Lecture 10:
Graph Data Structures

Steven Skiena

Department of Computer Science
State University of New York
Stony Brook, NY 11794–4400

http://www.cs.stonybrook.edu/~skiena

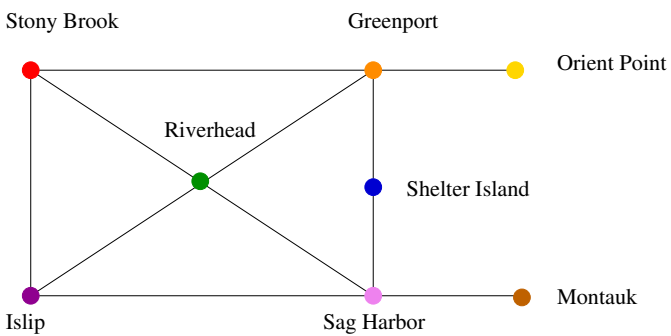
Topic: Introduction to Graphs

Graphs

Graphs are one of the unifying themes of computer science. A graph $G = (V, E)$ is defined by a set of *vertices* V, and a set of *edges* E consisting of ordered or unordered pairs of vertices from V.

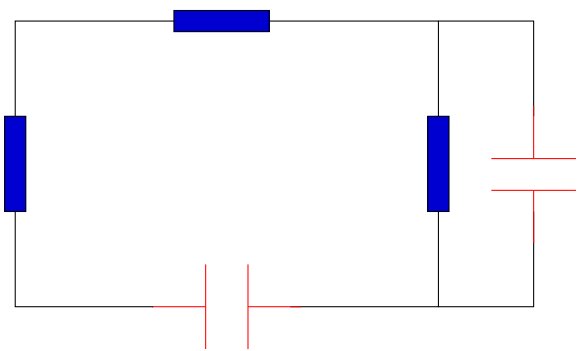
Road Networks

In modeling a road network, the vertices may represent the cities or junctions, certain pairs of which are connected by roads/edges.



Electronic Circuits

In an electronic circuit, with junctions as vertices as components as edges.



Other Graphs/Networks

- Social networks
- The World Wide-Web
- Control flow within a computer program
- Pairwise similarities between items

Questions?

Topic: Flavors of Graphs

Flavors of Graphs

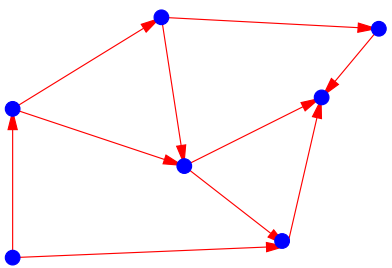
The first step in any graph problem is determining which flavor of graph you are dealing with.

Learning to talk the talk is an important part of walking the walk.

The flavor of graph has a big impact on which algorithms are appropriate and efficient.

Directed vs. Undirected Graphs

A graph $G = (V, E)$ is *undirected* if edge $(x, y) \in E$ implies that (y, x) is also in E.

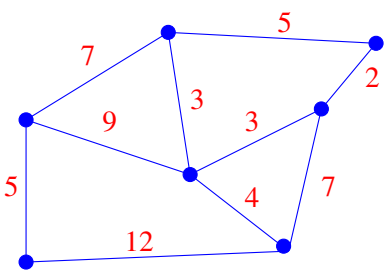


directed

Road networks *between* cities are typically undirected.
Street networks *within* cities are almost always directed because of one-way streets.
Most graphs of graph-theoretic interest are undirected.

Weighted vs. Unweighted Graphs

In *weighted* graphs, each edge (or vertex) of G is assigned a numerical value, or weight.



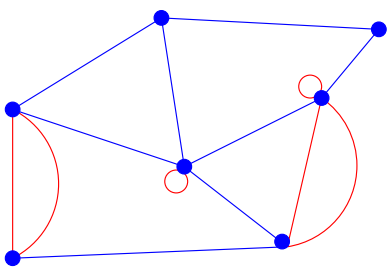
weighted

The edges of a road network graph might be weighted with their length, drive-time or speed limit.

In *unweighted* graphs, there is no cost distinction between various edges and vertices.

Simple vs. Non-simple Graphs

Certain types of edges complicate working with graphs. A *self-loop* is an edge (x, x) involving only one vertex.
An edge (x, y) is a *multi-edge* if it occurs more than once in the graph.

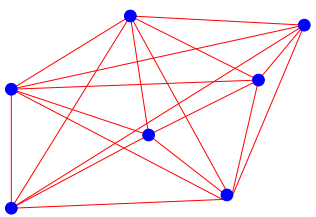


non−simple

Any graph which avoids these structures is called *simple*.
Are you your own friend?

Sparse vs. Dense Graphs

Graphs are *sparse* when only a small fraction of the possible number of vertex pairs have edges defined between them.



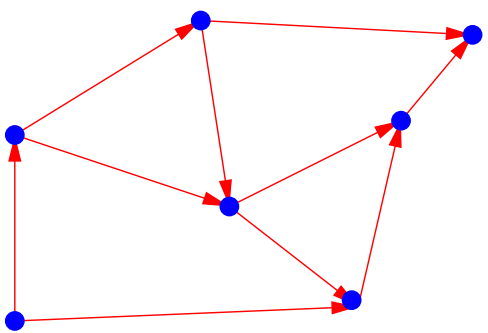
dense

Graphs are usually sparse due to application-specific constraints. Road networks must be sparse because of road junctions.

Typically dense graphs have a quadratic number of edges while sparse graphs are linear in size.

Cyclic vs. Acyclic Graphs

An *acyclic* graph does not contain any cycles. *Trees* are connected acyclic *undirected* graphs.

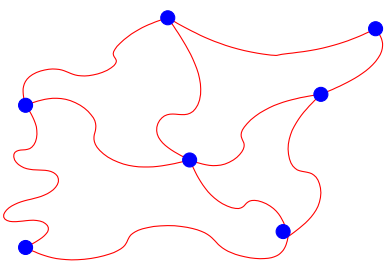


acyclic

Directed acyclic graphs are called *DAGs*. They arise naturally in scheduling problems, where a directed edge (x, y) indicates that x must occur before y.

Embedded vs. Topological Graphs

A graph is *embedded* if the vertices and edges have been assigned geometric positions.



topological

Is there a drawing of the graph that matters, as in the road and electrical circuit networks?

Example: TSP or Shortest path on points in the plane.

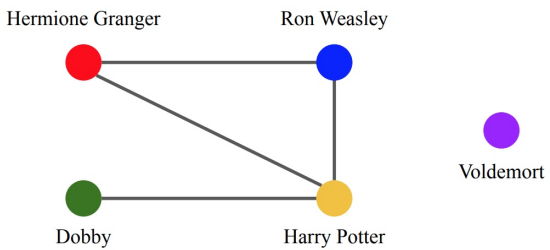
Example: Grid graphs and planar graphs.

Questions?

Topic: Social Networks

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: Stony Brook, New York, or the world.

What questions might we ask about the friendship graph?

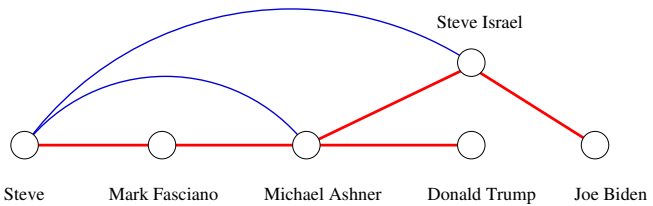
If I am your friend, does that mean you are my friend?

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 heard of me!
- The "had-sex-with" graph is presumably undirected, since it requires a partner.

Am I linked by some chain of friends to a particular celebrity?

A *path* is a sequence of edges connecting two vertices.



If I were trying to impress you with how tight I am with the President, I would be much better off saying that a guy in my book club knows them than to go into the details of how I got into the book club.

We are thus interested in the *shortest path* between 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.

The notation of *six degrees of separation* presumes the world's social network is a connected graph.

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.

Questions?

Topic: Data Structures for Graphs

Data Structures for Graphs: Adjacency Matrix

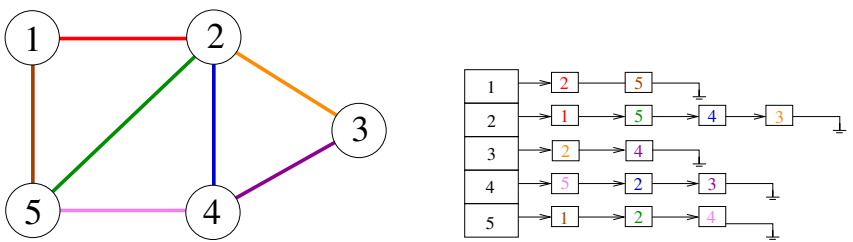
There are two main data structures used to represent graphs. We assume the graph $G = (V, E)$ contains n vertices and m edges.

We can represent G using an $n \times n$ matrix M, where element $M[i, j]$ is, say, 1, if (i, j) is an edge of G, and 0 if it isn't. It may use excessive space for graphs with many vertices and relatively few edges, however.

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

Adjacency Lists

An *adjacency list* consists of a $N \times 1$ 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 necessary, 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.

Edges in Adjacency Lists

```
typedef struct edgenode {
    int y;                  /* adjacency info */
    int weight;             /* edge weight, if any */
    struct edgenode *next;  /* next edge in list */
} edgenode;
```

Graph Representation

```
typedef struct {
    edgenode *edges[MAXV+1];  /* adjacency info */
    int degree[MAXV+1];       /* outdegree of each vertex */
    int nvertices;            /* number of vertices in the graph */
    int nedges;               /* number of edges in the graph */
    int directed;             /* is the graph directed? */
} graph;
```

The degree field counts the number of meaningful entries for the given vertex.

An undirected edge (x, y) appears twice in any adjacency-based graph structure, once as y in x's list, and once as x in y's list.

Initializing a Graph

```
void initialize_graph(graph *g, bool directed) {
    int i;    /* counter */

    g->nvertices = 0;
    g->nedges = 0;
    g->directed = directed;

    for (i = 1; i <= MAXV; i++) {
        g->degree[i] = 0;
    }

    for (i = 1; i <= MAXV; i++) {
        g->edges[i] = NULL;
    }
}
```

Reading a Graph

A typical graph format consists of an initial line featuring the number of vertices and edges in the graph, followed by a listing of the edges at one vertex pair per line.

```
void read_graph(graph *g, bool directed) {
    int i;              /* counter */
    int m;              /* number of edges */
    int x, y;           /* vertices in edge (x,y) */

    initialize_graph(g, directed);

    scanf("%d %d", &(g->nvertices), &m);

    for (i = 1; i <= m; i++) {
        scanf("%d %d", &x, &y);
        insert_edge(g, x, y, directed);
    }
}
```

Inserting an Edge

```
void insert_edge(graph *g, int x, int y, bool directed) {
    edgenode *p;    /* temporary pointer */

    p = malloc(sizeof(edgenode));    /* allocate edgenode storage */

    p->weight = 0;
    p->y = y;
    p->next = g->edges[x];

    g->edges[x] = p;    /* insert at head of list */

    g->degree[x]++;

    if (!directed) {
        insert_edge(g, y, x, true);
    } else {
        g->nedges++;
    }
}
```

Questions?