Graph Algorithms: Digraphs

Unit 4: Lecture 04

Traversal of Directed Graphs

Traversal of a digraph is **the process of visiting each vertex** in some order. Usually a graph is traversed in the order of **depth** (called depth first search) **or breadth** (called breadth first search)

Depth First Search (DFS):

- Let G be a digraph with vertices marked as unvisited (**initially**).
- DFS works by **selecting one vertex** (say v) as start vertex and mark it as visited.
- Each unvisited **vertex adjacent** to v is searched in a recursive manner producing a sequence of vertices.
- Once all vertices reachable from v are visited, the search of v **is complete**.
- If some vertices remain unvisited, **any of them is selected** as new start and repeated the process.

DFS

For given digraph the following DFS sequences (few) are obtained:

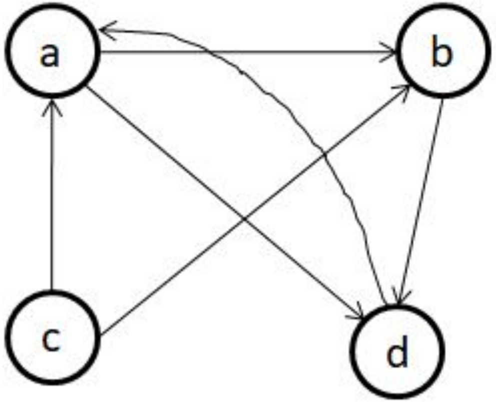
a b d c

a d b c

b d a c

c a b d

c b d a



$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

DFS (contd.)

The **pseudocode** for DFS:

```
procedure dfs(vertex v)
    mark[v]=visited
    for each vertex w on L[v]    //L[v] is the adjacency list for v
        if mark[w]==unvisited
            dfs(w)
end procedure
```

As DFS visits all the vertices, it would take **O(n+e) time** to visit the graph where $n \le e$ (n: no. of vertices, e: no. of arcs)

DFS (contd.)

The process is called DFS because it continues **searching in forward** (deeper) direction as long as possible.

DFS is a generalization of **preorder traversal** of a tree.

During DFS, certain arcs lead to unvisited vertices; the arcs leading to new vertices are called **tree arcs** which form a depth first **spanning forest**.

An arc that goes from a vertex **to one of its ancestors** is called **back arc**: an arc from a vertex **to itself** is also called a back arc.

A non-tree arc that goes from a vertex to a proper descendant is called a **forward arc**.

An arc that goes from a vertex to another that is neither ancestor nor a descendant is called a **cross arc**.

DFS (contd.)

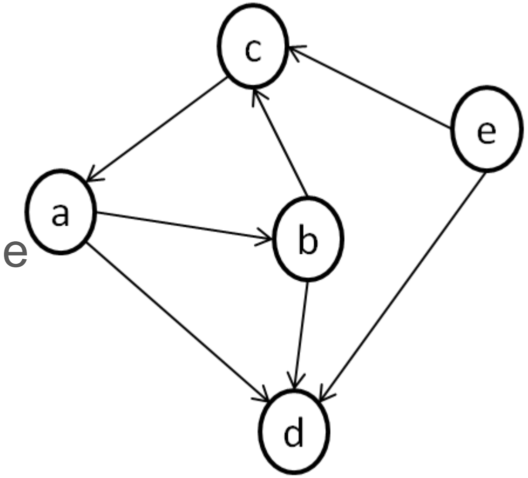
Example: Spanning forest- the vertices (a,b,c,d) form a DFS tree whereas the vertex (e) forms another

Tree arcs: solid lines

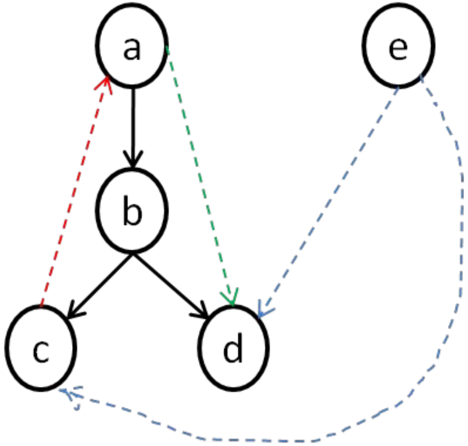
Back arc: dotted red line

Forward arc: dotted green line

Cross arcs: dotted blue lines



A digraph



DFS: Spanning Forest

DFS (contd.)

How to distinguish among various types of arcs?

The **vertices are numbered in order** in which they are marked visited during DFS, i.e., by the following modified dfs procedure

```
procedure dfs(vertex v)
    mark[v]=visited
    dfsno[v]=count          //dfs number of the digraph
    count=count+1
    for each vertex w on L[v]
        if mark[w]==unvisited
            dfs(w)
end procedure
```

DFS (contd.)

The array dfsno[] represents the **depth first search number** of the digraph.

All descendants of a vertex v are assigned dfsno greater than the number assigned to v. Thus, a vertex w is a descendant of v iff

$$\text{dfsno}(v) \leq \text{dfsno}(w) \leq \text{dfsno}(v)+\text{number of descendants of } v$$

Forward arcs go from low numbered to high numbered vertices

Back arcs go from high numbered to low numbered vertices

Directed Acyclic Graphs (DAG)

A DAG is a **digraph with no cycles**.

DAGs are **more general** than trees as compared to arbitrary directed graphs in terms of “the relation they represent”.

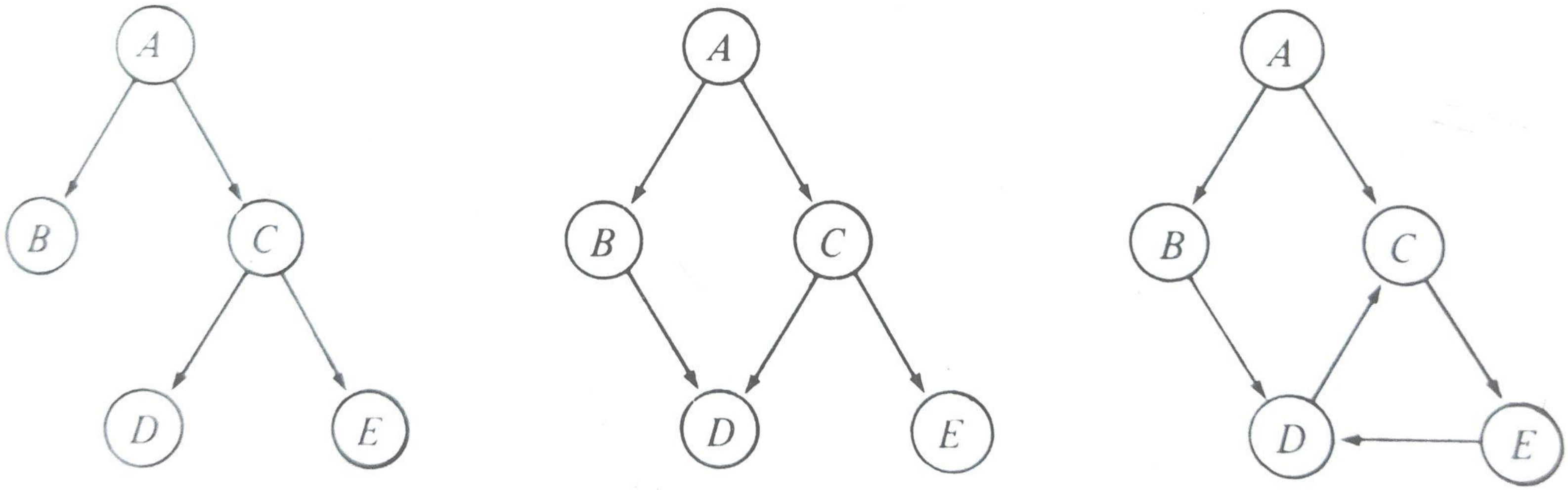
To determine whether a **digraph is a DAG**, DFS can be used (how?). The problem is stated as: Let G=(V,E) be a given digraph, determine **if G is acyclic**.

Applications of DAGs:

- **Expression trees** (with common subexpressions) representation
- Representation of **partial orders**

DAG (contd.)

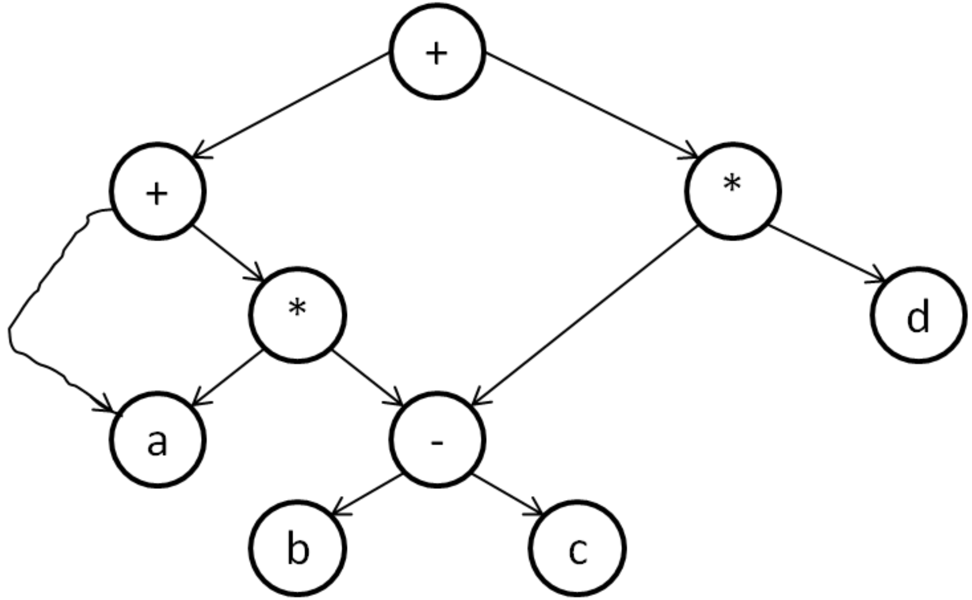
Example: (a) and (b) are DAGs whereas (c) is not (contains a cycle)



(a) (b) (c)

DAG (contd.): Expression Tree

Example: a + a * (b - c) + (b - c) * d



DAG (contd.)

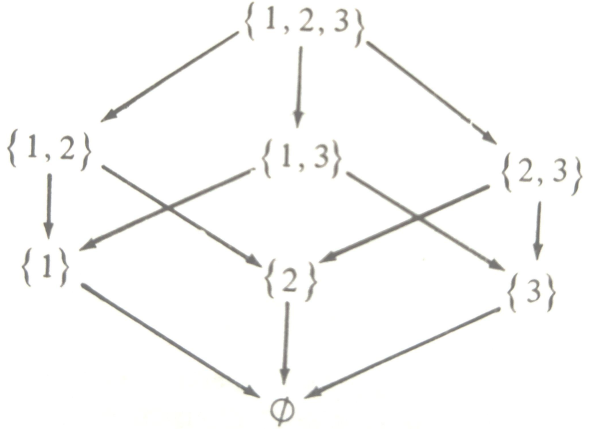
Topological Sort: a process of assigning a linear ordering to the vertices of DAG so that if there is an arc from i to j then i appears before j **in linear ordering**.

Topological sort can be **achieved by using DFS**: print the vertex after visit

```
procedure topologicalsort(vertex v)
    mark[v]=visited
    for each vertex w on L[v]
        if mark[w]==unvisited
            topologicalsort(w)
        print(v)                    //only change in DFS
end procedure
```

DAG (contd.)

Example: DAG for **partial order relation** R defined as ⊆ on the set power set of the set S={1,2,3}



Example: Topological sort of the given digraph:

{1,2,3}, {1,2}, {1,3}, {2,3}, {1}, {2}, {3}, Φ

{1,2,3}, {1,2}, {2,3}, {1,3}, {1}, {3}, {2}, Φ

{1,2,3}, {1,2}, {1,3}, {1}, {2,3}, {2}, {3}, Φ etc.

The last sequence shows that topological sort is **different from level order** traversal.

Strong Components

A **maximal set** of vertices in which there is **a path from any one vertex to any other** vertex is called a **strongly connected component**.

DFS can be used to determine strongly connected components.

Let G=(V,E) be a digraph. **V can be partitioned** into equivalence classes V_i, $1 \le i \le r$ s.t. the vertices v and w are equivalent iff there is a path from **v to w and w to v**.

Let E_i, $1 \le i \le r$ be the set of arcs with head and tail in V_i then the graphs $G_i=(V_i,E_i)$ are called the strongly connected components or **strong components** of G.

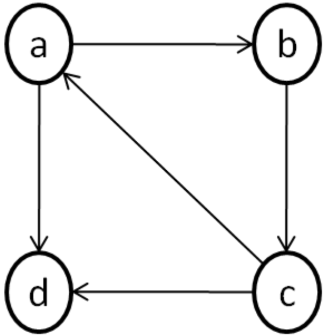
A digraph with **only one strong component** is called **strongly connected**.

Strong Components (contd.)

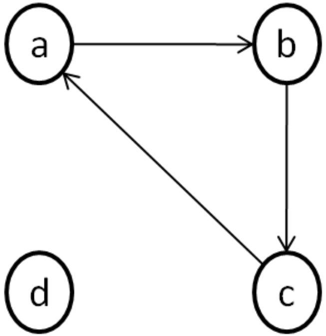
Example:

There are 2 strong components

of the digraph:



A Digraph



Strong components

Note: **Every vertex of a digraph is in some strong component** but that certain arcs may not be in any component

Strong Components (contd.)

Algorithm:

- Perform DFS and assign a number to the vertices in order of access
- Construct a new digraph Gr by reversing the directions
- Perform DFS on Gr starting the search from the highest numbered vertex. If DFS doesn’t reach all vertices, start DFS with next highest numbered vertex
- Each tree in the resulting spanning forest is a strongly connected component

The vertices of a strongly connected component correspond to the vertices of a tree in the spanning forest of the second DFS

Exercise

1. Devise a procedure to apply DFS on an adjacency matrix.
2. Write an algorithm to find the tree arcs in a digraph.
3. If a back arc is encountered during DFS, the graph is not acyclic. Use this fact to test if a given graph is a DAG.

Reference Book

Data Structures and Algorithms: Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, 10th Impression, Pearson Education, New Delhi