CSE221

Lecture 19:
Graph Traversals

Hyungon Moon

Acknowledgment: The content of this file is based on the slides of the textbook as well as the slides provided by Prof. Won-Ki Jeong.

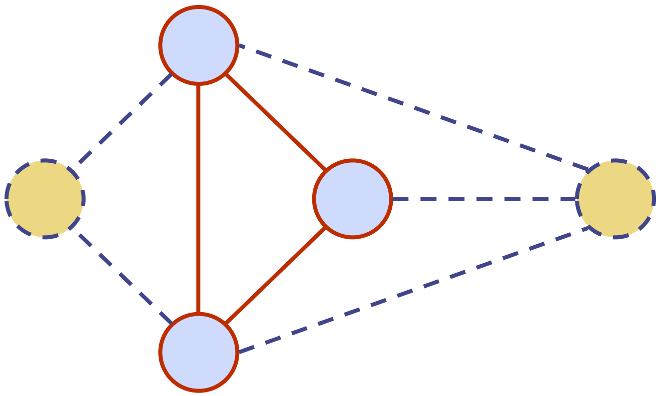
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Outline

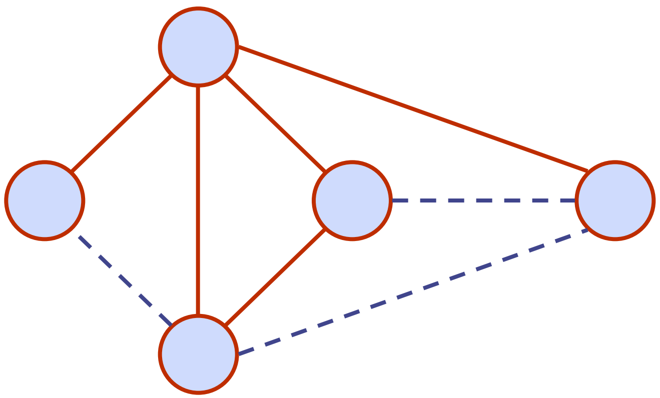
- Depth First Search (DFS)
- Breadth First Search (BFS)

Subgraphs

- A subgraph S of a graph G is a graph such that
  - The vertices of S are a subset of the vertices of G
  - The edges of S are a subset of the edges of G
- A spanning subgraph of G is a subgraph that contains all the vertices of G



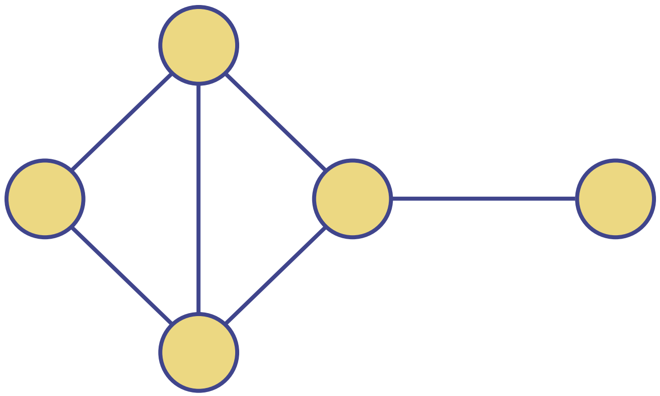
Subgraph



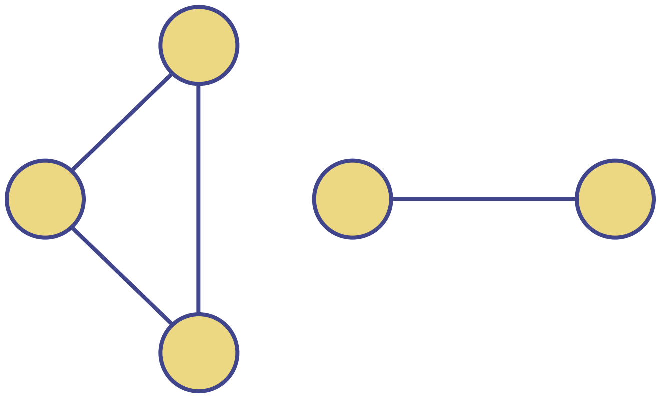
Spanning subgraph

Connectivity

- A graph is connected if there is a path between every pair of vertices
- A connected component of a graph G is a maximal connected subgraph of G



Connected graph

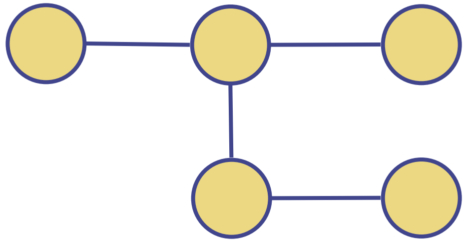


Non connected graph with two connected components

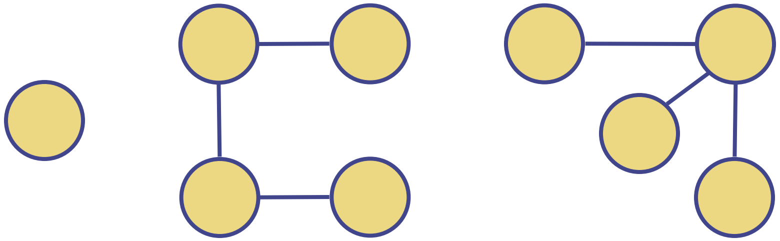
Trees and Forests

- A (free) tree is an undirected graph T such that
  - T is connected
  - T has no cycles

  This definition of tree is different from the one of a rooted tree
- A forest is an undirected graph without cycles
- The connected components of a forest are trees



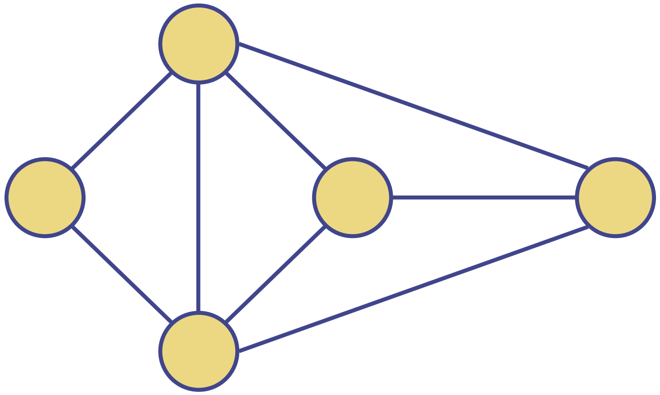
Tree



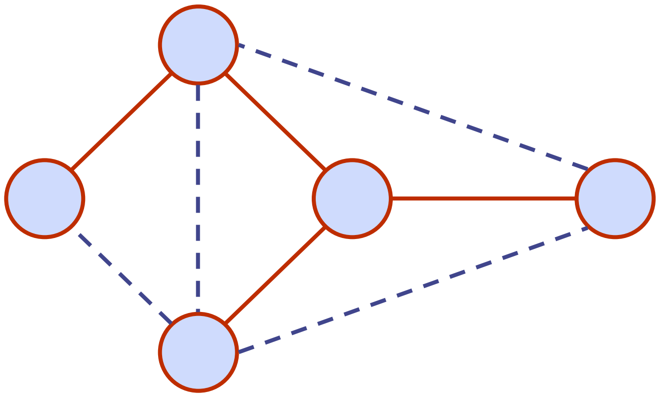
Forest

Spanning Trees and Forests

- A spanning tree of a connected graph is a spanning subgraph that is a tree
- A spanning tree is not unique unless the graph is a tree
- Spanning trees have applications to the design of communication networks
- A spanning forest of a graph is a spanning subgraph that is a forest



Graph



Spanning tree

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

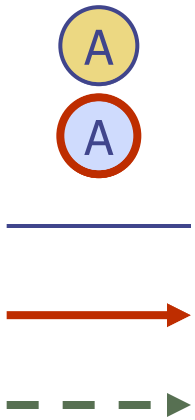
DFS Algorithm

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

```
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
  v.setLabel(VISITED)
  for all e ∈ G.incidentEdges(v)
    if e.getLabel() = UNEXPLORED
      w ← e.opposite(v)
      if w.getLabel() = UNEXPLORED
        e.setLabel(DISCOVERY)
        DFS(G, w)
      else
        e.setLabel(BACK)
```

Example



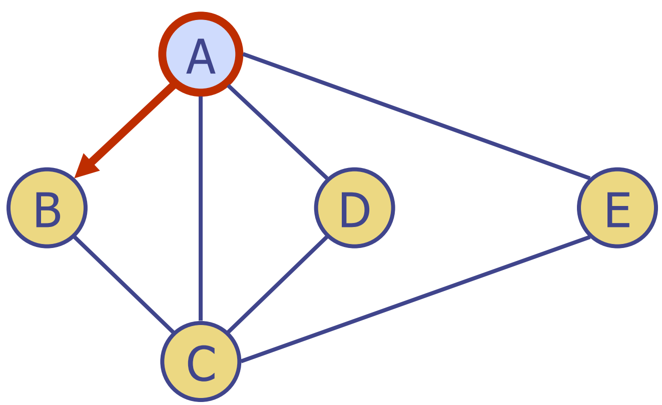
unexplored vertex

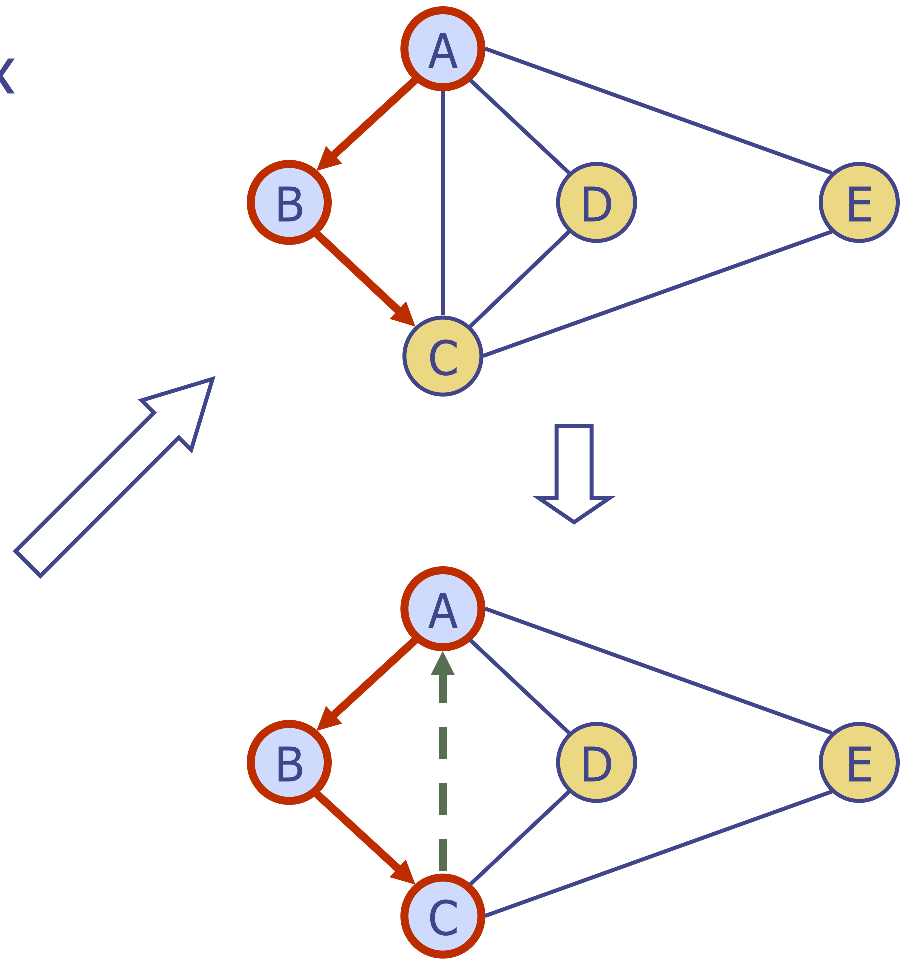
visited vertex

unexplored edge

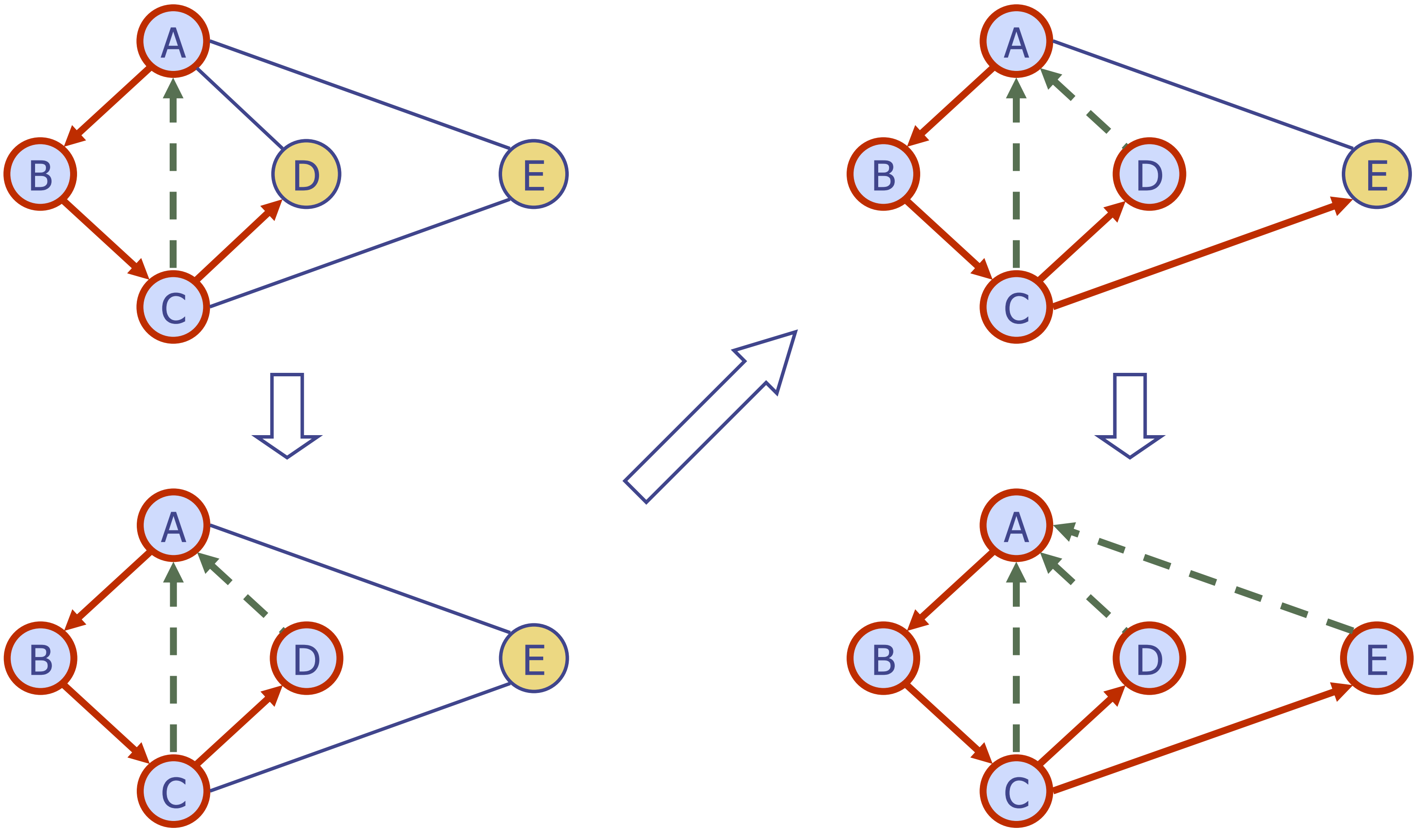
discovery edge

back edge

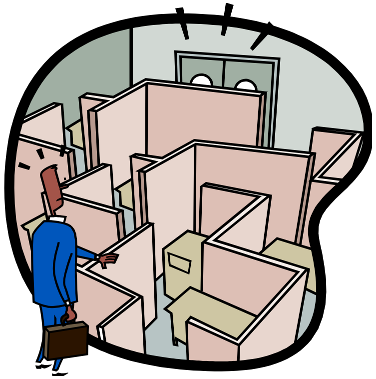




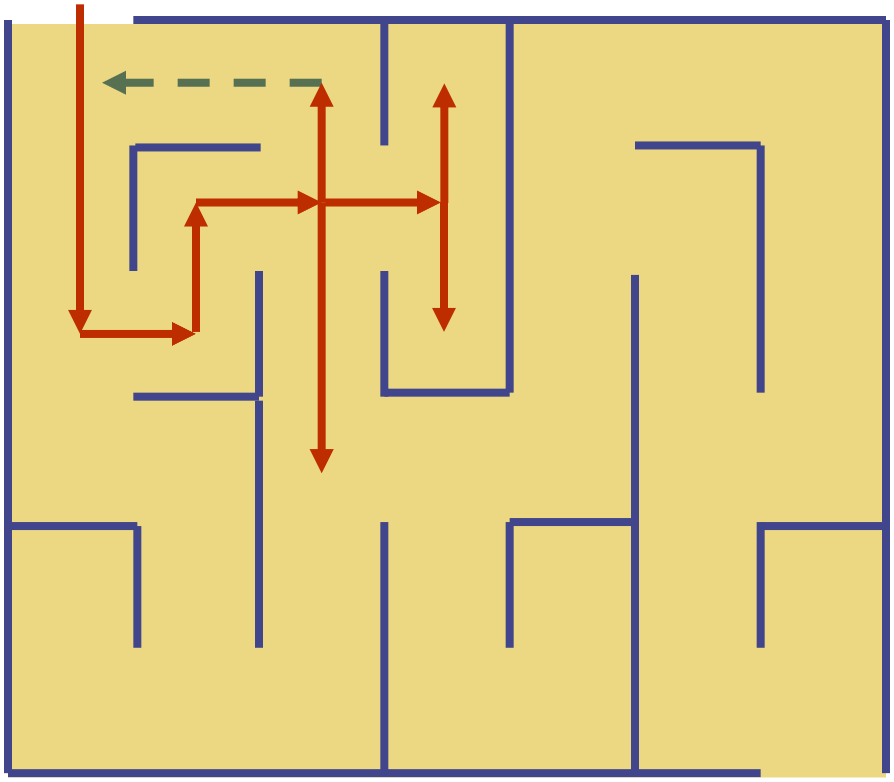
Example (cont.)



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)



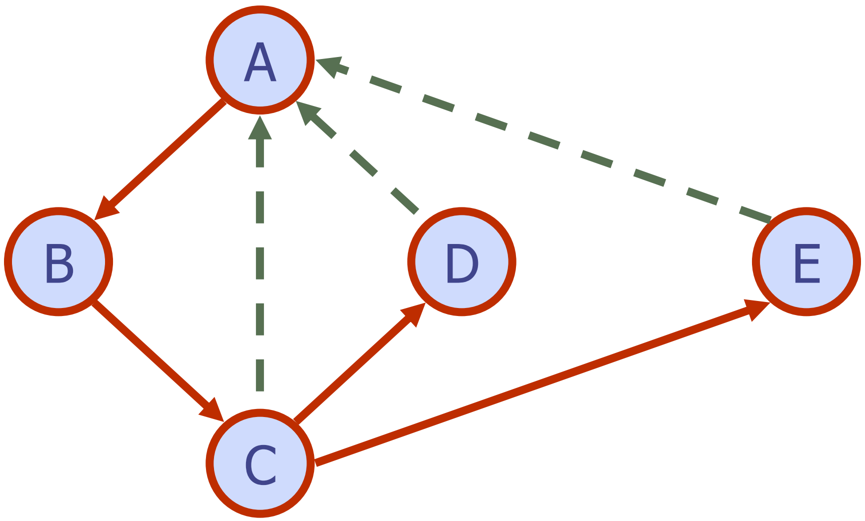
Properties of DFS

Property 1

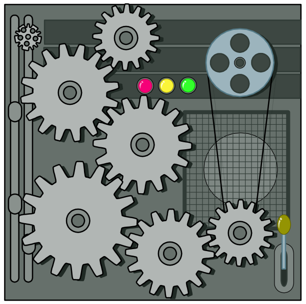
DFS(*G*, *v*) visits all the vertices and edges in the connected component of *v*

Property 2

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



Analysis of DFS



- 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 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
  - Recall that $\Sigma_v \deg(v) = 2m$

Path Finding

- We can specialize the DFS algorithm to find a path between two given vertices v and z 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 destination vertex z is encountered, we return the path as the contents of the stack

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

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 cycleDFS(G, v, z)
  v.setLabel(VISITED)
  S.push(v)
  for all  e ∈ v.incidentEdges()
    if  e.getLabel() = UNEXPLORED
      w ← e.opposite(v)
      if w.getLabel() = UNEXPLORED
        e.setLabel(DISCOVERY)
        cycleDFS(G, w, z)
      else
        S.push(w)
        return S
  S.pop()
```

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

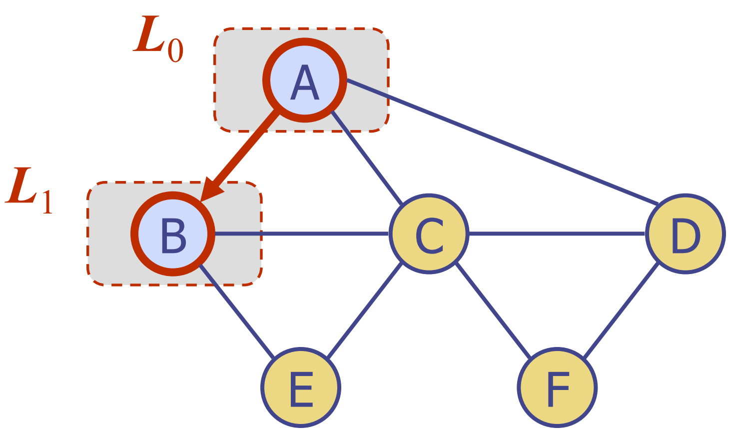
- The algorithm uses a mechanism for setting and getting "labels" of vertices and edges

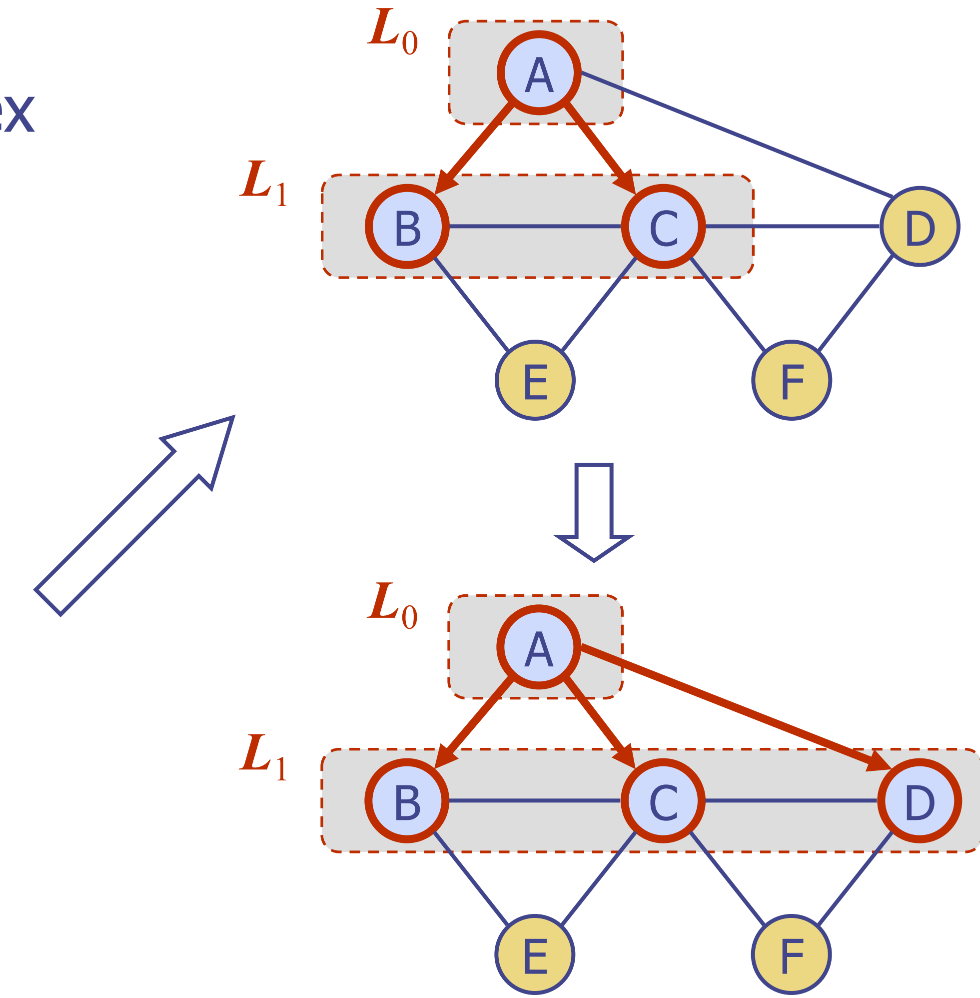
Algorithm $BFS(G)$
    Input graph G
    Output labeling of the edges and partition of the vertices of G
  for all $u \in G.vertices()$
    $u.setLabel(UNEXPLORED)$
  for all $e \in G.edges()$
    $e.setLabel(UNEXPLORED)$
  for all $v \in G.vertices()$
    if $v.getLabel() = UNEXPLORED$
      $BFS(G,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

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
            $e.setLabel(CROSS)$
    $i \leftarrow i + 1$

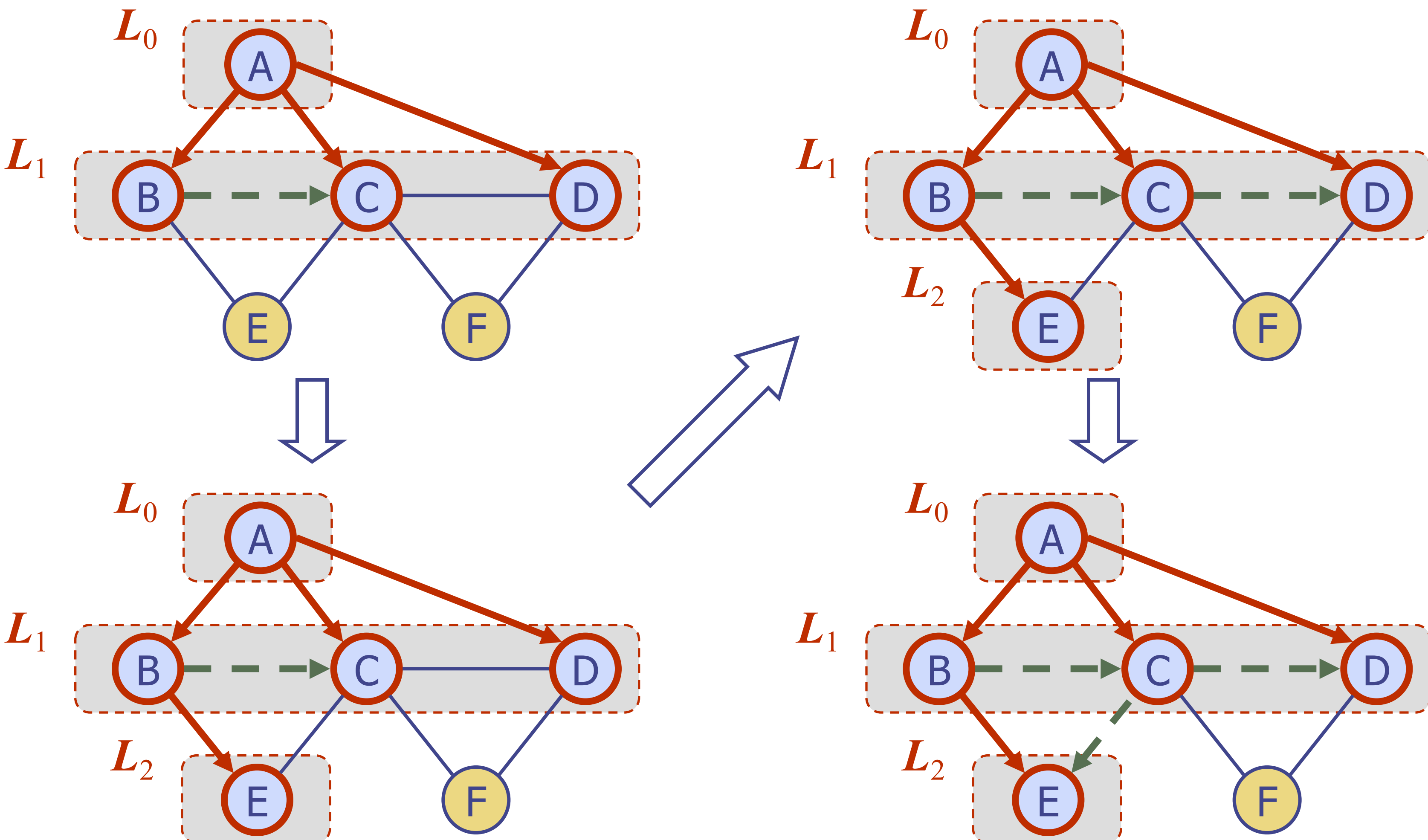
Example

A unexplored vertex
A visited vertex
unexplored edge
discovery edge
cross edge

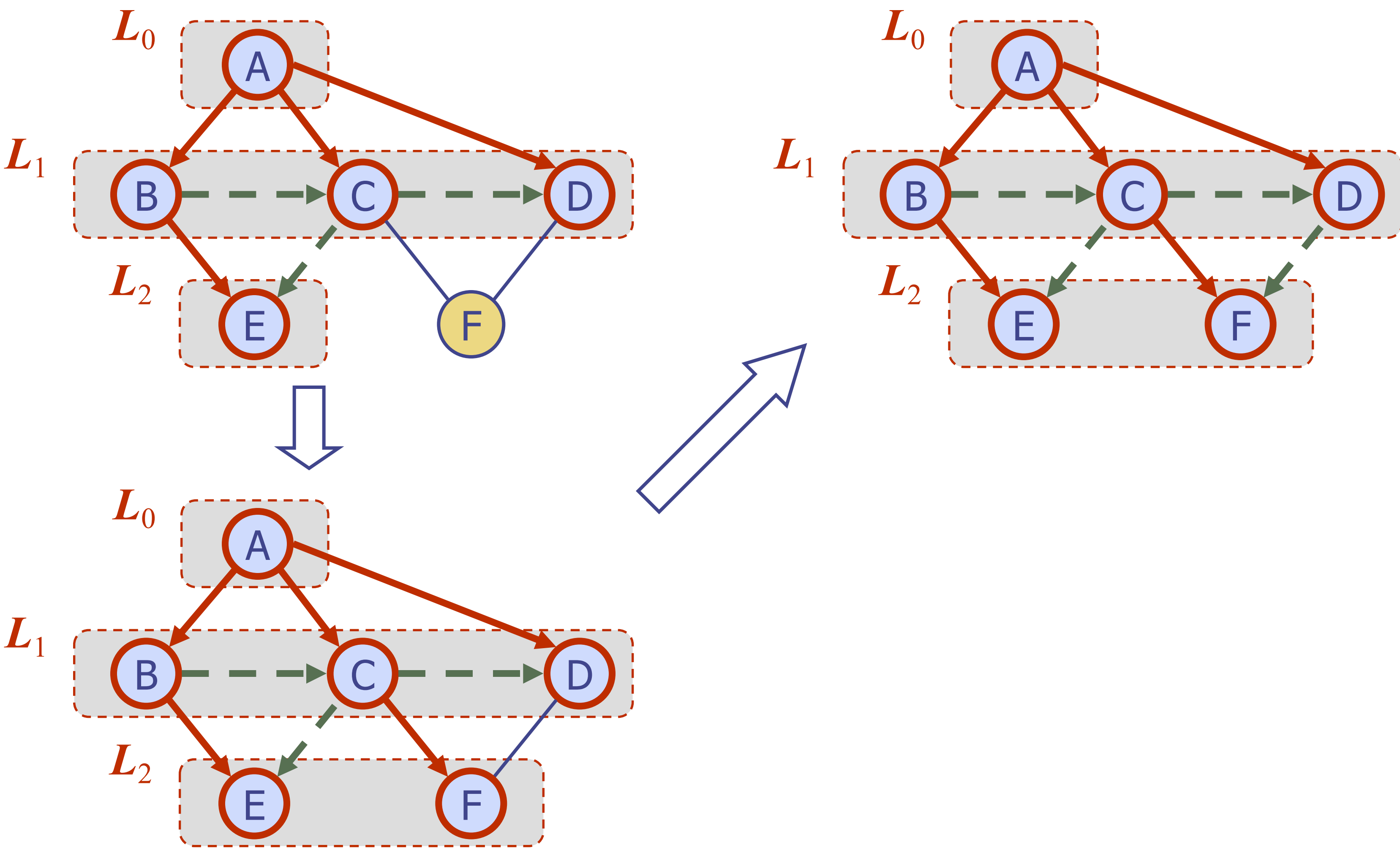




Example (cont.)



Example (cont.)



Properties

Notation

G_s: connected component of s

Property 1

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

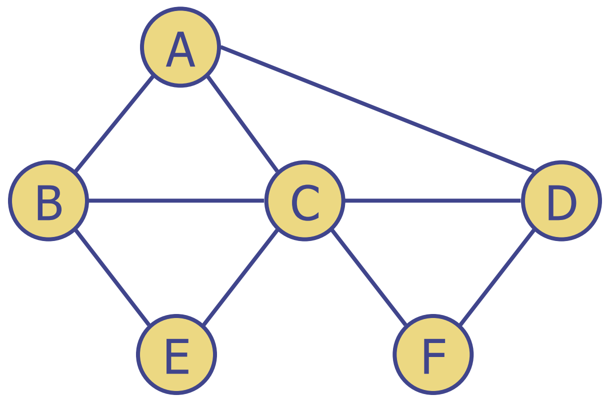
Property 2

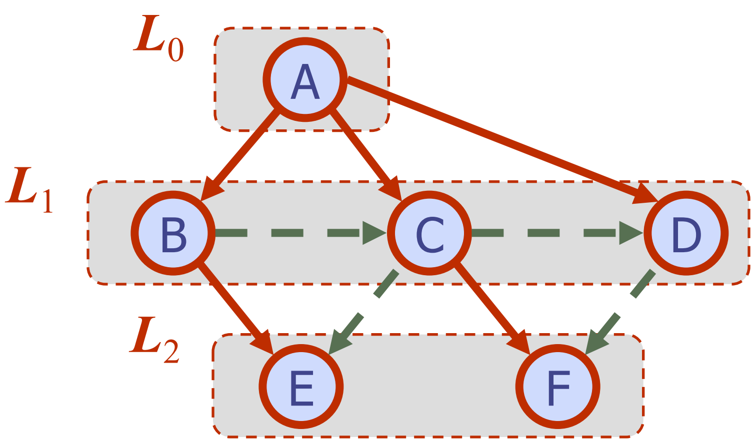
The discovery edges labeled by $BFS(G, s)$ form a spanning tree T_s of G_s

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

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

Questions?