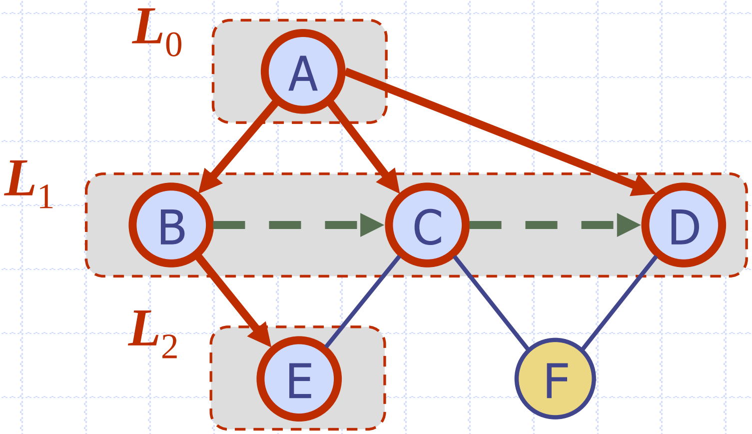
Breadth-First Search



Breadth-First Search (BFS)

- A general technique for traversing a graph
- BFS traversal of 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
- Complexity: $O(n + m)$ for a graph with n vertices and m edges
- BFS for other graph problems
  - Find a minimum path between two given vertices
  - Find a simple cycle
- BFS is to graphs what level-order is to binary/general rooted trees

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

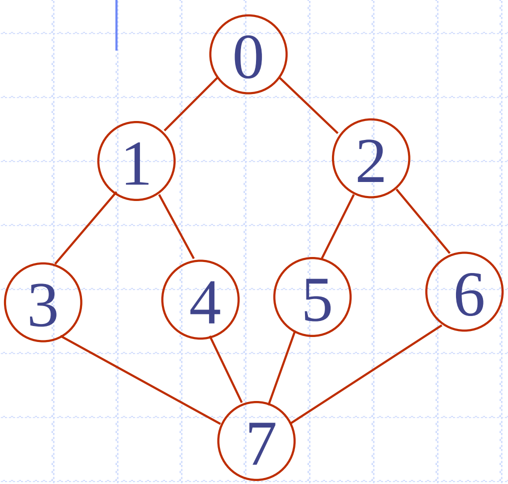
Algorithm $BFS(G, s)$

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

Example via Queue

Quiz!

- Start vertex: 0
- Traverse order: 0, 1, 2, 3, 4, 5, 6, 7



vertex 0 -> 1 -> 2
vertex 1 -> 0 -> 3 -> 4
vertex 2 -> 0 -> 5 -> 6
vertex 3 -> 1 -> 7
vertex 4 -> 1 -> 7
vertex 5 -> 2 -> 7
vertex 6 -> 2 -> 7
vertex 7 -> 3 -> 4 -> 5 -> 6

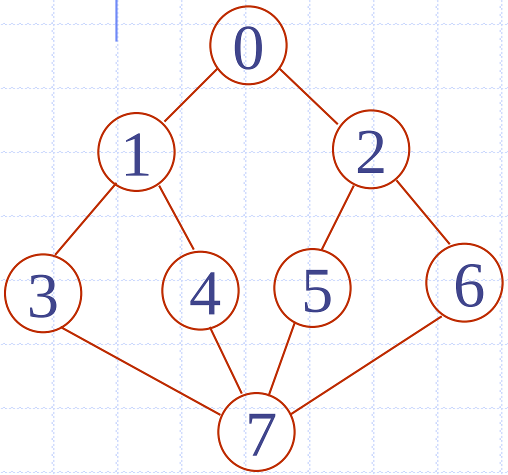
- Queue contents at each step:

Output	Queue
	0
0	1 2
1	2 0 3 4
2	3 4 0 5 6
3	4 5 6 1 7
4	5 6 7 1 7
5	6 7 7 2 7
6	7 7 7 2 7
7	7 7 7 3 4 5 6

Level-order!

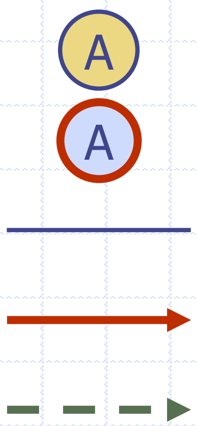
Another Example via Queue

- Start vertex: 4
- Traverse order: 4, 1, 7, 0, 3, 5, 6, 2
- Queue contents at each step:

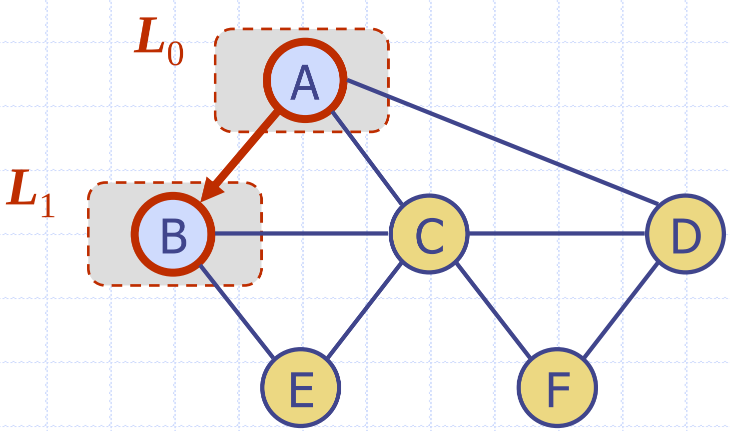


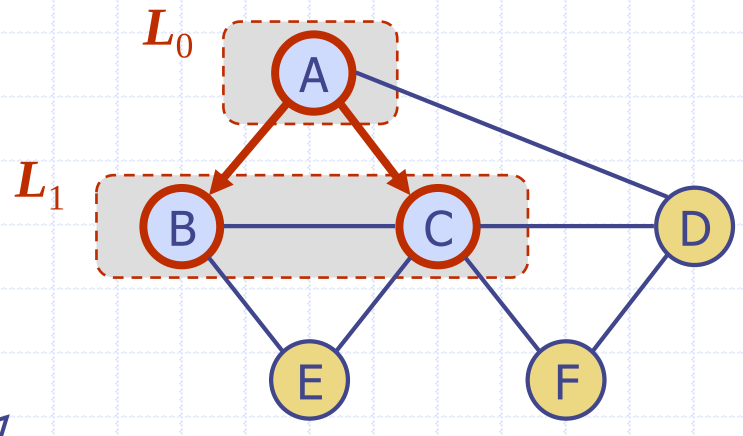
vertex 0 -> 1 -> 2
vertex 1 -> 0 -> 3 -> 4
vertex 2 -> 0 -> 5 -> 6
vertex 3 -> 1 -> 7
vertex 4 -> 1 -> 7
vertex 5 -> 2 -> 7
vertex 6 -> 2 -> 7
vertex 7 -> 3 -> 4 -> 5 -> 6

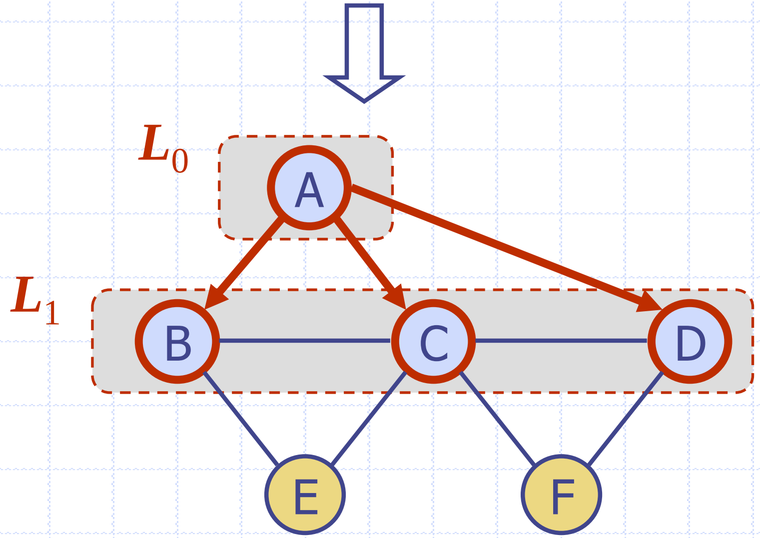
Example



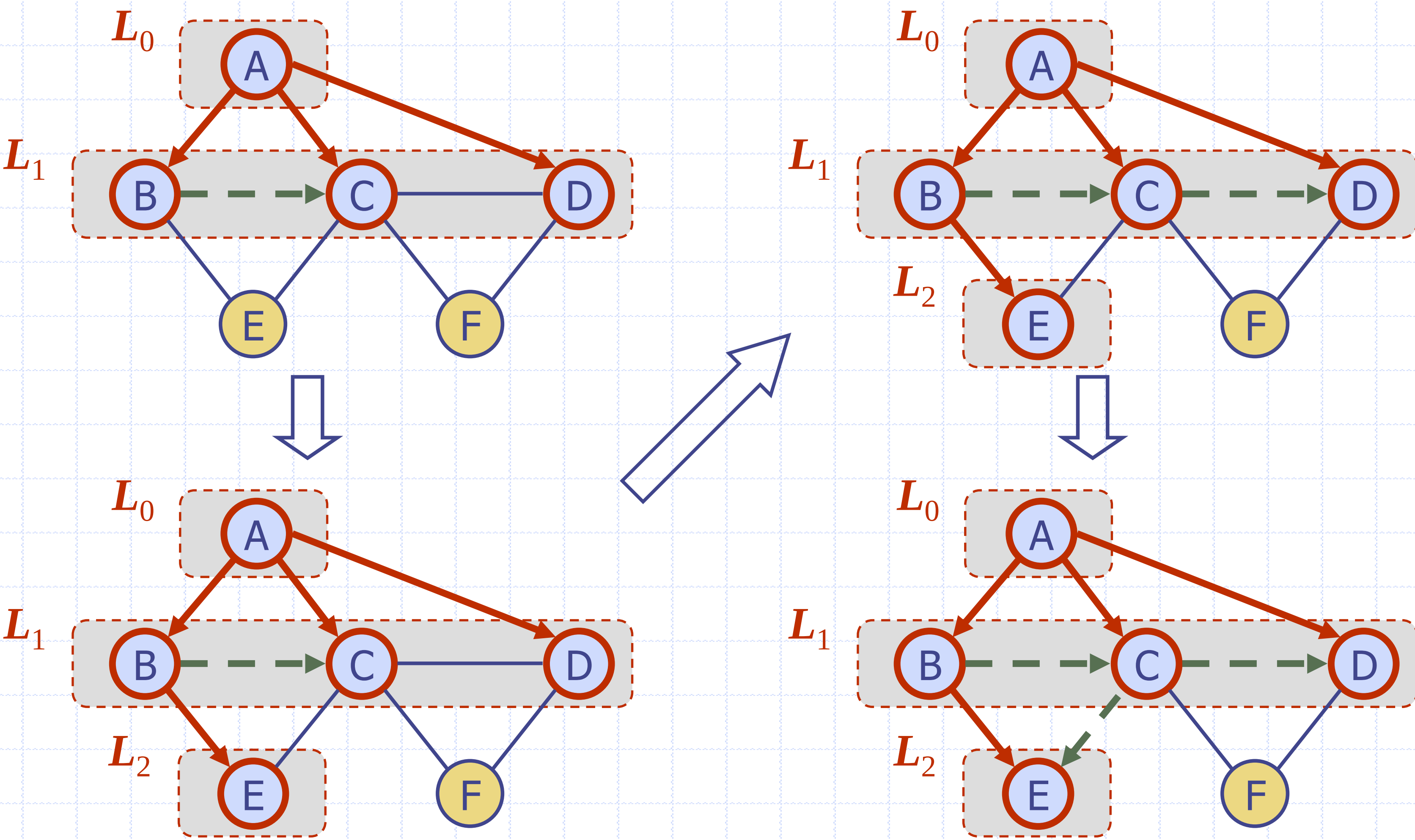
unexplored vertex
visited vertex
unexplored edge
discovery edge
cross edge



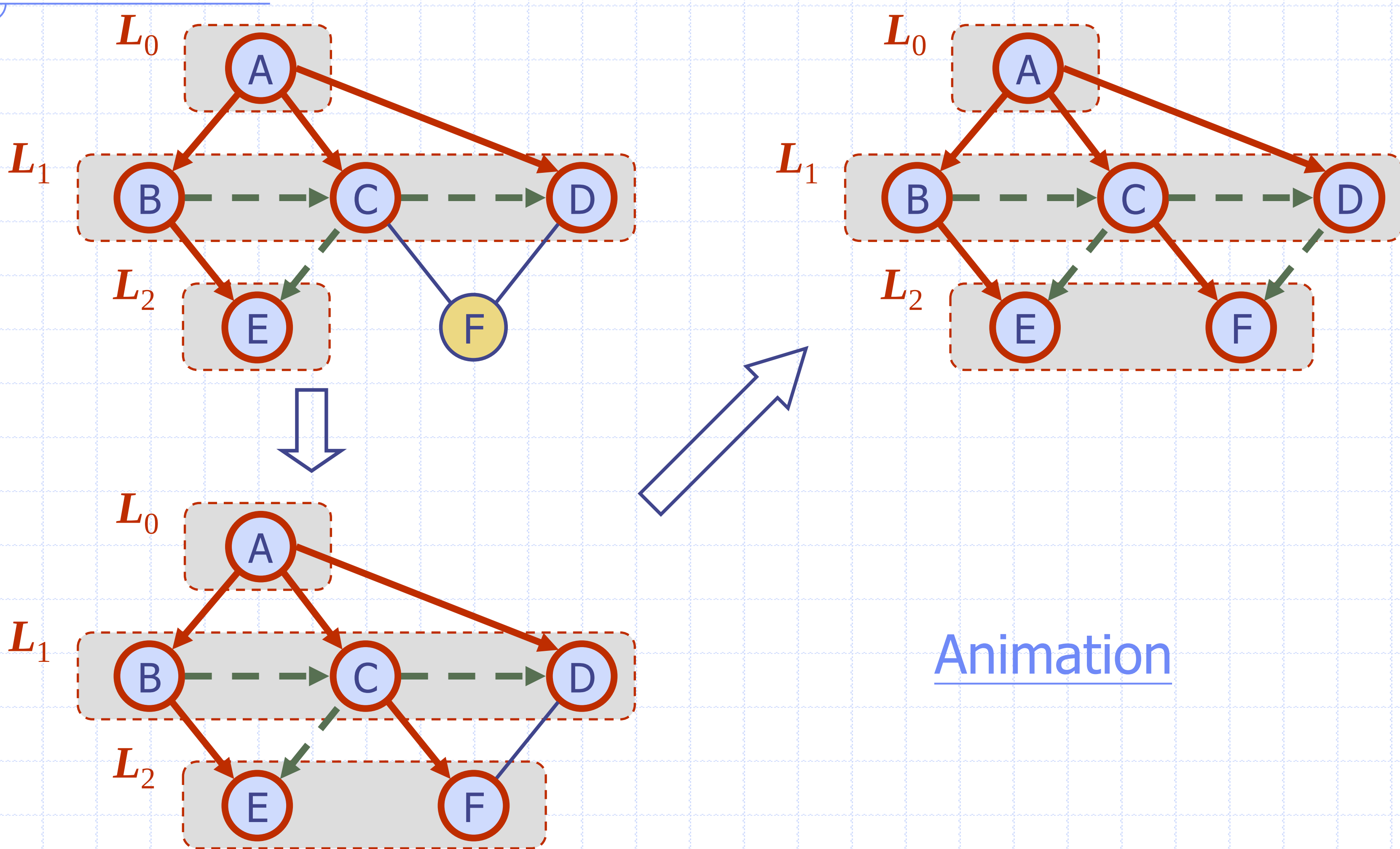




Example (cont.)



Example (cont.)



Animation

Properties

Notation

G_s: connected component of s

Property 1

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

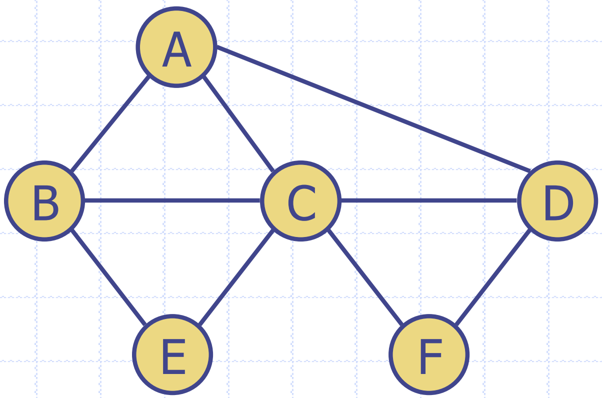
Property 2

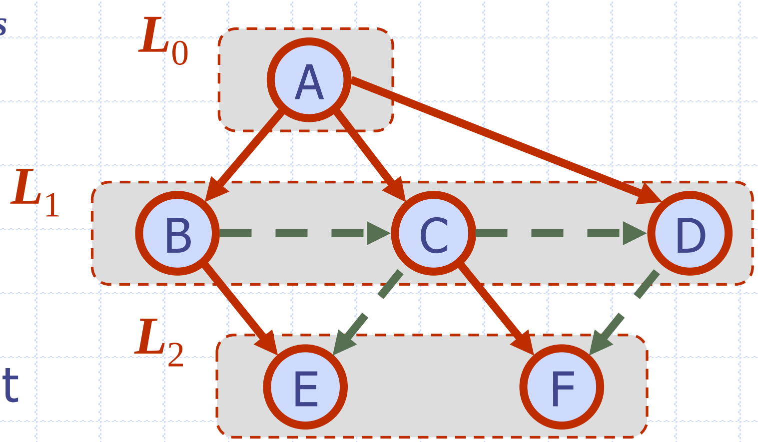
The discovery edges labeled by $BFS(G, s)$ form a spanning tree T_s of G_s

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 $\Sigma_v \deg(v) = 2m$

Applications

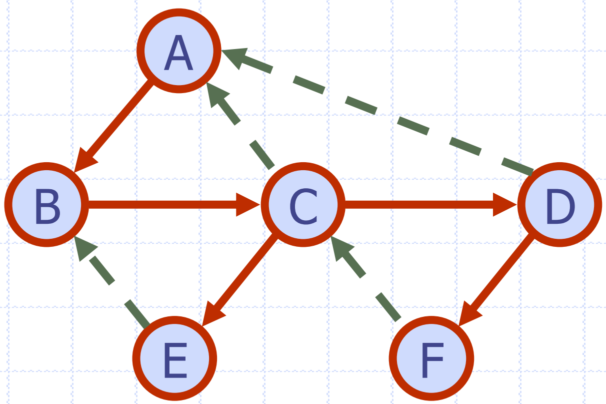
- Using the template method pattern, we can specialize the BFS traversal of a graph G to solve the following problems in $O(n + m)$ time
  - Compute the connected components of G
  - Compute a spanning forest of G
  - Find a simple cycle in G, or report that G is a forest
  - Given two vertices of G,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

Comparison: DFS vs. BFS

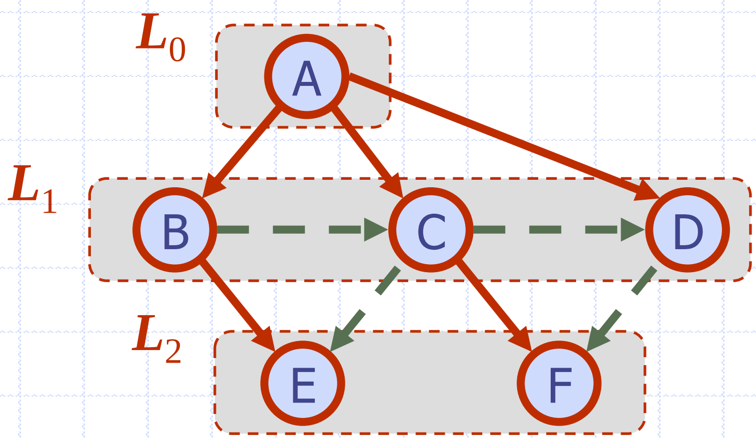
- DFS
  - Complexity: O(m+n)
  - Like preorder for binary trees
  - Can be achieved by a stack
  - Path finding with low memory
    - Game solution finding, such as maze traversal, 2048, nonograms, etc.
- BFS
  - Complexity: O(m+n)
  - Like level-order for binary tress
  - Can be achieved by a queue
  - Minimum path finding

DFS vs. BFS

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



DFS



BFS

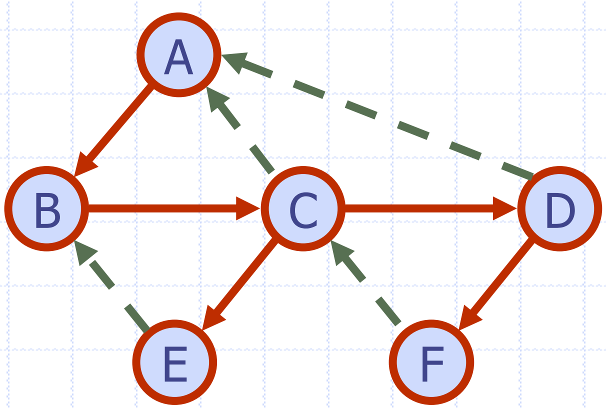
DFS vs. BFS (cont.)

Back edge (v,w)

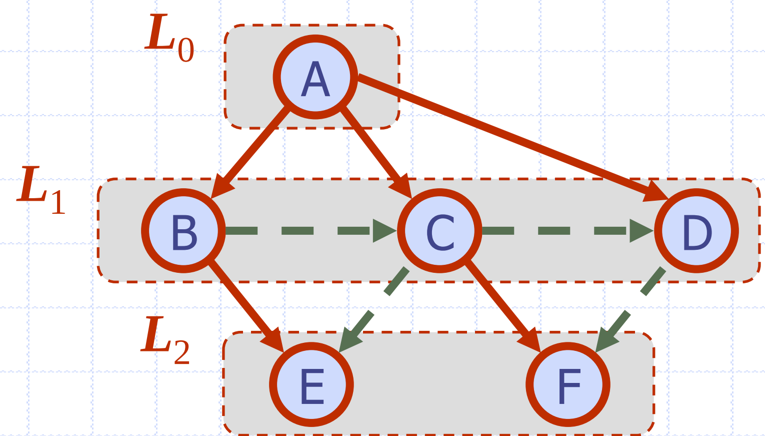
- w is an ancestor of v in the tree of discovery edges

Cross edge (v,w)

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



DFS



BFS

Applications of BFS

- BFS applications
  - Shortest path in a unweighted graph
  - Web crawler
  - Social network
  - Cycle detection
  - Bipartite graph determination
  - Broadcasting in a network
  - Folk-Fulkerson algorithm