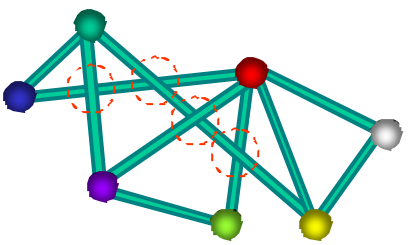
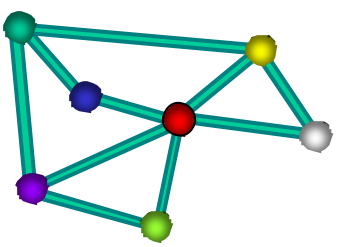
algoritmi per la planarità di grafi

disegni planari di grafi



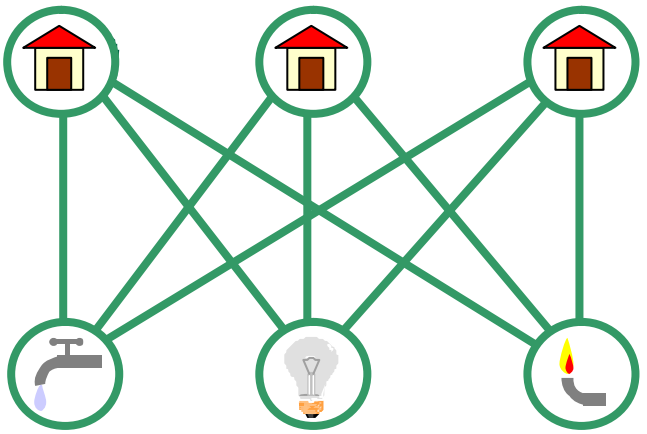
disegno non planare di un grafo



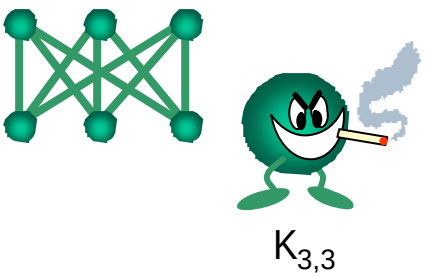
disegno planare dello stesso grafo

un grafo “non planare”

(che cioè non ammette un disegno planare)

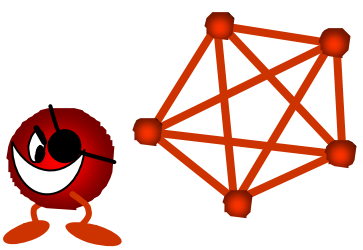


due (famosi) grafi non planari



$K_{3,3}$

grafo completo bipartito
di sei nodi



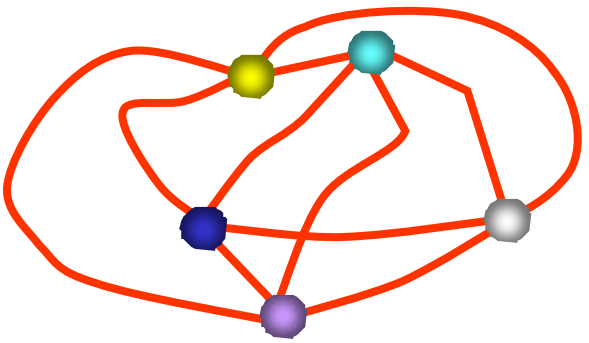
K_5

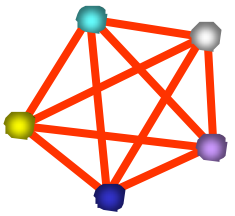
grafo completo di
cinque nodi

(tutti) i grafi non planari

kuratowsky (1930): un grafo non planare contiene al suo interno un sottografo “riconducibile” ad un K_5 o ad un $K_{3,3}$

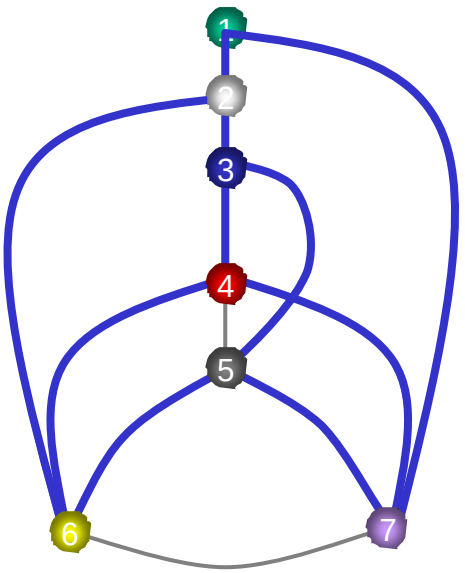
cosa vuol dire “riconducibile”?

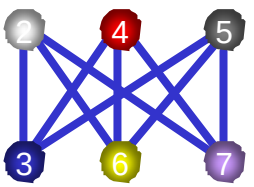




K_5

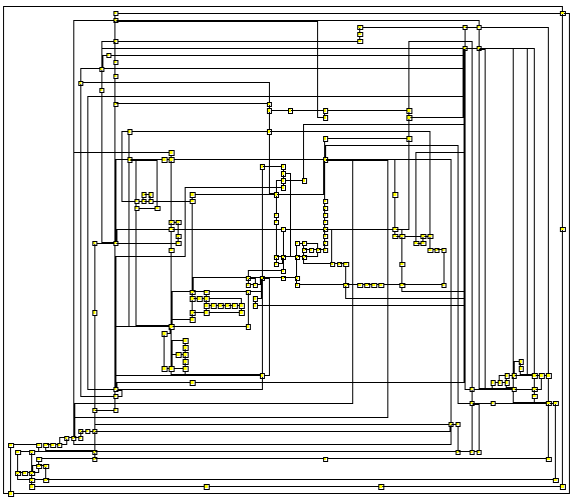
trova il $K_{3,3}$





$K_{3,3}$

perché ci piacciono i grafi planari (1)



le intersezioni generano equivoci

perché ci piacciono i grafi planari (2)

eulero: se un grafo con n nodi ed m archi è planare, allora $m \leq 3n-6$

se il grafo è planare, fare una operazione su ogni arco costa* quanto fare una operazione su ogni nodo

* = in termini di complessità asintotica

esempi di operazioni che vorremmo poter eseguire sugli archi:

- etichettare ogni arco con un valore
- percorrere ogni arco
- memorizzare tutti gli archi in una lista
- …

problemi sulla planarità

(1) test di planarità: dato un grafo, dire se è planare

(2) disegno planare: dato un grafo, trovare (ammesso che esista) un suo disegno planare

ovviamente risolvere (2) equivale ad aver trovato una risposta anche ad (1)

vorremmo poter risolvere questi problemi economizzando le risorse di calcolo

un po' di storia del test di planarità

- auslander e parter (1961): algoritmo quadratico
- hopcroft tarjan (1974): primo algoritmo a complessità lineare
- lempel, even e cederbaum (1966) + booth e luecker (1976): algoritmo complessivamente lineare, usa delle strutture dati dette pq-trees
- de fraisseix e rosenstiehl (mai pubblicato): costruttivo basato sulla visita in profondità del grafo (dfs)
- shih e hsu (1993): costruttivo, lineare, basato sulla dfs
- boyer e myrvold (1999): costruttivo, lineare basato sulla dfs

side effects

- primi lavori fondamentali sulla complessità computazionale asintotica
- strutture di dati
- vari problemi fondamentali risolti:
  - ✓ connettività
  - ✓ 2-connettività
  - ✓ 3-connettività

algoritmo di auslander e parter

problema:

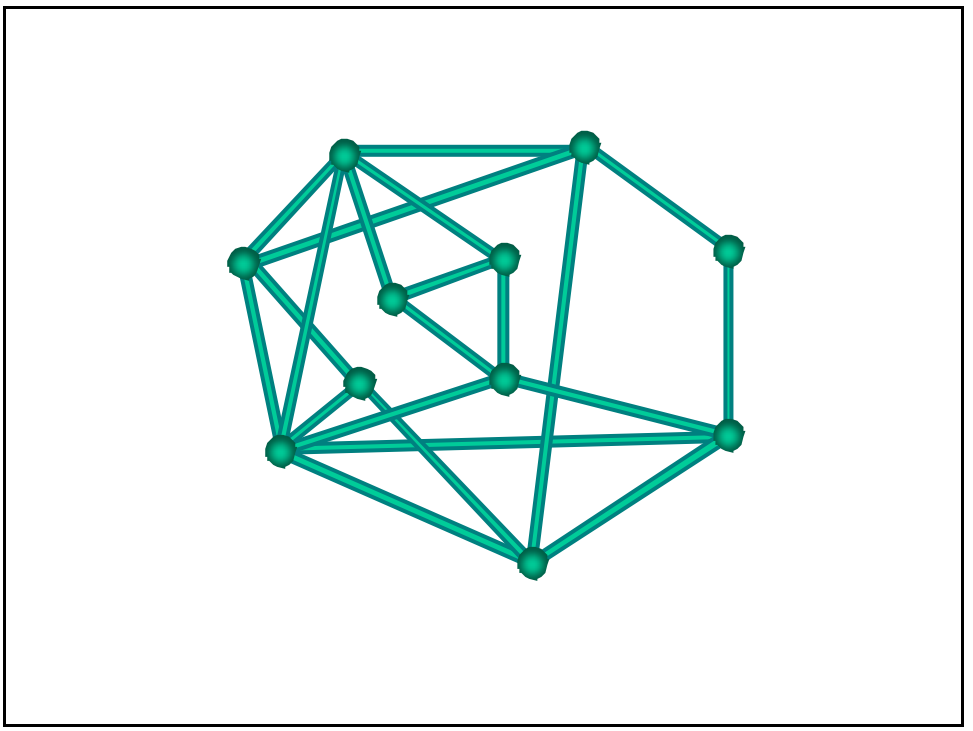
planarità

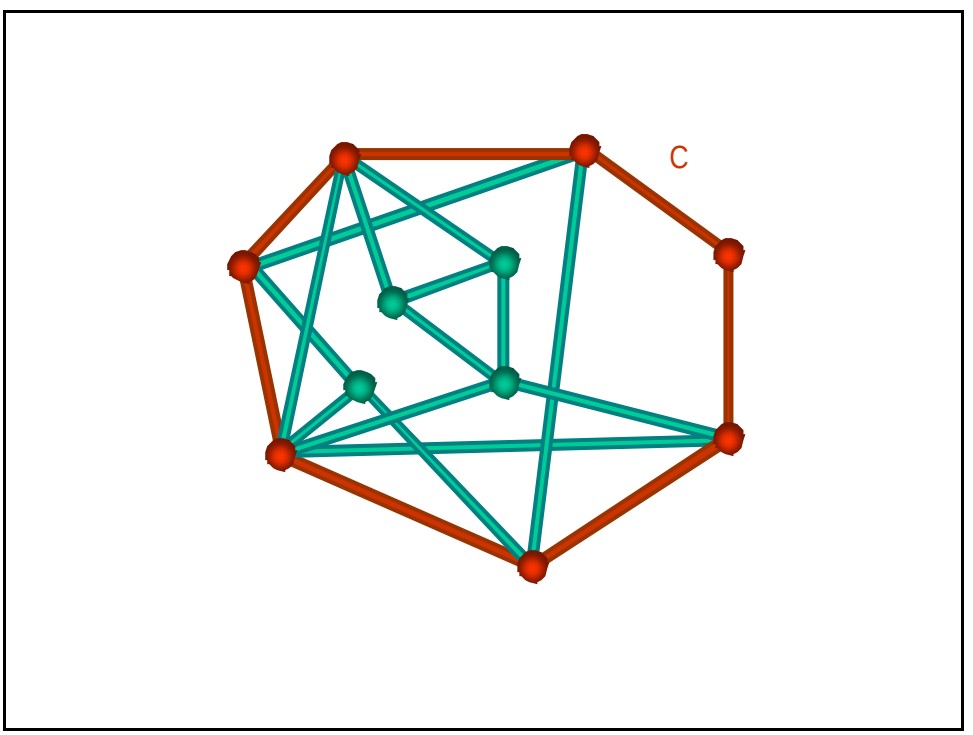
istanza:

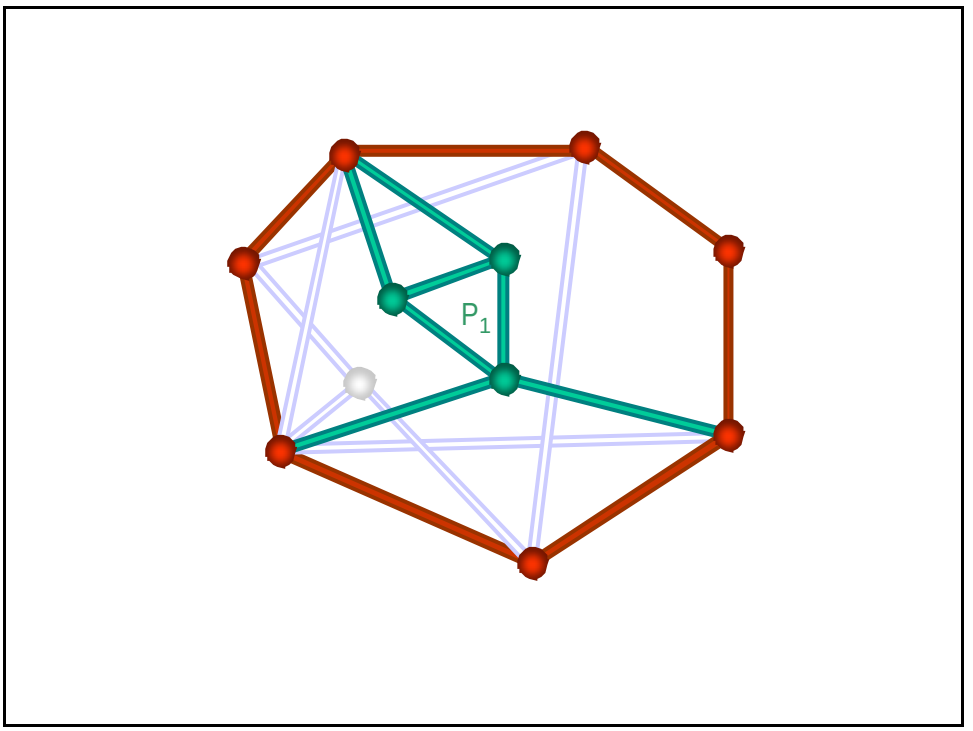
grafo G

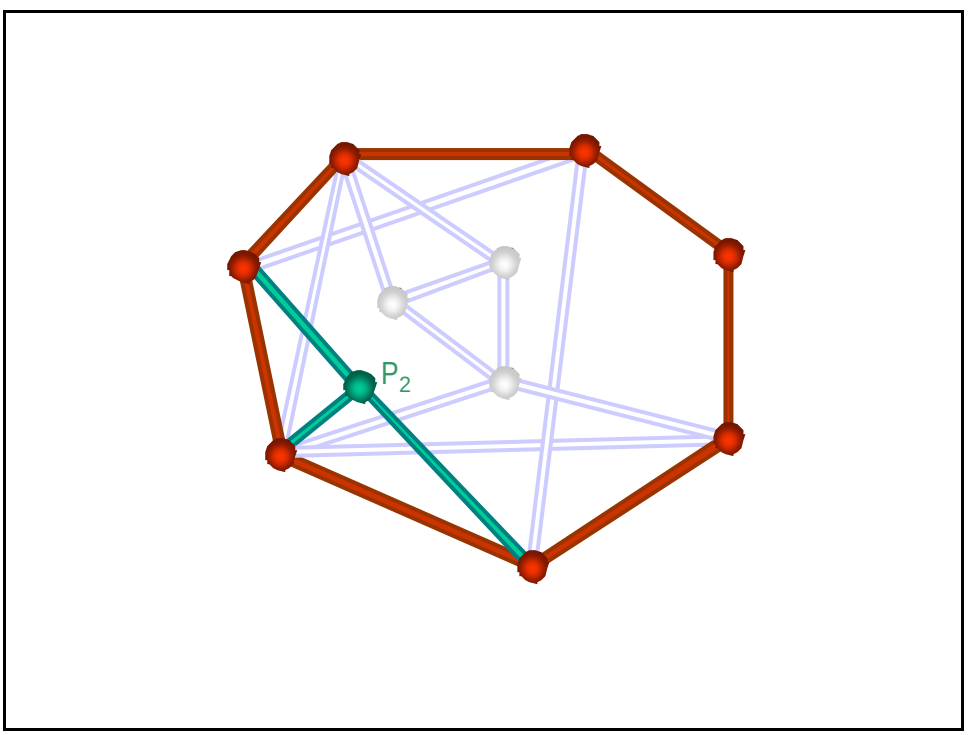
predicato:

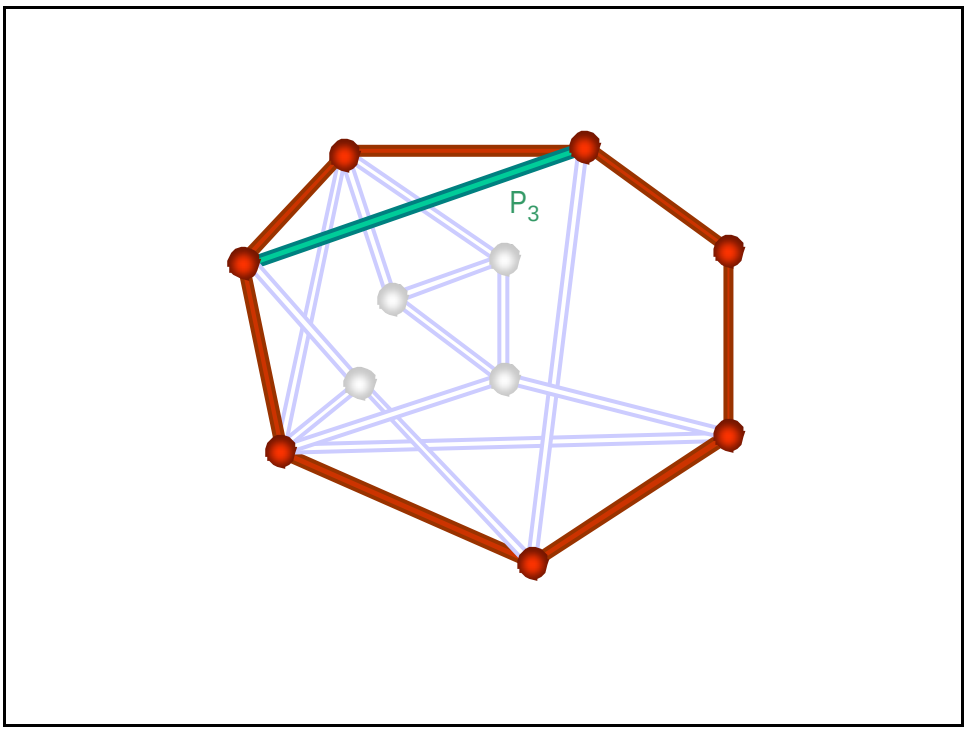
G è planare?

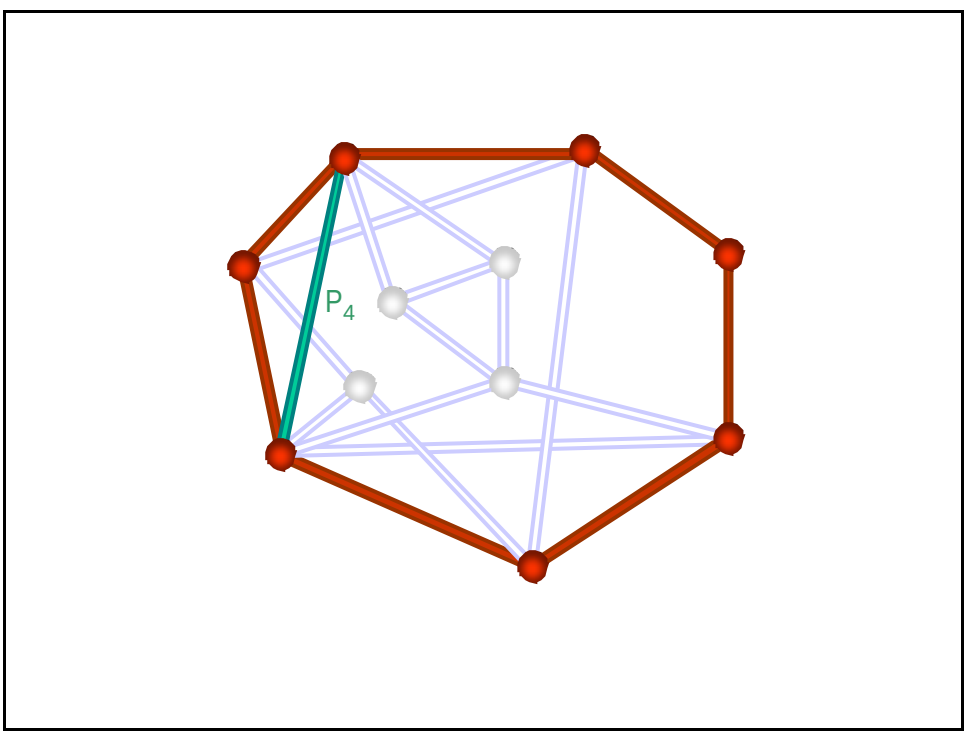


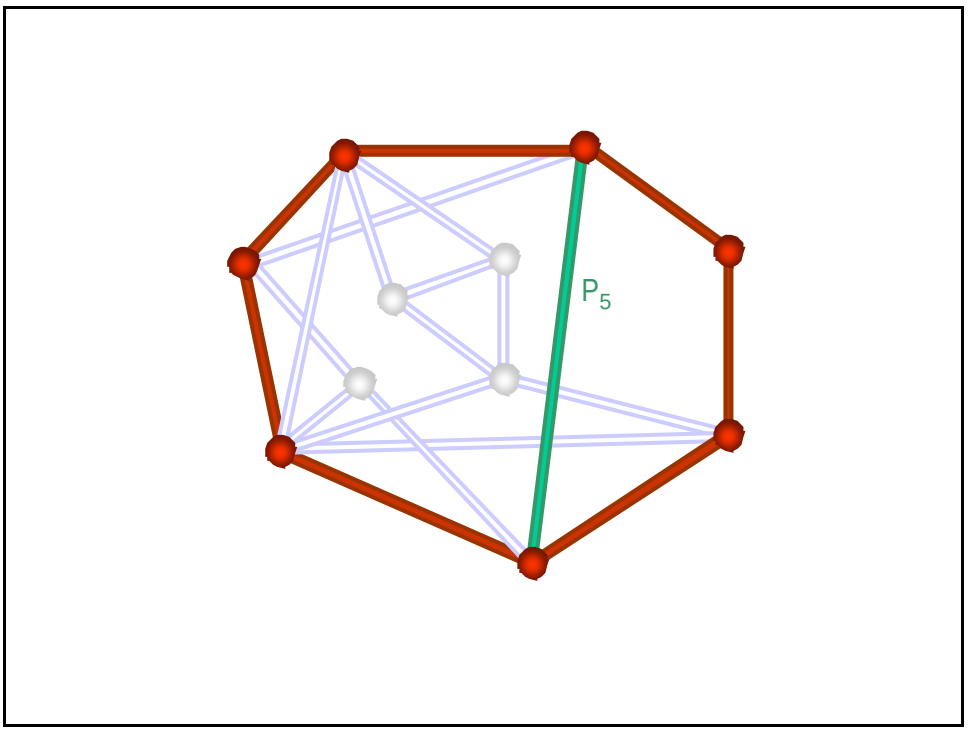

C

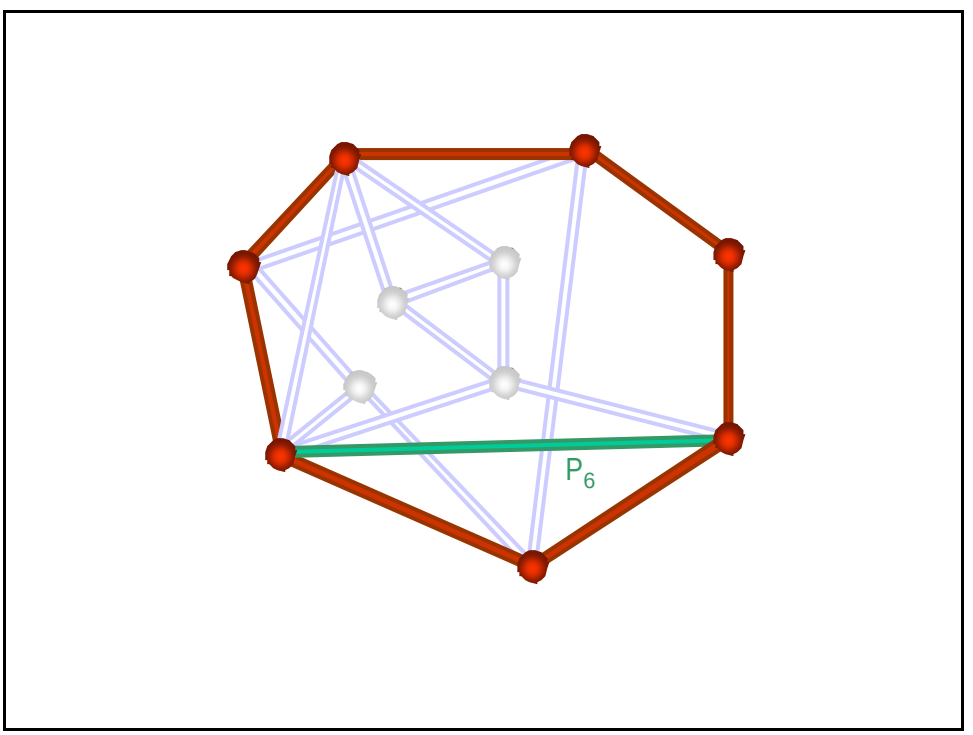

P1

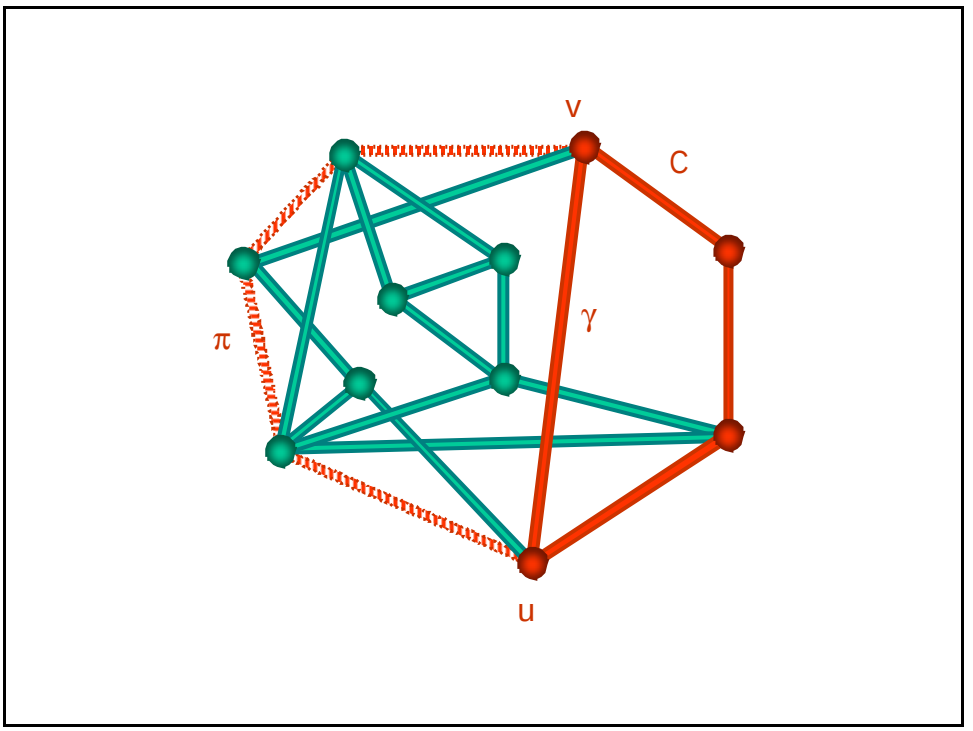

P2

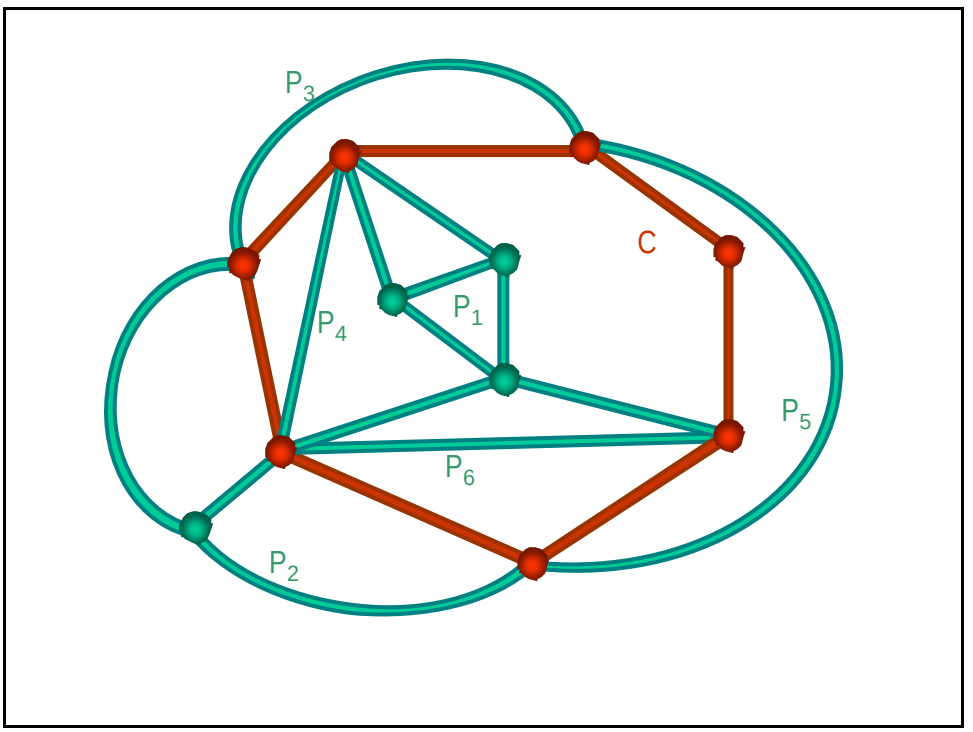

P_3

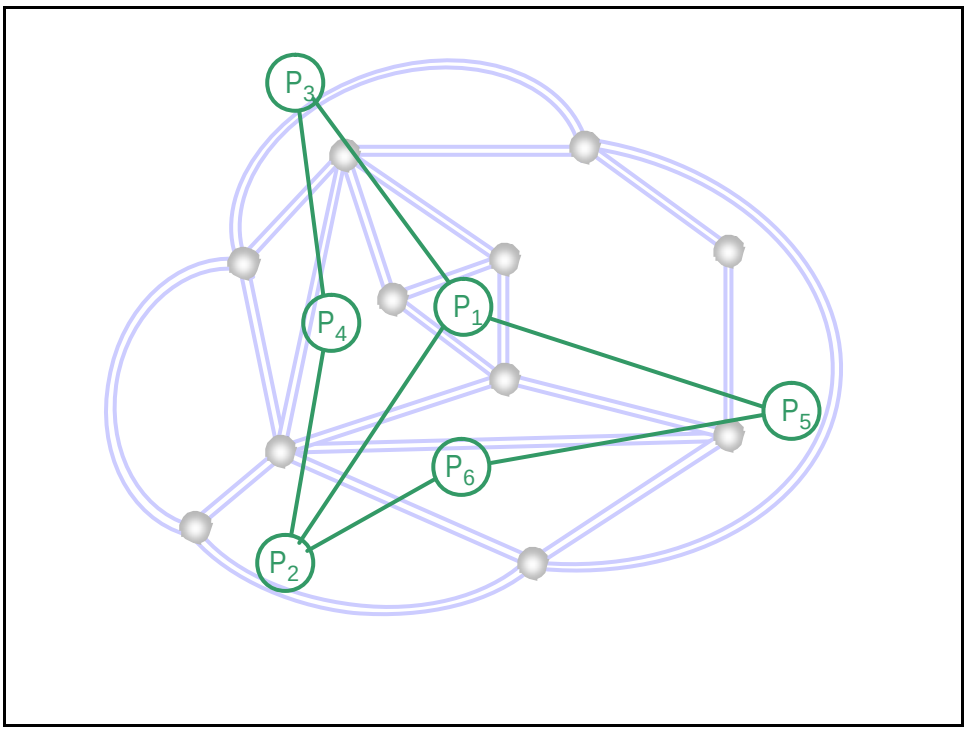

P_4

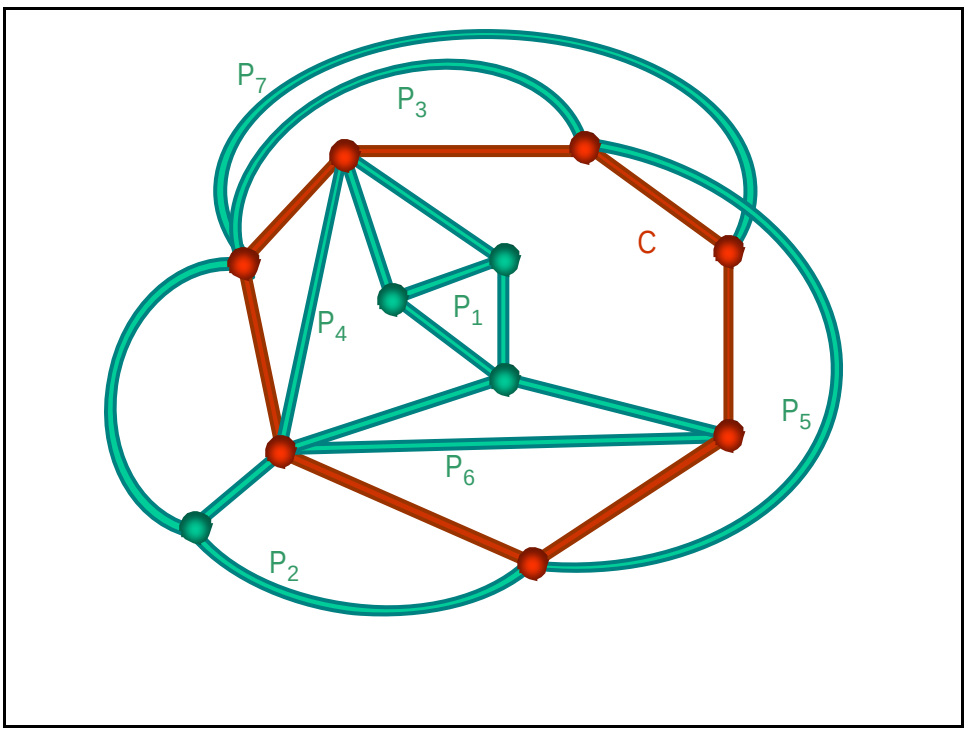

P_5

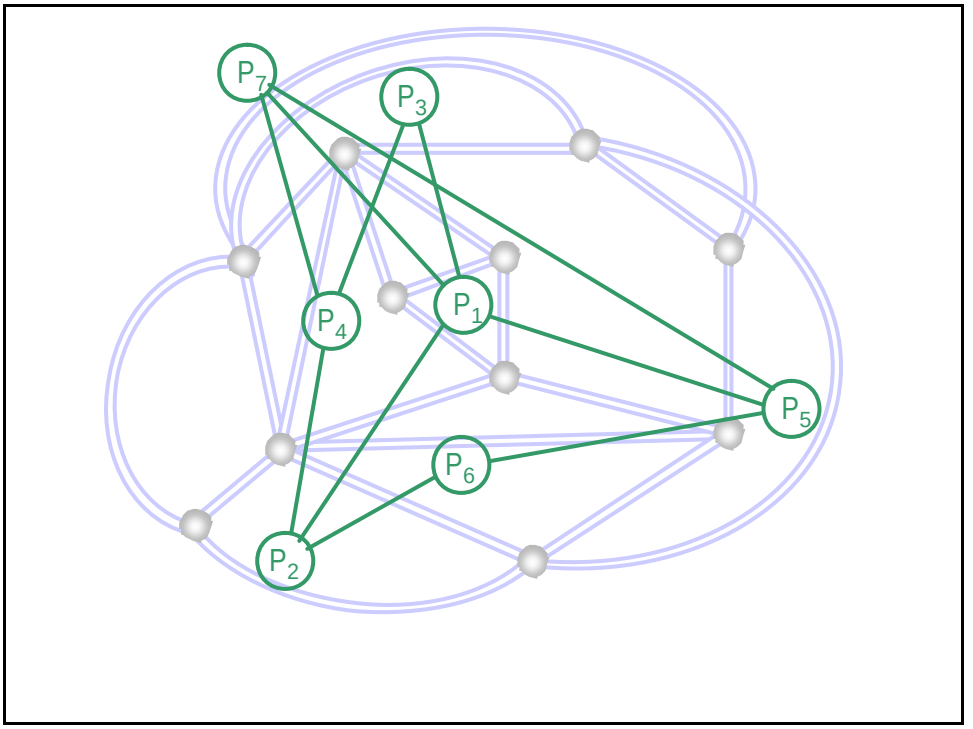

P_6

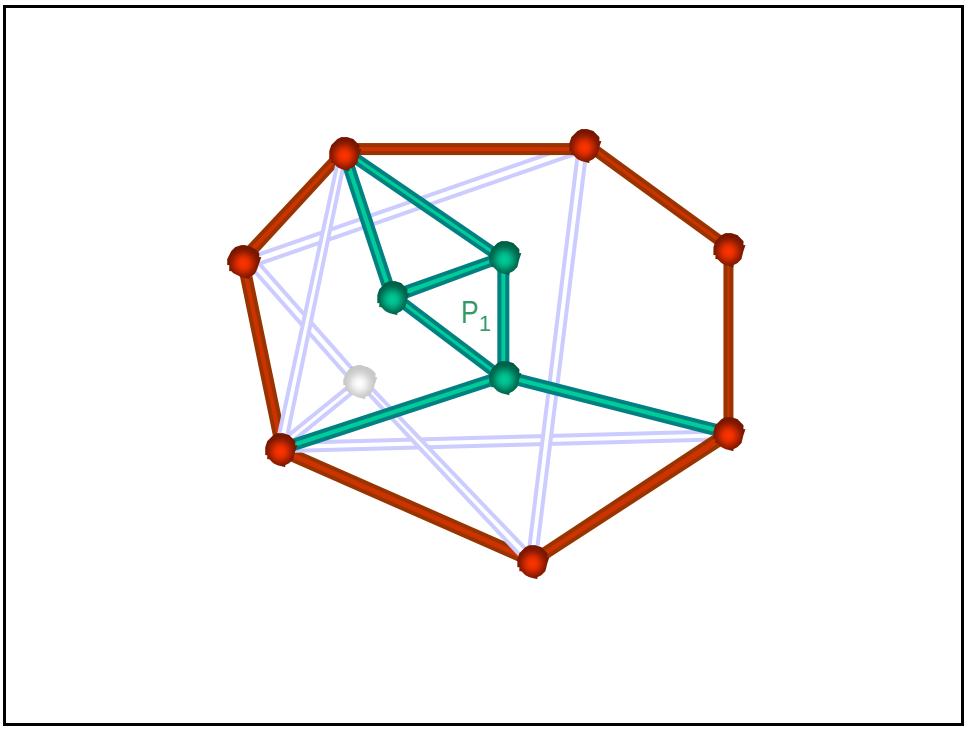

v
C
γ
π
u

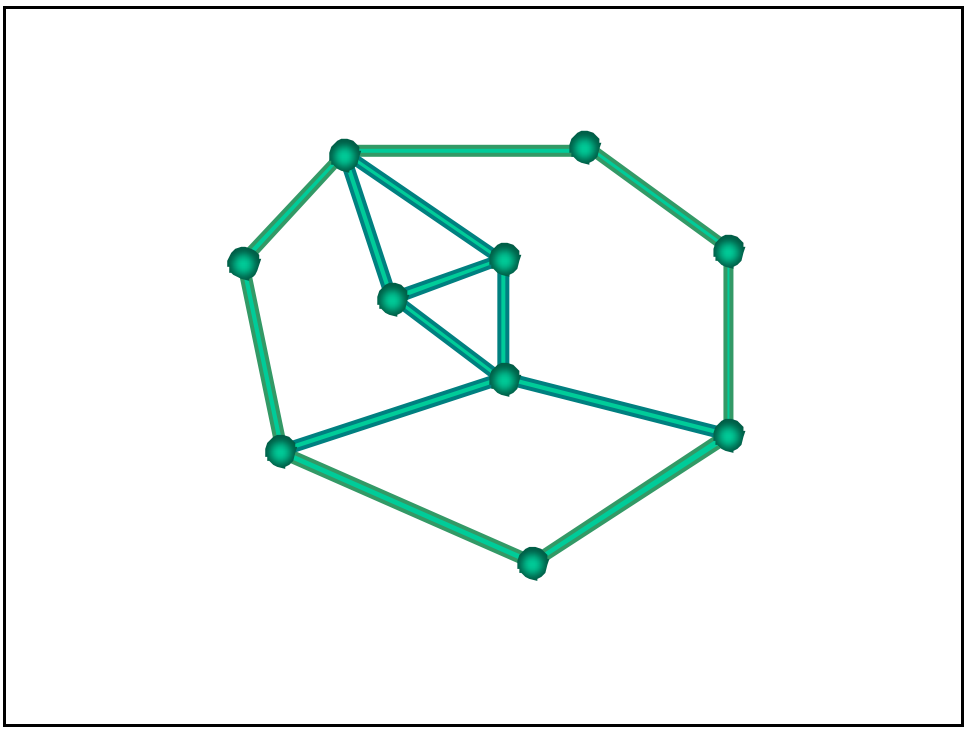

P3
C
P1
P4
P5
P6
P2

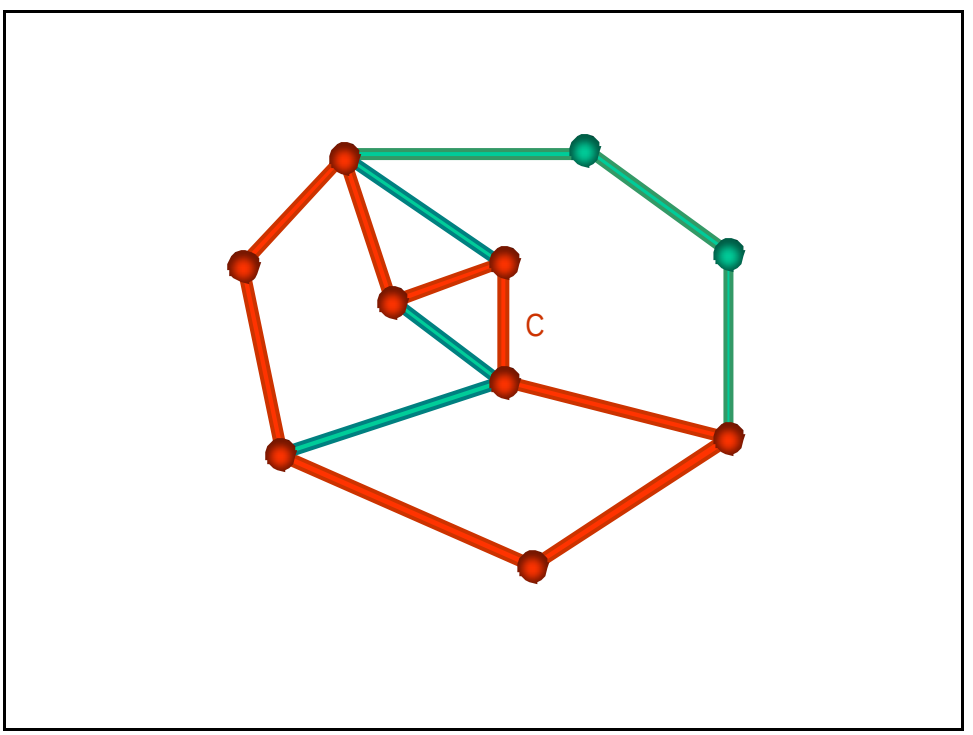

P3
P4
P1
P5
P6
P2


P7
P3
C
P4
P1
P5
P6
P2

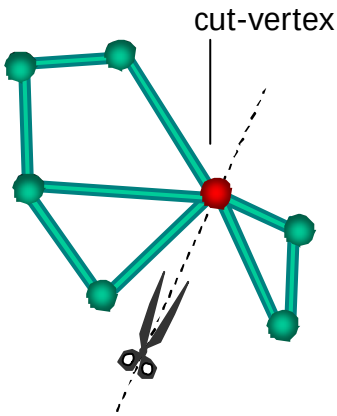

P7
P3
P4
P1
P5
P6
P2


P1

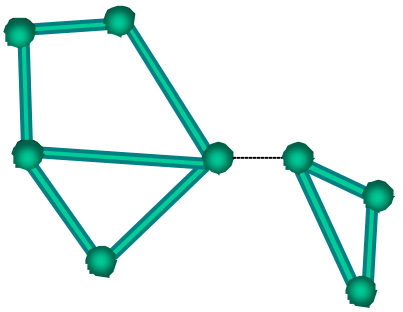



C

grafi biconnessi



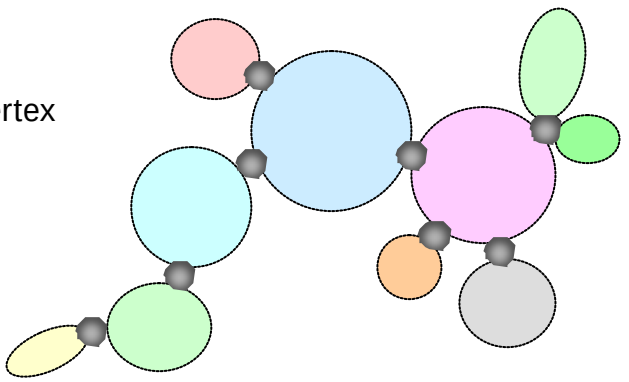
grafo non biconnesso
(ha un cut-vertex)



blocchi = sottografi biconnessi
(privi di cut-vertex)

block cut-vertex tree

block
cut-vertex
tree



determiniamo la planarità di ogni blocco (componente biconnessa) indipendentemente

algoritmo complessivo

1) decomporre un grafo arbitrario nelle sue componenti biconnesse (possiamo farlo in tempo lineare, come?)

2) planarizzare ogni componente biconnessa

algoritmo di boyer e myrvold

un metodo per trovare (se esiste) un disegno planare di un grafo in tempo lineare

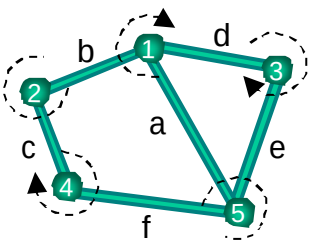
che cosa significa praticamente “tempo lineare”?

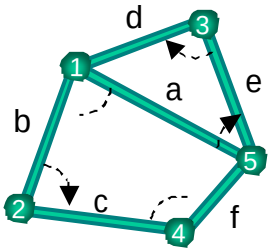
significa che posso compiere operazioni sugli oggetti in input (nodi ed archi) un numero limitato di volte (1,2,3,...,k)

intuitivamente: quando ho già considerato un oggetto non posso più tornare sui miei passi

ci basta un “embedding”

è facile trovare un disegno planare di un grafo una volta noto l’ordine circolare di incidenza degli archi sui nodi (detto “embedding”)





embedding

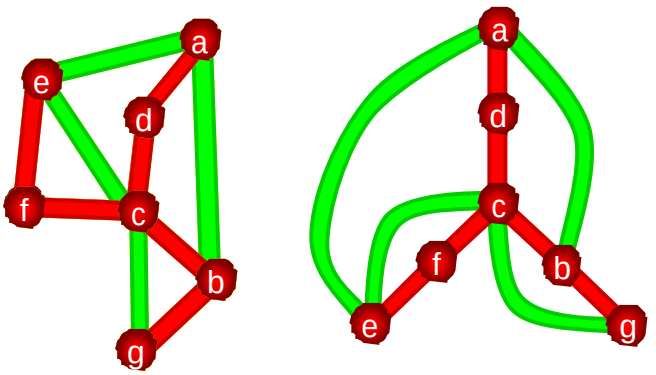
1: a, b, d
2: b, c
3: d, e
4: c, f
5: f, a, e

ingredienti

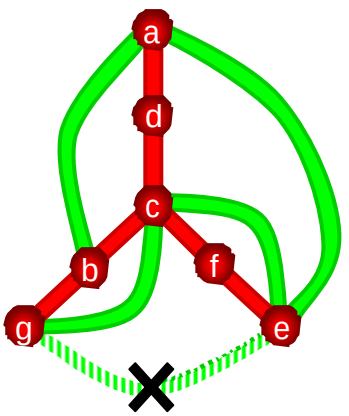
dfs = depth first search = visita in profondità del grafo ci aiuterà a dare una struttura al grafo

struttura dati efficiente per rappresentare componenti biconnesse (ed i loro embedding)

dfs - depth first search

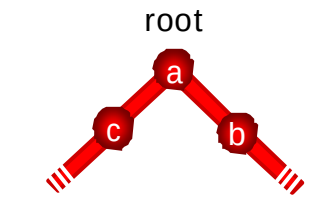


proprietà del dfs-tree

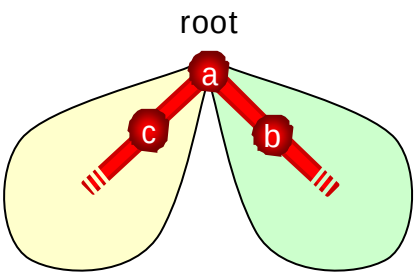


una fronda collega un nodo con un suo antenato

dfs-tree e connettività



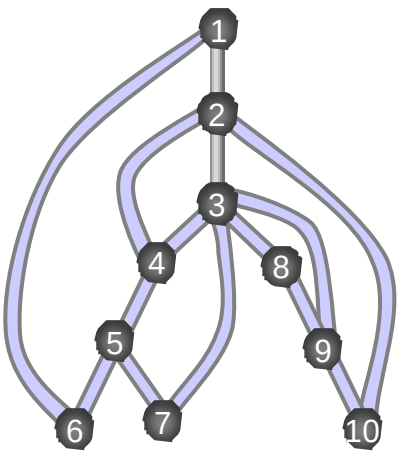
biforcazione alla radice



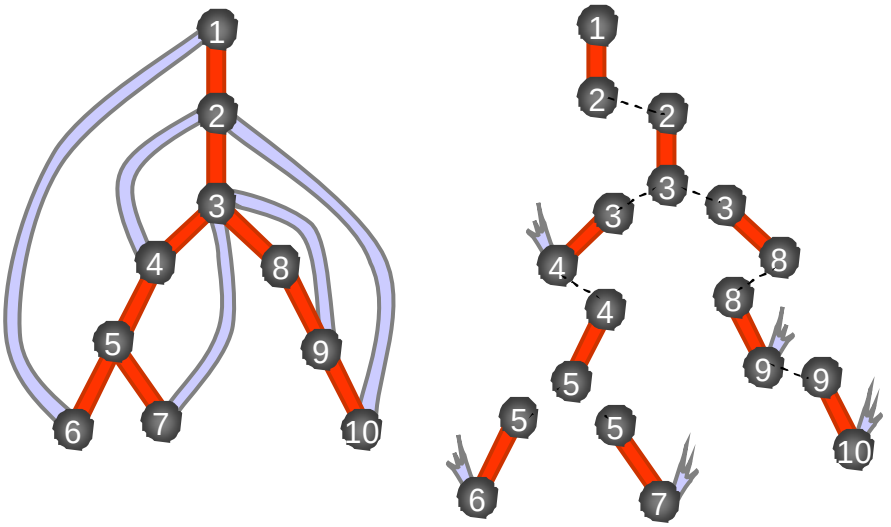
il nodo *a* è sicuramente un *cut-vertex*

il grafo non sarebbe biconnesso

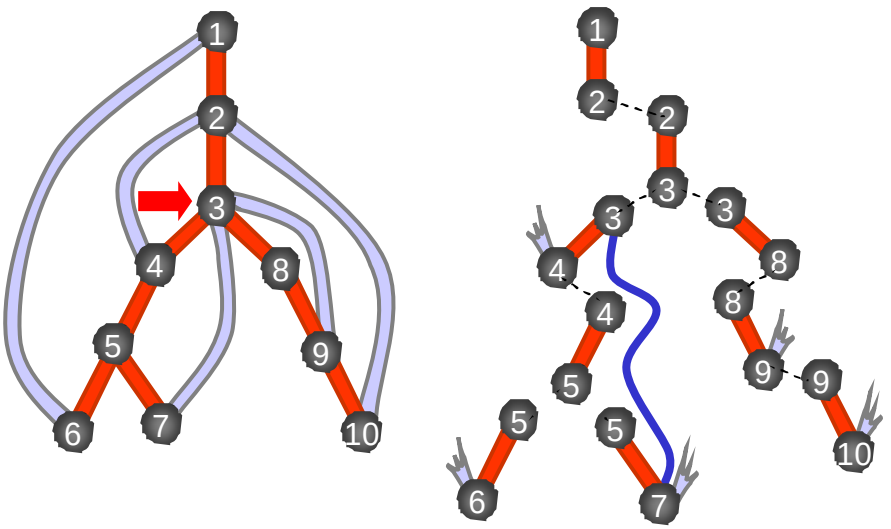
etichettatura dei nodi



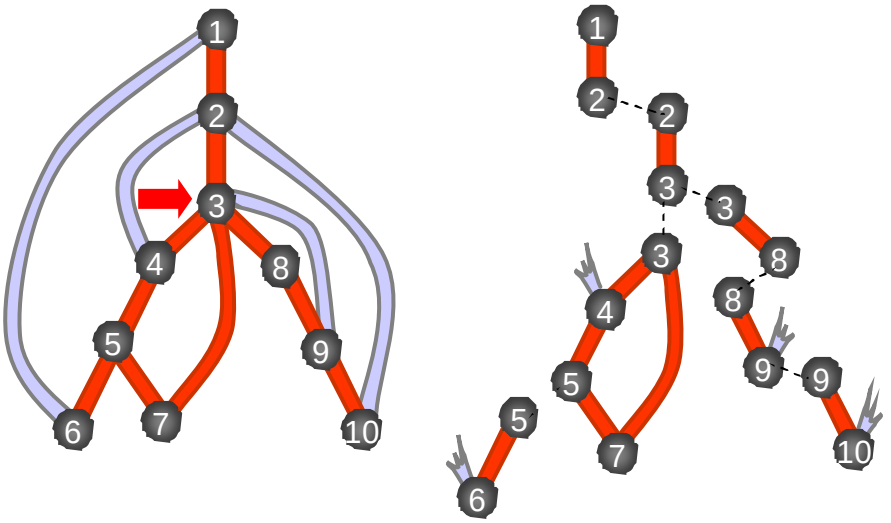
sketch dell'algoritmo (1)



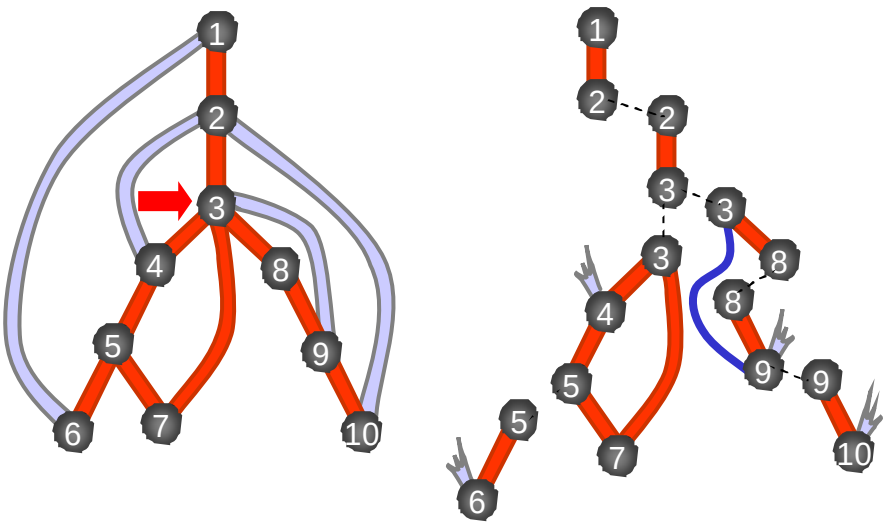
sketch dell'algoritmo (2)



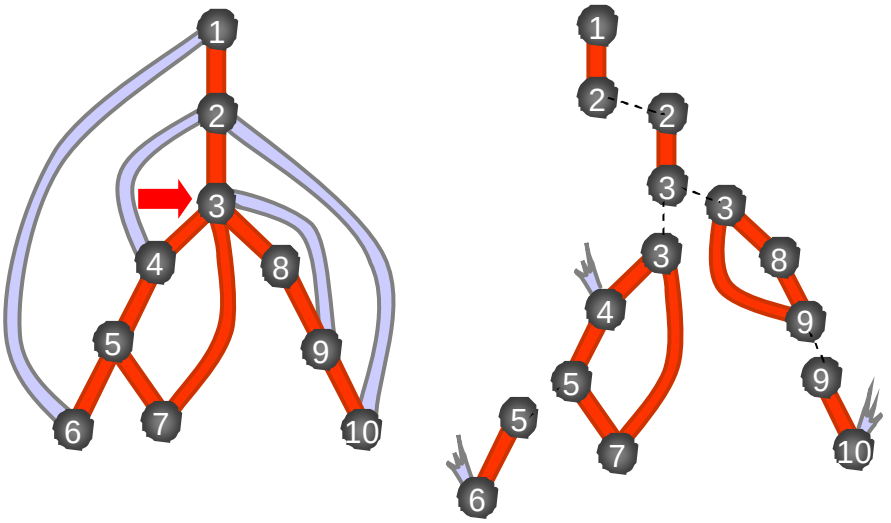
sketch dell'algoritmo (3)



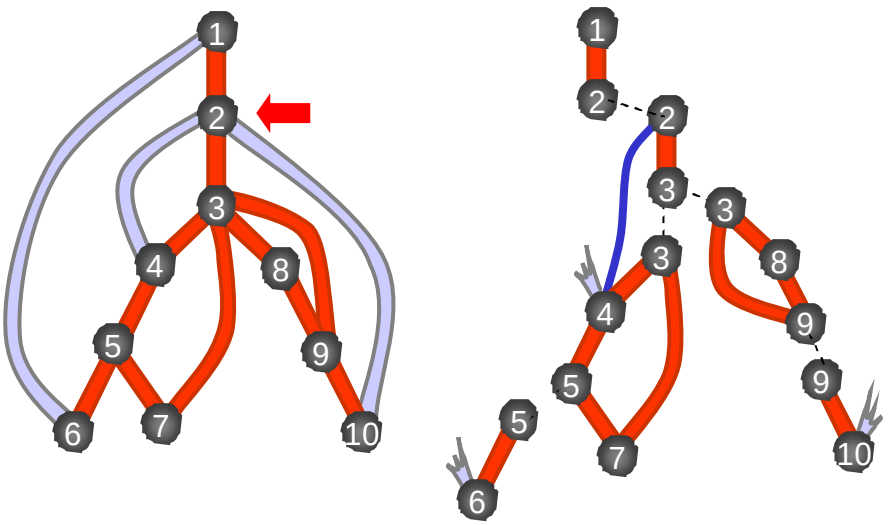
sketch dell'algoritmo (4)



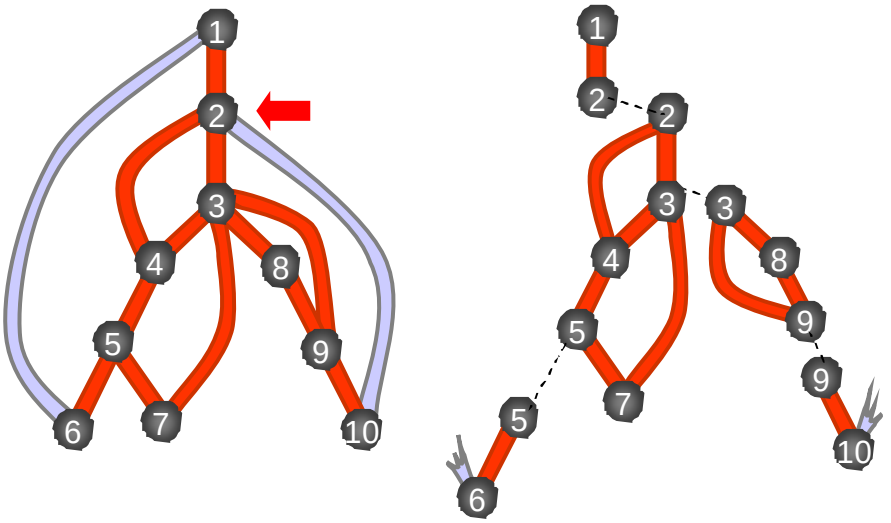
sketch dell'algoritmo (5)



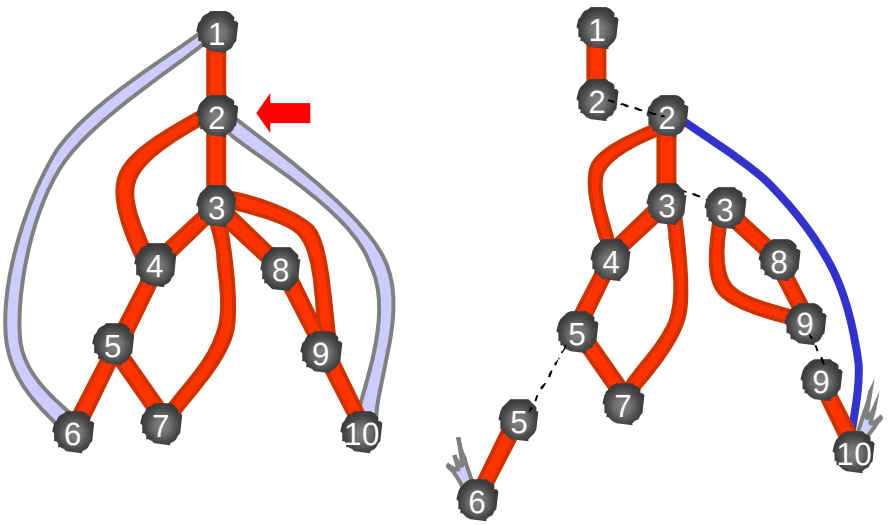
sketch dell'algoritmo (6)



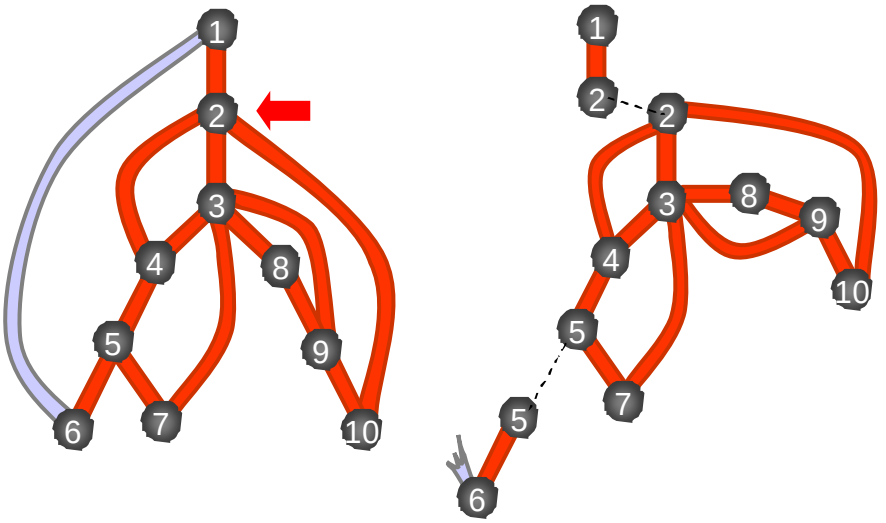
sketch dell'algoritmo (7)



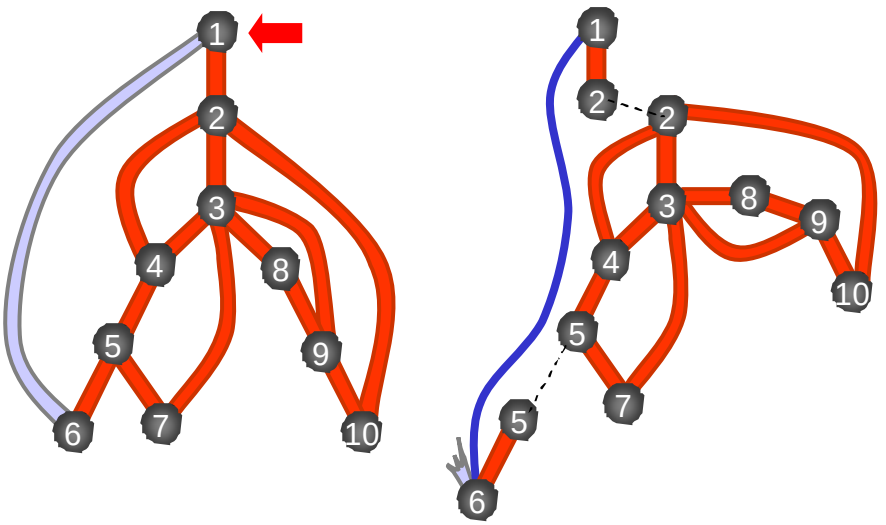
sketch dell'algoritmo (8)



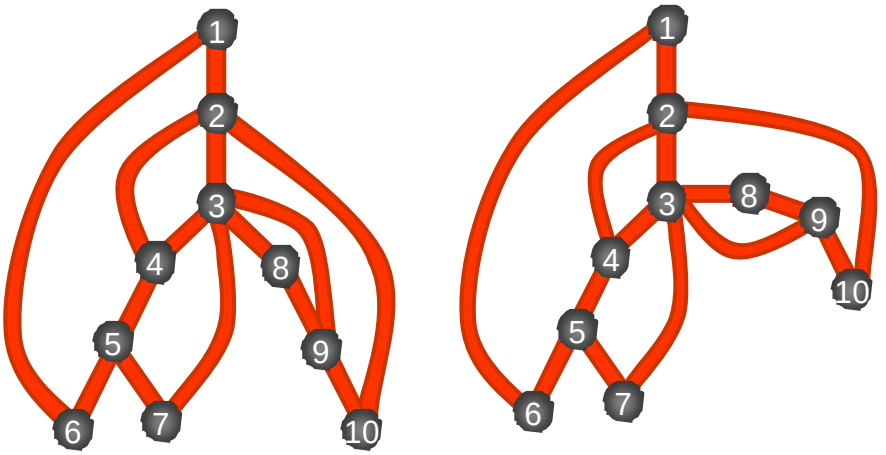
sketch dell'algoritmo (9)



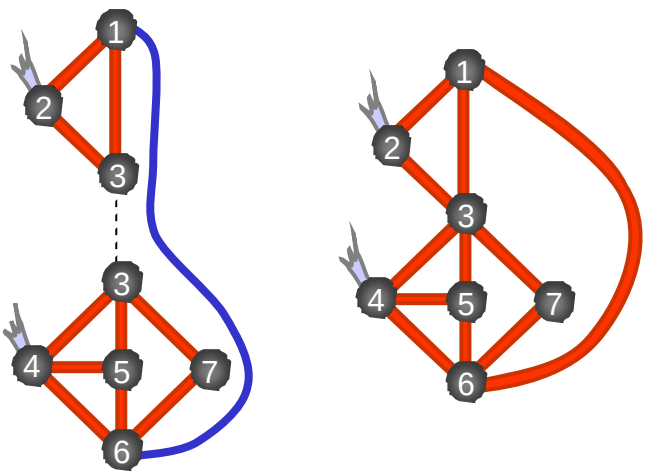
sketch dell'algoritmo (10)



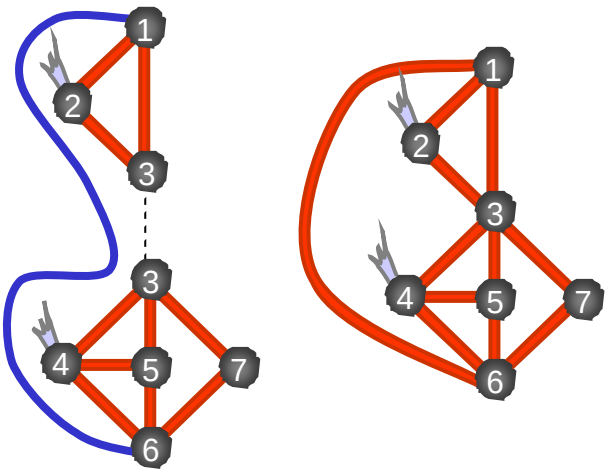
sketch dell'algoritmo (11)



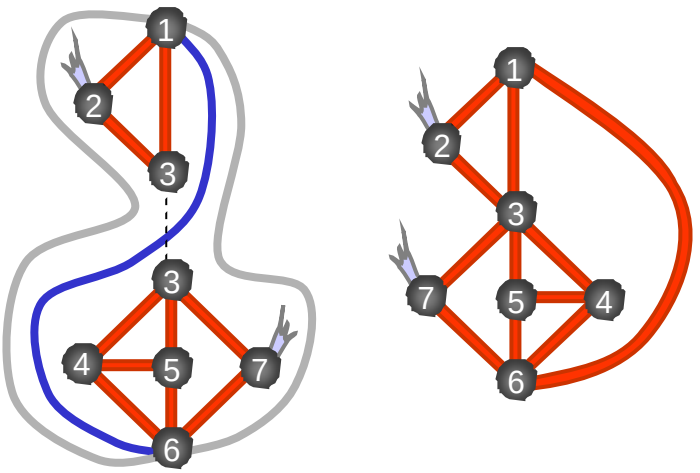
fusione delle componenti biconnesse



fusione non leggittima



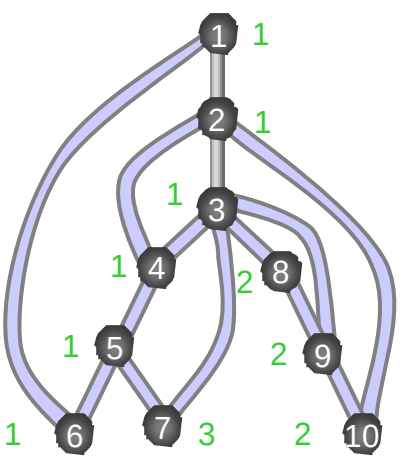
ribaltamento di una componente



problemi

- determinare (in tempo costante) se un nodo sia “attivo”, cioè debba rimanere sul bordo esterno della componente biconnessa (pena la perdita della planarità)
- trovare il modo di ribaltare una componente biconnessa in tempo costante

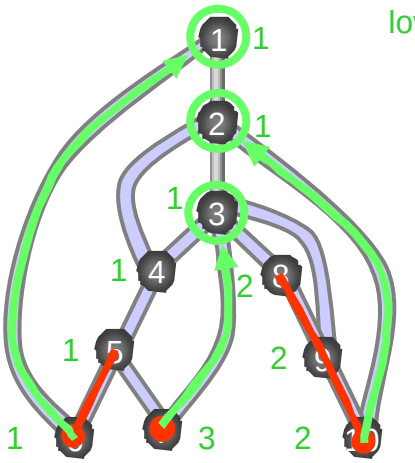
h(v) e lowpoint(v)



h(v) = minimo adiacente

lowpoint(v) =
min {h(v), lowpoint figli}

interpretazione intuitiva del lowpoint



lowpoint(v) =
min {h(v), lowpoint figli}

nodi attivi

supponiamo di aggiungere al disegno le fronde che entrano nel nodo w

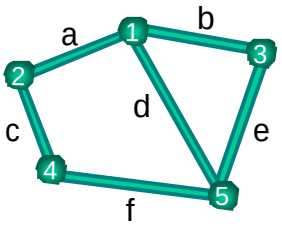
quali sono i nodi attivi?

1) i nodi che hanno una fronda diretta ad un antenato di w (come faccio a capirlo? non posso controllare tutte le fronde uscenti dal nodo)

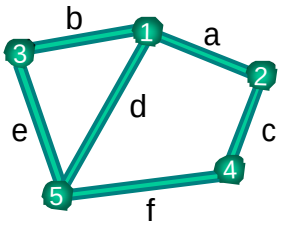
2) i nodi a cui si attaccherà una componente biconnessa, il cui primo nodo ha lowpoint minore di w (come faccio a capirlo in tempo costante?)

ribaltamento inefficiente

supponiamo di avere l'ordinamento circolare di incidenza degli archi sui nodi

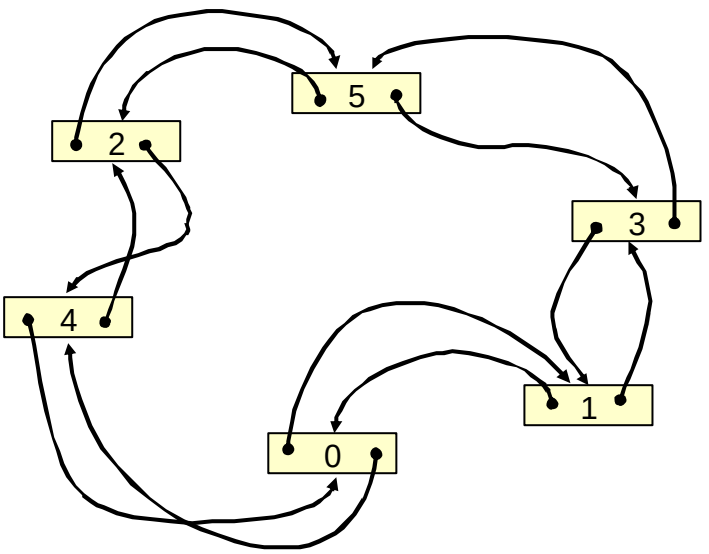


1: b, d, a
2: a, c
3: b, e
4: c, f
5: f, d, e

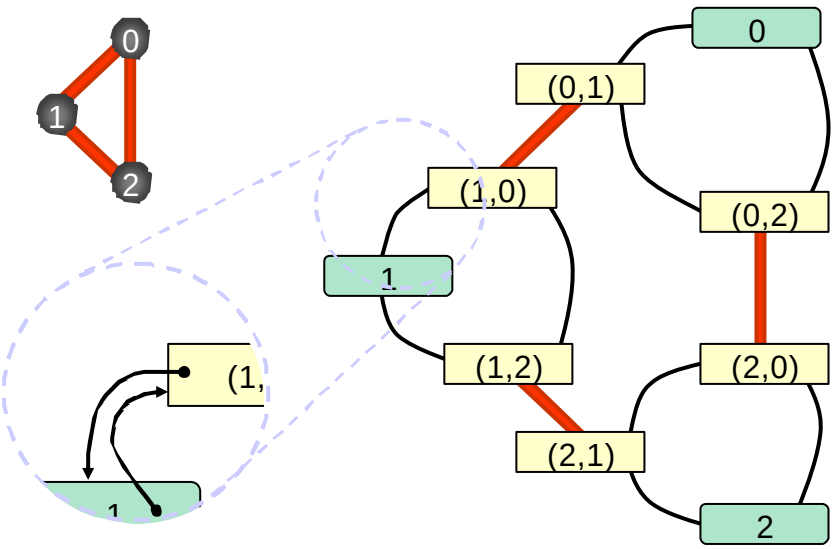


1: a, d, b
2: c, a
3: e, b
4: f, c
5: e, d, f

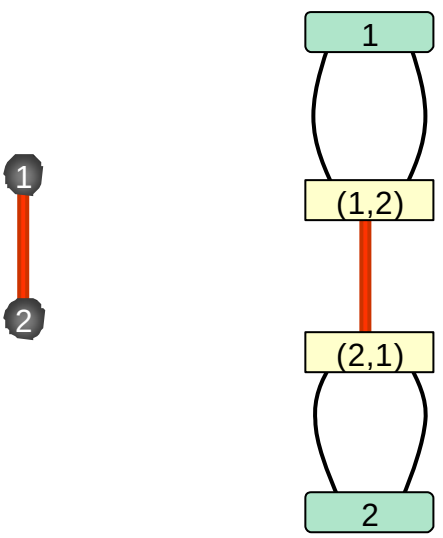
lista circolare



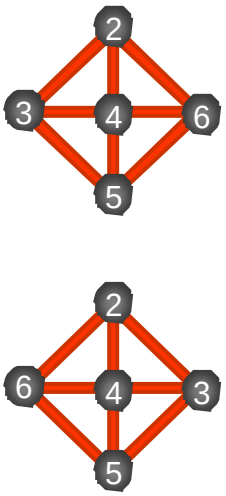
rappresentazione di una componente

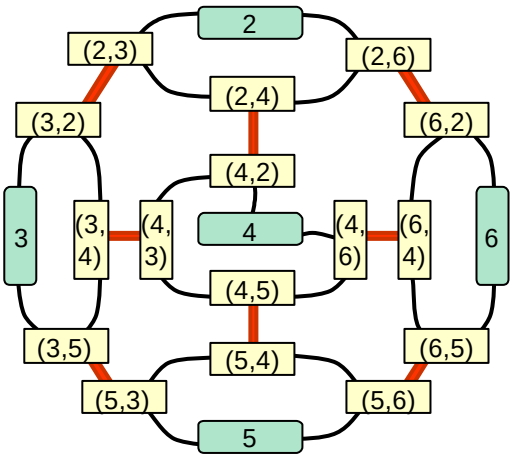


rappresentazione di una componente

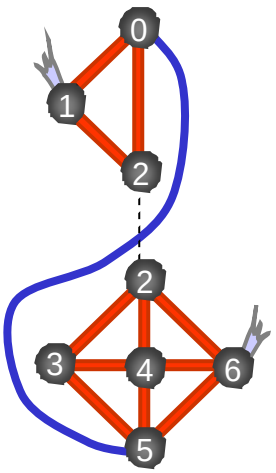


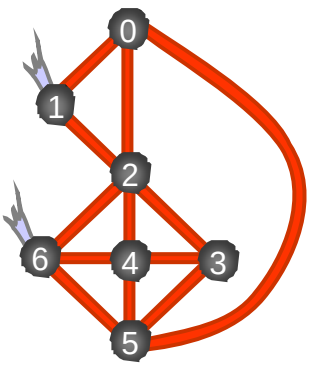
rappresentazione di una componente



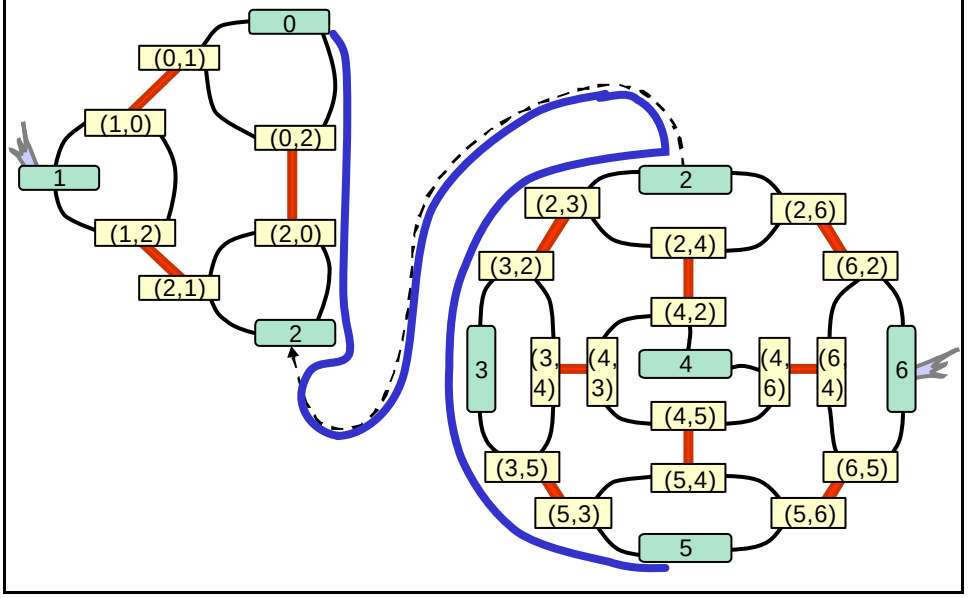


ribaltamento di una componente





ribaltamento di una componente



fusione di due componenti