Applications of BFS and DFS

cse2011

section 13.3 of textbook

Some Applications of BFS and DFS

- BFS
  - To find the shortest path from a vertex *s* to a vertex *v* in an unweighted graph
  - To find the length of such a path
  - To find out if a graph contains cycles
  - To construct a BSF tree/forest from a graph
- DFS
  - To find a path from a vertex *s* to a vertex *v.*
  - To find the length of such a path.
  - To construct a DSF tree/forest from a graph.

Testing for Cycles

Finding Cycles in Undirected Graphs

- To detect/find cycles in an ***undirected*** graph, we need to classify the edges into 3 categories during program execution:
  – unvisited edge: never visited.
  – discovery edge: visited for the very **first** time.
  – cross edge: edge that forms a cycle.
- When the BFS algorithm terminates, the discovery edges form a spanning tree.
- **If** there exists a **cross** edge, the undirected graph contains a cycle.

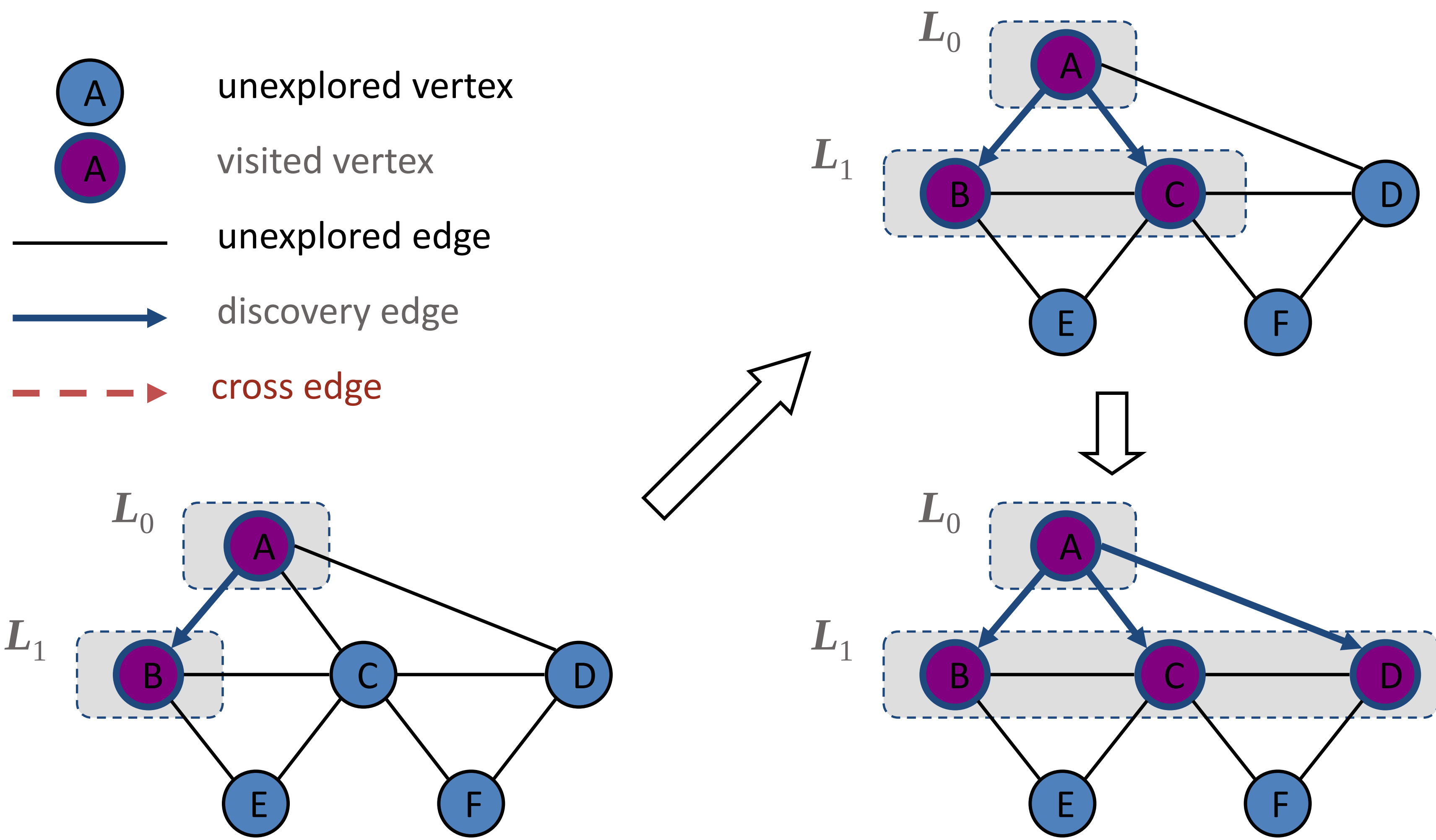
BFS Algorithm (in textbook)

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

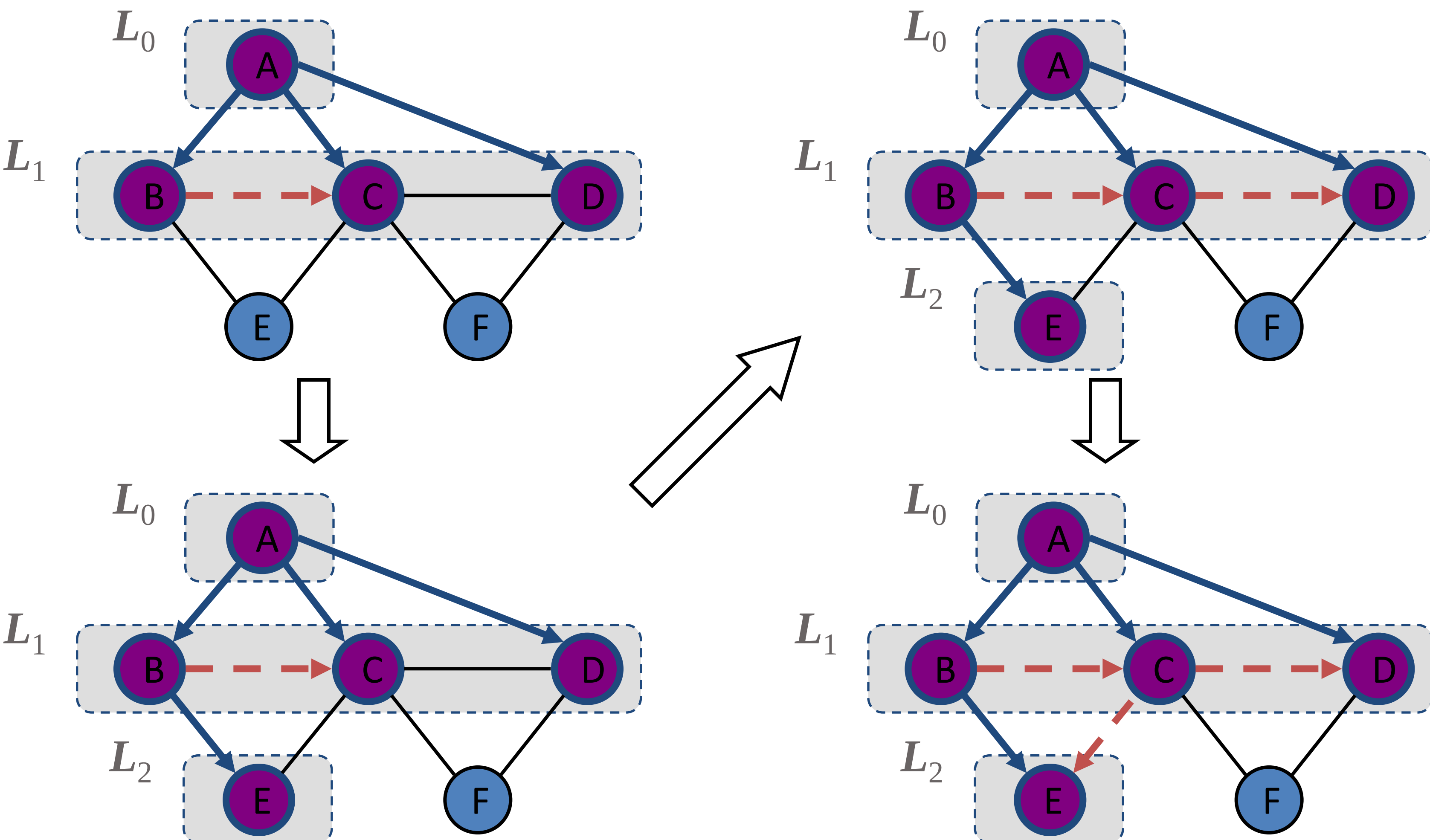
```
Algorithm BFS(G)
    Input graph G
    Output labeling of the edges
        and partition of the
        vertices  of G
  for all  u ∈ G.vertices()
    setLabel(u, UNEXPLORED)
  for all  e ∈ G.edges()
    setLabel(e, UNEXPLORED)
  for all  v ∈ G.vertices()
    if  getLabel(v) = UNEXPLORED
      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

```
Algorithm BFS(G, s)
  L_0 ← new empty sequence
  L_0.insertLast(s)
  setLabel(s, VISITED)
  i ← 0
  while  ¬L_i.isEmpty()
    L_{i+1} ← new empty sequence
    for all  v ∈ L_i.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)
            L_{i+1}.insertLast(w)
          else
            setLabel(e, CROSS)
    i ← i +1
```

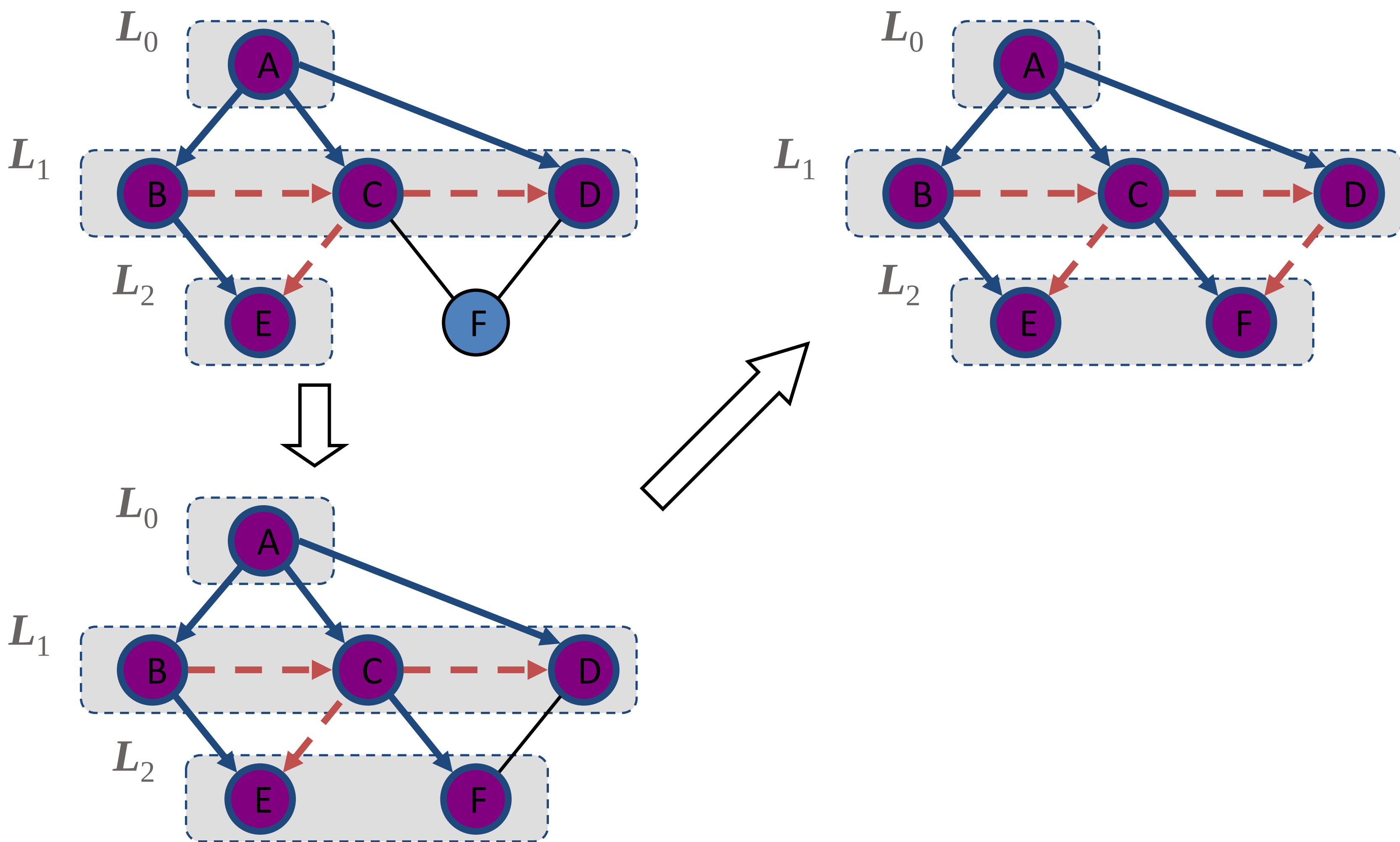
Example


A
unexplored vertex
A
visited vertex
unexplored edge
discovery edge
cross edge
L_0
L_1
A
B
C
D
E
F
L_0
L_1
A
B
C
D
E
F
L_0
L_1
A
B
C
D
E
F

Example (2)



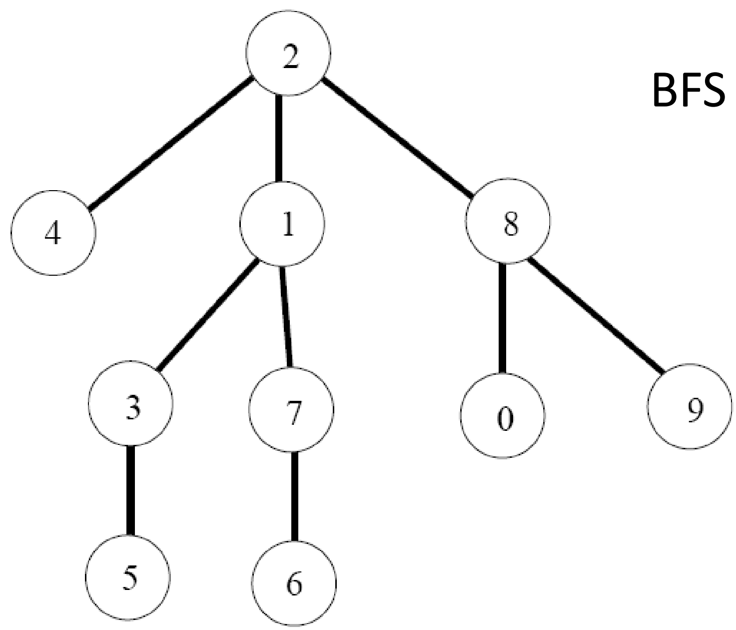
Example (3)



Computing Spanning Trees

Trees

- Tree: a connected graph without cycles.
- Given a connected graph, remove the cycles $\Rightarrow$ a tree.
- The paths found by *BFS*(*s*) form a rooted tree (called a *spanning tree*), with the starting vertex as the root of the tree.



BFS tree for vertex s = 2

What would a level-order traversal of the tree tell you?

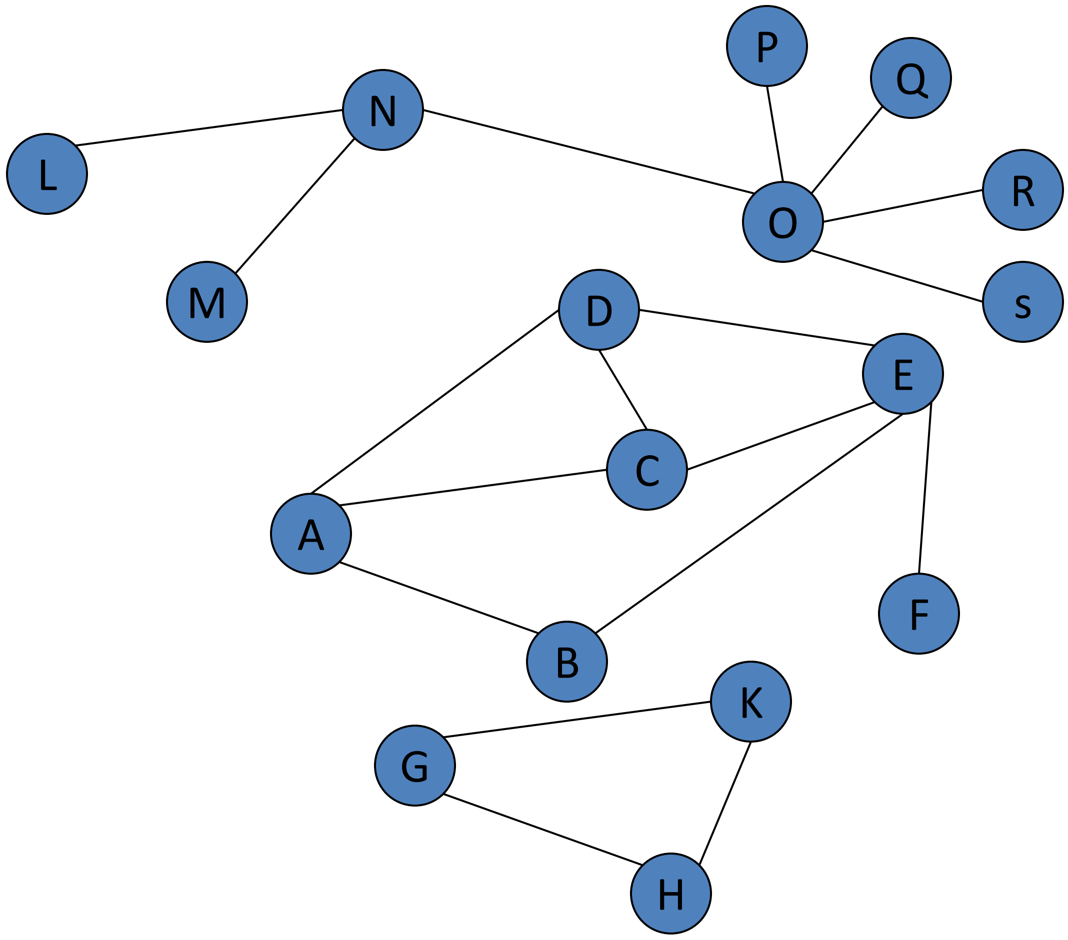
Computing Spanning Forests

Computing a BFS Forest

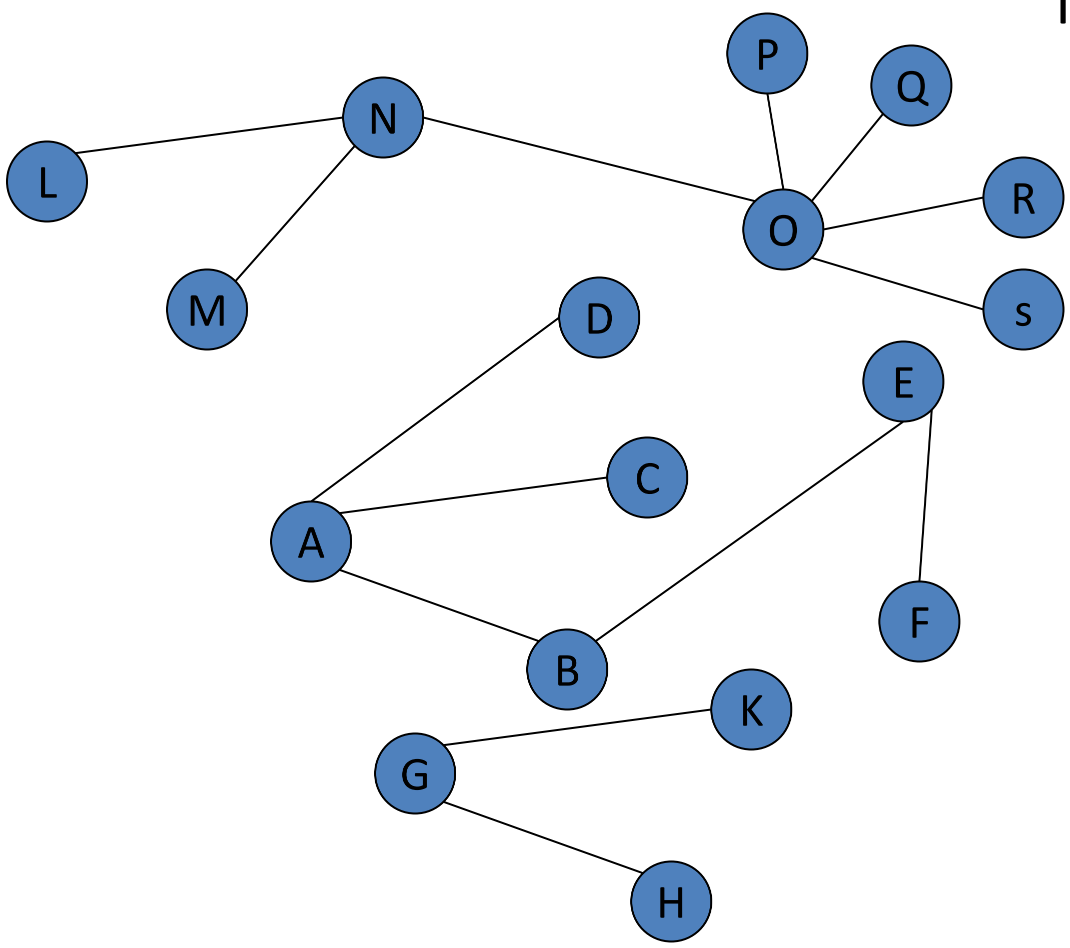
- A forest is a set of trees.
- A connected graph gives a tree (which is itself a forest).
- A connected component also gives us a tree.
- A graph with ***k*** components gives a **forest** of ***k*** **trees**.

Example

A graph with 3 components



Example of a Forest



We removed the cycles from the previous graph.

A forest with 3 trees

Applications of DFS

- Is there a path from source *s* to a vertex *v*?
- Is an undirected graph connected?
- Is a directed graph strongly connected?
- To output the contents (e.g., the vertices) of a graph
- To find the connected components of a graph
- To find out if a graph contains cycles and report cycles.
- To construct a DSF tree/forest from a graph

Finding Cycles Using DFS

- **Similar** to using **BFS**.

- For undirected graphs, classify the edges into 3 categories during program execution: **unvisited** edge, **discovery** edge, and **back (cross)** edge.
  - If there exists a back edge, the undirected graph contains a cycle.

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()
    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 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
    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)
```

Example

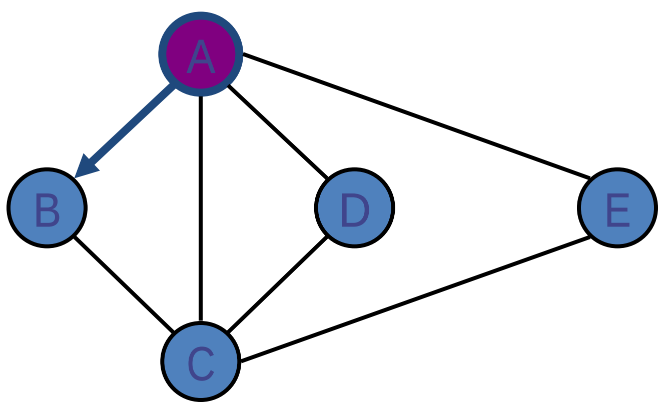
unexplored vertex

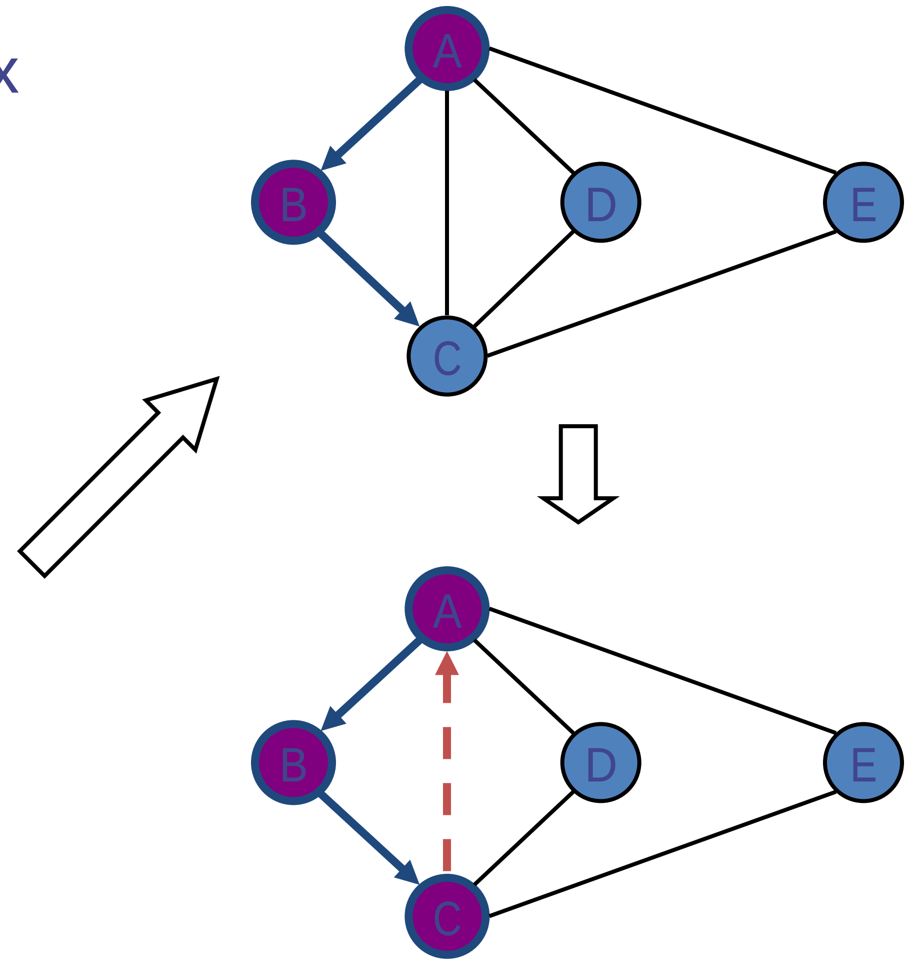
visited vertex

unexplored edge

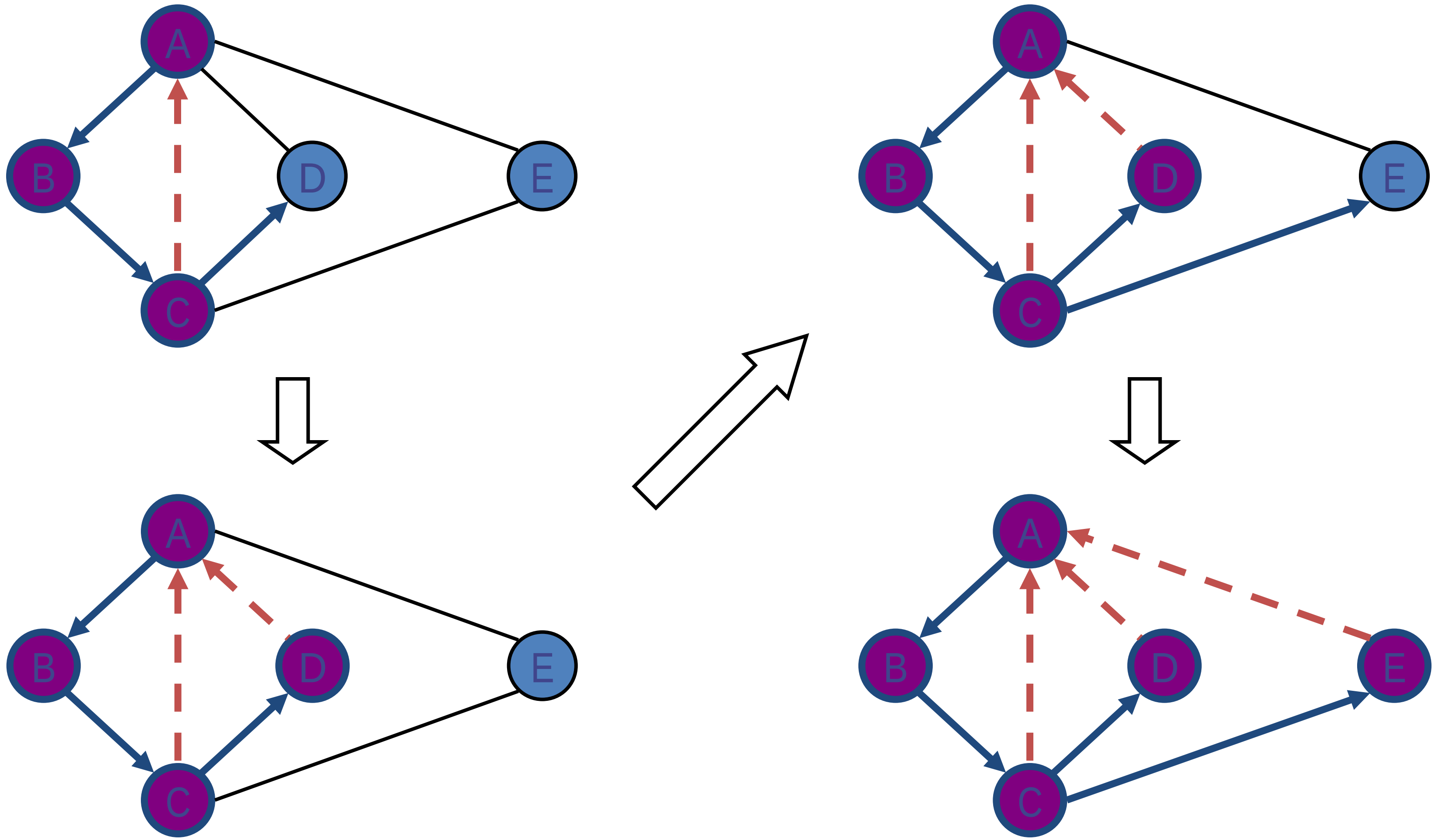
discovery edge

back edge



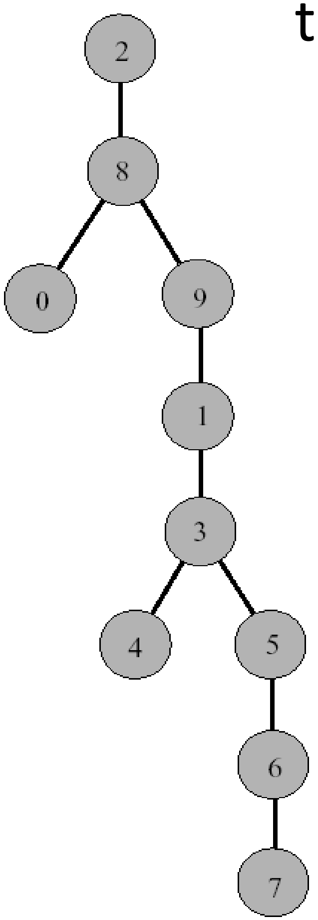


Example (cont.)



DFS Tree

Resulting DFS-tree.
Notice it is much "**deeper**"
than the BFS tree.



Captures the structure of the recursive calls:

- when we visit a neighbor w of v, we add w as child of v
- whenever DFS returns from a vertex v, we climb up in the tree from v to its parent

Next lecture …

- Review

- Final exam (Thursday August 8th in CLH E from 14:00 to 17:00.)