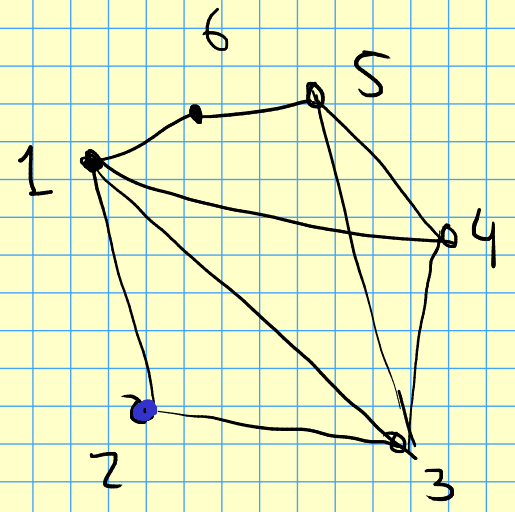
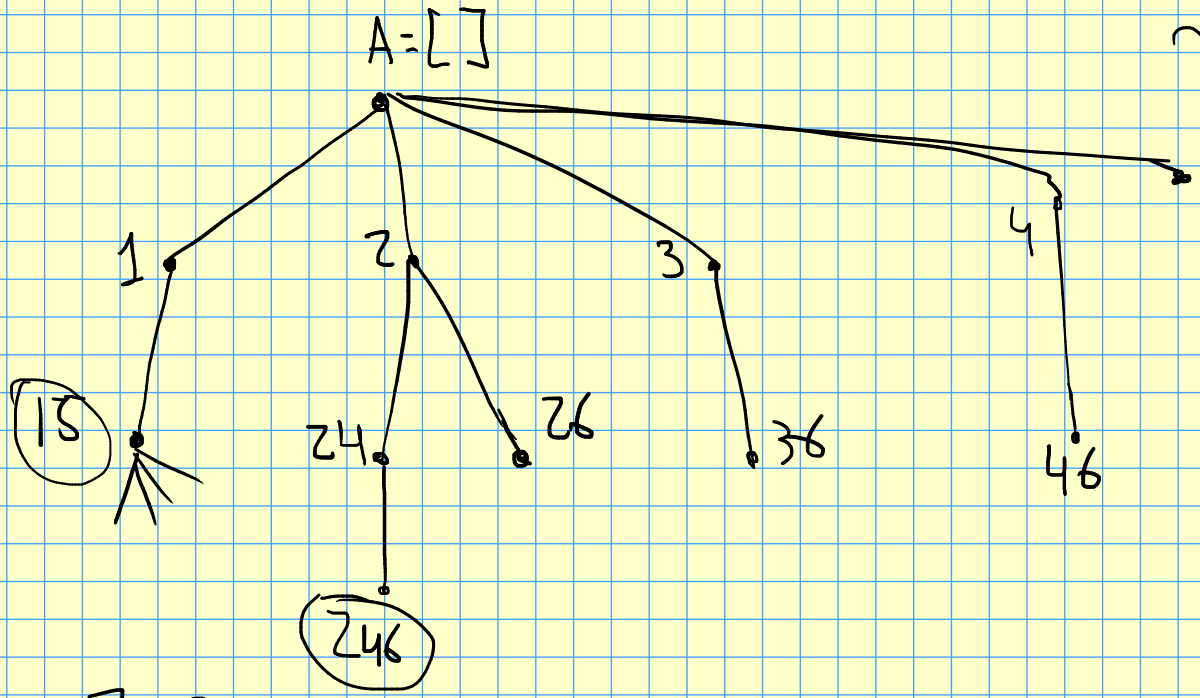
Examen 3

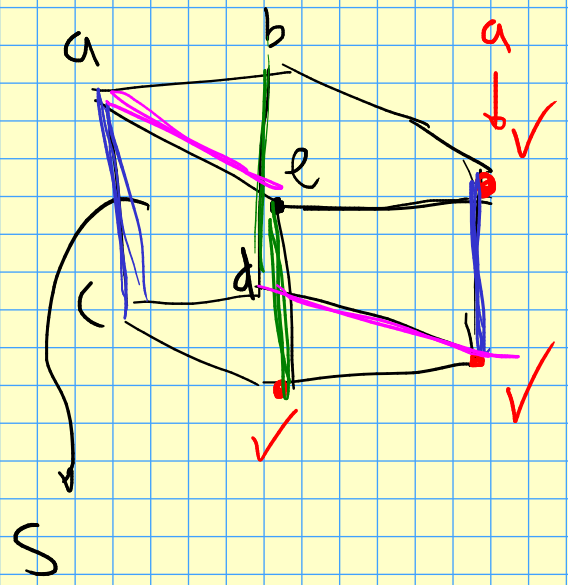


A = [1 ...

indep.
  iniciar A
  candidatos = a partir del último de A, los que no sean hijos de A.
  Para j en candidatos:
    A.app(j)
    ↝ indep(A)
    A.remove(j).



2, 3



$A = [a, b, c, d, e, \ldots]$

S

$25 \cdot 24 \cdot 23 \cdot$

$30 \cdot 24 \cdot 28 \cdot 27 \cdot 26 \; \underline{1} \; \underline{1} \; \underline{1} \cdot$

distinto a anteriores.

$a = 7, \; b = 2.$ Existe $c < 1$ tal que

$$7 \cdot 2^n = 7 \cdot 4^{n/2} < c \cdot 4^n \qquad c = 1/2$$

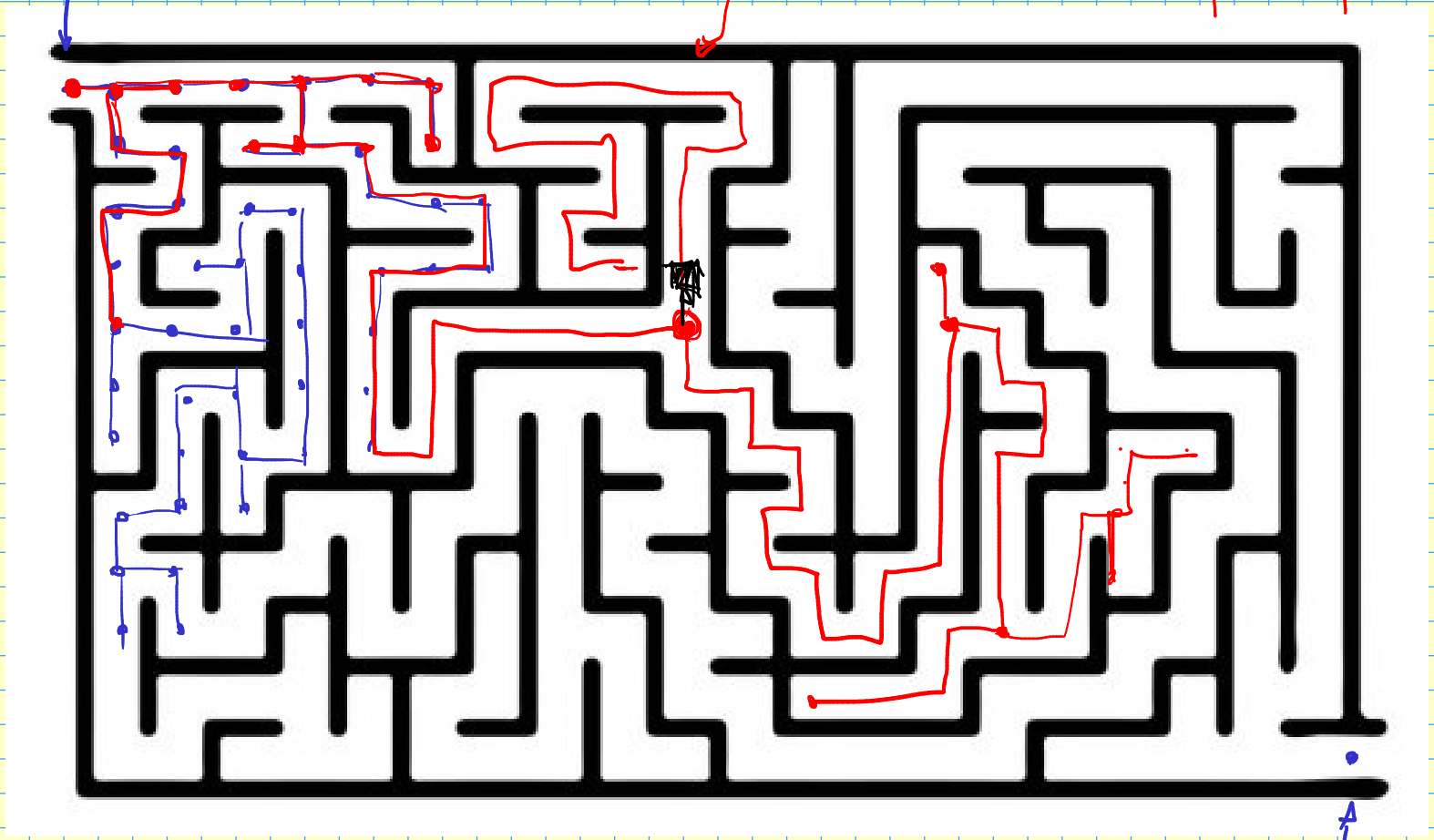
$$7 \cdot 2^n < \frac{1}{2} \cdot 4^n$$

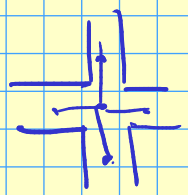
para n sul. grande.

para $n \geq 4$ se cumple.

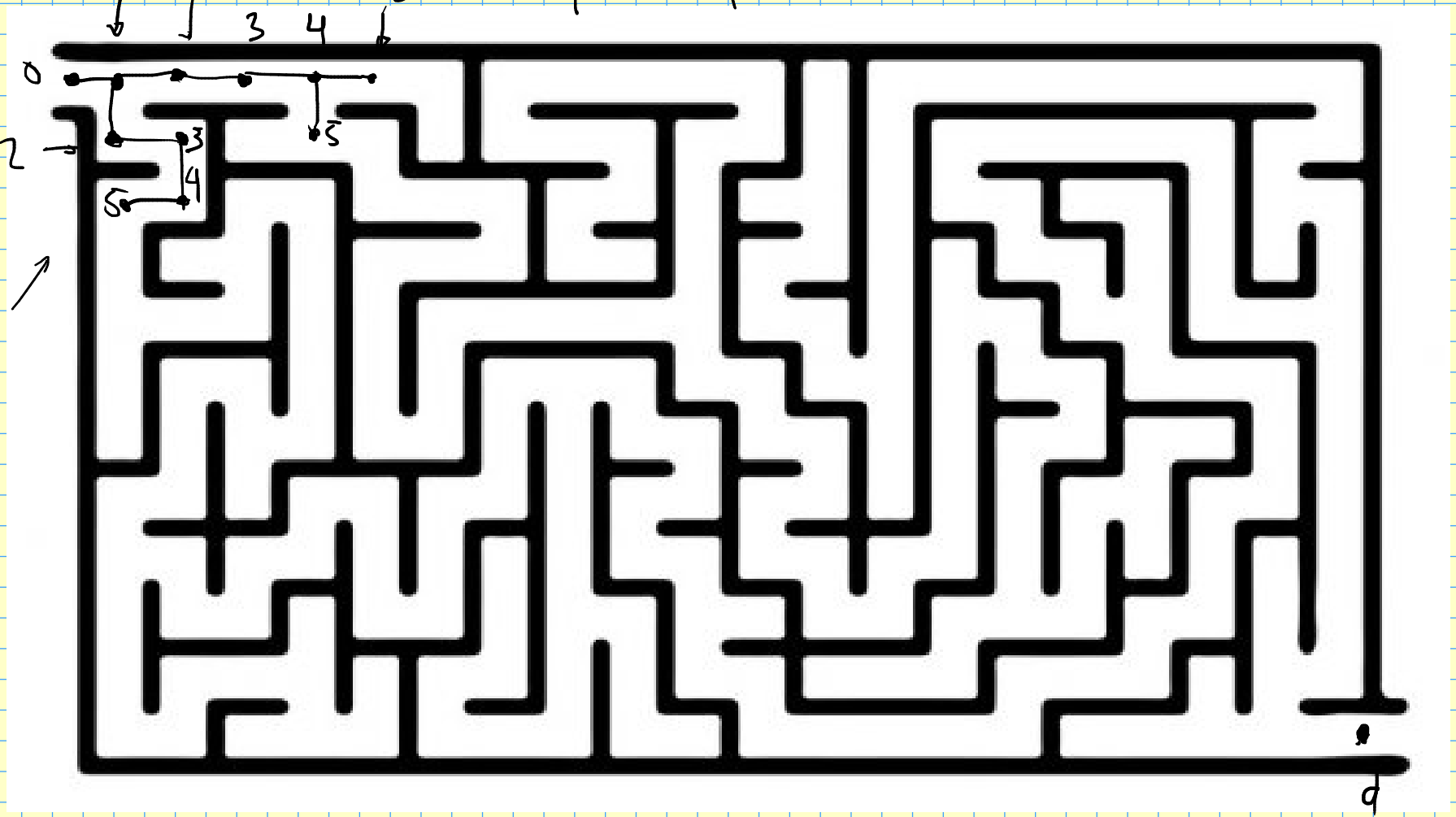
Búsqueda por profundidad

backtrack ~ búsqueda por profundidad



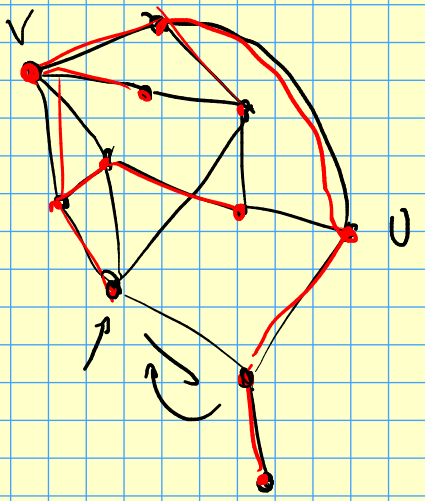


Búsqueda por anchura.



Pensemos que tenemos una gráfica G dada por listas de adyacencia.

Una búsqueda en G es un procedimiento que con esta información permite visitar todos los vértices y todas las aristas, pero sin redundancias.

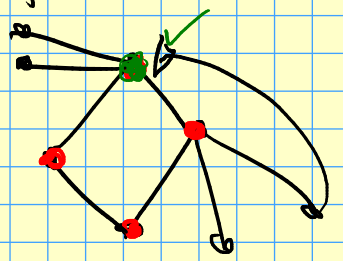


Una forma de garantizar que no hay redundancias es llevar una cuenta de vértices no visitados.

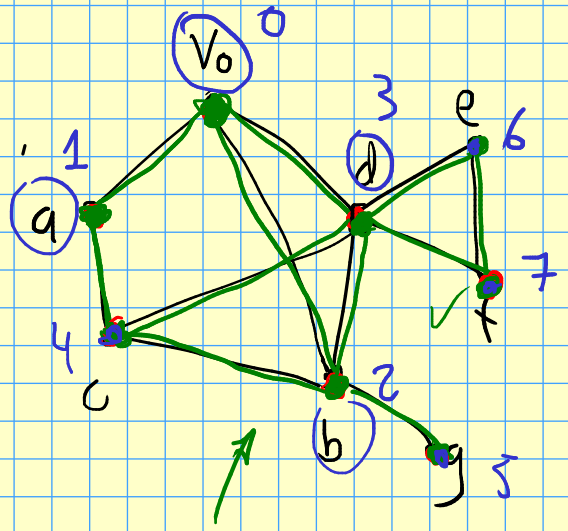
Para pasar por todos los vértices y aristas, cada que lleguemos a un vértice conviene inspeccionar todos sus vecinos.

De este modo, conviene también llevar una cuenta de vértices cuyos vecinos están en inspección. Así, mientras hacemos una búsqueda conviene pensar en que cada vértice está en uno de 3 posibles estados:

- Nunca visitado.
- Visitado, procesando.
- Procesado.



Una búsqueda por anchura hace una búsqueda en una gráfica de la siguiente manera:



v_0 = vértice inicial que queremos.

encontrados = $[v_0]$

procesados = $[]$

mientras encontrados $\neq \phi$:

  v = encontrados $[0]$.

  pre-procesar v.

  para w vecino de v:

    procesar la arista vw.

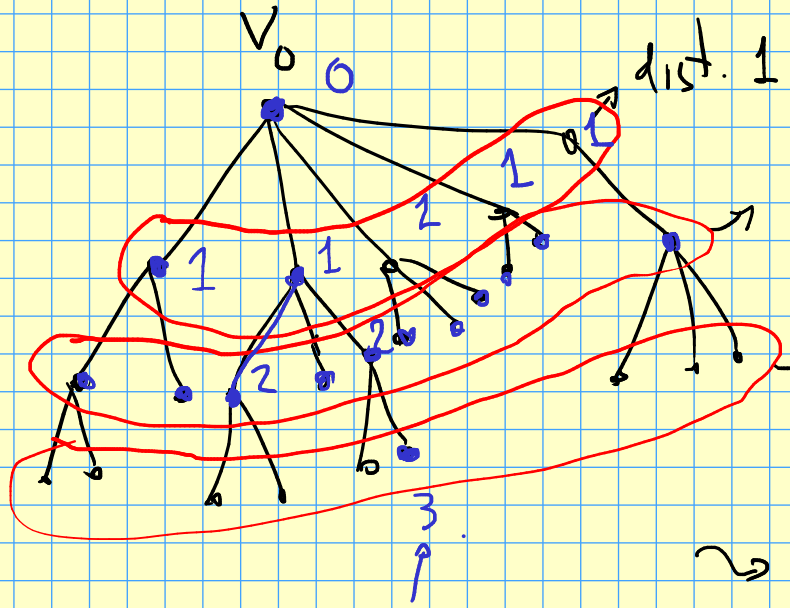
    si w no está encontrado ni procesado, lo agregamos a encontrados.

  pasamos v de encontrados a procesados.

E	P
v_0	ϕ
v_0, a, b, d	ϕ
a, b, d	v_0
a, b, d, c	v_0
b, d, c.	v_0, a.
b, d, c, g	v_0, a.
d, c, g, e, f.	

v_0, a, b, d, c, g, e, f.

Una búsqueda por anchura visita a los vecinos de acuerdo a su distancia a la raíz.

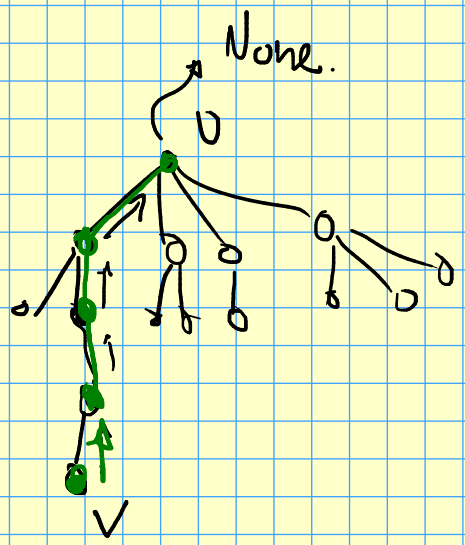


dist. 1 de v_0

dist 2 de v_0.

dist 3

$\rightsquigarrow G, U, V$.

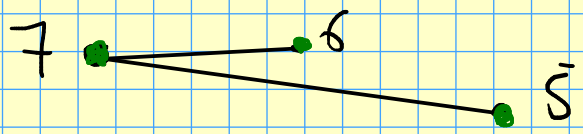


$d = 1$

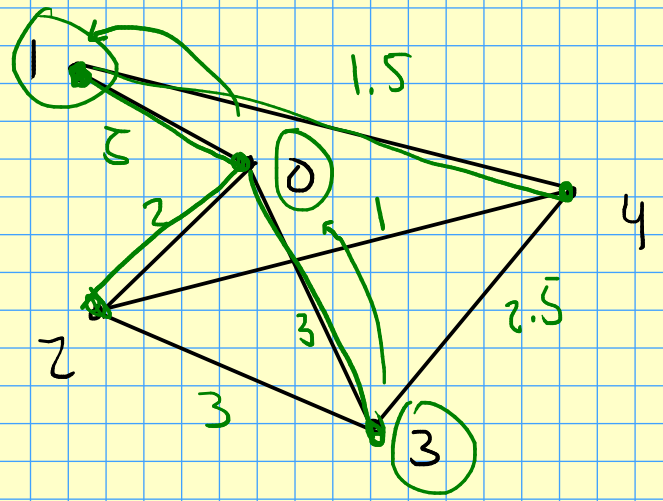
$d = 2$

$d = 3$

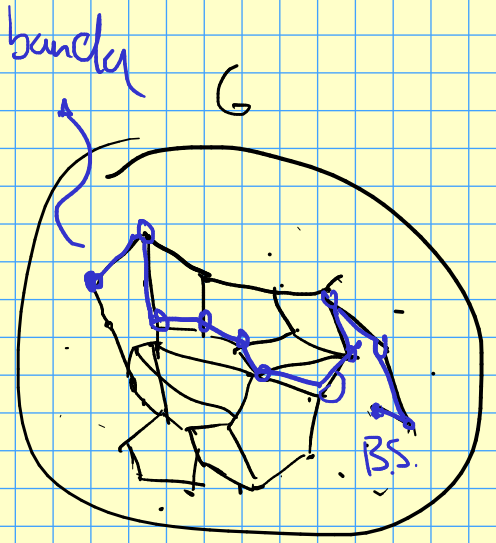
$\boxed{d = 4}$



7, 4, 1, 5



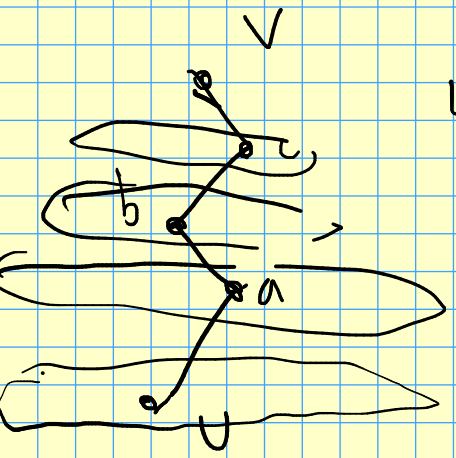
1,3



Cambia

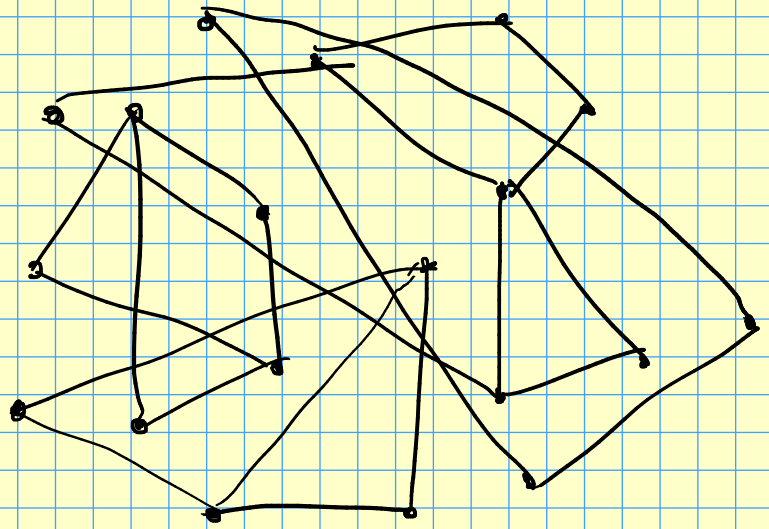
banda

- despacito



4

- bacilos
- shakira
- britney spears.

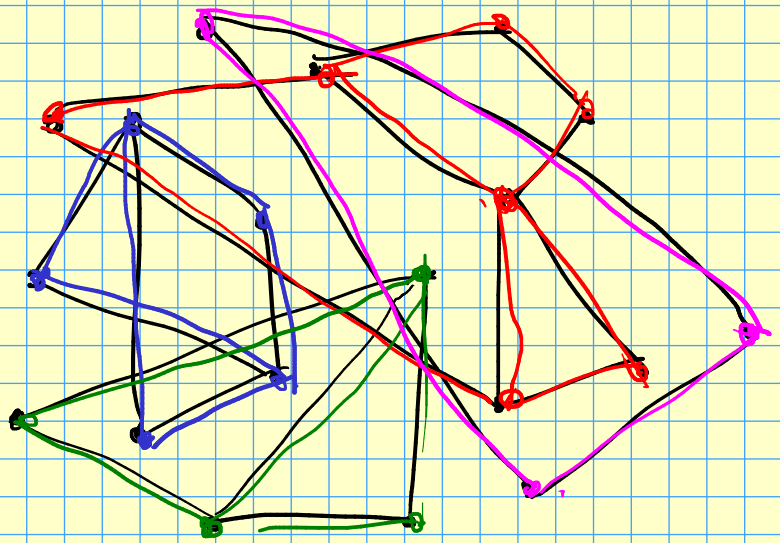


Preg.

¿Cuáles componentes conexos tiene?

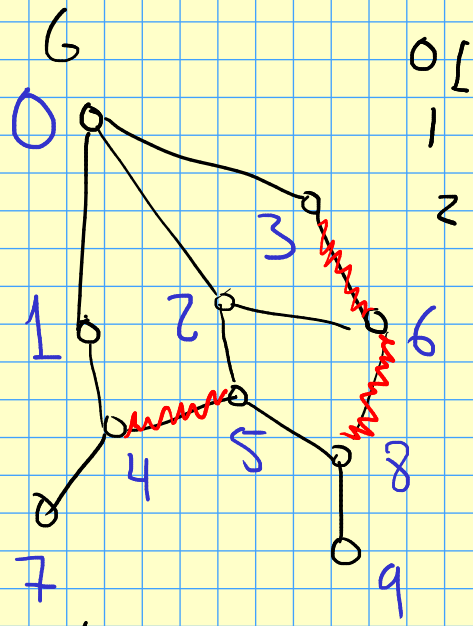
¿Cuántas " "?

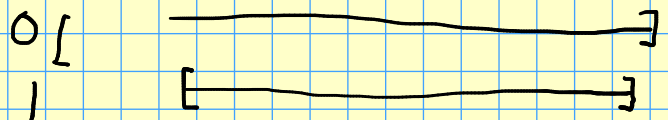
¿Cuál es la componente conexa más grande/chica/con más aristas/etc?



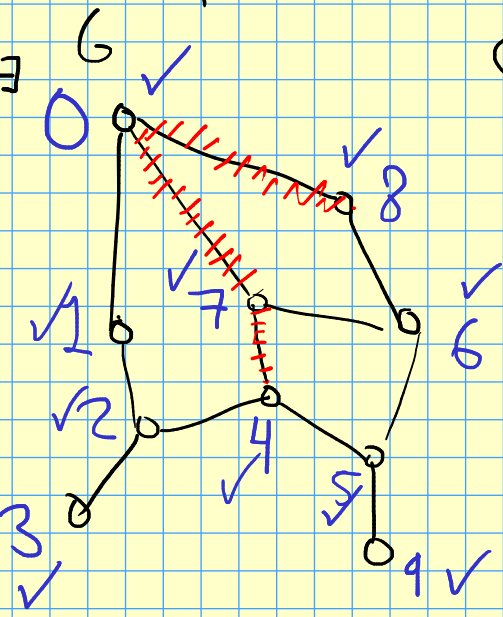
Búsqueda por profundidad.

Por anchura



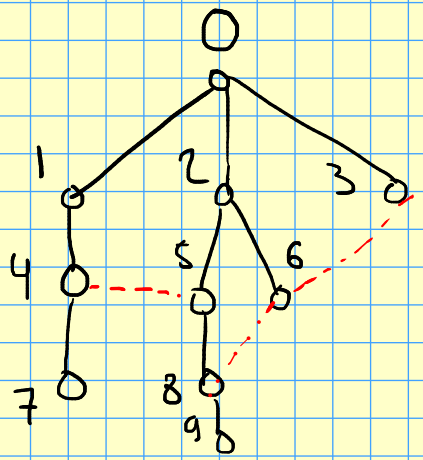


Por profundidad



Crucinómetro

Árbol por anchura

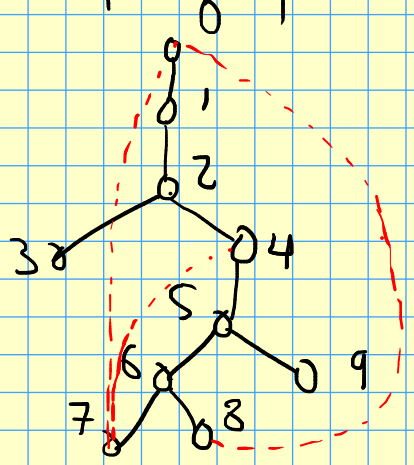


← nivel j.

La dist a 0 es j

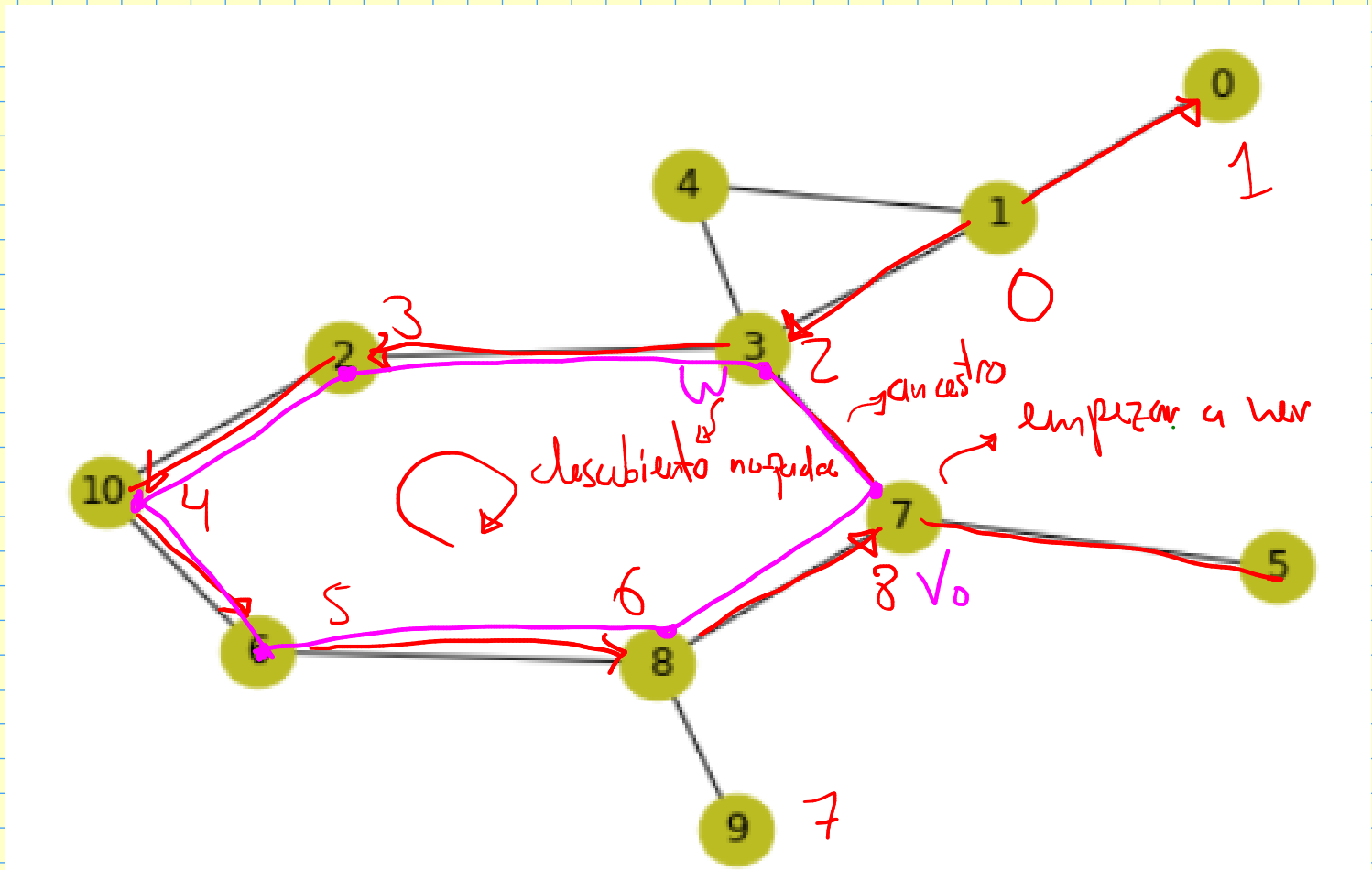
Árbol por profundidad.

Las aristas de G únicamente conectan vértices con ancestros.



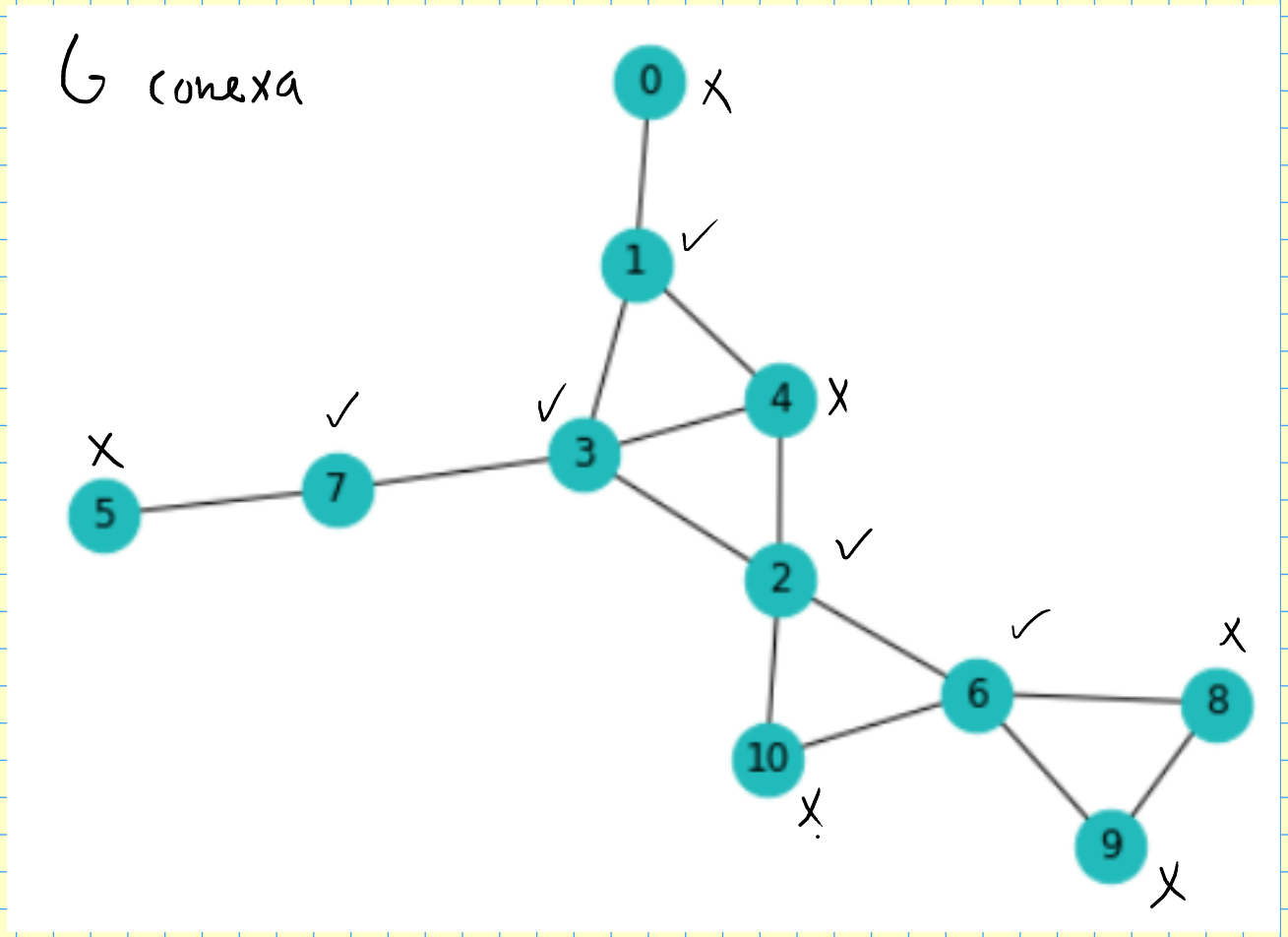
Se construyen poniendo quién descubrió a quién.

- Realmente son árboles.
- Son subgráficas de G.



Vértices de articulación.

n vértices
m aristas



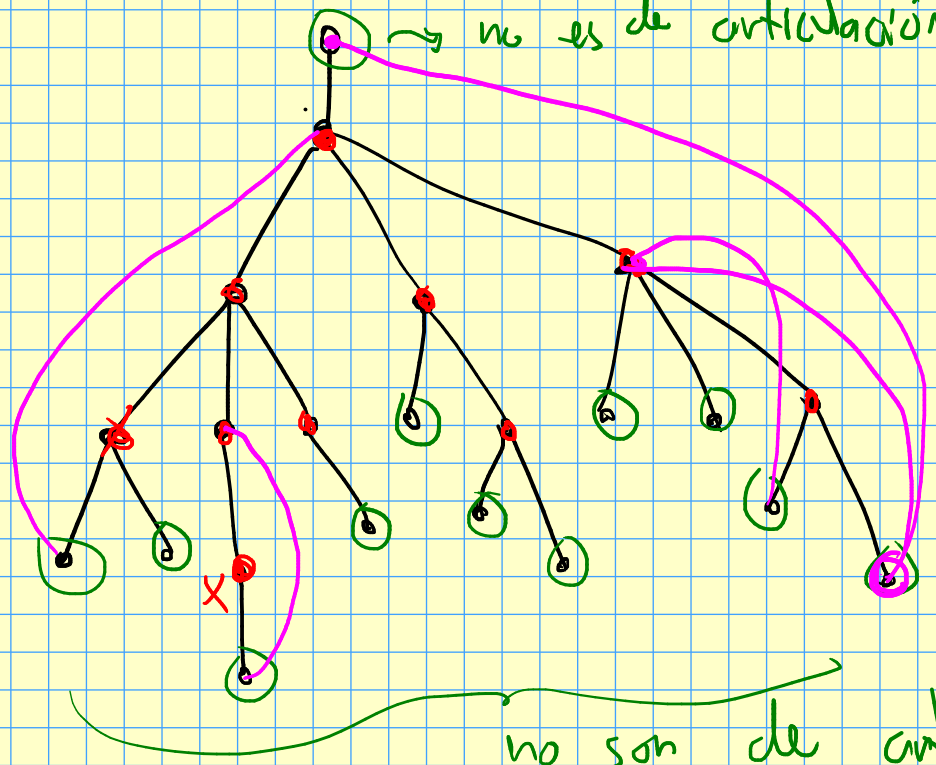
Para cada $v \in G$.

- quitar v de G.
- ver si lo que queda es conexo (con BFS, digamos)
- regresar v a G.

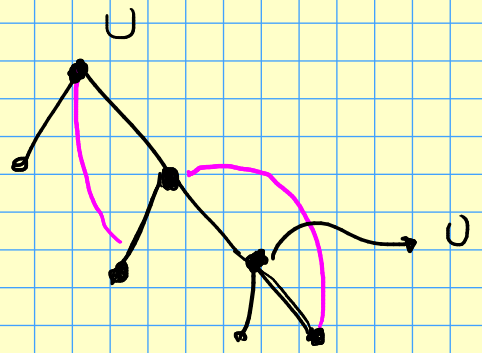
↑

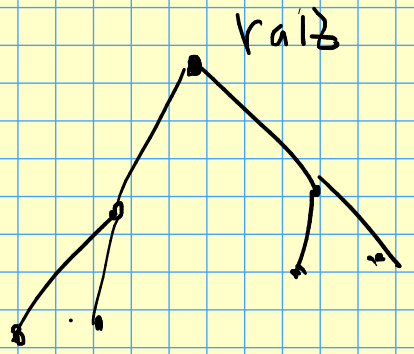
$O(n \cdot (m+n))$ tiempo.

Se pueden encontrar todos los vértices de articulación en tiempo $O(m+n)$ usando búsqueda por profundidad.



Para cada vértice de la gráfica iremos actualizando cuál es el ancestro más antiguo al que podemos llegar. yendo hacia abajo en las aristas de T o bien hacia arriba con aristas hacia ancestros.

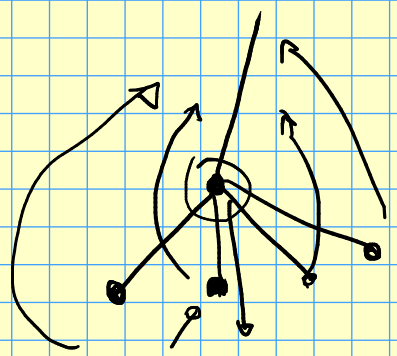




Para la raíz:

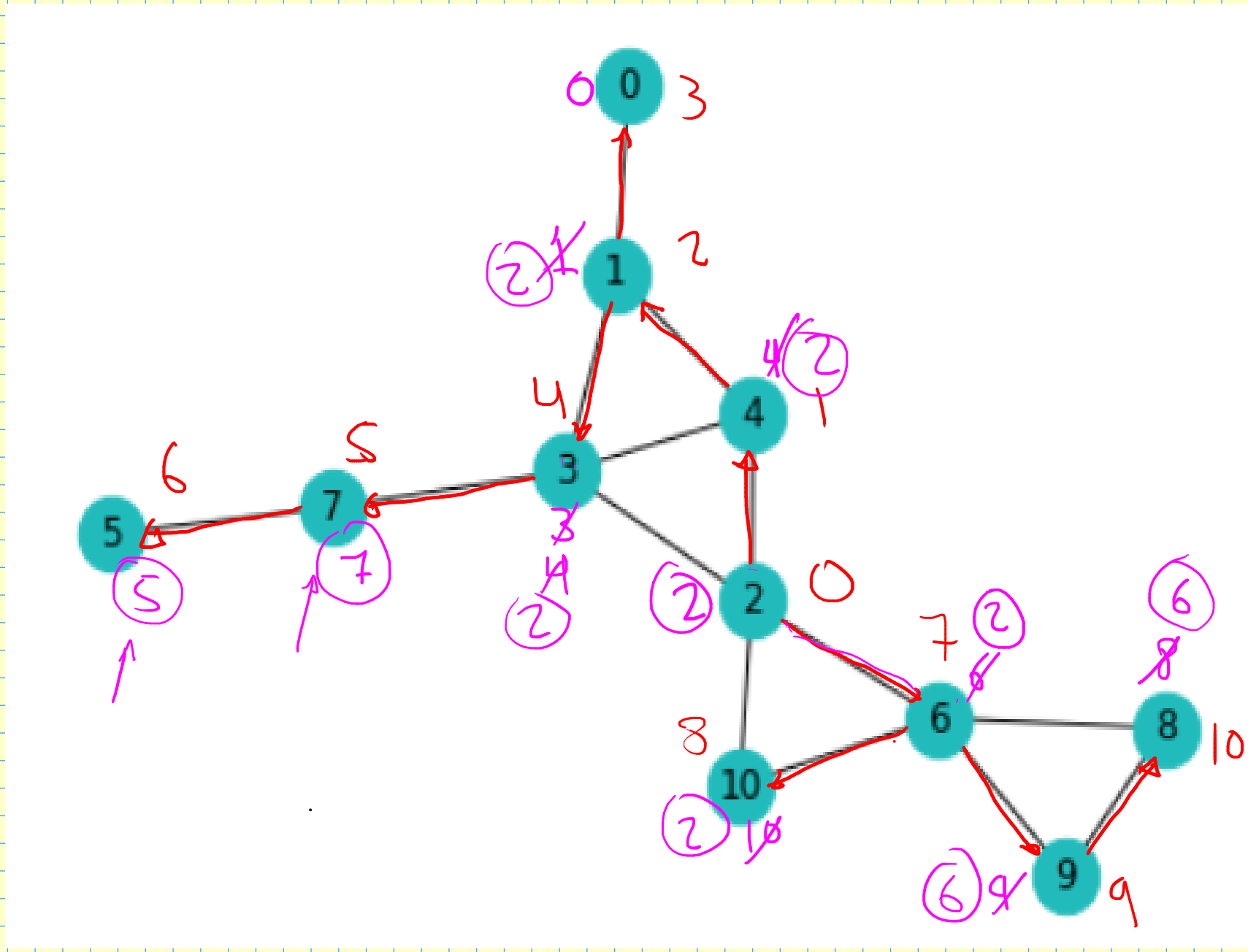
- Si salen al menos 2 aristas, forzosamente es de articulación
- Si salen ≤ 2 aristas, es hoja o no tiene vecinos, así que no es de articulación.

Para las hojas: Nunca son de articulación



Para los demás vértices v:

Si alguno de sus hijos cumple que el ancestro más antiguo es él mismo, o v, ent. v es de articulación.


0
1
2
3
4
5
6
7
8
9
10