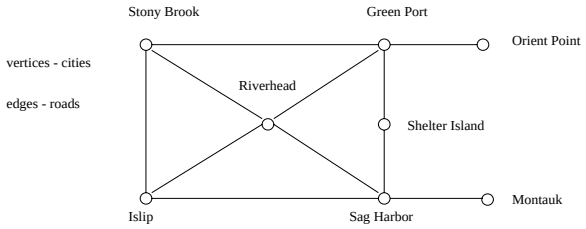
Graphs

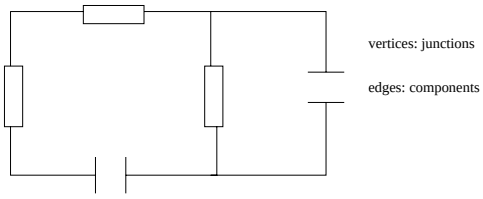
A graph G consists of a set of *vertices* V together with a set E of vertex pairs or *edges*.

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

Example: A network of roads, with cities as vertices and roads between cities as edges.



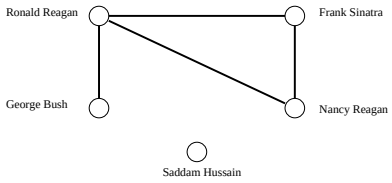
Example: An electronic circuit, with junctions as vertices as components as edges.



To understand many problems, we must think of them in terms of graphs!

The Friendship Graph

Consider a graph where the vertices are people, and there is an edge between two people if and only if they are friends.



This graph is well-defined on any set of people: SUNY SB, New York, or the world.

What questions might we ask about the friendship graph?

- **If I am your friend, does that mean you are my friend?**

  A graph is *undirected* if (x, y) implies (y, x). Otherwise the graph is directed. The "heard-of" graph is directed since countless famous people have never heared of me! The "had-sex-with" graph is presumably undirected, since it requires a partner.

- **Am I my own friend?**

  An edge of the form (x, x) is said to be a *loop*. If x is y's friend several times over, that could be modeled using *multiedges*, multiple edges between the same pair of vertices. A graph is said to be *simple* if it contains no loops and multiple edges.

- **Am I linked by some chain of friends to the President?**

  A *path* is a sequence of edges connecting two vertices. Since *Mel Brooks* is my father's-sister's-husband's cousin, there is a path between me and him!

- **How close is my link to the President?**

  If I were trying to impress you with how tight I am with Mel Brooks, I would be much better off saying that Uncle Lenny knows him than to go into the details of how connected I am to Uncle Lenny. Thus we are often interested in the *shortest path* between two nodes.

- **Is there a path of friends between any two people?**

  A graph is *connected* if there is a path between any two vertices. A directed graph is *strongly connected* if there is a directed path between any two vertices.

- **Who has the most friends?**

  The *degree* of a vertex is the number of edges adjacent to it.

- **What is the largest clique?**

  A social clique is a group of mutual friends who all hang around together. A graph theoretic *clique* is a complete subgraph, where each vertex pair has an edge between them. Cliques are the densest possible subgraphs. Within the friendship graph, we would expect that large cliques correspond to workplaces, neighborhoods, religious organizations, schools, and the like.

- **How long will it take for my gossip to get back to me?**

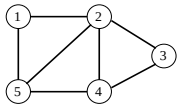
  A *cycle* is a path where the last vertex is adjacent to the first. A cycle in which no vertex repeats (such as 1-2-3-1 verus 1-2-3-2-1) is said to be *simple*. The shortest cycle in the graph defines its *girth*, while a simple cycle which passes through each vertex is said to be a *Hamiltonian cycle*.

Data Structures for Graphs

There are two main data structures used to represent graphs.

Adjacency Matrices

An *adjacency matrix* is an $n \times n$ matrix, where $M[i,j] =$ 0 iff there is no edge from vertex i to vertex j



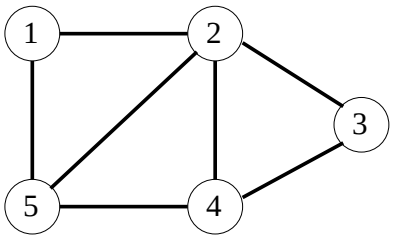
0	1	0	0	1
1	0	1	1	1
0	1	0	1	0
0	1	1	0	1
1	1	0	1	0

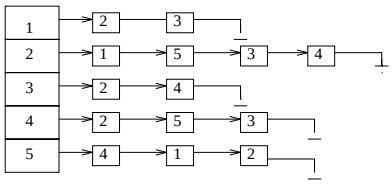
It takes $\Theta(1)$ time to test if edge (i,j) is in a graph represented by an adjacency matrix.

Can we save space if (1) the graph is undirected? (2) if the graph is sparse?

Adjacency Lists

An *adjacency list* consists of a $Nx1$ array of pointers, where the ith element points to a linked list of the edges incident on vertex i.





To test if edge (i, j) is in the graph, we search the ith list for j, which takes $O(d_i)$, where d_i is the degree of the ith vertex.

Note that d_i can be much less than n when the graph is sparse. If necessory, the two *copies* of each edge can be linked by a pointer to facilitate deletions.

Tradeoffs Between Adjacency Lists and Adjacency Matrices

Comparison	Winner
Faster to test if (x, y) exists?	matrices
Faster to find vertex degree?	lists
Less memory on small graphs?	lists $(m + n)$ vs. (n^2)
Less memory on big graphs?	matrices (small win)
Edge insertion or deletion?	matrices $O(1)$
Faster to traverse the graph?	lists $m + n$ vs. n^2
Better for most problems?	lists

Both representations are very useful and have different properties, although adjacency lists are probably better for most problems.

23.1-5 The square *of a directed graph $G = (V, E)$ is the graph $G^2 = (V, E^2)$ such that $(u, w) \in E^2$ iff for some $v \in V$, both $(u, v) \in E$ and $(v, w) \in E$; ie. there is a path of exactly two edges.*

Give efficient algorithms for both adjacency lists and matricies.

Given an adjacency matrix, we can check in constant time whether a given edge exists. To discover whether there is an edge $(u, w) \in G^2$, for each possible intermediate vertex v we can check whether (u, v) and (v, w) exist in $O(1)$.

Since there are at most n intermediate vertices to check, and n^2 pairs of vertices to ask about, this takes $O(n^3)$ time.

With adjacency lists, we have a list of all the edges in the graph. For a given edge (u, v), we can run through all the edges from v in $O(n)$ time, and fill the results into an adjacency matrix of G^2, which is initially empty.

It takes $O(mn)$ to construct the edges, and $O(n^2)$ to initialize and read the adjacency matrix, a total of $O((n + m)n)$. Since $n \leq m$ unless the graph is disconnected, this is usually simplified to $O(mn)$, and is faster than the previous algorithm on sparse graphs.

Why is it called the square of a graph? Because the square of the adjacency matrix is the adjacency matrix of the square! This provides a theoretically faster algorithm.