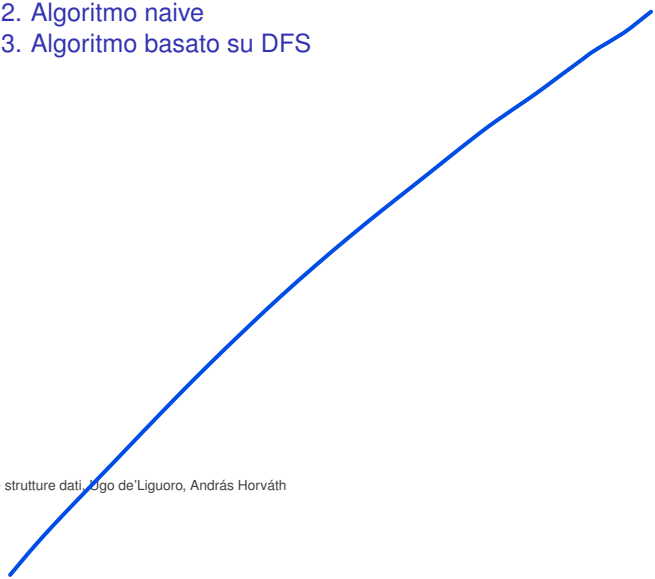
Grafi: componenti fortemente connesse

Corso di **Algoritmi e strutture dati**
Corso di Laurea in **Informatica**
Docenti: Ugo de'Liguoro, András Horváth

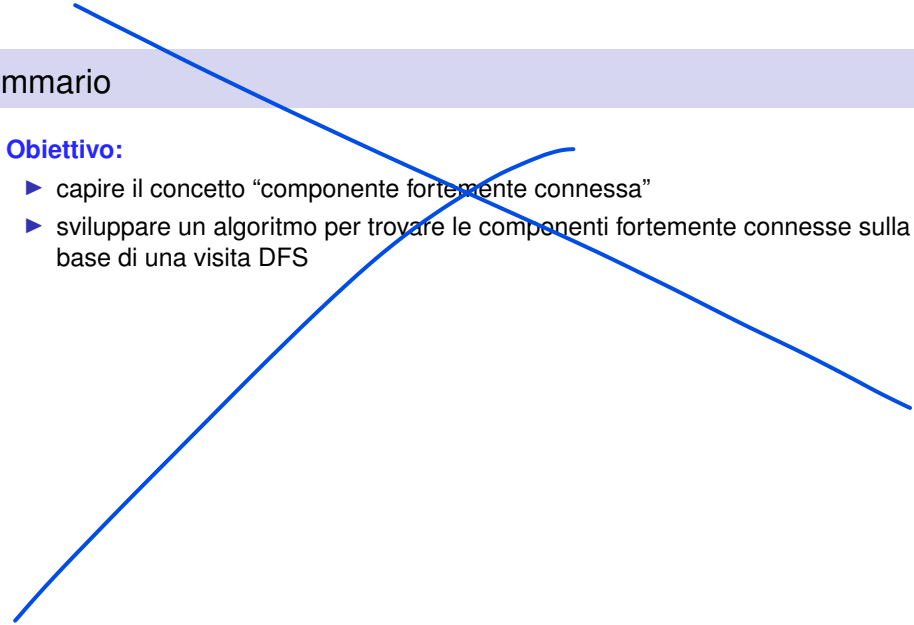
Indice



Sommario

Obiettivo:

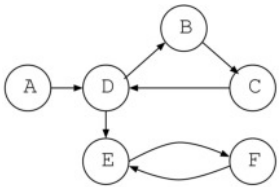
- capire il concetto “componente fortemente connessa”
- sviluppare un algoritmo per trovare le componenti fortemente connesse sulla base di una visita DFS



1. Definizione di componenti fortemente connessi (cfc)

- due nodi u e v sono mutuamente raggiungibili se u è raggiungibile da v e v è raggiungibile da u
- in un grafo $G = (V, E)$ orientato la relazione su $V \times V$ "mutuamente raggiungibile" è una relazione di equivalenza (riflessività, simmetria, transitività)
- le componenti fortemente connesse di un grafo orientato $G = (V, E)$ sono le classi della relazione di equivalenza su $V \times V$ "mutuamente raggiungibile"
- useremo la notazione $u \leftrightarrow v$ per indicare che i vertrici u e v sono mutuamente raggiungibili e quindi fanno parte della stessa cfc

1. Definizione di componenti fortemente connessi (cfc)



- 3 cfc: $\{A\}$,$\{D, B, C\}$,$\{E, F\}$

2. Calcolo della cfc di un vertice

- sia x un vertice di un grafo orientato $G = (V, E)$
  1. calcola l'insieme $D(x)$ dei vertici di G che sono raggiungibili da x
  2. calcola l'insieme $A(x)$ dei vertici di G da cui x è raggiungibile
  3. calcola $D(x) \cap A(x)$

2. Calcolo della cfc di un vertice

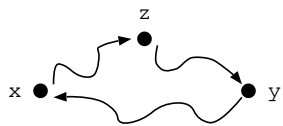
- la complessità è $O(|V| + |E|)$:
  1. è sufficente una visita a partire da x marcando i vertici raggiunti (costo: $O(|V| + |E|)$)
  2. è sufficente calcolare il grafo trasposto (G^T) di G (tutti gli archi invertiti, costo: $O(|V| + |E|)$) ed effettuare di nuovo una visita a partire da x marcando i vertici raggiunti (costo: $O(|V| + |E|)$)
  3. l'intersezione può essere calcolata durante la 2. visita

2. Calcolo di tutte le cfc

- sia $G = (V, E)$ un grafo orientato
- per ogni vertice x non ancora marcato, calcola la componente fortemente connessa di x marcandone tutti i vertici
- costo: $O(|V|^2 + |V| \cdot |E|)$

3. Algoritmo basato su DFS

- lemma I: se $x \leftrightarrow y$ allora nessun cammino tra essi può uscire dalla loro cfc
- dimostrazione:
  - sia $x \leftrightarrow y$
  - sia z tale che esiste un cammino $x \rightarrow z \rightarrow y$
  - siccome $x \leftrightarrow y$, esiste un cammino $y \rightarrow x$
  - quindi esiste un cammino $z \rightarrow y \rightarrow x$
  - $z \leftrightarrow x$

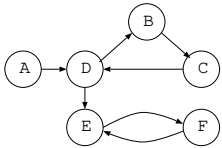


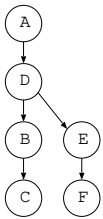
3. Algoritmo basato su DFS

- teorema I: in una qualunque DFS di un grafo G orientato tutti i vertici di una cfc vengono collocati nello stesso albero
- dimostrazione:
  - sia r il primo vertice scoperto di una data cfc
  - al momento della scoperta di r, tutti gli altri vertici della cfc sono bianchi
  - tutti i nodi raggiungibili da r saranno discendenti di r nell'albero DFS

3. Algoritmo basato su DFS

- un esempio: visitiamo il grafo a partire da A

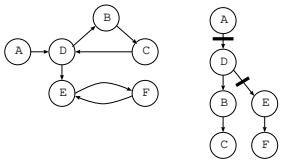




- 3 cfc: $\{A\}$,$\{D, B, C\}$,$\{E, F\}$
- nello stesso albero ci sono nodi di più cfc
- con una visita a partire dal nodo B la foresta di scoperta contiene due alberi di cui il primo contiene i nodi B,C,D,E e F (e quindi contiene nodi di 2 cfc) e il secondo solo il nodo A

3. Algoritmo basato su DFS

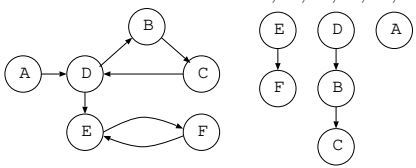
- gli alberi della foresta di scoperta si possono potare in modo da separare le cfc



- questo è sempre possibile:
  - sia u discendente di v in un albero della DFS e $u \nleftrightarrow v$
  - discendenti di u non possono appartenere al cfc di v
  - per assurdo, se k è discendente di u e $k \leftrightarrow v$ allora esistono i cammini $v \rightarrow u$ (u è discendente di v) e anche $u \rightarrow v$ (tramite k che è discendente u ed e mutuamente raggiungibile con v) e quindi non è vero che $u \nleftrightarrow v$ ↯

3. Algoritmo basato su DFS

- basterebbe trovare un ordine “giusto” per quanto riguarda la scelta del nodo bianco da dove fare (ri)partire la visita per ottenere una foresta di scoperta in cui ogni albero corrisponde ad una cfc
- ordine giusto per scegliere fra i nodi bianchi: E, F, D, B, C, A



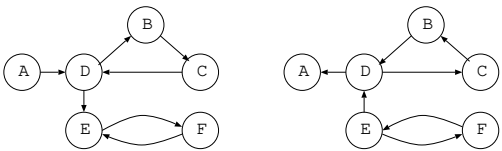
- ma non conosciamo l’ordine “giusto”

3. Algoritmo basato su DFS

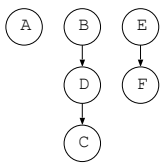
- gli alberi implicano l'esistenza di cammini in una direzione (se u è discendente di v in un albero allora nel grafo esiste un cammino da v ad u)
- dobbiamo verificare l'esistenza di cammini nell'altra direzione
- idea naturale: utilizziamo il grafo trasposto

3. Algoritmo basato su DFS

- grafo G e suo trasposto G^T:

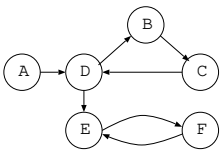


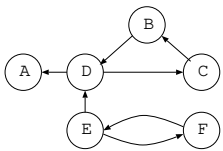
- DFS del grafo trasposto col vertici in ordine alfabetico per quanto riguarda la scelta del nodo bianco da dove fare (ri)partire la visita:



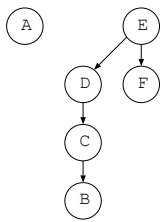
3. Algoritmo basato su DFS

- grafo e suo trasposto:





- naturalmente anche per il grafo trasposto esiste un ordine non “giusto”: A, E, B, C, D, F

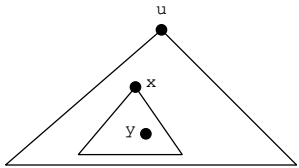


3. Algoritmo basato su DFS

- idea continua: troviamo l'ordine (sempre per quanto riguarda la scelta del bianco) della visita del grafo trasposto sfruttando informazioni ottenuti durante la prima visita
- dati due nodi x e y, quale visitare per primo nel grafo trasposto?
- assumiamo $x \nleftrightarrow y$ e dunque vogliamo che i due nodi finiscano in due alberi distinti della foresta
- dopo la prima DFS due casi sono possibili:
  1. x è discendente di y in un albero della foresta (o viceversa: y è discendente di x)
  2. x e y non sono discendenti uno dell'altro (sono in alberi distinti o su rami distinti)

3. Algoritmo basato su DFS

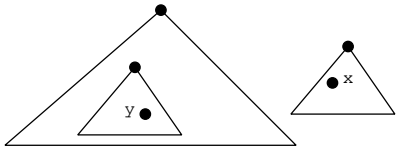
- consideriamo il caso 1
- y è discendente di x in un albero della foresta:



- esiste un cammino da x a y
- se $x \leftrightarrow y$ allora
  - non esiste cammino da y a x in G
  - non esiste cammino da x a y in G^T

3. Algoritmo basato su DFS

- consideriamo il caso 2
- x e y non sono discendenti uno dell'altro



- può esistere un cammino da x a y (archi di attraversamento)
- se $x \leftrightarrow y$ allora
  - non esiste cammino da y a x in G
  - non esiste cammino da x a y in G^T

3. Algoritmo basato su DFS

- in entrambi i casi conviene iniziare la visita di G^T da x perchè, se i vertici non sono nella stessa cfc, non saranno collocati nello stesso albero
- sembra che i vertici nella visita di G^T debbano essere considerati:
  - dall'alto verso il basso
  - da destra verso sinistra

3. Algoritmo basato su DFS

- gli intervalli di attivazione dei due vertici nei due casi:
  1. $x.d < y.d < y.f < x.f$
  2. $y.d < y.f < x.d < x.f$
- in entrambi i casi $x.f > y.f$: i vertici vanno considerati in ordine decrescente di tempo di fine visita

3. Algoritmo basato su DFS

- dalle osservazioni precedenti ricaviamo l'algoritmo:
  1. visita G in profondità preparando una lista dei vertici in ordine decrescente dei tempi di fine visita
  2. costruisci G^T
  3. visita G^T in profondità considerando i vertici secondo la lista restituita dal passo 1 per quanto riguarda la scelta del nodo bianco da dove fare (ri)partire la visita
- complessità: $O(|V| + |E|)$

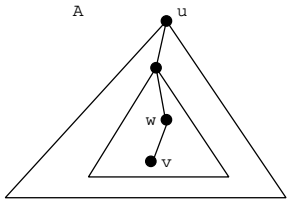
3. Dimostrazione della correttezza

- per dimostrare la correttezza:
  - teorema I: in una qualunque DFS di un grafo G orientato tutti i vertici di una cfc vengono collocati in uno stesso albero (già dimostrato)
  - lemma II: un grafo e il suo trasposto hanno le stesse cfc (dimostrazione immediata)
  - lemma III:
    - sia A un albero ottenuto con la visita in profondità di G^T considerando i vertici in ordine decrescente dei tempi di fine visita su G
    - sia u la radice di A
    - allora per ogni vertice v discendente di u: $u \leftrightarrow v$

3. Dimostrazione di lemma III

- dimostreremo che ogni discendente di u in A è anche discendente di u in un albero della foresta generata dalla visita in profondità di G
- la dimostrazione è fatta per assurdo
- consideriamo un cammino sull'albero A a partire dalla radice u
- sia v il primo vertice sul cammino per cui il lemma non vale
- sia w il predecessore di v sul cammino

3. Dimostrazione di lemma III



- v è il primo per cui il lemma non vale
- l'enunciato vale per w, quindi: $u.d \leq w.d < w.f \leq u.f$ (w può essere anche u)
- la visita di G^T considera i vertici in ordine decrescente di fine visita, quindi per ogni discendente di u in A, e quindi anche per v vale $v.f < u.f$

3. Dimostrazione di lemma III

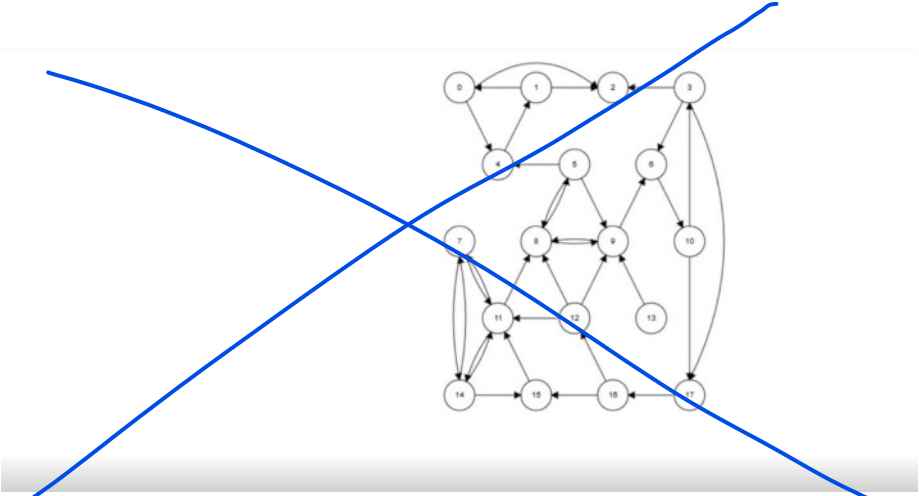
- per il teorema delle parentesi, se v non è discendente di u nella prima visita, i due intervalli di visita di u e v devono essere distinti: $v.d < v.f < u.d < u.f$
- e quindi: $v.d < v.f < u.d \leq w.d < w.f \leq u.f$
- ma questo non è possibile:
  - in G^T esiste l'arco (w, v)
  - in G esiste l'arco (v, w)
  - la visita di v non può terminare prima che sia iniziata la visita di un suo adiacente, cioè $v.f$ non può precedere $w.d$

3. Dimostrazione di lemma III

- concludiamo la dimostrazione del lemma III
  - v è discendente di u in A
  - quindi esiste un cammino da v a u in G
  - abbiamo dimostrato che esiste anche un cammino da u a v in G
  - quindi $u \leftrightarrow v$
- per dimostrare lemma III era essenziale che la visita DFS di G^T consideri i vertici in ordine decrecsente di fine visita per quanto riguarda la scelta del nodo bianco da dove fare (ri)partire la visita

3. Dimostrazione della correttezza del algoritmo

- i lemmi permettono di dimostrare che ogni albero della foresta ottenuta con la visita in profondità di G^T contiene:
  - tutti i vertici di una cfc di G (teorema I e lemma II)
  - solo i vertici di una cfc di G (lemma III)
- **l'algoritmo è corretto**


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

```
CC #1
Vertex 13   DFS(13)
CC #2
Vertex 3    DFS(3)
Vertex 10     DFS(10)
Vertex 6      DFS(6)
Vertex 9        DFS(9)
Vertex 5          DFS(5)
Vertex 8            DFS(8)
Vertex 11             DFS(11)
Vertex 7                DFS(7)
Vertex 14                 DFS(14)
Vertex 12               DFS(12)
Vertex 16                 DFS(16)
Vertex 17                   DFS(17)
Vertex 15               DFS(15)
CC #3
Vertex 0    DFS(0)
Vertex 1      DFS(1)
Vertex 4        DFS(4)
CC #4
Vertex 2    DFS(2)
```