

## Grafi (III parte)

Progettazione di Algoritmi a.a. 2022-23

Matricole congrue a 1

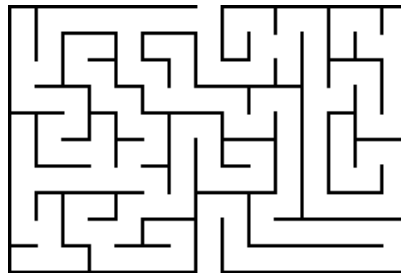
Docente: Annalisa De Bonis

1

1

### Depth first search (visita in profondità)

- La visita in profondità riproduce il comportamento di una persona che esplora un labirinto di camere interconnesse
- La persona parte dalla prima camera (nodo  $s$ ) e si sposta in una delle camere accessibili dalla prima (nodo adiacente ad  $s$ ), di lì si sposta in una delle camere accessibili dalla seconda camera visitata e così via fino a quando raggiunge una camera da cui non è possibile accedere a nessuna altra camera non ancora visitata. A questo punto torna nella camera precedentemente visitata e di lì prova a raggiungere nuove camere.



2

2

### Depth first search (visita in profondità)

- La visita DFS parte dalla sorgente  $s$  e si spinge in profondità fino a che non è più possibile raggiungere nuovi nodi.
  - La visita parte da  $s$ , segue uno degli archi uscenti da  $s$  ed esplora il vertice  $v$  a cui porta l'arco.
  - Una volta in  $v$ , se c'è un arco uscente da  $v$  che porta in un vertice  $w$  non ancora esplorato allora l'algoritmo esplora  $w$
  - Una volta in  $w$  segue uno degli archi uscenti da  $w$  e così via fino a che non arriva in un nodo del quale sono già stati esplorati tutti i vicini.
  - A questo punto l'algoritmo fa **backtrack** (torna indietro) fino a che torna in un vertice a partire dal quale può visitare un vertice non ancora esplorato in precedenza.

PROGETTAZIONE DI ALGORITMI a.a. 2022-23  
A. DE BONIS

3

3

### Depth first search: pseudocodice

DFS( $u$ ):

```

Mark  $u$  as "Explored" and add  $u$  to  $R$ 
For each edge  $(u, v)$  incident to  $u$ 
  If  $v$  is not marked "Explored" then
    Recursively invoke DFS( $v$ )
  Endif
Endfor

```

$R$  = insieme dei vertici raggiunti

Analisi: (assumendo  $G$  rappresentato con liste di adiacenza)

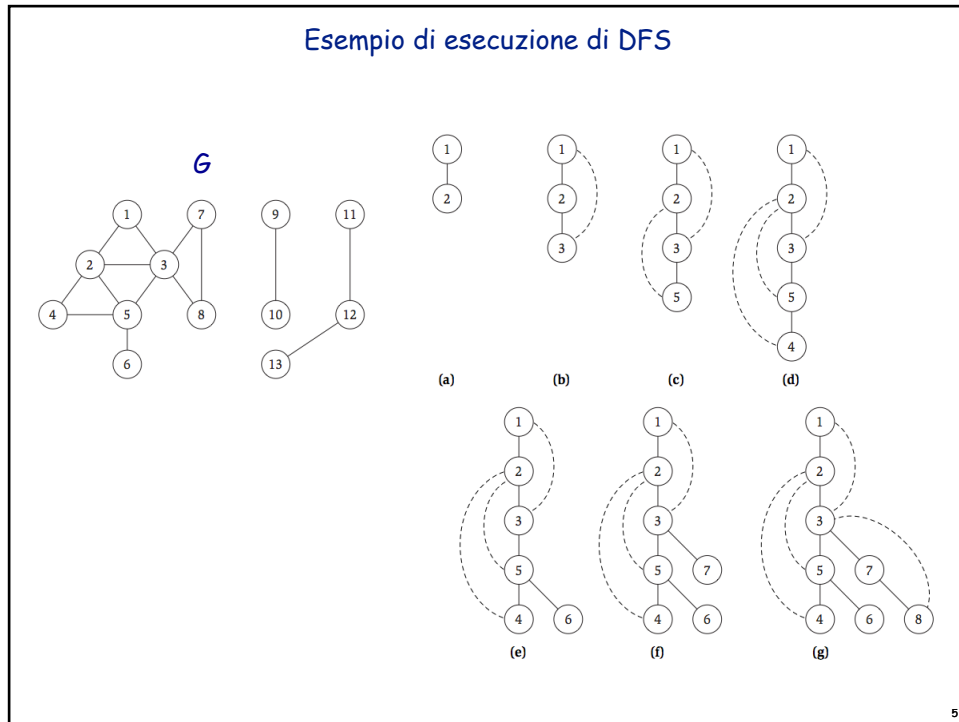
- Se ignoriamo il tempo delle chiamate ricorsive al suo interno, ciascuna visita ricorsiva richiede tempo  $O(1 + \text{deg}(u))$ :  $O(1)$  per marcare  $u$  e aggiungerlo ad  $R$  e  $O(\text{deg}(u))$  per eseguire il for.
- Se inizialmente invochiamo DFS su un nodo  $s$ , allora DFS viene invocata ricorsivamente su tutti i nodi raggiungibili a partire da  $s$ . Il costo totale è quindi al più

$$\begin{aligned} \sum_{u \in V} O(1 + \text{deg}(u)) &= O(\sum_{u \in V} 1 + \sum_{u \in V} \text{deg}(u)) \\ &= O(n + m) \end{aligned}$$

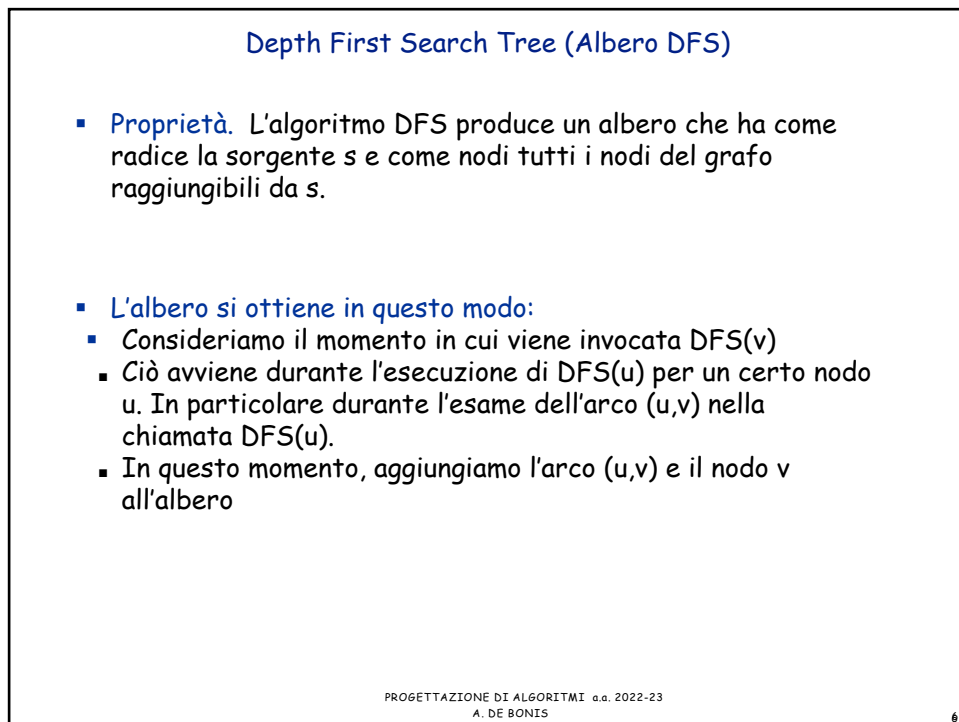
4

4

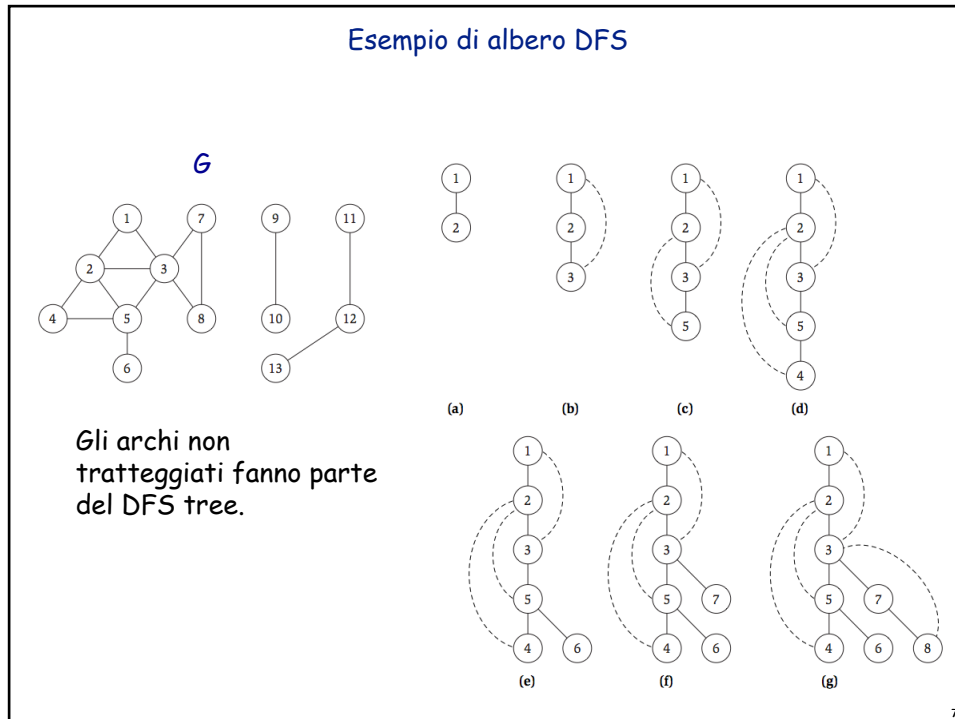
4



5



6



7

**Albero DFS**

- **Proprietà 1.** Per una data chiamata ricorsiva  $DFS(u)$ , tutti i nodi che vengono etichettati come "Esplorati" tra l'inizio e la fine della chiamata  $DFS(u)$ , sono discendenti di  $u$  nell'albero DFS.

**Dim. Proprietà 1.**

- Sia  $x$  un nodo esplorato tra l'inizio e la fine della chiamata  $DFS(u)$
- La dimostrazione è per induzione sul numero  $m$  di chiamate ricorsive iniziate dopo l'inizio di  $DFS(u)$  e non ancora terminate **quando viene invocata**  $DFS(x)$  (esclusa  $DFS(u)$  e inclusa  $DFS(x)$ ).
- **Base.  $m=1$**  → Una sola chiamata cominciata dopo l'inizio di  $DFS(u)$  e che si deve ancora concludere nel momento in cui esploriamo  $x$  → questa chiamata è proprio  $DFS(x)$  →  $DFS(x)$  è invocata nel foreach di  $DFS(u)$  e in questo caso  $x$  diventa figlio di  $u$  → la proprietà è soddisfatta (es.  $u=5, x=6$  in slide precedente)
- **Passo induttivo.** Supponiamo vera la proprietà fino ad  $m-1 \geq 1$  e dimostriamo che è vera per  $m$ . Siccome  $m \geq 2$  → ci deve essere almeno una chiamata a DFS che non è ancora terminata prima che venga invocata  $DFS(x)$  →  $DFS(x)$  non è invocata nel foreach di  $DFS(u)$ . Infatti, se  $DFS(x)$  venisse invocata nel foreach di  $DFS(u)$  allora in quel momento sarebbero già terminate tutte le chiamate sui nodi adiacenti ad  $u$  esaminati prima di  $x$  e tutte le chiamate da esse innescate e sarebbe  $m=1$ .
- Sia  $z$  il nodo adiacente ad  $x$  per cui si ha che  $DFS(z)$  invoca  $DFS(x)$ . **Questo vuol dire dopo l'inizio di  $DFS(u)$  fino al momento in cui viene invocata  $DFS(z)$  sono state effettuate al più  $m-1$  chiamate ricorsive.** Possiamo quindi applicare l'ipotesi induttiva e dire che  $z$  è discendente di  $u$ . Siccome  $x$  è figlio di  $z$  allora  $x$  è anch'esso discendente di  $u$ .

PROGETTAZIONE DI ALGORITMI a.a. 2022-23  
A. DE BONIS

8

8

### Albero DFS

Questa proprietà vale solo se il grafo è non direzionato.

- **Proprietà 2.** Sia  $T$  un albero DFS e siano  $x$  e  $y$  due nodi di  $T$  collegati dall'arco  $(x,y)$  in  $G$ . Si ha che  $x$  e  $y$  sono l'uno antenato dell'altro in  $T$ .

#### Dim. Proprietà 2

- **Caso  $(x,y)$  e' in  $T$ .** In questo caso la proprietà e' ovviamente soddisfatta.
- **Caso  $(x,y)$  non e' in  $T$ .** Supponiamo senza perdere di generalità che  $DFS(x)$  venga invocata prima di  $DFS(y)$ . Ciò vuol dire che quando viene invocata  $DFS(x)$ ,  $y$  non è ancora etichettato come "Esplorato".
- La chiamata  $DFS(x)$  esamina l'arco  $(x,y)$  e per ipotesi non inserisce  $(x,y)$  in  $T$ . Ciò si verifica solo se  $y$  è già stato etichettato come "Esplorato". Siccome  $y$  non era etichettato come "Esplorato" all'inizio di  $DFS(x)$  vuol dire è stato esplorato tra l'inizio e la fine della chiamata  $DFS(x)$ . La proprietà 1 implica che  $y$  è discendente di  $x$ .

PROGETTAZIONE DI ALGORITMI a.a. 2022-23  
A. DE BONIS

9

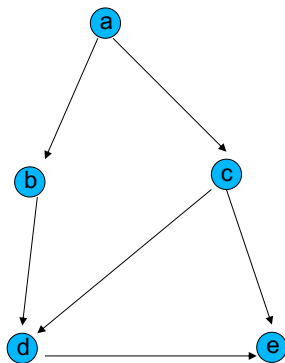
9

### Albero DFS

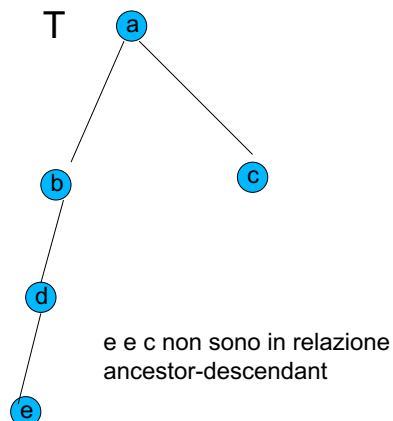
Facciamo vedere che la proprietà 2 non vale in generale per i grafi direzionati.

Usiamo un controesempio:

G



T



PROGETTAZIONE DI ALGORITMI a.a. 2022-23  
A. DE BONIS

10

10

### Implementazione di DFS mediante uno stack

#### DFS(s):

1. Poni Explored[s] = true ed Explored[v] = false per tutti gli altri nodi
2. Inizializza S con uno stack contenente s
3. **While** S non è vuoto
4.     Metti in u il nodo al top di S
5.     **If** c'è un arco (u, v) incidente su u non ancora esaminato **then**
6.         **If** Explored[v] = false **then**
7.             Poni Explored[v] = true
8.             Inserisci v al top di S
9.         **Endif**
10.     **Else** // tutti gli archi incidenti su u sono stati esaminati
11.     Rimuovi il top di S
12.     **Endif**
13. **Endwhile**

- Per implementare la linea 6 in modo efficiente possiamo mantenere per ogni vertice u un puntatore al nodo della lista di adiacenza di u corrispondente al prossimo arco (u,v) da scandire.
- Si noti che un nodo u rimane nello stack fino a che non vengono scanditi tutti gli archi incidenti su u.

11

11

### Analisi di DFS implementata mediante uno stack

Assumiamo G rappresentato con liste di adiacenza

#### DFS(s):

1. Poni Explored[s] = true ed Explored[v] = false per tutti gli altri nodi  $O(n)$
2. Inizializza S con uno stack contenente s  $O(1)$
3. **While** S non è vuoto
4.     Metti in u il nodo al top di S
5.     **If** c'è un arco (u, v) incidente su u non ancora esaminato **then**
6.         **If** Explored[v] = false **then**
7.             Poni Explored[v] = true
8.             Inserisci v al top di S
9.         **Endif**
10.     **Else** // tutti gli archi incidenti su u sono stati esaminati
11.     Rimuovi il top di S
12.     **Endif**  $O(n+m)$
13. **Endwhile**

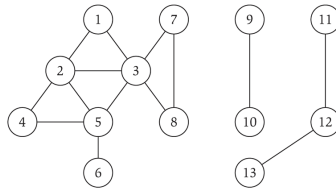
- **Analisi linee 3-13:** Il while viene iterato  $\deg(v)+1$  volte per ogni nodo v inserito in S: ogni volta che v viene a trovarsi al top dello stack viene esaminato uno dei suoi archi non ancora esaminati oppure se non esiste un tale arco, v viene rimosso dallo stack. Vengono quindi effettuate  $\deg(v)$  iterazioni del while prima di quella in cui v viene rimosso dallo stack  $\rightarrow$  in totale il while è iterato un numero di volte pari al più a  $\sum_{v \in V} (\deg(v) + 1) = \sum_{v \in V} \deg(v) + \sum_{v \in V} 1 \leq 2m + n$
- Se manteniamo traccia del prossimo arco da scandire (vedi slide precedente), la linea 6 richiede tempo  $O(1)$ . Di conseguenza il corpo del while richiede  $O(1)$  per ogni iterazione  $\rightarrow$  tempo totale per tutte le iterazioni  $O(2m+n)=O(m+n)$ .

12

12

### Componente connessa

- **Componente connessa.** Sottoinsieme di vertici tale per ciascuna coppia di vertici  $u$  e  $v$  esiste un percorso tra  $u$  e  $v$
- **Componente connessa contenente  $s$ .** Formata da tutti i nodi raggiungibili da  $s$



- Componente connessa contenente il nodo 1 è  $\{1, 2, 3, 4, 5, 6, 7, 8\}$ .

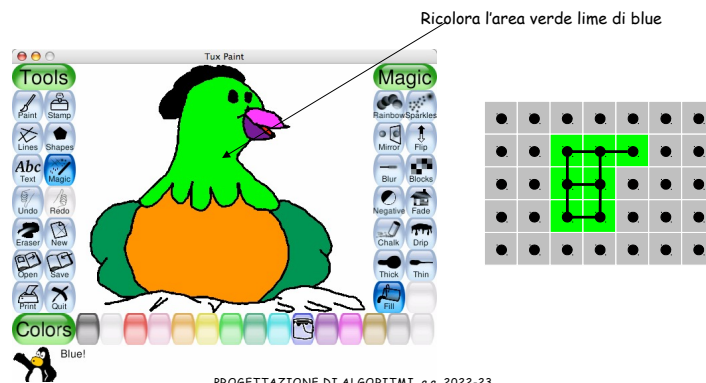
PROGETTAZIONE DI ALGORITMI a.a. 2022-23  
A. DE BONIS

13

13

### Flood Fill

- **Flood fill.** Data un'immagine, cambia il colore dell'area di pixel vicini di colore verde lime in blu.
  - Nodo: pixel.
  - Arco: due pixel vicini di colore verde lime.
  - Area di pixel vicini di colore verde lime: componente connessa di nodi associati a pixel verde lime.



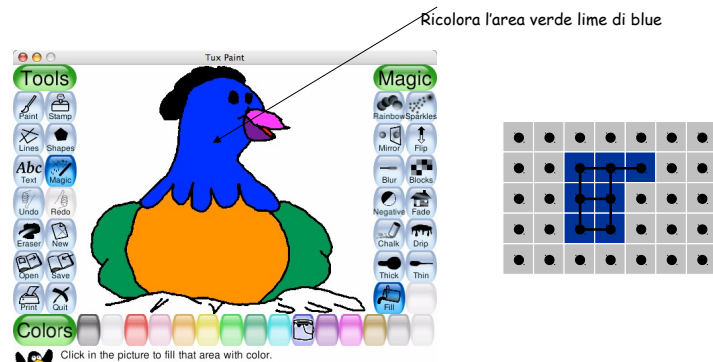
PROGETTAZIONE DI ALGORITMI a.a. 2022-23  
A. DE BONIS

14

14

### Flood Fill

- **Flood fill.** Data un'immagine, cambia il colore dell'area di pixel vicini di colore verde lime in blu.
- **Nodo:** pixel.
- **Arco:** due pixel vicini di colore verde lime.
- **Area di pixel vicini:** componente connessa di pixel di colore verde lime.



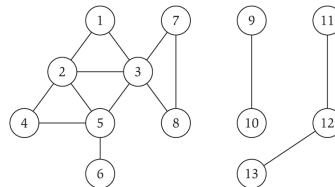
PROGETTAZIONE DI ALGORITMI a.a. 2022-23  
A. DE BONIS

15

15

### Componente connessa

- **Componente connessa contenente s.** Trova tutti i nodi raggiungibili da s
  - **Come trovarla.** Esegui BFS o DFS utilizzando s come sorgente
- **Insieme di tutte le componenti connesse.** Trova tutte le componenti connesse
  - **Come trovarlo.** Fino a quando ci sono nodi che non sono stati scoperti (esplorati), scegli uno di questi nodi ed esegui BFS (o DFS) su u utilizzando questo nodo come sorgente



Esempio: il grafo sottostante ha tre componenti connesse

PROGETTAZIONE DI ALGORITMI a.a. 2022-23  
A. DE BONIS

16

16



### Insieme di tutte componenti connesse

- **Teorema.** Per ogni due nodi  $s$  e  $t$  di un grafo, le loro componenti connesse o sono uguali o disgiunte
- **Dim.**
- **Caso 1.** Esiste un percorso tra  $s$  e  $t$ . In questo caso ogni nodo  $u$  raggiungibile da  $s$  è anche raggiungibile da  $t$  (basta andare da  $t$  ad  $s$  e da  $s$  ad  $u$ ) e ogni nodo  $u$  raggiungibile da  $t$  è anche raggiungibile da  $s$  (basta andare da  $s$  ad  $t$  e da  $t$  ad  $u$ ). Ne consegue che un nodo  $u$  è nella componente connessa di  $s$  **se e solo se** è anche in quella di  $t$  e quindi le componenti connesse di  $s$  e  $t$  sono uguali.
- **Caso 2.** Non esiste un percorso tra  $s$  e  $t$ . In questo caso non può esserci un nodo che appartiene sia alla componente connessa di  $s$  che a quella di  $t$ . Se esistesse un tale nodo  $v$  questo sarebbe raggiungibile sia da  $s$  che da  $t$  e quindi potremmo andare da  $s$  a  $v$  e poi da  $v$  ad  $t$ . Ciò contraddice l'ipotesi che non c'è un percorso tra  $s$  e  $t$ .

PROGETTAZIONE DI ALGORITMI a.a. 2022-23  
A. DE BONIS

17

17

### Insieme di tutte componenti connesse

- Il teorema precedente implica che le componenti connesse di un grafo sono a due a due disgiunte.
- Algoritmo per trovare l'insieme di tutte le componenti connesse

AllComponents( $G$ )

Per ogni nodo  $u$  di  $G$  setta discovered[ $u$ ]=false

For each node  $u$  of  $G$

  If Discovered[ $u$ ] = false

    BFS( $u$ )

  Endif

Endfor

- BFS modificata in modo tale che nella fase di inizializzazione non vengano settati a False le entrate dell'array Discovered
- **Al posto della BFS possiamo usare la DFS e al posto dell'array Discovered l'array Explored**

PROGETTAZIONE DI ALGORITMI a.a. 2022-23  
A. DE BONIS

18

18

### Insieme di tutte componenti connesse: analisi

- Indichiamo con  $k$  il numero di componenti connesse
- Indichiamo con  $n_i$  e con  $m_i$ , rispettivamente il numero di nodi e di archi della componente  $i$ -esima
- L'esecuzione della visita BFS o DFS sulla componente  $i$ -esima richiede tempo  $O(n_i+m_i)$
- Il tempo totale richiesto da tutte le visite BFS o DFS e`

$$\sum_{i=1}^k O(n_i+m_i) = O\left(\sum_{i=1}^k (n_i+m_i)\right)$$

- Poiche' le componenti sono a due a due disgiunte, si ha che

$$\sum_{i=1}^k (n_i+m_i) = n+m$$

- e il tempo totale di esecuzione dell'algoritmo che scopre le componenti connesse e`  $O(n)+O(n+m)=O(n+m)$

19

### Insieme di tutte componenti connesse: alcune considerazioni

- Se l'algoritmo utilizza BFS allora BFS deve essere modificata in modo che non resetti a false ogni volta i campi discovered.
- E` possibile modificare AllComponents in modo che assegni a ciascun nodo la componente di cui fa parte. A questo scopo usiamo:
  - contatore delle componenti.
  - array Component t.c.  $\text{Component}[u] = j$  se  $u$  appartiene alla componente  $j$ -esima.
- **Esercizio:** modificare lo pseudocodice dell'algoritmo AllComponents in modo che assegni a ciascun nodo la componente di cui fa parte. Ricordatevi che occorre modificare anche l'algoritmo di visita invocato da AllComponents.

20