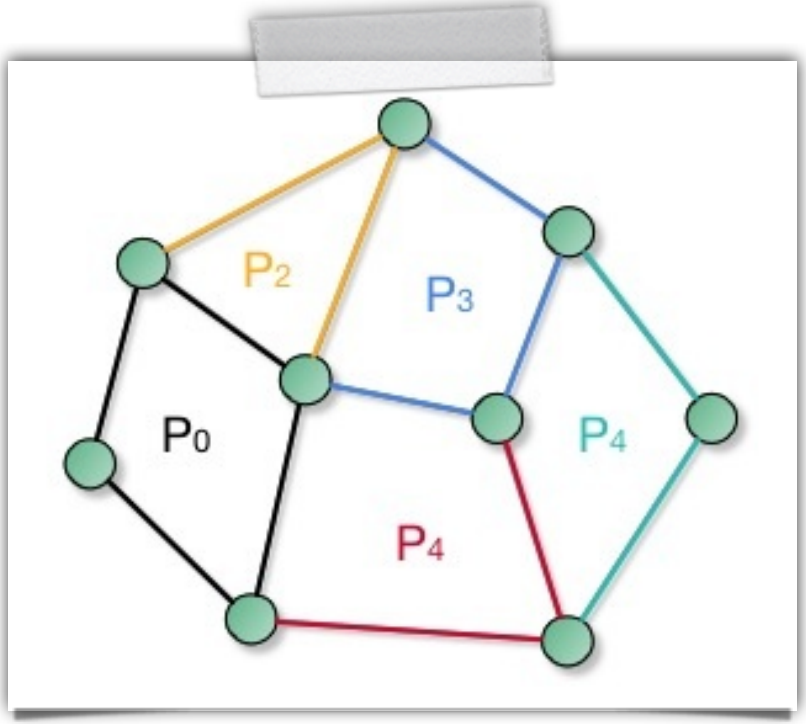
Corso di laurea magistrale in Informatica

Open Ear Decomposition



Corso di Algoritmi avanzati
2010-2011

Docente: Rossella Petreschi

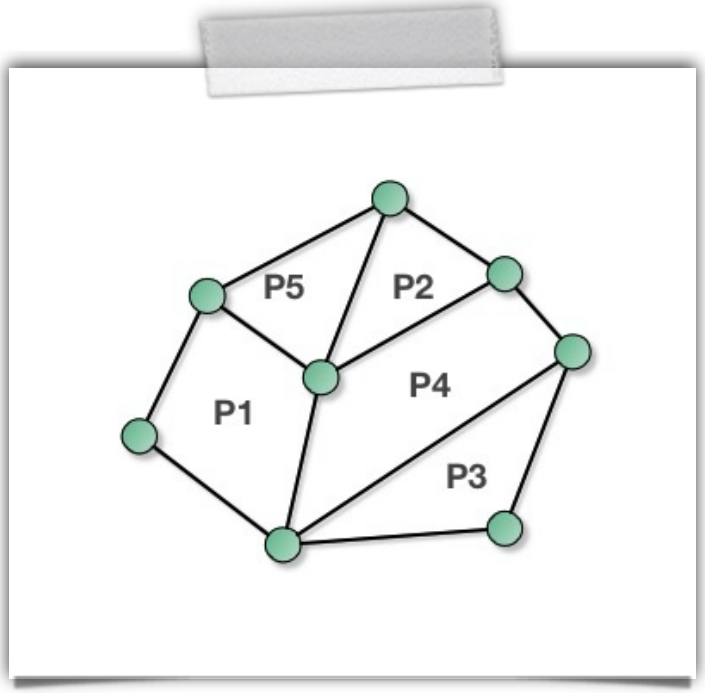
Relatore: **Marco Galletti**

Open Ear Decomposition

- Parte 1: Visite in un grafo
- Parte 2: Definizione di Ear Decomposition
- Parte 3: Un algoritmo efficente per la Ear
- Parte 4: Analisi dell'algoritmo

Open Ear Decomposition

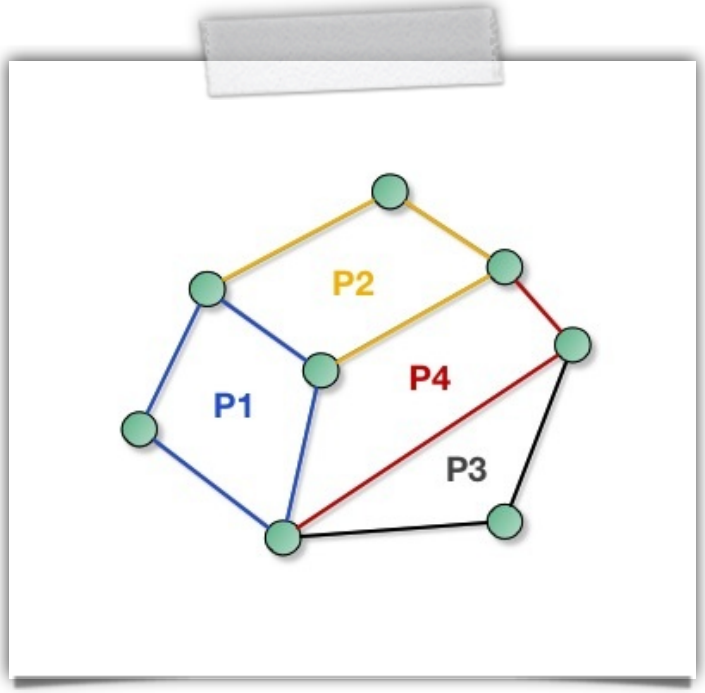
- **Parte 1**: Visite in un grafo
- Parte 2: Definizione di Ear Decomposition
- Parte 3: Un algoritmo efficente per la Ear
- Parte 4: Analisi dell'algoritmo



dfs, bfs, ... ear ?

Open Ear Decomposition

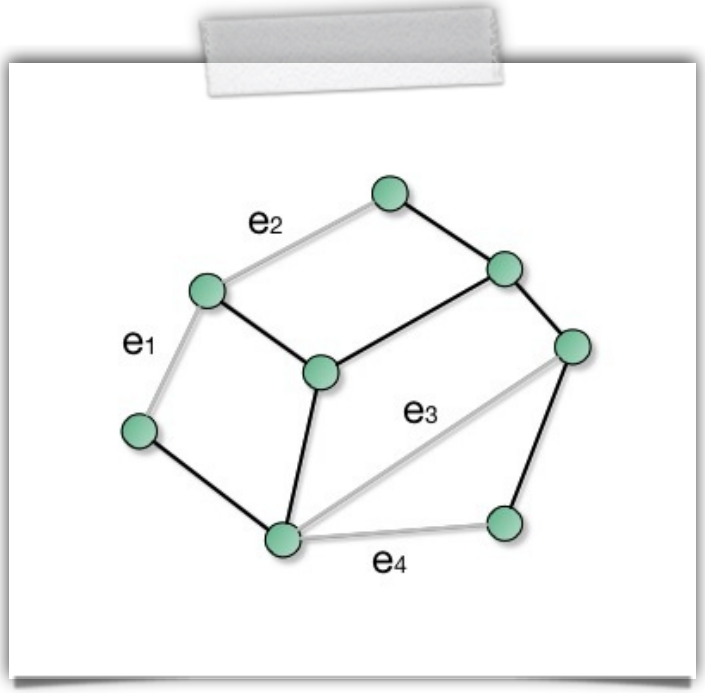
- Parte 1: Visite in un grafo
- **Parte 2:** Definizione di Ear Decomposition
- Parte 3: Un algoritmo efficente per la Ear
- Parte 4: Analisi dell'algoritmo



$E = P_1 \cup P_2 \cup \ldots \cup P_k$

Open Ear Decomposition

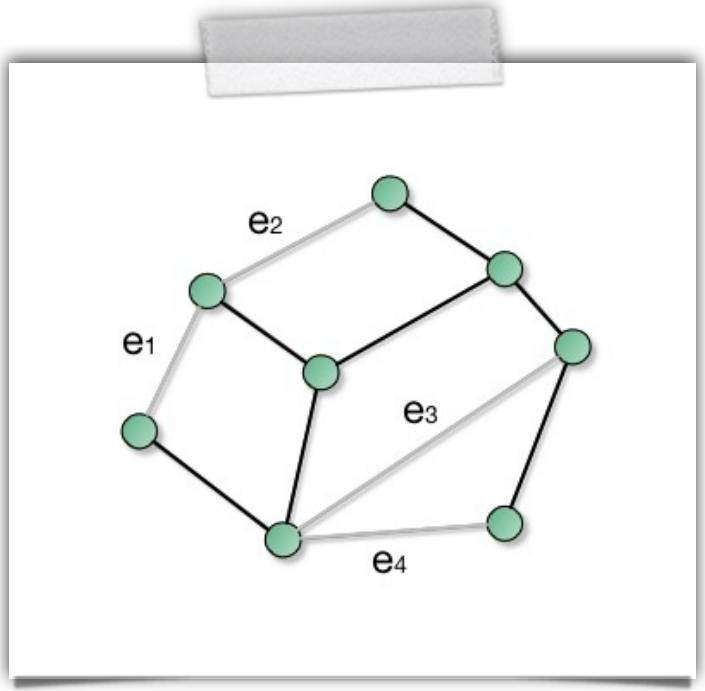
- Parte 1: Visite in un grafo
- Parte 2: Definizione di Ear Decomposition
- **Parte 3**: Un algoritmo efficente per la Ear
- Parte 4: Analisi dell'algoritmo



Vijaya Ramachandran

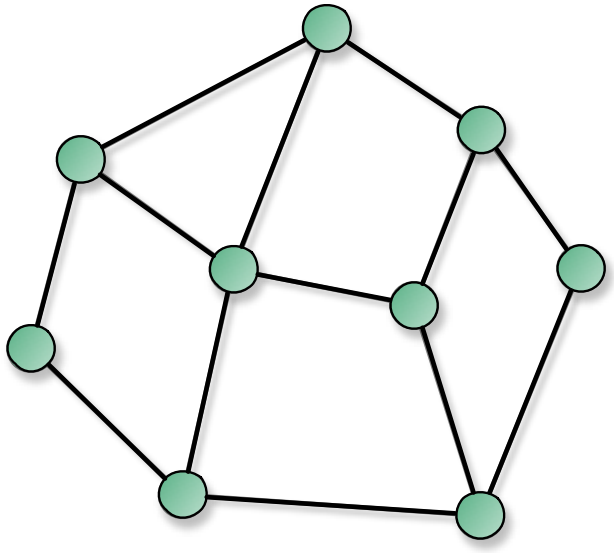
Open Ear Decomposition

- Parte 1: Visite in un grafo
- Parte 2: Definizione di Ear Decomposition
- Parte 3: Un algoritmo efficente per la Ear
- **Parte 4**: Analisi dell'algoritmo



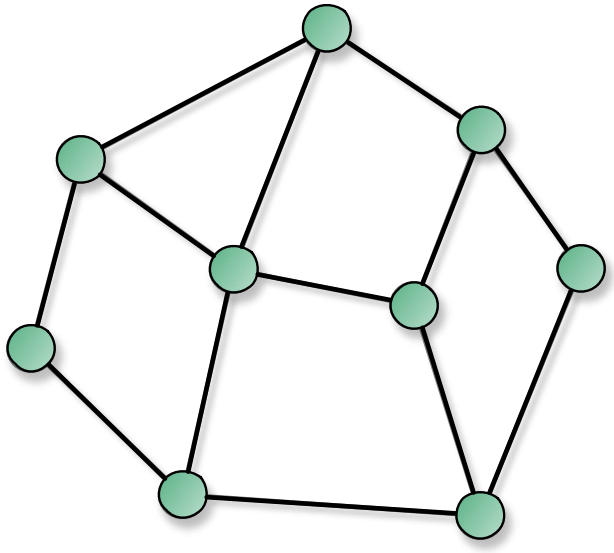
Di cosa parliamo ?

- Grafi non orientati
- Metodi di attraversamento non classici (bfs, dfs)
- Contesto di una macchina parallela: PRAM
- Perchè ?



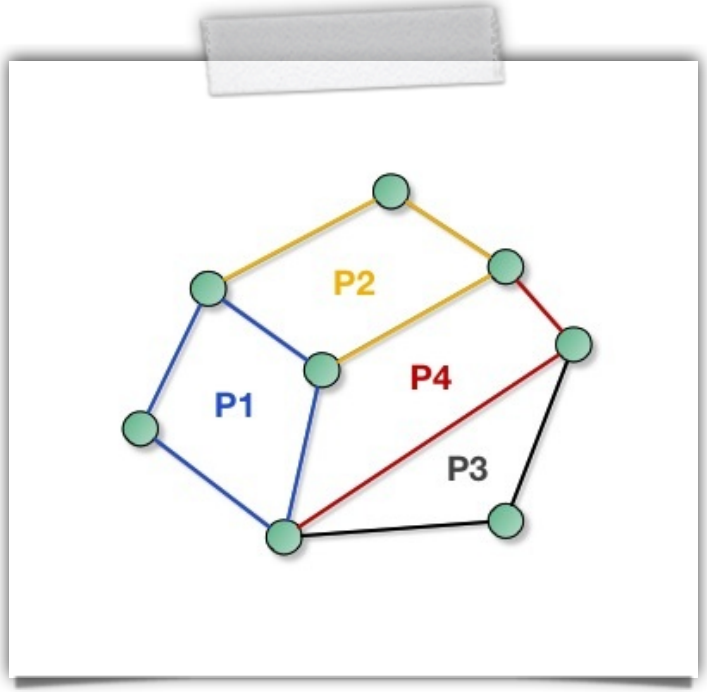
Di cosa parliamo ?

- Grafi non orientati
- Metodi di attraversamento non classici (bfs, dfs)
- Contesto di una macchina parallela: PRAM
- Perchè ?



Non esistono implementazioni efficenti per il parallelo delle classiche visite di bfs e dfs: studiamo delle soluzioni alternatve.

Definizione di Ear Decomposition



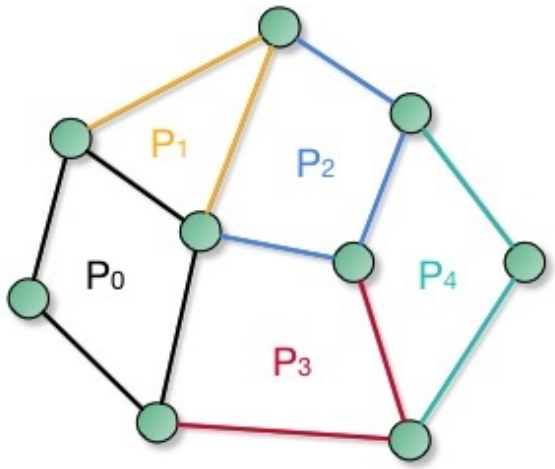
- Definizione Intuitiva
- Definizione Formale
- Caratterizzazione

Definizione Formale

- Sia dato un grafo non orientato $G=(V,E)$ con $|V|=n$ e $|E|=m$ e sia P_0 un ciclo semplice di G.

- Una ear decomposition $H=(P_0, P_1, \ldots, P_k)$ del grafo G è una partizione ordinata di $E = P_0 \cup P_1 \cup \ldots \cup P_k$ che inizi con P_0, tale che ciascun ear P_i sia un cammino semplice.

- H è tale che per ogni $i > 0$, P_i è un cammino semplice i cui vertici estremi appartengono a $H_{i-1} = P_0 \cup P_1 \cup \ldots \cup P_{i-1}$ e nessun altro vertice interno vi appartiene.

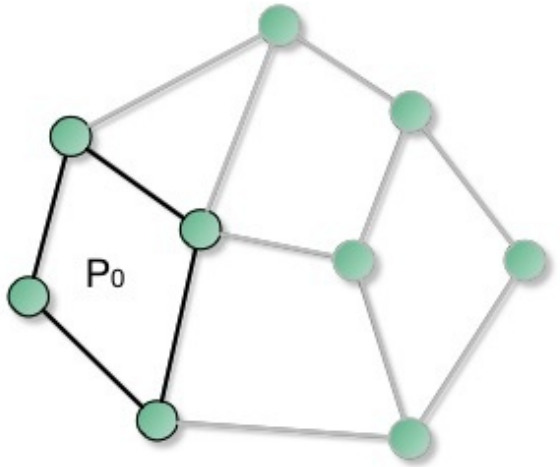


Definizione: Nucleo

- Sia dato un grafo non orientato **G=(V,E)** con |V|=n e |E|=m e sia **P_0** un ciclo semplice di G.

- Una ear decomposition $H=(P_0, P_1, \ldots, P_k)$ del grafo G è una partizione ordinata di $E = P_0 \cup P_1 \cup \ldots \cup P_k$ che inizi con P_0, tale che ciascun ear P_i sia un cammino semplice.

- H è tale che per ogni $i > 0$, P_i è un cammino semplice i cui vertici estremi appartengono a $H_{i-1} = P_0 \cup P_1 \cup \ldots \cup P_{i-1}$ e nessun altro vertice interno vi appartiene.



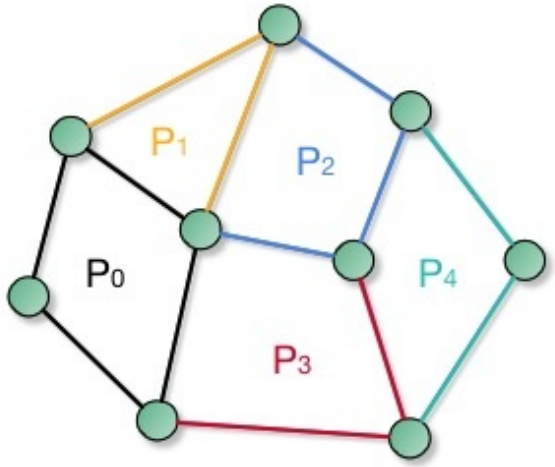
- Deve esistere un ciclo semplice **P_0** in G. Ovvero ...
- ... un grafo può non avere una ear decomposition.
- ... un grafo può averne più di una.

Definizione: Le Ear

- Sia dato un grafo non orientato G=(V,E) con |V|=n e |E|=m e sia P_0 un ciclo semplice di G.

- Una ear decomposition **$H=(P_0, P_1, \ldots, P_k)$** del grafo G è una partizione ordinata di $E = P_0 \cup P_1 \cup \ldots \cup P_k$ che inizi con P_0, tale che ciascun P_i sia un cammino semplice.

- H è tale che per ogni $i > 0$, P_i è un cammino semplice i cui vertici estremi appartengono a $H_{i-1} = P_0 \cup P_1 \cup \ldots \cup P_{i-1}$ e nessun altro vertice interno vi appartiene.



- Ciascun **P_i** viene chiamato Ear ed è un cammino semplice.

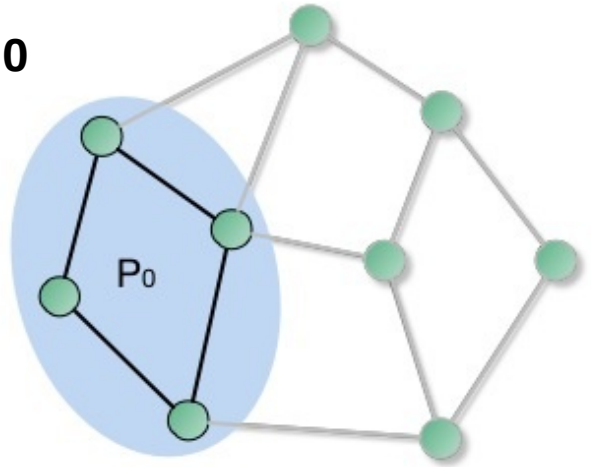
- Se tutti gli ear con i>0 NON sono cicli, la scomposizione è detta Aperta.

- Gli Ear devono essere ordinati e vengono costruiti in stretta relazione agli ear che li precedono.

Definizione Costruttiva

- Sia dato un grafo non orientato G=(V,E) con |V|=n e |E|=m e sia P_0 un ciclo semplice di G.
- Una ear decomposition H=(P_0, P_1, ... , P_k) del grafo G è una partizione ordinata di E = $P_0 \cup P_1 \cup \ldots \cup P_k$ che inizi con P_0, tale che ciascun ear P_i sia un cammino semplice.
- H è tale che per ogni i > 0, P_i è un cammino semplice i cui vertici estremi appartengono a $\mathbf{H_{i-1} = P_0 \cup P_1 \cup \ldots \cup P_{i-1}}$ e nessun altro vertice interno vi appartiene.

H_0

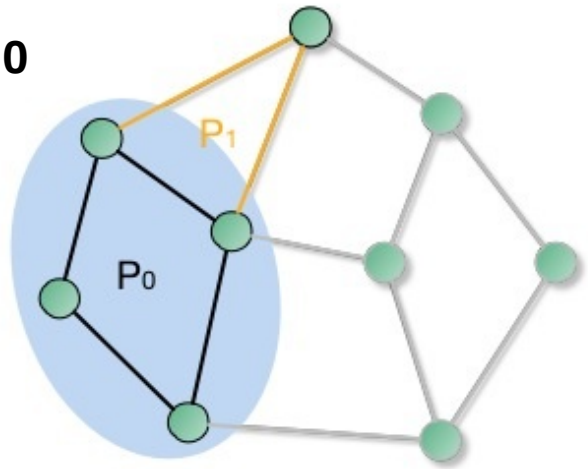


- Si parte da una partzione $\mathbf{H_0}$ con nucleo $\mathbf{P_0}$
- Ad ogni passo **i** si estende la partizione $\mathbf{H_{i-1}}$ aggiungendo un cammino semplice $\mathbf{P_i}$ non usato.
- Il cammino $\mathbf{P_i}$ aggiunto deve avere i vertici estremi nella partizione $\mathbf{H_{i-1}}$ già trovata.

Definizione Costruttiva

- Sia dato un grafo non orientato G=(V,E) con |V|=n e |E|=m e sia P_0 un ciclo semplice di G.
- Una ear decomposition H=(P_0, P_1, ... , P_k) del grafo G è una partizione ordinata di E = $P_0 \cup P_1 \cup \ldots \cup P_k$ che inizi con P_0, tale che ciascun ear P_i sia un cammino semplice.
- H è tale che per ogni i > 0, P_i è un cammino semplice i cui vertici estremi appartengono a $\mathbf{H_{i-1} = P_0 \cup P_1 \cup \ldots \cup P_{i-1}}$ e nessun altro vertice interno vi appartiene.

H_0



- Si parte da una partzione $\mathbf{H_0}$ con nucleo $\mathbf{P_0}$
- Ad ogni passo **i** si estende la partizione $\mathbf{H_{i-1}}$ aggiungendo un cammino semplice $\mathbf{P_i}$ non usato.
- Il cammino $\mathbf{P_i}$ aggiunto deve avere i vertici estremi nella partizione $\mathbf{H_{i-1}}$ già trovata.

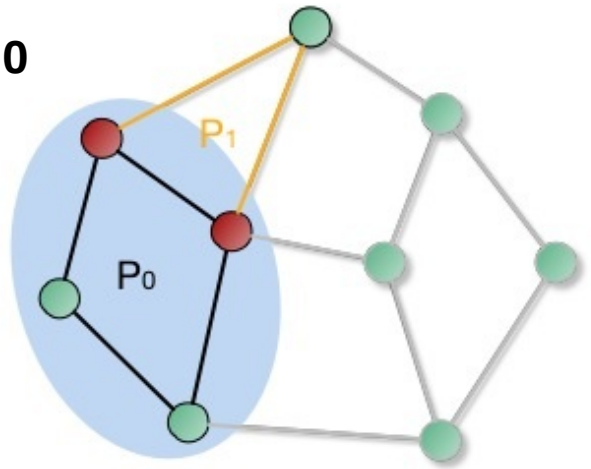
Definizione Costruttiva

- Sia dato un grafo non orientato $G=(V,E)$ con $|V|=n$ e $|E|=m$ e sia P_0 un ciclo semplice di G.

- Una ear decomposition $H=(P_0, P_1, \ldots, P_k)$ del grafo G è una partizione ordinata di $E = P_0 \cup P_1 \cup \ldots \cup P_k$ che inizi con P_0, tale che ciascun ear P_i sia un cammino semplice.

- H è tale che per ogni $i > 0$, P_i è un cammino semplice i cui vertici estremi appartengono a $\mathbf{H_{i-1} = P_0 \cup P_1 \cup \ldots \cup P_{i-1}}$ e nessun altro vertice interno vi appartiene.

H_0



- Si parte da una partzione $\mathbf{H_0}$ con nucleo $\mathbf{P_0}$

- Ad ogni passo **i** si estende la partizione $\mathbf{H_{i-1}}$ aggiungendo un cammino semplice $\mathbf{P_i}$ non usato.

- Il cammino $\mathbf{P_i}$ aggiunto deve avere i vertici estremi nella partizione $\mathbf{H_{i-1}}$ già trovata.

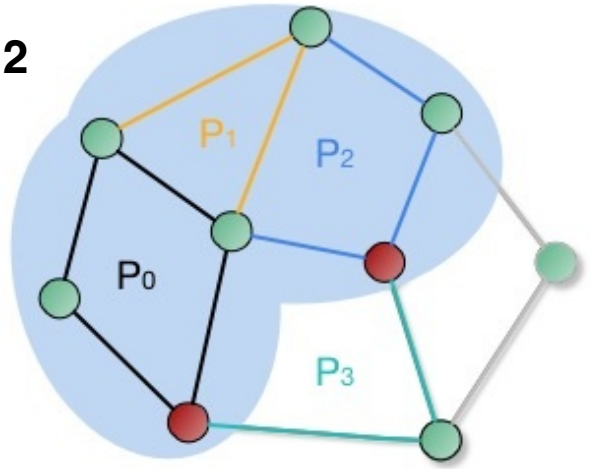
Definizione Costruttiva

- Sia dato un grafo non orientato G=(V,E) con |V|=n e |E|=m e sia P_0 un ciclo semplice di G.

- Una ear decomposition H=(P_0, P_1, ... , P_k) del grafo G è una partizione ordinata di E = $P_0 \cup P_1 \cup \ldots \cup P_k$ che inizi con P_0, tale che ciascun ear P_i sia un cammino semplice.

- H è tale che per ogni i > 0, P_i è un cammino semplice i cui vertici estremi appartengono a $\mathbf{H_{i-1} = P_0 \cup P_1 \cup \ldots \cup P_{i-1}}$ e nessun altro vertice interno vi appartiene.

H_2



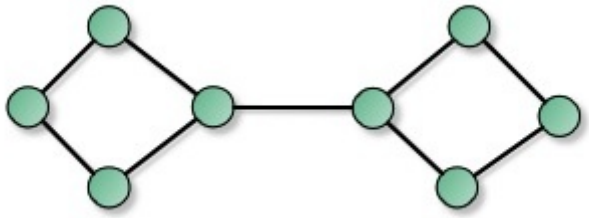
- Si parte da una partzione $\mathbf{H_0}$ con nucleo $\mathbf{P_0}$

- Ad ogni passo **i** si estende la partizione $\mathbf{H_{i-1}}$ aggiungendo un cammino semplice $\mathbf{P_i}$ non usato.

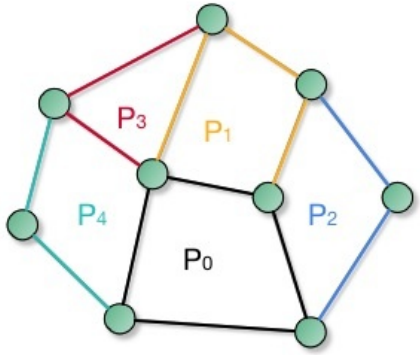
- Il cammino $\mathbf{P_i}$ aggiunto deve avere i vertici estremi nella partizione $\mathbf{H_{i-1}}$ già trovata.

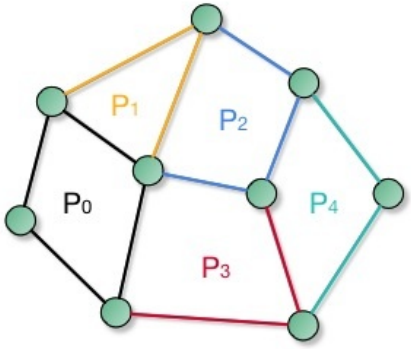
Caratterizzazione Ear

- In generale un Grafo può non ammettere una ear o open ear decomposition.



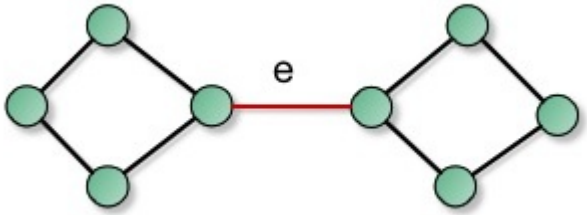
- Se un Grafo ammette una ear o open ear decomposition, ne può ammettere più di una.





Esistenza di una Ear decomposition

- Un grafo non orientato G=(V,E) ammette una Ear decomposition se e solo se è connesso e non ha ponti in E.

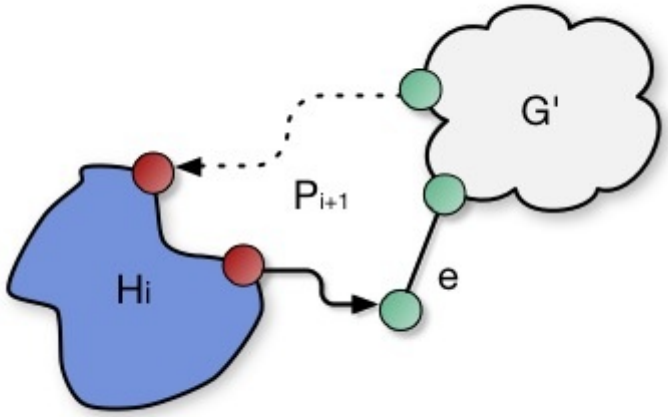


Ponte: Un arco e di G che se eliminato aumenta il numero di componenti connesse.

Dimostrazione per assurdo:

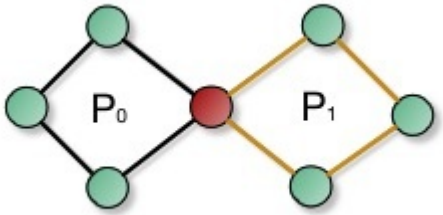
- Esiste $H=(P_0, P_1, \ldots, P_k)$ e un ponte **e** in P_{i+1}

- Nessun percorso **semplice** puó avere estremi contemporaneamente in $\mathbf{H_i}$ se passa per il ponte **e**



Esistenza di una Open Ear decomposition

- Un grafo non orientato G=(V,E) ammette una **Open** Ear decomposition se e solo se è biconnesso.

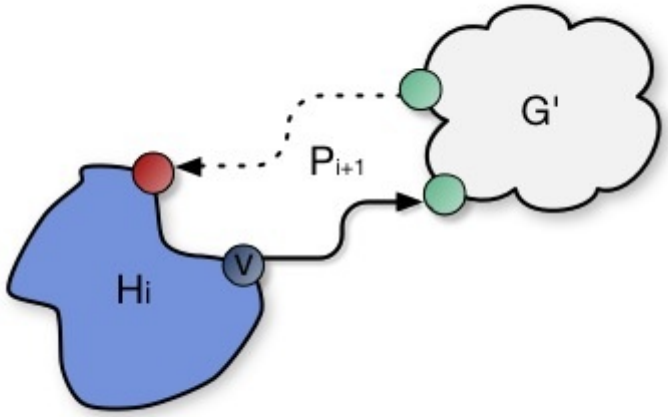


Biconnesso: Un grafo G che non ha punti di articolazione in V.

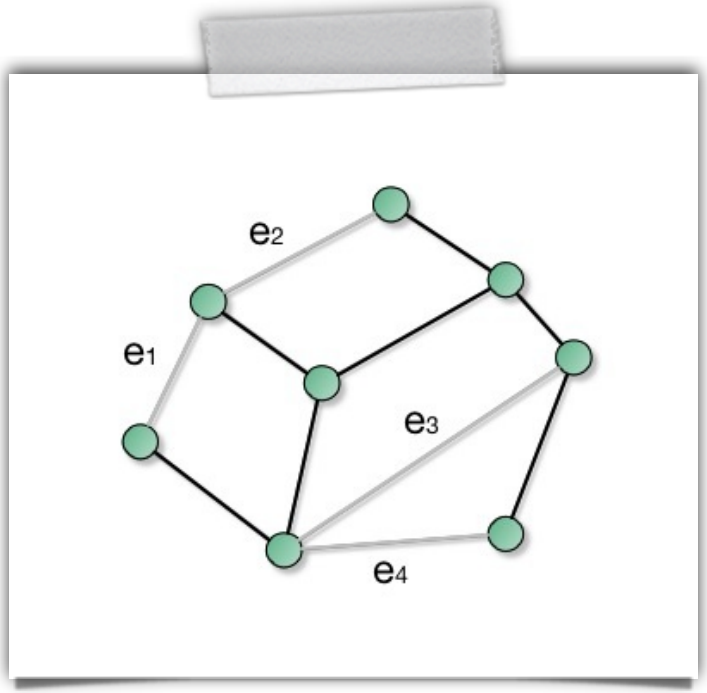
Dimostrazione per assurdo:

- Esiste $H=(P_0, P_1, \ldots, P_k)$ e un punto di articolazione **v** in P_{i+1}

- Nessun percorso **semplice** puó avere estremi **distinti** contemporaneamente in H_i

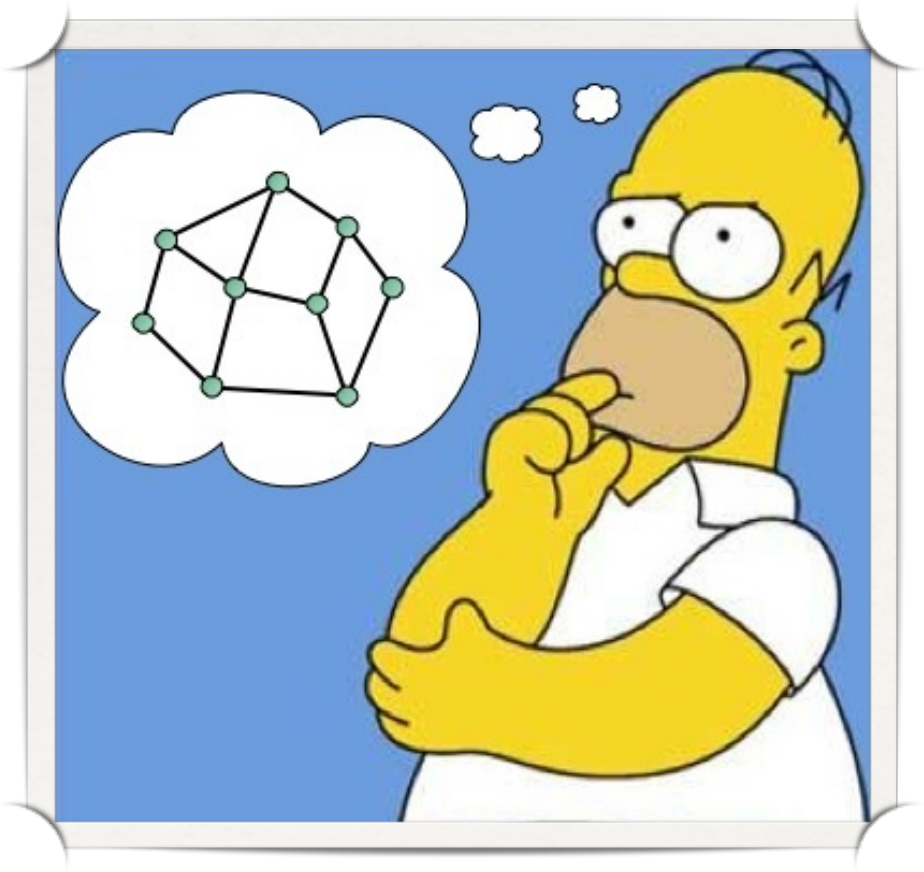


Algoritmo di Ear Decomposition



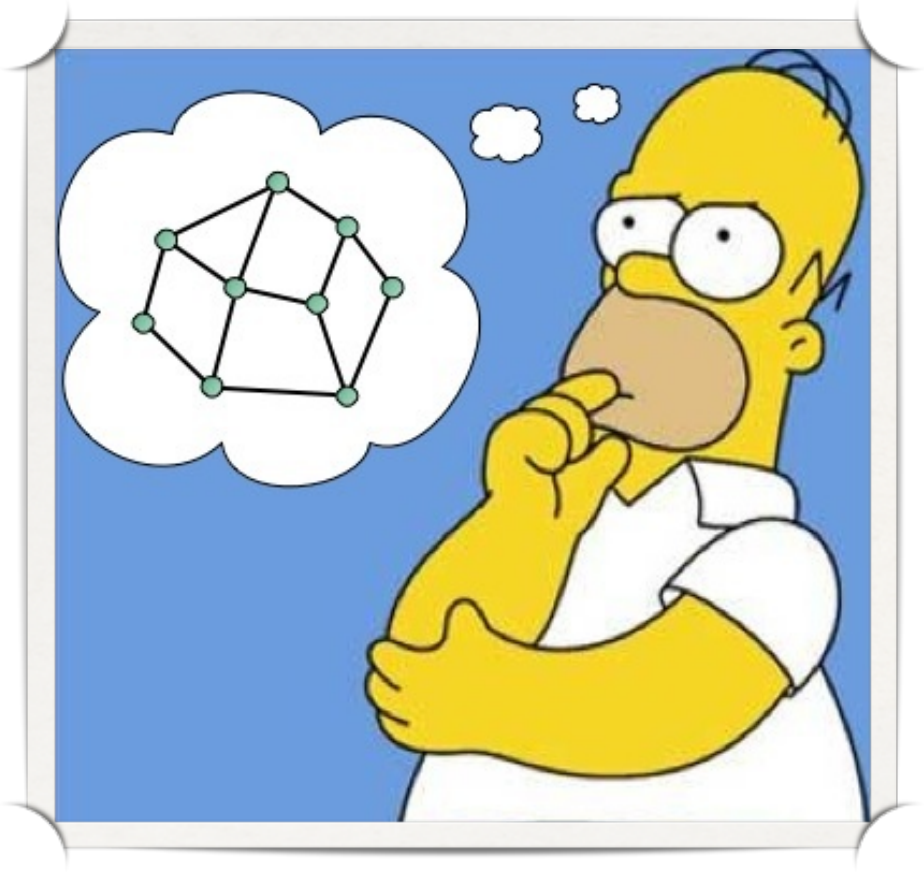
- Concetti alla base dell'algoritmo
- Un Algoritmo per PRAM
- Un'istanza d'esempio

Concetti alla base dell'algoritmo



Come trovo una ear decomposition?

Concetti alla base dell'algoritmo



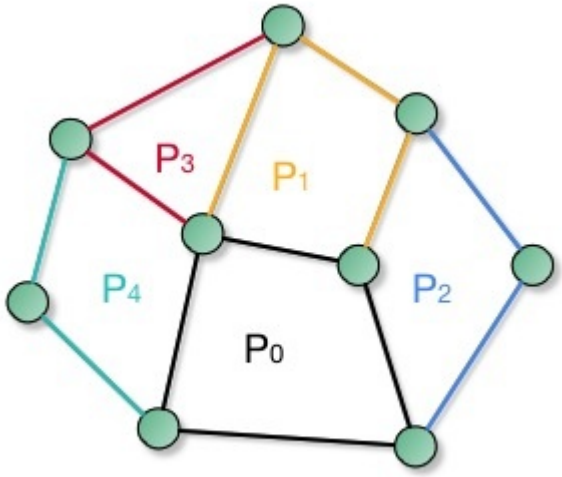
Partiamo dalla fine,
la scomposizione H.

Come trovo una ear decomposition?

Idea 1: Da un insieme di cicli …

Sia dato un grafo non orientato G=(V,E) biconnesso con una data open ear decomposition **$H=(P_0, P_1, \ldots, P_k)$**.

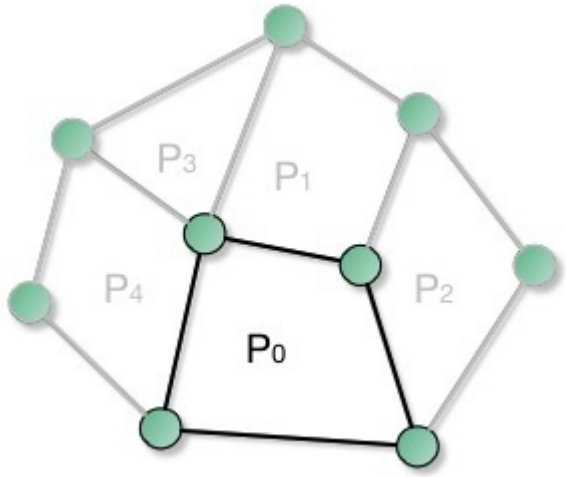
- **P_0** è un ciclo semplice
- **P_1** è un cammino semplice con due estremi distinti e nessun arco in **P_0**
- **P_2** è un cammino semplice con due estremi distinti e nessun arco in **$H_1 = P_0 \cup P_1$**
- In generale **P_{i+1}** è un cammino semplice, che froma a sua volta un ciclo passando per **$H_i = P_0 \cup P_1 \cup \ldots \cup P_i$**



Idea 1: Da un insieme di cicli …

Sia dato un grafo non orientato G=(V,E) biconnesso con una data open ear decomposition **$H=(P_0, P_1, \ldots, P_k)$**.

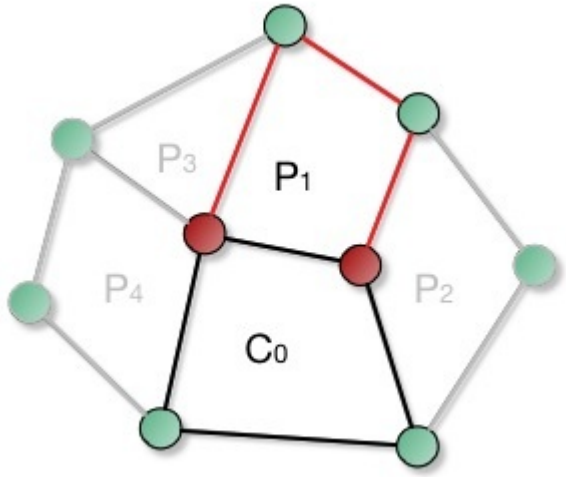
- **P_0** è un ciclo semplice
- **P_1** è un cammino semplice con due estremi distinti e nessun arco in **P_0**
- **P_2** è un cammino semplice con due estremi distinti e nessun arco in **$H_1 = P_0 \cup P_1$**
- In generale **P_{i+1}** è un cammino semplice, che forma a sua volta un ciclo passando per **$H_i = P_0 \cup P_1 \cup \ldots \cup P_i$**



Idea 1: Da un insieme di cicli …

Sia dato un grafo non orientato G=(V,E) biconnesso con una data open ear decomposition **$H=(P_0, P_1, \ldots, P_k)$**.

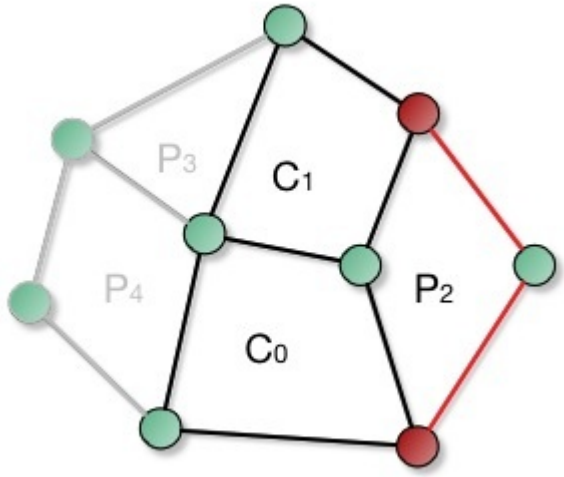
- **P_0** è un ciclo semplice
- P_1 è un cammino semplice con due estremi distinti e nessun arco in P_0
- **P_2** è un cammino semplice con due estremi distinti e nessun arco in **$H_1 = P_0 \cup P_1$**
- In generale **P_{i+1}** è un cammino semplice, che forma a sua volta un ciclo passando per **$H_i = P_0 \cup P_1 \cup \ldots \cup P_i$**



Idea 1: Da un insieme di cicli …

Sia dato un grafo non orientato G=(V,E) biconnesso con una data open ear decomposition **$H=(P_0, P_1, \ldots, P_k)$**.

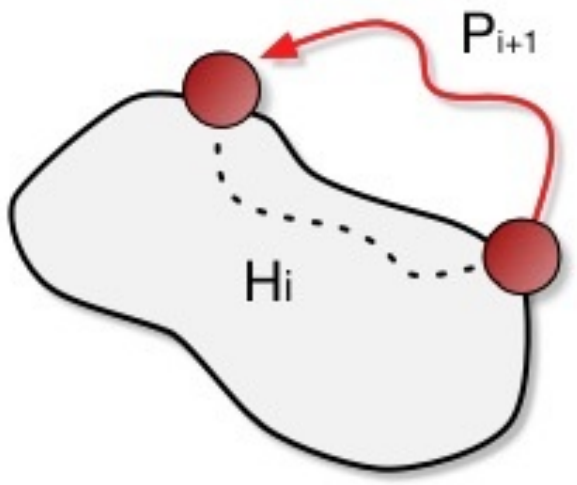
- **P_0** è un ciclo semplice
- **P_1** è un cammino semplice con due estremi distinti e nessun arco in **P_0**
- **P_2** è un cammino semplice con due estremi distinti e nessun arco in **$H_1 = P_0 \cup P_1$**
- In generale **P_{i+1}** è un cammino semplice, che forma a sua volta un ciclo passando per **$H_i = P_0 \cup P_1 \cup \ldots \cup P_i$**



Idea 1: Da un insieme di cicli ...

Sia dato un grafo non orientato G=(V,E) biconnesso con una data open ear decomposition **$H=(P_0, P_1, \ldots, P_k)$**.

- **P_0** è un ciclo semplice
- **P_1** è un cammino semplice con due estremi distinti e nessun arco in **P_0**
- **P_2** è un cammino semplice con due estremi distinti e nessun arco in **$H_1 = P_0 \cup P_1$**
- In generale **P_{i+1}** è un cammino semplice, che forma a sua volta un ciclo passando per **$H_i = P_0 \cup P_1 \cup \ldots \cup P_i$**

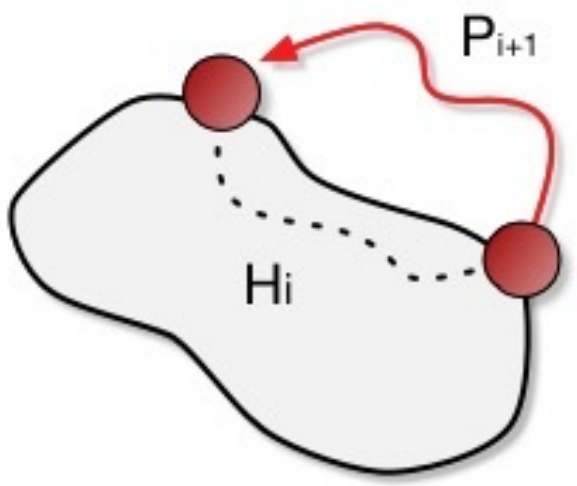


Dim. per induzione

Idea 1: Da un insieme di cicli …

Sia dato un grafo non orientato G=(V,E) biconnesso con una data open ear decomposition **$H=(P_0, P_1, \ldots, P_k)$**.

- **P_0** è un ciclo semplice
- **P_1** è un cammino semplice con due estremi distinti e nessun arco in **P_0**
- **P_2** è un cammino semplice con due estremi distinti e nessun arco in **$H_1 = P_0 \cup P_1$**
- In generale **P_{i+1}** è un cammino semplice, che forma a sua volta un ciclo passando per **$H_i = P_0 \cup P_1 \cup \ldots \cup P_i$**

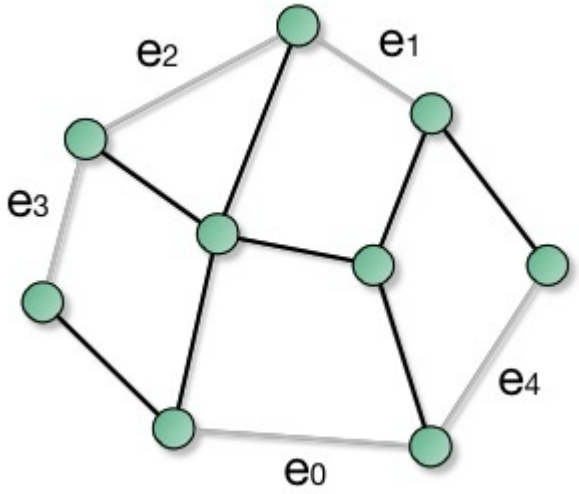


È evidente allora che una ear decomposition H di G porta ad un insieme di cicli nel grafo stesso.

Idea 1: ... All'albero di copertura

Dati G=(V,E) e la sua ear decomposition **$H=(P_0, P_1, \ldots, P_k)$**, se eliminiamo un arco **e** di P_i per ogni **i**, otteniamo un albero di copertura.

- Effettuiamo l'operazione inversa:
- Cosideriamo uno spanning tree $T=(V_T,E_T)$ di G.
- Per ogni arco **e** non in E_T, se lo aggiungiamo a **T** otteniamo un ciclo fondamentale C_e
- Alla fine otteniamo un sistema di cicli fondamentali per G.

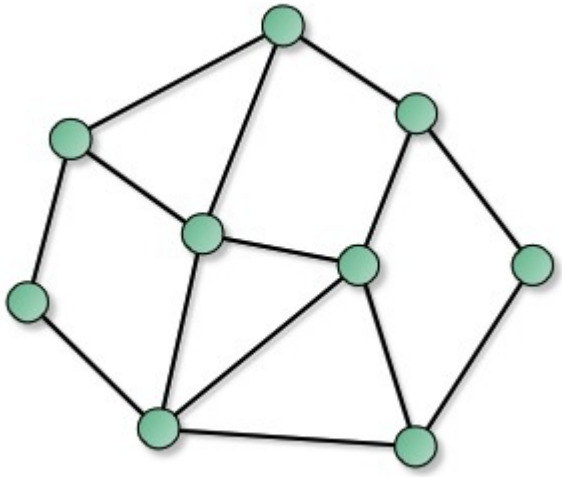


Dim. per induzione

Idea 1: ... All'albero di copertura

Dati G=(V,E) e la sua ear decomposition **$H=(P_0, P_1, \ldots, P_k)$**, se eliminiamo un arco **e** di P_i per ogni **i**, otteniamo un albero di copertura.

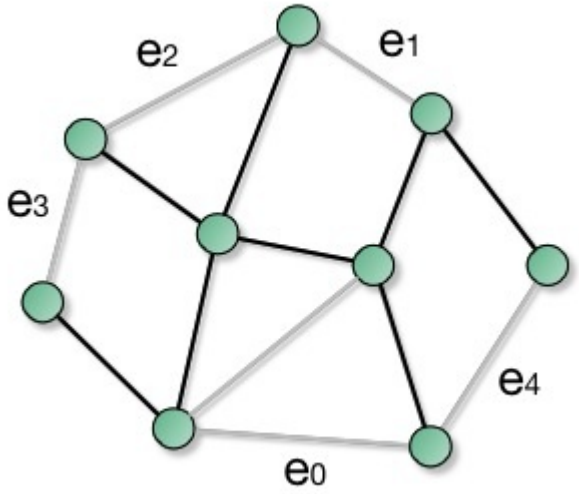
- Effettuiamo l'operazione inversa:
- Cosideriamo uno spanning tree $T=(V_T,E_T)$ di G.
- Per ogni arco **e** non in E_T, se lo aggiungiamo a **T** otteniamo un ciclo fondamentale C_e
- Alla fine otteniamo un sistema di cicli fondamentali per G.



Idea 1: ... All'albero di copertura

Dati G=(V,E) e la sua ear decomposition **$H=(P_0, P_1, \ldots, P_k)$**, se eliminiamo un arco **e** di P_i per ogni **i**, otteniamo un albero di copertura.

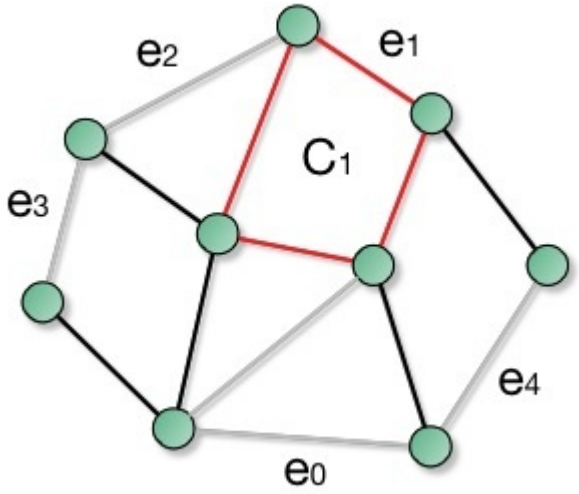
- Effettuiamo l'operazione inversa:
- Cosideriamo uno spanning tree **$T=(V_T,E_T)$** di G.
- Per ogni arco **e** non in E_T, se lo aggiungiamo a **T** otteniamo un ciclo fondamentale C_e
- Alla fine otteniamo un sistema di cicli fondamentali per G.



Idea 1: ... All'albero di copertura

Dati G=(V,E) e la sua ear decomposition **$H=(P_0, P_1, \ldots, P_k)$**, se eliminiamo un arco **e** di P_i per ogni **i**, otteniamo un albero di copertura.

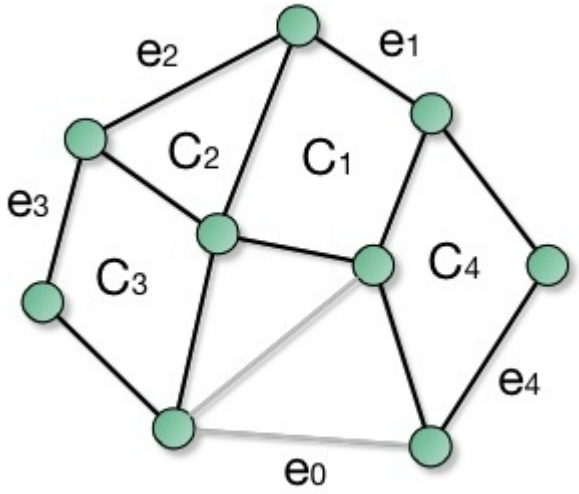
- Effettuiamo l'operazione inversa:
- Cosideriamo uno spanning tree **$T=(V_T,E_T)$** di G.
- Per ogni arco **e** non in E_T, se lo aggiungiamo a **T** otteniamo un ciclo fondamentale C_e
- Alla fine otteniamo un sistema di cicli fondamentali per G.



Idea 1: ... All'albero di copertura

Dati G=(V,E) e la sua ear decomposition **$H=(P_0, P_1, \ldots, P_k)$**, se eliminiamo un arco **e** di P_i per ogni **i**, otteniamo un albero di copertura.

- Effettuiamo l'operazione inversa:
- Cosideriamo uno spanning tree **$T=(V_T,E_T)$** di G.
- Per ogni arco **e** non in E_T, se lo aggiungiamo a **T** otteniamo un ciclo fondamentale C_e
- Alla fine otteniamo un sistema di cicli fondamentali per G.

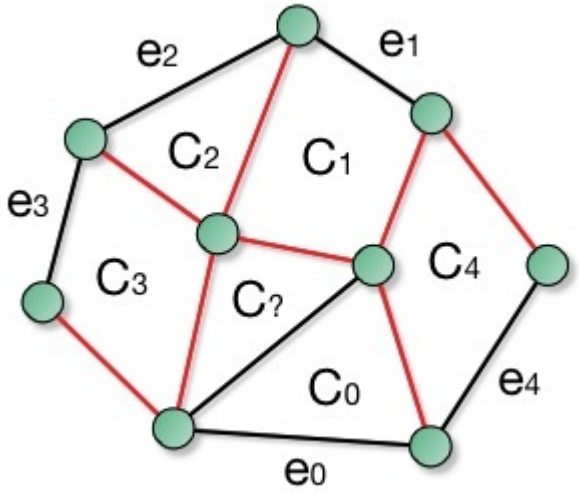


Dim. per induzione

Idea 1: ... All'albero di copertura

Dati G=(V,E) e la sua ear decomposition **$H=(P_0, P_1, \ldots, P_k)$**, se eliminiamo un arco **e** di P_i per ogni **i**, otteniamo un albero di copertura.

- Effettuiamo l'operazione inversa:
- Cosideriamo uno spanning tree **$T=(V_T,E_T)$** di G.
- Per ogni arco **e** non in E_T, se lo aggiungiamo a **T** otteniamo un ciclo fondamentale C_e
- Alla fine otteniamo un sistema di cicli fondamentali per G.

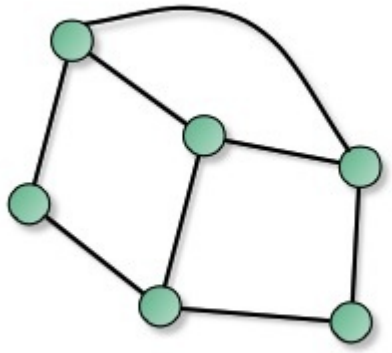


Possiamo trovare un metodo per calcolare la ear decomposition a partire da uno spanning tree.

Idea 2: Algoritmo ingenuo

Proviamo a costruire un algoritmo partendo dall'idea 1:

- Calcoliamo uno spanning Tree $\mathbf{T=(V_T,E_T)}$
- Associamo un ear P_e ad ogni e non in $\mathbf{E_T}$
- Dato $\mathbf{C_e}$ ciclo fondamentale indotto da e poniamo $P_e = C_e$.



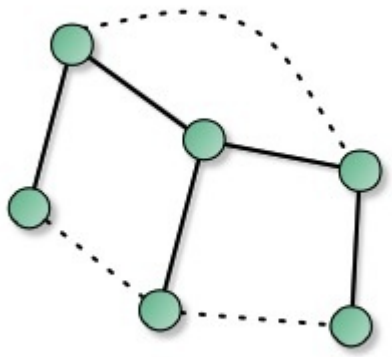
Ovviamente l'algoritmo non funziona a causa delle **sovrapposizioni**:

- C_e quindi potrebbe sovrapporsi con qualche altro ciclo fondamentale.
- Quindi ear diverse potrebbero avere archi in comune: Non è permesso!

Idea 2: Algoritmo ingenuo

Proviamo a costruire un algoritmo partendo dall'idea 1:

- Calcoliamo uno spanning Tree $\mathbf{T=(V_T,E_T)}$
- Associamo un ear P_e ad ogni e non in $\mathbf{E_T}$
- Dato $\mathbf{C_e}$ ciclo fondamentale indotto da e poniamo $P_e = C_e$.



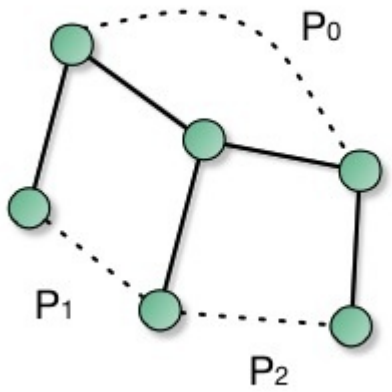
Ovviamente l'algoritmo non funziona a causa delle **sovrapposizioni**:

- C_e quindi potrebbe sovrapporsi con qualche altro ciclo fondamentale.
- Quindi ear diverse potrebbero avere archi in comune: Non è permesso!

Idea 2: Algoritmo ingenuo

Proviamo a costruire un algoritmo partendo dall'idea 1:

- Calcoliamo uno spanning Tree $\mathbf{T=(V_T,E_T)}$
- Associamo un ear $\mathbf{P_e}$ ad ogni $\mathbf{e}$ non in $\mathbf{E_T}$
- Dato $\mathbf{C_e}$ ciclo fondamentale indotto da e poniamo $P_e = C_e$.



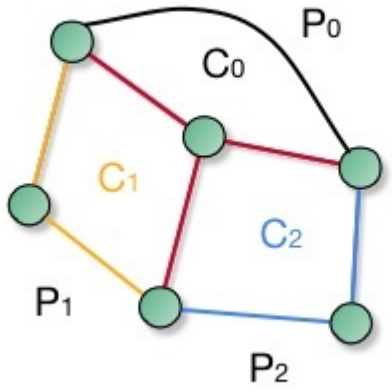
Ovviamente l'algoritmo non funziona a causa delle **sovrapposizioni**:

- C_e quindi potrebbe sovrapporsi con qualche altro ciclo fondamentale.
- Quindi ear diverse potrebbero avere archi in comune: Non è permesso!

Idea 2: Algoritmo ingenuo

Proviamo a costruire un algoritmo partendo dall'idea 1:

- Calcoliamo uno spanning Tree $\mathbf{T=(V_T,E_T)}$
- Associamo un ear $\mathbf{P_e}$ ad ogni $\mathbf{e}$ non in $\mathbf{E_T}$
- Dato $\mathbf{C_e}$ ciclo fondamentale indotto da $\mathbf{e}$ poniamo $\mathbf{P_e = C_e}$.



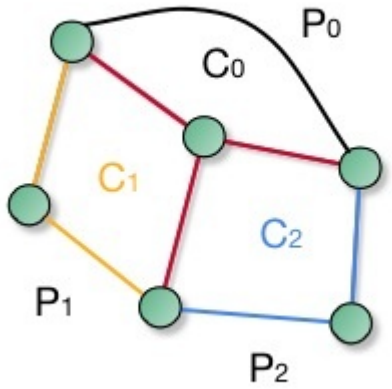
Ovviamente l'algoritmo non funziona a causa delle **sovrapposizioni**:

- C_e quindi potrebbe sovrapporsi con qualche altro ciclo fondamentale.
- Quindi ear diverse potrebbero avere archi in comune: Non è permesso!

Idea 2: Algoritmo ingenuo

Proviamo a costruire un algoritmo partendo dall'idea 1:

- Calcoliamo uno spanning Tree $\mathbf{T=(V_T,E_T)}$
- Associamo un ear $\mathbf{P_e}$ ad ogni $\mathbf{e}$ non in $\mathbf{E_T}$
- Dato $\mathbf{C_e}$ ciclo fondamentale indotto da $\mathbf{e}$ poniamo $\mathbf{P_e = C_e}$.



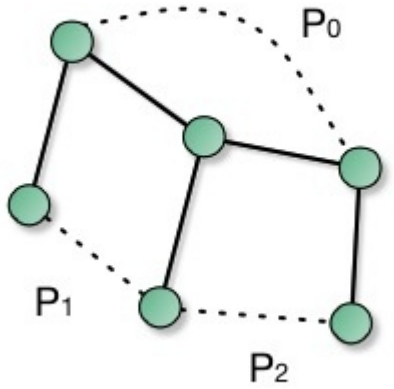
Ovviamente l'algoritmo non funziona a causa delle **sovrapposizioni**:

- C_e potrebbe sovrapporsi con qualche altro ciclo fondamentale.
- Quindi ear diverse potrebbero avere archi in comune: Non è permesso!

Idea 2: Etichettare gli archi

L'idea di partire dagli archi esterni all'albero per ottenere dei cicli è giusta, dobbiamo solo risovere le sovrapposizioni tra ear indotte da tali cicli.

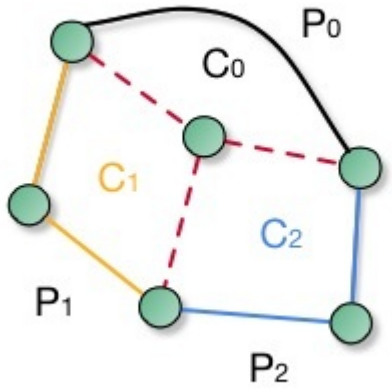
- Vogliamo considerare un ear P_e indotto da C_e a meno di sovrapposizioni.
- Possiamo eliminare tali conflitti scomponendo i cicli C_e in cammini semplici in modo tale che ogni cammino appartenga a ear diverse.
- La scomposizione cercata dei cicli verrá ottenuta tramite un opportuno etichettamento degli archi del grafo.



Idea 2: Etichettare gli archi

L'idea di partire dagli archi esterni all'albero per ottenere dei cicli è giusta, dobbiamo solo risovere le sovrapposizioni tra ear indotte da tali cicli.

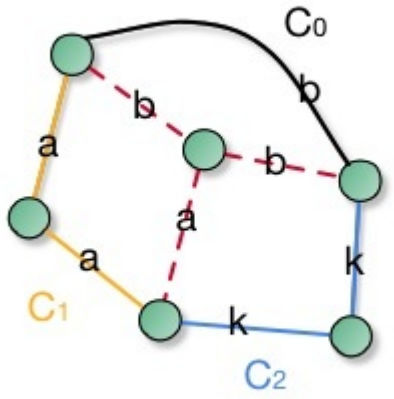
- Vogliamo considerare un ear P_e indotto da C_e a meno di sovrapposizioni.
- Possiamo eliminare tali conflitti scomponendo i cicli C_e in cammini semplici in modo tale che ogni cammino appartenga a ear diverse.
- La scomposizione cercata dei cicli verrá ottenuta tramite un opportuno etichettamento degli archi del grafo.



Idea 2: Etichettare gli archi

L'idea di partire dagli archi esterni all'albero per ottenere dei cicli è giusta, dobbiamo solo risovere le sovrapposizioni tra ear indotte da tali cicli.

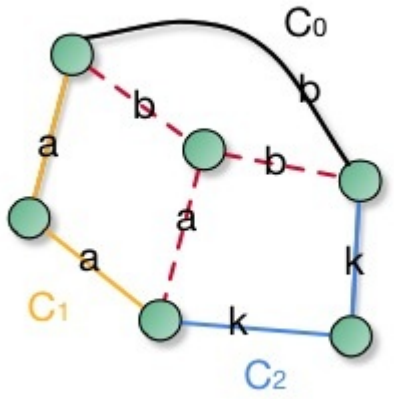
- Vogliamo considerare un ear P_e indotto da C_e a meno di sovrapposizioni.
- Possiamo eliminare tali conflitti scomponendo i cicli C_e in cammini semplici in modo tale che ogni cammino appartenga a ear diverse.
- La scomposizione cercata dei cicli verrá ottenuta tramite un opportuno etichettamento degli archi del grafo.



Idea 2: Etichettare gli archi

L'idea di partire dagli archi esterni all'albero per ottenere dei cicli è giusta, dobbiamo solo risovere le sovrapposizioni tra ear indotte da tali cicli.

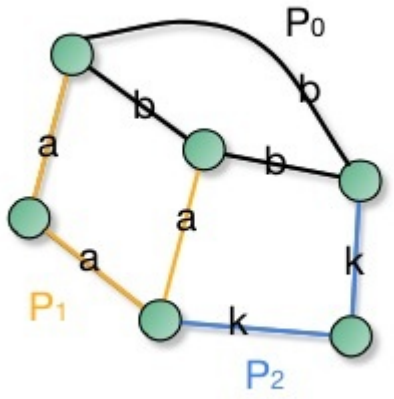
- Vogliamo considerare un ear $\mathbf{P_e}$ indotto da $\mathbf{C_e}$ a meno di sovrapposizioni.
- Possiamo eliminare tali conflitti scomponendo i cicli $\mathbf{C_e}$ in cammini semplici in modo tale che ogni cammino appartenga a ear diverse.
- La scomposizione cercata dei cicli verrá ottenuta tramite un opportuno etichettamento degli archi del grafo.



Idea 2: Etichettare gli archi

L'idea di partire dagli archi esterni all'albero per ottenere dei cicli è giusta, dobbiamo solo risovere le sovrapposizioni tra ear indotte da tali cicli.

- Vogliamo considerare un ear $\mathbf{P_e}$ indotto da $\mathbf{C_e}$ a meno di sovrapposizioni.
- Possiamo eliminare tali conflitti scomponendo i cicli $\mathbf{C_e}$ in cammini semplici in modo tale che ogni cammino appartenga a ear diverse.
- La scomposizione cercata dei cicli verrá ottenuta tramite un opportuno etichettamento degli archi del grafo.



L' algoritmo

Dato un grafo **G=(V,E)** che ammetta una open ear decomposition **H**, possiamo trovare H nel seguente modo:

1. Si calcola uno spanning tree $\mathbf{T=(V_T,E_T)}$ per G.

2. Si radica l'albero T su un arbitrario vertice **r**. Per ogni v in V:

Si calcola il livello **level[v]** in T.

Si calcola il padre **p[v]** in T.

3. Si etichettano gli archi **e** di **E** secondo una fissata numerazione **Id[e]**.

4. Per ogni arco **e**=(u,v) **non** in $\mathbf{E_T}$:

Si calcola il lowest common ancestor di e: **lca[e]**=lca[u,v]

Si etichetta **e** con **label[e]**=(level[**lca[e]**], id[e])

Si calcola il corrispondente ciclo indotto $\mathbf{C_e}$

L' algoritmo

5. Per ogni arco **g**=(u,v) in $\mathbf{E_T}$:

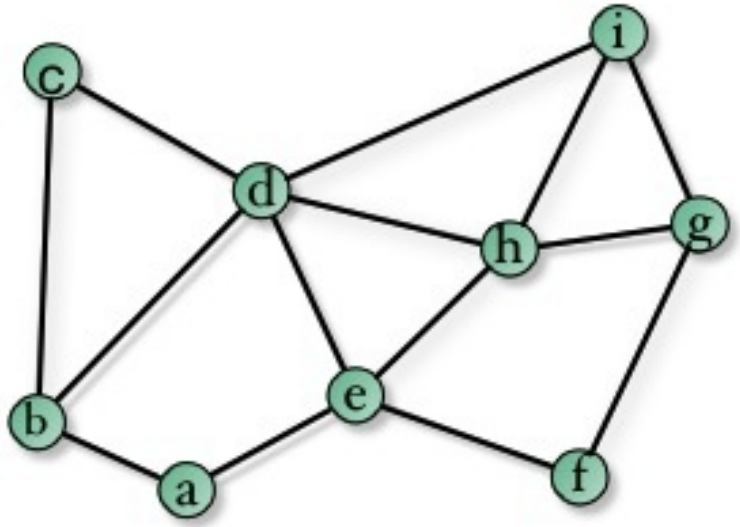
Si considerano tutti i cicli indotti che contengono **g**.

Si etichetta **g** con **label[g]** = min{label[e] | e in E_T, g in C_e}

6. Per ogni arco **e**=(u,v) **non** in $\mathbf{E_T}$ si calcola il sio ear:

$\mathbf{P_e}$ = {e} U {g in $\mathbf{E_T}$ | label[g] = label[e] }

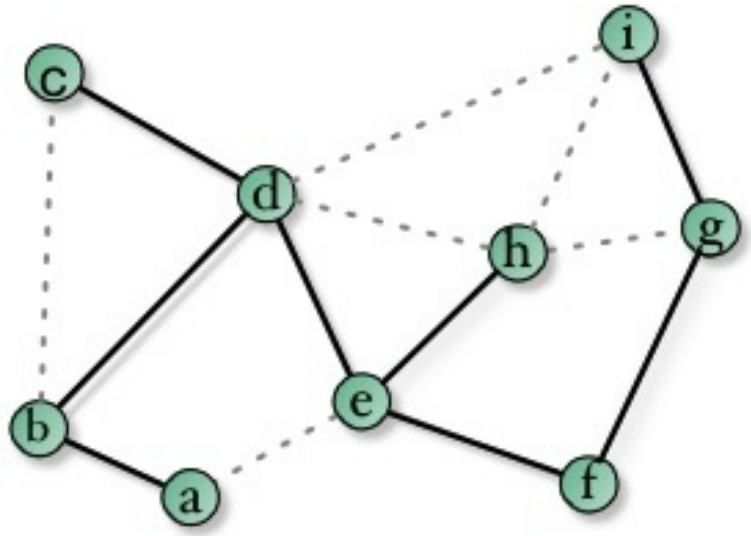
7. Si ordinano i $\mathbf{P_e}$ in base alla label[e]



STEP 1: Spanning Tree

Step 1. Si calcola uno spanning tree **$T=(V_T,E_T)$** per G.

- Usato come struttura con cui trovare i cicli dai quali indurremo gli ear.
- Può essere calcolato efficentemente su PRAM:
  - Algoritmo di Sollin
  - Si assumono archi di peso unitario
  - Complessità O(log n) su crcw
  - Complessità O(log n * log n) su erew



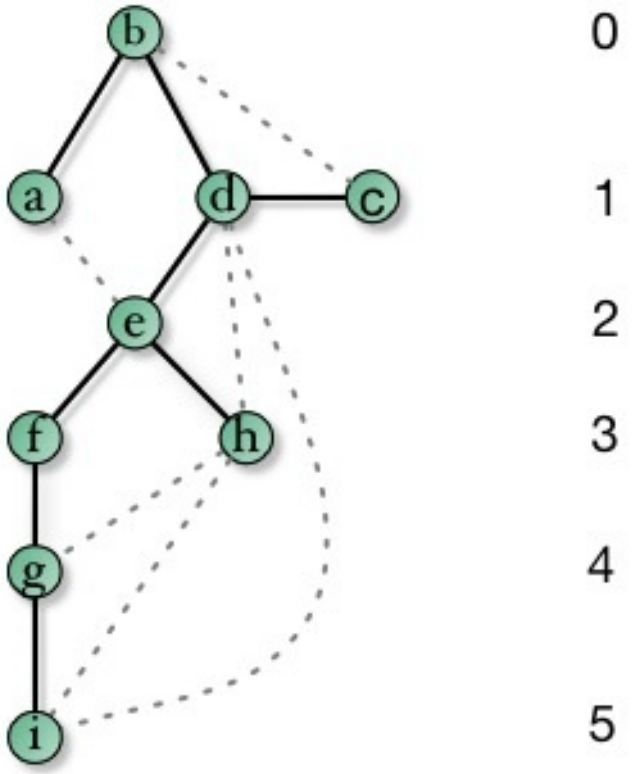
STEP 2: Strutturare l'albero

Step 2. Si radica l'albero T su un arbitrario vertice **r**. Per ogni v in V:

Si calcola il livello **level[v]** in T.

Si calcola il padre **p[v]** in T.

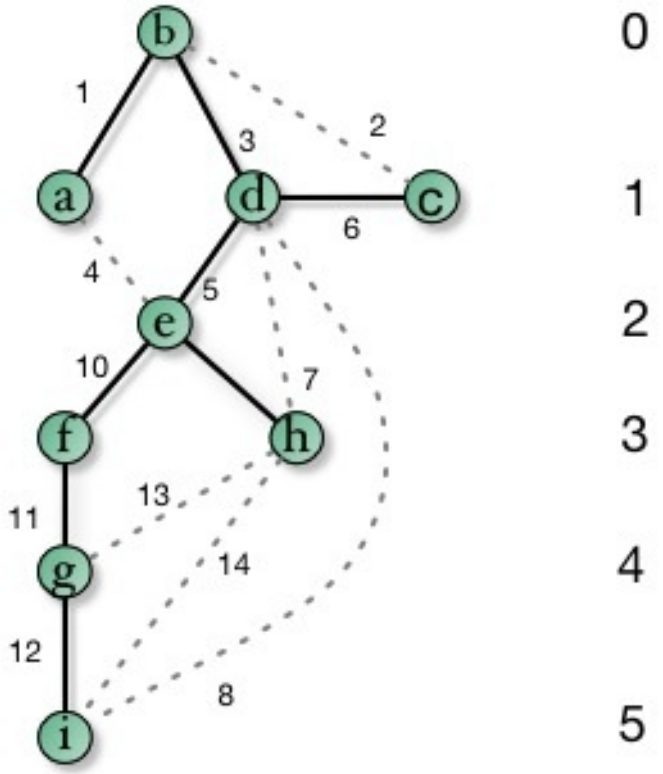
- La struttura ad albero radicato serve a semplificare le fasi successive:
  - I livelli aiutano a risolvere i conflitti tra archi appartenenti a più di un ciclo (quelli dell'albero).
  - Durante il radicamento si possono calcolare gli lca.
  - Permette di calcolare i cicli indotti.
- Può essere calcolato efficentemente su PRAM con l'algoritmo del Tour di Eulero.



STEP 3: Enumerazione archi

Step 3. Si etichettano gli archi **e** di E secondo una fissata numerazione **Id[e]** da 1 a m.

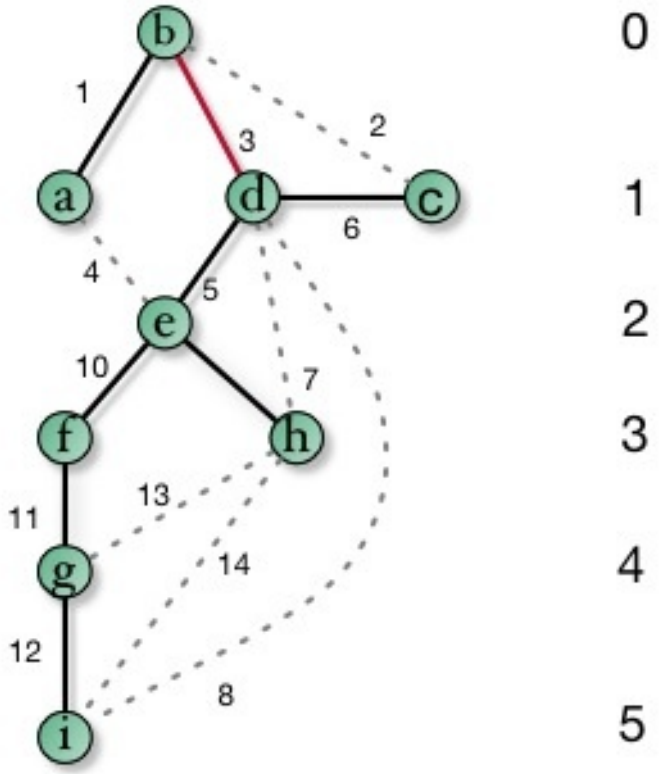
- L'indice degli archi serve a risolvere i conflitti quando anche il livello non basta.
- Può essere calcolato efficentemente su PRAM:
  - Algoritmo delle somme prefisse.



STEP 3: Enumerazione archi

Step 3. Si etichettano gli archi **e** di E secondo una fissata numerazione **Id[e]** da 1 a m.

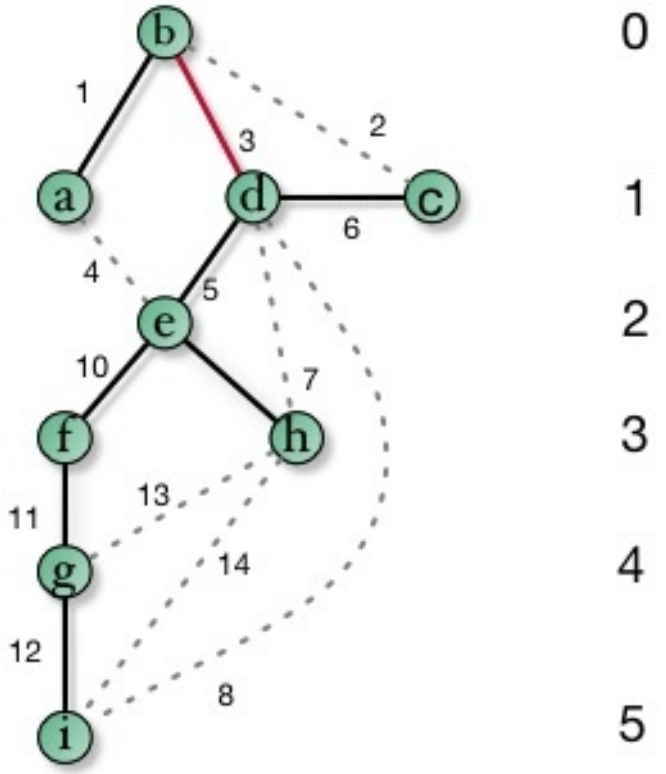
- L'indice degli archi serve a risolvere i **conflitti** quando anche il livello non basta.
- Può essere calcolato efficentemente su PRAM:
  - Algoritmo delle somme prefisse.



STEP 3: Enumerazione archi

Step 3. Si etichettano gli archi **e** di E secondo una fissata numerazione **Id[e]** da 1 a m.

- L'indice degli archi serve a risolvere i **conflitti** quando anche il livello non basta.
- Può essere calcolato efficentemente su PRAM:
  - Algoritmo delle somme prefisse.



STEP 4: Etichette esterne a T

Step 4: Per ogni arco **e**=(u,v) **non** in E_T:

- Si calcola il lowest common ancestor dei vertici dell'arco **e**: **lca[e]**=lca[u,v].

  Gli lca si trovano tramite Tour di eulero.

- Si etichetta **e** con **label[e]**=(level[**lca[e]**], id[e])

  Ciascuna etichetta rappresenta un ear.

  Il valore dell'etichetta serve a risolvere i conflitti quando due ear si contendono uno stesso arco (ovviamente dell'albero).

- Si calcola il corrispondente ciclo indotto C_e

ID[e]	Label
2	(0,2)
4	(0,4)
7	(1,7)
8	(1,8)
13	(2,13)
14	(2,14)

STEP 5: Etichette interne a T

Step 5: Per ogni arco **g**=(u,v) in $\mathbf{E_T}$:

Si considerano tutti i cicli indotti che contengono **g**.

Si etichetta **g** con **label[g]** = min{label[e] | e in E_T, g in C_e}

- Gli archi dell'albero vengono etichetatti con valori presi dagli archi esterni all'albero stesso.
- In questa fase vengono risolti i conflitti dovuti alla sovrapposizione dei cicli indotti dagli archi esterni.
- L'algoritmo usato è la ricerca del minimo.

STEP 5: Etichette interne a T

ID[g]	e: Ce contenenti g	label[e]	label[g]
1	4	(0,4)	(0,4)
3	2 4	(0,2) (0,4)	(0,2)
5	4 7 8	(0,4) (1,7) (1,8)	(0,4)
6	2	(0,2)	(0,2)
9	7 13 14	(1,7) (2,13) (2,14)	(1,7)
10	8 13 14	(1,8) (2,13) (2,14)	(1,8)
11	8 13 14	(1,8) (2,13) (2,14)	(1,8)
12	8 14	(1,8) (2,14)	(1,8)

STEP Finale

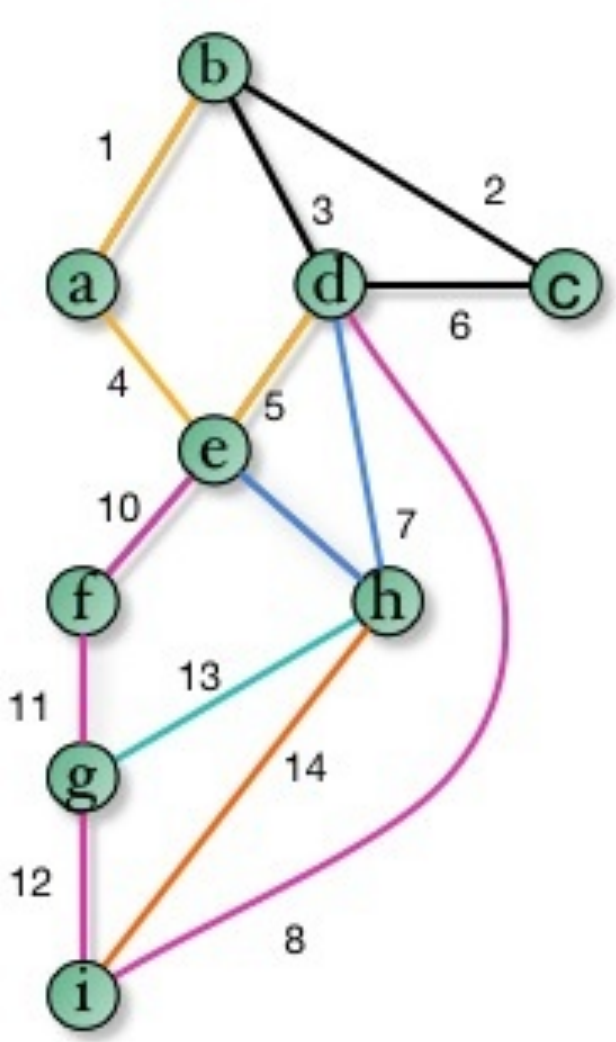
6. Per ogni arco **e**=(u,v) **non** in E_T si calcola il suo ear:

Pe = {e} U {g in E_T | label[g] = label[e] }

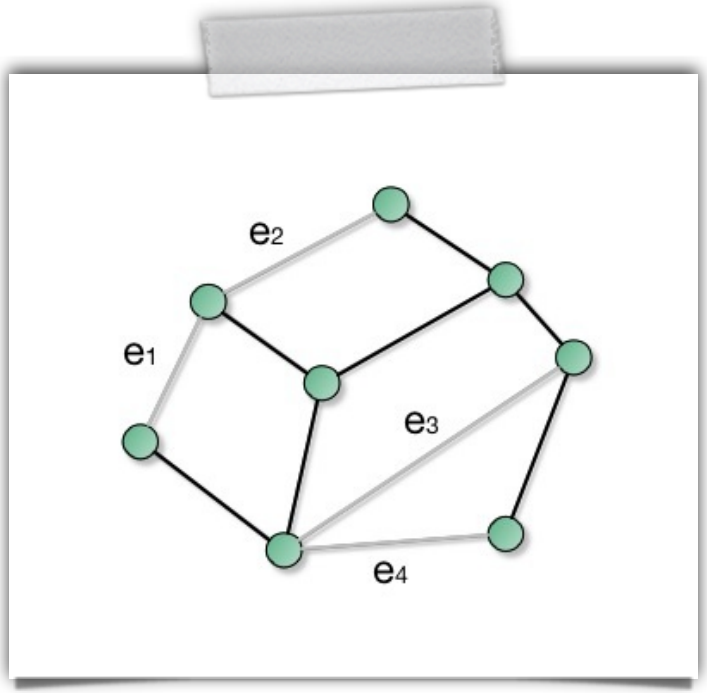
- Ovvero si aggregano tutti gli archi con etichetta uguale.

7. Si ordinano i **Pe** in base alla label[e]

Ear	Label	Archi
P0	(0,2)	2 3 6
P1	(0,4)	1 4 5
P2	(1,7)	7 9
P3	(1,8)	8 10 11 12
P4	(2,13)	13
P5	(2,14)	14



Analisi dell'algoritmo



- Correttezza
- Complessità

Analisi della Complessità

Se consideriamo una macchina parallela crcw l'algoritmo di ear decomposition si basa su una serie di algoritmi di complessità nota:

Step 1. Calcolo dello spanning tree T di G: alg. di Sollin **O(log n)**

Step 2. Radicamento di T e calcolo livelli: **O(log n)**

Step 3. Etichettamento degli archi: Alg. delle somme prefisse **O(log n)**

Step 4. Calcolo degli lce per ogni arco esterno a T: **O(log n)**

Step 5. Calcolo delle etichette per gli archi di T: Ricerca del minimo **O(log n)**

Step 6-7. Ordinamento delle ear: Alg. di ordinamneto **O(log n)**

Analisi della Complessità

- Per una macchina crcw allora l'algoritmo ha una complessità parallela di **O(log n)** e un numero di prcessori utilizzati lineari su m: costo **O(m * log n)**.

- Per una macchina crew il tempo aumenta di un fattore logaritmico su n dovuto in generale alla necessità di fare bradcasting delle informazioni ai vari processori.

- In dettaglio il costo e il tempo su una crew è dominato dallla ricerca dello spanning tree: tempo parallelo **O(log n * log n)** e costo n*n.