Graph Algorithm

(Chapter 4.2, 5.3)
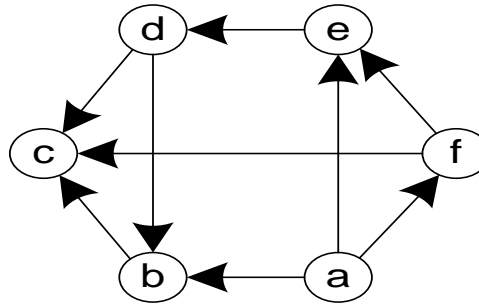
Graph Algorithm

- Topological Sorting
  - Using DFS
  - Decrease by one

- Binary Tree Traversal
  - Preorder
  - Inorder
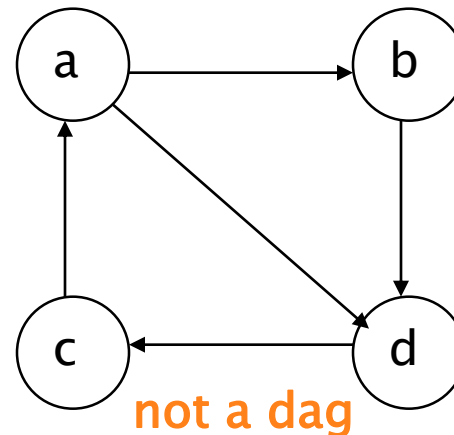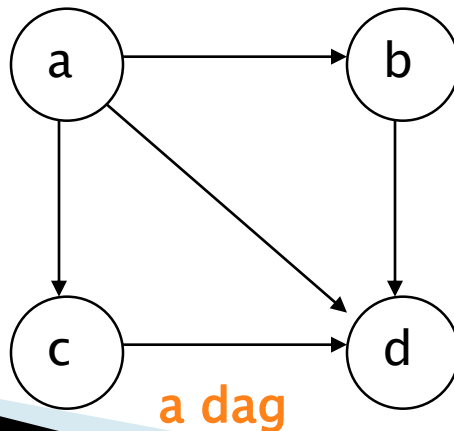 - Postorder

DAG's (Directed Acyclic Graphs)

- ▸ recall that a **directed graph** is a graph that uses arrows to show direction
  - ◦ for example:

- ▸ a **directed acyclic graph**, aka **DAG**, is a directed graph that contains no cycles

a dag

not a dag

Topological Sort

▸ **Problem:** We have a set of tasks and a set of dependencies (precedence constraints) of form "task A must be done before task B"

▸ **Goal:** Find a linear ordering that satisfies all dependencies

Topological Sort

◦ *Input might look like this ....*

2, 1 ← (task 2 must be done before task 1)
4, 3 ← (task 4 must be done before task 3)
1, 4 ← (task 1 must be done before task 4)
5, 2 ← (task 5 must be done before task 2)

one possible solution (topologically sorted order):

- 5 2 1 4 3 6

Topo Sort Algo 1:DFS

To obtain a topological sort order for a set of items:

1. represent the items as a directed graph G such that:
   a) vertices are the items that are tasks
   b) edges are the dependencies (constraints) between tasks
      - an edge from v to w (eg: v→w) means that v is dependent on w ... ie ... v must be done before w
2. apply the DFS algorithm to G
3. the order in which vertices become dead ends gives the reverse topological sort order

*Note: Topological Sort produces no solution if the graph contains a cycle*

Topo Sort Algo 1:DFS

Recall:

◦ the DFS implementation is recursive

◦ each time a recursive call is made is equivalent to "pushing a vertex on a stack"

◦ the "order in which vertices become dead ends" is given by the "order in which vertices are popped off the stack"
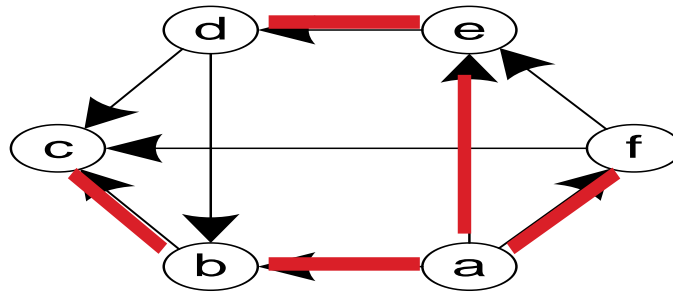
Example 1: Work Tasks

▸ Assume you have a set of 6 tasks (a, b, c, d, e, f) with the following dependencies:

   ◦ a must be done before b, e, f
   ◦ b must be done before c
   ◦ d must be done before b and c
   ◦ e must be done before d
   ◦ f must be done before c and e

▸ <u>Step 1:</u> Draw a directed graph to represent these dependencies.
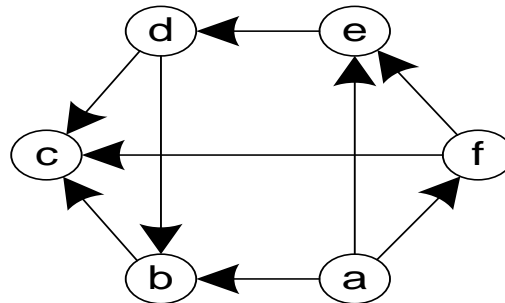
Example 1 (cont)

▸ <u>Step 2:</u> Apply DFS

Order vertices become dead ends:
c b d e f a

▸ <u>Step 3:</u>
Reverse this order for the solution:
a f e d b c

Example 2: Work Tasks

2 1 ← (2 before 1) 4 3 ← (4 before 3)
1 4 ← (1 before 4) 5 2 ← (5 before 2)
2 3 ← (2 before 3) 5 1 ← (5 before 1)
5 6 ← (5 before 6) 6 3 ← (6 before 3)
2 4 ← (2 before 4) 6 2 ← (6 before 2)

▸ Step 1: draw the graph (and verify it is a DAG)
▸ Step 2: apply DFS
▸ Step 3: find the order vertices were removed from stack, and reverse this order to get topological sort order

Topo Sort Algo 2: Decrease by One

▸ Observe:
  ◦ if a vertex v in the dependency graph G has no incoming arrows (ie: in-degree(v) == 0), then v does not have any dependencies
  ◦ it follows that any v that does not have dependencies is a candidate to be visited next in topographical order

▸ A Decrease-by-One approach:
  ◦ identify a v ∈ V that has in-degree = 0
  ◦ delete v and all of its edges
  ◦ when all vertices have been deleted, the topo sort order is given by the order of deletion
  ◦ if there are v ∈ V, but no v has in-degree = 0, the graph G is not a DAG (no feasible solution exists)

Example
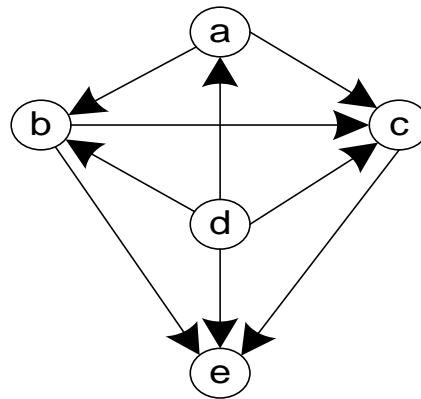
Topo Sort Algo 2: Decrease by One

- More detailed algorithm:
  - need a set to store the candidate v's (in-degree = 0)
    - I will use a TreeSet. Any ordered set will do.
  - need an ordered list to store the delete order
    - I will use an ArrayList. Any ordered list will do.

- Then the algorithm is:

Topo Sort Algo 2: Decrease by One

```
topo(G)
   create an empty ArrayList Soln
   create an empty TreeSet Candidates
   add all v with inDegree=0 to Candidates
   while Candidates is not empty
      v ← Candidates.first()
      add v to Soln
      for each vertex w adjacent to v
         remove edge (v,w) from G
         if w has inDegree=0
            add w to Candidates
      remove vertex v from G
   if there are vertices remaining in G
      no feasible solution exists
   else
      solution is in Soln
```

Graph Algorithm

▸ Topological Sorting
  ◦ Using DFS
  ◦ Decrease by one

▸ Binary Tree Traversal
  ◦ Preorder
  ◦ Inorder
  ◦ Postorder

Binary Tree
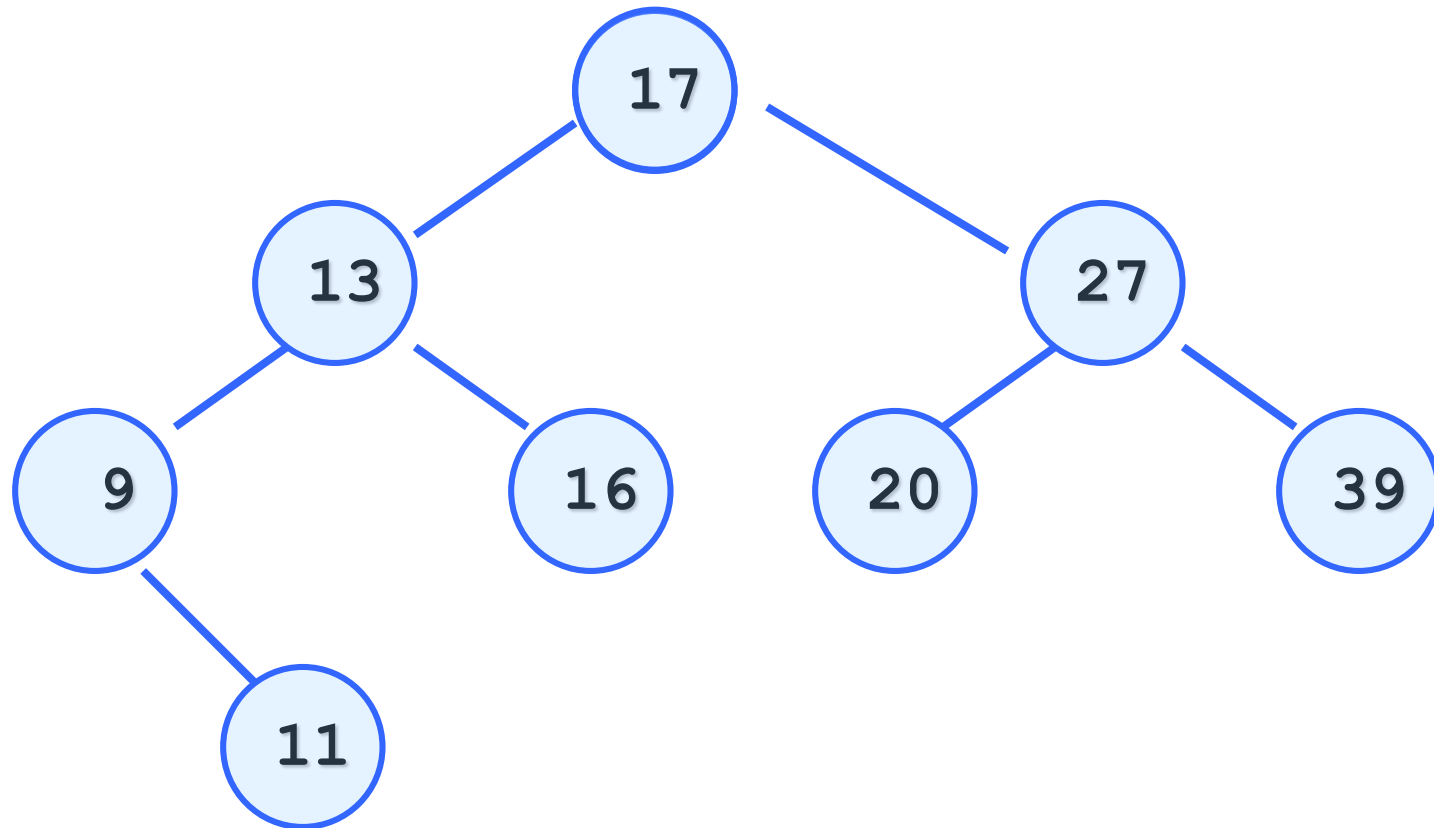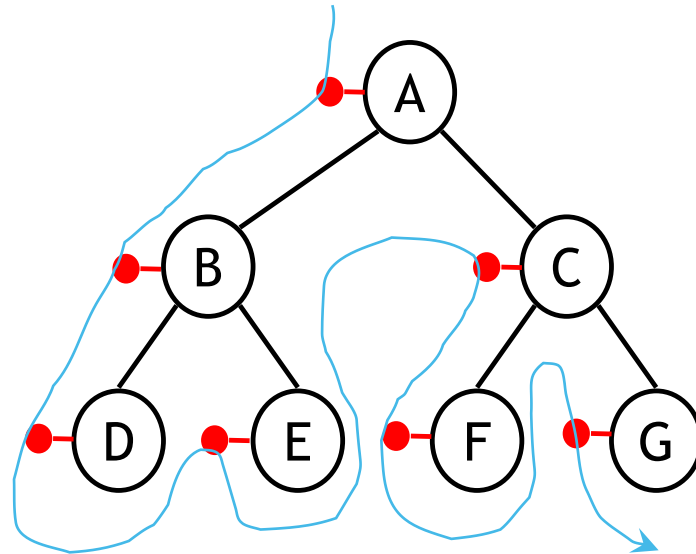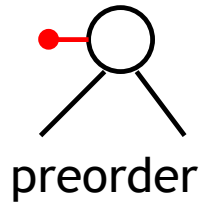
Preorder

a b c

Preorder

```java
public void preorderPrint(Node N) {

    if (N == null) return;
    System.out.println(N.value);
    preorderPrint(N.leftChild);
    preorderPrint(N.rightChild);
}
```
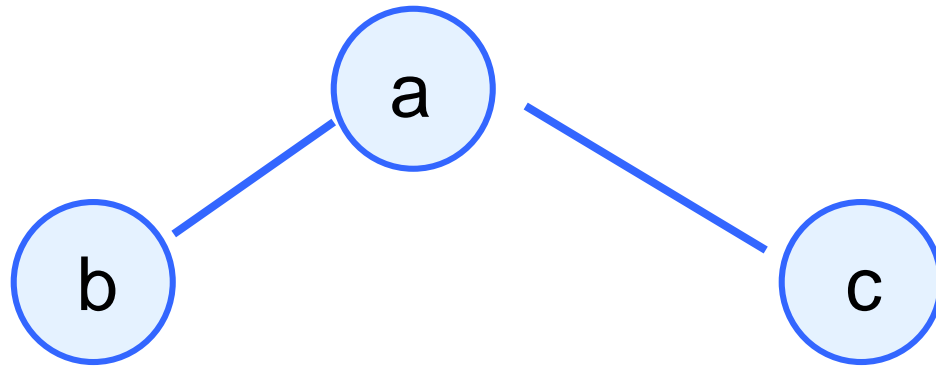
Example

Preorder: A B D E C F G
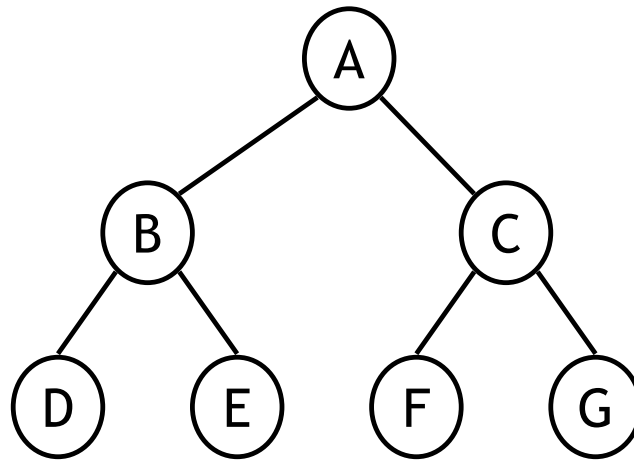
Example

preorder

Preorder: A B D E C F G
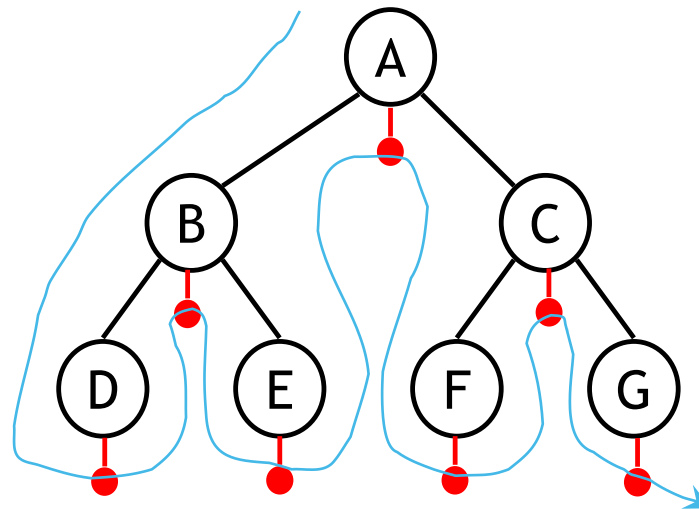
Inorder
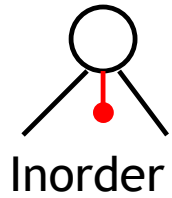

b a c

Inorder

```java
public void preorderPrint(Node N) {

        if (N == null) return;
        preorderPrint(N.leftChild);
        System.out.println(N.value);
        preorderPrint(N.rightChild);
}
```
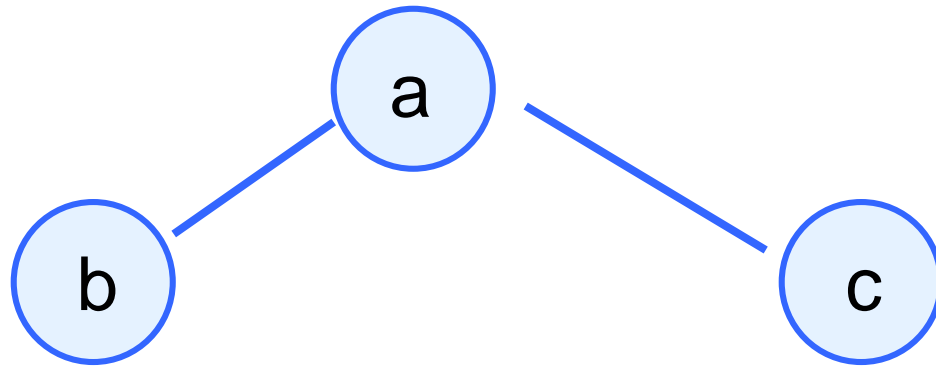
Example

Inorder: D B E A F C G

Example

Inorder

Inorder: D B E A F C G

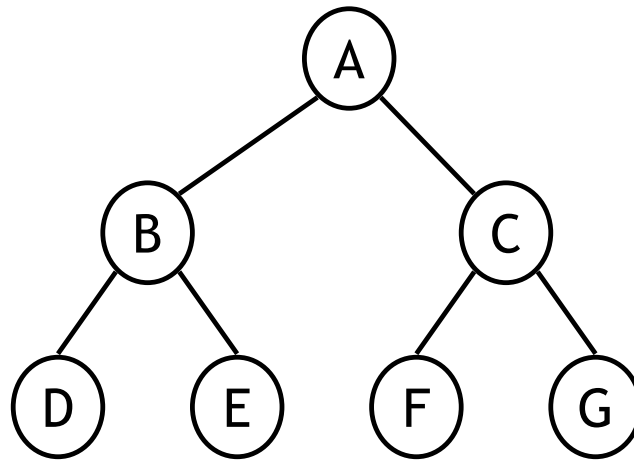# Postorder

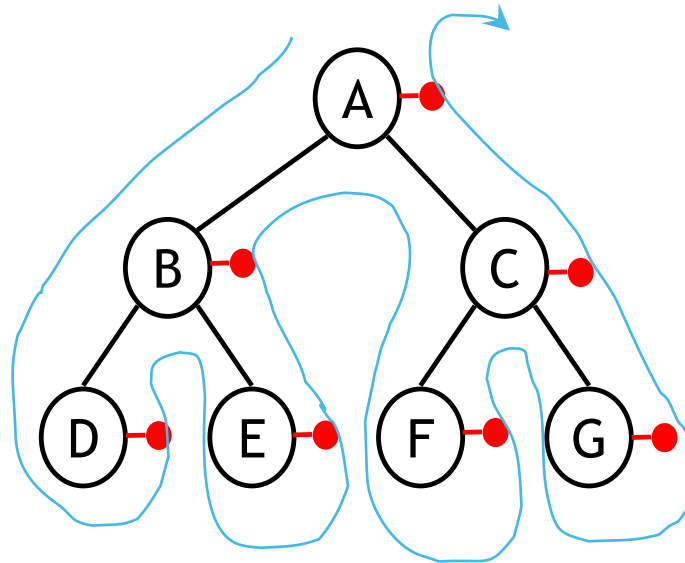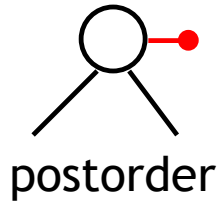

b c a

postorder

```java
public void preorderPrint(Node N) {

        if (N == null) return;
        preorderPrint(N.leftChild);
        preorderPrint(N.rightChild);
        System.out.println(N.value);
}
```
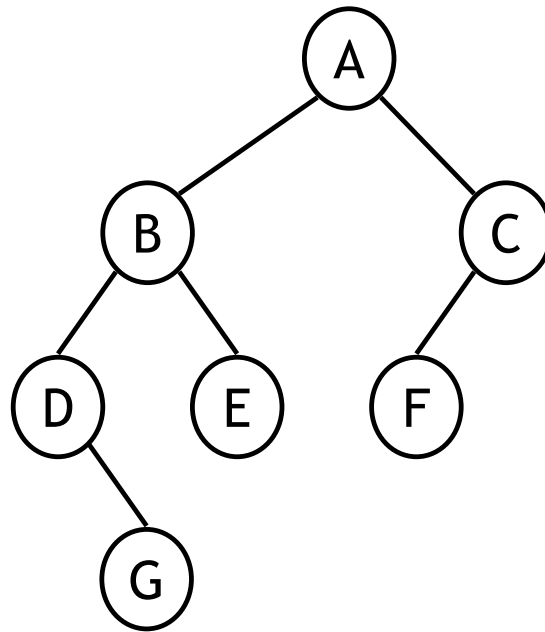
Example

postorder: D E B F G C A

Example

postorder

Postorder: D E B F G C A

Example

Try it/ homework

1. Chapter 4.2, page 142, question 1
2. Chapter 5.3, page 185, questions 5,6