Graphs

Comp Sci 1575 Data Structures

Missouri S&T | Computer Science

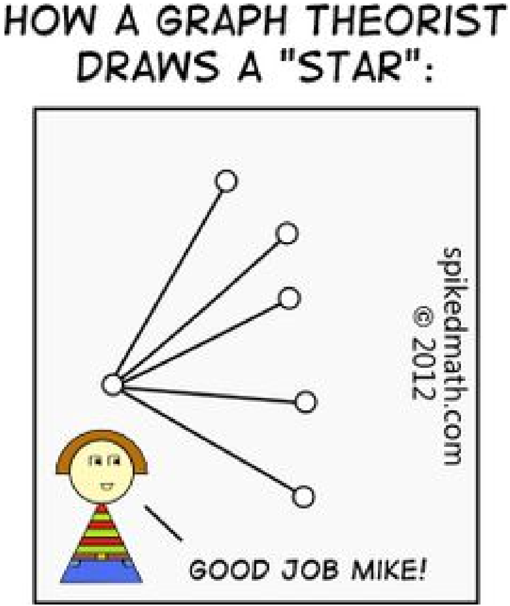
Outline

Outline

Graphs and networks big picture

- Mathematical structure used to model pairwise relations between objects.
- Objects correspond to mathematical abstractions called vertices (also called nodes or points) and each of the related pairs of vertices is called an edge (also called an arc or line)
- Network theory is a part of graph theory: a network can be defined as a graph in which nodes and/or edges have attributes (e.g. names).

Layouts do not necessarily imply position



Graphs and networks big picture

```
                         COLLECTION
                        /          \
UNORDERED_COLLECTION     ORDERED_COLLECTION
        |                      |              |
       BAG                  LINEAR        NON_LINEAR
        |                     |              |          |
       SET               LINKED_LIST     TREE      DIRECTED_GRAPH
        |                     |            |              |
    HASH_TABLE              DEQUE       BIN_TREE   UNDIRECTED_GRAPH
                           /     \       /     \
                       STACK   QUEUE   BST    HEAP
```

- Ordered collection of data (there may be a connection between any two data values in the collection)
- Non-linear (every element doesn't just have a previous and next element like linked lists)
- Differs from tree in that there is no "root" node, each node can have more than 1 "parent", and there can be loops

Outline

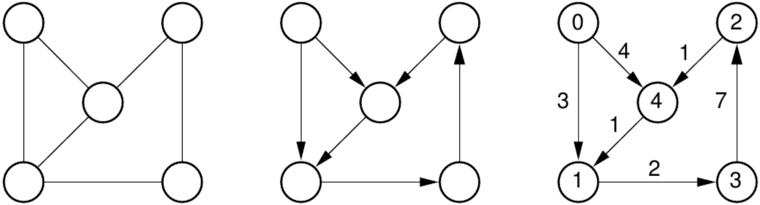
Applications of graph theory

Very widely used at a higher-level problem solving

- Modeling social networks, the spread of ideas or diseases
- Modeling brain networks
- Modeling networks of gene-protein interactions
- Modeling connectivity in computer and communications networks.
- Integrated circuit design
- Representing a travel map as a set of locations with distances between locations; used to compute shortest routes between locations.
- Modeling flow capacities in transportation networks.
- Finding a path from a starting condition to a goal condition; for example, in artificial intelligence problem solving.
- Modeling computer algorithms, showing transitions from one program state to another.
- Finding an acceptable order for finishing sub-tasks in a complex activity, such as constructing large buildings.
- Modeling relationships such as family trees, business or military organizations, and scientific taxonomies.
- Modeling relationships between businesses, markets, and economies

Outline

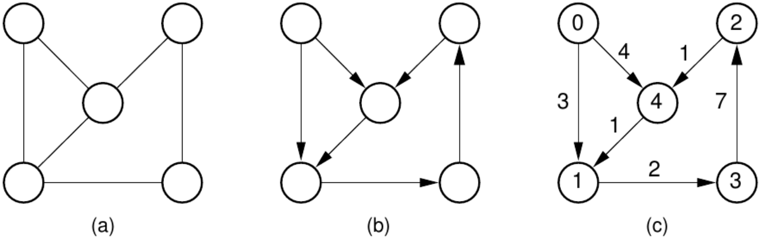
Graphs



(a) (b) (c)

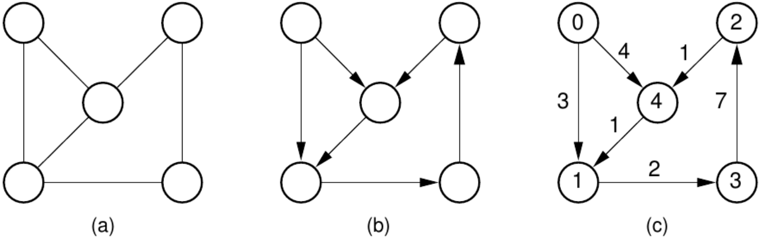
- Finite set of vertices, nodes, or points which are connected by edges, arcs, or lines defined as a set of pairs of:
  1) unordered nodes to define edges, arcs, or lines for an **undirected graph**, or
  2) ordered pairs to define directed edges, directed arcs, or directed lines, or arrows for a **directed graph**.
- May also associate with each edge some edge value, such as a symbolic label or a numeric attribute (cost, capacity, length, etc.).

Graphs



- A graph $G = (V, E)$ consists of a set of vertices V and a set of edges E, such that each edge in E is a connection between a pair of vertices in V (and defined by that pair).
- The number of vertices is written $|V|$, and the number of edges is written $|E|$. $|E|$ can range from zero to a maximum of $|V|^2 - |V|$. Why?? Draw it

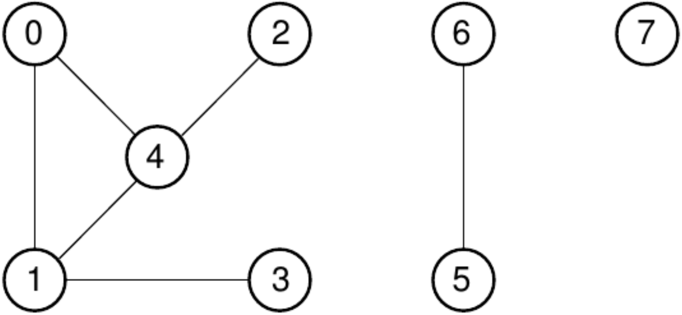
Graphs



(a) A graph. **(b)** A directed graph (digraph). **(c)** A labeled (directed) graph with **weights** associated with the edges. In this example, there is a **simple path** from Vertex 0 to Vertex 3 containing Vertices 0, 1, and 3. Vertices 0, 1, 3, 2, 4, and 1 also form a path, but not a simple path because Vertex 1 appears twice. Vertices 1, 3, 2, 4, and 1 form a **simple cycle**.

Outline

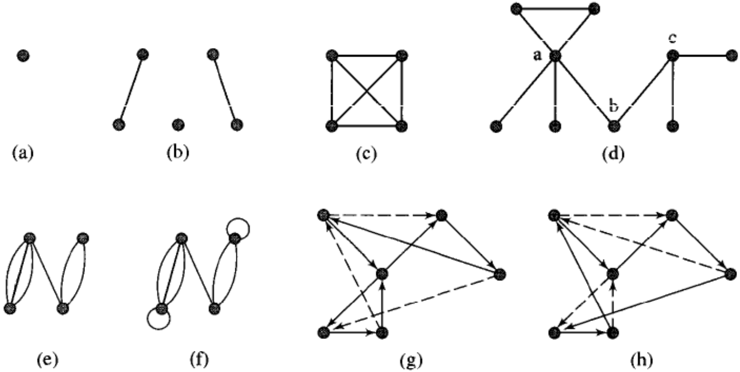
Connected components



- Undirected graph is connected if there is at least one path from any vertex to any other.
- Maximally connected subgraphs of an undirected graph are called connected components (3 above)
- Vertices 0, 1, 2, 3, and 4 form one connected component.
- Vertices 5 and 6 form a second connected component.
- Vertex 7 by itself forms a third connected component.

Outline

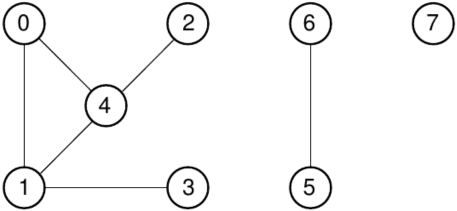
Types of graph



Simple (a-d); Complete (c); Multigraph (e); Pseudograph (f); Circuit in a digraph (g); Cycle in the digraph

Outline

Degree



- **indegree of vertex:** for digraph this is number of incoming edges (not depicted above)
- **outdegree of vertex:** for digraph this is number of outgoing edges (not depicted above)
- **degree of vertex:** in undirected graph this is number of edges incident on vertex; note: a loop in an undirected graph counts as 2
- **degree of graph:** in digraph this is sum of indegree and outdegree of every vertex; in undirected graph this is sum of degree of every vertex; number of edges * 2

Outline

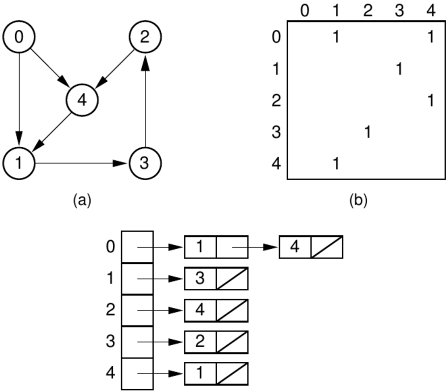
Common operations

- $adjacent(G, x, y)$: tests whether there is an edge from the vertex x to the vertex y;
- $neighbors(G, x)$: lists all vertices y such that there is an edge from the vertex x to the vertex y; note: directed graphs just include outgoing neighbors in this list
- $add\_vertex(G, x)$: adds vertex x, if it is not there;
- $remove\_vertex(G, x)$: removes vertex x, if it is there;
- $add\_edge(G, x, y)$: adds edge from vertex x to vertex y, if it is not there;
- $remove\_edge(G, x, y)$: removes edge from vertex x to vertex y, if it is there;
- $get\_vertex\_value(G, x)$: returns value associated with vertex x;
- $set\_vertex\_value(G, x, v)$: sets value associated with vertex x to v.
- $get\_edge\_value(G, x, y)$: returns value associated with edge (x, y);
- $set\_edge\_value(G, x, y, v)$: sets value associated with edge (x, y) to v.

Outline

Outline

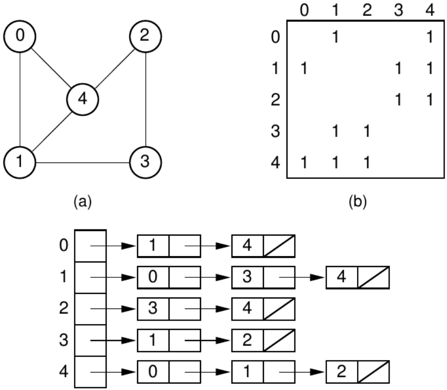
Adjacency with a directed graph (a) below



(a) (b) (c)

- (b) Adjacency matrix $|V|x|V|$ array. Vertices: v_0 through v_{n-1}. Space use is $\Theta((|V|)^2)$. Data are weights, or 0/1.
- (c) Adjacency list $|V|$ items long, with position i storing edges for Vertex v_i. Space use is $\Theta(|V| + |E|)$

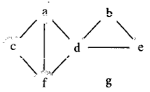
Adjacency with an undirected graph (a) below



(c)

- (b) Adjacency matrix $|V| x |V|$ array. Vertices: v_0 through v_{n-1}. Space use is $\Theta(|V|)^2)$. Data are weights or $0/1$.
- (c) Adjacency list $|V|$ items long, with position i storing edges for Vertex v_i

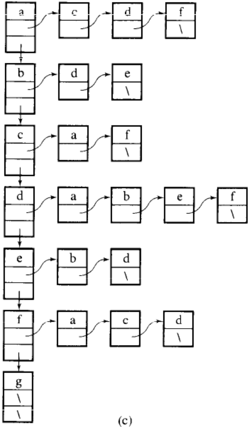
Adjacency with undirected graph: table vs. array



(a)

a	c	d	f	
b	d	e		
c	a	f		
d	a	b	e	f
e	b	d		
f	a	c	d	
g				

(b)



(c)

	a	**b**	**c**	**d**	**e**	**f**	**g**
a	0	0	1	1	0	1	0
b	0	0	0	1	1	0	0
c	1	0	0	0	0	1	0
d	1	1	0	0	1	1	0
e	0	1	0	1	0	0	0
f	1	0	1	1	0	0	0
g	0	0	0	0	0	0	0

(d)

a Graph

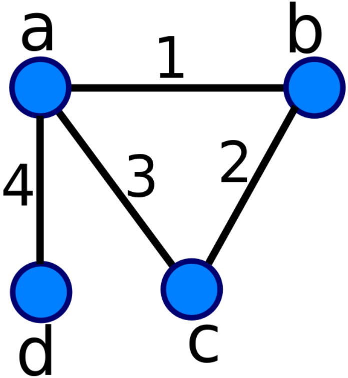
b Adjacency list (table)

c Adjacency list (linked)

d Adjacency matrix

Outline

Incidence matrix



	1	2	3	4
a	1	0	1	1
b	1	1	0	0
c	0	1	1	0
d	0	0	0	1

- An edge connecting Vertices U and V is written (U, V). Such an edge is said to be **incident** on Vertices U and V.

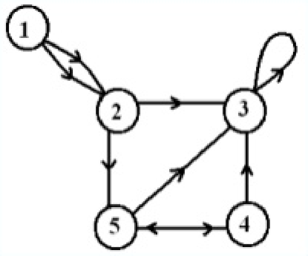
Outline

Edge list

Edge List

Edge List

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



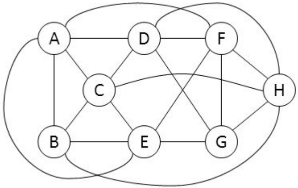
Adjacency List (node list)

Node List

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

Edge list

Graph Representations



node list

A - B C D E F
B - A C E H
C - A B D E H
D - A C F G H
E - A B C F G
F - A D E G H
G - D E F H
H - B C D F G

adjacency matrix

	A	B	C	D	E	F	G	H
A	–	1	1	1	1	1	0	0
B	1	–	1	0	1	0	0	1
C	1	1	–	1	1	0	0	1
D	1	0	1	–	0	1	1	1
E	1	1	1	0	–	1	1	0
F	1	0	0	1	1	–	1	1
G	0	0	0	1	1	1	–	1
H	0	1	1	1	0	1	1	–

edge list

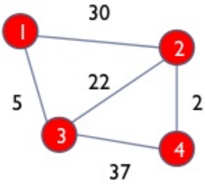
AB
AC
AD
AE
AF
BA
BC
BE
BH
CA
CB
CD
CE
CH
DA
DC
DF
DG
DH
EA
EB
EC
EF
EG
FA
FD
FE
FG
FH
GD
GE
GF
GH
HB
HC
HD
HF
HG

node list - lists the nodes connected to each node

edge list - lists each of the edges as a pair of nodes
undirected edges may be listed twice XY and YX
in order to simplify algorithm implementation

adjacency matrix - for an n-node graph we build an nxn array with 1's indicating edges and 0's no edge the main diagonal of the matrix is unused unless a node has an edge connected to itself. If graph is weighted, 1's are replaced with edge weight values

Edge list



Weights could be:

- *Frequency of interaction in period of observation*
- *Number of items exchanged in period*
- *Individual perceptions of strength of relationship*
- *Costs in communication or exchange, e.g. distance*
- *Combinations of these*

Edge list: add column of weights

Vertex	Vertex	Weight
1	2	30
1	3	5
2	3	22
2	4	2
3	4	37

Adjacency matrix: add weights instead of 1

Vertex	1	2	3	4
1	-	30	5	0
2	30	-	22	2
3	5	22	-	37
4	0	2	37	-

Outline

Asymptotic comparison

	Adjacency list	Adjacency matrix	Incidence matrix
Store graph	$O(\|V\| + \|E\|)$	$O(\|V\|^2)$	$O(\|V\| * \|E\|)$
Add vertex	$O(1)$	$O(\|V\|^2)$	$O(\|V\| * \|E\|)$
Add edge	$O(1)$	$O(1)$	$O(\|V\| * \|E\|)$
Remove vertex	$O(\|E\|)$	$O(\|V\|^2)$	$O(\|V\| * \|E\|)$
Remove edge	$O(\|V\|)$	$O(1)$	$O(\|V\| * \|E\|)$
Query adjacency	$O(\|V\|)$	$O(1)$	$O(\|E\|)$

- **Adjacency list:** Slow to remove vertices and edges, because it needs to find all vertices or edges
- **Adjacency matrix:** Slow to add or remove vertices, because matrix must be resized/copied
- **Incidence matrix:** Slow to add or remove vertices and edges, because matrix must be resized/copied