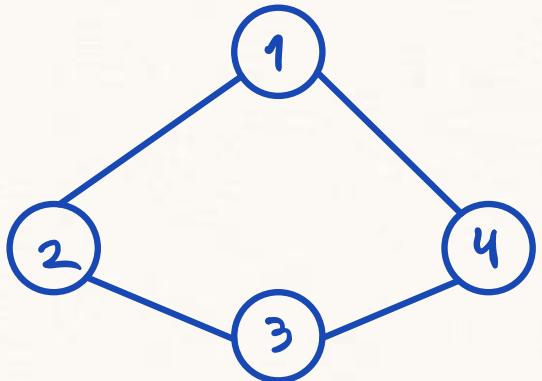


# Algoritmo de Blossom

1) Se trata de un algoritmo de EMPATEJAMIENTO de grafos. Se puede aplicar a cualquier tipo de grafo.

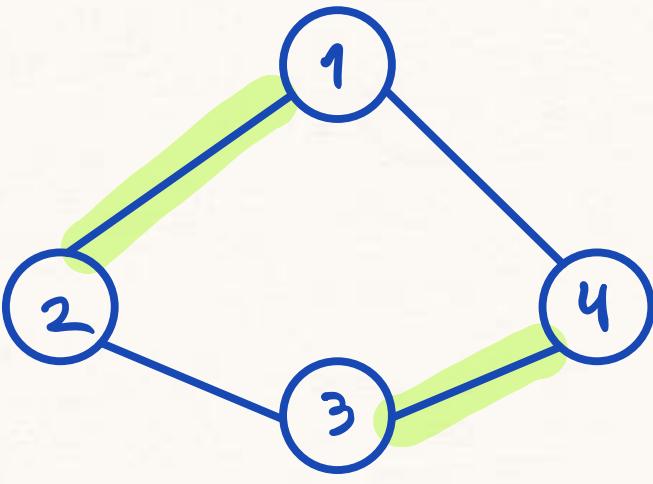


Un emparejamiento es básicamente un conjunto de PARES de vértices conectados, es decir, de ARISTAS.

- En este caso, un ejemplo podría ser:

$$\{(1,2), (3,4)\}$$

2) Gráficamente lo podríamos representar así:

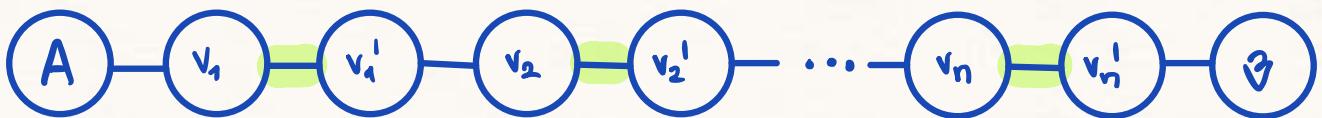


# Caminos de Aumento

(1) Son la clave principal del algoritmo, ya que estos nos permiten MEJORAR / AUMENTAR el emparejamiento que tenemos actualmente.

\* Un camino de aumento se caracteriza porque sus EXTREMOS son vértices NO emparejados y el resto SI están emparejados:

↳ los vértices  
del MEDIO



• Notar que los caminos de aumento siempre tienen un número PAR de vértices.

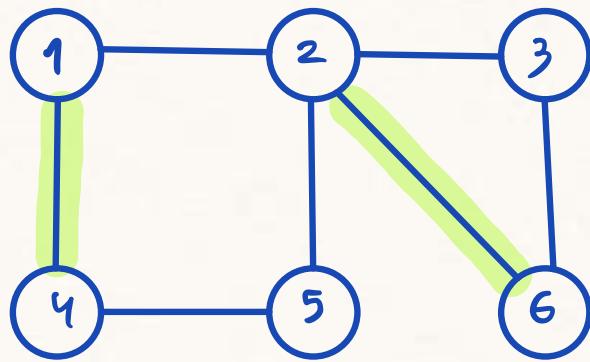
$$\begin{aligned} &\text{↳ } n \text{ gajetas internas (2 · n vértices)} \\ &\text{↳ } \text{los extremos (2 vértices)} \end{aligned} \quad \left. \begin{array}{l} \text{P PARI} \\ 2 + 2n = 2(n+1) \text{ vértices} \\ \text{en total} \end{array} \right\}$$

\* Se llaman caminos de AUMENTO porque nos permiten mejorar en UNA gajeta el emparejamiento de los vértices que conforman el camino

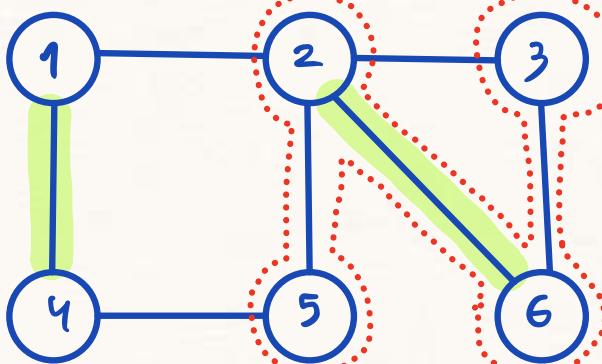
(1) Para este INVERTIR las ARISTAS: empareja  $\rightarrow$  no empareja:



• Veamos un ejemplo con un grafo:

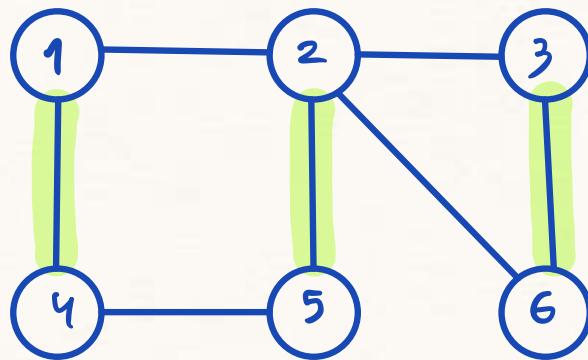


→ Claramente el emparejamiento actual no es el mejor posible. Podríamos reorganizar un poco las parejas y obtener uno perfecto. Un camino de aumento nos dice justamente.



- El camino  $(3, 6, 2, 5)$  es de aumento porque los EXTRÉMOS ( $3$  y  $6$ ) están NO emparejados y el resto del camino lo conforman nodos emparejados ( $2-5$ )

4) Podemos mejorar el emparejamiento utilizando dicho camino de aumento, lo cual consigue un emparejamiento PERFECTO:



① Ya que los caminos de aumento DEPENDEN del emparejamiento actual, se suelen llamar caminos de M-aumento. Esto porque M es la letra que se suele usar para denotar un emparejamiento (M de MATCHING).

② El algoritmo de Blossom proporciona un procedimiento para encontrar caminos de aumento en un grafo, lo cual nos permite, ITERATIVAMENTE, mejorar el emparejamiento que llevamos con cada paso.

③ Para asegurarnos que siempre funciona, sabemos que un emparejamiento es MAXIMO  $\leftrightarrow$  No existen más caminos de aumento, esto se conoce como el Teorema de Berge, el cual se fundamenta en el Lema del mismo autor.

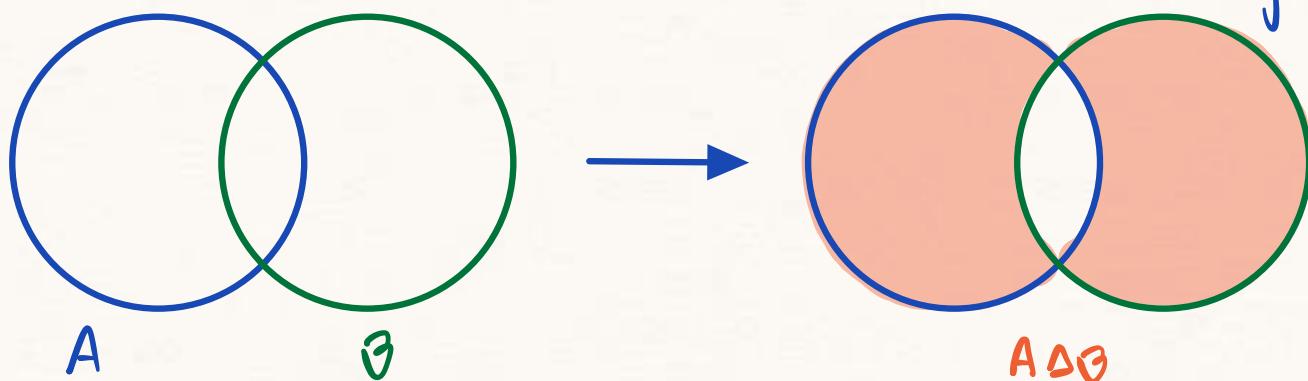
## Lema de Detog

- \* Sea  $G$  grafo y sean  $M, M'$  empaquetamientos sobre  $G$ .
- \* Sea  $G'$  grafo resultante de tomar los vértices de  $G$  con las aristas del conjunto  $M \Delta M'$

Se verifica que  $G'$  sera' compuesto por componentes de la forma :

- i) Vértices aislados
- ii) Ciclos PARES con aristas ALTERNANDOSE en  $M$  y  $M'$
- iii) Caminos con aristas ALTERNANDOSE en  $M$  y  $M'$

• Pruebe que  $A \Delta B$  es la DIFERENCIA SIMÉTRICA de dos conjuntos:



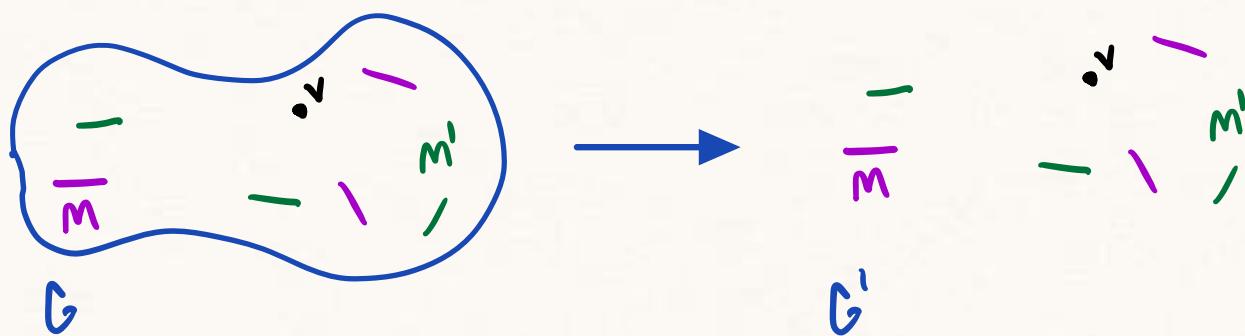
D) Son los elementos que están en un conjunto o (EXCLUSIVO) en otro.

También se puede escribir así:  $A \Delta B = (A \setminus B) \cup (B \setminus A)$

Vémost.

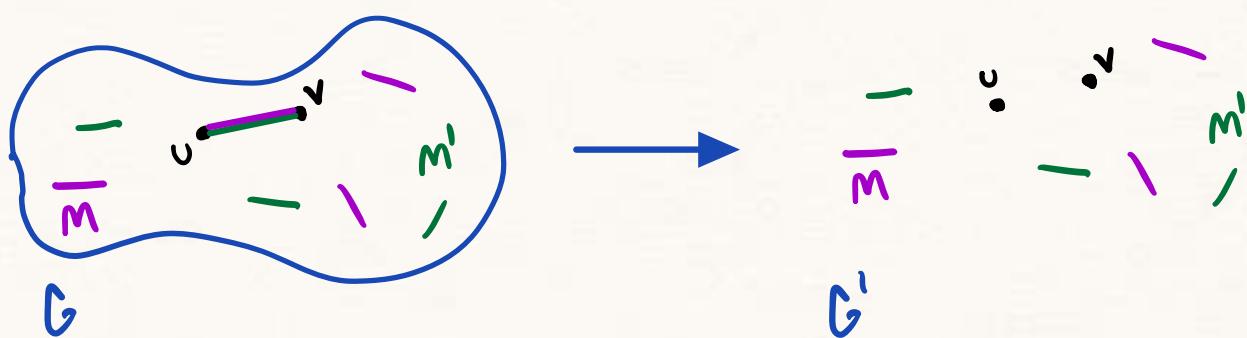
Abuso de notación, aquí nos referimos a que ~~no~~ ~~ninguna~~ existe que incluya a  $v$  en los enzorzamientos  $M, M'$ .

- Caso  $v$ , vértice en  $G_1 \notin M, M'$



- Ya que las únicas existen que habrá en  $G'$  son, como mucha, las que hay en  $M, M'$ . Y  $v \notin M, M'$  ENTONCES  $v$  OTRO AISLADO en  $G'$  (caso i)

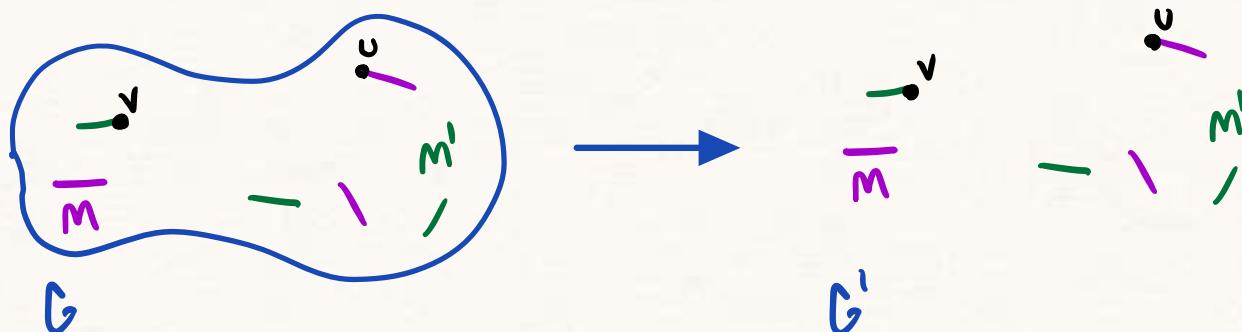
- Caso  $u, v$  vérticos /  $uv \in M \cap M'$



- Ya que los aristas de  $G'$  son las de  $M \Delta M'$ , aquellas que están TANTO en  $M$  como en  $M'$  se QUITAN en  $G'$ . Luego,  $u, v$  estarán aislados en  $G'$ . (caso i otra vez)

- Caso  $u, v$  vérticos /  $u \in M \wedge v \notin M \wedge v \in M'$

▷ cuando los vértices usados en  $M$  NO son usados en  $M'$  y viceversa.



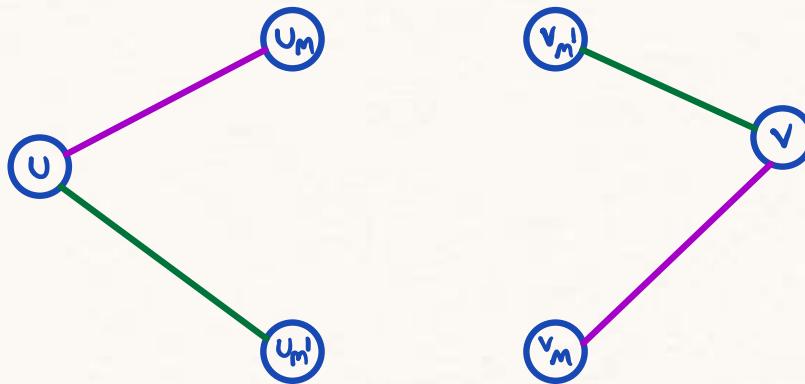
- (1) En este caso, en  $G'$  los vértices solo se conectan con su gente (la cual está especificada en  $M$  o bien en  $M'$ )
- ▷ EXCLUSIVO
- (2) De este forma, los vértices de este caso tendrán, como MAXIMO, grado 1. Estamos entonces en el caso iii, porque tenemos en  $G'$  caminos (de

LONGITUD 1, pero al fin y al cabo caminos) de vértices en  $M$  y  $M'$  (no hay alternación  $M \leftrightarrow M'$  pero esto es porque la longitud de los caminos es 1).

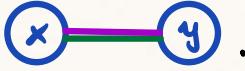
- Caso  $u, v$  vérticos /  $u, v \in M, M'$  ↗ Caso más general

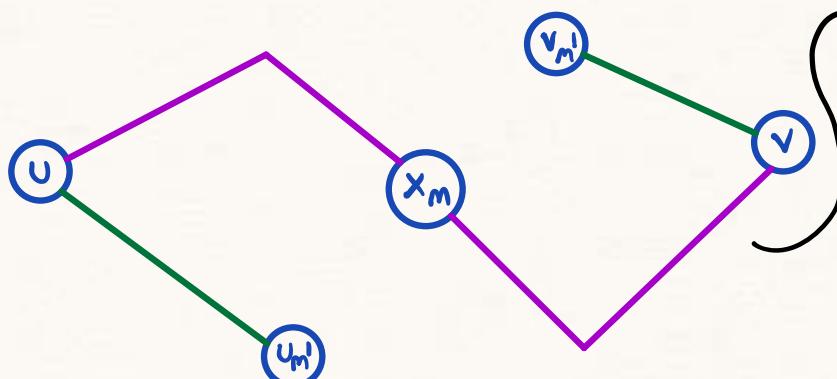
↓ Vamos a desarrollar esto:

- Si  $u \in M \rightarrow \exists uu_M \in M$   
 Si  $u \in M' \rightarrow \exists uu_{M'} \in M'$ ; Esto análogo para  $v \in M, M'$ :



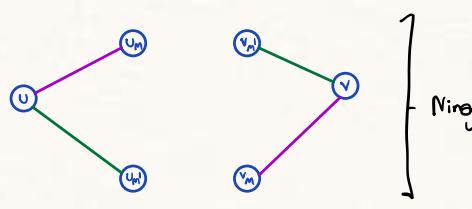
- De este dibujo se sacan cosas interesantes.

- Si  $u_M = u_{M'}$  o bien  $v_M = v_{M'}$  entonces tendríamos algo así:  En cuyo caso, en  $G'$   $x, y$  están unidos (caso i)
- Se verifica que  $u_M \neq v_M \wedge u_{M'} \neq v_{M'}$ . Y si que, de lo contrario, tendríamos algo así:

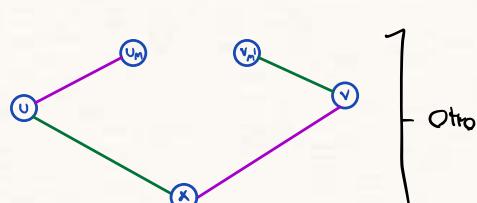


El vértice en cuestión tendría DOS arcos, lo que es imposible. En este dibujo se refleja el caso  $U_M = V_M$  (que denotamos  $X_M$ ), pero será análogo que el caso de los vértices de  $M'$  (o ambos a la vez)

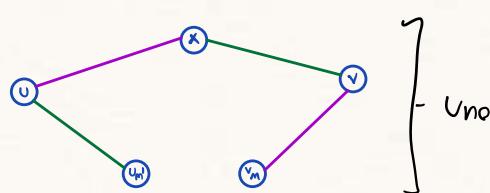
i) Si queremos que  $U_M = V_M$  o bien  $U_M^1 = V_M$  (o ninguno).



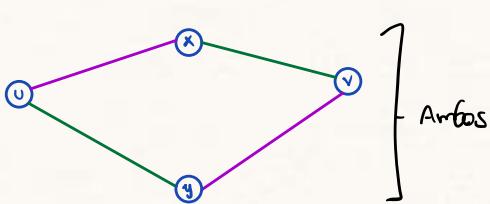
Ninguno



Otro



Uno

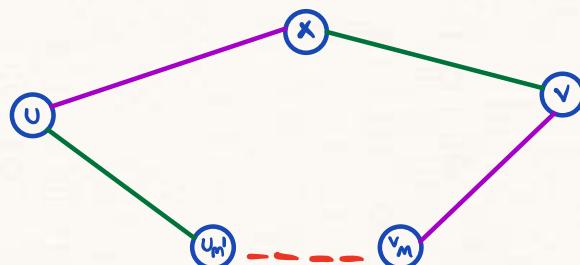


Ambos

ii) Notese que el resultado son siempre caminos con estos alternándose en  $M$  y  $M'$ , que

para conectar siempre tenemos que ALTERNAR. Esto da como resultados casos iii o bien, si hacemos ciclos, casos ii

iii) Notese que es IMPOSIBLE hacer ciclos impar:



iv) Si quisieramos conectar

aquí c' que empaquetamiento  
 tenemos?  $\{M \cup \text{bien } M'\}$ ?  
 Ninguno es viable porque d  
 resultado da' un vértice con  
 DOS estrechos en  $M$  o bien  $M'$ ,  
 lo cual no es válido.

## Teatema de Petje

- Sea  $G$  grafo y sea  $M$  un empaquetamiento sobre  $G$ .
  - ↳  $M$  es MAXIMO  $\rightarrow$  ↳ caminos de  $M$ -aumento

Demost

- ( $\rightarrow$ )  $M$  es MAXIMO  $\rightarrow$  ↳ caminos de  $M$ -aumento
- Pas. Absurdo

- Supongamos que  $M$  es máxima y, sin embargo,  $\exists P$  camino de  $M$ -aumento.
  - ↳ Por definición, si  $P$  es un camino de  $M$ -aumento entonces, invirtiendo sus aristas, podemos conseguir un empaquetamiento  $M' / |M'| = |M| + 1$ , es decir, un empaquetamiento mejor que  $M$ .
  - ↳ Luego,  $M$  NO es máximo ¡contradicción! Por lo que se concluye que ↳ caminos de  $M$ -aumento.

( $\rightarrow$ )  $\nexists$  caminos de  $M$ -aumento  $\rightarrow M$  es MAXIMO

Pred.  
Absurdo

- Supongamos que  $\exists$  caminos de aumento en  $G$  con  $M$  y, sin embargo,  $M$  no es máximo.

↳ Si  $M$  no es máximo  $\rightarrow \exists M'$  emparejamiento en  $G$  /  $|M'| > |M|$

- Consideremos  $G'$  el grafo compuesto por los vértices de  $G$  y las aristas del conjunto  $M \Delta M'$

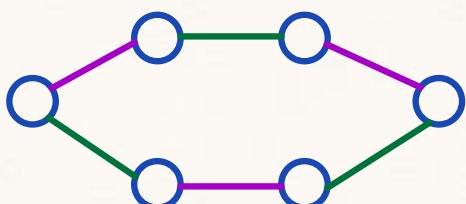
↳ El LEMA de Berge nos describe la estructura del grafo  $G'$ : vértices aislados, ciclos pares ALTERNADOS o bien caminos ALTERNADOS.

- Como  $M'$  es MEJOR que  $M$ , se tiene que verificat que  $M'$  tiene MAS aristas que  $M$ . Esto, en términos de  $G'$ , se traduce en que EXISTE un camino alternado IMPAR en  $G'$  con MAS aristas de  $M'$  que de  $M$ .

↳ Esto se puede ver dándonos cuenta que los vértices aislados y los ciclos pares alternados tienen el MISMO número de aristas tanto de  $M$  como de  $M'$ .

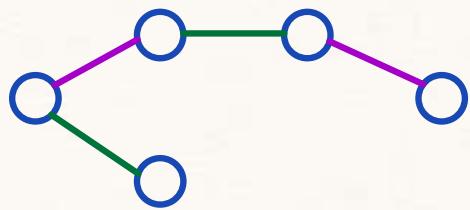
↳ Los vértices aislados tienen 0 aristas.

↳ Los ciclos pares alternados tienen  $2n$  aristas ( $n$  de  $M$  y  $n$  de  $M'$ )

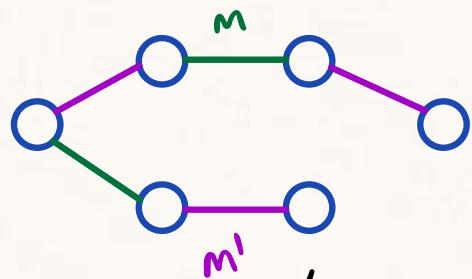


↳ Los caminos alternados de longitud par, al igual que los ciclos pares, tienen

el mismo número de aristas tanto de  $M$  como de  $M'$ .

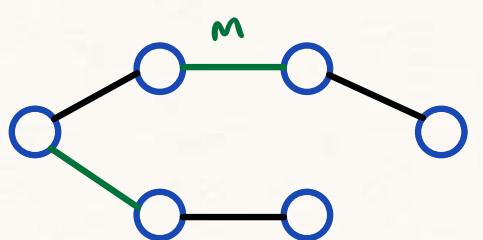


De este modo, la única forma en la que  $M'$  sea mayor que  $M$  es que existe algún camino alternado IMPAR con más aristas de  $M'$ :



Por construcción, los extremos de este camino han de ser vértices empaquetados con  $M'$  (de lo contrario habrían más aristas de  $M$  que de  $M'$ )

Dicha camino es de  $M$ -aumento:

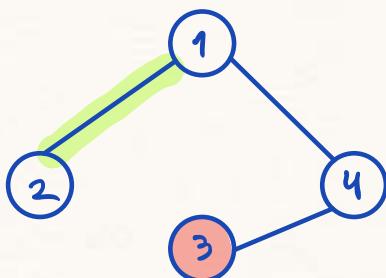


Los extremos estaban empaquetados en  $M'$ . Luego NO lo están en  $M$  y el interior del camino se compone por vértices  $M$ -empaquetados (por construcción de  $G'$ ).

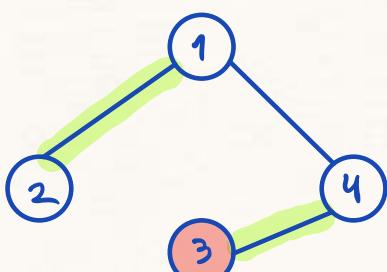
Hemos encontrado un camino de  $M$ -aumento sugiriendo que no existe ninguna contradicción! Por lo que se concluye que  $M$  es MAXIMO.

# Idea del Algoritmo de Blossom

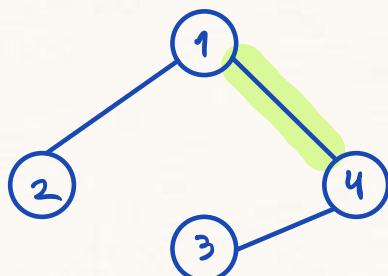
- 1) Así y como comentamos el principio, se trata de un algoritmo que ENCUENTRA caminos de aumento en el grafo, incrementando / mejorando el emparejamiento actual mediante dichos caminos.
- 2) Un camino de aumento inicia y termina en vértices NO emparejados, por lo que, intuitivamente, tendría sentido que dichos vértices sean el punto de partida del algoritmo.
- 3) Vamos a explicar una **VERSIÓN SIMPLIFICADA** de la dinámica del algoritmo, vemos los problemas que surgen y, luego, las soluciones que utiliza el algoritmo real / completo para solventarlos:
- 4) Vamos a iniciar escogiendo ALAATORIAMENTE un vértice no emparejado e intentaremos movernos desde este hasta otro vértice no emparejado utilizando estas ALTERNANCIAS, básicamente que construye el camino de aumento en cuestión.
- El caso fácil es que encontramos un vecino que tampoco esté emparejado. En cuyo caso habremos encontrado el camino de aumento más sencillo.



→ 4 vecino no emparejado.  
El camino 3,4 es de aumento. Podemos repetir el emparejamiento con él:



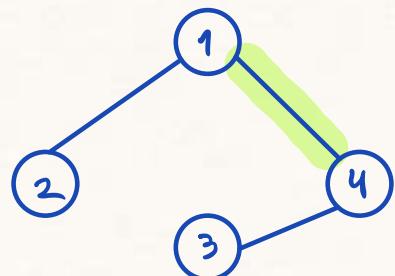
- El otro caso es que encontramos un vecino YA empaquetado. Cuando esto ocurre lo que se hace es meter a SU PAQUETA en la lista de vértices que comprobaremos (lista que se inicializa con el vértice no empaquetado que se escoge aleatoriamente al principio).



`toCheck = []`



↓  
Se escoge 3  
aleatoriamente  
(vértice no empaquetado)



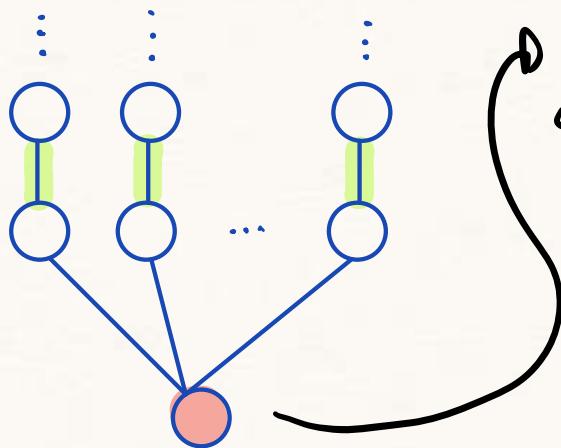
`toCheck = [3]`

→  
↓  
Empaquetamos a  
elevat al siguiente  
de la lista

1) Luego valiéndonos a bajar en nodo no emparejado que intentar construir otro camino de cumento. Pero como todos los nodos ya están emparejados nos  $\Rightarrow$  ninguno. Luego, hemos encontrado un emparejamiento MAXIMA y fin.

\* ① En casos fáciles se puede reconstruir el camino de cumento a ojo gato, en general, sea necesario llevar un REGISTRO de lo que hemos ido visitando. Esto, en el algoritmo real se corresponde con la estructura de datos POKET, que se denota con F normalmente.

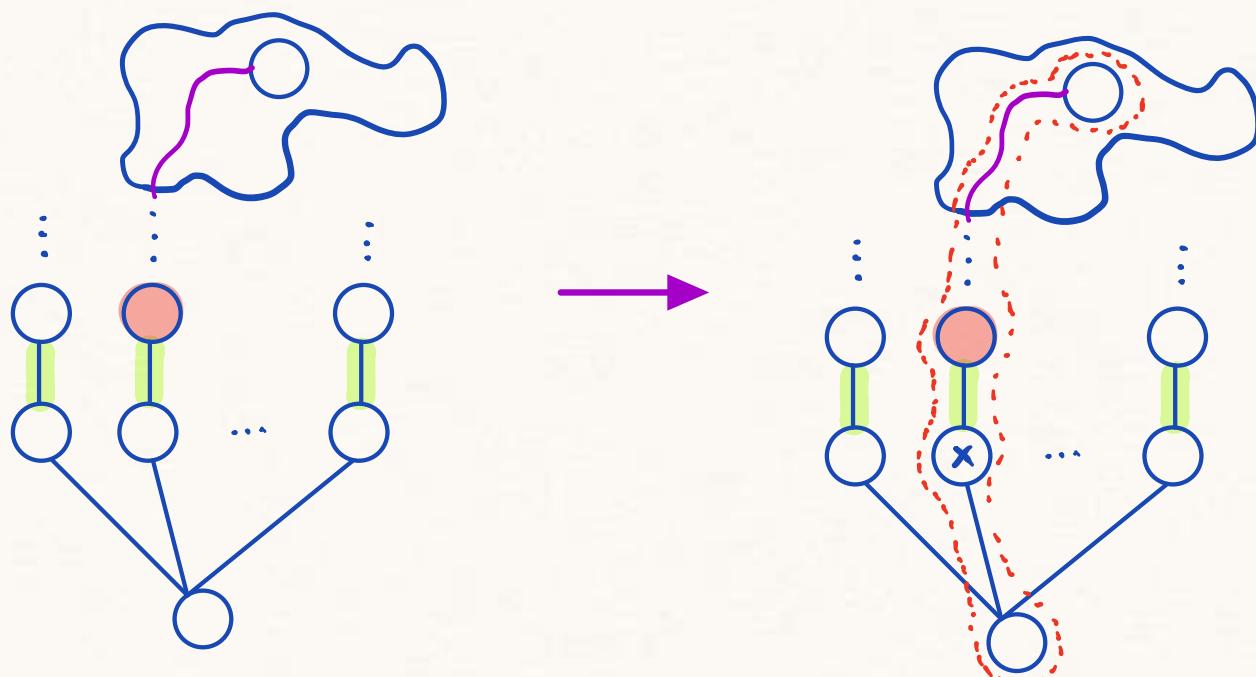
② El approach anterior es sencillo y utiliza una idea muy intuitiva e intuitiva que es la de: "preguntar" a nuestros vecinos EMPAREJADOS si SUS vecinos quedan libres a DTRD con quién emparejarse que es NOSOTROS emparejados con ellos.



→ Cuando evaluamos este nodo, nos damos cuenta que TODAS las posibles parejas ya están ocupadas (vemos que este es el caso :/ ).

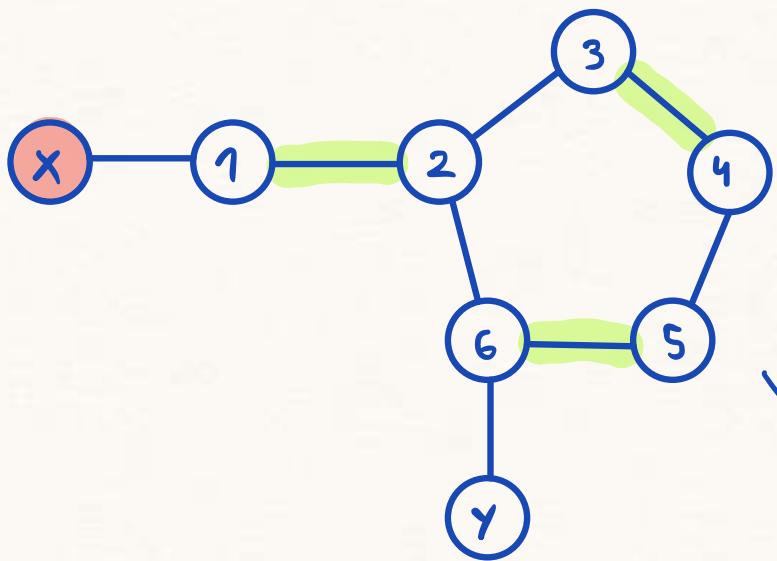
→ La única forma que este nodo se empareje es que algunas de sus posibles parejas CAMBIE su actual:

Por el b, cuando se detecta un vecino YA empaquetado, el algoritmo añade a la lista por comprobar justamente a sus hijos, desde los cuales buscamos otros nodos no empaquetados que así poder hacer los SWITCHES de apertura necesarios para que así los nodos solos "listos" queden empaquetarse:

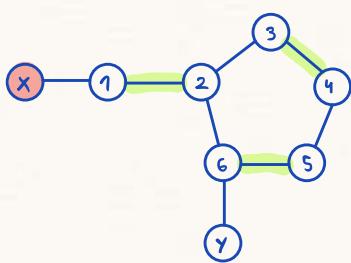


④ Comino de aumento encontrado, nuestro nodo inicial no empaquetado podrá tener apertura gracias a que le que tiene X ahora le podrá combinar.

⑤ Podímos pensar que este versión del algoritmo funciona siempre pero hay ciertos casos en los que surgen problemas:

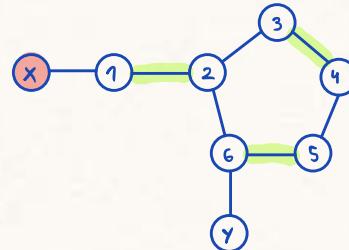


Notase que  $3$  es en camino de aumento entre  $X$  e  $Y$  ( $X-1-2-3-4-5-6-Y$ ). Pero el algoritmo NO lo encuentra.

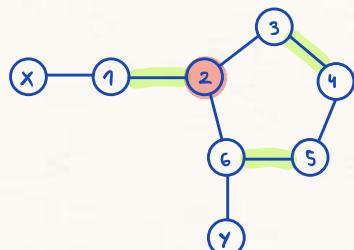


`toCheck = []`

Vertice 1 vecino  
empaquetado. Añadimos su  
vecino (2) a la lista



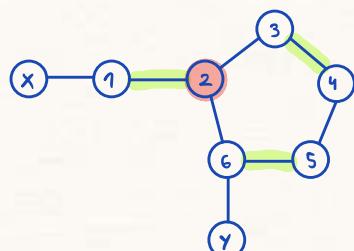
`toCheck = [2]`



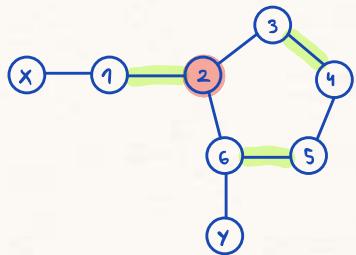
`toCheck = []`

Se acabaron los vecinos.  
Paramos el siguiente de  
la lista.

Vertice 3 vecino  
empaquetado. Añadimos su  
vecino (4) a la lista



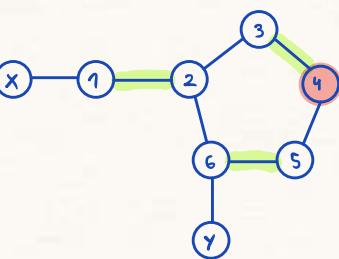
`toCheck = [4]`



Vertice 6 vecino  
empejado. Añadimos su  
vecino (5) a la lista

`toCheck = [4, 5]`

Se acaban los vecinos.  
Paramos el siguiente de  
la lista.



`toCheck = [5]`

Vertice 5 vecino  
empejado pero su  
vecino 6 esté en la lista  
se comprobó. Lo omitimos.  
Y se acaban los vecinos.  
Paramos al siguiente de la lista.

`toCheck = []`

Vertice 4 vecino  
empejado pero que  
YA ha sido visitado. Lo  
omitimos. Lista vacía  
entonces ya termina de  
cumentar ¡Falso!

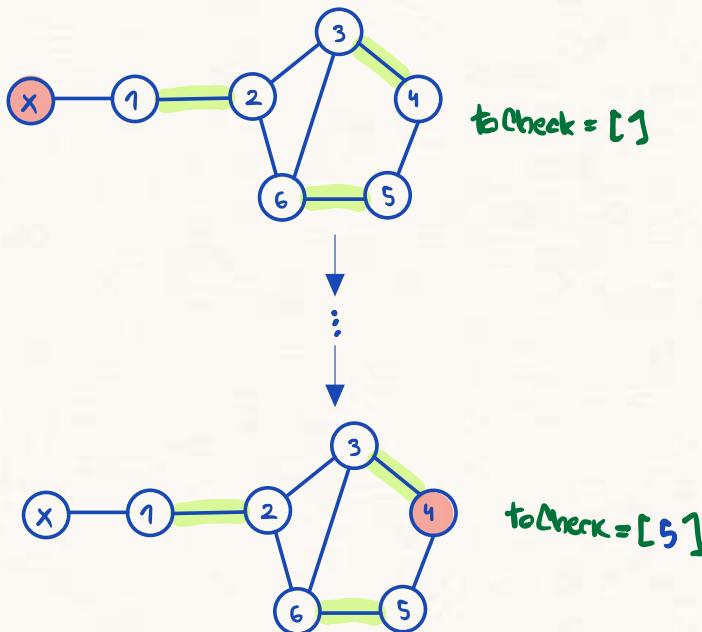


Estas decisiones de "omitar" quedan bastante arbitrarias pero se puede apreciar  
que tienen bastante sentido si mostramos la estructura de datos de nodos  
visitados:

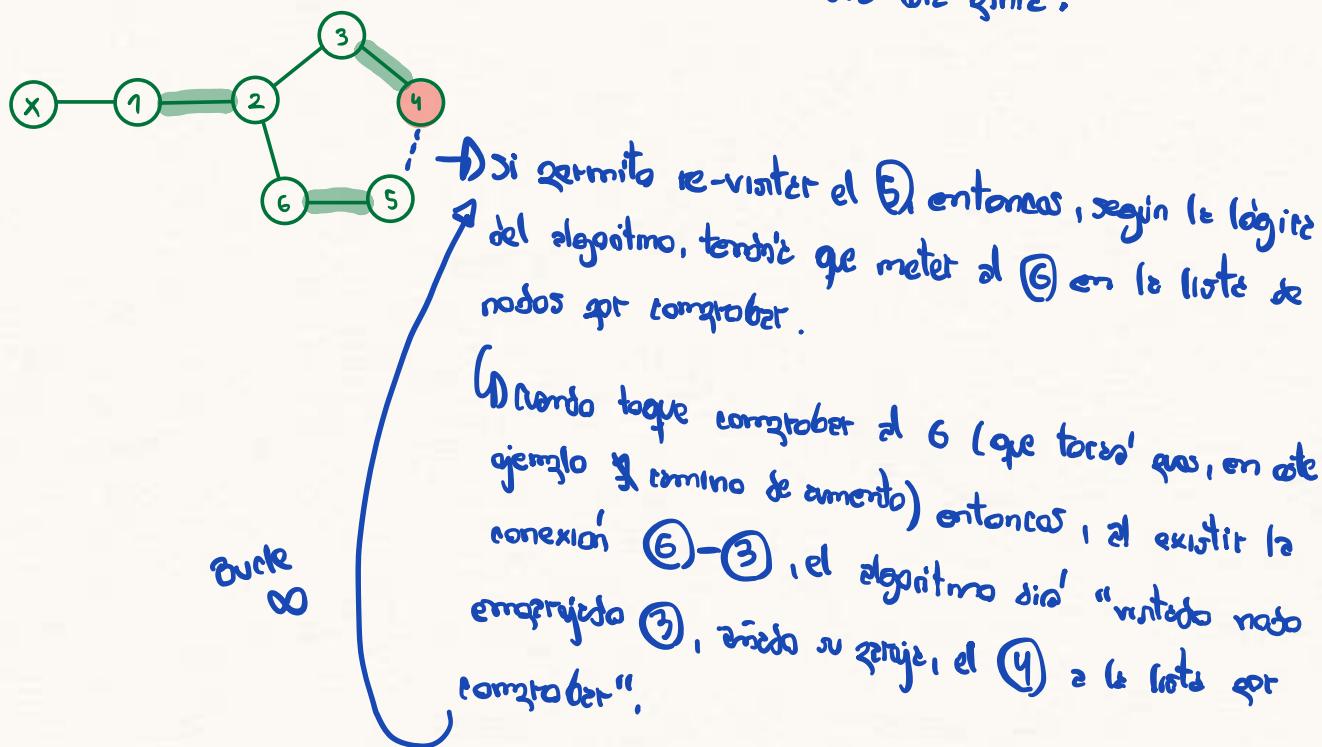
La estructura que usamos es un ÁRBOL justo que NO podemos  
permitir CICLOS en la malla, o sea, no podemos re-visitar nodos.

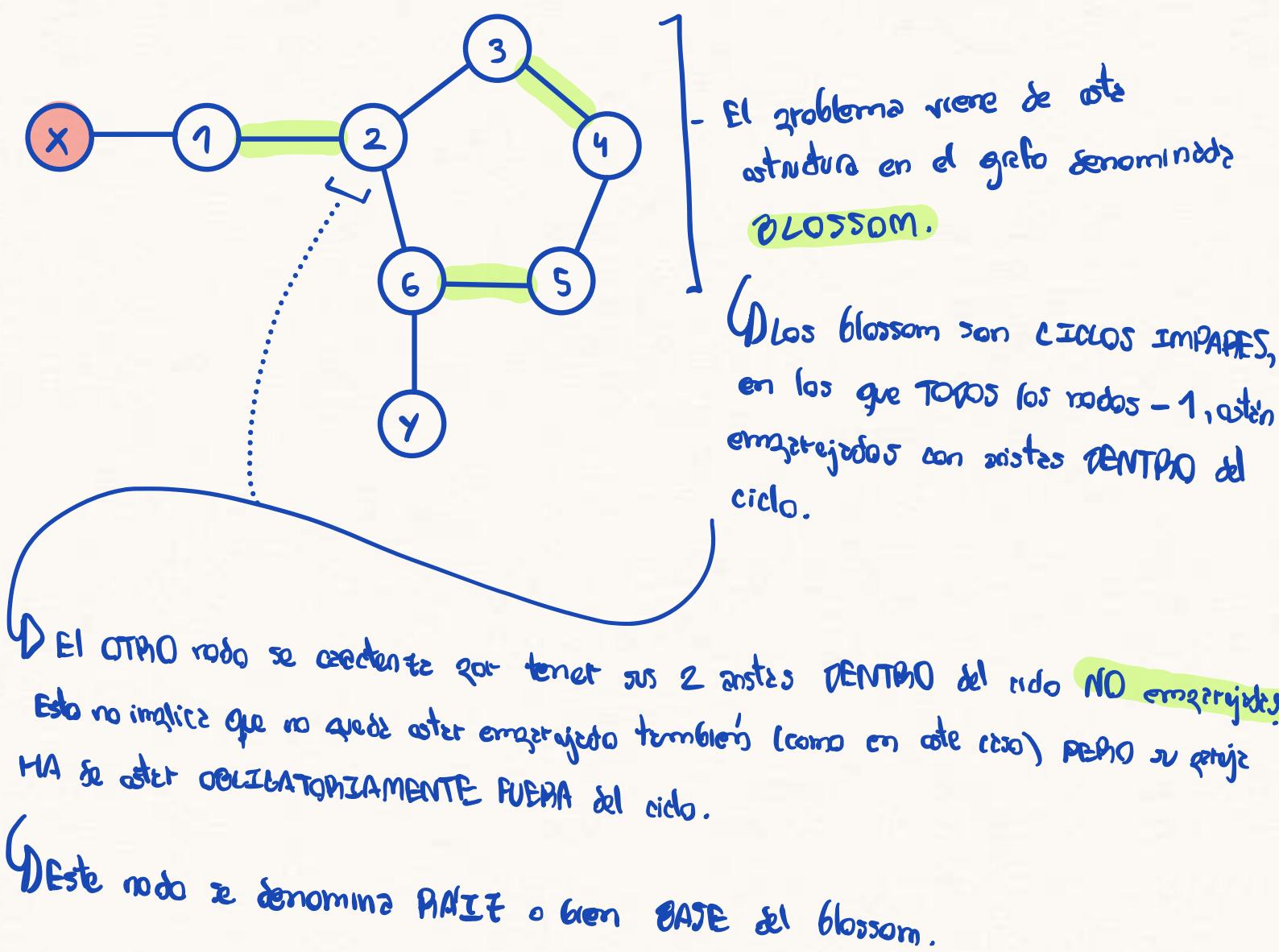
¿Por qué? Pw, por ejemplo, podemos leer en un bucle así:

Ej.

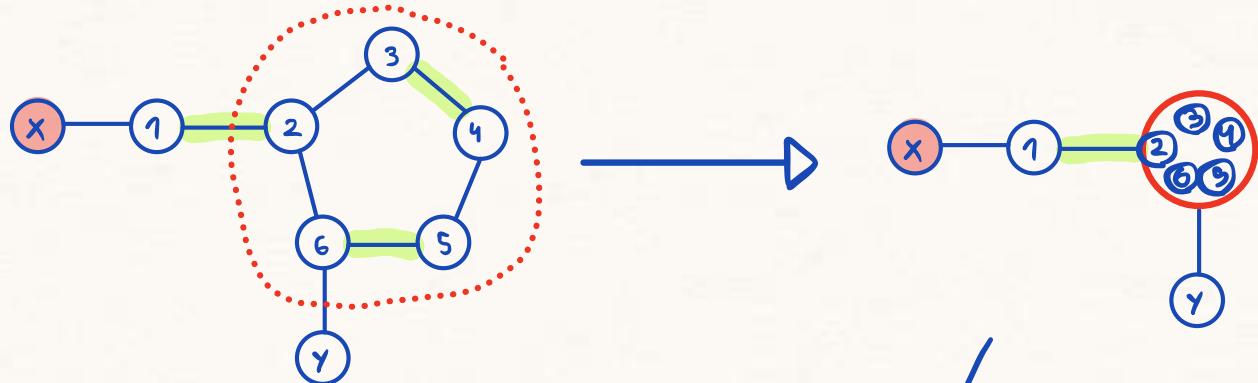


En este punto la estructura de datos de visitados tendrá este valor:





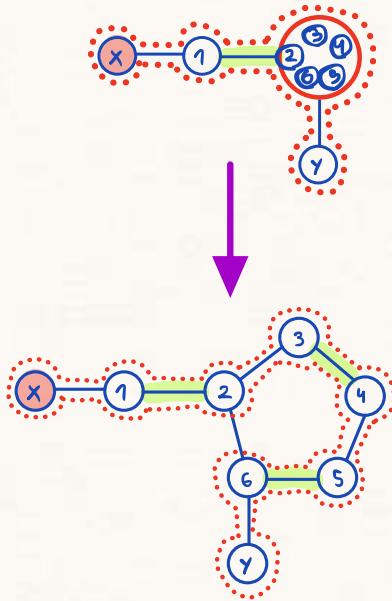
① La forma en la que el algoritmo soluciona el problema de los  
blossoms es mediante su CONTRACCIÓN:



④ A partir de aquí el algoritmo  
continua de la misma forma.

caminos de aumento como si se tratase de otro grafo diferente.

Al final, si encontramos un camino de aumento ~~básicamente reconstruido DESHALAZO~~ las contracciones que hará (de manera inversa) nos conseguirán un camino de aumento en el grafo original.

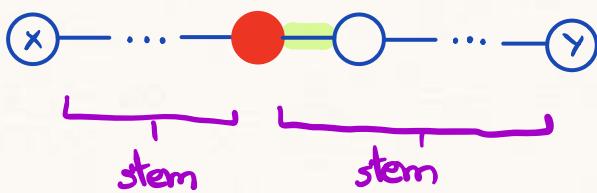


¶ Para asegurarnos que todo esté funcione son necesarios algunos resultados...

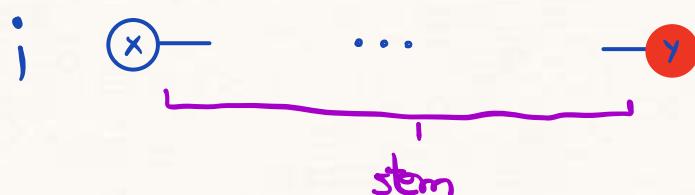
## Tallo / Stem de un Blossom

¶ Cuando CONTRAEMOS un blossom obtenemos un nuevo grafo sobre el cual el algoritmo operará. Si ocurre que encuentra un camino de aumento que CONTENGA a algún nodo-blossom contraído entonces podemos tener, principalmente, dos casísticas:

- Que esté en medio



- Que esté en un extremo

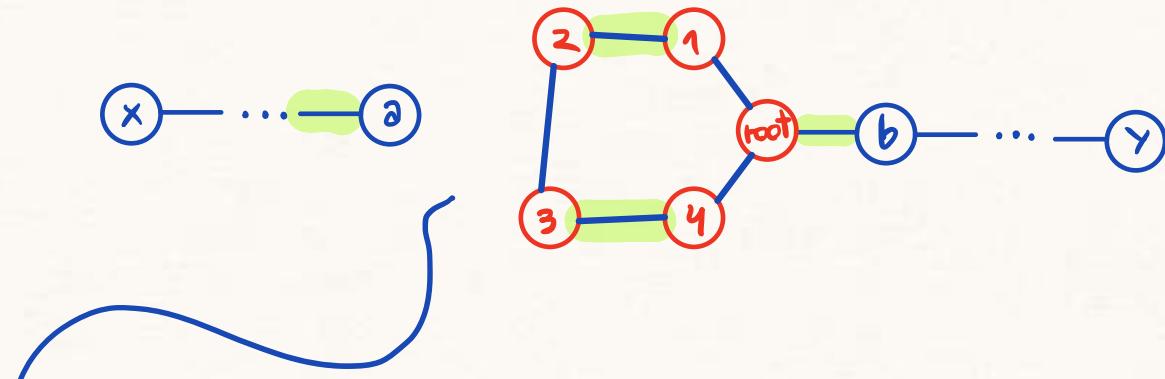
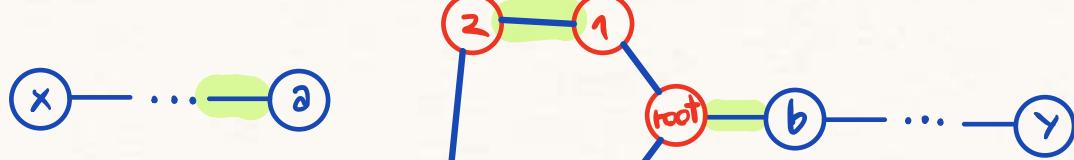


▷ Los caminos alternados que conectan los extremos del camino de aumento  $(x, y)$  con el vértice -blossom se denominan TALLOS o bien STEMS.

① Cuando tenemos un blossom en MFGO, la manera de deshacer la contracción es mirando a los nodos con los que se conecta en el camino de aumento:

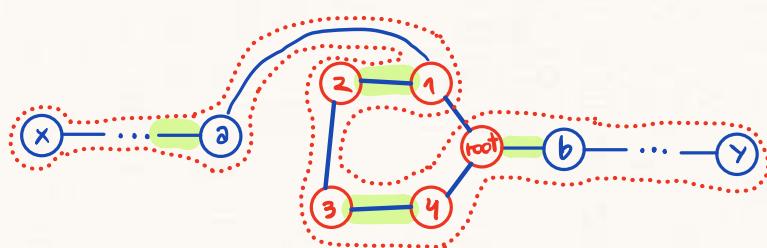


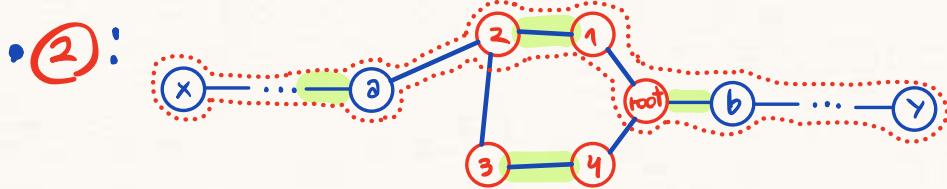
▷ Sabemos que sea la RAÍZ que es el único nodo de un blossom que queda fuera del círculo.



▷ Regeniendo de donde se enganche el nodo ② al blossom reconstruiremos el camino de una forma u otra:

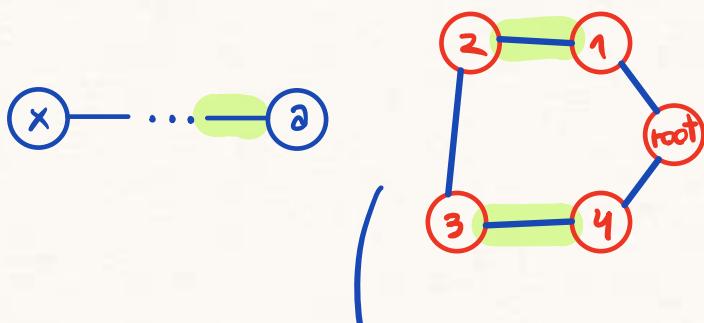
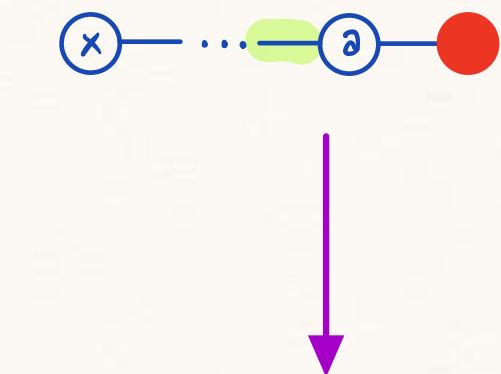
• ① :





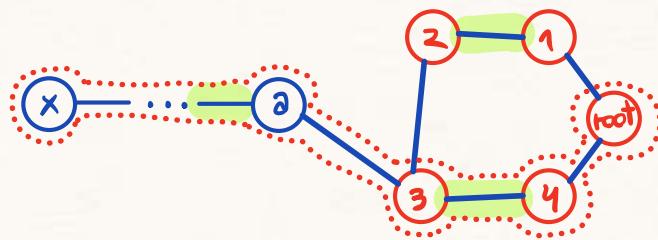
6) Y así con el resto...

• ① Cuando tenemos un blossom en los EXTREMOS se sigue la misma dinámica que antes, solo que en este caso sabemos que la raíz del blossom se convertirá en uno de los extremos del camino de currento.



▷ Otra vez depende de cómo esté el nodo ② enganchado para recorrer el ciclo pero SIEMPRE la raíz será un nodo extremo.

• Veamos qué ocurriría que ② esté enganchado con ③:



## Teorema

- \* Sea  $G$  grafo y sea  $\beta$  blossom sobre  $G$ .
  - \* Sea  $G'$  grafo resultante de CONTRAER  $\beta$  en  $G$
- ↗ El camino de aumento en  $G \iff$  El camino de aumento en  $G'$

### Demost

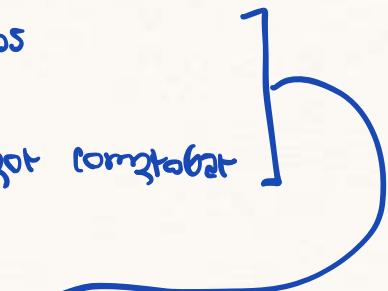
- Para grabarlo basta ver que todo camino de aumento en  $G$  se quede convertit en uno en  $G'$  y viceversa.
  - 1) Si  $P'$  es un camino de aumento en  $G' \rightarrow$  Puedo construir uno válido en  $G$  doblando la compresión de  $\beta$  tal y como vimos en la explicación de los stems.
  - 1) Si  $P$  es un camino de aumento en  $G \rightarrow$  Puedo construir uno válido en  $G'$  comprimiendo  $\beta$  y siguiendo una lógica análoga para mantener la validez del camino de aumento.
- \* La potencia de este Teorema es que se quede aplicar de forma RECURSIVA, es justo lo que el Algoritmo de Blossom aprovecha.

# Algoritmo de Blossom

→ Proporciona una manera estructurada para encontrar caminos de aumento en cualquier grafo, contienda además con la capacidad de DETECTAR y MANEJAR la existencia de BLOSSOMS.

→ Para ello mantiene 2 estructuras de datos durante su ejecución:

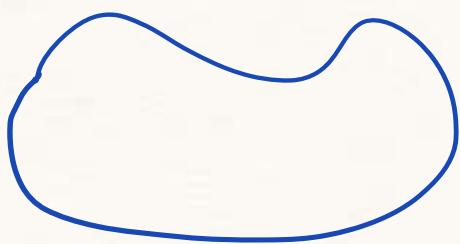
- Un BOSQUE de nodos
- Una COLA de nodos que comprobar



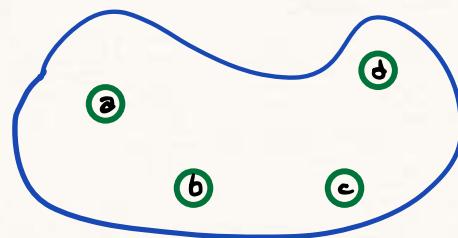
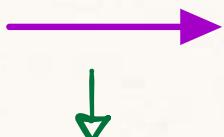
→ La dinámica que sigue el algoritmo es "plantar" por cada nodo NO enraizado un ÁRBOL. Inicialmente cada árbol sólo consta de su raíz (el nodo no enraizado correspondiente).

→ El conjunto formado por todos estos árboles es justamente la estructura de BOSQUE que comentamos. Lo que intentará el algoritmo será CONECTAR dos árboles mediante un camino de aumento.

→ Por otra parte, el algoritmo inicializa la cola de nodos que comprobar también con los nodos no enraizados del grafo.



Punto de Partida



Iniciaremos  
el bosque y la cdz  
de nodos qd corroboret

$\text{toCheck} = [a,b,c,d]$

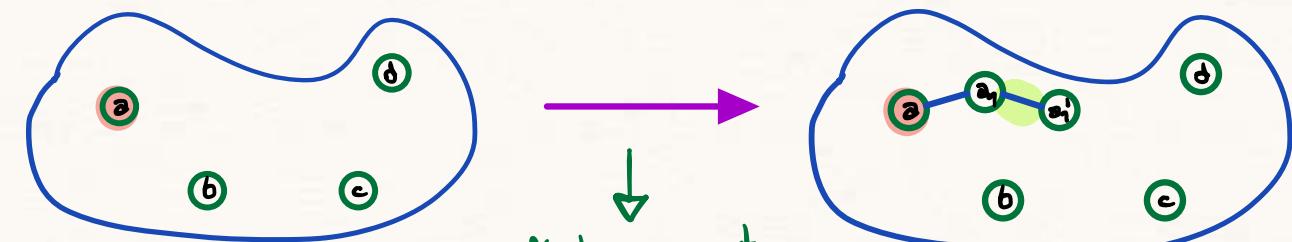
① A partir de ahora viene la fase INTERATIVA del algoritmo, donde iteramos evaluando nodo a nodo los de la cola. Lógicamente, el algoritmo para si encuentra un camino de aumento o bien se vacía la cola.

↳ 1) Para cada iteración, desencolamos un nodo y EVALUAMOS sus vecinos; en este punto lo interesante es ver si el vecino está o NO en el bosque

② Si el vecino NO está en el bosque entonces YA está ENPAPEJADO. Esto lo sabemos porque el algoritmo inicializamos justamente con los nodos NO enpaapejados

↳ 2) Cuando esto sucede digamos que no hemos tenido suerte, puesto que este vecino como tal no nos vale para un camino de aumento PERO quizás si continuamos con su vecindad igual encontraremos alguno, cosa es la idea de la versión simplificada del algoritmo de "Pregúntale a tu vecino" a ver si se queda empaapejado con otro y así ya podríamos encontrarlo".

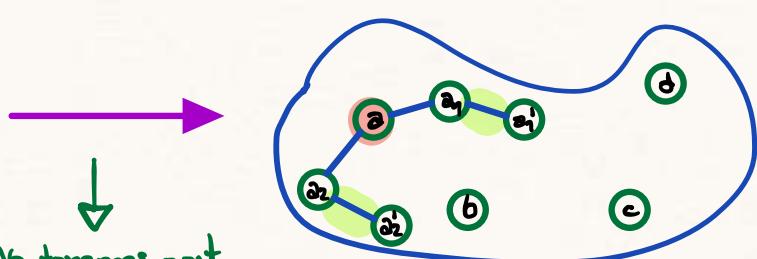
↳ 3) En este sentido el algoritmo lo que hace es añadir al vecino y a su PAPEJA el bosque y, también, a su vecindad a la cdz de nodos qd corroboret, qd es si, si no encontramos más en las, intentar encontrar un camino de aumento a través de la vecindad.



$\text{toCheck} = [b, c, d]$

No tenemos verde.  
Encontramos al vecino  
 $a_1$  YA empaquetado con  
el nodo  $a'_1$

$\text{toCheck} = [b, c, d, a'_1]$



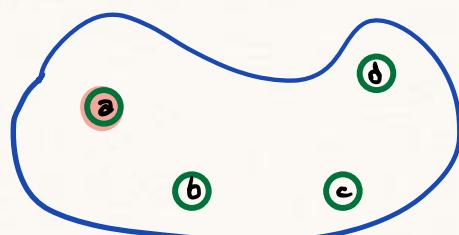
No tenemos verde.  
Encontramos al vecino  
 $a_2$  YA empaquetado con  
el nodo  $a'_2$

$\text{toCheck} = [b, c, d, a'_1, a'_2]$

...

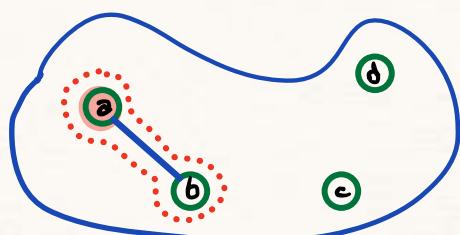
① El otro caso sería que el vecino YA esté en el árbol. En cuyo caso tenemos varias posibilidades: la más "fácil" se da cuando existe alguna conexión entre los nodos no empaquetados iniciales.

② En cuyo caso habremos encontrado el camino de aumento más sencillo: el de solo dos nodos.



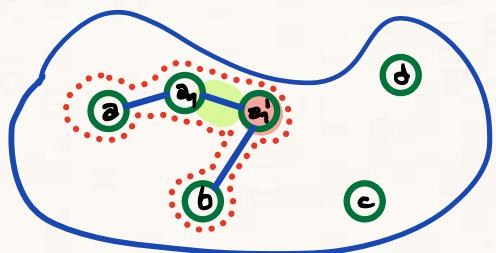
$\text{toCheck} = [b, c, d]$

Caso fácil, tenemos  
a b como vecino. Camino  
de aumento encontrado



- Si existe este caso, normalmente se da al principio. Esto se debe a que la lista de nodos que comprobamos se inicia con los NO empaquetados y, luego, se van añadiendo las zonas de los vecinos según los vamos encontrando.

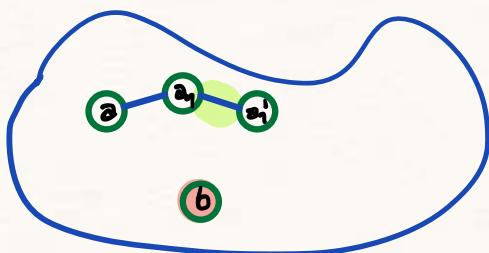
▷ Es decir, sería algo así:



. Básicamente

que tendímos que haber detectado la zona de un vecino ANTES que alguno de los no empaquetados del principio. Ojo esto es perfectamente válido pues el algoritmo incluso permite que la elección del siguiente nodo a comprobar sea ALEATORIA, pero lo usual es seguir el orden de la lista, lo cual hace imposible ver el caso anterior (que, adelantemos, antes de llegar a la zona del vecino, ya habremos construido el camino por otro lado).

- ④ La otra posibilidad es que el vecino ya esté en el bosque que reta que no sea un nodo no empaquetado. En cuyo caso volvemos a tener un set de posibilidades.

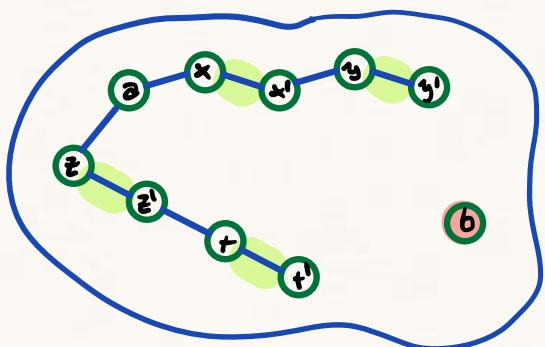


$toCheck = [a_1']$

- El set de posibilidades viene de ver DÓNDE se engancha el nodo  $b$ , es decir, si tiene como vecino a  $a_1$  o bien  $a_1'$

▷ La diferencia viene de que  $a_1$  NO nos vale para caminar de aumento y, sin embargo,  $a_1'$  SÍ.

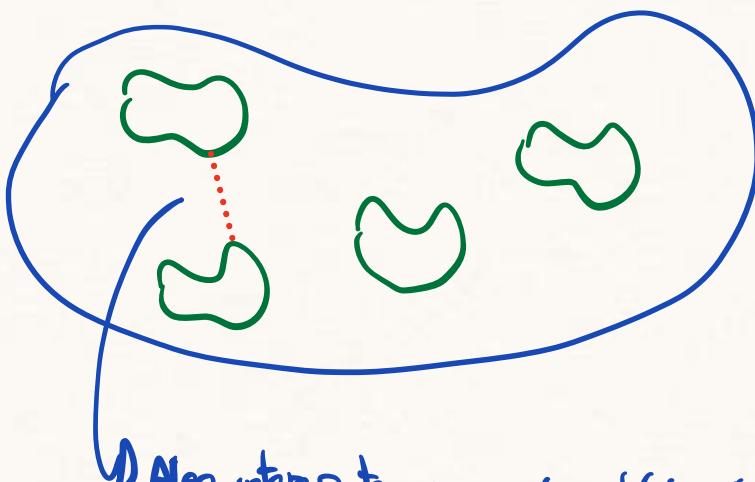
- Vamos a ponerlo más general:



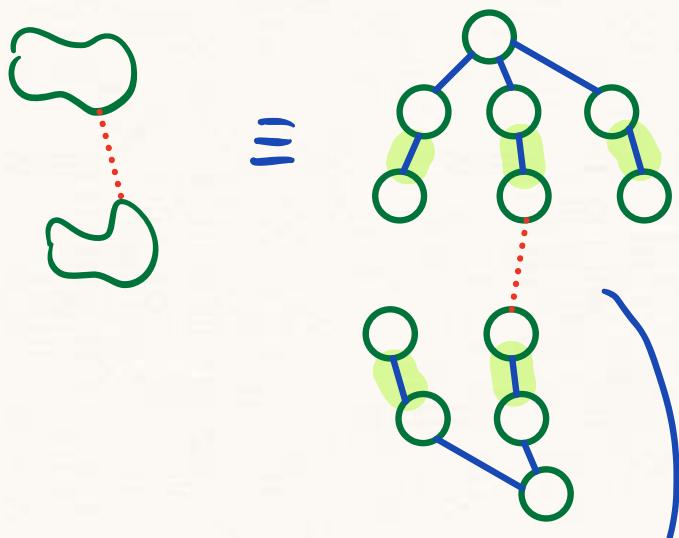
1

Notese que los casos buenos son siempre los nodos con una ' $\backslash$ ', es decir, las solapas de los vecinos viendo desde la PERSPECTIVA de la raíz.

↳ Al fin y al cabo, por construcción, un camino de suministro en el bosque lo que sea es una CONEXIÓN entre dos árboles iniciales

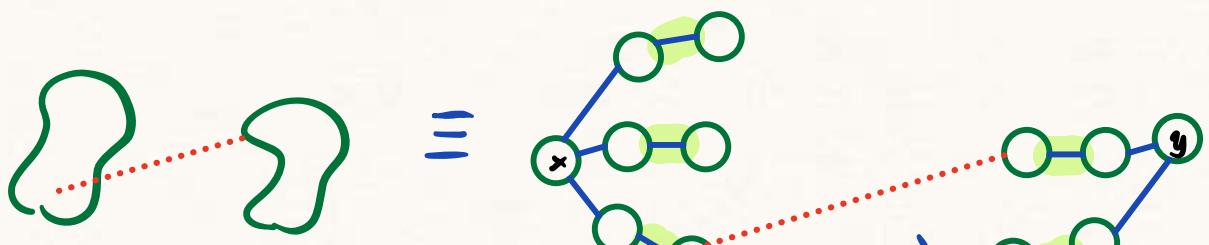


↳ Algo interesante es que los árboles "crecen" de forma PAR: siempre añadimos un vecino emparejado y su PARIEJA (esto para mantener la consistencia del hipotético camino de suministro).



1) Sabemos que este conexión ha de ser, obligatoriamente, una arista que NO engarza (si lo fuese tendríamos nodos con raíces distintas, lo cual no tiene sentido).

2) De este forma, si conectamos con esta arista, la siguiente que habrá que tomar, que mantener la consistencia del camino de suministro, será una que EMPAREJE. Podriamos pensar que tendremos que conectar mediante una HOJA del otro árbol porque realmente nos vale algún nodo más próximo cercano a la raíz:

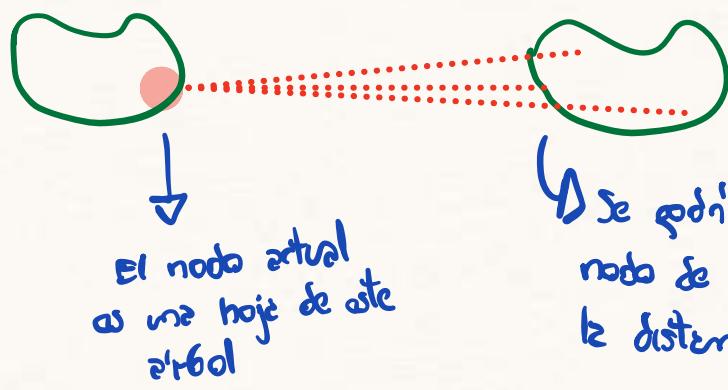


3) Esta conexión TAMBIÉN será válida, notarse que el camino de suministro entre x e y es perfectamente correcto.

• De este manera, la conexión entre los árboles se quedó hacer con cualquier nodo cuya siguiente arista (en su camino a la raíz) EMPAREJE. Notarse que estos nodos son, por CONSTRUCCIÓN, los que tienen una distancia PATH hasta la raíz.

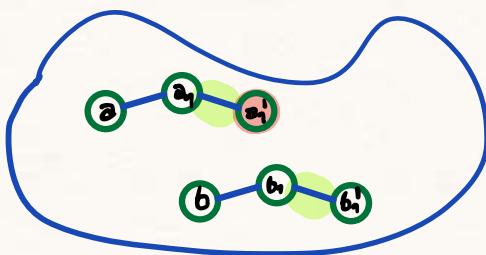
4) Algo que es importante especificar es que SIEMPRE el otro árbol si

se va a conectar donde una de sus hojas, esto porque en la lista de nodos que comprobamos siempre vamos metiendo hojas. Luego, este caso que estamos describiendo ocurre cuando nuestro nodo actual es una hoja de su árbol, que podríamos conectarlo a  $\underline{\underline{no}}$  a la hoja de otro árbol, por lo que el dibujo ilustrativo sería algo así:



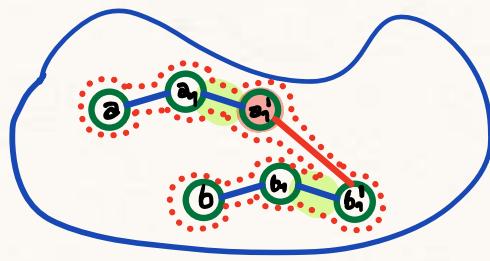
↳ Se podrían conectar con cualquier nodo de otro árbol siempre que la distancia a la raíz sea PAP.

- Presumiendo: cuando el vecino YA esté en el bosque y NO es alguno de los nodos no emparejados del gríñezin entonces tocó verificar si podemos CONECTAR de forma consistente los árboles formando un camino de suministro.
- ↳ Si la distancia del vecino a su raíz es IMPAP entonces NO nos vale, que no podemos conectar los árboles con una raíz que no emparej. En este caso simplemente IGNORAMOS al vecino y pasamos al siguiente (no hay nodo que encajar en el bosque ni a la lista porque al ya haber salido antes ya se habrían añadido las cosas correspondientes).
- ↳ Si la distancia del vecino a su raíz es PAP entonces hemos encontrado un camino de suministro: raíz del árbol del nodo actual  $\rightarrow \dots \rightarrow$  nodo actual  $\rightarrow$  vecino  $\rightarrow \dots \rightarrow$  raíz del vecino.



$\text{toCheck} = [b_1']$

vecino  $b_1'$  encontrado,  
que ya esté en el bosque  
y tiene distancia 2 (2<sup>st</sup>)  
a su raiz. ¡Bingo!

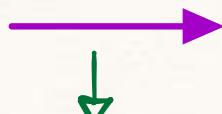
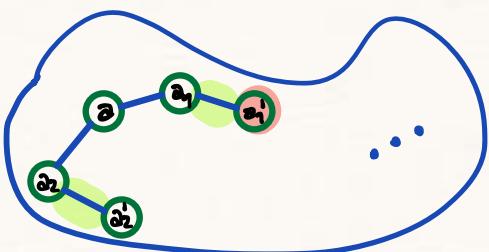


$\text{toCheck} = [b_1']$

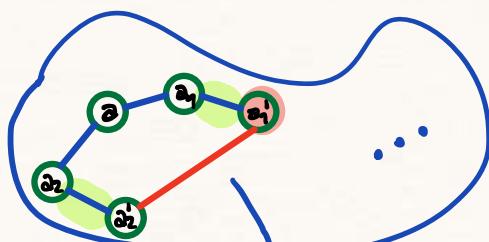
④ El algoritmo evalúa iterativamente todas las posibles maneras de hacer un camino de círculo desde TODOS los nodos no emparejados. Luego, si no consigue encontrar ninguno entonces es que ~~la~~ ninguna. Luego, por el Teorema de Berge, el emparejamiento sea máximo.

⑤ Puede darse que todo lo explicado hasta ahora cubre todos los casos pero nos falte ver el tema de los blossoms.

⑥ El algoritmo los detecta de una manera muy inteligente: si cuando vamos a construir un camino de círculo ocurre que los padres de los árboles es la misma, es decir, no estamos conectando árboles sino que estamos en el mismo que hemos detectado un blossom:



Encontrada vecino  $a_2'$  que  
ya esté en el bosque y tiene  
distancia 2<sup>st</sup> a su raiz. ¿Bingo?



▷ No realmente, no es un camino de aumento válido. De hecho es un blossom: desde nuestro nodo actual a la raíz hay un camino alternado de longitud  $q+1$ , lo mismo ocurre con el vecino encontrado. Si conectamos ambos caminos tendremos uno alternado de longitud IMPAR, siendo la raíz del blossom la otrariz  $q+1$  del árbol que es el único nodo en el ciclo que tiene sus dos aristas incidentes como NO encajadoras.

- Cuando se detecta el blossom, el algoritmo lo que hace es COMPROMIRLO y enquista otra vez el procedimiento con dicho nuevo grafo. Si encuentra un camino de aumento entonces lo que hace es DESCOMPROMIRLO adecuadamente los blossoms involucrados para que el camino sea válido en el grafo original. Si no encuentra ningún camino de aumento en el grafo comprimido entonces se que si ninguna en el grafo original que dí Teorema que vimos arriba.