Elementary Graph Algorithms

Graphs

- Motivation and Terminology
- Representations
- Traversals

Graphs

- A graph G = (V, E)
  - V = set of vertices
  - E = set of edges = subset of $V \times V$
  - Thus $|E| = O(|V|^2)$

Graph Variations

- Variations:
  - A *connected graph* has a path from every vertex to every other
  - In an *undirected graph:*
    - Edge (u,v) = edge (v,u)
    - No self-loops
  - In a *directed* graph:
    - Edge (u,v) goes from vertex u to vertex v.
    - It is denoted $u \rightarrow v$

Graph Variations

- More variations:
  - A *weighted graph* associates weights with either the edges or the vertices
    - E.g., a road map: edges might be weighted as distance
  - A *multigraph* allows multiple edges between the same vertices
    - E.g., the call graph in a program (a function can get called from multiple points in another function)

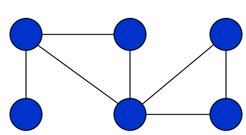
Graphs

A graph *G* consists of a set of *vertices V* together with a set *E* of vertex pairs or *edges*.

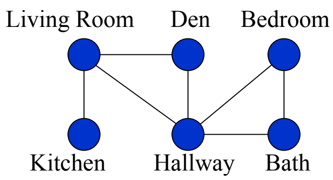
G = (V,E)   [in some texts they use G(V,E)].

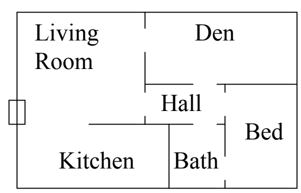
We also use V and E to represent # of nodes, edges

Graphs are important because any binary relation is a graph, so graphs can be used to represent essentially *any* relationship.



Graph Interpretations





Apartment Blueprint

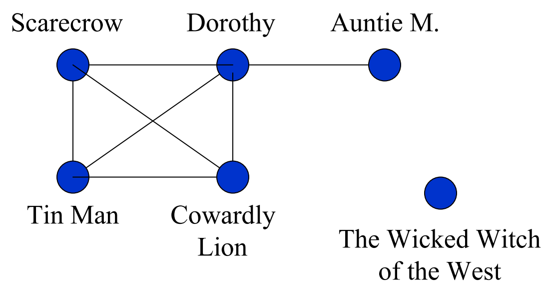
The vertices could represent rooms in a house, and the edges could indicate which of those rooms are connected to each other.

Sometimes a using a graph will be an easy simplification for a problem.

More interpretations

- Vertices are cities and edges are the roads connecting them.
- Edges are the components in a circuit and vertices are junctions where they connect.
- Vertices are software packages and edges indicate those that can interact.
- Edges are phone conversations and vertices are the households being connected.

Friendship Graphs



Each vertex represents a person, and each edge indicates that the two people are friends.

Graph Terminology

Directed and undirected graphs

A graph is said to be ***undirected*** if edge (x, y) always implies (y, x). Otherwise it is said to be ***directed***. Often called an ***arc***.

Loops, multiedges, and simple graphs

An edge of the form (x, x) is said to be a ***loop***. If x was y's friend several times over, we can model this relationship using ***multiedges***. A graph is said to be ***simple*** if it contains no loops or multiedges.

Weighted edges

A graph is said to be ***weighted*** if each edge has an associated numerical attribute. In an ***unweighted*** graph, all edges are assumed to be of equal weight.

Graph Terminology - contd.

Paths

A ***path*** is a any sequence of edges that connect two vertices. A ***simple path*** never goes through any vertex more than once. The ***shortest path*** is the minimum number edges needed to connect two vertices.

Connectivity

The "six degrees of separation" theory argues that there is always a short path between any two people in the world. A graph is ***connected*** if there is there is a path between any two vertices. A directed graph is ***strongly connected*** if there is always a directed path between vertices. Any subgraph that is connected can be referred to as a ***connected component***.

Graph Terminology - contd.

Degree and graph types

The ***degree*** of a vertex is the number of edges connected to it. The most popular person will have a vertex of the highest degree. Remote hermits may have degree-zero vertices. In ***dense*** graphs, most vertices have high degree. In ***sparse*** graphs, most vertices have low degree. In a ***regular graph***, all vertices have exactly the same degree.

Clique

A graph is called ***complete*** if every pair of vertices is connected by an edge. A ***clique*** is a sub-graph that is complete.

Graph Terminology - contd.

Cycles and Dags

A ***cycle*** is a path where the last vertex is adjacent to the first. A cycle in which no vertex is repeated is said to be a ***simple cycle***. The shortest cycle in a graph determines the graph's ***girth***. A simple cycle that passes through every vertex is said to be a ***Hamiltonian cycle***. An undirected graph with no cycles is a ***tree*** if it is connected, or a ***forest*** if it is not. A directed graph with no directed cycles is said to be a ***directed acyclic graph*** (or a DAG)

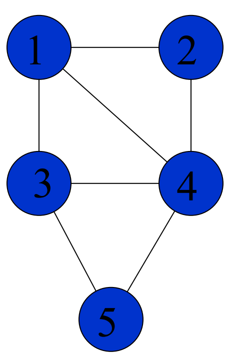
Graphs

- We will typically express running times in terms of |E| and |V| (often dropping the |'s)
  - If $|E| \approx |V|^2$ the graph is *dense*
  - If $|E| \approx |V|$ the graph is *sparse*
- If you know you are dealing with dense or sparse graphs, different data structures may make sense

Graphs

- Motivation and Terminology
- Representations
- Traversals

Adjacency Matrix



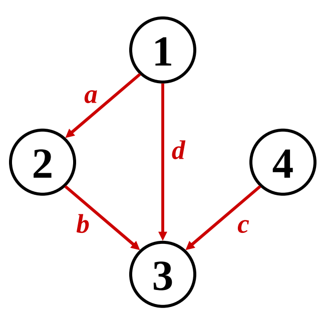
	1	2	3	4	5
1	0	1	1	1	0
2	1	0	0	1	0
3	1	0	0	1	1
4	1	1	1	0	1
5	0	0	1	1	0

Representing Graphs

- Assume V = {1, 2, …, n}
- An *adjacency matrix* represents the graph as a n x n matrix A:
  - A[i, j] = 1 if edge (i, j) $\in$ E
    = 0 if edge (i, j) $\notin$ E

Graphs: Adjacency Matrix

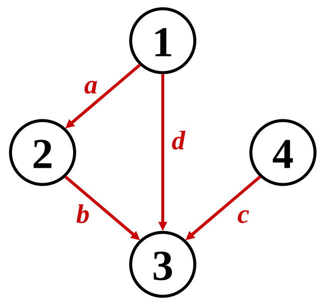
- Example:



A	1	2	3	4
1				
2				
3			??	
4				

Graphs: Adjacency Matrix

- Example:



A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

Graphs: Adjacency Matrix

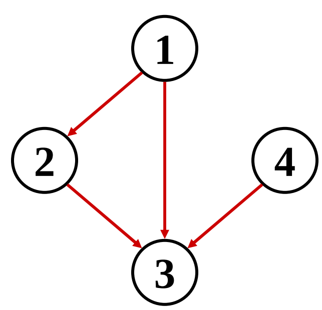
- *How much storage does the adjacency matrix require?*
- A: $O(V^2)$
- *What is the minimum amount of storage needed by an adjacency matrix representation of an undirected graph with 4 vertices?*
- A: ? bits
  - Undirected graph → matrix is symmetric
  - No self-loops → don't need diagonal

Graphs: Adjacency Matrix

- The adjacency matrix is a dense representation
  - Usually too much storage for large graphs
  - But can be very efficient for small graphs
- Most large interesting graphs are sparse
  - E.g., planar graphs, in which no edges cross, have $|E| = O(|V|)$ by Euler's formula
  - For this reason the *adjacency list* is often a more appropriate respresentation

Graphs: Adjacency List

- Adjacency list: for each vertex $v \in V$, store a list of vertices adjacent to v
- Example:
  - Adj[1] = {2,3}
  - Adj[2] = {3}
  - Adj[3] = {}
  - Adj[4] = {3}

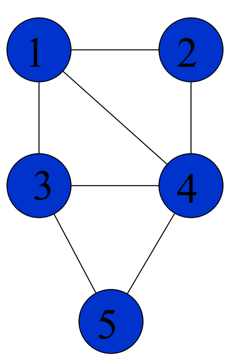


- Variation: can also keep a list of edges coming *into* vertex

Graphs: Adjacency List

- How much storage is required?
  - The *degree* of a vertex v = # incident edges
    - Directed graphs have in-degree, out-degree
  - For directed graphs, # of items in adjacency lists is
    Σ out-degree(v) = |E|
    takes Θ(V + E) storage (*Why?*)
  - For undirected graphs, # items in adj lists is
    Σ degree(v) = 2 |E| (*handshaking lemma*)
    also Θ(V + E) storage
- So: Adjacency lists take O(V+E) storage

Adjacency List



1 → 2 → 3 → 4
2 → 1 → 4
3 → 1 → 4 → 5
4 → 1 → 2 → 3 → 5
5 → 3 → 4

Tradeoffs Between Adjacency Lists and Adjacency Matrices

Comparison	**Winner (for worst case)**
Faster to test if (x, y) exists?	matrices: $\Theta(1)$ vs. $\Theta(V)$
Faster to find vertex degree?	lists: $\Theta(1)$ vs. $\Theta(V)$
Less memory on sparse graphs?	lists: $\Theta(V+E)$ vs. $\Theta(V^2)$
Less memory on dense graphs?	matrices: (small win)
Edge insertion or deletion?	matrices: $\Theta(1)$ vs. $\Theta(V)$
Faster to traverse the graph?	lists: $\Theta(E+V)$ vs. $\Theta(V^2)$
Better for most problems?	**lists**