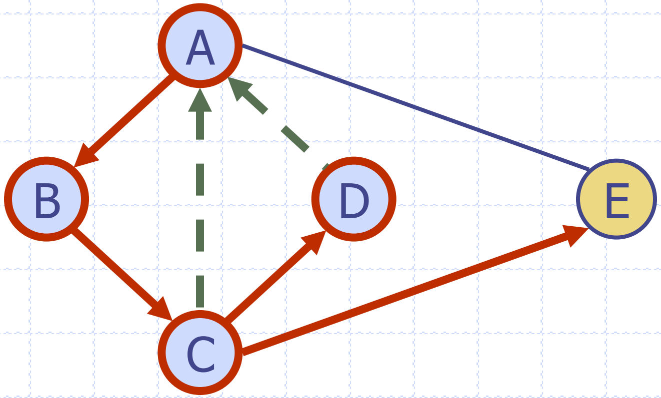
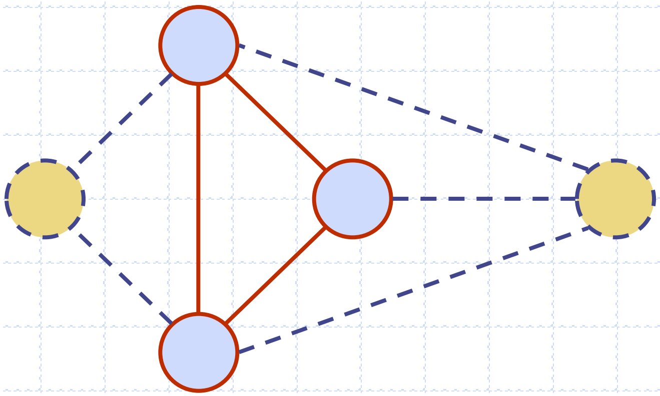
Visita in profondità (DFS)

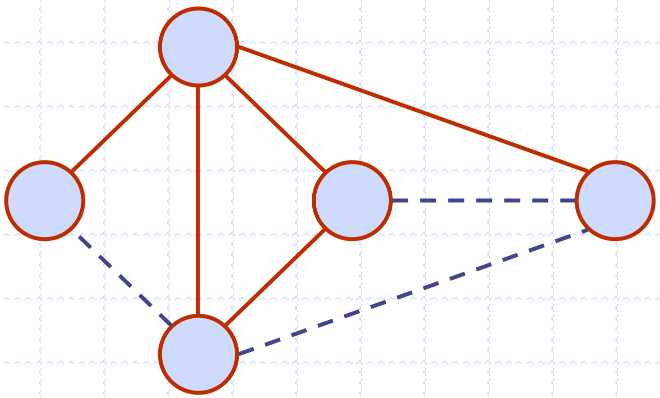


Sottografi

- Un sottografo S di un grafo G è un grafo tale che
  - i vertici di S sono un sottoinsieme di quelli di G
  - gli spigoli di S sono un sottoinsieme di quelli di G
- Un sottografo ricoprente (spanning) di G è un sottografo di G che contiene tutti i vertici di G



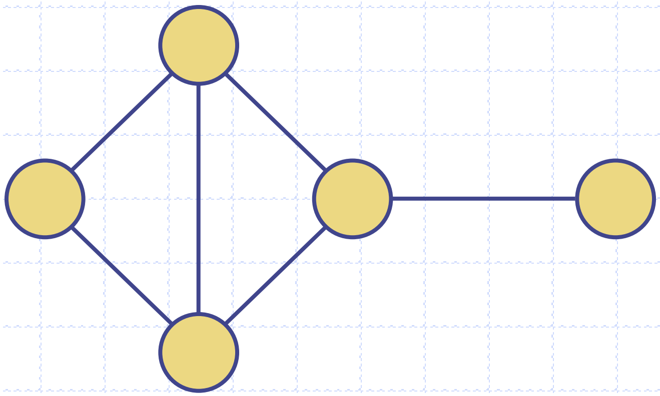
Sottografo



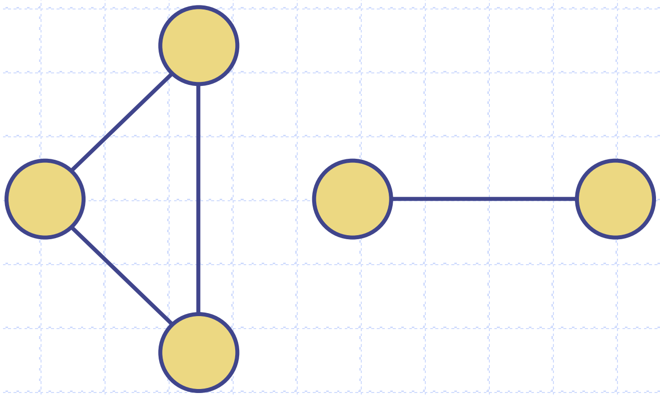
Sottografo ricoprente

Connettività

- Un grafo è detto connesso se esiste un percorso (path) fra ogni coppia di vertici
- Una componente connessa di un grafo G è un sottografo di G connesso e massimale



Grafo connesso

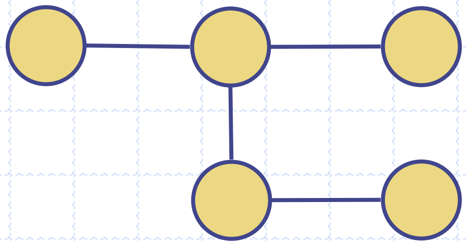


Grafo non connesso con due componenti connesse

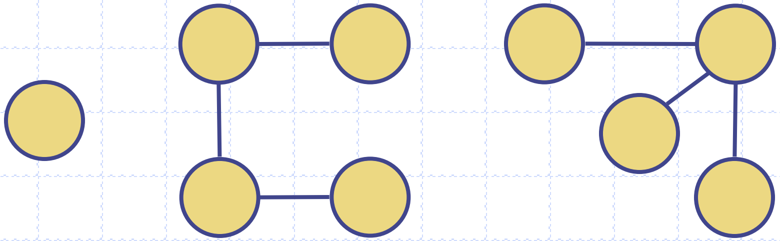
Alberi e foreste

- Un albero (libero) è un grafo non orientato T tale che
  - T è connesso
  - T è aciclico (senza cicli)

  Definizione differente da quella di albero radicato (rooted)
- Una foresta è un grafo non orientato e aciclico
- Le componenti connesse di una foresta sono alberi



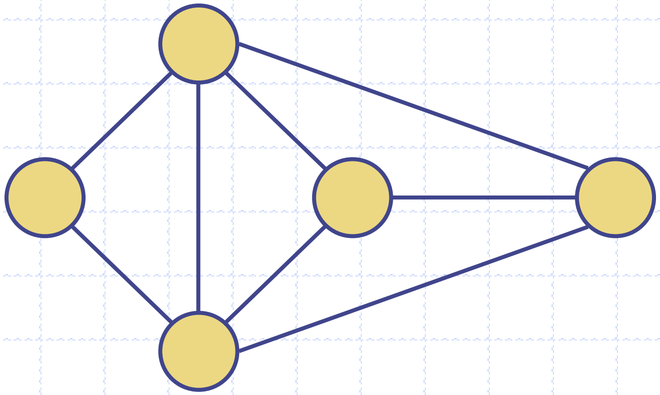
Albero



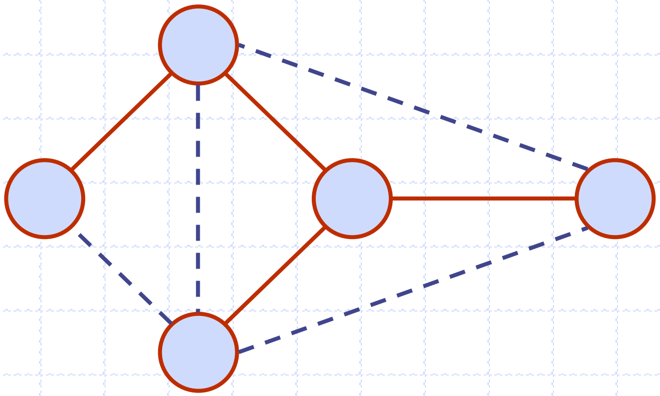
Foresta

Alberi e foreste ricoprenti

- Un albero ricoprente di un grafo connesso è un sottografo ricoprente con l'ulteriore caratteristica di essere un albero
- Un albero ricoprente non è unico, a meno che il grafo non sia un albero
- Gli alberi ricoprenti sono utilizzati nel progetto di reti di comunicazione
- Una foresta ricoprente di un grafo è un grafo ricoprente con l'ulteriore caratteristica di essere una foresta



Grafo



Albero ricoprente

Visità in profondità (§ 13.3.1)

- La visita in profondità (depth-first search, DFS) è una tecnica generale di attraversamento di un grafo
- Una visita DFS di un grafo G
  - Visita tutti i vertici e gli spigoli di G
  - Determina se G è connesso
  - Calcola le componenti connesse di G
  - Calcola una foresta ricoprente di G
- Una DFS su un grafo con n vertici ed m spigoli richiede tempo $O(n+m)$
- La DFS può essere estesa per risolvere altri problemi su grafi
  - Trovare e restituire un percorso fra due vertici assegnati
  - Individuare un ciclo in un grafo
- La visita in profondità sta ai grafi come la visita con cammino Euleriano sta agli alberi binari

Algoritmo DFS

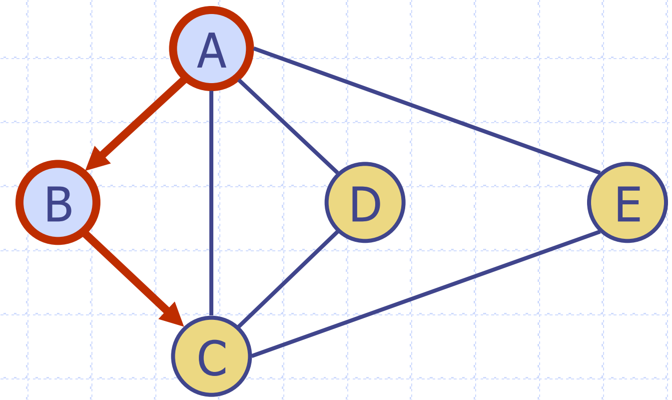
- L'algoritmo usa un meccanismo per assegnare e consultare etichette di vertici e spigoli

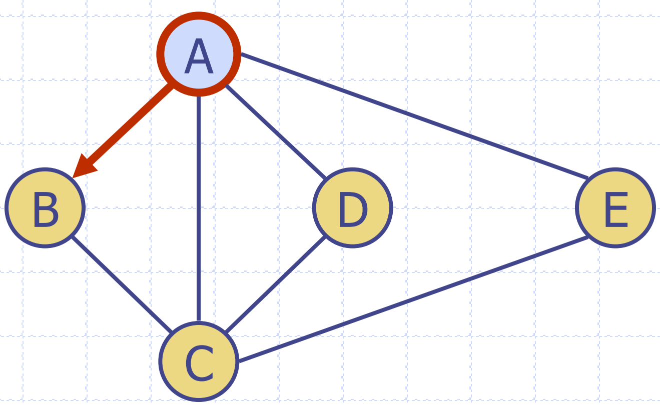
Algorithm *DFS*(*G*)
  Input grafo *G*
  Output etichettatura degli spigoli di *G* come tree-edge e back-edge
  for all *u* ∈ *G.vertices*()
    setLabel(*u, UNEXPLORED*)
  for all *e* ∈ *G.edges*()
    setLabel(*e, UNEXPLORED*)
  for all *v* ∈ *G.vertices*()
    if *getLabel*(*v*) = *UNEXPLORED*
      DFS(*G, v*)

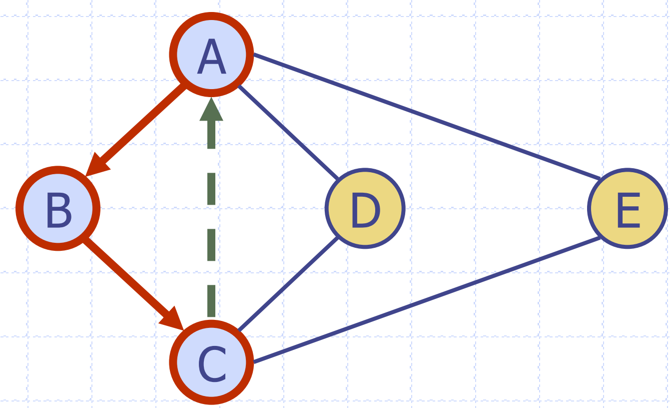
Algorithm *DFS*(*G, v*)
  Input grafo *G* vertice iniziale *v* di *G*
  Output etichettatura degli spigoli di *G* nella componente connessa di *v* come tree-edge e back-edge
  setLabel(*v, VISITED*)
  for all *e* ∈ *G.incidentEdges*(*v*)
    if *getLabel*(*e*) = *UNEXPLORED*
      w ← *opposite*(*v,e*)
      if *getLabel*(*w*) = *UNEXPLORED*
        setLabel(*e, DISCOVERY*)
        DFS(*G, w*)
      else
        setLabel(*e, BACK*)

Esempio

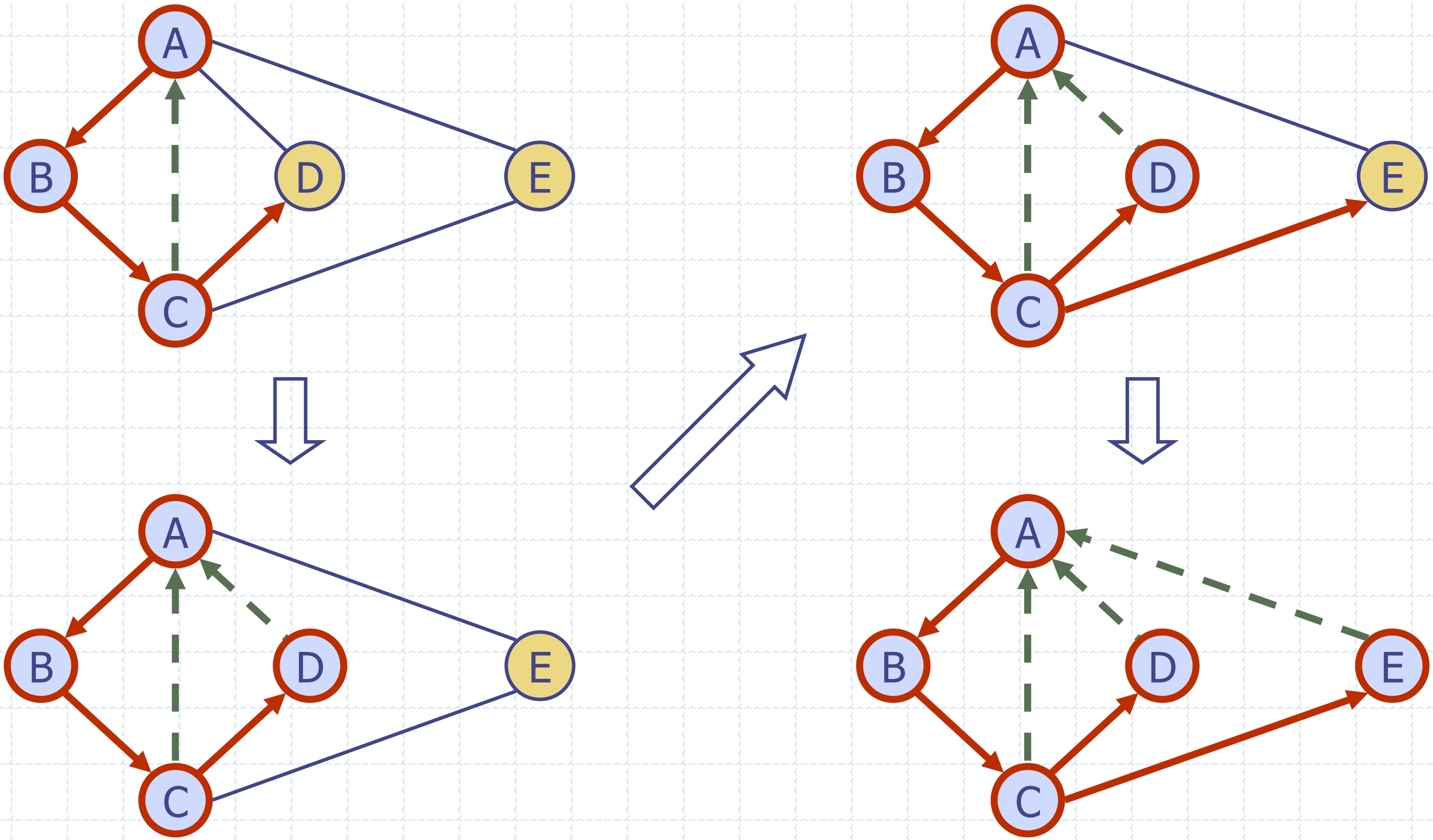
- A vertice inesplorato
- A vertice visitato
- spigolo inesplorato
- tree-edge
- back-edge



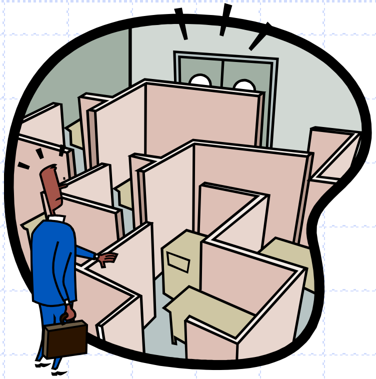




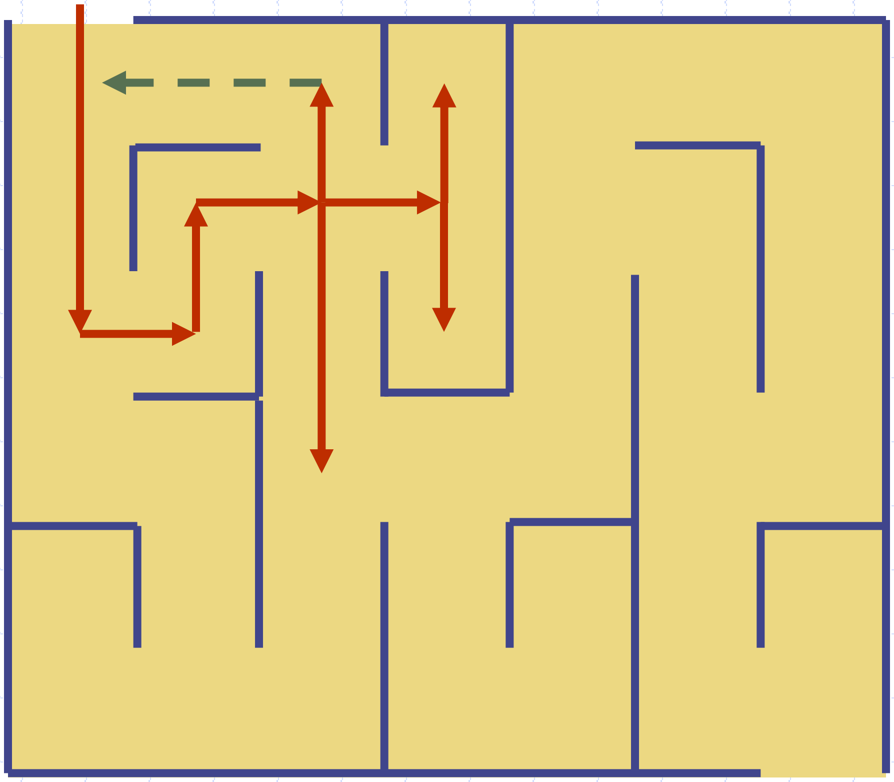
Esempio (cont.)



DFS ed attraversamento di labirinti



- L'algoritmo DFS è simile alla classica strategia di attraversamento di un labirinto
  - marchiamo ogni incrocio, angolo e vicolo cieco (vertice) visitato
  - marchiamo ogni corridoio attraversato (spigolo)
  - teniamo traccia del percorso che riporta all'ingresso (vertice iniziale) attraverso una fune (pila delle ricorsioni)



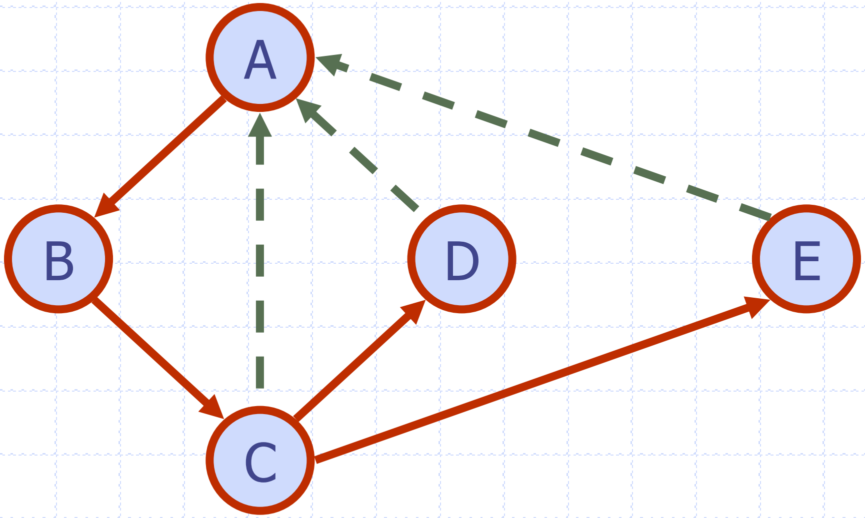
Proprietà della DFS

Proprietà 1

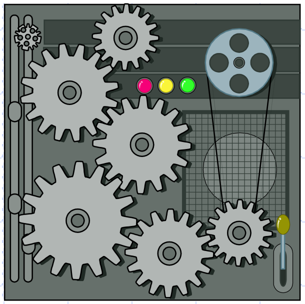
DFS(*G*, *v*) visita tutti i vertici e gli spigoli nella componente connessa che contiene *v*

Proprietà 2

I tree-edge etichettati da *DFS*(*G*, *v*) formano un albero ricoprente della componente connessa che contiene *v*



Analisi della DFS



- Il set/get di etichette di vertici/spigoli richiede tempo $O(1)$
- Ciascun vertice viene etichettato due volte
  - una volta come UNEXPLORED
  - una volta come VISITED
- Ciascuno spigolo viene etichettato due volte
  - una volta come UNEXPLORED
  - una volta come DISCOVERY (TREE) o BACK
- Il metodo incidentEdges viene chiamato una volta per ciascun vertice
- La DFS viene eseguita in tempo $O(n + m)$ purché il grafo sia rappresentato dalla lista delle adiacenze
  - rammentare che $\Sigma_v \deg(v) = 2m$

Ricerca di percorsi

- Possiamo specializzare l'algoritmo DFS per individuare un percorso fra due vertici assegnati u e z
- Eseguiamo una DFS con u come vertice iniziale
- Usiamo una pila S per tener traccia del percorso fra il vertice iniziale e quello corrente
  - pila ibrida (vertici e spigoli)
- Non appena viene incontrato il vertice destinazione z, restituiamo il percorso contenuto nella pila

```
Algorithm pathDFS(G, u, z)
  setLabel(u, VISITED)
  S.push(u)
  if  u = z
     return S.elements()
  for all  e ∈ G.incidentEdges(u)
     if  getLabel(e) = UNEXPLORED
        w ← opposite(u,e)
        if getLabel(w) = UNEXPLORED
           setLabel(e, DISCOVERY)
           S.push(e)
           pathDFS(G, w, z)
           S.pop(e)
        else
           setLabel(e, BACK)
  S.pop(u)
```

Ricerca di cicli

- Possiamo specializzare l'algoritmo DFS per trovare un ciclo semplice nella componente connessa che contiene un vertice v
- Usiamo una pila S per tener traccia del percorso fra il vertice iniziale e quello corrente
  - pila ibdrida (vertici e spigoli)
- Appena si incontra un back-edge (v, w), restituiamo il ciclo presente nella porzione di pila che va dalla cima al vertice w

```
Algorithm cycleDFS(G, v)
    setLabel(v, VISITED)
    S.push(v)
    for all  e ∈ G.incidentEdges(v)
        if  getLabel(e) = UNEXPLORED
            w ← opposite(v,e)
            S.push(e)
            if getLabel(w) = UNEXPLORED
                setLabel(e, DISCOVERY)
                pathDFS(G, w)
                S.pop(e)
            else
                T ← new empty stack
                repeat
                    o ← S.pop()
                    T.push(o)
                until o = w
                return T.elements()
    S.pop(v)
```