CS583 Lecture 05
Basic Graph Algorithms

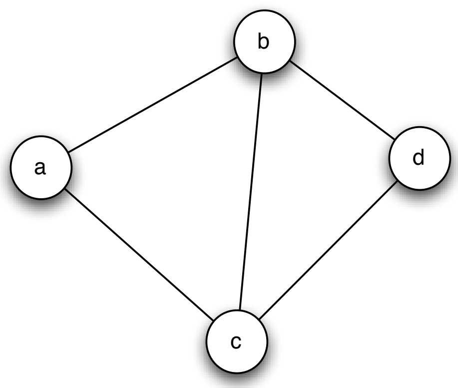
Jyh-Ming Lien

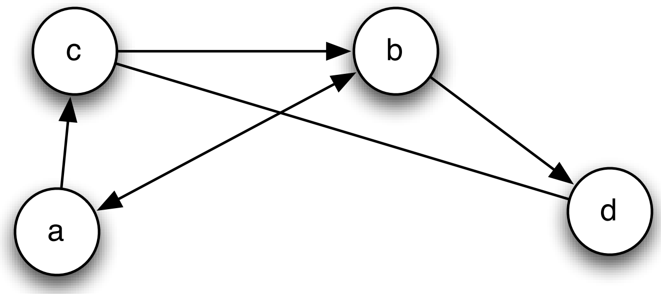
some materials here are based on Prof. Shehu, and Prof. Wang's past lecture notes

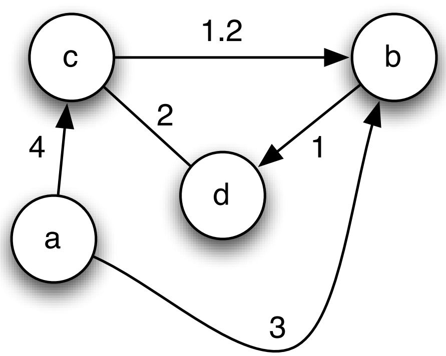
Graph Representation

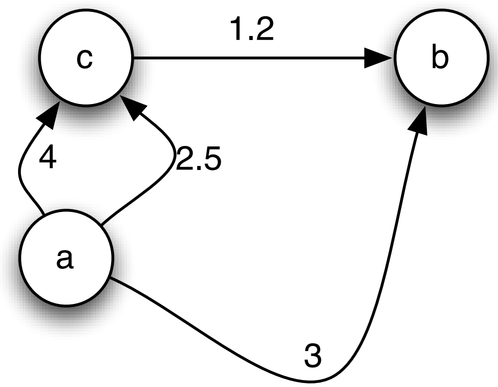
- What is a graph
- What can we do with a graph?

Graph Types

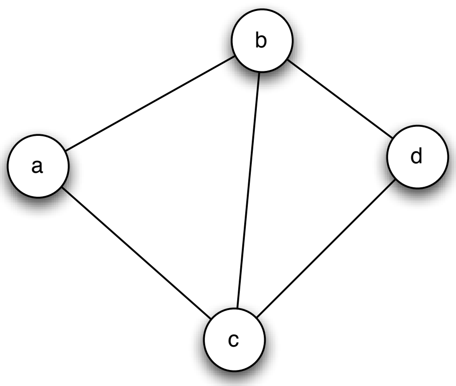


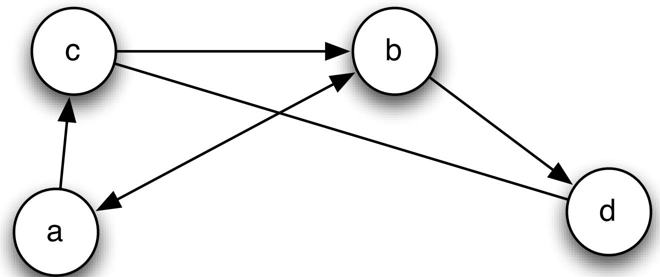


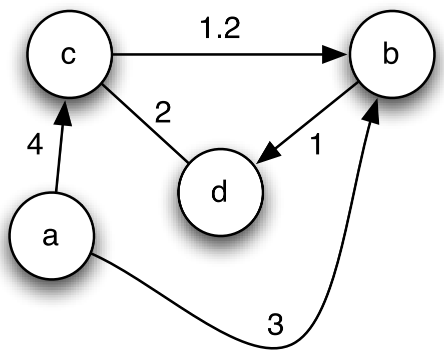


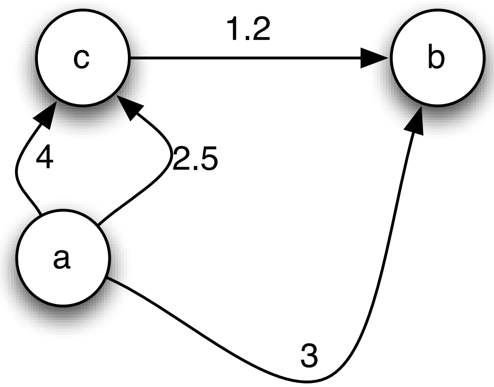


Graph Representation









Graph Representation

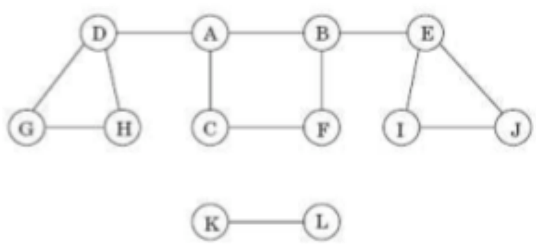
- Representations
  - Adjacency matrix
  - Adjacency list (a list of vertices and each vertex has a list of edges)
- Basic Operations
  - add/delete vertices/edges
  - count edges/vertices/degree
  - check the existence of a vertex or an edge
- Represent each of the previous graphs in both format and discuss the time complexity of each operation

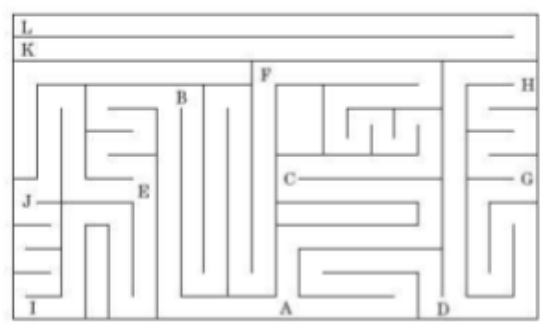
Graph Representation

- Advanced operations (a very short list....)
  - check if two vertices are connected
  - compute # of connect components
  - find one or many (shortest/longest) path(s) between two or all pairs of vertices
  - find a spanning tree
  - compare two graphs (isomorphism)
  - cut graphs (graph partitioning)
  - linearize a graph
  - cluster vertices/edges
  - visualize a graph

Explore Graph

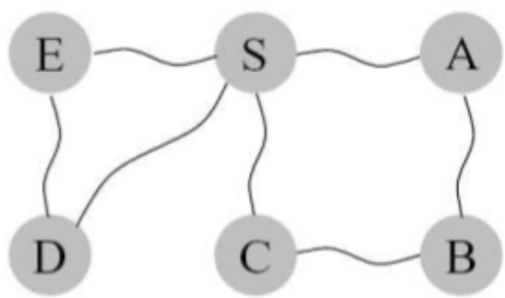
- Many of the aforementioned problems depends on a systematical way of exploring a graph
- Two basic tools to safely explore an unknown environment
  -
  -

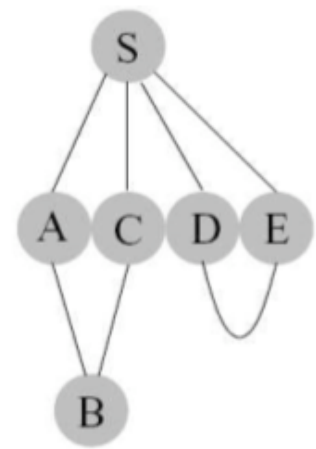




Breath First Search

- Basic idea: sort graph vertices level by level





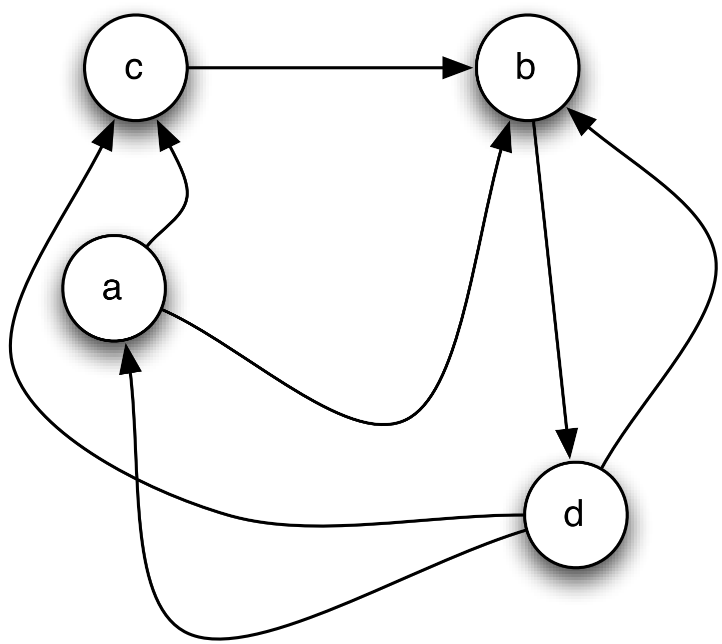
BFS

- Algorithm

- Time complexity:

BFS

- Examples



BFS

- Properties:
  - creates a spanning tree
  - optimal way for finding the short path for unweighted graph
    - proof:

Depth First Search

- Basic idea: deepening as much as possible before coming back

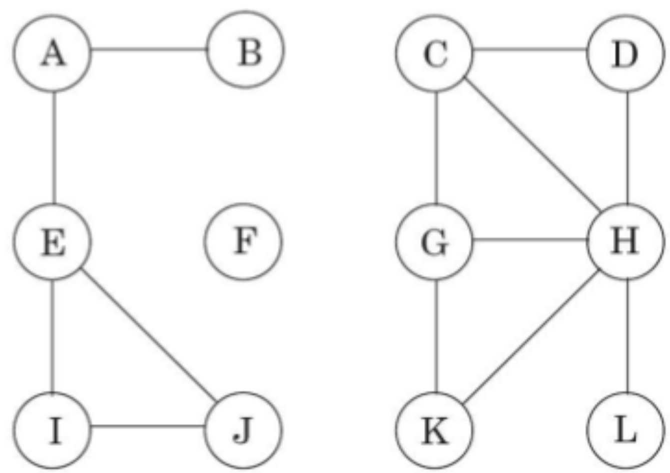
DFS

- Algorithms

- Time complexity

DFS

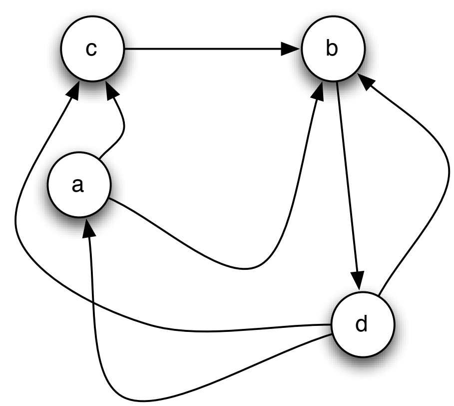
- Examples



Problems

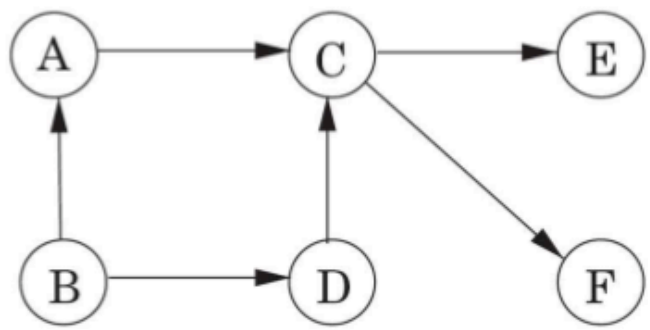
- Examples

- Given an undirected graph G, report the number of connect components (CC) in G
- Given two vertices u and v, check if u and v are from the same (CC)
- Given a tree T, can you preprocess T so that you can answer if u is the ancestor of v for a give pair of nodes u and v.
- Identify types (tree, back, forward, and cross) of edges in a directed graph



Topological Sort

- A graph G without (directed) cycle is a *directed acyclic graphs* (DAG)
- DAG can be found in modeling many problems that involve prerequisite constraints (construction projects, document version control)

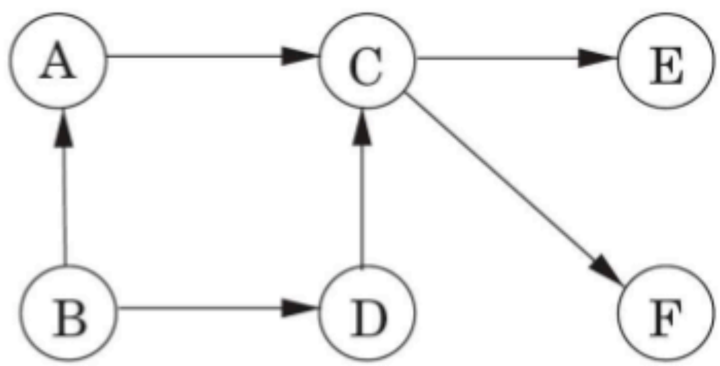


- Given a *directed* graph G, identify *cycles* in G

  – *proof*

Topological Sort

- **Topological sorting** or **Linearization**: Vertices of a DAG can be linearly ordered so that:
  - Every edge its starting vertex is listed before its ending vertex
  - Being a DAG is also a necessary condition for topological sorting be possible
- Example:



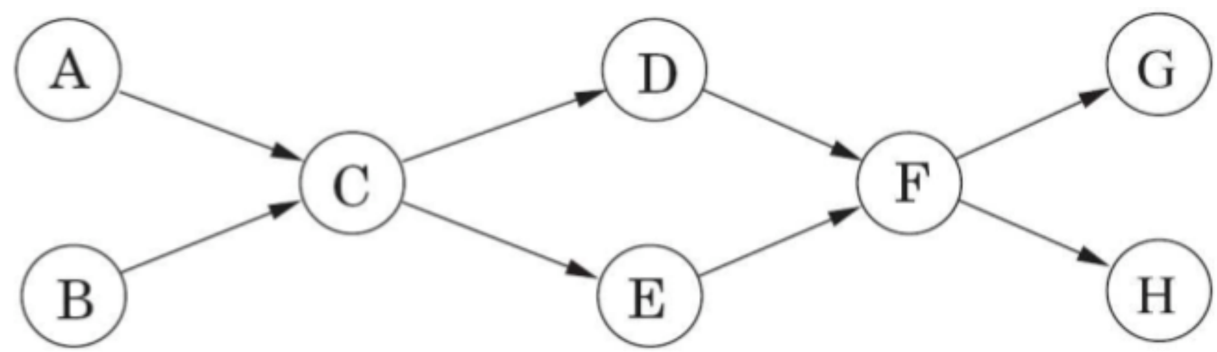
Topological Sort

- Algorithms

- Time complexity

Topological Sort

- example



Topological Sort

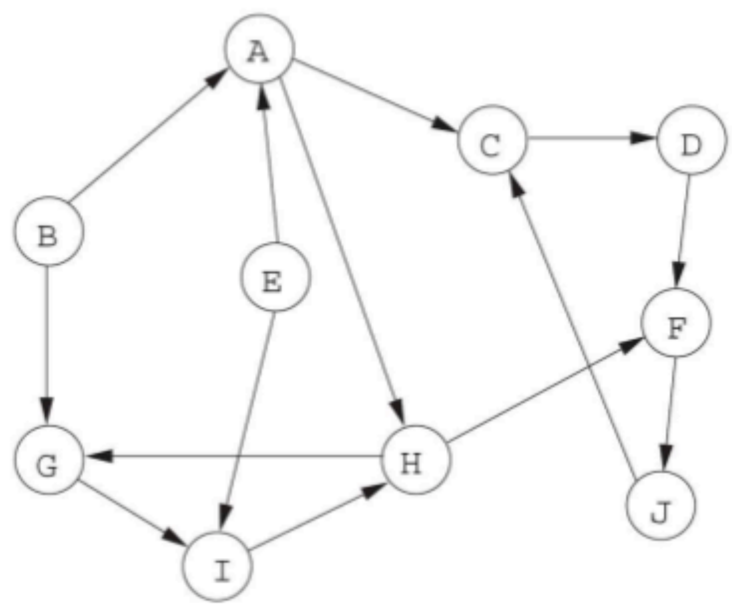
- Why does it work?

- Another method:

Strongly Connected Components

- **Definition**: Two nodes u and v are from the connected if and only if there is a path from u to v and a path from v to u.

- **Definition**: A set of vertices form a strongly connected component (SCC) iff any pairs of vertices are connected.



SCC Observations

1. SCC and DAG
    - converting each SCC to a node
    - the resulting meta-graph is a dag!
2. Any node in a SCC in the sink of the dag can only reach nodes in the same SCC
3. Finding a node in a source SCC is easier than finding a node in a sink SCC
    - Note that we DO NOT know how the dag looks like!

SCC Observations

- A node in a source SCC can be found by the largest post visit number
  - Proof

- A reversed graph G' of a graph G, G and G' have the same # of SCCs
  - Proof

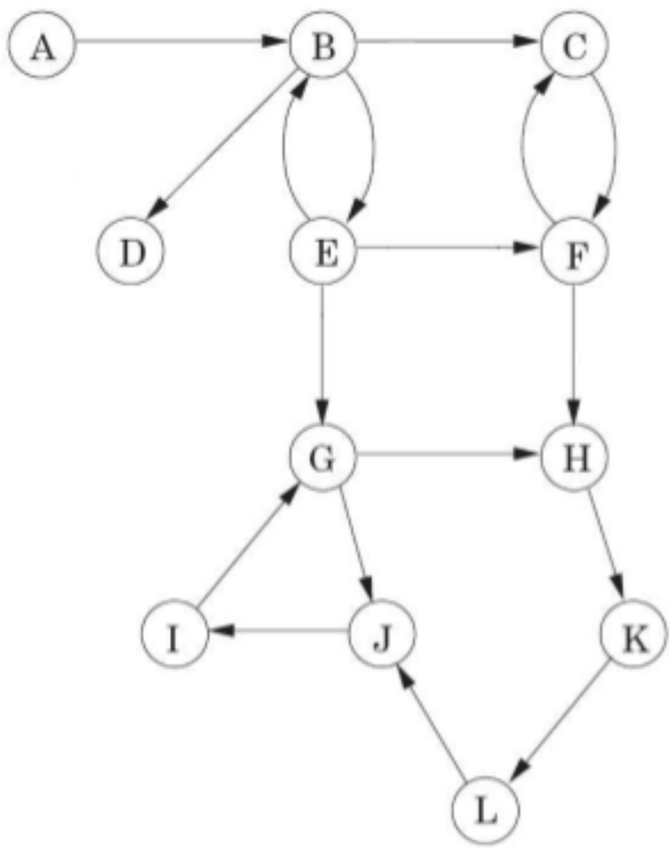
SCC

- Algorithms

- Time complexity

SCC

- Examples:



SCC

- Why do we need to find a SCC?