Graph Traversal Algorithms II

Instructor: Meng-Fen Chiang

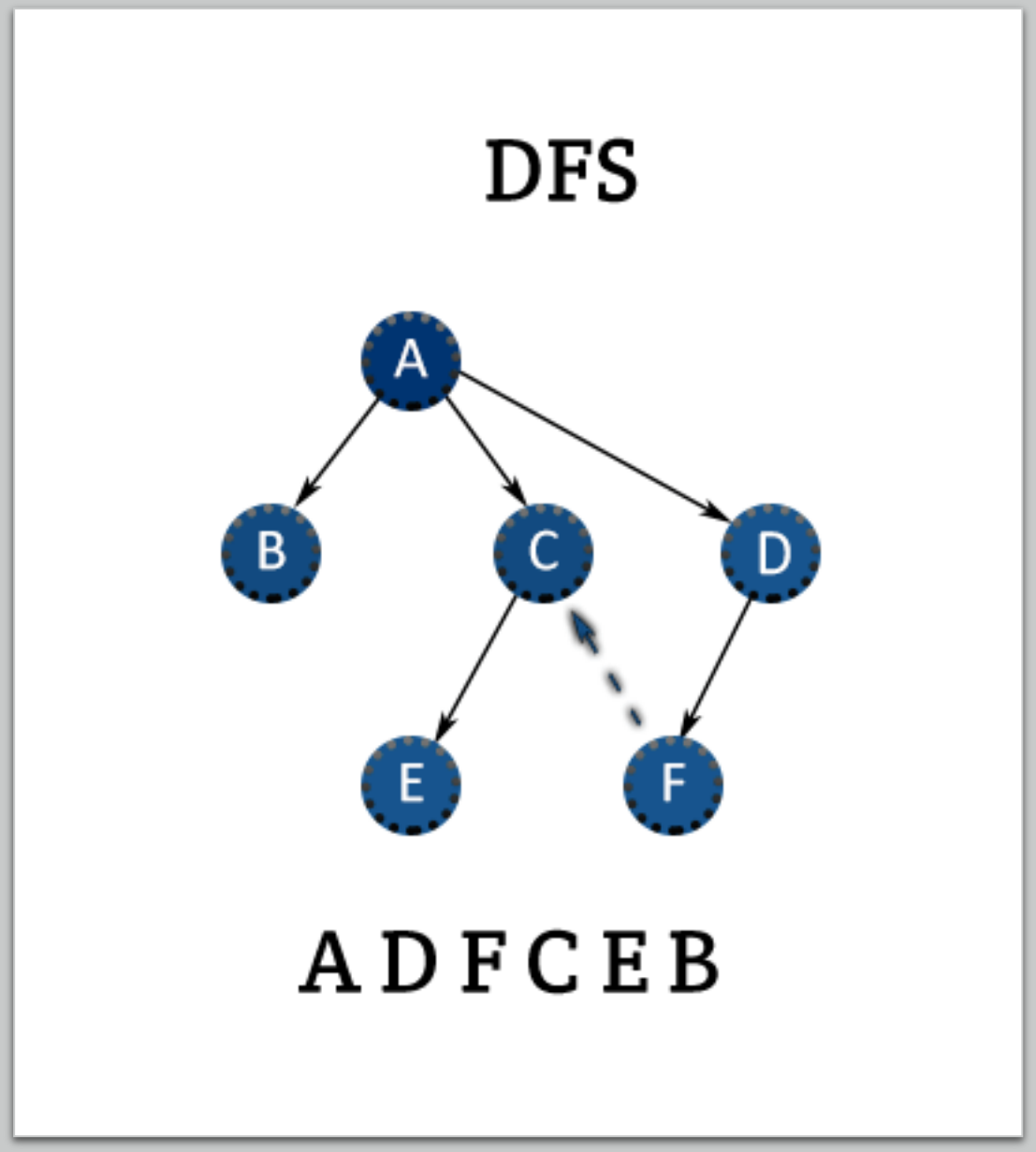
COMPSCI: WEEK 9.4

Slides adapted from Mark Wilson, Georgy Gimel'farb, Simone Linz and Tanya Gvozdeva

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

OUTLINE

- Graph Traversal Algorithms
  - Depth-first Search (DFS)
  - Breadth-first Search (BFS)
  - Priority-first Search (PFS)
- Implementation
  - Stack – DFS
  - Queue – BFS
  - Priority Queues – PFS



Graph Traversals

Algorithm 1 Visit.

```
1: function VISIT(node s of digraph G)
2:         color[s] ← Grey
3:         pred[s] ← Null
4:         while there is a Grey node do
5:             choose a Grey node u
6:             if u has a WHITE (out-)neighbour then
7:                 choose such a (out-)neighbour v
8:                 color[v] ← Grey
9:                 pred[v] ← u
10:            else
11:                color[u] ← Black
```

how to choose?

Implementation of the List of Frontiers

- The implementation of the list that stores the frontiers (grey nodes) will directly affect the order we traverse the graph nodes. Three types of implementations will be discussed and result in three different traversal strategies:

- Stack – Depth-first search (DFS)

- Queue – Breadth-first search (BFS)

- Priority Queues – Priority first search (PFS)

The Abstract Data Type: Stack

- Special list in which all operations occur at the same end (top) (last in first out).
- Add an element to the list (INSERT or PUSH).
- Delete and element (DELETE or POP).
- Return top element without deleting it (GETTOP or PEEK)

Depth-first Search Algorithm (DFS)

- DFS is a specific implementation of our fundamental graph traversal algorithm (also known as depth-first traversal)

- It specifies that we select the next grey vertex to pick as the youngest remaining grey vertex.

Depth-first-search (DFS) Algorithm

Algorithm 1 Depth-first search algorithm

```
1: function DFS(digraph G)
2:         stack S
3:         array colour[0..n − 1], pred[0..n − 1], seen[0..n − 1], done[0..n − 1]
4:         for u ∈ V(G) do
5:                 colour[u] ← WHITE
6:                 pred[u] ← null
7:         time ← 0
8:         for s ∈ V(G) do
9:                 if colour[s] = WHITE then
10:                        DFSVISIT(s)
11:        return pred, seen, done
```

Iterative View of DFSVISIT

Algorithm 2 Depth-first visit algorithm.

```
1: function DFSVISIT(node s)
2:         color[s] ← GREY
3:         seen[s] ← time; time ← time + 1
4:         S.insert(s)
5:         while not S.isEmpty() do
6:             u ← S.peek()
7:             if there is a neighbour v with colour[v] = WHITE then
8:                 colour[v] ← GREY; pred[v] ← u
9:                 seen[v] ← time; time ← time + 1
10:                S.insert(v)
11:            else
12:                S.delete()
13:                colour[u] ← BLACK
14:                done[u] ← time; time ← time + 1
```

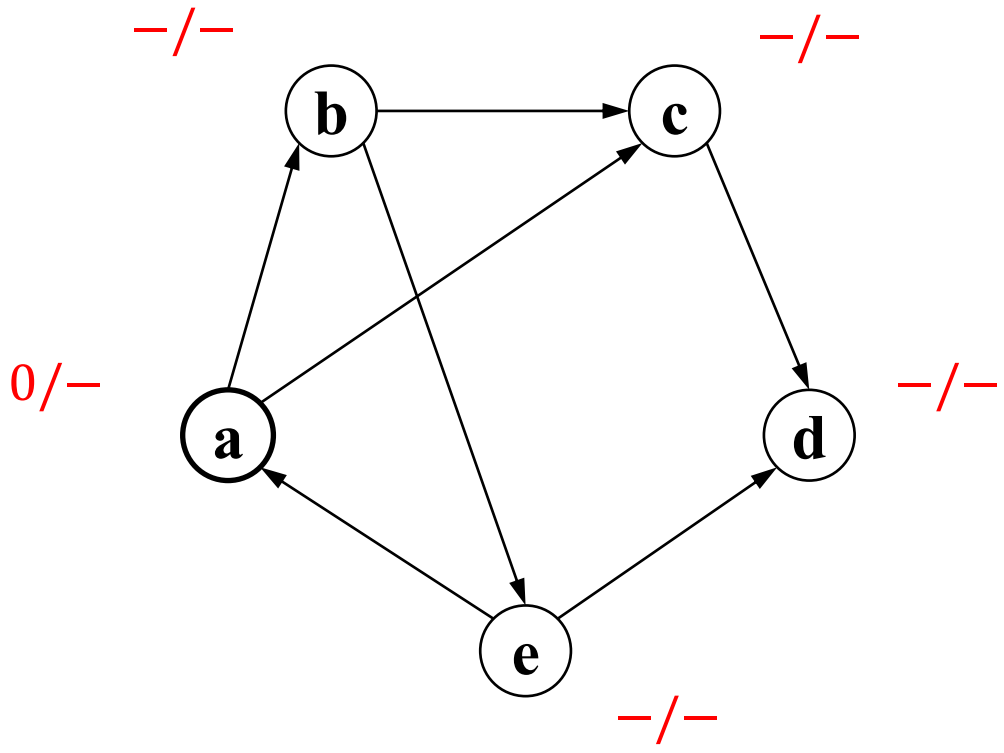
Recursive View of DFSVISIT

Algorithm 3 Recursive DFS visit algorithm.

```
1: function REC_DFSVISIT(node s)
2:         color[s] ← GREY
3:         seen[s] ← time; time ← time + 1
4:         for each v adjacent to s do
5:             if colour[v] = WHITE then
6:                 pred[v] ← s
7:                 REC_DFSVISIT(v)
8:         colour[s] ← BLACK
9:         done[s] ← time; time ← time + 1
```

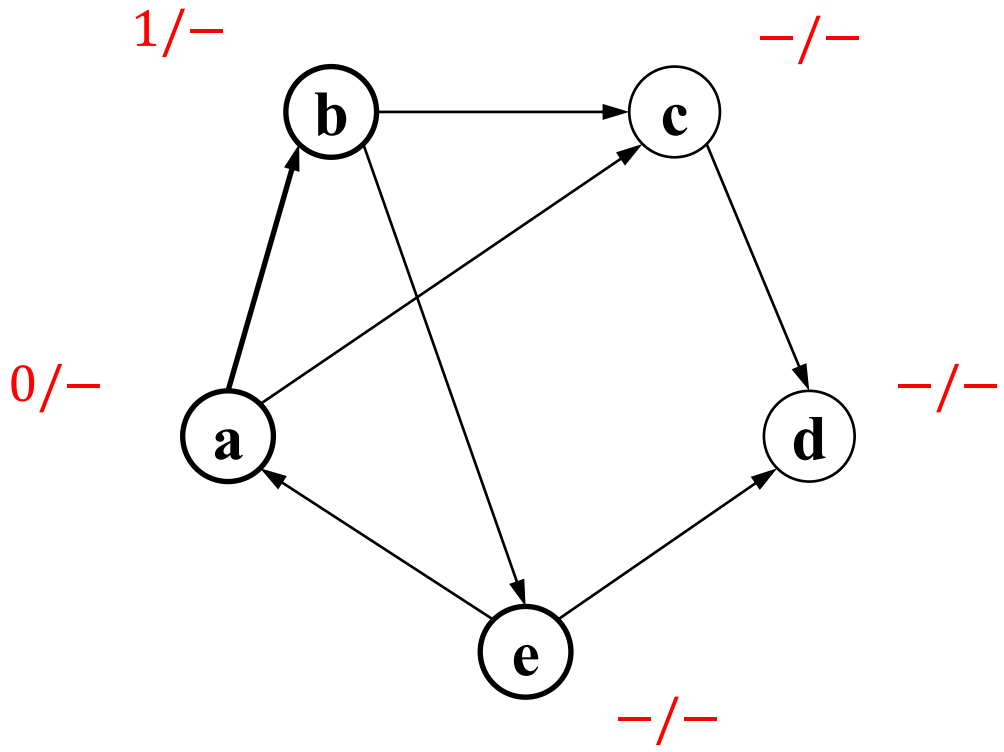
A DFS example (1)

seen/done



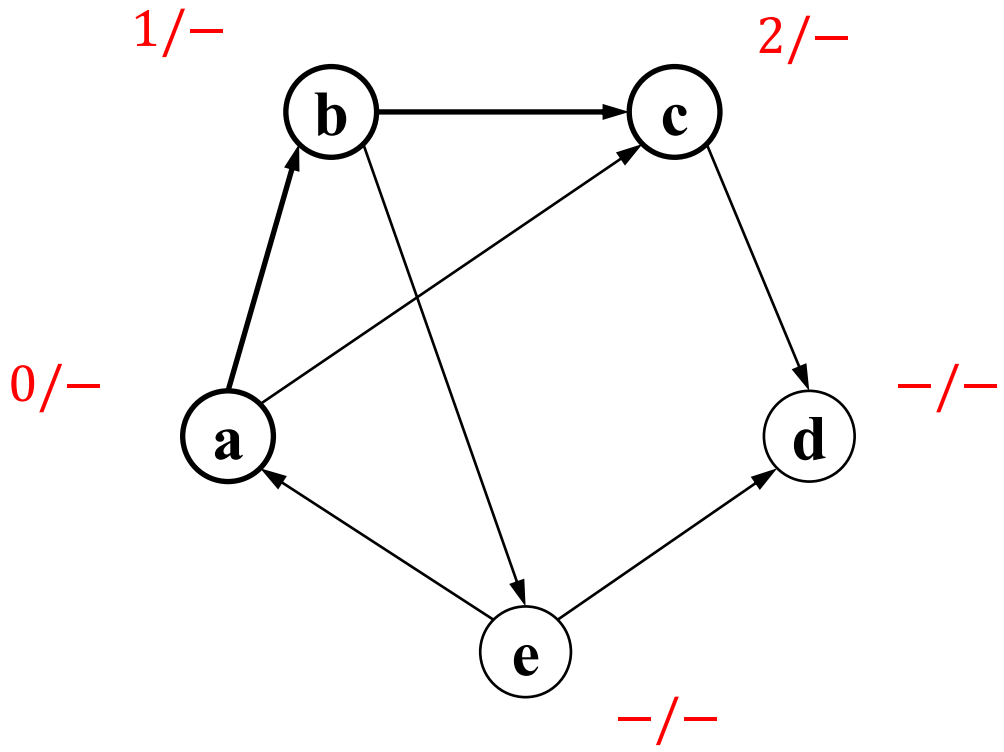
A DFS example (1)

seen/done



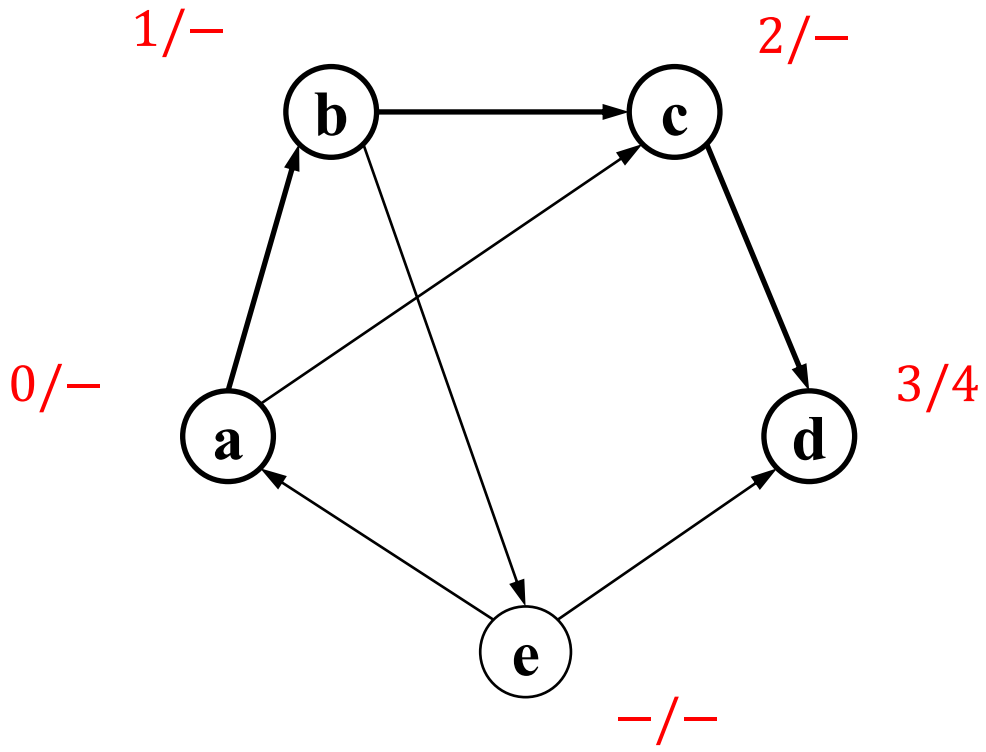
A DFS example (1)

seen/done



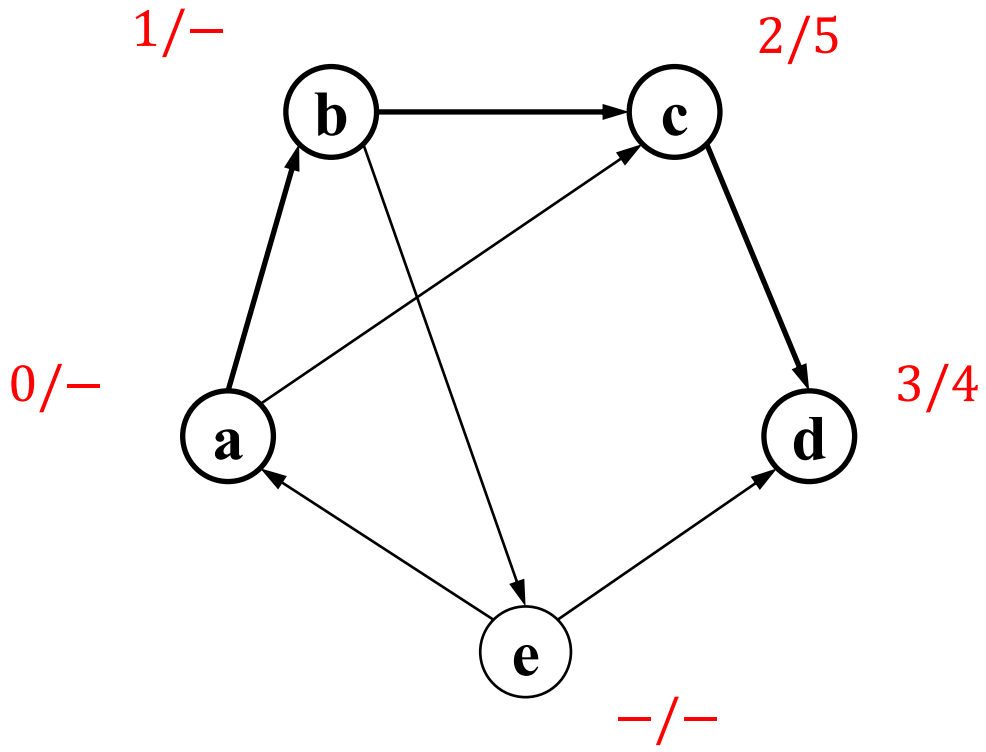
A DFS example (1)

seen/done



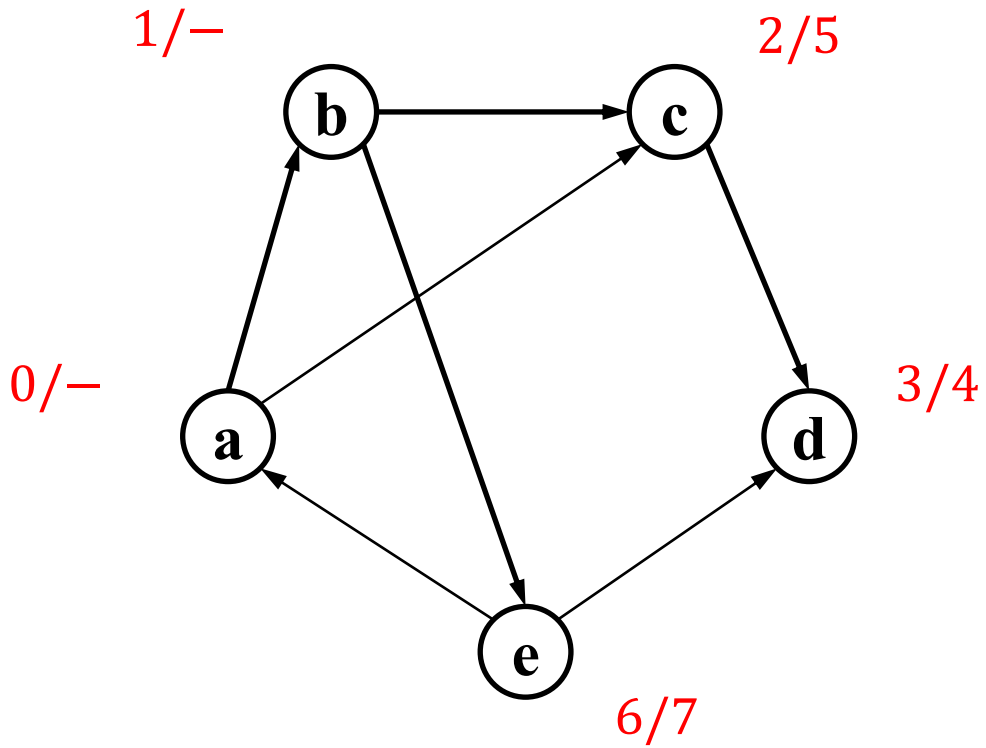
A DFS example (1)

seen/done



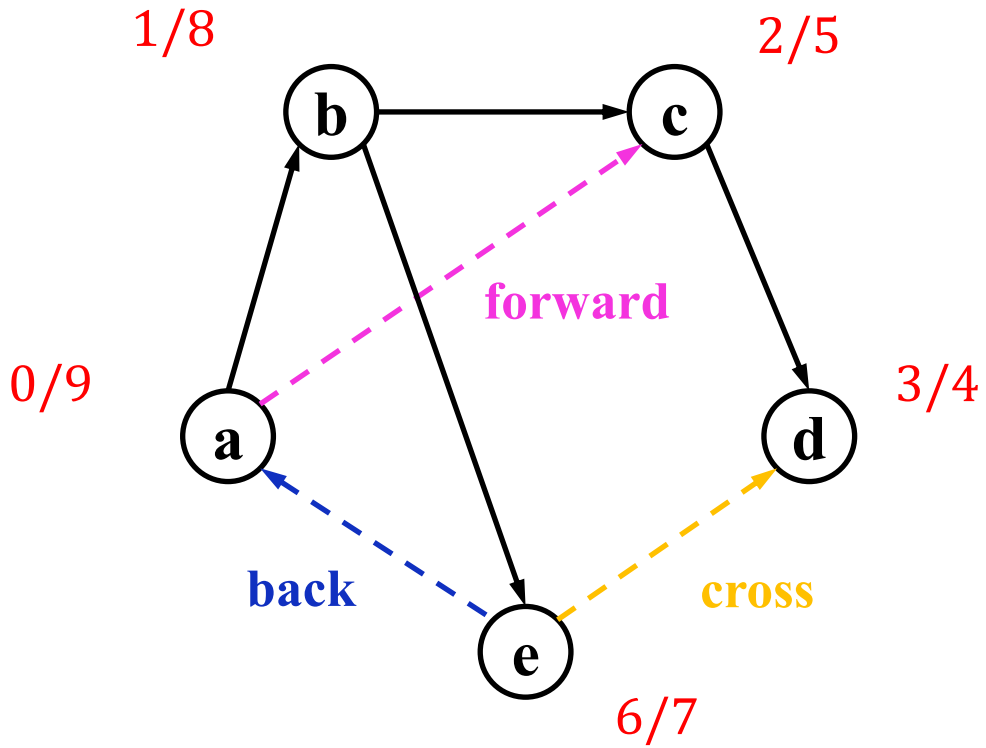
A DFS example (1)

seen/done



A DFS example (1)

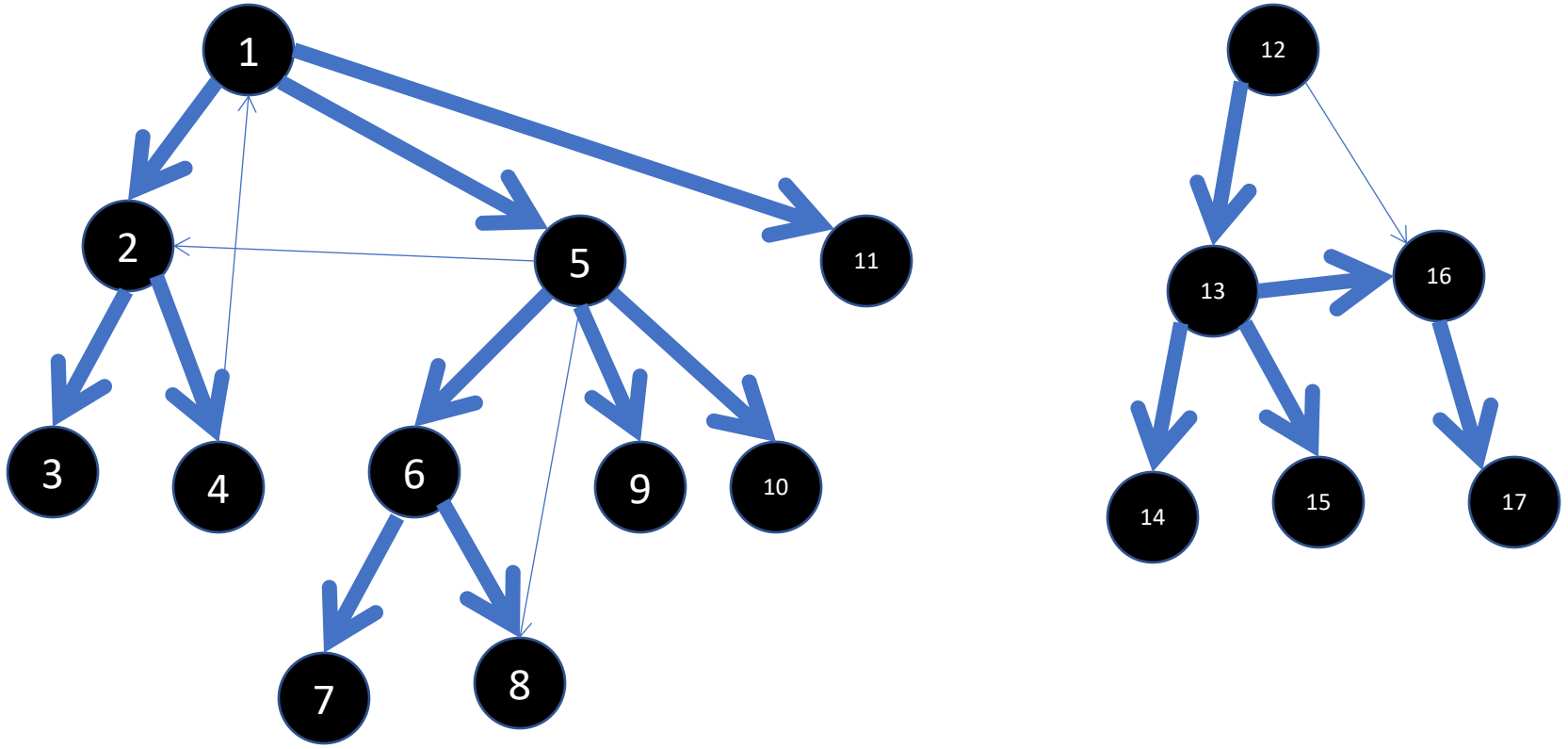
seen/done



RECAP: Four Types of Arcs

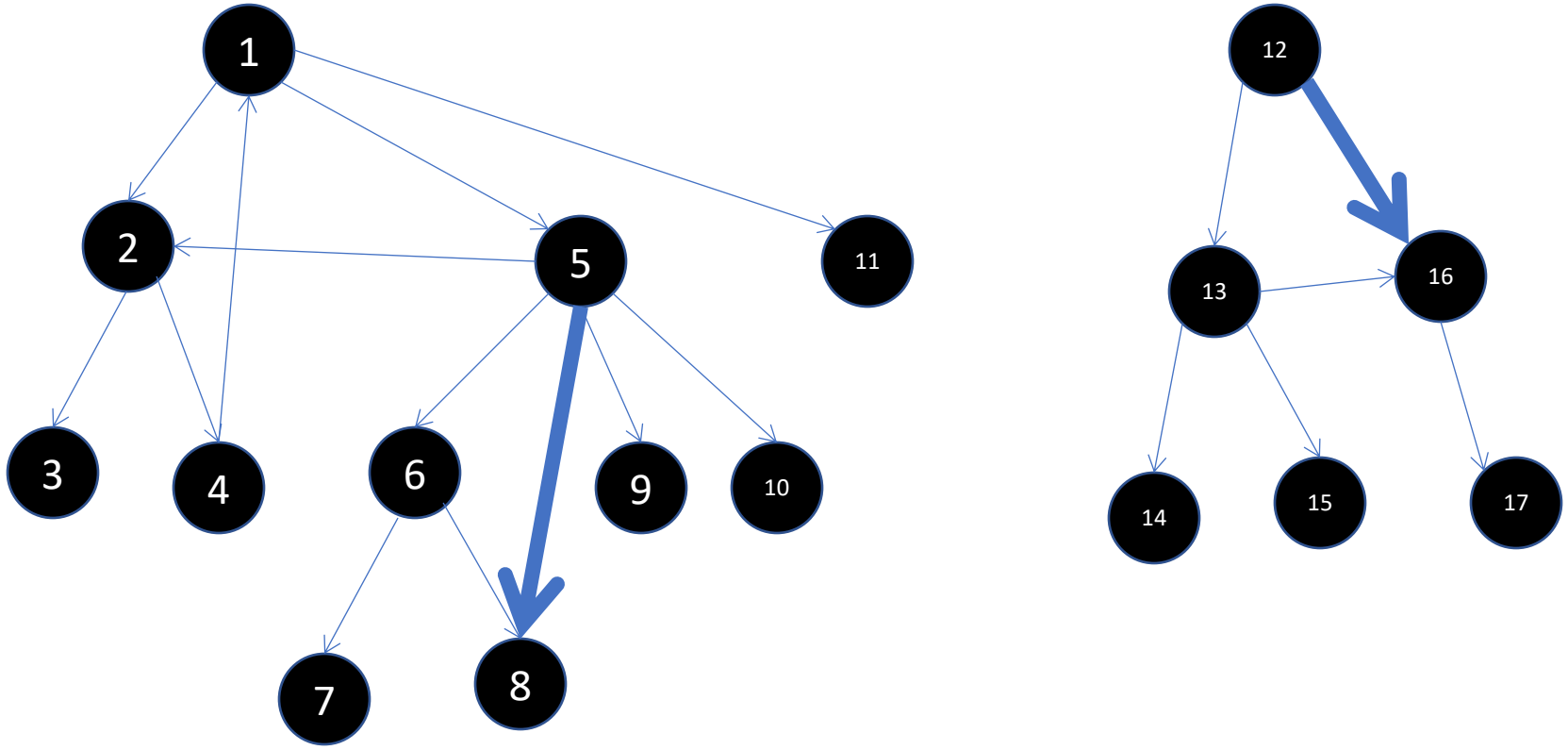
- In a search forest F of a graph G, we can find four different kinds of arcs:
- Tree arc: an arc in G that connects a vertex in G to one of its immediate descendants in the tree of F that the vertex belongs to, i.e., if the arc belongs to the tree
- Forward arc: an arc that does not belong to a tree in F but that connects a vertex to one of its descendants in the tree
- Back arcs: an arc that does not belong to a tree in F but that connects a vertex to one of its ancestors in the tree
- Cross arcs: arcs that fall into neither of the above categories

DFS Traversal: Tree Arcs



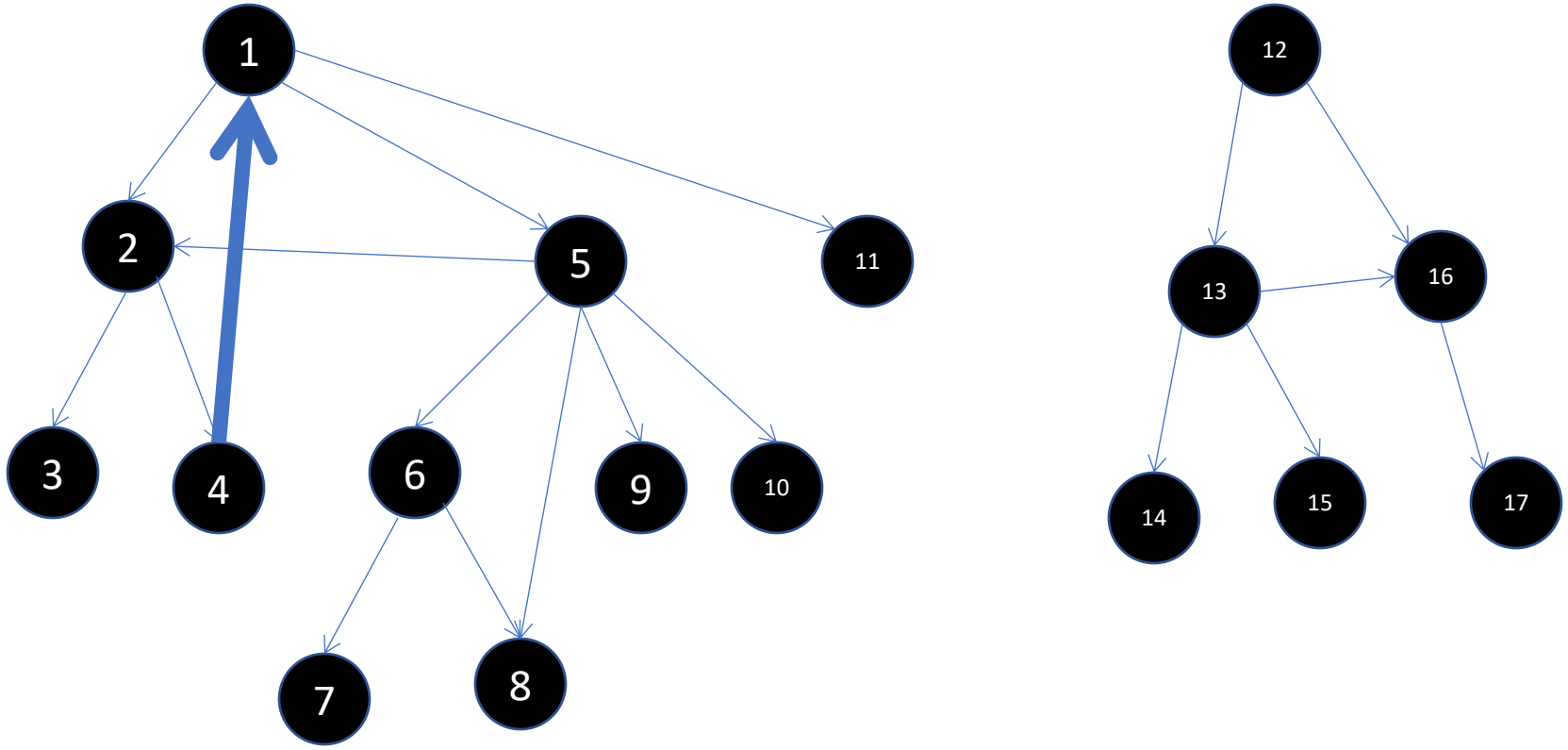
DFS Traversal: Forward Arcs

Forward arc: an arc that does not belong to a tree in F but that connects a vertex to one of its descendants in the tree



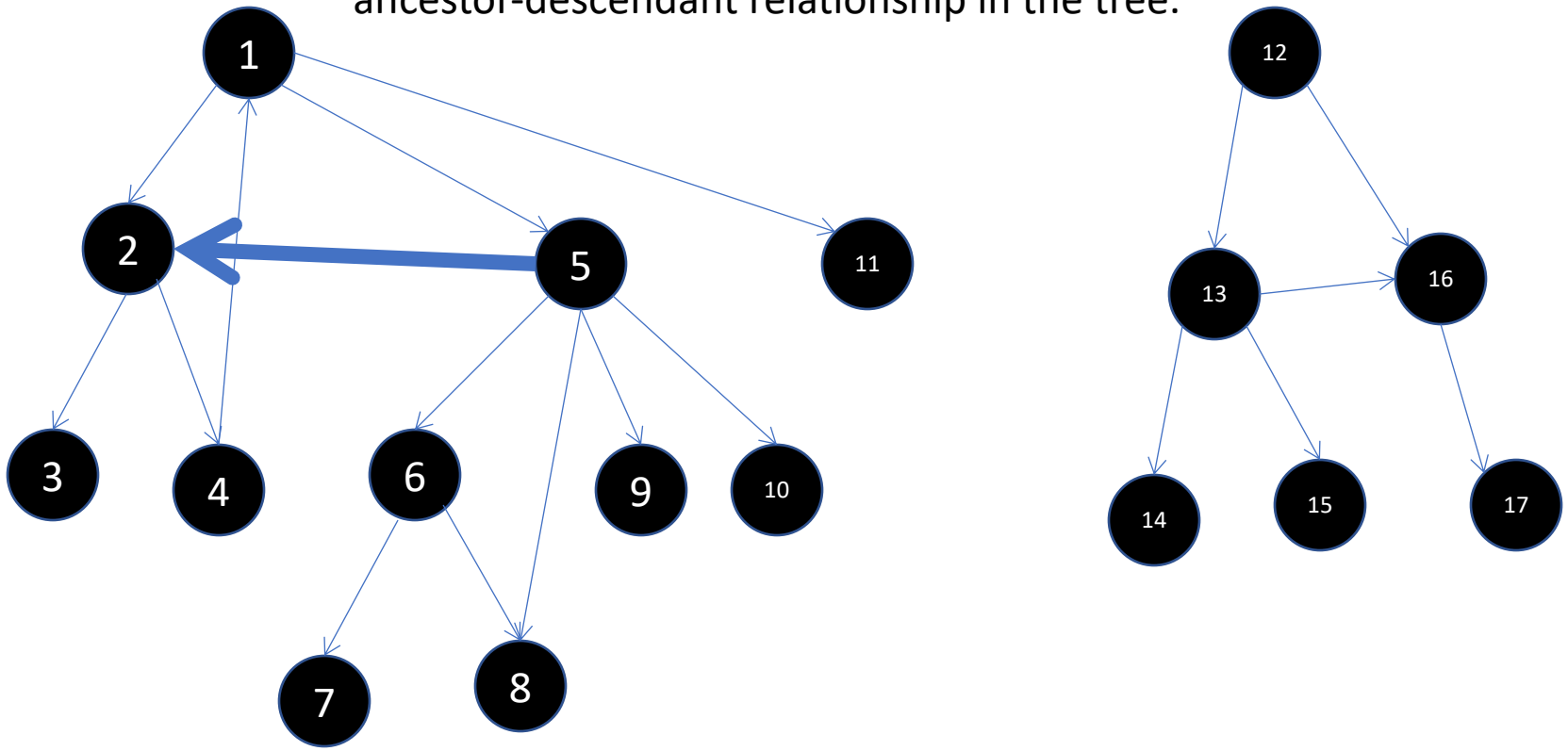
DFS Traversal: Back Arc

Back arcs: an arc that does not belong to a tree in F but that connects a vertex to one of its ancestors in the tree



DFS Traversal: Cross Arc

Cross arcs: an arc that does not belong to a tree in F and connects nodes that do not form an ancestor-descendant relationship in the tree.



Basic properties of depth-first search

- Each call to DFSVISIT(v) terminates only when all nodes reachable from v via a path of white nodes have been seen.

- Suppose that (v,w) is an arc of a digraph. Cases:
  - tree or forward arc: $seen[v]<seen[w]<done[w]<done[v]$;
  - back arc: $seen[w]<seen[v]< done[v]< done[w]$;
  - cross arc: $seen[w]<done[w]<seen[v]<done[v]$.
- Note that there are no cross edges on a graph G. (Why?)

Using DFS to Determine Ancestors of a Tree

- **Theorem**: Suppose that we have performed DFS on a digraph G, resulting in a search forest F. Let $v, w \in V(G)$ and suppose that $seen[v] < seen[w]$.

- If v is an ancestor of w in F, then

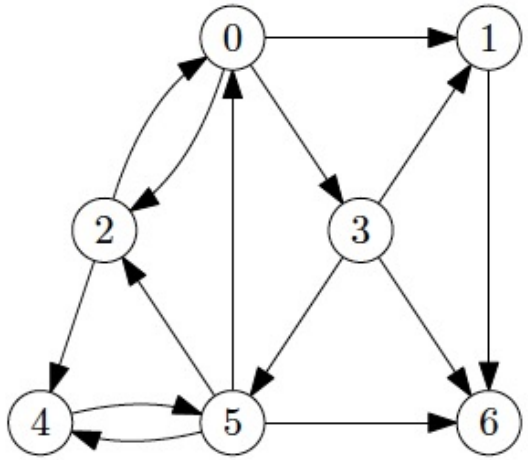
  $$seen[v] < seen[w] < done[w] < done[v] .$$

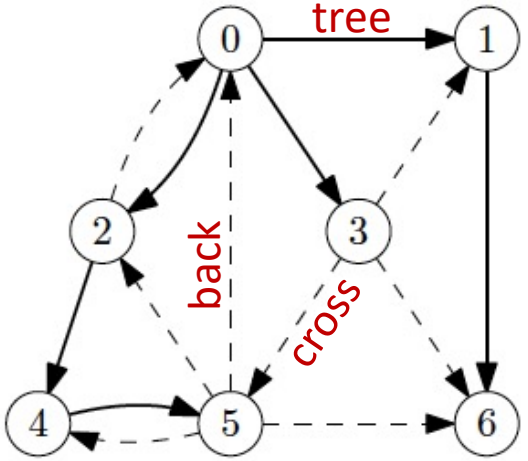
- If v is not an ancestor of w in F, then

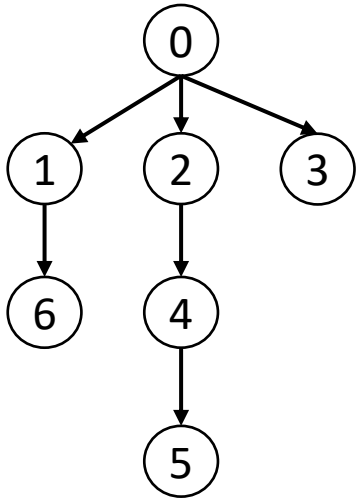
  $$seen[v] < done[v] < seen[w] < done[w] .$$

Example 23.2

Example 23.2. A digraph and its DFS search tree, rooted at node 0. The dashed arcs indicate the original arcs that are not part of the DFS search tree.



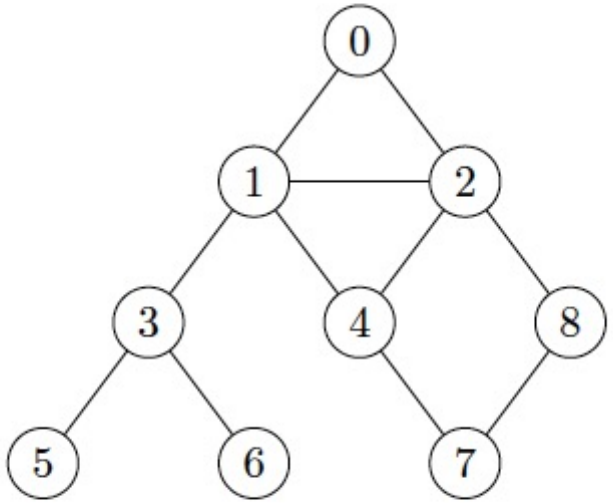


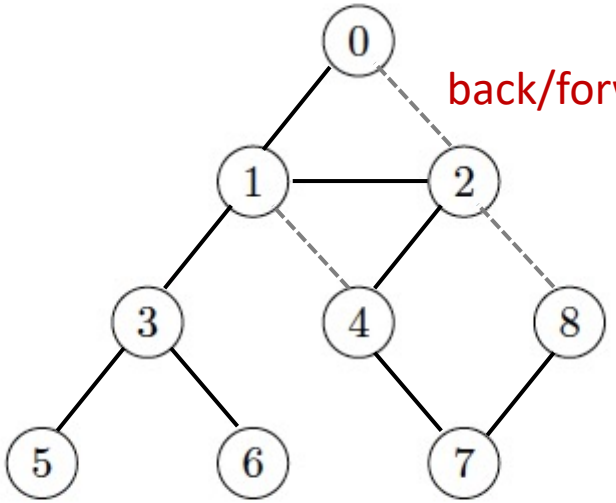


Find a tree arc, cross arc, forward arc and back arc (or say if that type of arc does not exist for that traversal).

Example 23.3

Example 23.3. Use the nodes on the right to draw the search tree you obtain by running DFS on the graph on the left, starting at vertex 0. Use dashed edges to indicate edges that are not arcs in the search tree.





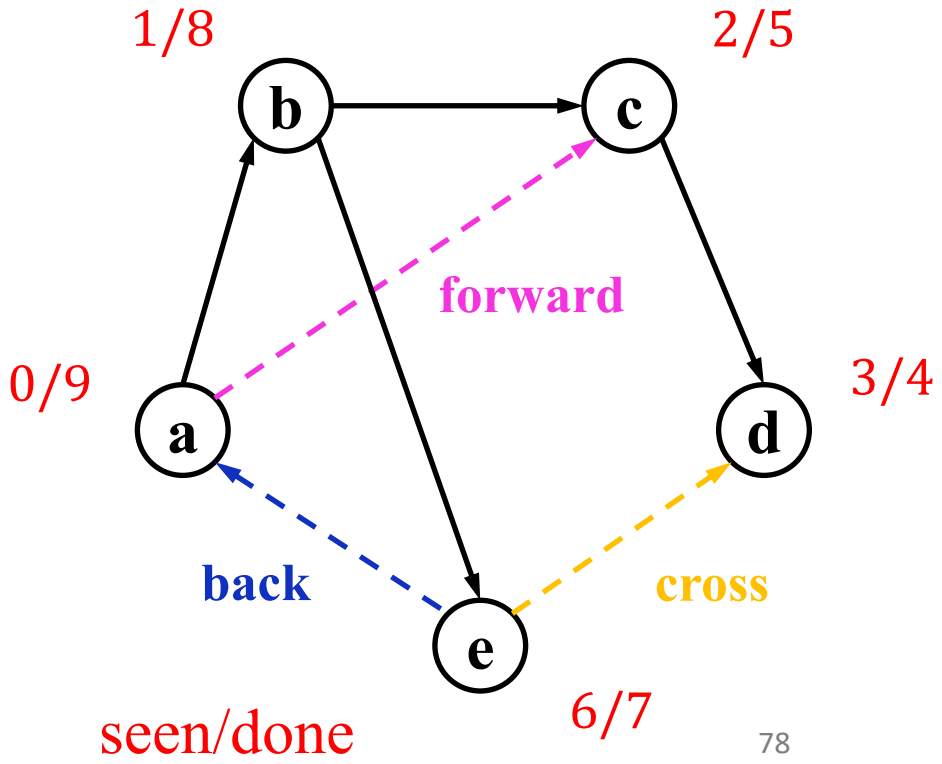
back/forward arcs

Find a tree arc, cross arc, forward arc and back arc (or say if that type of arc does not exist for that traversal). 8 tree arcs, 3 back/forward arcs, no cross arcs in graph

Example 23.7

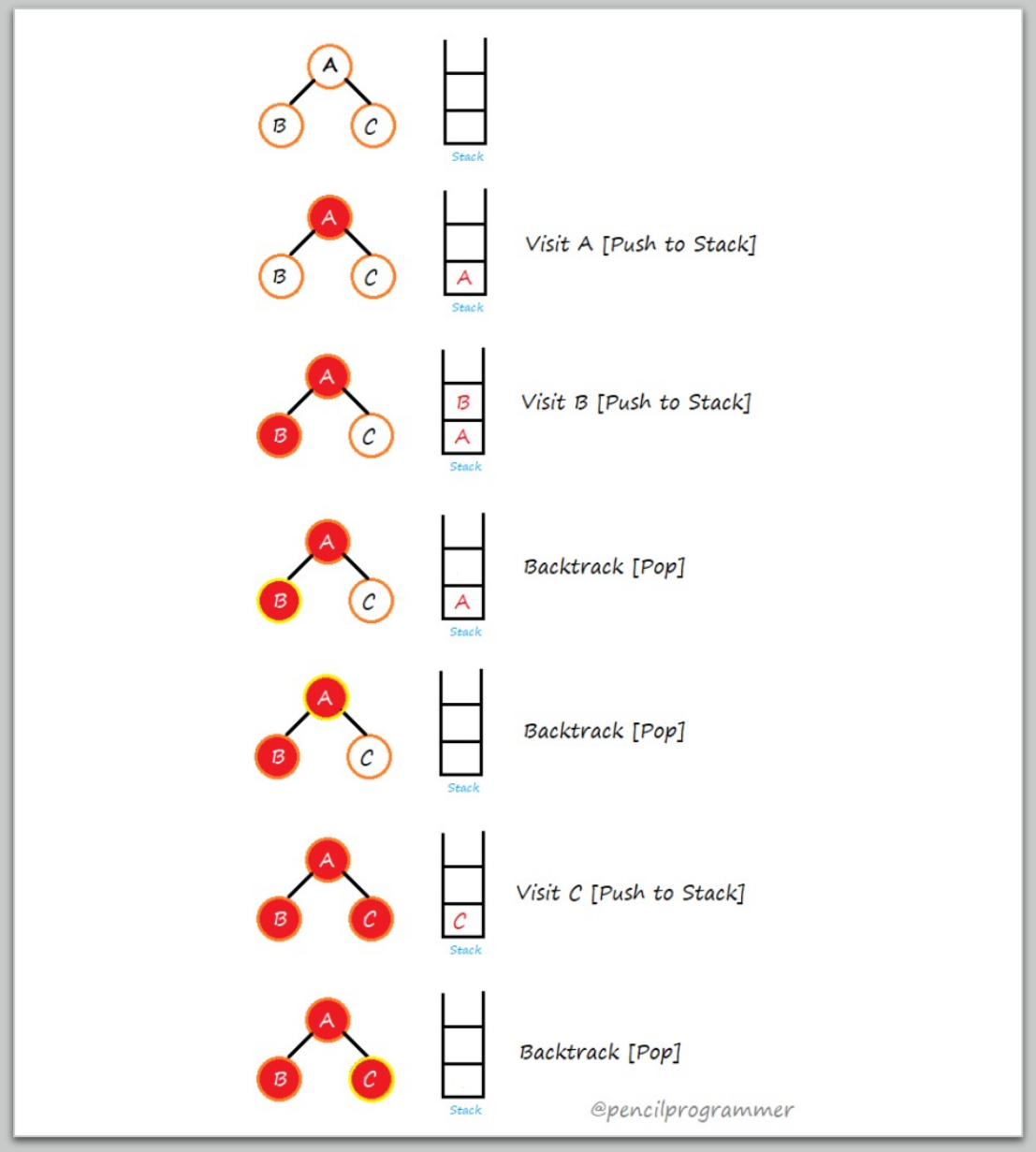
- Explain how to determine (u,v) in DFS algorithm whether it is a tree-, back-, forward- or cross-arc?

1. If v is white, then (u,v) is a **tree** arc
2. If v is grey, then (u,v) is a **back** arc
3. If v is black, then (u,v) is
   - a **cross** arc($seen[v]<seen[u]$, $done[v]<seen[u]$), or
   - a **forward** arc($seen[u]<seen[v]<done[v]<done[u]$).



SUMMARY

- Graph Traversal Algorithms
  - Depth-first Search (DFS)
  - Breadth-first Search (BFS)
  - Priority-first Search (PFS)
- Implementation
  - Stack – DFS
  - Queue – BFS
  - Priority Queues – PFS



Source: https://pencilprogrammer.com/algorithms/depth-first-search-dfs/