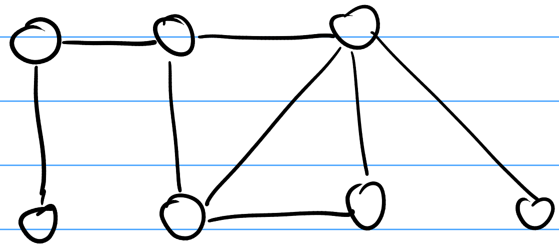
Algoritmi di Approssimazione (sezioni del libro 35.1 e 35.3)

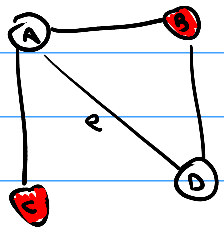
Considerate il seguente problema:

- Dato un grafo semplice G trovate il più piccolo insieme di vertici
$$U \subseteq V(G)$$
tale che ogni arco del grafo tocchi un vertice in U

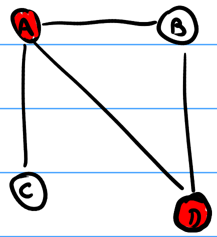


Def Vertex Cover: un vertex cover di $G=(V,E)$ è un insieme $U \subseteq V$ tale che
$$\forall \{v,w\} \in E \quad , \quad v \in U \text{ oppure } w \in U$$

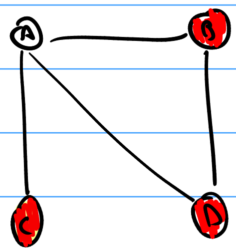
E.s.



$\{B,C\}$ non è un vertex cover perché l'arco e non è coperto



$\{A,D\}$ è un vertex cover di dimensione 2.



$\{B,C,D\}$ è un vertex cover di dimensione 3.

Vedi esempio precalcolato 1

Q: Qual è il minimo numero di vertici in una soluzione per il grafo nell'esempio

- Vediamo un potenziale algoritmo greedy e particolarmente semplice:
  - vogliamo coprire tutti gli archi del grafo con meno vertici possibili
  - allora prendiamo il vertice che copre più archi possibili tra quelli scoperti

```
VertexCover (G):
    S = {}
    while E(G) ≠ ∅:
        prendi il vertice v di
        S := S ∪ {v}
        elimina da G tutti gli archi adiacenti a v
    return S
```

Vedi esempi precalcolato 2

- Nell'esempio che abbiamo visto, l'approccio greedy trova un vertex cover di dimensione 4.

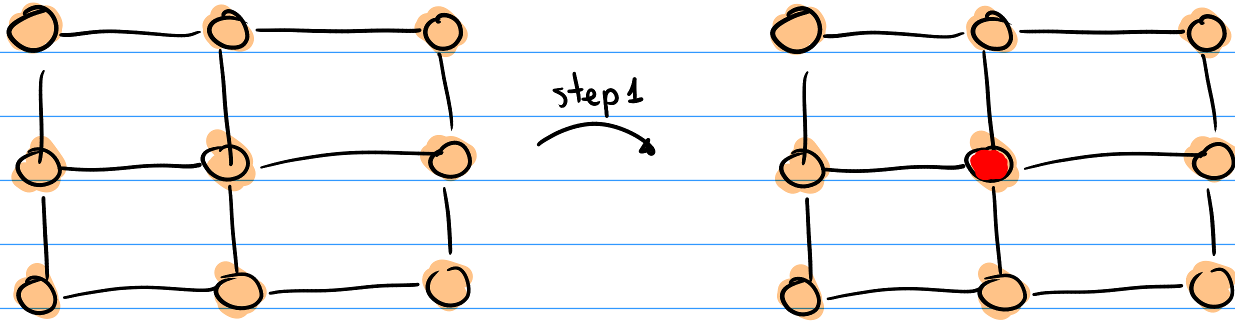
- È possibile fare di meglio? Osservate il cammino

$$0 - 4 - 10 - 6 - 1 - 3 - 7 - 2 - 9$$

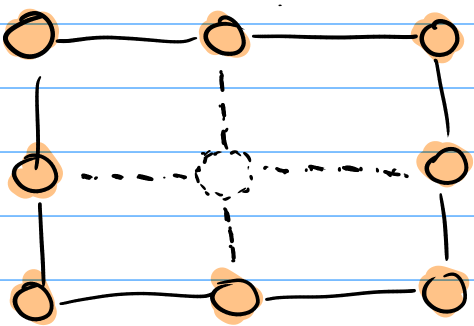
Il cammino contiene 8 archi: ogni vertice nella copertura non può coprirne più di due

L'algoritmo allora risolve il problema?

Vediamo questo esempio:



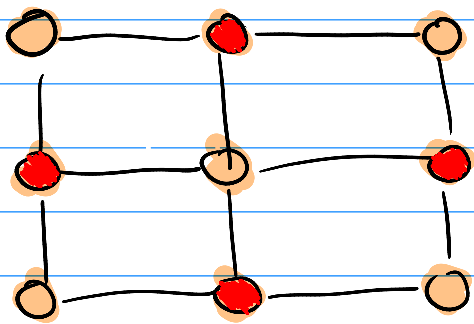
- il vertice scelto è quello con grado più grande (unico vertice di grado 5)



- il resto del grafo è un ciclo con 8 archi e 8 vertici
- per ragioni simili ci vogliono altri ≥4 vertici per completare la copertura

- Quindi l'algoritmo trova una copertura di 5 vertici. È ottimo?

- No! L'ottimo è in effetti 4.



- Quindi abbiamo dimostrato che l'algoritmo greedy non è ottimo.
- È sempre una buona idea provare a trovare algoritmi
  - SEMPLICI
  - CAPIRE SE SONO OTTIMI, TROVANDO MAGARI CONTRO ESEMPI

- In realtà il problema del Vertex Cover non possiede nessun algoritmo efficiente che risolva il problema su tutti gli input.

- Vertex Cover è un problema NP-hard, ovvero uno di quei problemi per cui
  - non si conoscono algoritmi polinomiali
  - si pensa che algoritmi del genere non esistano

- Di fatto per vertex cover gli algoritmi noti hanno tutti complessità $2^{\Theta(n)}$, ovvero ESPONENZIALE.

Come fare a risolvere questi problemi?

(A) Usare l'algoritmo esponenziale?
(B) Accettare delle soluzioni non ottime, ottenute con algoritmi efficienti?

L'opzione (A) è praticabile solo per input PICCOLISSIMI. Su input anche solo moderatamente grandi un algoritmo di complessità esponenziale è inutilizzabile

L'opzione (B) invece può essere valida se la soluzione ottenuta non è troppo peggiore di quella ottima.
In questo caso abbiamo due tipi di algoritmi.

ALGORITMI DI APPROSSIMAZIONE: algoritmi che riescono a trovare soluzioni la cui vicinanza alla soluzione ottima è DIMOSTRABILMENTE GARANTITA
E.g. un algoritmo che trova una soluzione di costo ≤ 2.37 volte il costo minimo

ALGORITMI EURISTICI: algoritmi che trovano soluzioni per cui non si ha nessuna garanzia riguardo alla qualità delle soluzioni, ma che magari sembrano comportarsi bene "in pratica".

Usando un algoritmo EURISTICO non si ha nessuna garanzia.
- la bontà presunta di algoritmi del genere potrebbe essere stata stimata su input MOLTO DIFFERENTI da quelli che dovete risolvere voi

In alcuni casi un algoritmo EURISTICO potrebbe semplicemente essere un ottimo algoritmo di approssimazione ma nessuno è riuscito a DIMOSTRARLO

In altri magari già si sa che l'algoritmo si comporta male su degli input patologici, ma viene usato lo stesso perché è semplice e si comporta bene sugli input provenienti da applicazioni pratiche

ALGORITMI di APPROSSIMAZIONE per problemi di OTTIMIZZAZIONE (massimizzazione e minimizzazione)

Abbiamo visto problemi di ottimizzazione, per esempio
- minimo albero di copertura, vertex cover (minimizzazione)
- selezione di attività (massimizzazione)

In un problema di ottimizzazione abbiamo un input del quale vogliamo trovare una soluzione

input I
soluzione S

per esempio I è un insieme di attività e S è un sottoinsieme di I

e una funzione che assegna un valore v: Soluzioni $\to \mathbb{R}$ alle soluzioni. Il problema di ottimizzazione consiste nel trovare una soluzione S tale che $v(S)$ sia ottimo. Cioè tale che
- $v(S)$ sia minimo; oppure
- $v(S)$ sia massimo

Dato un input I definiamo il suo ottimo come

$$OPT(I) : \max\{v(S) \text{ dove } S \text{ è una soluzione per } I\}$$

oppure

$$OPT(I) : \min\{v(S) \text{ dove } S \text{ è una soluzione per } I\}$$

Dato un algoritmo A, ci interessa il rapporto tra la qualità della soluzione trovata da A e quella ottima

Def: Rapporto di approssimazione $\dfrac{A(I)}{OPT(I)}$

Si dice che un algoritmo per un problema P abbia un rapporto di approssimazione γ se per qualunque input I

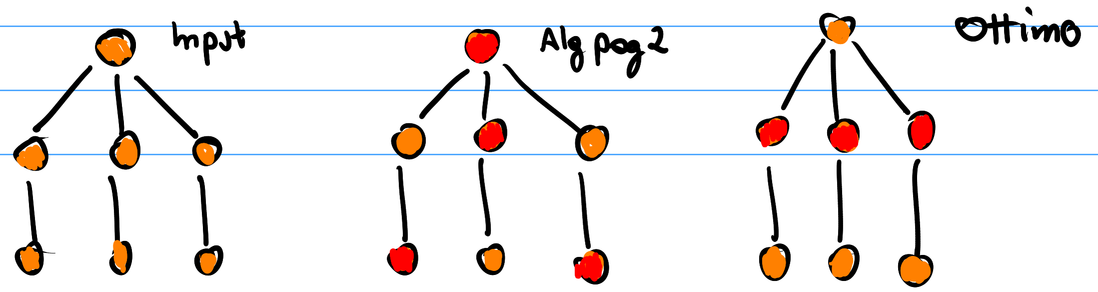
$$\frac{A(I)}{OPT(I)} \leq \gamma$$ se P è un problema di minimizzazione, e allora $\gamma \geq 1$.

$$\frac{A(I)}{OPT(I)} \geq \gamma$$ se P è un problema di massimizzazione, e allora $0 < \gamma \leq 1$

L'esempio a pag 3 ci dice che l'algoritmo greedy a pag 2

NON HA RAPPORTO DI APPROSSIMAZIONE $< 5/4$

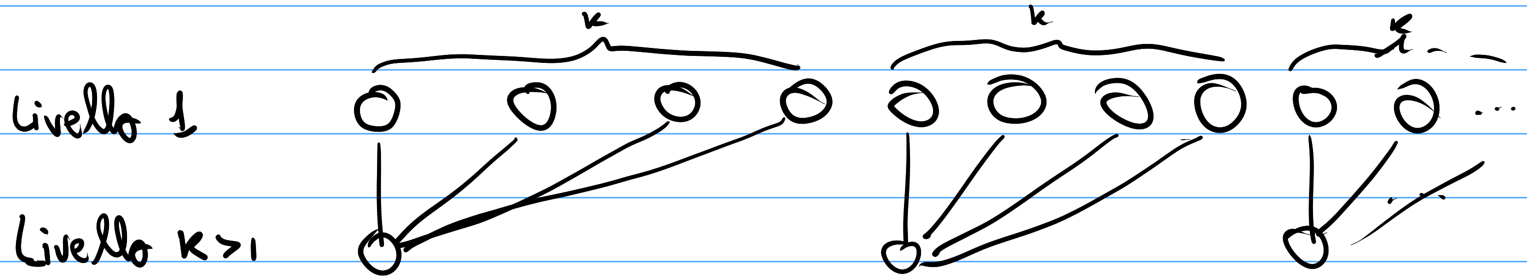
Altro esempio che dimostra che il rapporto non è neppure $< 4/3$



In effetti l'algoritmo a pag 2 non può garantire nessun rapporto di approssimazione costante.

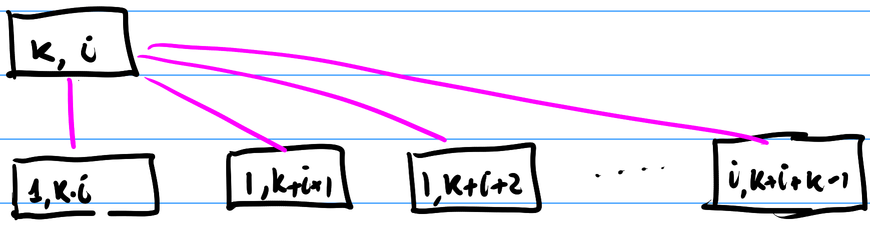
Thm L'algoritmo a pag 2 ha un rapporto di approssimazione $\Omega(\log n)$

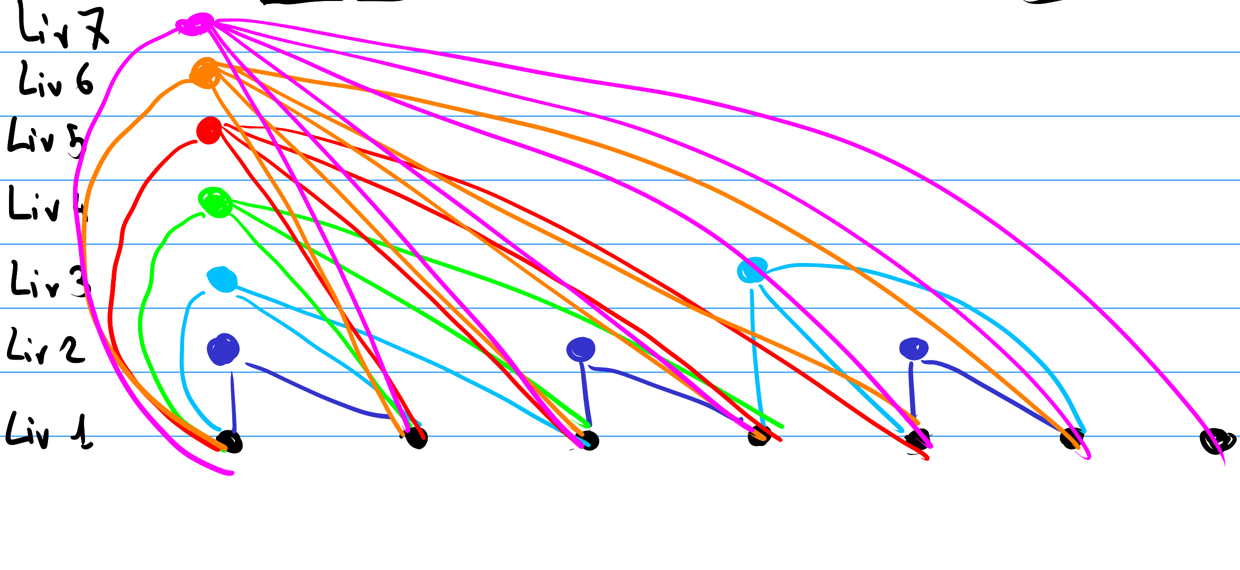
dim Costruiamo un grafo su cui l'algoritmo sbaglia di un fattore almeno $\Omega(\log n)$
Consideriamo il grafo G_ℓ con ℓ livelli



- Al livello 1 abbiamo ℓ vertici $\boxed{1,0}$ $\boxed{1,1}$ $\boxed{1,2}$ $\boxed{1,3}$ . . . . . $\boxed{1,\ell-1}$
- Al livello k abbiamo $\left\lfloor \frac{\ell}{k} \right\rfloor$ vertici $\boxed{k,0}$ $\boxed{k,1}$ $\boxed{k,2}$

Ci sono archi solo tra vertici di qualche livello k e il livello 1





- Vedremo come si comporta l'algoritmo a pag 2 su questo grafo G_ℓ

Passo 1: i vertici a livello $k>1$ hanno grado k e non hanno archi in comune
i vertici a livello 1 hanno grado $\leq \ell-1$

Viene scelto il vertice a livello ℓ

Passo 2: i vertici a livello $k>1$ ...
i vertici a livello 1 hanno grado $\leq \ell-2$
vengono scelti i vertici al livello $\ell-1$

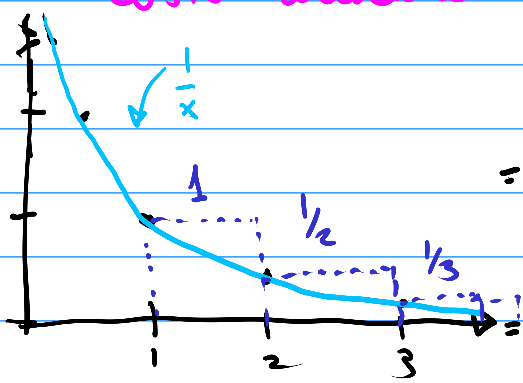
⋮

Passo i: i vertici a livello $k>1$ ...
i vertici a livello 1 hanno grado $\leq \ell-i$
vengono scelti tutti i vertici a livello $\ell-i+1$

Quindi alla fine l'algoritmo a pag 2 prende tutti i vertici di tutti i livelli >1.

costo soluzione $= \sum_{k=2}^{\ell} \left\lfloor \frac{\ell}{k} \right\rfloor \geq \sum_{k=2}^{\ell} \left(\frac{\ell}{k} - 1\right) \geq \left(\sum_{k=1}^{\ell} \frac{\ell}{k}\right) - 2\ell+1 =$

$= \ell\left(\sum_{k=1}^{\ell} \frac{1}{k}\right) - 2\ell+1 \geq \ell\left(\int_1^{\ell+1} \frac{1}{x}\,dx\right) - 2\ell+1$



$= \ell \cdot \ln(\ell+1) - \ell+1 = \Omega(\ell \cdot \log \ell)$

- Quindi l'algoritmo trova un vertex cover di dimensione $\Omega(\ell \cdot \log \ell)$ ma esiste una soluzione di dimensione ℓ. Quale?

- Perché $n \approx \ell \cdot \log \ell$ allora $\log \ell = \Theta(\log n)$ e quindi il rapporto di approssimazione é $\frac{\Omega(\ell \cdot \log \ell)}{\ell} = \Omega(\log \ell) = \boxed{\Omega(\log n)}$

Quindi abbiamo visto il problema del vertex cover

- abbiamo proposto un semplice algoritmo greedy
- abbiamo visto come si comporta . su certi esempi
- abbiamo visto (con degli esempi) che non raggiunge approx $\frac{4}{3}$
- abbiamo mostrato una famiglia di grafi su cui non raggiunge approssimazione $\Omega(\log n)$

tentativo 2 di un Algoritmo per Vertex cover, addirittura piú semplice

- elenchiamo tutti gli archi nel grafo. Per ogni arco scoperto $\{u,v\}$ prendiamo u e v nella copertura

Vertex Cover (G):

```
Cover := ∅
Scoperti := E
while |Scoperti| > 0 :
    prendi un arco {u,v} arbitrario da Scoperti
    Cover := Cover ∪ {u,v}
    Elimina da Scoperti tutti gli archi che toccano u o v
return Cover
```

$\boxed{\text{Vedi esempio precalcolato 3}}$

• Questo algoritmo trova un vertex cover di dimensione 8
Quello a pag. 2 di dimensione 4, tuttavia ...

- L'algoritmo a pagina 9 GARANTISCE UN FATTORE di approssimazione 2

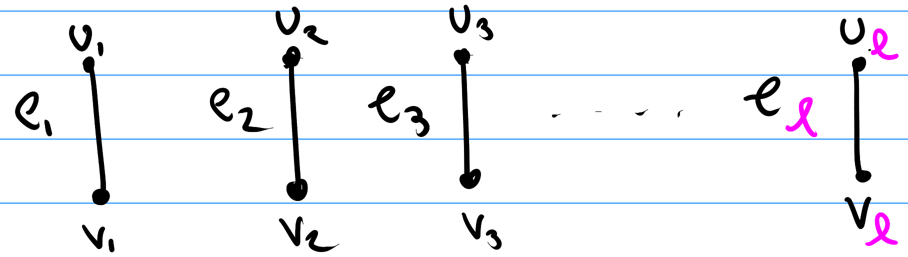
Ovvero: se un grafo G ha un vertex cover di dimensione C l'algoritmo trova un vertex cover di dimensione $2 \cdot C$

<u>oss</u> L'algoritmo a pagina 2 è più complicato di quello a pagina 9 eppure quello a pag 9 GARANTISCE UN FATTORE DI APPROSSIMAZIONE COSTANTE

<u>Thm</u> L'algoritmo a pagina 9 garantisce un fattore di approssimazione 2

<u>dim</u> • Quando l'algoritmo prende un arco $\{u,v\}$ allora tutti gli archi che toccano u oppure v saranno ignorati nei cicli successivi

Se l'algoritmo nella prima riga del ciclo while prende archi



- $W = \{u_1, u_2, \dots u_\ell, v_1, v_2, \dots v_\ell\}$ tutti vertici distinti, ovvero $|W| = 2\ell$
- Per coprire un arco e_i è necessario almeno un vertice da W
- Ogni vertice da W copre al massimo un arco
- Quindi: QUALUNQUE VERTEX COVER DEVE CONTENERE ℓ VERTICI
- L'algoritmo trova un vertex cover di dimensione $2 \cdot \ell$

Vedere esempio precalcolato 4

Esercizio Considerate un algoritmo che mette nella copertura TUTTI i vertici del grafo.

- In quel caso $A(I) = n$
- Si calcoli il rapporto di approssimazione di questo algoritmo.

ESERCIZIO (35.1-4)

- Dato un albero di n vertici connesso e rappresentato come
  - lista di adiacenza
  - sequenza di genitori

  } SCEGLIETE QUELLO CHE PREFERITE

trovate un algoritmo $O(n)$ che trova un vertex cover ottimo