Puntos de Articulación y aristas puente

Bibliografía: “Introduction to Algorithms”. Third Edition.

The MIT Press. Massachusetts Institute of Technology. Cambridge, Massachusetts 02142.

http://mitpress.mit.edu

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein

Facultad de Matemática y Computación
Universidad de La Habana

Conectividad (Grafo *k*-conexo)

Un grafo tiene **conectividad k**, si al eliminar del mismo **hasta k-1 vértices** **(** *y las aristas que inciden en él* **)** **cualesquiera a la vez** **!!**, el mismo NO se desconecta

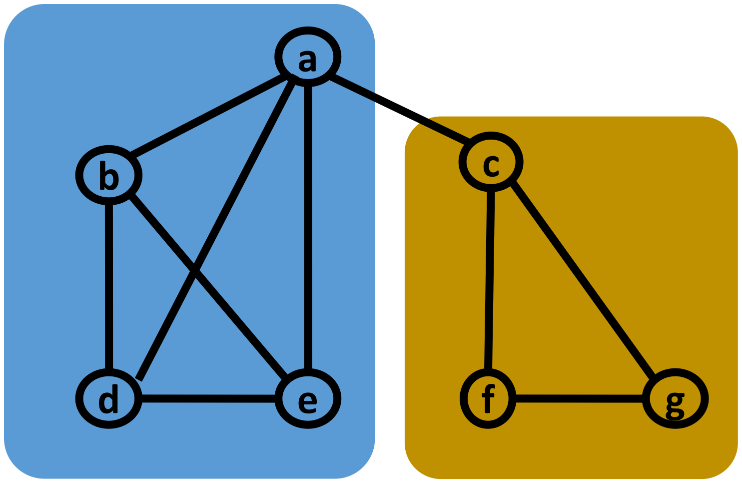
A un grafo 2-conexo, se le llama **grafo biconexo**

Ejemplos de grafos biconexos

Conectividad (componente *k*-conexa)

Se le llama **componente *k*-conexa** a un subgrafo *k*-conexo maximal de G

A una componente 2-conexa se le llama **componente biconexa**

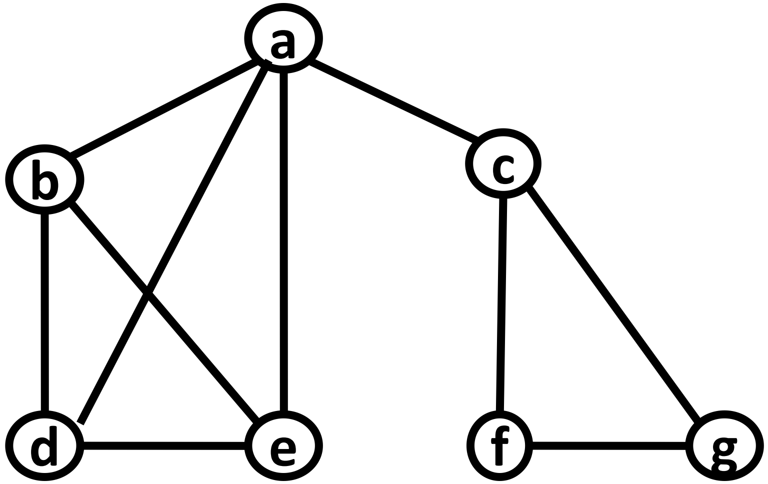


COMP. BICONEXAS

Los subgrafos formados por *b,d,e,a* y *c,f,g* son componentes biconexas de G

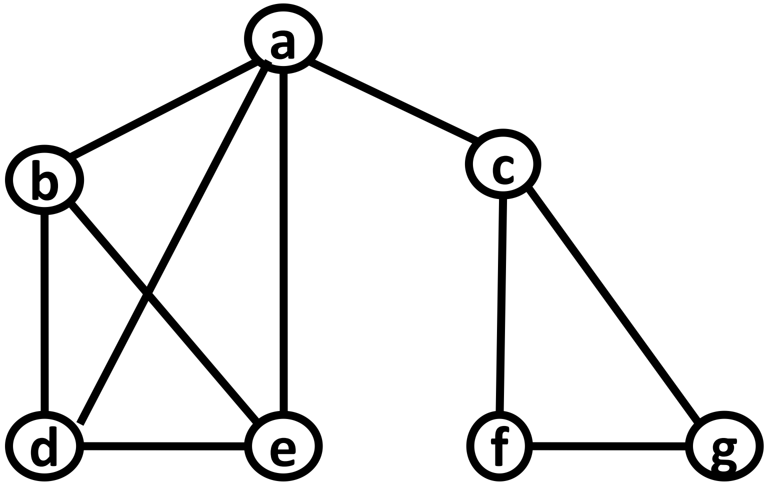
Punto de articulación

Un **punto de articulación** de un grafo G=(V, E), es un vértice $v \in V$**,** tal que, al eliminar *v* de G, y todas las aristas incidentes en él, se divide una **componente conexa** en **dos o más componentes conexas**



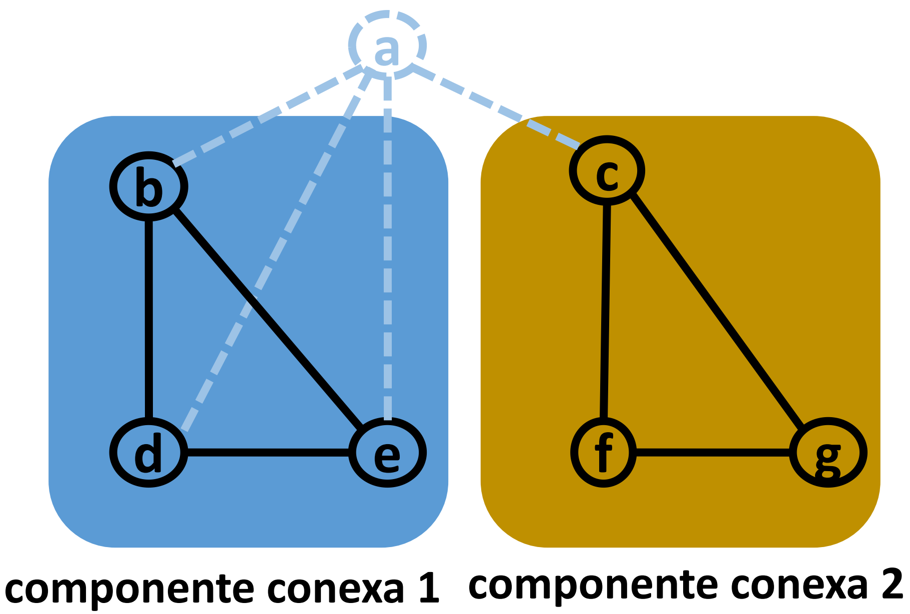
¿Quienes serían los puntos de articulación en este grafo ?

Punto de articulación



PTO DE ARTICULACIÓN

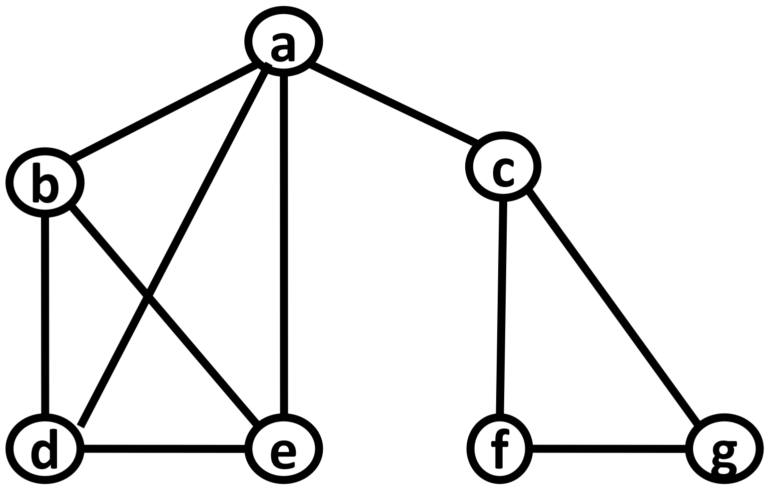
El vértice ***a*** es un punto de articulación

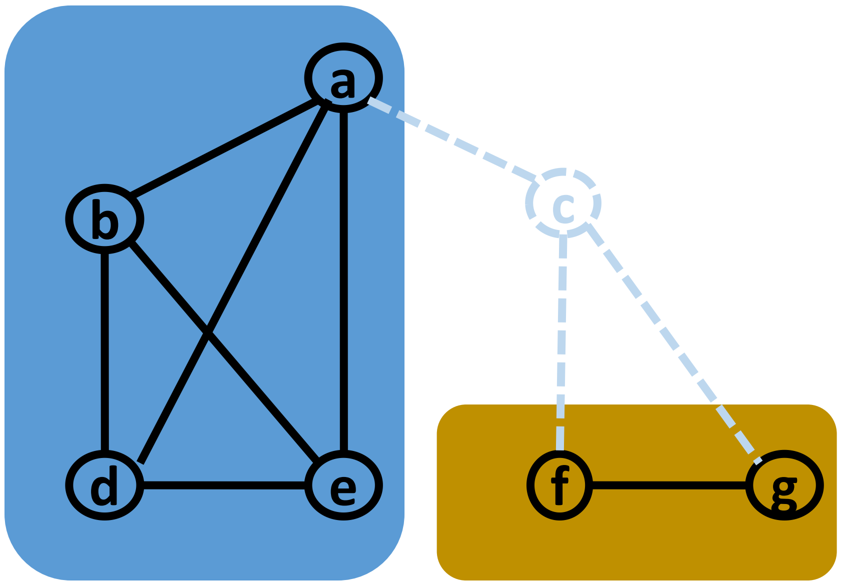


GRAFO BICONEXO

Note que un **grafo es biconexo, si y solo si, no tiene puntos de articulación**

Punto de articulación





componente conexa 1 **componente conexa 2**

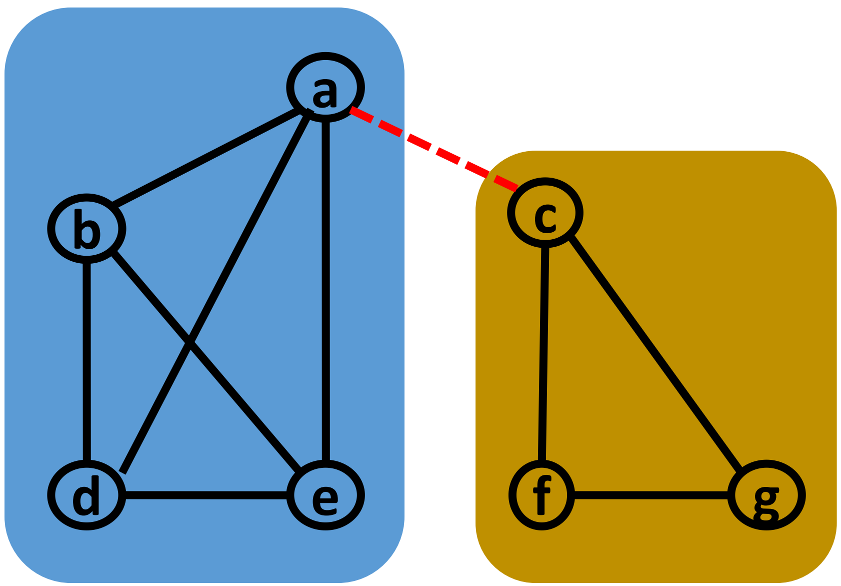
PTO DE ARTICULACIÓN

El vértice ***c*** es un punto de articulación

Cualquier otro vértice que se elimine (que no sea ***a*** o ***c***) no provoca la división de la componente conexa

Arista puente

Dado un grafo $G =< V, E >$ no dirigido, se dice que $e \in E$ es una **arista puente** si al eliminarla se desconecta el grafo G (aumenta en uno la cantidad de componentes conexas del mismo)

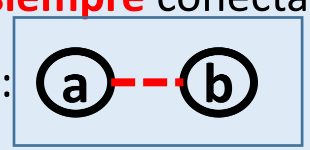


componente conexa 1 **componente conexa 2**

ARISTA PUENTE

$< a, c >$ desconecta al grafo en dos components conexas

NOTA: Las aristas puentes **no siempre** conectan dos puntos de articulación

Ejemplo: 

G

Detección de puntos de articulación

PROBLEMA

Determinar los ***puntos de articulación***, y las ***aristas puente***, para un **GRAFO** **CONEXO** ***G***

Ejemplos de aplicación práctica

En una red (**eléctrica**, **mapa de calles**, etc) determinar:

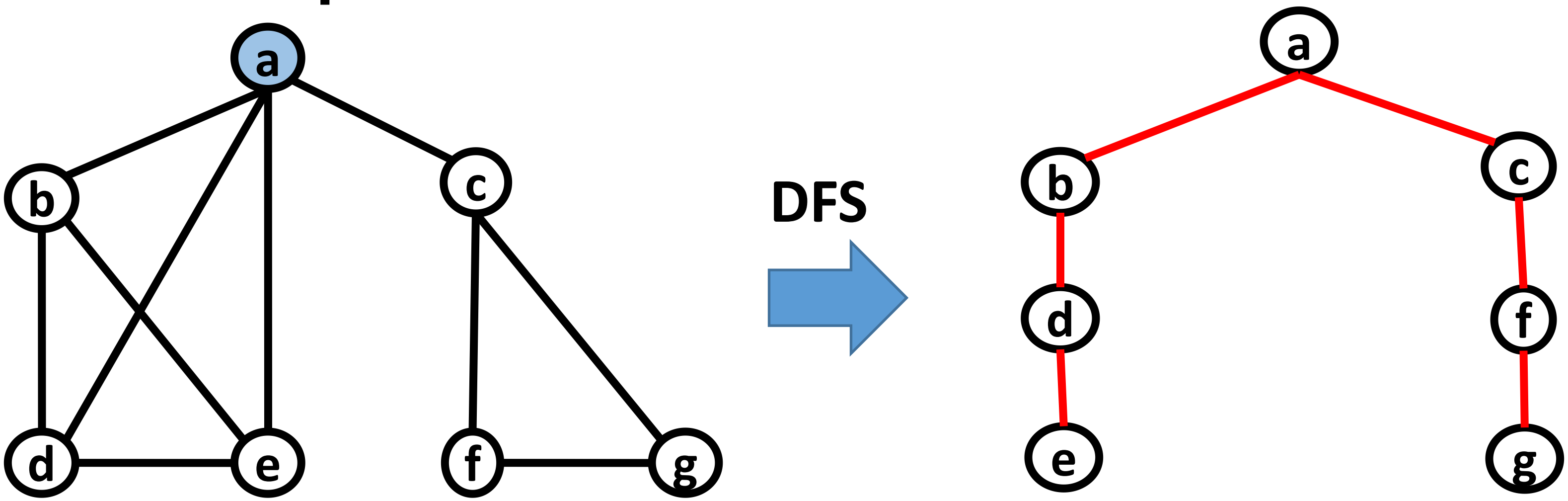
- Cuáles conexiones (**cables**, **calles**) son **críticas**
- Cuáles intersecciones (repetidores, esquinas) son **críticas**

crítico $\Rightarrow$ que desconectan la red

Mientras más alto sea el *k* en el grafo *k*-conexo que representa la red, más confiable será la misma (más conexiones pueden fallar sin afectar el servicio en la red)

Análisis para el caso en que el *origen* es un punto de articulación

Aplicando un DFS sobre el Grafo



árbol abarcador en profundidad

IDEA INTUITIVA:
Si se toma cualquier **punto de articulación** como punto de partida del **DFS**, se forma un árbol cuya **raíz tiene, al menos, dos hijos**

Lema 1

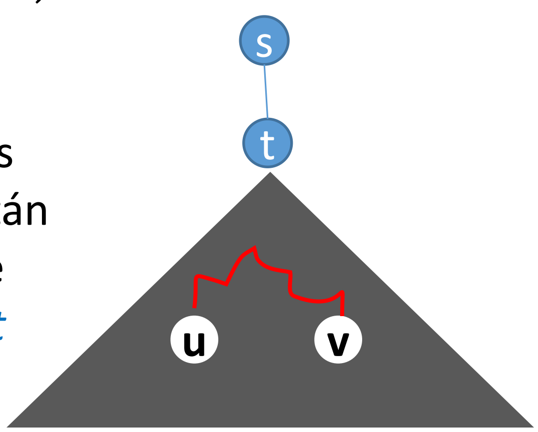
Sea G=(V, E) conexo y no dirigido. La raíz del árbol abarcador en profundidad G_π es un punto de articulación $\Longleftrightarrow$ **tiene, al menos, dos hijos en G_π**

DEMOSTRACIÓN

($\Leftarrow$) Sea s la raíz de G_π:

- Si ***s* no tiene hijos**: s **NO** es punto de articulación por definición (el grafo resultante, tras eliminar ese vértice, es vacío)
- Si ***s* tiene un único hijo**: sea $< s, t >$ la ***arista de árbol*** que se forma entre s y su único hijo *t***.** Sean u, v dos vértices cualesquiera diferentes de s. Como G es conexo, entonces, existe un ***camino simple*** entre u y v en $G_\pi \Rightarrow$ si en G hay aristas de retroceso entre los restantes vertices de V y s, entonces, al eliminar $< s, t >$, G no se desconecta

u y *v* son dos vértices cualesquiera que están en el árbol que tiene como raíz al vértice *t*

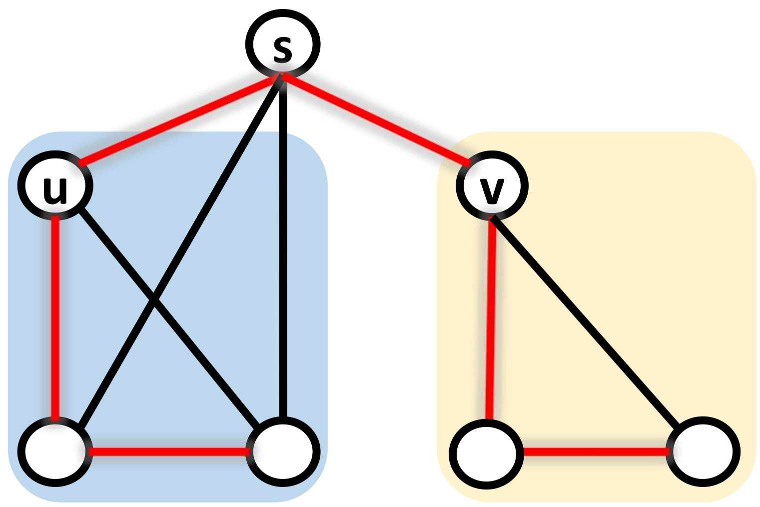


OJO: En este caso, todos los vertices, excepto s, son descendientes de t

Lema 1 … continuación de la demostración

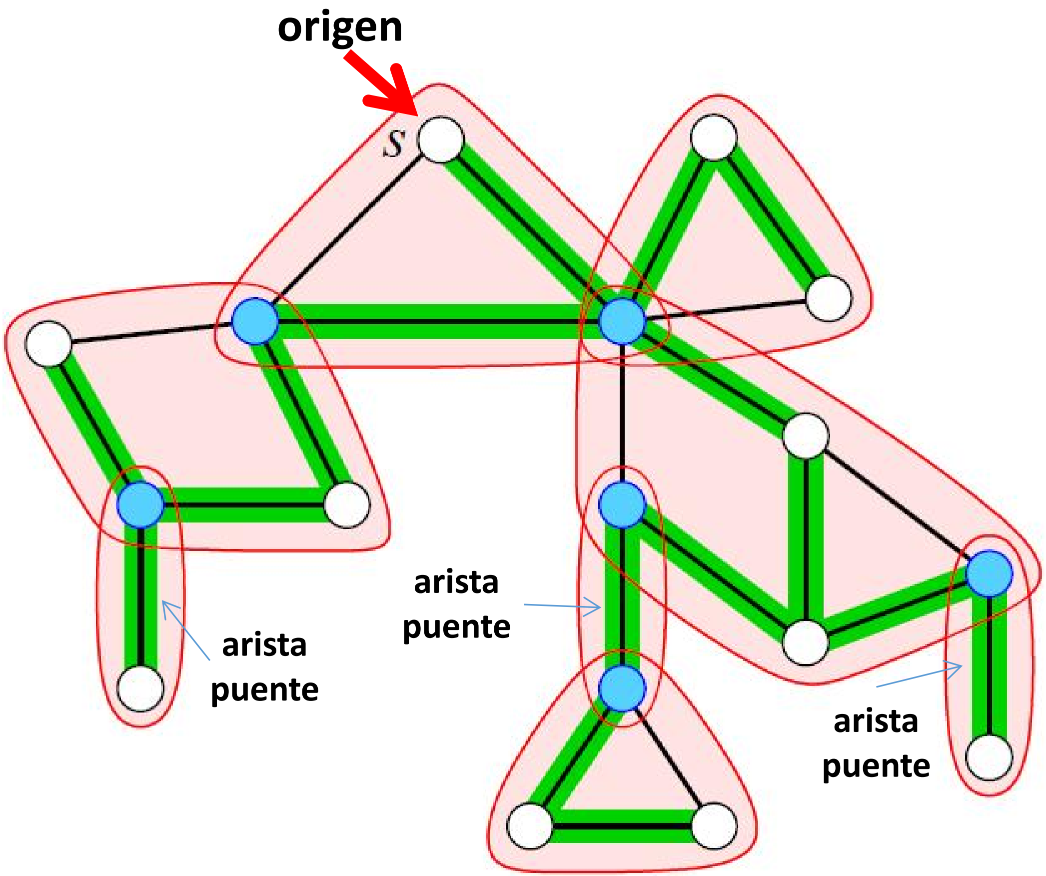
DEMOSTRACIÓN (⇐) (continuación)

- Si s **tiene 2 o más hijos**. Sean $u,\ v$ dos hijos cualesquiera de s. Cualquier camino entre u y v tiene que pasar por s (**en G no existe otra arista que conecte a un vértice de un subárbol con uno del otro subárbol**) por lo que s es punto de articulación.



(⇒) Trivial. Si es punto de articulación, desconecta el grafo en dos o más componentes conexas por lo que $\boldsymbol{s}$ **tiene que tener una arista en** $\boldsymbol{G_\pi}$ **por cada CC**

Punto de Articulación, Componente Biconexa y Arista Puente



LEYENDA

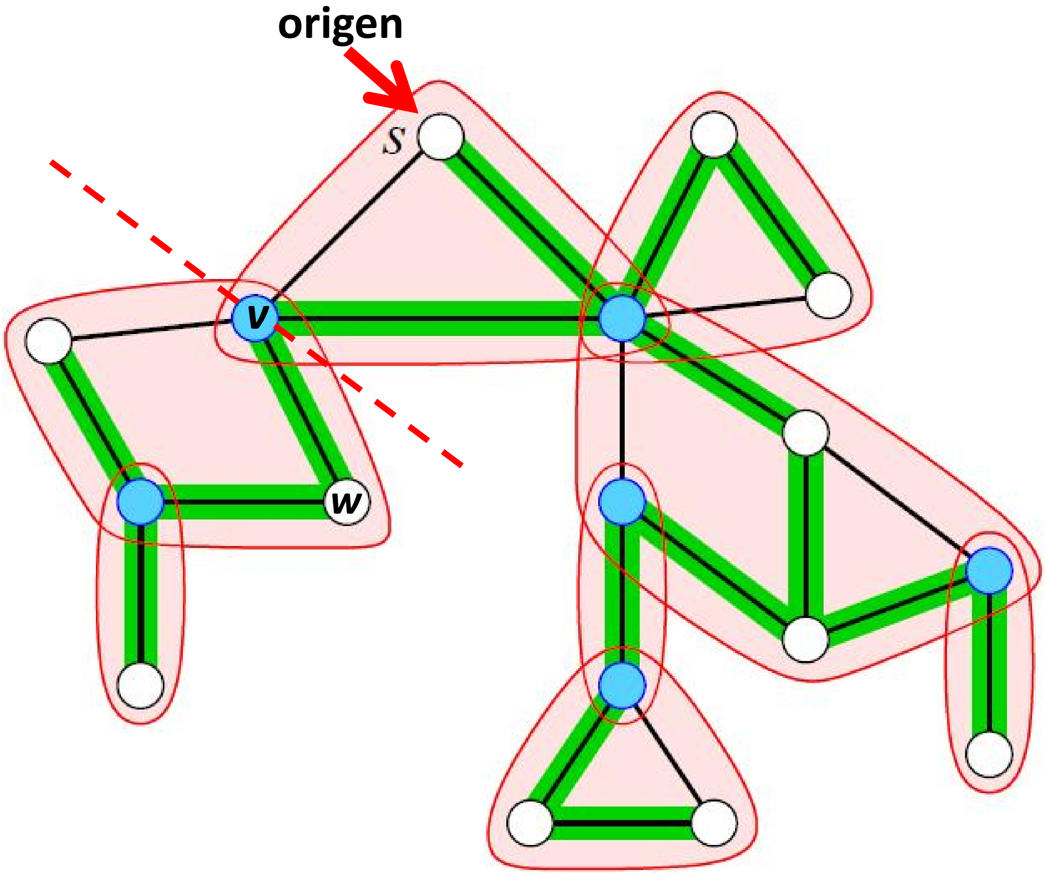
Las **componentes biconexas** (incluyendo **aristas puente**) se circulan en **rojo**

Los vértices **azules** son los **puntos de articulación**

El árbol de cubrimiento (DFS) se sombrea en **verde**

Análisis para cuando el origen NO es un punto de articulación

Sugerencia: aplicar DFS sobre el Grafo para resolver el problema



IDEA INTUITIVA

Los $v \neq s$ que son **puntos de articulación** tienen, al menos, un hijo w, tal que; **NO existe una *arista de retroceso* desde w, o desde cualquiera de sus descendientes, hacia un ancestro de v**

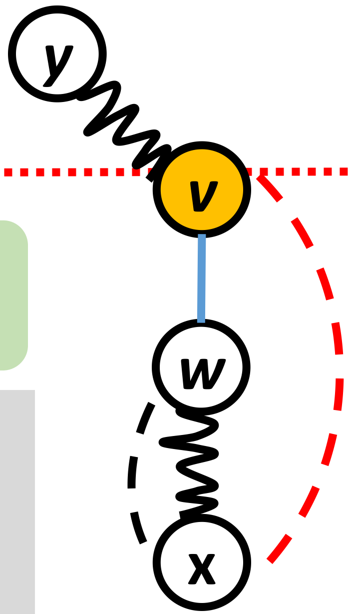
Lema 2

Sea G=(V, E) conexo y no dirigido. Un vértice v que no sea raíz del árbol abarcador en profundidad G_π es punto de articulación

$\Leftrightarrow$

v tiene, al menos, un hijo w tal que no existe una arista de retroceso desde w, o desde cualquiera de sus descendientes, hacia un ancestro de v

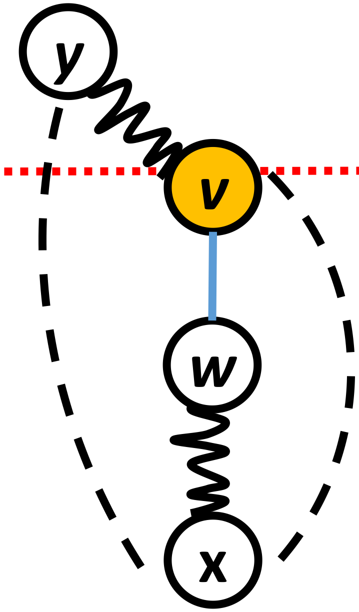
EJEMPLO 1



v **ES** punto de articulación

NOTE que, sin embargo, **<*v*, *w*>** no es **arista puente,** pues existe la **arista de retroceso** -----

EJEMPLO 2



v **NO ES** punto de articulación

Basta con que exista un hijo w que cumpla la proposición para que v sea punto de articulación (pueden haber hijos de v que no la cumplan)

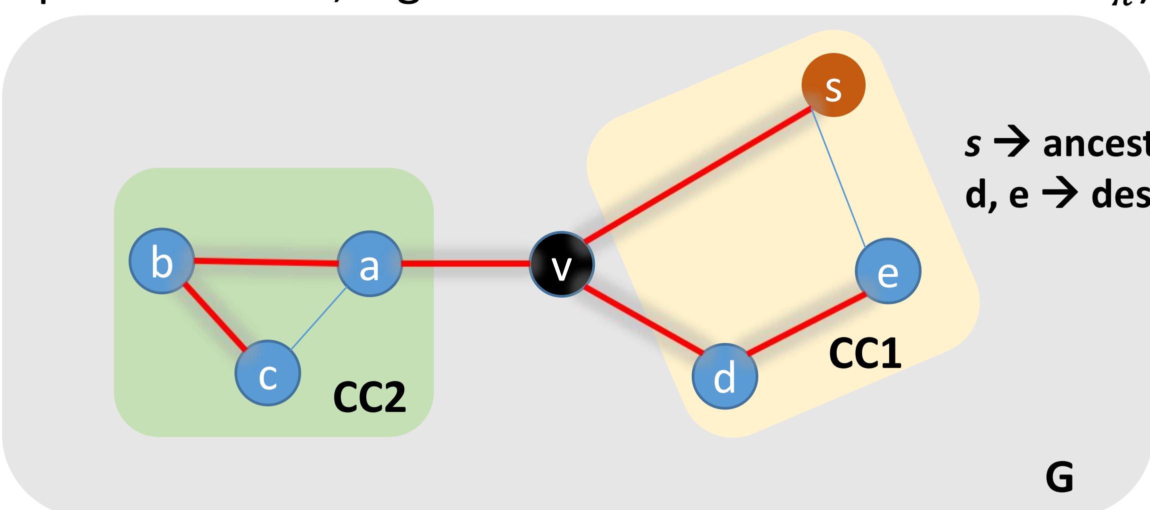
Lema 2

Sea G=(V, E) conexo y no dirigido. Un vértice v que no sea raíz del árbol abarcador en profundidad G_π es punto de articulación $\Longleftrightarrow$ **tiene, al menos, un hijo w tal que no existe una arista de retroceso desde w, o desde cualquiera de sus descendientes, hacia un ancestro de v**

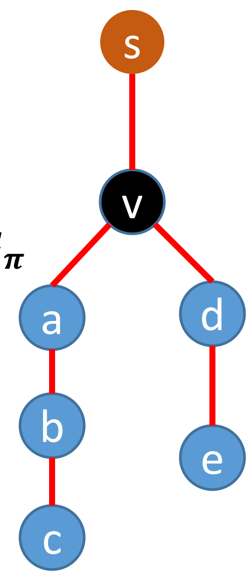
DEMOSTRACIÓN

($\Rightarrow$) Como v es punto de articulación, entonces, al eliminar v y sus aristas adyacentes de G, dicho grafo (conexo) se divide en, al menos, dos componentes conexas

En una de estas componentes (**CC1**), estarán los ancestros de v en G_π (y posiblemente, algunos de sus descendientes en G_π)

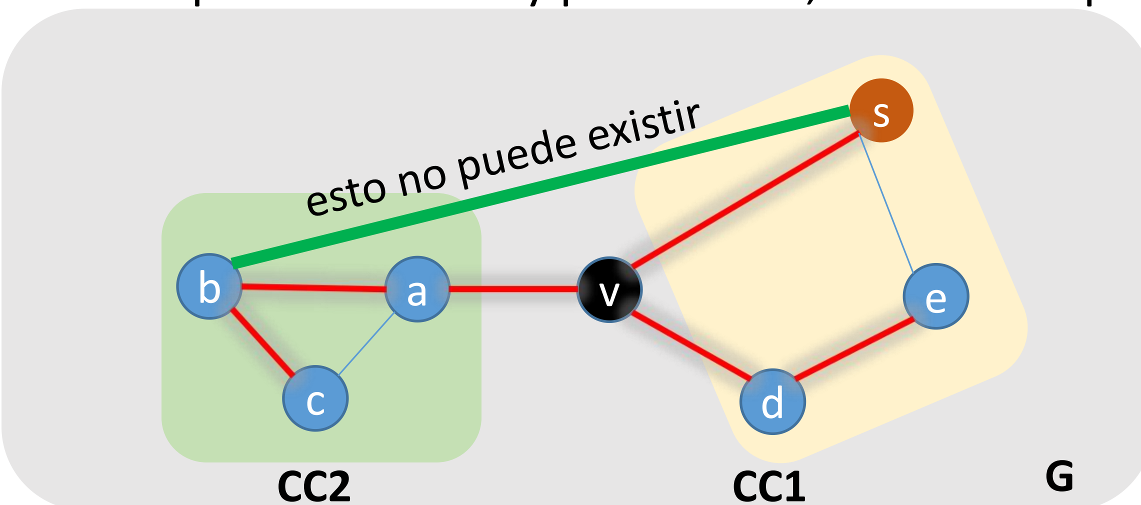


s → ancestro de *v* en G_π
d, e → descendientes de v en G_π



Lema 2 ... continuación de la demostración ($\Rightarrow$)

Toda **arista de retroceso** en la otra componente conexa (CC2) tiene que conectar, necesariamente, a descendientes de v entre si, o con el propio v, pues si no fuera así, existiría una forma (dicha arista de retroceso) de llegar de un vértice de una componente a otra y por tanto v, no sería un punto de articulación



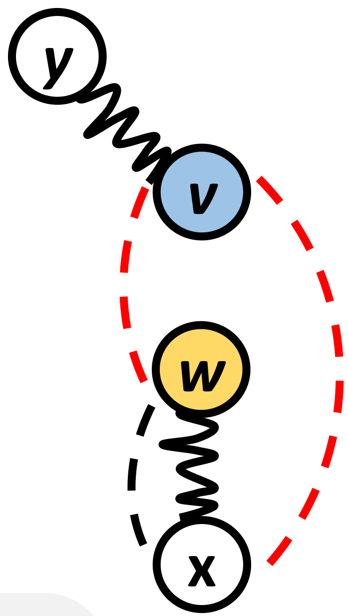
Note además, que **no pueden haber ancestros de v en la otra componente**

Por otra parte, tiene que existir $< v, w >$ con $w \in$CC2 pues en dicha componente, al menos, <u>tiene que existir un vértice adyacente a v, sea w</u>

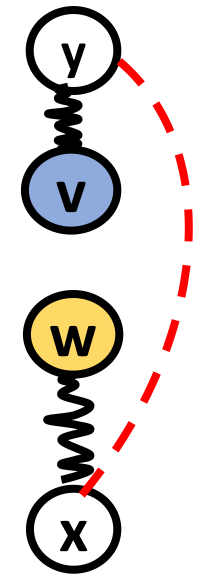
Lema 2 ... continuación de la demostración

DEMOSTRACIÓN(⇐)

todo camino desde w, o desde un descendiente de w, hacia un ancestro de v tiene que pasar por v **necesariamente**



Si existiese un camino, que no pasase por v, entonces habría una arista de retroceso con un ancestro de v, llegando a una contradicción con la hipótesis

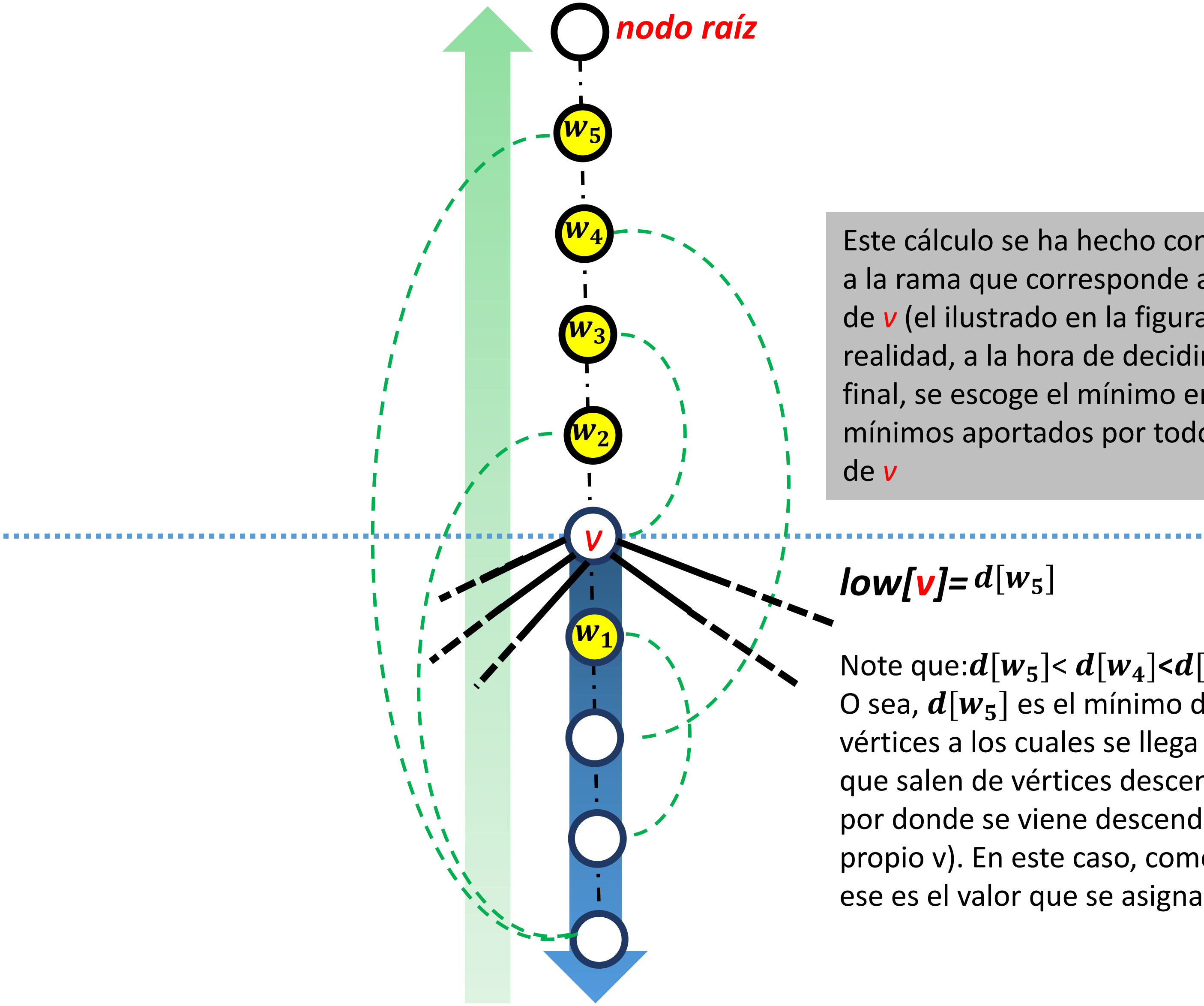


Por tanto, al eliminar v de **G**, se crean, al menos, dos componentes conexas: una con los descendientes de w y otra con los ancestros de v (y probablemente otros vértices).

Una aplicación del DFS: Algoritmo de detección de puntos de articulación

IDEA GENERAL:

- Realizar una **búsqueda en profundidad (DFS)** del grafo e ir numerando los vértices a medida que se van visitando.
- *array* ***d[v]***: almacena dicho número, $1 \leq v \leq |V|$**.**
  d[v]*: (discover time)* momento de **descubrimiento de *v***.
- La raíz se numera con 1
- La numeración que se establece a partir del *array* ***d[]*** permite **ordenar los vértices según el recorrido en preorden del árbol abarcador** que se obtiene tras el DFS
- Se establece también el *array* ***low[v]*** tal que: ***low[v]*** *= min* **(*d[v]*, *d[w]*),**
  $1 \leq v \leq |V|$, ***w:*** es cualquier vértice, alcanzable desde ***v***, al cual se llega ***bajando***, **cero o más** niveles, por cualquier rama que salga de ***v*** en el **árbol abarcador** hasta llegar a un descendiente ***u*** **del propio *v*** (puede suceder ***u*=*v***) y luego, ***subir* por UNA arista de retroceso** <***u***, ***w***>
- Al calcular el valor de ***low[v]*** se tienen en cuenta todos los adyacentes (***hijos***, determinados por ***aristas árbol*** y ***ancestros*** determinados por ***aristas de retroceso***) a ***v***

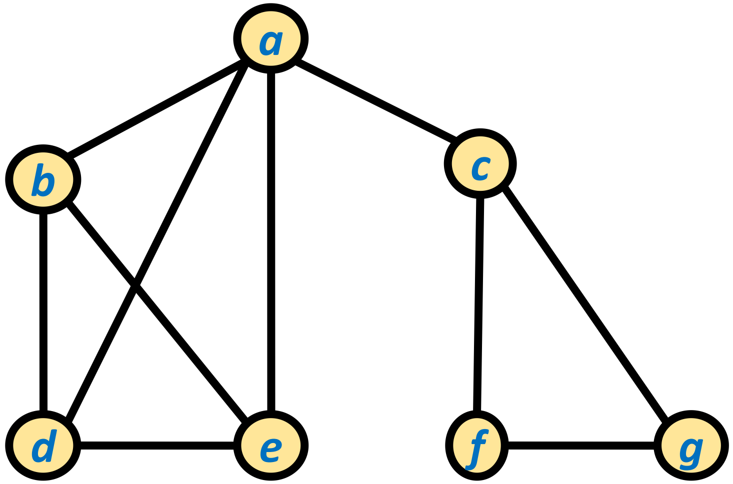


Este cálculo se ha hecho con respecto a la rama que corresponde a UN hijo de *v* (el ilustrado en la figura) pero en realidad, a la hora de decidir el low[*v*] final, se escoge el mínimo entre todos mínimos aportados por todos los hijos de *v*

low[v]= $d[w_5]$

Note que: $d[w_5] < d[w_4] < d[w_3] < d[w_2] < d[v] < d[w_1]$
O sea, $d[w_5]$ es el mínimo de los **d[]** de todos los vértices a los cuales se llega por aristas de retroceso que salen de vértices descendientes de v en la rama por donde se viene descendiendo (incluyendo al propio v). En este caso, como $d[w_5]$ es menor que **d[v],** ese es el valor que se asigna a **low[v]**

Interpretación intuitiva del *array* *low[v]*: da la idea de ***cuan alto se puede subir,*** partiendo de ***v*** y **tratando de alcanzar la raíz, por otra vía** (**una arista de retroceso**) que no sea una rama del árbol por la que se va descendiendo

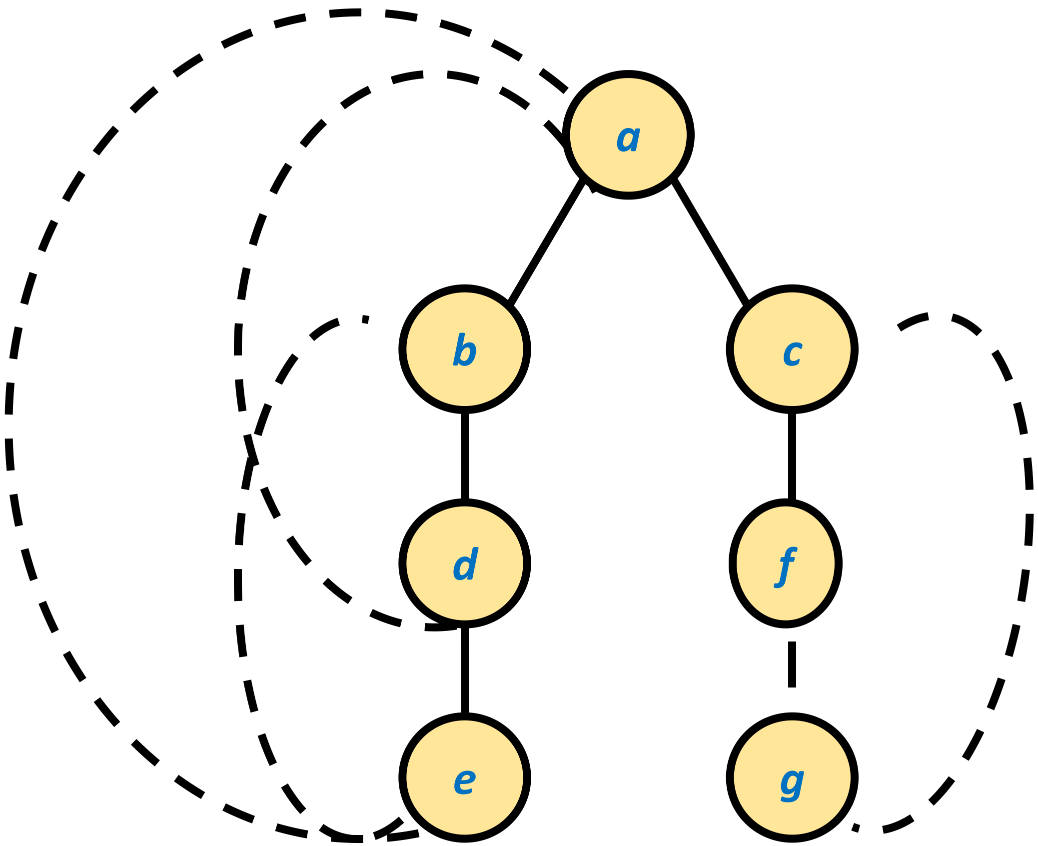


low[v] = min (d[v], d[w])

d(*a*)=1
low(*a*)=1

d(*c*)=5
low(*c*)=5

d(*b*)=2
low(*b*)=1



d(*f*)=6
low(*f*)=5

d(*d*)=3
low(*d*)=1

d(*g*)=7
low(*g*)=5

d(*e*)=4
low(*e*)=1

Teorema:

Un vértice ***v*****,** distinto de la raíz, es **punto de articulación** $\Leftrightarrow$ existe un vértice ***w***, hijo de ***v*** en el **árbol abarcador** correspondiente, tal que $low[w] \geq d[v]$

Demostración: Es una aplicación directa del **Lema 2**

Observaciones:

- En este caso, ***v*** desconecta a ***w*** y a sus descendientes del resto del grafo
- Es suficiente que suceda con, al menos, un solo hijo de ***v***, para afirmar que ***v*** es punto de articulación pues ***w*** y todos sus descendientes, se desconectan de una componente mayor a la cual pertenecían antes

DFS – Detección de puntos de articulación

```
DFS-VISIT-PA(G, u)
   u ← visited
   time = time + 1
   d[u] = time
   low[u]= d[u]
   for each v ∈ Adj[u]
     do if v not visited
         π[v] ← u
         DFS-VISIT-PA(G, v)
         low[u]=min(low[u],low[v])
         if low[v]≥d[u]
             print u is art. point
       else if π[u] ≠ v
           low[u]=min(low[u],d[v])
   return
```

caso base 1: time = time + 1; d[u] = time; low[u]= d[u]

caso 2: DFS-VISIT-PA(G, v); low[u]=min(low[u],low[v]); if low[v]≥d[u] print *u is art. point*

// *u* es padre de *v* y <*u*, *v*> es una *arista árbol*

caso 3: else if $\pi[u] \neq v$; low[u]=min(low[u],d[v])

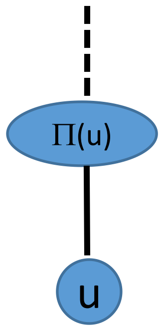
// *v* ya visitado, *u* NO ES el padre de *v*, o sea

// <*u*, *v*> es una *arista de retroceso*

NOTA:
G es un **grafo conexo**

LA RAIZ hay que verla como un CASO ESPECIAL: o sea, hay que verificar (fuera de **DFS-VISIT-PA**) si la raíz es **punto de articulación** o no [según el **Lema 1**]. Notar que, bajo el criterio seguido en el algoritmo, siempre será un **punto de articulación** pues tiene el menor **d[]** posible

caso base 1: en el DFS se llega a un nodo ***u*** que está en un nivel donde ya no se puede *seguir bajando*, pues ***u*** no tiene adyacentes (es una hoja), por tanto, a partir de él, se empiezan a resolver, por *backtrack*, los llamados recursivos que quedaron pendientes

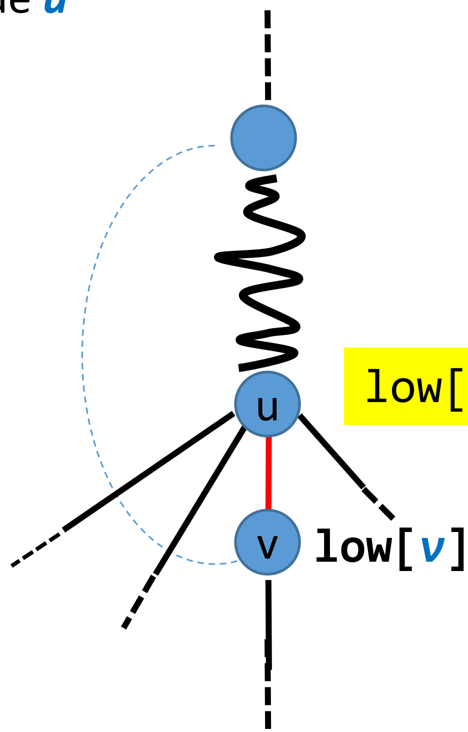



```
low[u]= d[u]
```

caso base 1: *u* no tiene adyacentes
(excepto su padre en el árbol abarcador, Π(*u*),
que ya fue visitado)

caso 2: En este caso, <*u*,*v*>es una <u>arista árbol</u> y ***v*** es uno de los hijos de ***u*** en el árbol abarcador. En este momento, el algoritmo determina si es necesario actualizar el `low[u]` teniendo en cuenta el `low[v]` calculado en el proceso recursivo

En general, el posible valor final de `low[u]` se determina cuando hayan sido resueltos los llamados recursivos de TODOS sus hijos, o sea, el valor final que adquiere `low[u]` es el `min(low[u],low[v'])`, donde **v'** es el hijo de ***u*** en el árbol tal que su `low[]`es el mínimo con respecto al de los restantes hijos de ***u***

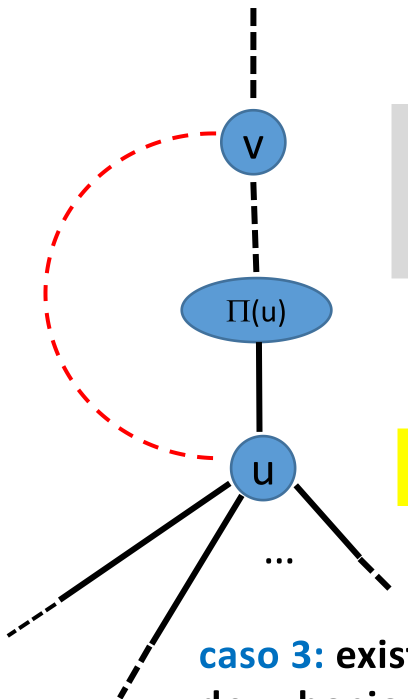


En este caso, en el cálculo del min, interviene el `low[v]` porque **v** es un hijo de ***u*** en el árbol, con su `low[]` ya determinado y por tanto, puede ser posible que desde él, se pueda subir a un nivel que esté por encima del nivel de ***u***

`low[u]=min(low[u],low[v])`

En este momento, el `low[v]`ya está determinado, pues ya se acabó el análisis del vértice ***v***

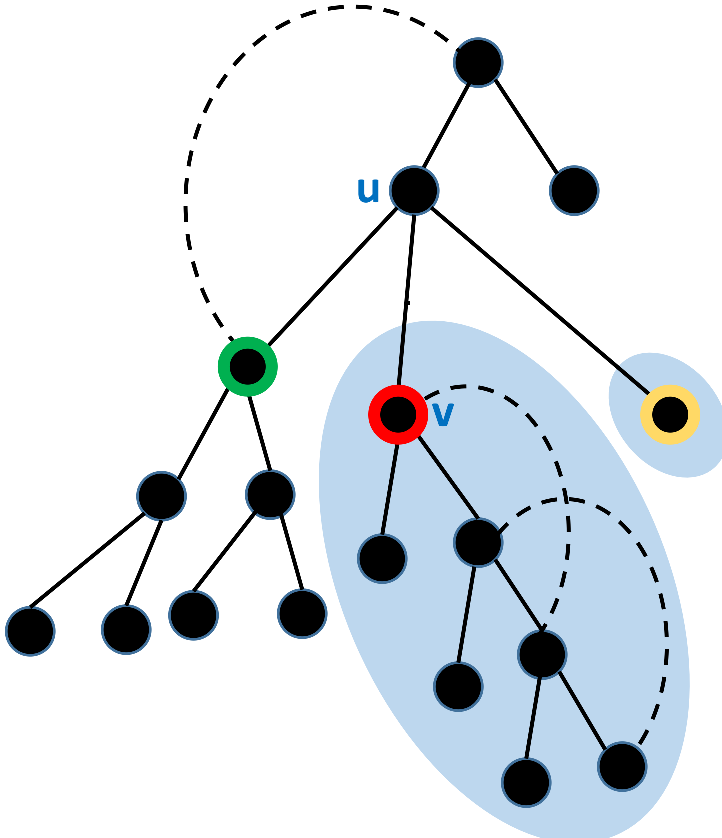
caso 3: la arista <*u*,*v*> NO ESTA en el árbol abarcador, o sea, es una ***arista de retroceso***. Al alcanzarse en el ***dfs*** una arista de este tipo, entonces, puntualmente, la actualización que ese ***v*** implicaría en el `low[u]` estaría determinada por el `min(low[u],d[v])` y ese valor sería el que se le asigna, en ese momento, a `low[u]`. El mismo puede ser cambiado, nuevamente, en el análisis de otros vértices adyacentes a ***u***



En este caso, en el mínimo interviene `d[v]` y no `low[v]` porque ***v*** no es un hijo de ***u***

`low[u]=min(low[u],d[v])`

caso 3: existe una *arista de retroceso* que sale de *u* hacia un vértice *v*, adyacente a él, ya visitado y que *no es su padre*



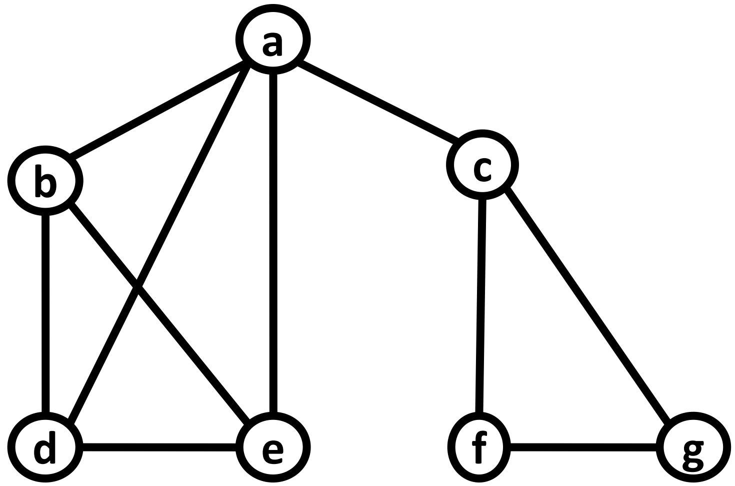
low[*v*]≥d[*u*]

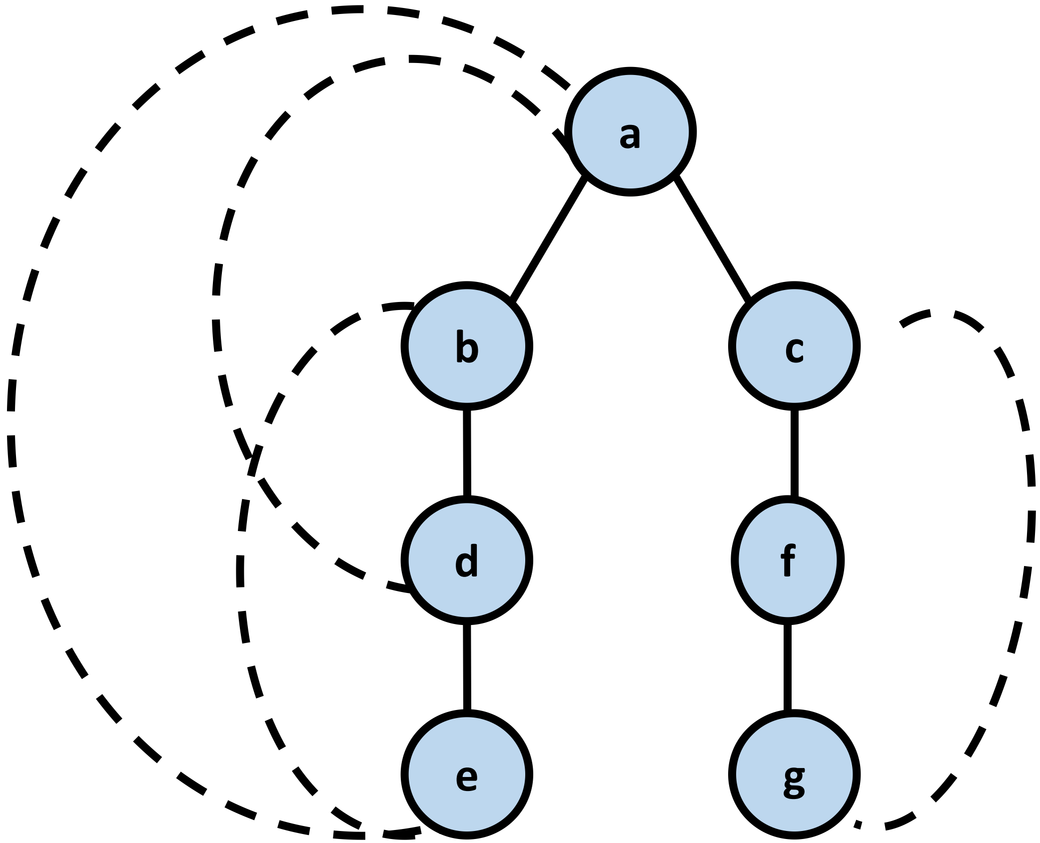
u es punto de articulación , pues al quitar la arista (***u***, ***v***) se desconectan ***v*** y todos sus descendientes

Los tres hijos de *u* en el árbol:

Tras el análisis del vértice también se determina que ***u*** es punto de articulación

DFS – Detección de puntos de articulación [Ejemplo]





d[a]=1
low[a]=1

d[b]=2
low[b]=1

d[c]=5
low[c]=5

d[d]=3
low[d]=1

d[f]=6
low[f]=5

d[e]=4
low[e]=1

d[g]=7
low[g]=5

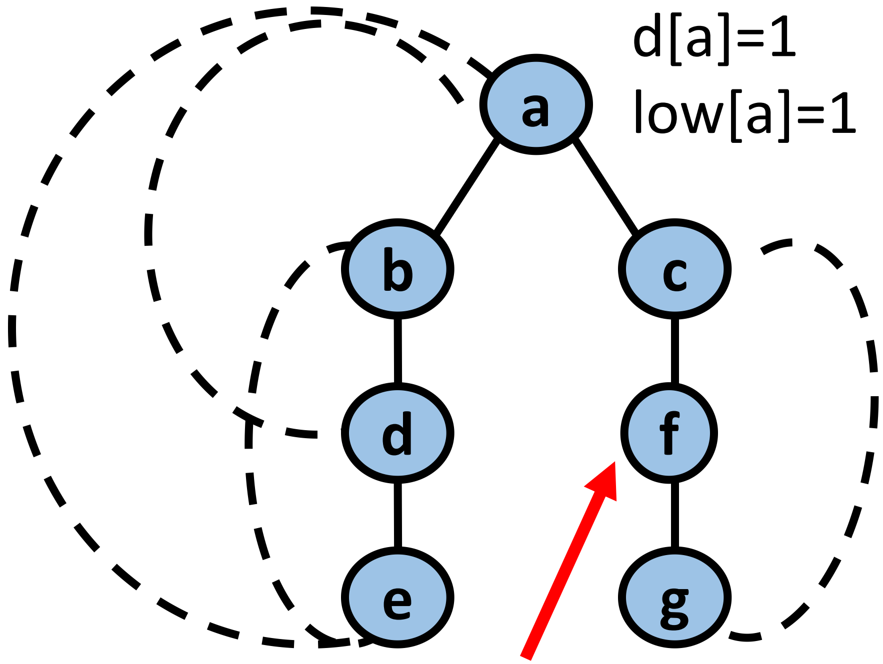
DFS – Detección de puntos de articulación [Ejemplo]

- a es un punto de articulación porque tiene dos hijos.
- c es un punto de articulación porque tiene un hijo f para el cual se cumple que, tope(f) ≥ profundidad(c)
- Los otros vértices no son puntos de articulación

d[a]=1
low[a]=1

d[b]=2
low[b]=1

d[c]=5
low[c]=5



d[d]=3
low[d]=1

d[f]=6
low[f]=5

d[e]=4
low[e]=1

d[g]=7
low[g]=5

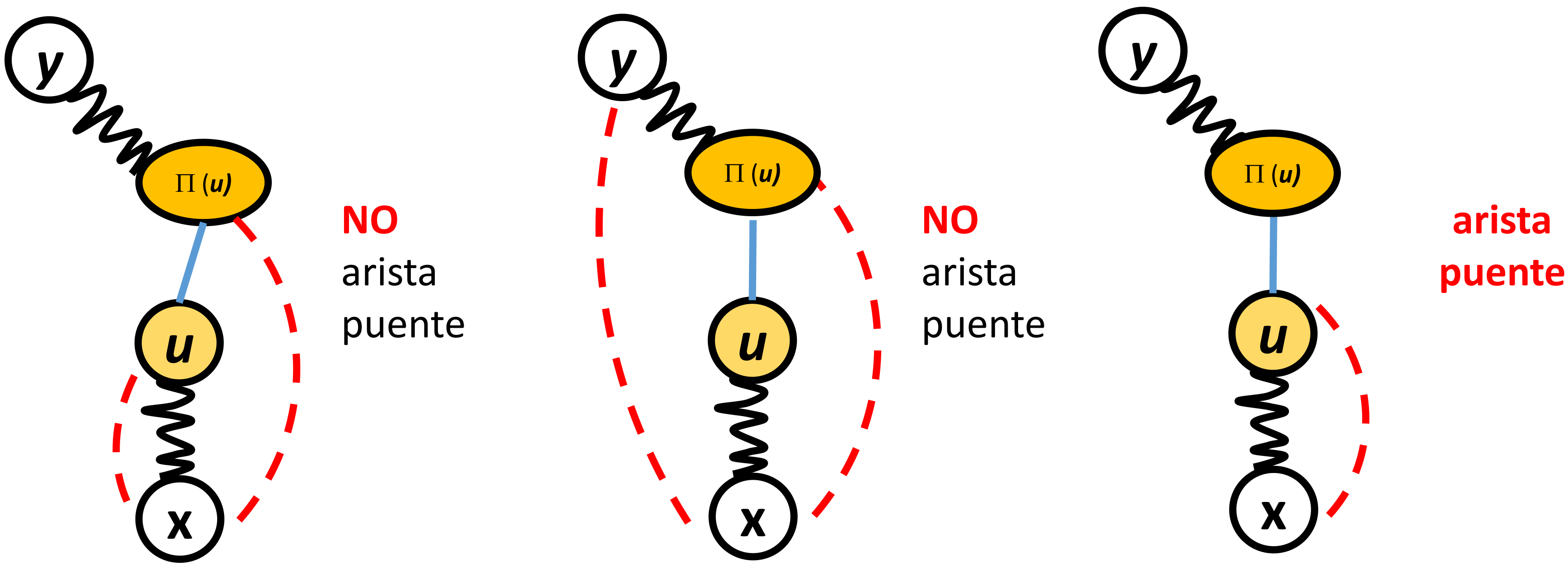
Note que desde f, hasta donde más se puede subir, es hasta c, por eso, c es punto de articulación

DFS – Algoritmo de detección de aristas puente

IDEA INTUITIVA:

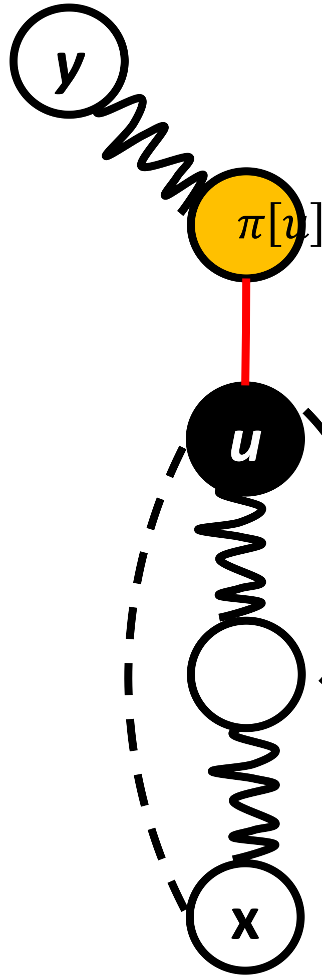
Sea ***u*** un nodo y sea ***Π(u)*** su padre tras aplicar DFS al grafo G, entonces:

- Si, cuando se hizo el análisis del nodo ***u*** en el DFS, no aparecieron aristas de retroceso desde dicho nodo, o desde alguno de sus descendientes, hacia algún ancestro de ***u*** (incluído su padre ***Π(u)***), entonces al eliminar <***Π(u)***, u> de G se desconectan ***u*** y sus descendientes de ***Π(u)***, por tanto, la arista <***Π(u)***, ***u***> es una arista puente



DFS – Algoritmo de detección de aristas puente

```
DFS-VISIT(G, u)
  u ← visited
  time = time + 1
  d[u] = time
  low[u]=d[u]
  for each v ∈ Adj[u]
    do if v not visited
         π[v] ← u
         DFS-VISIT(G, v)
         low[u]=min(low[u],low[v])
         if low[v]≥d[u]
             print u is art. point
    else if π[u] ≠ v
         low[u]=min(low[u],d[v])
  //u no es la raíz
  if π[u] ≠ null  ^ low[u]=d[u]
    print <π[u],u> is bridge
  return
```



$<\pi[u],u>$

low[u] = d[u]

$<\pi[u],u>$

arista puente:

Todos los hijos de ***u***, llegan, a lo sumo, a ***u***

NOTA: Para determinar si $<\pi[u], u>$ es ***arista puente***, **es necesario que antes hayan sido analizados todos los adyacentes de u**, por tal motivo es que la prueba se realiza cuando se finaliza el análisis de u (se pinta de negro)

Puede alguna arista que no sea ***de árbol***, ser una ***arista puente*** ?

NO, pues en tal caso, dicha arista sería una ***arista de retroceso*** y la existencia de ella, implica la existencia de un ciclo al cual ella pertenece y por tanto, como está dentro de un ciclo, no será ***puente***, pues al eliminarla, los restantes vértices no se desconectan