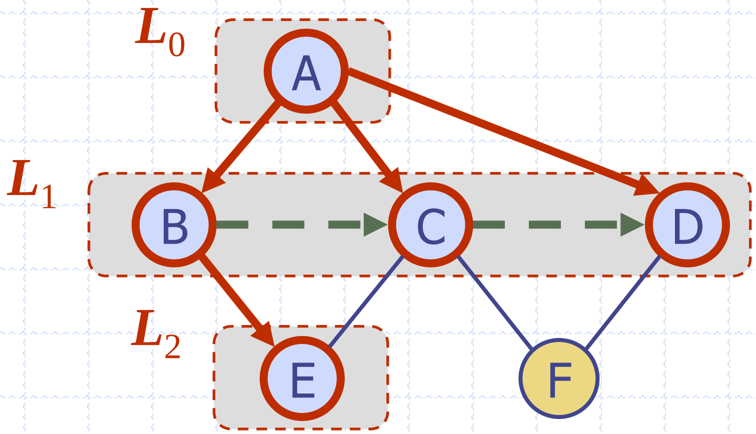
Breadth-First Search



Breadth-First Search

- La Breadth-first search (BFS) e’ una tecnica generale per visitare grafi
- Una visita BFS di un grafo G
  - Visita tutti i vertici e gli archi di G
  - Determina se G e’ connesso
  - Calcola le componenti connesse di G
  - Calcola una spanning forest di G
- La BFS su un grafo con n vertici e m archi richiede tempo $O(n+m)$
- La BFS puo’ essere ulteriormente estesa per risolvere altri problemi su grafi:
  - Calcolare un cammino con numero minimo di archi tra due vertici
  - Trovare un ciclo semplice se ne esiste uno

Algoritmo BFS

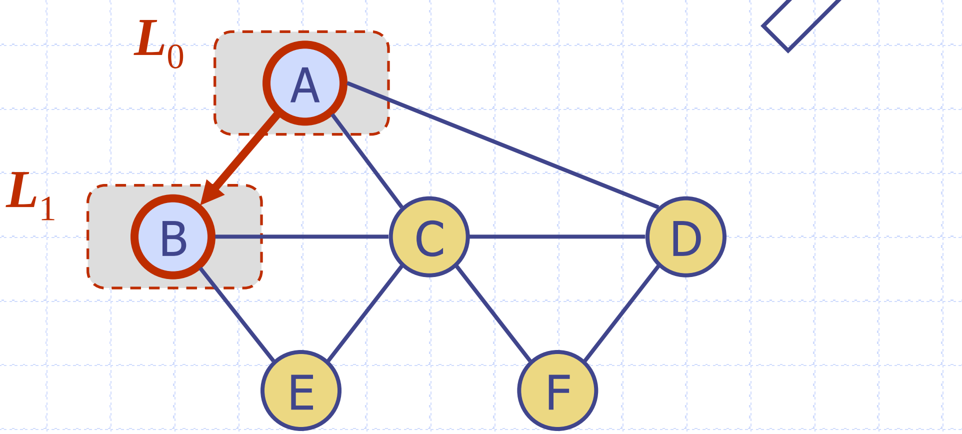
- L'algoritmo usa un meccanismo per assegnare e accedere "etichette" di vertici e archi

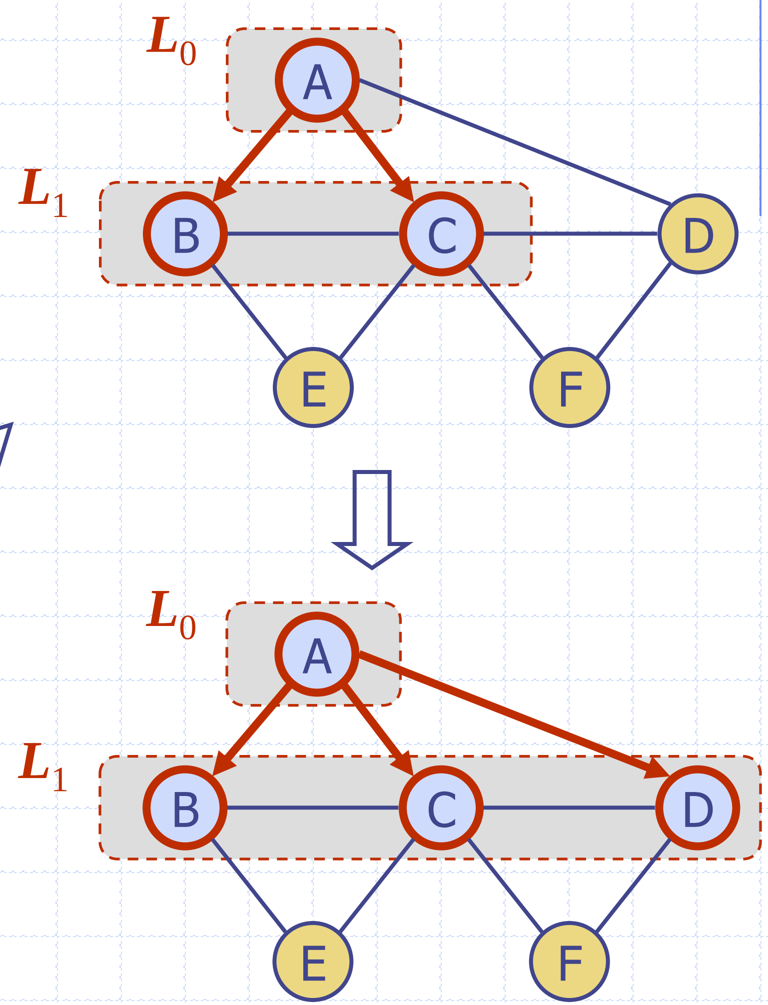
```
Algorithm BFS(G)
    Input graph G
    Output labeling of the edges
        and partition of the
        vertices of G
  for all u ∈ G.vertices()
    setLabel(u, UNEXPLORED)
  for all e ∈ G.edges()
    setLabel(e, UNEXPLORED)
  for all v ∈ G.vertices()
    if getLabel(v) = UNEXPLORED
        BFS(G, v)
```

```
Algorithm BFS(G, s)
   L0 ← new empty sequence
   L0.insertLast(s)
   setLabel(s, VISITED)
   i ← 0
   while ¬Li.isEmpty()
      Li+1 ← new empty sequence
      for all v ∈ Li.elements()
         for all e ∈ G.incidentEdges(v)
            if getLabel(e) = UNEXPLORED
               w ← opposite(v,e)
               if getLabel(w) = UNEXPLORED
                  setLabel(e, DISCOVERY)
                  setLabel(w, VISITED)
                  Li+1.insertLast(w)
               else
                  setLabel(e, CROSS)
      i ← i +1
```

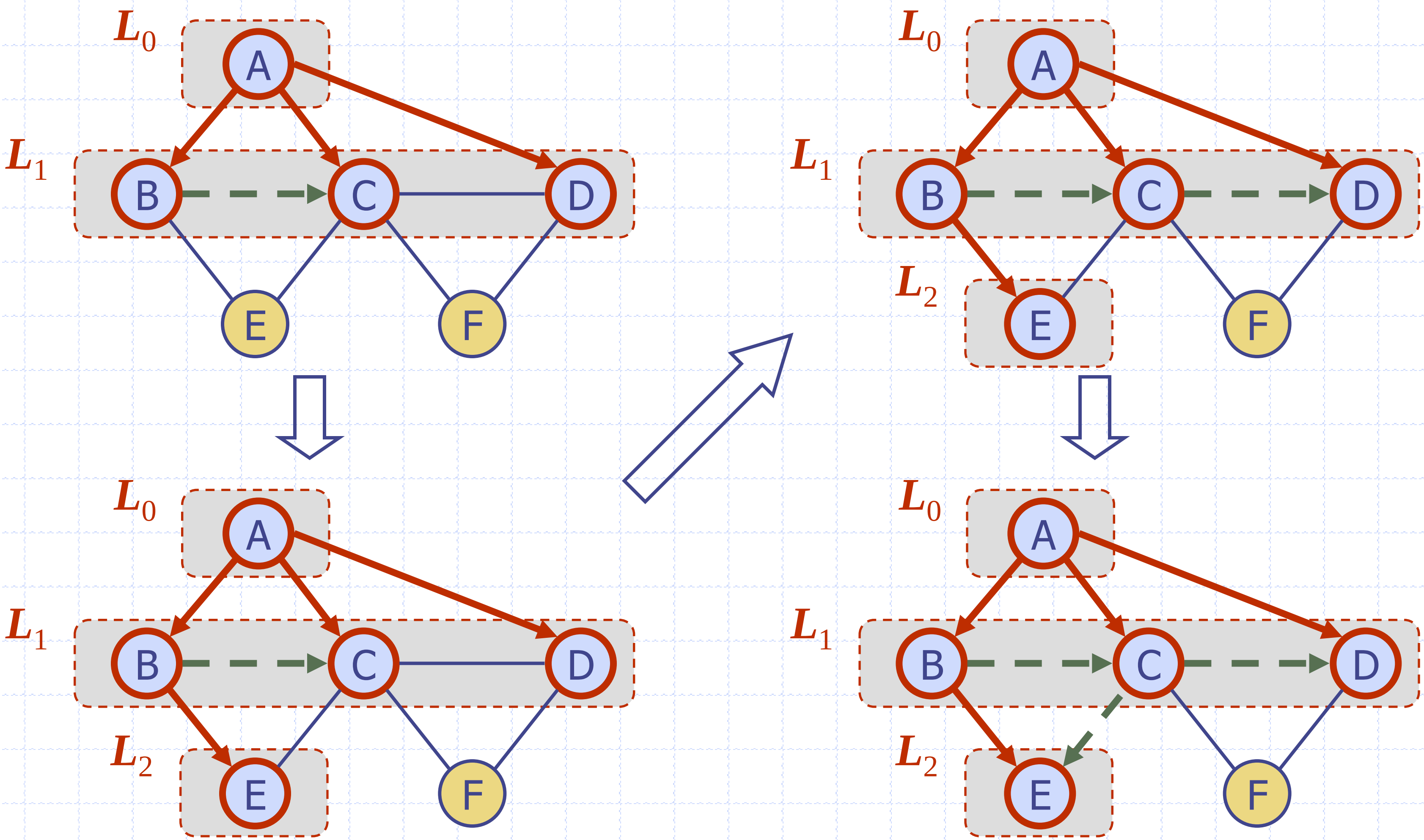
Esempio

- vertice unexplored
- vertice visited
- arco unexplored
- arco discovery
- arco cross

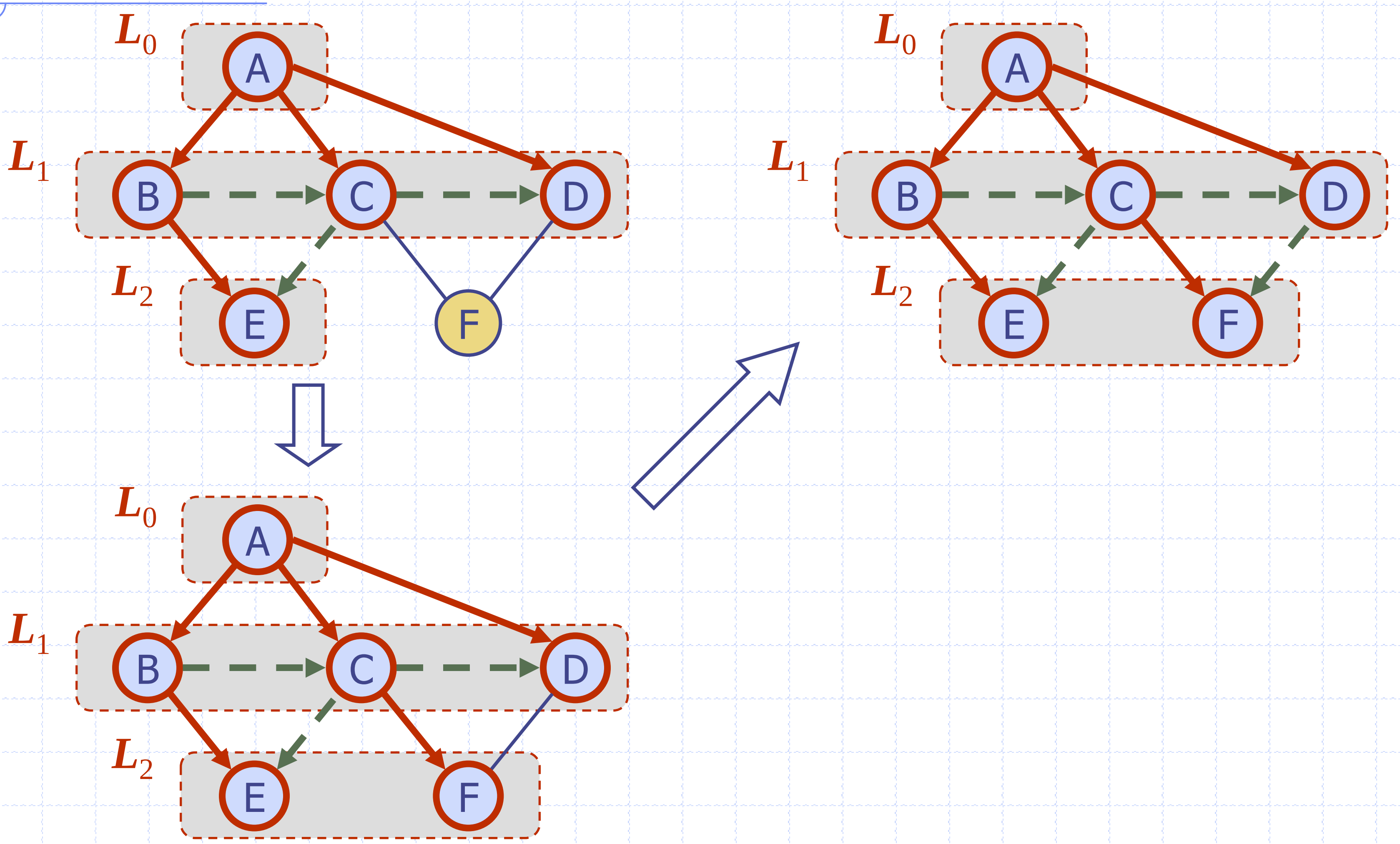




Esempio (cont.)

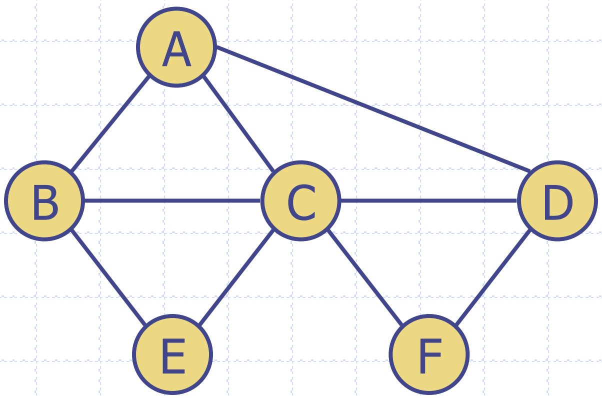


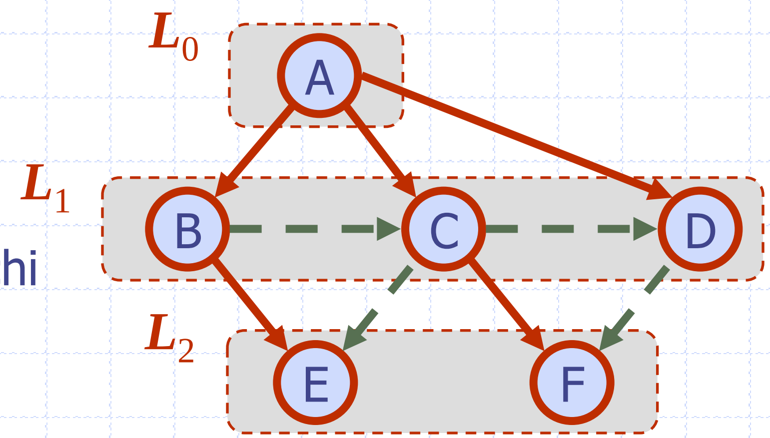
Esempio (cont.)



Proprieta’

- Notazione
  - G_s: componente connessa di s
- Proprieta’ 1
  - $BFS(G, s)$ visita tutti i vertici e gli archi di G_s
- Proprieta’ 2
  - Gli archi discovery etichettati da $BFS(G, s)$ formano uno spanning tree T_s di G_s
- Proprieta’ 3
  - Per ogni vertice v in L_i
    - Il cammino di T_s da s a v ha i archi
    - Ogni cammino da s a v in G_s ha almeno i archi





Analisi

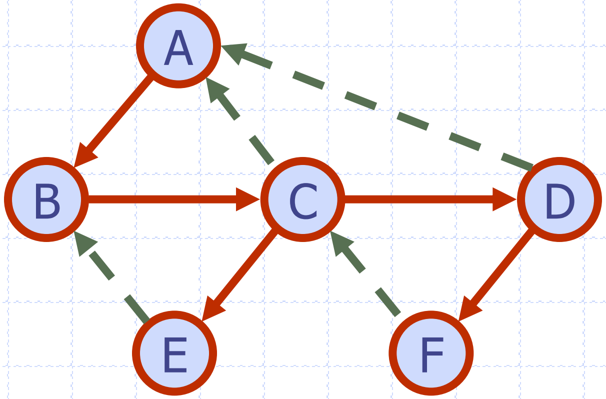
- Assegnare/accedere l'etichetta di un vertice/arco ha costo $O(1)$
- Ogni vertice e' etichettato due volte
  - una volta come UNEXPLORED
  - una volta come VISITED
- Ogni arco e' etichettato due volte
  - una volta come UNEXPLORED
  - una volta come DISCOVERY o CROSS
- Ogni vertice e' inserito una volta in una sequenza L_i
- Il metodo incidentEdges e' chiamato una volta per ogni vertice
- BFS richiede tempo $O(n + m)$ se il grafo e' rappresentato come lista di adiacenza
  - Ricorda che $\Sigma_v \deg(v) = 2m$

Applicazioni

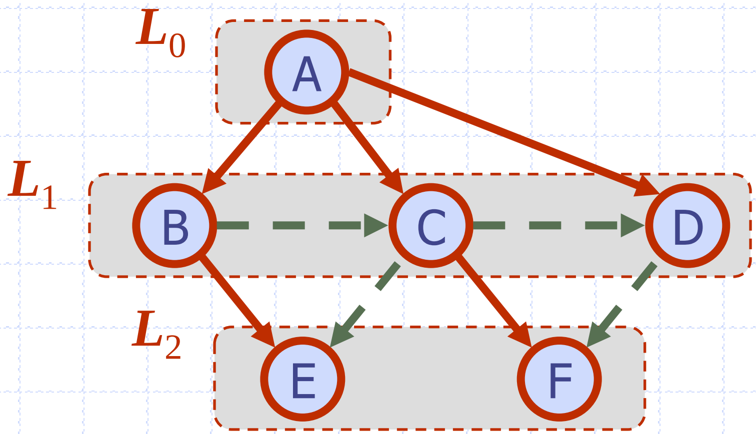
- Usando il metodo template pattern, possiamo specializzare la visita BFS di un grafo G per risolvere i seguenti problemi in tempo $O(n + m)$
  - Calcolare le componenti connesse di G
  - Calcolare una spanning forest di G
  - Trovare un ciclo semplice in G, o stabilire che G e' una foresta
  - Dati due vertici di G, trova un cammino tra loro in G con il minimo numero di archi, o stabilire che tale cammino non esiste

DFS vs. BFS

Applicazioni	DFS	BFS
Spanning forest, componenti connesse, cammini, cicli	√	√
Cammini minimi		√
Componenti biconnesse	√	



DFS



BFS

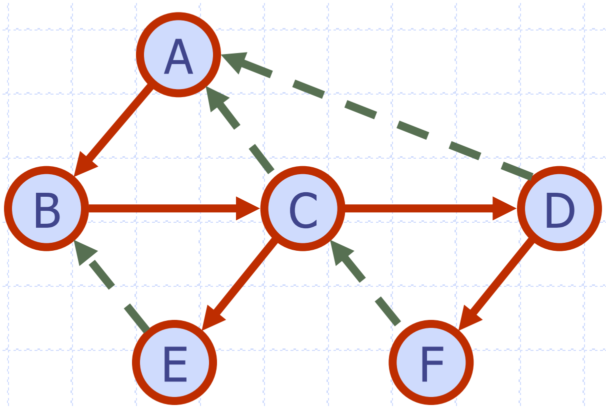
DFS vs. BFS (cont.)

Back edge (v,w)

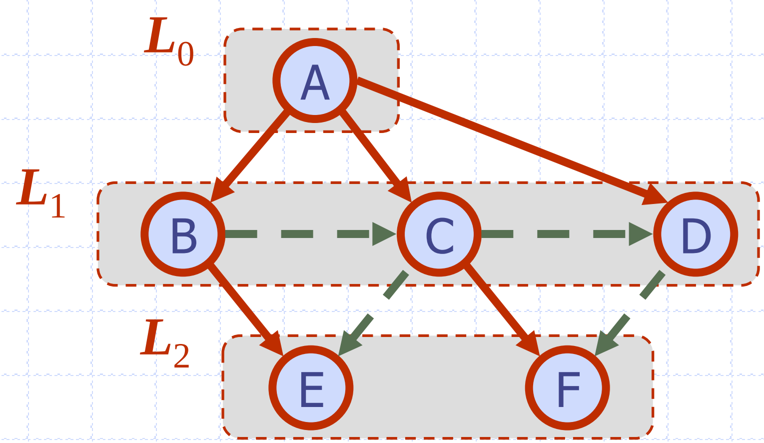
- w e' un antenato di v nell'albero degli archi discovery

Cross edge (v,w)

- w e' nello stesso livello di v o nel livello successivo nell'albero degli archi discovery



DFS



BFS