir, $low[y] >= depth[x]$
- Si ese es el caso, desapilamos aristas hasta desapilar la arista x, y, que fue la primera que pusimos al bajar y marca el comienzo de la componente. Todas esas forman la componente biconexa.

Contenidos

Referencias

- *Introduction to Algorithms, 2nd Edition*. MIT Press.
  Thomas H. Cormen
  Charles E. Leiserson
  Ronald L. Rivest
  Clifford Stein
  Sección 22 (DFS y temas relacionados)

¿Preguntas?