Lecture 38: Graphs

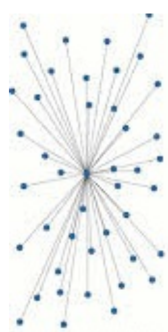
Today

- Reading
  - Weiss Chapter 16
- Objectives
  - Graphs

+ Announcements

- No more quizzes!
- Assignment 11/12 is due this Tuesday by 11:59pm
- Assignment 13 is already posted!
  - The final assignment
  - Can work in pairs
  - Due Monday May 5th
  - Talk about it more on Wednesday

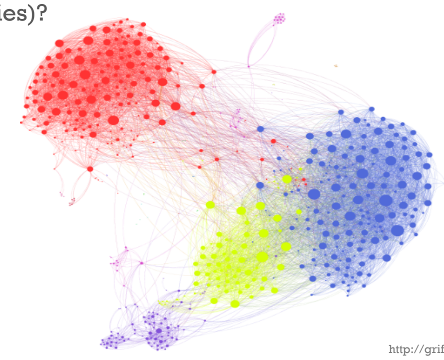
+ Graphs in Real Life



- Transportation networks
  - Airline flight paths
  - Roads, interstates, etc. (Google maps)
  - Finding shortest route, cheapest route
  - Find route that minimizes fuel costs
- Communication networks
  - Electrical grid, phone networks, computer networks
  - Minimize cost for building infrastructure
  - Minimize losses, route packets faster

More Graphs

- Social networks
  - People and relationships (e.g. Facebook)
  - Does this person know this person?
  - Can this person introduce me to that person (e.g. job opportunities)?



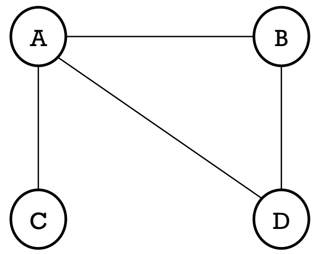
http://griffsgraphs.com/2012/07/02/a-facebook-network/

Definitions

- A graph is a generalization of a tree
- A graph G is a set (V,E)
  - V is a finite, non-empty, set of vertices
  - E is the set of edges that connect pairs of vertices
  - Called "vertices" or "nodes"

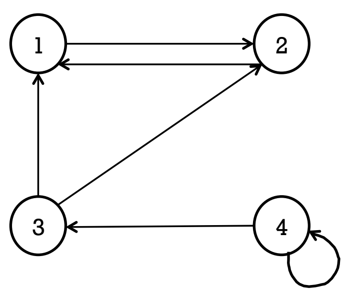
Example: Undirected Graph

- G = (V,E) where
- V = {A, B, C, D}
- E = {(A, C), (A,B), (A,D), (B,D)}

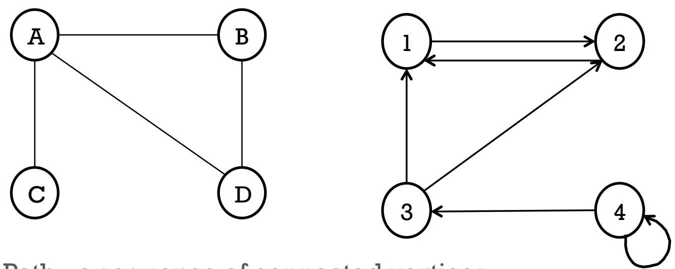


Example: Directed Graph

- G = (V,E) where
- V = {1, 2, 3, 4}
- E = {(1,2), (2,1), (3,1), (4,3), (4,4), (3,2)}



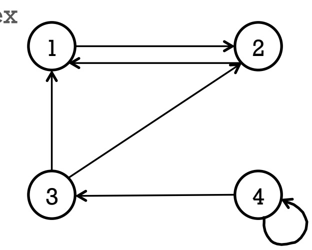
+ Definitions



- Path - a sequence of connected vertices
  - A simple path – a path where all vertices occur only once
- Path length - the number of edges in the path
- A cycle - a path of length ≥ 1 that begins and ends at the same vertex
- Simple cycle – a simple path that begins and ends at the same vertex

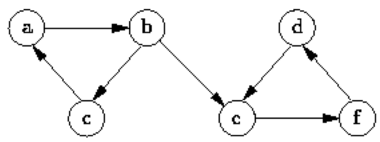
+ Definitions

- self loop – A cycle consisting of one edge and one vertex
- incident – Edge (x,y) is incident on vertex x and y
- adjacent – Vertices x and y are adjacent if they are connected by an edge (x,y)
- degree – number of incident edges for a vertex
- simple graph – a graph with no self loops
- acyclic graph – a graph with no cycles

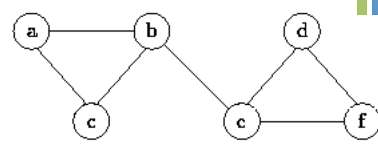


Connected Components

weakly connected



connected

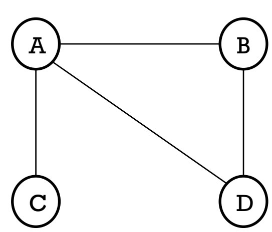


- (undirected) connected graph– Every pair of vertices is connected by a path
- (directed) weakly connected – A directed graph that would be connected if all its directed edges were replaced with undirected
- (directed) strongly connected –Every pair (u,v) there is a path from u to v and from v to u

Adjacency Matrix

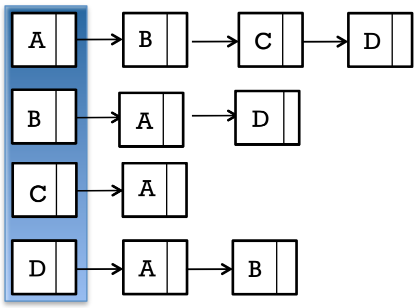
- Store a |V|-by-|V| boolean matrix (two-dimensional array)
  - Entry (i,j) is 1 if there is an edge from vertex i to vertex j
  - Symmetric if undirected
  - Space? Time to lookup edge?

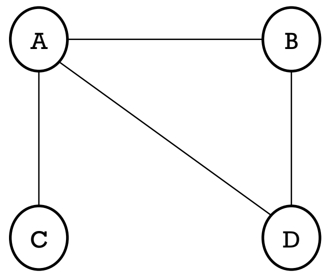
	A	B	C	D
A	0	1	1	!
B	1	0	0	1
C	1	0	0	0
D	1	1	0	0



+ Adjacency List

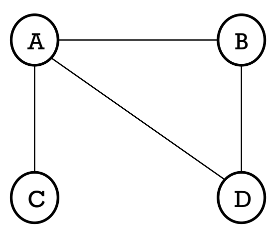
- Store a list of linked lists
  - Use map from vertex labels to lists
  - Space? Time to lookup edge?





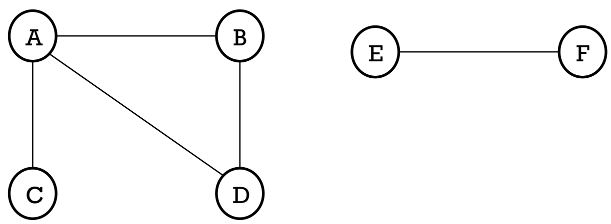
+ Breadth-first Search

- Equivalent to a level-order traversal of a tree
  - Search all nodes 1 away, 2 away, 3 away, etc
- Uses a queue data structure
- Basic algorithm:
  - Enqueue the start node
  - While the queue is not empty:
    - Dequeue a node
    - Check if node previously visited
    - If not, mark as visited and enqueue all children



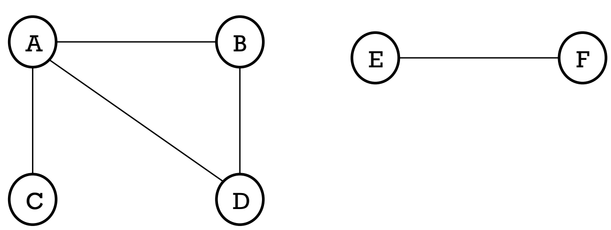
Breadth-first Search

- If graph has multiple connected components
  - Wrap BFS inside a for-loop that iterates through all nodes
- See *bfs_dfs_demo.cpp*
  - Uses a `typedef` (allows you to rename a type)
  - Better to use `map<string, vector<string>>` instead of `pair`



Depth-first search

- Equivalent to a pre-order traversal of a tree
  - except may get stuck in cycles
- Can use the same algorithm as BFS
  - Either use a stack or use recursion



+

Detecting Cycles

- Can use depth-first search to see if we loop back
- How can we detect a loop?
  - A node in our adjacency list has already been visited but it is not the node that added us (we call this node our parent)
  - Works for an undirected graph