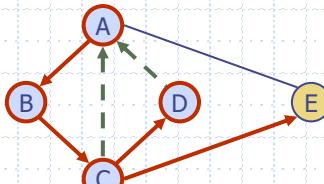


Se il grafo è connesso riesce a visitare l'intero grafo.

Dobbiamo esplorare anch'essi entrambi di passare più volte. Cosa fare noi.

Depth-First Search

Ora non tutti gli algoritmi sono
traversi.



Cerco di seguire tutti i percorsi possibili

Tecnica generale perché può essere specializzata per altre cose

Depth-First Search

- Depth-first search (DFS) is a **general technique** for **traversing a graph**
- A DFS traversal of a graph G
 - Visits all the vertices and edges of G
 - Determines whether G is **connected**
 - Computes the **connected components** of G
 - Computes a **spanning forest** of G
- DFS on a graph with n vertices and m edges **takes $O(n + m)$ time***
- DFS can be further **extended to solve other graph problems**
 - Find and report a path between two given vertices
 - Find a cycle in the graph
- Depth-first search is to graphs what Euler tour is to binary trees

Pero in modo che
sto visitando, devo
fare elenco di tutti gli
archi e li deve scorrere.
Questo però ogni modo.

Se devo un modo mai
avendo ad alcuno, faccio
dipartire algoritmo su
quelli.

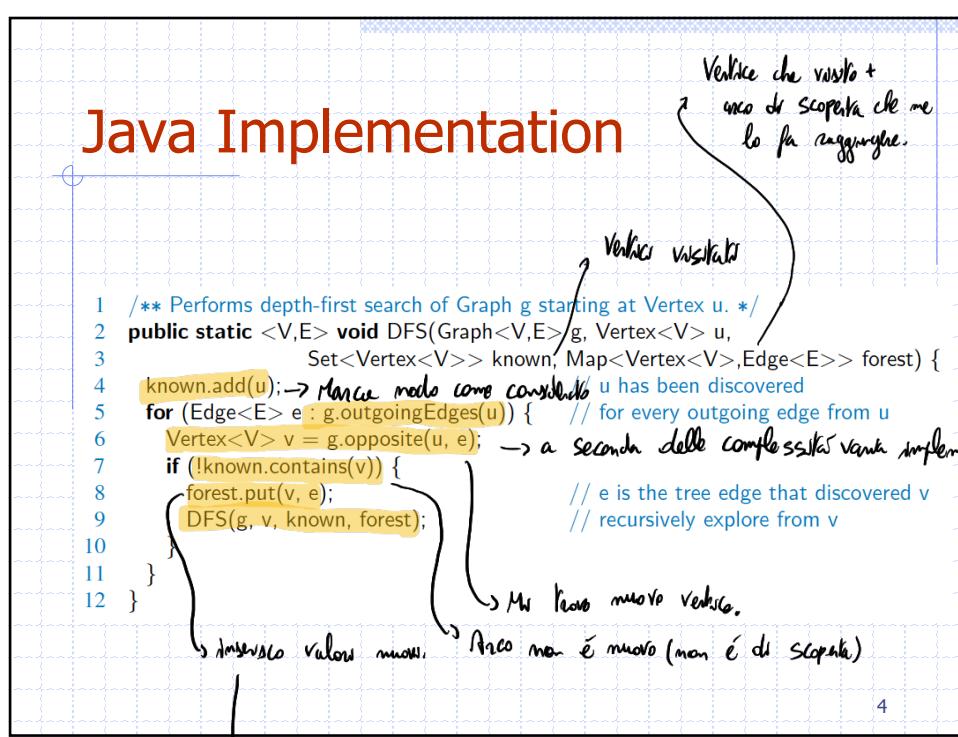
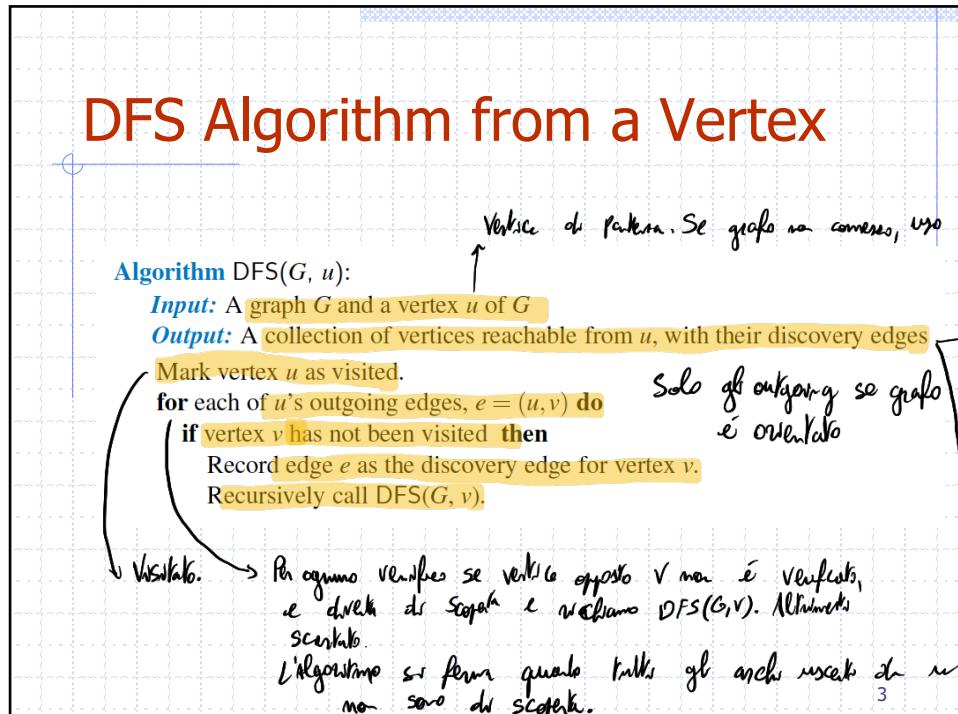
*Vengono esaminati
tutti i archi e
i nodi!
↳ Però!

Potrei implementare la marcatura nella struttura ma è poco efficiente.

Nota: Posso scegliere ordine con cui passare archi a DFS: 2 risultati diversi. Dipende da implementaz. di OutgoingEdges.

5/7/19 10:58 AM

A seconda delle rappres. del grafo, tu conoscenza del vertice opposto può essere stessa o no, ma ho metodo opposte.

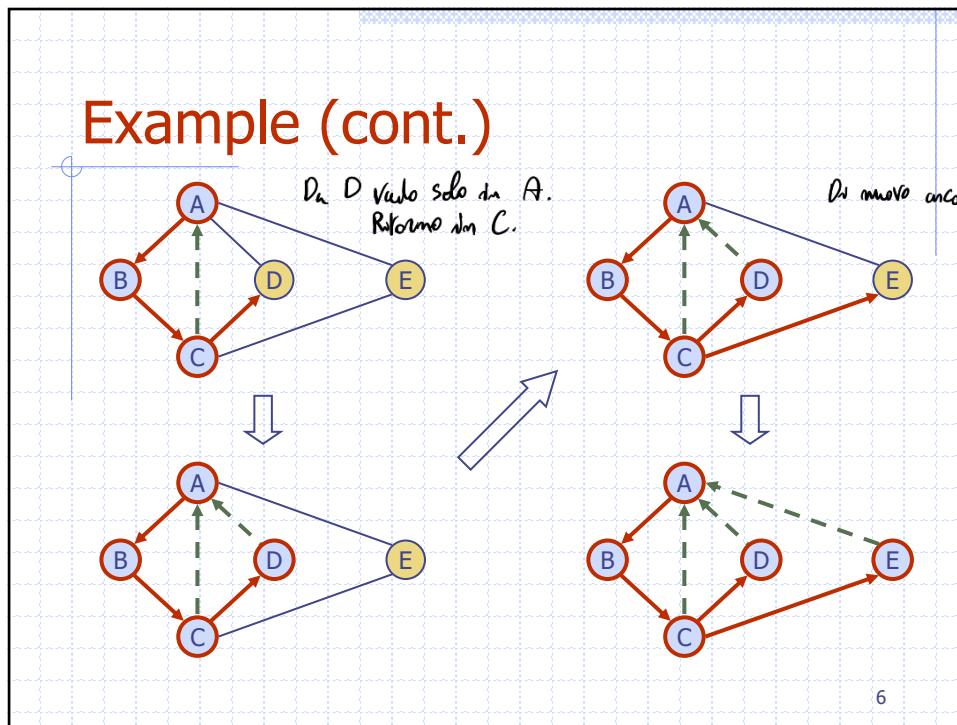
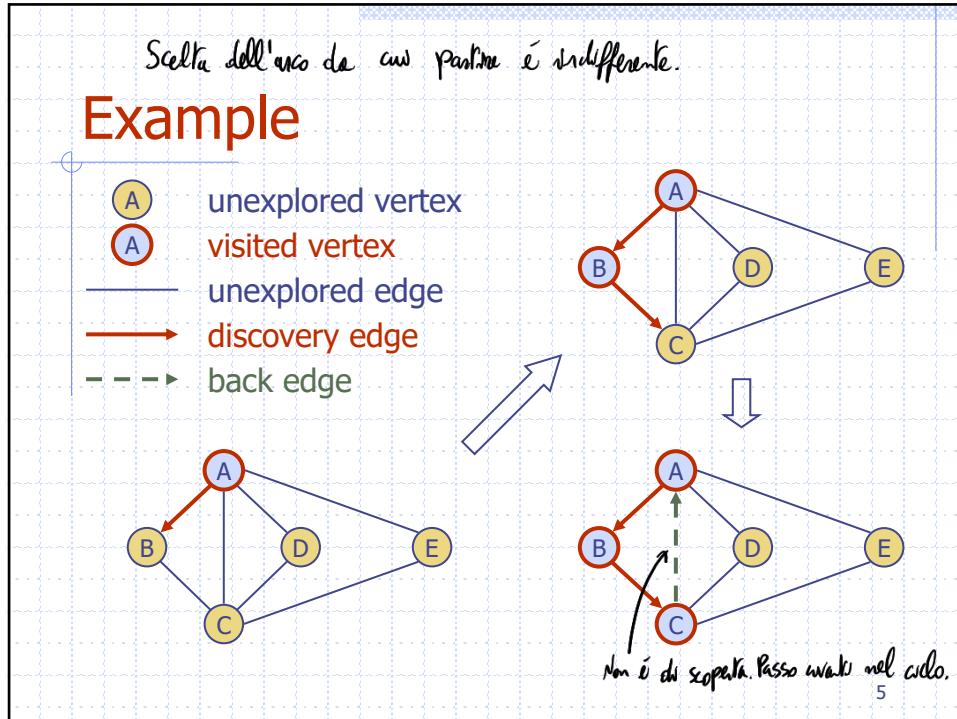


→ Mettere un elemento nella foresta significa marcarlo: unico archio di scoperta, modo obiettivo risultato

Anco di scoperta: anco che durante la visita ci ponci a un vertice non visitato.

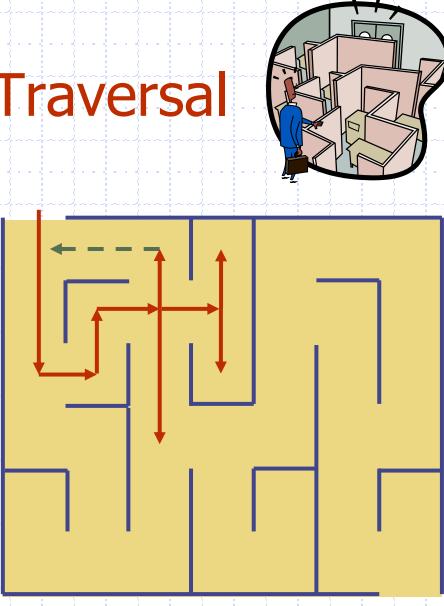
Per mancare grafo si usa una struttura dati aggiuntiva, es. un set che contiene vertici già visitati. Per un vertice che è nel set.

- Oltreogni anco dí solo anchi usciti nell'ordine.
- Se vertice v non é stato ancora visitato, manca anco come anco di scoperta e chiama $DFS(G, v)$.



DFS and Maze Traversal

- The DFS algorithm is similar to a classic strategy for exploring a maze
 - We mark each intersection, corner and dead end (vertex) visited
 - We mark each corridor (edge) traversed
 - We keep track of the path back to the entrance (start vertex) by means of a rope (recursion stack)

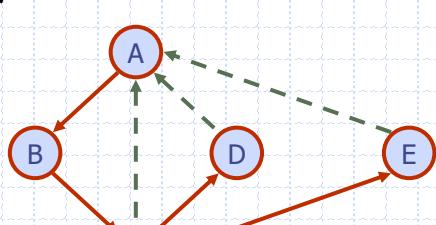


7

Properties of DFS

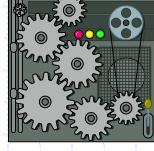
Property 1
 $DFS(G, v)$ visits all the vertices and edges in the connected component of v ✓

Property 2
The discovery edges labeled by $DFS(G, v)$ form a spanning tree of the connected component of v



8

Analysis of DFS



- Setting/getting a vertex/edge label takes $O(1)$ time
- Each vertex is labeled twice
 - once as UNEXPLORED Doppia visita
 - once as VISITED
- Each edge is labeled twice
 - once as UNEXPLORED
 - once as DISCOVERY or BACK
- Method incidentEdges is called once for each vertex
- DFS runs in $O(n + m)$ time provided the graph is represented by the adjacency list structure Così c'è complessità $n+m$

$2m + 2m \leftarrow$
ogni volta;
 $O(n+m)$

→ Sarebbe in realtà
outgoing edges se orientato.

→ grado di uscita
dei capioutari.

Con lista di adiacenza, outgoing edges è $O(\deg(v))$.

9

Non deterministica l'outgoing edges.

Path Finding



- We can specialize the DFS algorithm to find a path between two given vertices u and z using the template method pattern
- We call $DFS(G, u)$ with u as the start vertex
- We use a stack S to keep track of the path between the start vertex and the current vertex
- As soon as destination vertex z is encountered, we return the path as the contents of the stack

```

Algorithm pathDFS( $G, v, z$ )
  setLabel( $v, VISITED$ ) → Punto che  $v$ ,  

   $S.push(v)$  visitato.
  if  $v = z$  →
    return  $S.elements()$  ⇒ Ritorno elenco dei
    for all  $e \in G.incidentEdges(v)$  →
      if getLabel( $e$ ) = UNEXPLORED →
         $w \leftarrow$  opposite( $v, e$ )
        if getLabel( $w$ ) = UNEXPLORED →
          setLabel( $e, DISCOVERY$ ) → Arco di scoperta
           $S.push(e)$  → Faccio push e set
          pathDFS( $G, w, z$ )
           $S.pop(e)$ 
        else →
          setLabel( $e, BACK$ ) → Ritorno ed
           $S.pop(v)$  elenco arco after
          visitato.
    
```

Ma non mi fermo.
Faccio pop e carico con
il elemento già visitato.

Come lo raggiro z .

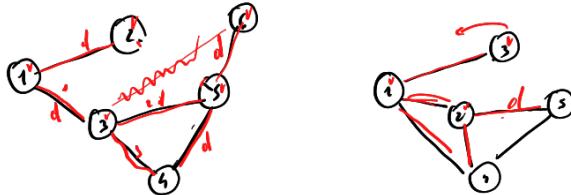
Non esplorato

Arco di scoperta

Ritorno ed
elenco arco after
visitato.

10

In realtà non posso usare stack solo per modi che per archi



Path Finding in Java

```

1  /** Returns an ordered list of edges comprising the directed path from u to v. */
2  public static <V,E> PositionalList<Edge<E>> Anchi che collega vertici.
3  constructPath(Graph<V,E> g, Vertex<V> u, Vertex<V> v,
4                Map<Vertex<V>, Edge<E>> forest) {
5    PositionalList<Edge<E>> path = new LinkedPositionalList<>();
6    if (forest.get(v) != null) { → V c'è! // v was discovered during the search
7      Vertex<V> walk = v; // we construct the path from back to front
8      while (walk != u) {
9        Edge<E> edge = forest.get(walk); → Vertice che fa parte della strada.
10       path.addFirst(edge); // add edge to *front* of path
11       walk = g.opposite(walk, edge); // repeat with opposite endpoint
12     }
13   }
14   return path;
15 }
```

11

Ora prima prendendo la foresta e poi trova il path
⇒ Riuso la foresta per più path.

Cycle Finding

- We can specialize the DFS algorithm to find a simple cycle using the template method pattern
- We use a stack S to keep track of the path between the start vertex and the current vertex
- As soon as a **back edge** (v, w) is encountered, we return the cycle as the portion of the stack from the top to vertex w

Algorithm $\text{cycleDFS}(G, v, z) \rightarrow$ (grafico, vertice di partenza e arrivo).

$S.push(v)$

for all $e \in G.\text{incidentEdges}(v)$

if $\text{getLabel}(e) = \text{UNEXPLORED}$ \rightarrow Arco è sconosciuto.

$w \leftarrow \text{opposite}(v, e)$

$S.push(e)$

if $\text{getLabel}(w) = \text{UNEXPLORED}$

$\text{setLabel}(e, \text{DISCOVERY}) \rightarrow$ Arco di scoperta e lo segno.

cycle $\text{DFS}(G, w, z) \rightarrow$ Ricomincia.

$S.pop(e) \rightarrow$ Non ha trovato nessun ciclo in questo percorso.

else \leftarrow

$T \leftarrow \text{new empty stack}$

repeat

$o \leftarrow S.pop()$

$T.push(o)$

until $o = w$

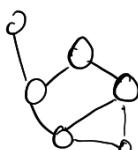
return $T.\text{elements}()$

$S.pop(v) \rightarrow$ butta fuori e ricomincia.

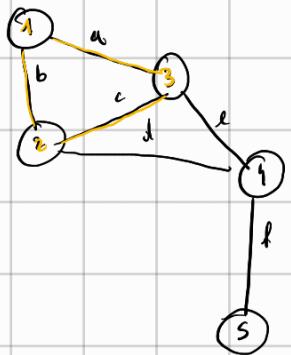
ma modo visitab. Ciclo.
 fino al vertice
 di partenza fino alla
 chiamata ricorsiva.

12

- Dallo un percorso tra V e Z c'è un ciclo?
- Questa è la domanda.



Usa un altro stack.
Non siamo finiti perché potrei avere altri cicli.



STACK S

$\sqrt{1}$
 a
 $\sqrt{3}$
 c
 $\sqrt{2}$
 b

II
 \downarrow

STACK S

-

$$T = b \sqrt{2} c \sqrt{3} a \sqrt{1}$$

Poi si ripete del modo 2 con stack vuoto come se fosse grafo nuovo.

Ma loop 1243 come lo trovi?

DFS for an Entire Graph

- ❑ The algorithm uses a mechanism for setting and getting “labels” of vertices and edges

Algorithm $DFS(G)$

Input graph G

Output labeling of the edges of G as discovery edges and back edges

```

for all  $u \in G.vertices()$ 
    setLabel( $u$ , UNEXPLORED)
for all  $e \in G.edges()$ 
    setLabel( $e$ , UNEXPLORED)
for all  $v \in G.vertices()$ 
    if getLabel( $v$ ) = UNEXPLORED
        DFS( $G, v$ )

```

Scelta casuale ← pm vertex

Algorithm $DFS(G, v)$

Input graph G and a start vertex v of G

Output labeling of the edges of G in the connected component of v as discovery edges and back edges

```

setLabel( $v$ , VISITED)
for all  $e \in G.incidentEdges(v)$ 
    if getLabel( $e$ ) = UNEXPLORED
         $w \leftarrow \text{opposite}(v, e)$ 
        if getLabel( $w$ ) = UNEXPLORED
            setLabel( $e$ , DISCOVERY)
            DFS( $G, w$ )
        else
            setLabel( $e$ , BACK)

```

13

All Connected Components

- ❑ Loop over all vertices, doing a DFS from each unvisited one.

```

1  /** Performs DFS for the entire graph and returns the DFS forest as a map.*/
2  public static <V,E> Map<Vertex<V>,Edge<E>> DFSComplete(Graph<V,E> g) {
3      Set<Vertex<V>> known = new HashSet<>(); Forest
4      Map<Vertex<V>,Edge<E>> forest = new ProbeHashMap<>();
5      for (Vertex<V> u : g.vertices())
6          if (!known.contains(u))
7              DFS(g, u, known, forest); // (re)start the DFS process at u
8      return forest;
9  }

```

14

I livelli non si seguono per tempo.

Vista 2: Non funziona: organizzare grafo direttamente su L_0, L_1, L_2, \dots . Questo è il risultato finale.

Breadth-First Search

Core: non visita partendo da 1 nodo e arrivando alla fine, ma visita in ampiezza. Vista tutti i nodi connessi al nodo di partenza. Poi visita tutti i nodi a partire dal livello successivo.

Punto da A, che ha elenco di archi a cui posso arrivare e molti raggiungibili con 1 solo salto $\Rightarrow L_1$ della BFS. Ora, molti raggiungibili con 1 salto dal livello 1! E F è L_2 . Questi dovranno altri posti raggiungibili? No. Finito.

15

Breadth-First Search

- Breadth-first search (BFS) is a general technique for traversing a graph
- A BFS traversal of a graph G
 - Visits all the vertices and edges of G
 - Determines whether G is connected
 - Computes the connected components of G
 - Computes a spanning forest of G
- BFS on a graph with n vertices and m edges takes $O(n + m)$ time
- BFS can be further extended to solve other graph problems
 - Find and report a path with the minimum number of edges between two given vertices
 - Find a simple cycle, if there is one \rightarrow Ciclo di 3

L'elenco ha solo tutti i punti.

Quindi no.

16

Parecchi punti in comune: $O(n+m)$: etichettatura di archi e nodi. Ogni nodo visitato 2 volte, una volta per effettuando, un'altra quando ci passo.

Qua non ci sono archi di back, ma archi di cross. Non manca posta struttura, ma al massimo al livello attuale o avanti.

Nota: Archi o già marcati da scoperta, o mi portano sullo stesso livello.

5/7/19 10:58 AM

BFS Algorithm

- The algorithm uses a mechanism for setting and getting "labels" of vertices and edges

Algorithm $BFS(G)$

Input graph G

Output labeling of the edges and partition of the vertices of G

```
for all  $u \in G.vertices()$ 
    setLabel( $u$ , UNEXPLORED)
for all  $e \in G.edges()$ 
    setLabel( $e$ , UNEXPLORED)
for all  $v \in G.vertices()$ 
    if getLabel( $v$ ) = UNEXPLORED
        BFS( $G$ ,  $v$ )
```

Algorithm $BFS(G, s)$

$L_0 \leftarrow$ new empty sequence

$L_0.addLast(s)$ Liste che hanno vertice dell'

setLabel(s , VISITED)

$i \leftarrow 0$

while $\neg L_i.isEmpty()$

$L_{i+1} \leftarrow$ new empty sequence

for all $v \in L_i.elements()$

for all $e \in G.incidentEdges(v)$ Non

if getLabel(e) = UNEXPLORED

$w \leftarrow$ opposite(v, e)

if getLabel(w) = UNEXPLORED

setLabel(e , DISCOVERY)

setLabel(w , VISITED)

$L_{i+1}.addLast(w)$

else

setLabel(e , CROSS)

$i \leftarrow i + 1$

Al livello 0 solo
vertice di partenza

è unico
caso.

overall,

17

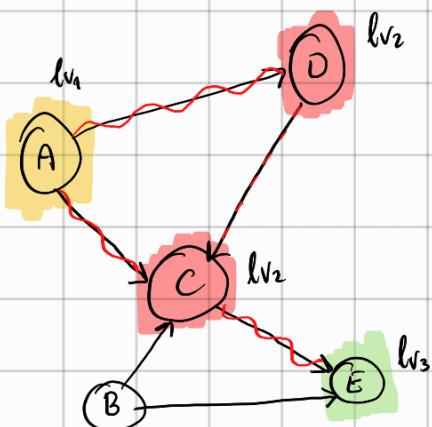
→ L_0 vuoto, controllo dopo $\neg L_i$ se L_1 è vuoto

Java Implementation

```
1  /** Performs breadth-first search of Graph g starting at Vertex u. */
2  public static <V,E> void BFS(Graph<V,E> g, Vertex<V> s,
3                                Set<Vertex<V>> known, Map<Vertex<V>,Edge<E>> forest) {
4      PositionalList<Vertex<V>> level = new LinkedPositionalList<>(); Futura non cambia
5      known.add(s);
6      level.addLast(s); // first level includes only s
7      while (!level.isEmpty()) {
8          PositionalList<Vertex<V>> nextLevel = new LinkedPositionalList<>();
9          for (Vertex<V> u : level)
10             for (Edge<E> e : g.outgoingEdges(u)) {
11                 Vertex<V> v = g.opposite(u, e);
12                 if (!known.contains(v)) {
13                     known.add(v);
14                     forest.put(v, e); // e is the tree edge that discovered v
15                     nextLevel.addLast(v); // v will be further considered in next pass
16                 }
17             }
18         level = nextLevel; → Dovete m // relabel 'next' level to become the current
19     }
20 }
```

Ovvio comunque che sarà vuoto se mai ha fatto nulla.

18



$L_0 = A \Rightarrow$ outgoing edges:
 $L_1 = C, D$
 $L_2 = E$
 $L_3 = \emptyset$
 C non ha solo E, D connette solo u.c.
 Valore in E: u.c.

B rimane insolubile. Riparto da B:

$L_3 = B$ b ha uscite sole per E e C. Ma C ed E già visitati.

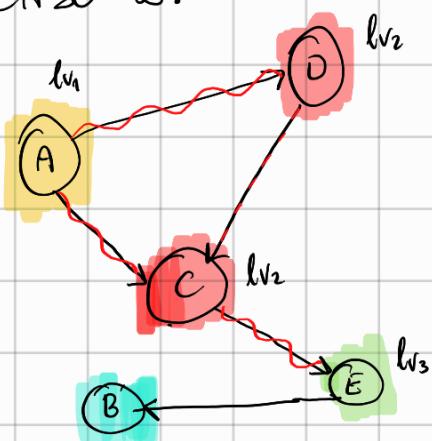
Sono ancora anche da back, già visitati. Non li considero.

Non ho livello 4.

Ho trovato 2 componenti connesse.

Componente connessa sempre relativa a dove si parte.

CASO 2:



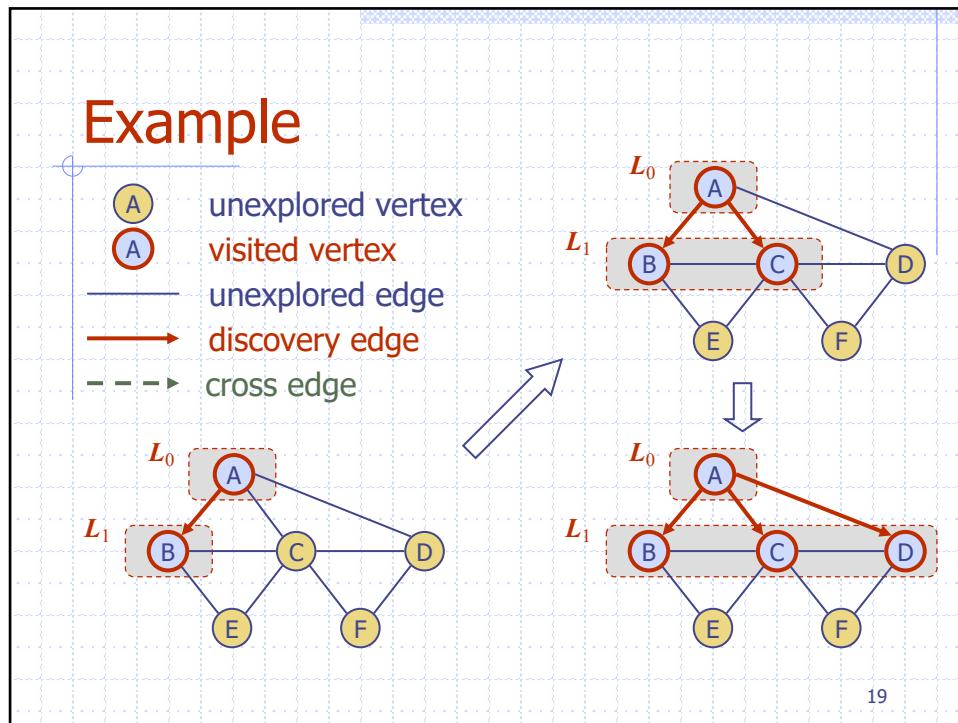
$B \notin L_3$.

Ho una componente connessa che si estende a tutto il grafo, ma il grafo non è connesso.

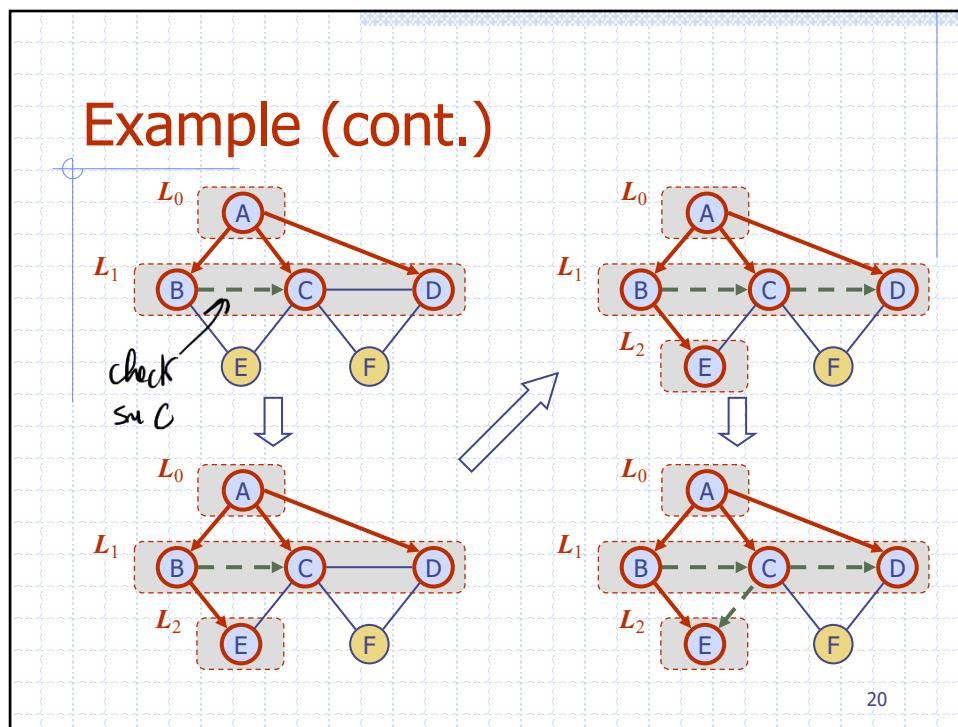
Trovare tutti i vertici a partire da un vertice di partenza \neq connesso.

Come faccio? Trucco: inverti tutti gli archi in un nuovo grafo. Ripeti visita a partire dallo stesso nodo di prima con 1 dei 2 algoritmi. Se ritrovi a tutta la fine i nodi, grafo è FORTEMENTE CONNESSO. Faccio visita 2 volte, cambio grafo ecc. con outgoing edges.

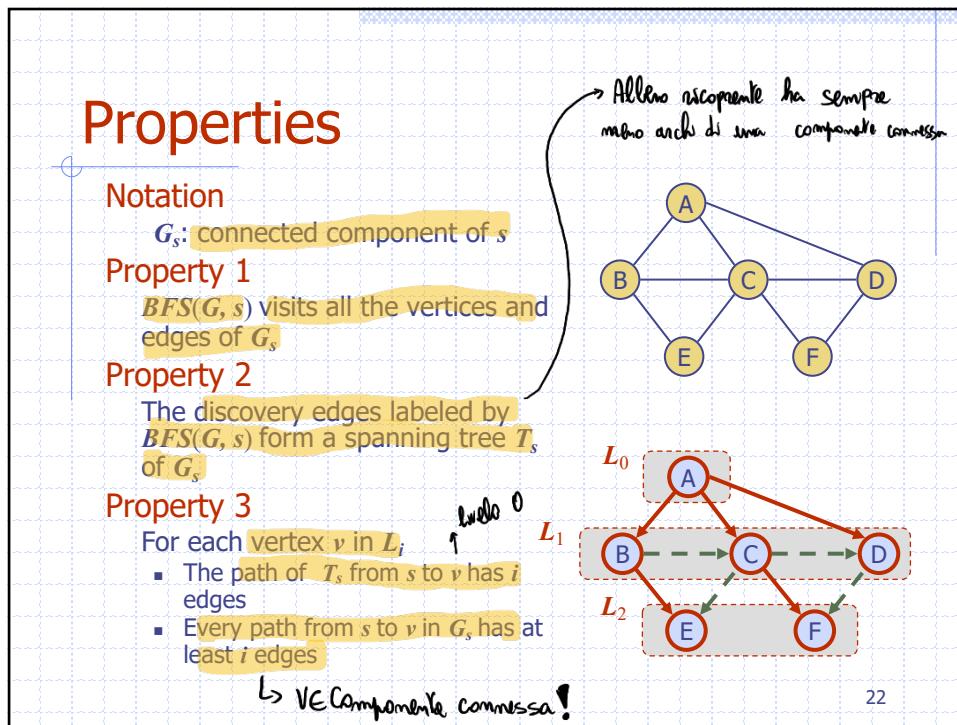
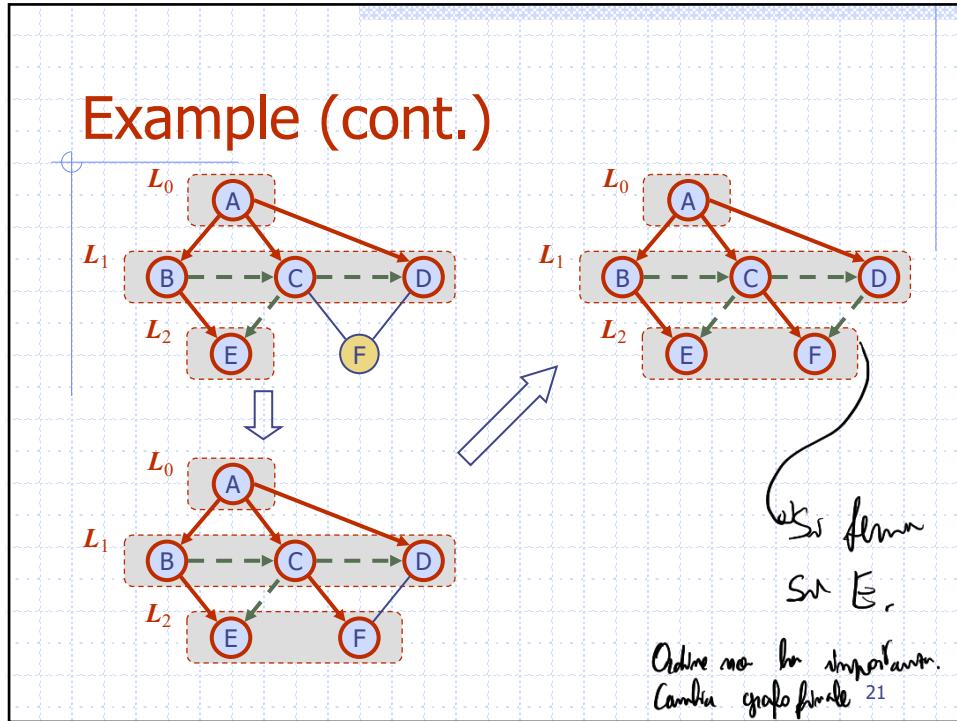
Tra D ed A qual è la diff. di livello? m. Altra con m soluz d'arco. Percorso più breve.



19



20



Analysis

- Setting/getting a vertex/edge label takes $O(1)$ time
- Each vertex is labeled twice
 - once as UNEXPLORED
 - once as VISITED
- Each edge is labeled twice
 - once as UNEXPLORED
 - once as DISCOVERY or CROSS
- Each vertex is inserted once into a sequence L
- Method `incidentEdges` is called once for each vertex
- BFS runs in $O(n + m)$ time provided the graph is represented by the adjacency list structure
 - Recall that $\sum_v \deg(v) = 2m$

al igual del
 supone vertex
 en $O(1)$
 \Rightarrow tiempo m .
 2m

23

Applications

- Using the template method pattern, we can specialize the BFS traversal of a graph G to solve the following problems in $O(n + m)$ time
 - Compute the connected components of G
 - Compute a spanning forest of G
 - Find a simple cycle in G , or report that G is a forest
 - Given two vertices of G , find a path in G between them with the minimum number of edges, or report that no such path exists

24

DFS vs. BFS

Applications	DFS	BFS
Spanning forest, connected components, paths, cycles	✓	✓
Shortest paths		✓
Biconnected components	✗	

DFS

BFS

25

DFS vs. BFS (cont.)

Back edge (v, w)

- w is an ancestor of v in the tree of discovery edges

Cross edge (v, w)

- w is in the same level as v or in the next level

DFS

BFS

26