CSE 373 Analysis of Algorithms
Fall 2016
Instructor: Prof. Sael Lee

LEC09: GRAPH DATA STRUCTURES

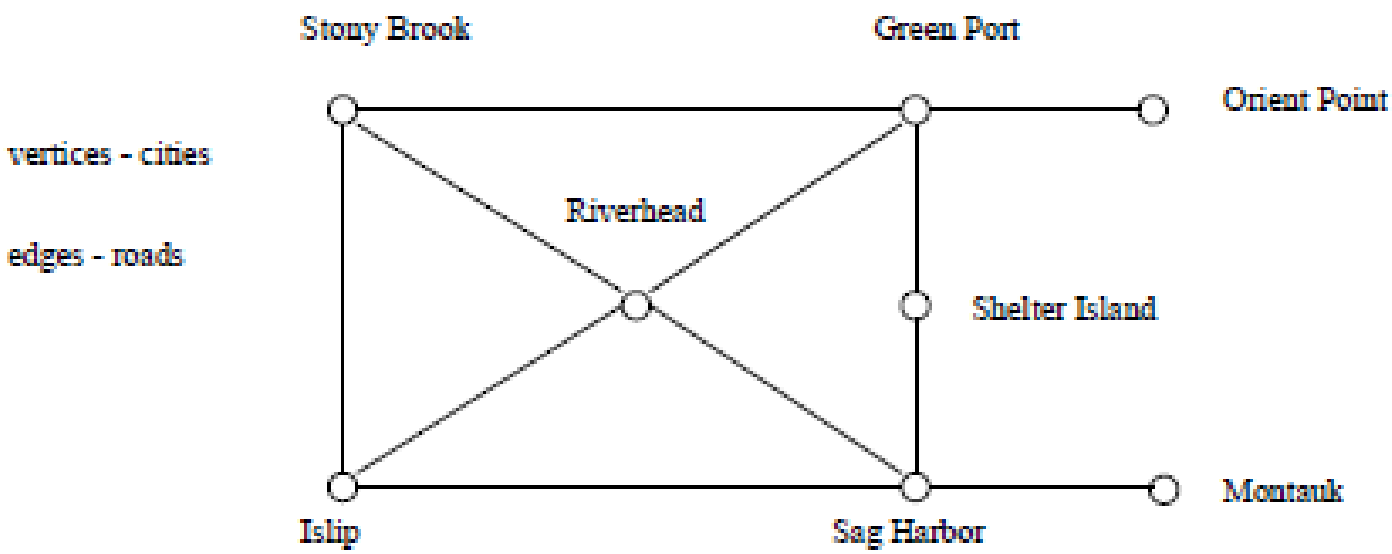
Lecture slide courtesy of Prof. Steven Skiena

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 .

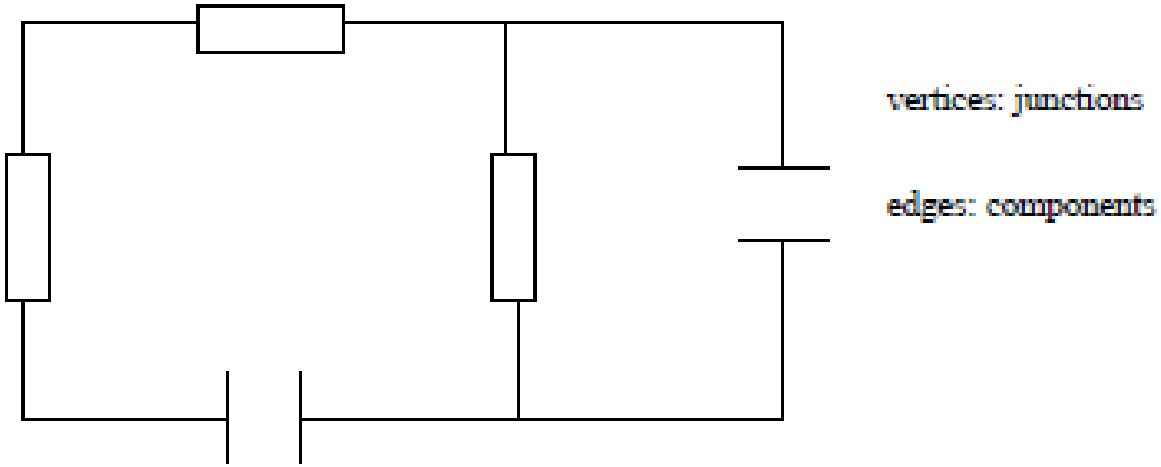
GRAPH EXAMPLES: ROAD NETWORKS

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



GRAPH EXAMPLES: ELECTRONIC CIRCUITS

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



FLAVORS OF GRAPHS

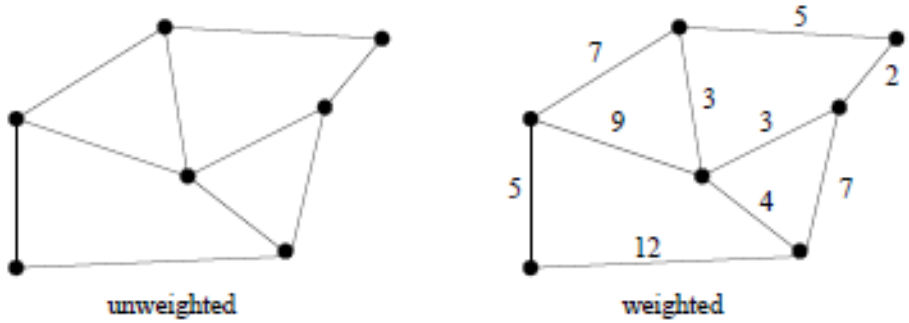
- The first step in any graph problem is determining which flavor of graph you are dealing with.
- Learning to talk the talk is an important part of walking the walk.
- The flavor of graph has a big impact on which algorithms are appropriate and efficient.

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

WEIGHTED VS. UNWEIGHTED GRAPHS

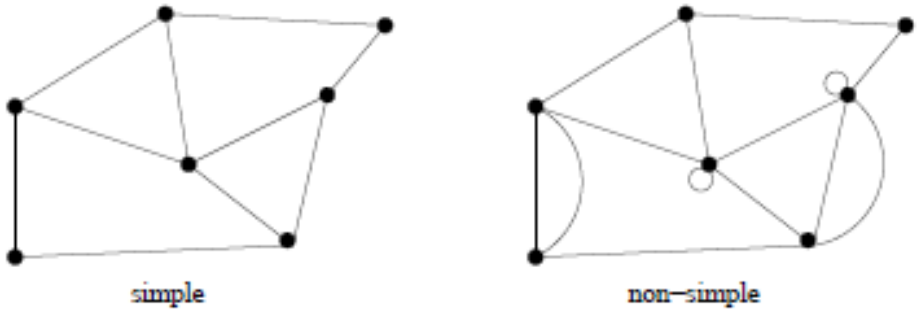
- In *weighted* graphs, each edge (or vertex) of G is assigned a numerical value, or weight.



- 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

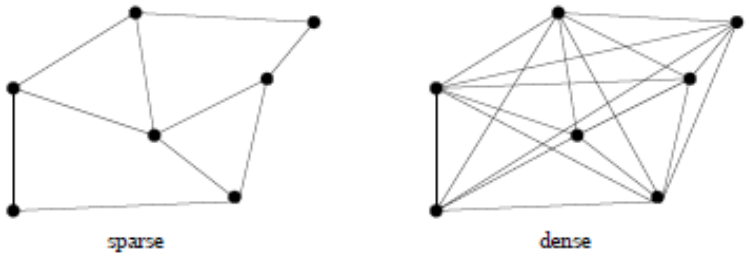
- Certain types of edges complicate the task of working with graphs. A *self-loop* is an edge (x; x) involving only one vertex.
- An edge (x; y) is a *multi-edge* if it occurs more than once in the graph.



- Any graph which avoids these structures is called *simple*.

SPARSE VS. DENSE GRAPHS

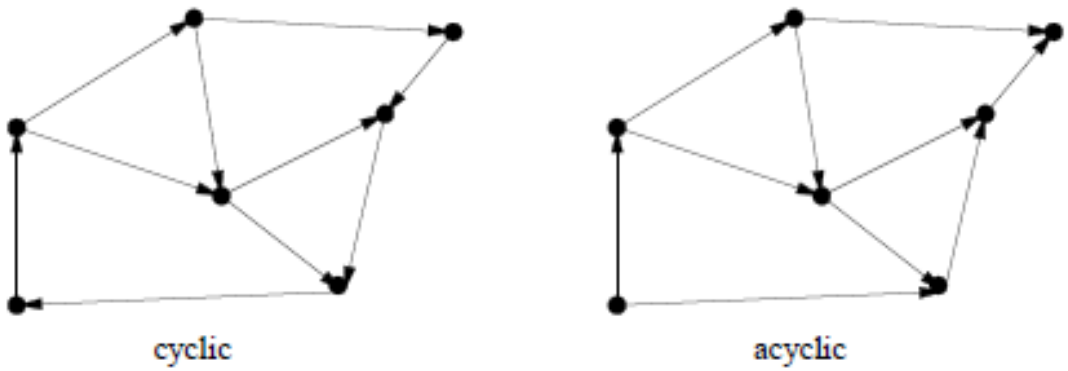
- Graphs are ***sparse*** when only a small fraction of the possible number of vertex pairs actually have edges defined between them.



- 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

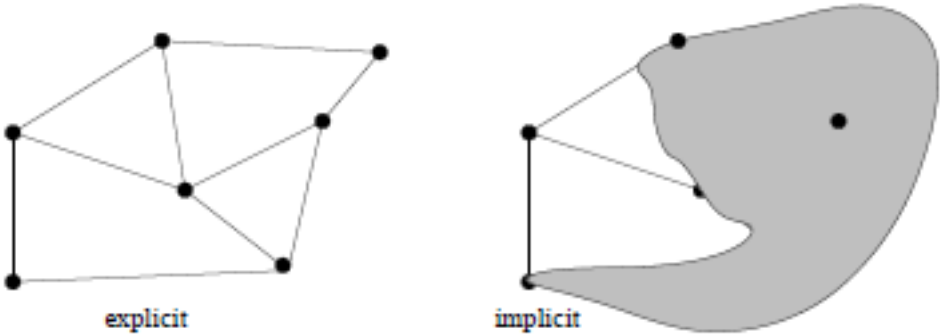
- An ***acyclic* graph** does not contain any cycles.
  - ***Trees*** are connected acyclic *undirected* graphs.



- **Directed acyclic graph**s are called ***DAG***s. They arise naturally in scheduling problems, where a directed edge (x; y) indicates that x must occur before y.

IMPLICIT VS. EXPLICIT GRAPHS

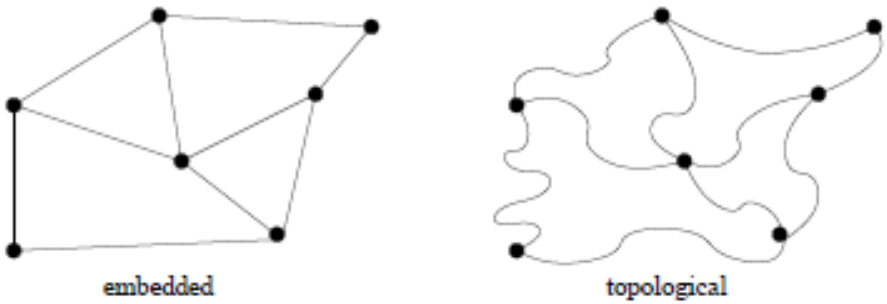
- Many graphs are not explicitly constructed and then traversed, but built as we use them.



- A good example arises in backtrack search.

EMBEDDED VS. TOPOLOGICAL GRAPHS

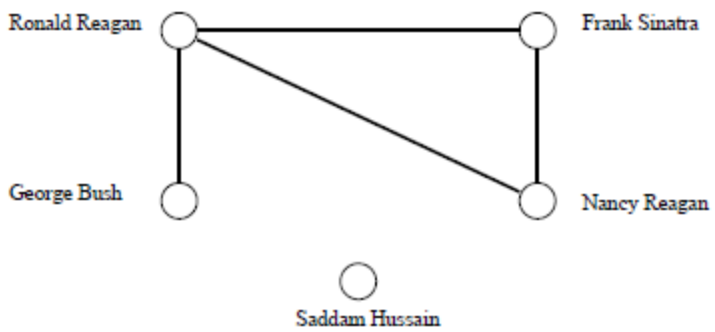
- A graph is ***embedded*** if the vertices and edges have been assigned geometric positions.



  - Example: TSP or Shortest path on points in the plane.
  - Example: Grid graphs.
  - Example: Planar 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?

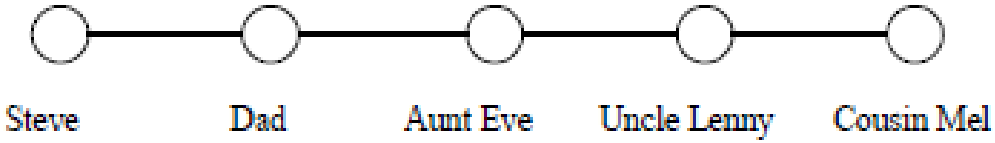
- A graph is ***undirected*** if (x, y) implies (y, x). Otherwise the graph is **directed**.
  - The "heard-of" graph is directed since countless famous people have never heard of me!
  - The "play-table tennis-with" graph is presumably undirected, since it requires a partner.

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 (Steve) and him (Mel)!



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.

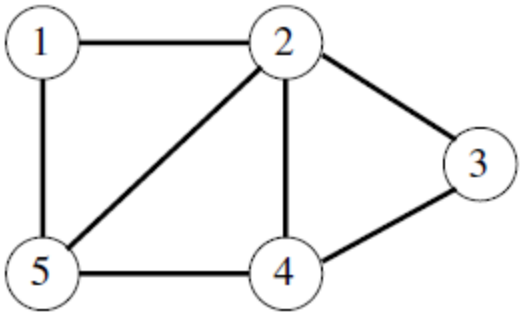
WHO HAS THE MOST FRIENDS?

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

DATA STRUCTURES FOR GRAPHS: ADJACENCY MATRIX

- We assume the graph G = (V, E) contains *n* vertices and *m* edges.
- There are two main data structures used to represent graphs.
- We can represent G using an *nxn* matrix *M*, where element *M[i, j]* is, say, 1, if (i, j) is an edge of G, and 0 if it isn't.

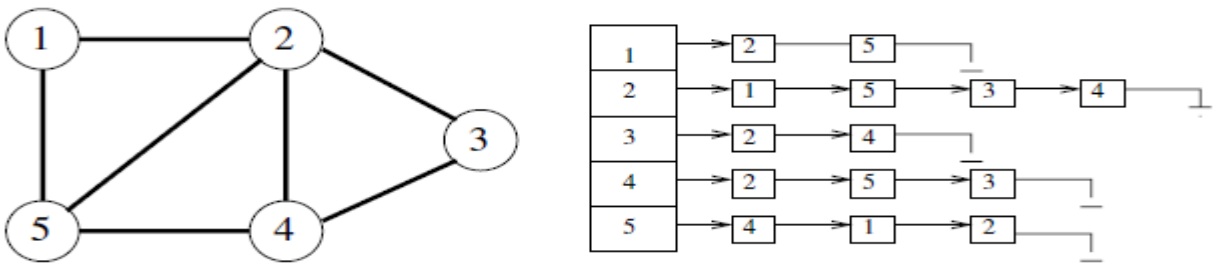
	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0



- 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?

DATA STRUCTURES FOR GRAPHS: ADJACENCY LISTS

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



- To test if edge (i, j) is in the graph
  - 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.

ADJACENCY LIST

- Keep count of the number of vertices, and
- Assign each vertex a unique identification number fro m 1 to *nvertices*
- Represent the edges using an array of linked lists

```
#define MAXV 1000                    /* maximum number of vertices */
typedef struct {
        int y;                       /* adjacency info */
        int weight;                  /* edge weight, if any */
        struct edgenode *next;       /* next edge in list */
} edgenode;
```

Represent directed edge (*x*, *y*) by an *edgenode y* in *x*'s adjacency list.

ADJANCENCY LIST & EDGE REPRESENTATION

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

- The *degree* field of the graph counts the number of meaningful entries for the given vertex.
- An undirected edge (*x*, *y*) appears twice
  - once as *y* in *x*'s list, and once as *x* in *y*'s list.
- The boolean flag *directed* identifies whether the given graph is to be interpreted as directed or undirected.

INITIALIZING A GRAPH

```
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;
}
```

EXAMPLE USAGE: READING A GRAPH

- A typical graph file 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.

```
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++) f
                scanf(”%d %d”,&x,&y);
                insert_edge(g,x,y,directed);
        }
}
```

INSERTING AN EDGE

```
Insert_edge(graph *g, int x, int y, bool directed)
{
        edgenode *p;                        /* temporary pointer */
        p = malloc(sizeof(edgenode));       /* allocate edgenode storage */

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

        g->edges[x] = p;                    /* insert at head of list */
        g->degree[x] ++;
        if (directed == FALSE)
                insert_edge(g,y,x,TRUE);
        else
                g->nedges ++;
}
```

```
print_graph(graph *g)
{
        int i;                  /* counter */
        edgenode *p;            /* temporary pointer */

        for (i=1; i<=g->nvertices; i++) {
                printf("%d: ",i);
                p = g->edges[i];
                while (p != NULL) {
                        printf(" %d",p->y);
                        p = p->next;
                }
                printf("\n");
        }
}
```

- It is a good idea to use a well-designed graph data type as a model for building your own, or even better as the foundation for your application.
- Recommended as the best-designed general-purpose graph data structures currently available
  - LEDA (see Section 19.1.1 (page 658)) or
  - Boost (see Section 19.1.3 (page 659))