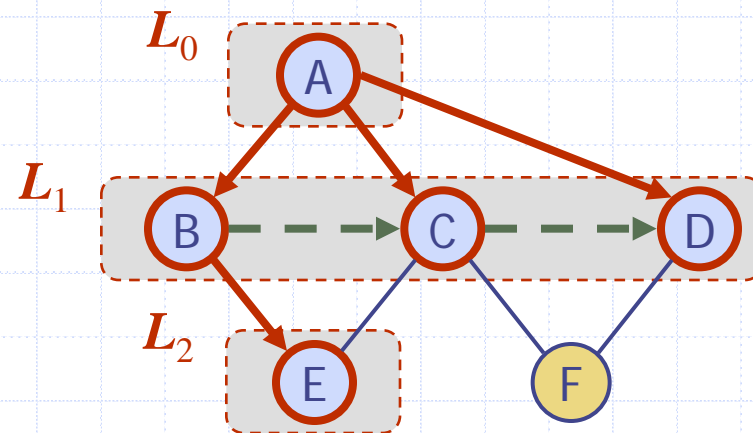


Breadth-First Search



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

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

Algorithm **BFS**(G, s)

```
 $L_0 \leftarrow$  new empty sequence
 $L_0.insertBack(s)$ 
 $s.setLabel(VISITED)$ 
 $i \leftarrow 0$ 
while  $\neg L_i.empty()$ 
     $L_{i+1} \leftarrow$  new empty sequence
    for all  $v \in L_i.elements()$ 
        for all  $e \in v.incidentEdges()$ 
            if  $e.getLabel() = UNEXPLORED$ 
                 $w \leftarrow e.opposite(v)$ 
                if  $w.getLabel() = UNEXPLORED$ 
                     $e.setLabel(DISCOVERY)$ 
                     $w.setLabel(VISITED)$ 
                     $L_{i+1}.insertBack(w)$ 
                else
                     $e.setLabel(CROSS)$ 
     $i \leftarrow i + 1$ 
```

Example



unexplored vertex



visited vertex



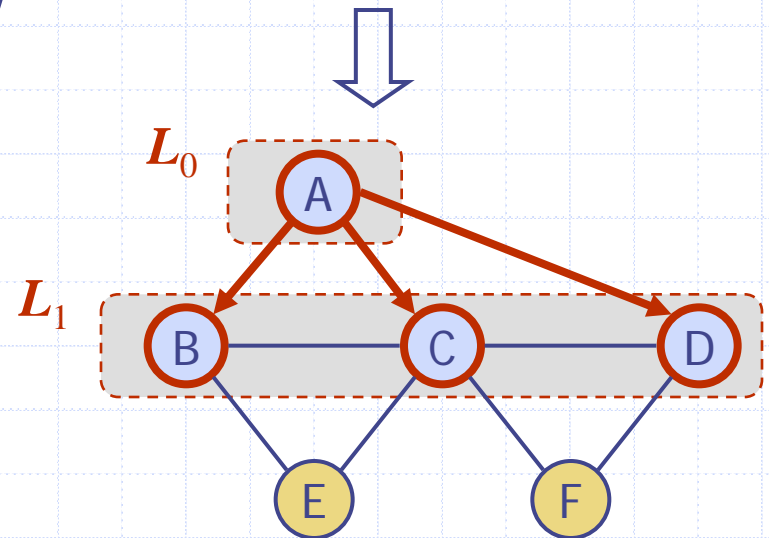
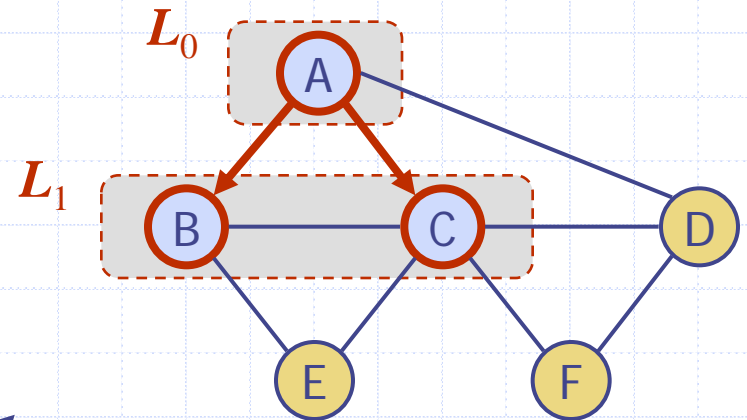
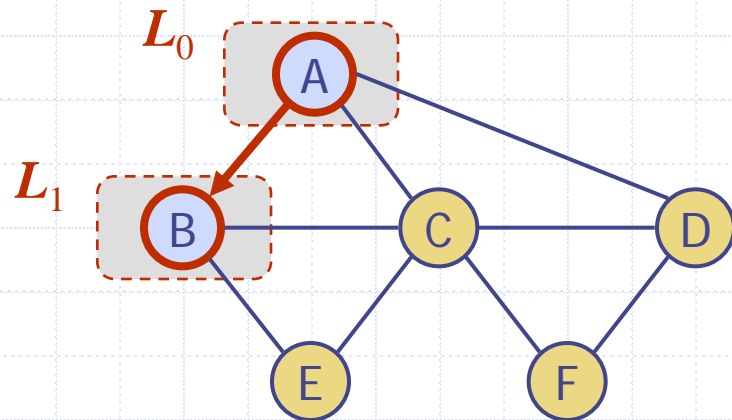
unexplored edge



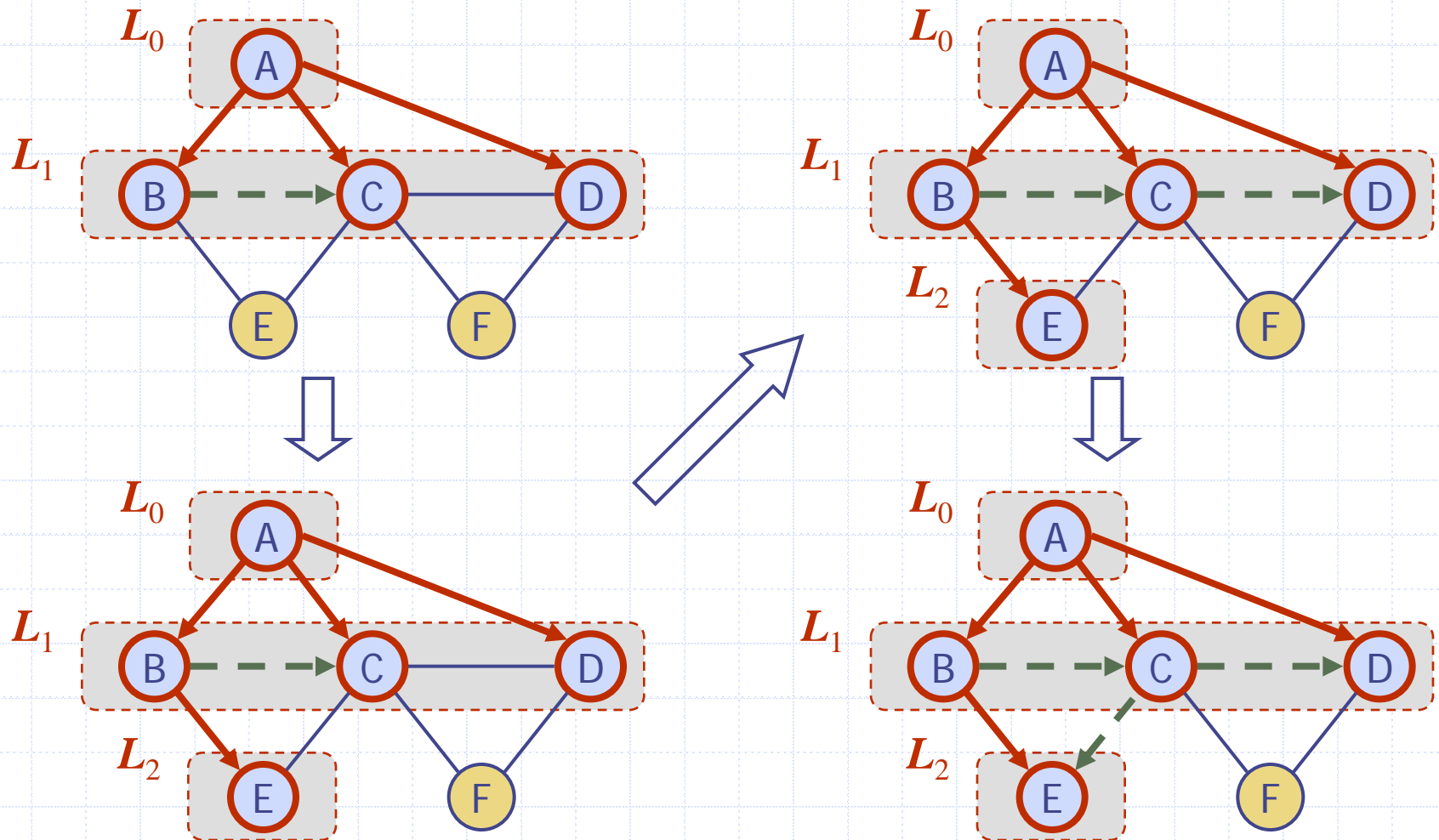
discovery edge



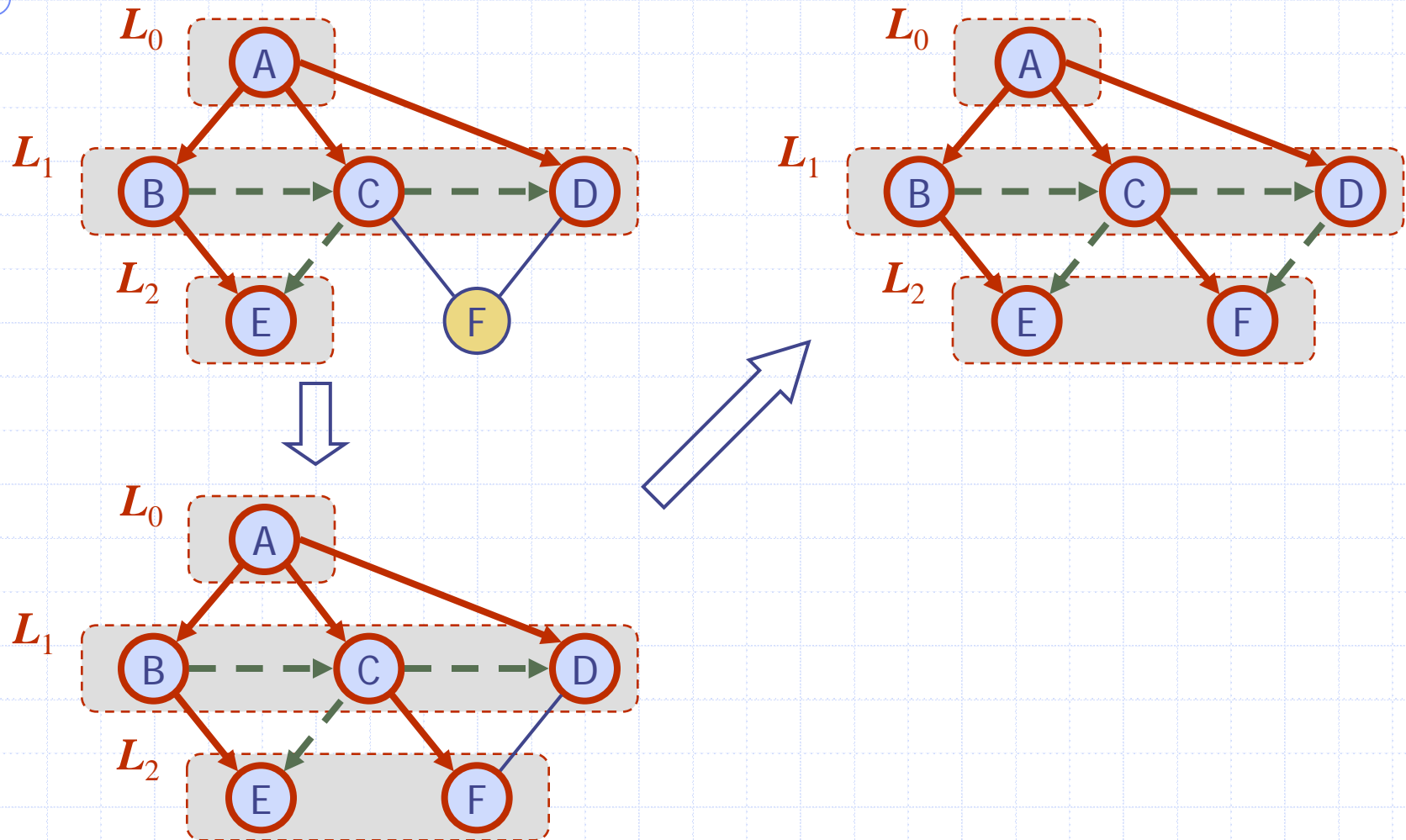
cross edge



Example (cont.)



Example (cont.)



Properties

Notation

G_s : connected component of s

Property 1

$BFS(G, s)$ visits all the vertices and edges of G_s

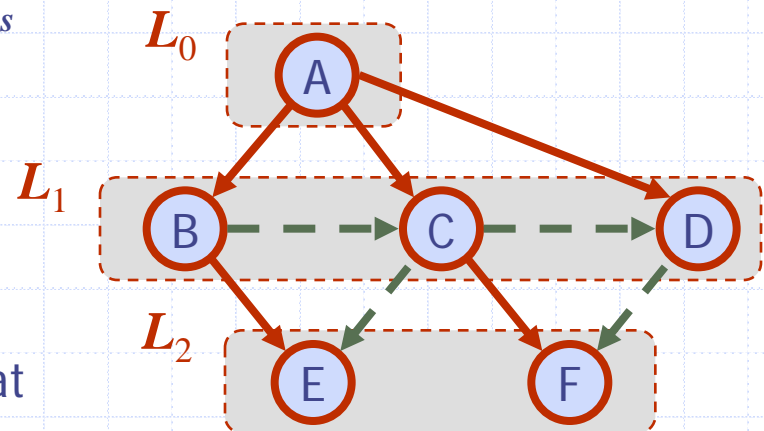
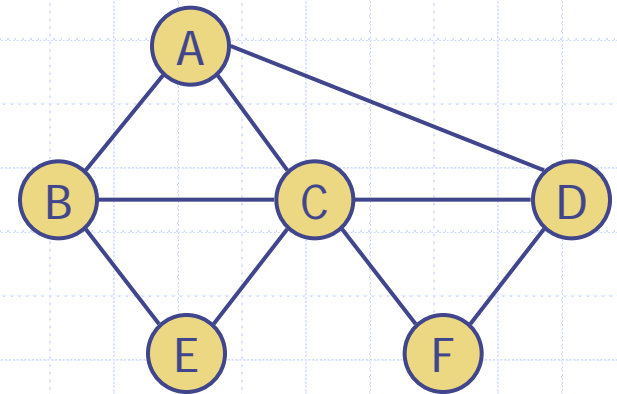
Property 2

The discovery edges labeled by $BFS(G, s)$ form a spanning tree T_s of G_s

Property 3

For each vertex v in L_i

- The path of T_s from s to v has i edges
- Every path from s to v in G_s has at least i edges



Analysis

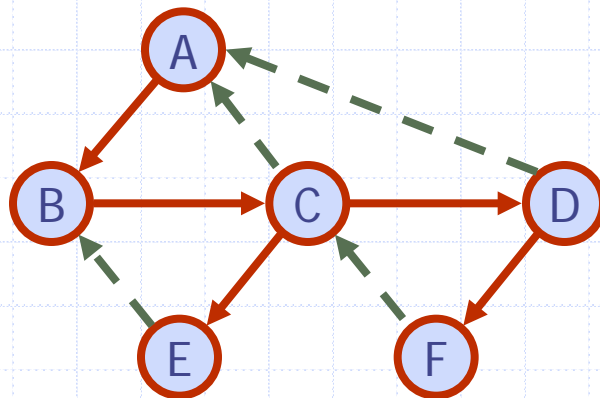
- ❑ Setting/getting a vertex/edge label takes $O(1)$ time
- ❑ Each vertex is labeled twice
 - once as UNEXPLORED
 - once as VISITED
- ❑ Each edge is labeled twice
 - once as UNEXPLORED
 - once as DISCOVERY or CROSS
- ❑ Each vertex is inserted once into a sequence L_i
- ❑ Method incidentEdges is called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

Applications

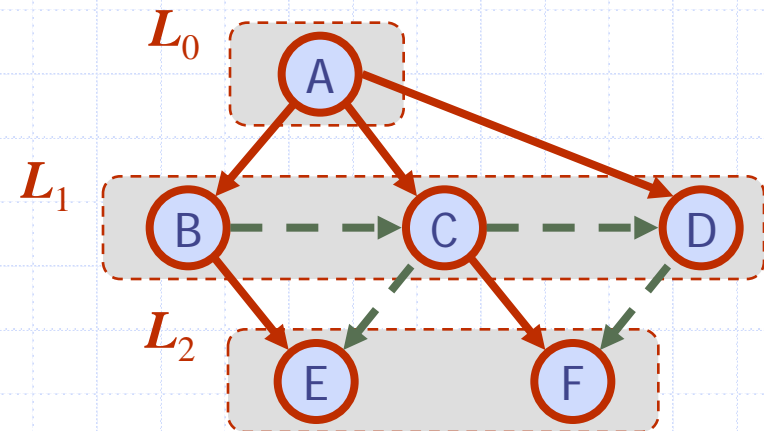
- Using the **template method pattern**, we can specialize the BFS traversal of a graph G to solve the following problems in $O(n + m)$ time
 - Compute the connected components of G
 - Compute a spanning forest of G
 - Find a simple cycle in G , or report that G is a forest
 - Given two vertices of G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

DFS vs. BFS

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



DFS

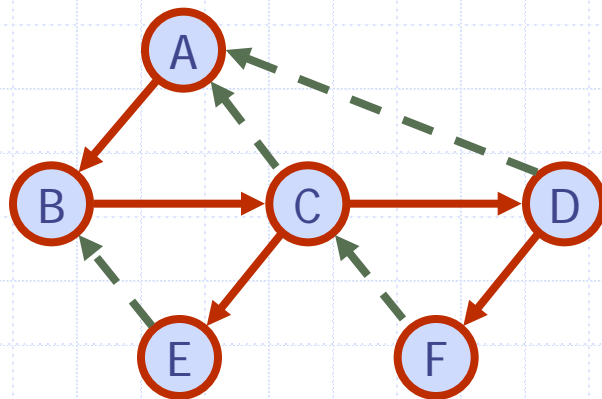


BFS

DFS vs. BFS (cont.)

Back edge (v, w)

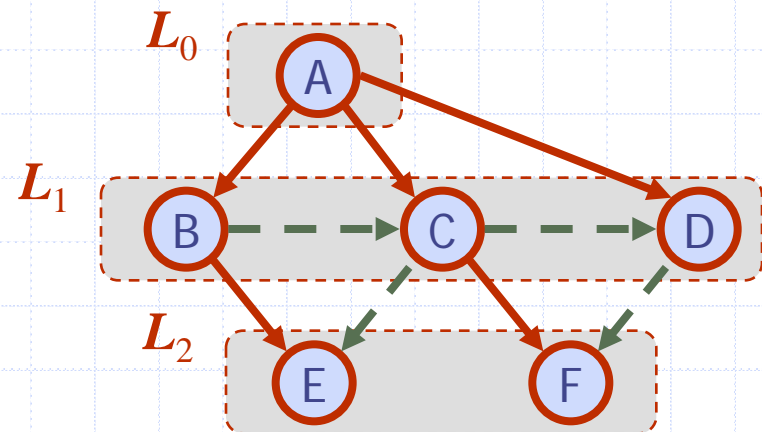
- w is an ancestor of v in the tree of discovery edges



DFS

Cross edge (v, w)

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



BFS