Chapter 5

Graphs and Directed Graphs (Digraphs)

5.1 INTRODUCTION

This chapter introduces the mathematics of graph theory, the formal representation of a relation (or function) among elements of a set or a pair of sets. The concept of a relation discussed in this chapter is the same concept introduced in Chapter 4. A graph in mathematics is a set of nodes and a set of edges between pairs of those nodes; the edges are ordered or nonordered pairs, or a relation, that defines the pairs of nodes for which the relation being examined is valid. As an example, the people working as systems engineers on a project could be the members of a set. One relation defined over this set could be "works for." Another relation could be "respects." The edges can either be undirected or directed; directed edges depict a relation that requires the nodes to be ordered while an undirected edge defines a relation in which no ordering of the edges is implied. The "works for" and "respects" relations would be examples of ordered relations. An example of an undirected relation would be "sits next to."

A graph enables us to visualize a relation over a set, which makes the characteristics of relations such as transitivity and symmetry easier to understand. The reader will hopefully comprehend the power of visualizing mathematical concepts, as enabled by mathematical graph theory, by the end of reading this chapter.

There is a great deal of terminology associated with graph theory; most of the basics are introduced in this chapter. Notions such as paths and cycles are key to understanding the more complex and powerful concepts of graph theory. There are many degrees of connectedness that apply to a graph; understanding these types of connectedness enables the engineer to understand the basic properties that can be defined for the graph representing some aspect of his or her system. The concepts of adjacency and reachability are the first steps to understanding the ability of an allocated architecture of a system to execute properly.

In addition to aiding in the visualization of relations, graph theory is the basis of many modeling languages. However, there are many more modeling languages, such as IDEF0, that look like graphs but which have no underlying mathematics. The material presented in this chapter is necessary but not sufficient to be able to detect when a modeling language with graphical representations has a mathematical basis or not. For example, understanding the seven properties of unary relations presented in this chapter will enable the reader to detect key assumptions such as transitivity being made or assumed by a modeling language.

Similarly, understanding the difference between a partial order and a total order will give the reader an appreciation of the restrictions and power of a modeling language. A specific example of the use of some of the key concepts in this chapter relates to total and partial orders of elements of a set based upon the relation defined over the set. When a relation induces a total order, the elements of the set over which the relation is defined can be numbered from 1 to n. However, the concept of a partial order suggests that there is more than one possible order from 1 to n of the set's elements that is consistent with the relation. There are a number of applications of a partial order in systems engineering. For example, the set of functions being executed by the system's components can often be executed in more than one sequence. Understanding the many partial orders of functional execution is key to developing test plans to verify the system's performance characteristics. The interested reader is referred to Goodaire and Parmentar [1998], Harary [1972], and Harary et al. [1965] for more details on graph theory. Shin and Levis [2003] provide a performance prediction model based upon a creative application of Petri nets, which is a graph theoretic modeling language based on set theory.

Another specific example of the use of concepts from this chapter relates to the power of hierarchies in the engineering of systems; hierarchies for requirements, functions, and physical components are discussed in Chapter 2. In graph theory a hierarchy is represented as a directed tree. This chapter introduces the terminology associated with trees in graph theory.

The state-of-the-art practice in the engineering of systems is to use a number of graphical concepts that have various amounts of grounding in mathematics as communication mechanisms. The challenge for the future is to develop additional modeling techniques that have significantly more grounding in mathematics while maintaining the quality of the communication among the stakeholders and the engineers in the various disciplines. The software engineering community has been moving in this direction for at least 15 years. The systems engineering community has just started this trek with SysML.

5.2 TERMINOLOGY

A *graph*, G, is a pair of sets, $V(G)$ and $E(G)$. $V(G) = \{n_1, n_2, \ldots, n_N\}$ is the set of vertices or nodes. $E(G) = \{e_{ij}\} \subseteq (V(G) \times V(G))$ is a relation that defines the set of edges that are unordered, not necessarily distinct pairs of nodes. $V(G)$ is a finite, nonempty set; $E(G)$ may be empty and is a subset of the Cartesian product of $V(G)$ with itself.

Due to the undirected nature of the edges in a graph, the edges represent symmetric relations such as "____ is next to ____", "____ is the sibling of ____", "____ is married to ____." Due to the symmetry the order in which the nodes are placed does not matter.

The following Konigsberg bridge problem is one of the earliest known graph theory problems (See Sidebar 5.1). Euler's graph of the Konigsberg bridge problem is known as a *multigraph,* in which two or more edges connecting the same nodes is possible. This graph is also known as a *simple* graph because there are no loops. A *loop* is an edge connecting a node to itself, e_{ii}.

A *directed graph* or *digraph*, G, is a pair of sets, $V(G)$ and $E(G)$; $V(G) = \{n_1, n_2, \ldots, n_N\}$ is the set of vertices or nodes. $V(G)$ is again a finite, nonempty set; $E(G) = \{e_{ij}\}$ is a subset of $V \times V$ or *ordered* pairs of nodes; e_{ij} is said to be from n_i to n_j. Again $E(G)$ may be empty.

The edges in a digraph represent *antisymmetric* or asymmetric relations. Examples are "____ is a parent of ____" and "____ is higher than ____." Here the order in which the nodes are placed in the blanks does matter. Examples include Markov chains and Program Evaluation Review Technique (PERT) charts.

Figure 5.1 shows a sample digraph for the relation "is the parent of." Nodes that are connected by a directed edge are often discussed in terms of parent and child. The node at the tail of the edge is often called the *parent* and the node at the arrow of the edge is called the *child.*

The definitions of loop and simple digraph are the same as above. A multigraph digraph requires multiple copies of e_{ij} for the same i and j in $E(G)$. The presence of e_{ij} and e_{ji} are not sufficient for G to be a multigraph digraph.

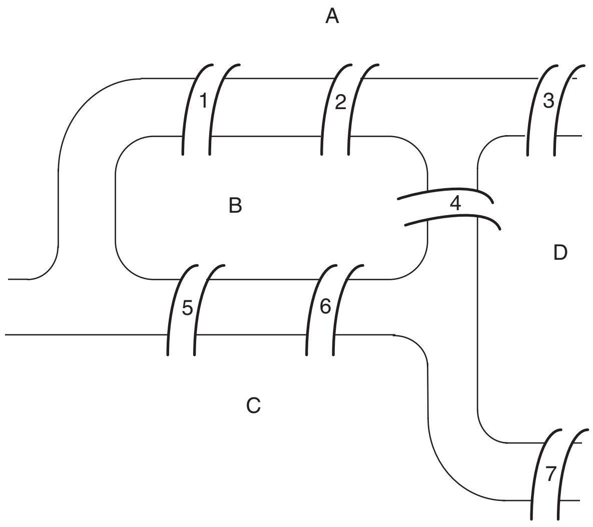
Cardinality of a set $A = |A| =$ the number of elements of A. Note, the cardinality of ϕ is 0. If A has n elements, then $\mathbf{P}(A)$ has cardinality is 2^n.

Order of $G = |V(G)| =$ the number of nodes of G.

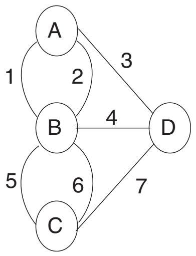
Size of $G = |E(G)| =$ the number of edges of G.

SIDEBAR 5.1: THE KONIGSBERG BRIDGE PROBLEM

In the 1700s the inhabitants of Konigsberg in eastern Prussia were entertained by a puzzle involving seven bridges over the Pregel River. The puzzle posed by mathematicians was whether it was possible to start at any one of the four distinct parcels of land (A, B, C, or D) and find a tour that crossed every bridge once and only once in such a way that the tourer ends up at the same parcel of land from which the tour began. L. Euler, the Swiss mathematician, proved that such a tour could not be done, and in 1736 gave precise conditions for when such a tour could be defined for any system of interconnected bridges.



The following graph is a mathematical representation that Euler created as part of his mathematical proof. The parcels of land are the nodes and the bridges are the edges. Would it be possible to define a graph for this problem in which the bridges were nodes and the parcels were edges?



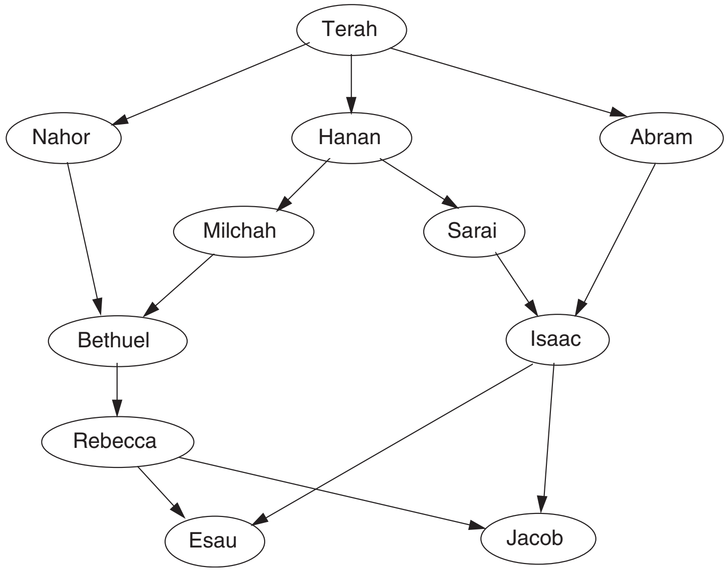


FIGURE 5.1 Sample directed graph for "is the parent of."

The incidence of edges (Fig. 5.2) is defined as: (a) e_{ij} is *incident* on n_i and n_j in a graph and (b) e_{ij} is *incident* from n_i to n_j in a digraph.

Degree of node $n_i =$ the number of edges connected to n_i in a graph, $\deg(n_i)$.

Out degree of node $n_i =$ the number of edges incident from (or exiting) n_i in a digraph, $\deg^-_G(n_i)$.

In degree of node $n_i =$ the number of edges incident to (or entering) n_i in a digraph, $\deg^+_G(n_i)$.

Adjacency – two nodes n_i and n_j are said to be adjacent if e_{ij} or $e_{ji} \in E(G)$.

If $V = \{n_1, n_2, \ldots, n_N\}$ is the set of nodes of an undirected graph G, then

$$\sum_{i=1}^{N} \deg(n_i) = 2|E(G)|.$$

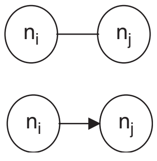


FIGURE 5.2 Samples of incidence.

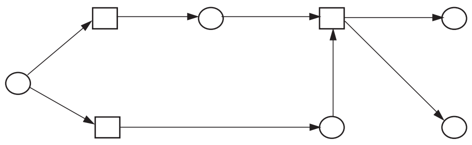


FIGURE 5.3 Sample bipartite graph.

If G is a digraph, then

$$\sum_{\mathrm{i}=1}^{\mathrm{N}} \deg_G^-(n_\mathrm{i}) = \sum_{\mathrm{i}=1}^{\mathrm{N}} \deg_G^+(n_\mathrm{i}) = |E(G)|.$$

Edge labeling of a graph or digraph G is a function f: $E(G) \rightarrow D$, where D is a domain of labels.

Node labeling of a graph or digraph G is a function f: $V(G) \rightarrow D$, where D is a domain of labels.

Recall from Chapter 3 that IDEF0 (Integrated Definition for Function Modeling) uses edge and node labeling.

A *bipartite graph* is a graph (digraph) whose set of nodes can be partitioned into two sets A and B such that no edge connects a node in A to another node in A and, similarly, no edge connects a node in B to another node in B. See Figure 5.3. Is the family tree in Figure 5.1 a bipartite graph?

5.3 PATHS AND CYCLES

A *walk* in a digraph is a sequence of one or more nodes $\{n_0, n_1, \ldots, n_\mathrm{k}\}$ and zero or more edges $\{e_{01}, e_{12}, \ldots, e_{\mathrm{k}-1,\mathrm{k}}\}$. See Figure 5.4. A walk may revisit the same node more than once. A walk is *closed* if its initial and end vertices are the same; otherwise it is *open*. A walk is nontrivial if it has one or more edges.

A *path* is a walk in which each node is distinct (i.e., there are no repeats), except possibly the end nodes. See Figure 5.4. Note since the nodes cannot repeat, the edges cannot repeat.

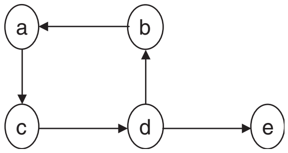


FIGURE 5.4 Digraph with a walk (d-b-a-c-d-e), closed walk, path, and a cycle (a-c-d-b).

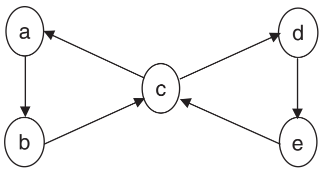


FIGURE 5.5 Digraph with 2 cycles (a-b-c and c-d-e) and a circuit (c-a-b-c-d-e-c).

A *trail* is a walk in which each edge is distinct. Note the same node may be revisited more than once. A closed trail is a *circuit*.

A *circuit* is a nontrivial walk with no repeated edges and whose endpoints are the same. Figure 5.5 has a circuit: *a, b, c, d, e, c, a*.

A *cycle* is a circuit in which all of the nodes are distinct except the first and last. See Figures 5.4 and 5.5. The nodes *a, c, d, b* in Figure 5.4 are a cycle. This cycle could be defined as (*d, b, a, c*) or (*b, a, c, d*) or (*c, d, b, a*) as well, but there is only a single cycle in this graph.

A *nondirected walk* (or *semiwalk*) in a digraph is a sequence of one or more nodes $\{n_0, n_1, \ldots, n_k\}$ and zero or more edges $\{e_{10}$ or e_{01}, e_{21} or e_{12}, …, $e_{k,k-1}$ or $e_{k-1,k}\}$. A semiwalk can travel the wrong way on a directed edge.

A *semipath* (or *chain*) is a semiwalk in which each node is distinct, again with the possible exception of the end nodes. See Figure 5.6.

A semicircuit is a nontrivial semiwalk in which the first and last nodes are the same and no edges are repeated.

A semicycle is semicircuit in which the only repeated nodes are the first and last. See Figure 5.6.

A digraph is *acyclic* if there exists no subgraph that is a cycle.

By now most readers are probably wondering how these definitions are going to be useful. The vocabulary provided by these definitions is very useful in describing when a graph has the seven unary characteristics (e.g., reflexivity, transitivity) from Section 4.3.3. In addition, there are other concepts that will be introduced in this chapter that have general applicability to the engineering of a system, for which this vocabulary will also be useful.

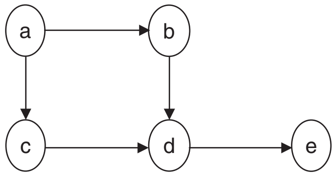


FIGURE 5.6 Digraph with a semipath (b-a-c-d-e) and semicycle (d-b-a-c).

5.4 CONNECTEDNESS

Another vocabulary that proves very useful is connectedness. A *pair of nodes* in a digraph is *weakly connected* if there is a semipath between them, for example, nodes *b* and *c* in Figure 5.6. The nodes are *unilaterally connected* if there is a path between them, for example, all of the pairs of nodes in Figure 5.6 except *b* and *c*. Finally, the nodes are *strongly connected* if there is a path in both directions. No pair of the nodes in Figure 5.6 is strongly connected; every pair of nodes in Figure 5.5 is strongly connected. Note a pair of nodes that is strongly connected is also weakly and unilaterally connected.

A *digraph is weakly* (*unilaterally, strongly*) *connected* if every pair of nodes in the graph is weakly (unilaterally, strongly) connected. The digraph in Figure 5.6 is weakly connected because of the weak connection between nodes *b* and *c*. The digraph in Figure 5.4 is unilaterally connected because node *e* is unilaterally connected with the other four nodes, even though each of the other four nodes is strongly connected to each of the other three. The digraph in Figure 5.5 is strongly connected. The digraphs in Figures 5.1 and 5.3 are weakly connected.

A pair of nodes is *disconnected* if there is no path or semipath between them. A digraph is disconnected if one of its nodes is disconnected from any other node of the graph. A graph is connected if it is not disconnected. All of the digraphs presented so far are connected.

5.5 ADJACENCY AND REACHABILITY*

The adjacency matrix of a graph *G*, *A*(*G*), provides a mathematical representation of which nodes in a digraph are adjacent to each other. Recall that a relation from *N*(*G*) to *N*(*G*) is defined by the edges of *G*, *E*(*G*). So in fact, *A*(*G*) is a description of the relation *E*(*G*) from *N*(*G*) to *N*(*G*).

$$A(G) = [a_{\mathrm{ij}}]$$

is an $N \times N$ Boolean matrix where *N* is the order (number of nodes) of *G*.

$$a_{\mathrm{ij}} = \begin{cases} 1 & \text{if} \quad e_{\mathrm{ij}} \in E(G) \\ 0 & \text{if} \quad e_{\mathrm{ij}} \notin E(G) \end{cases}$$

Note a Boolean matrix is one whose elements are 0 or 1. The row sums of *A*(*G*) give the out-degrees of the associated node; the column sums give the in-degrees. If *G* is not a digraph but a graph, *A*(*G*) will be a symmetric matrix.

* Advanced material.

A node n_j of G is said to be reachable from node n_i of G if there exists a path from n_i to n_j in G. The reachability matrix, $R(G)$, is a Boolean matrix that indicates which nodes can be reached from which other nodes.

$$R(G) = [r_{ij}]$$

is an $N \times N$ Boolean matrix where N is the order of G. To compute $R(G)$ we first compute A, A^2, A^3, ..., $A^{|E(G)|}$

$$r_{ij} = \begin{cases} 1 & \text{if } i = j \\ 1 & \text{if } a_{ij}^{(k)} > 0 \text{ for some } A^k \\ 0 & \text{otherwise} \end{cases}$$

Node n_i is reachable from node n_j if $r_{ij} = 1$. $R(G)$ is also called the *transitive, reflexive closure* of $E(G)$ because $R(G)$ is defined to be a reflexive relation that adds the edges necessary to make $E(G)$ a transitive relation. $R(G)$ is sometimes denoted $R^*(G)$.

The *transitive closure*, $R^+(G)$, is defined to be $R^+(G) = \left[r_{ij}^+\right]$, where

$$r_{ij}^+ = \begin{cases} 1 & \text{if } a_{ij}^{(k)} > 0 \text{ for some } A^k \\ 0 & \text{otherwise} \end{cases}$$

Note in this case the reflexivity of the transitive closure is determined by the reflexivity of $E(G)$.

The *distance* between two nodes is the smallest number of edges between the nodes on any path connecting the two nodes. The distance matrix, $D(G)$, reflects these numbers.

$$D(G) = [d_{ij}]$$

is an $N \times N$ matrix where N is the order of G.

$$d_{ij} = \begin{cases} 0 & \text{if } i = j \\ k & \text{if } n_j \text{ is reachable from } n_i,\ k \text{ is the exponent} \\ & \text{of the first } A^k \text{ in which } a_{ij}^{(k)} > 0 \\ \infty & \text{if there is no path from } n_i \text{ to } n_j \end{cases}$$

5.6 UNARY RELATIONS AND DIGRAPHS

Now directed graphs will be used to visualize the seven properties of unary relations that were introduced in Chapter 4.

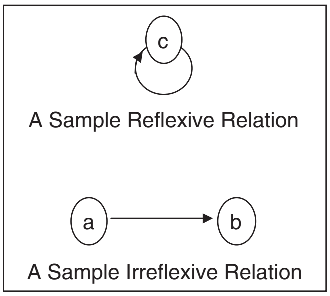


FIGURE 5.7 Reflexive and irreflexive relations.

Reflexivity: $\forall x, x\, R\, x$. That is, all nodes must have loops. The top of Figure 5.7 shows a reflexive relation.

Irreflexivity: $\forall x, x \not R\, x$. That is, no nodes can have loops. The relations shown in the digraphs of Figures 5.1 and 5.3 through 6 are irreflexive. The bottom of Figure 5.7 shows an irreflexive relation.

Note digraphs can depict relations that are neither reflexive nor irreflexive when some of the nodes have loops and others do not.

Symmetry: $\forall x, y$, if $x\, R\, y$, then $y\, R\, x$. That is, there must be a cycle between any two nodes that are adjacent to each other. There is no limitation about arcs besides this. The relations shown in the digraphs of Figures 5.4, 5.5, and 5.6 are not symmetric. The relation in the digraph shown in Figure 5.8 is symmetric.

Antisymmetry: $\forall x, y$, if $x\, R\, y$ and $y\, R\, x$, then $x = y$. That is, there cannot be a cycle between any two nodes that are adjacent to each other. Again, there is no limitation about arcs besides this one; so cycles containing three or more nodes can exist. Any node can have a loop. The digraphs in Figure 5.1 and 5.3 through 5.6 show antisymmetric relations; the relation in the digraph shown in Figure 5.8 is not.

Asymmetry: $\forall x, y$, if $x\, R\, y$, then $y \not R\, x$. That is, there can be no cycle between any two nodes, and there can be no loops. Asymmetric relations must be

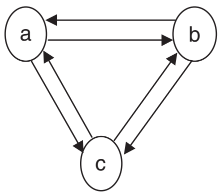


FIGURE 5.8 Digraph of a symmetric relation.

irreflexive. Again cycles among three or more nodes are allowed. The relations in the digraphs shown in Figures 5.1 and 5.3 through 5.6 are asymmetric; the digraph in Figure 5.8 shows a relation that is not.

Note a relation that is irreflexive but in which no node is adjacent to any other node (completely disconnected) is symmetric, antisymmetric, and asymmetric due to the vacuous proof in Chapter 4.

Transitivity: $\forall x, y, z$ if $x\,R\,y$ and $y\ R\ z$, then $x\ R\ z$. This condition only applies to triplets of nodes and requires that there be a semicycle among the three nodes in the triplet. (Note the first and third node in a triplet can be the same, in which case there must be cycle between the two nodes and loops at each node.) A relation to which this condition, or left-hand side, is not applicable (i.e., the "if condition" is never satisfied) will be transitive. Figure 5.9 shows a transitive relation:

dRa and *aRb dRb,*
aRb and *bRe aRe*,
dRb and bRe dRe,
dRa and *aRe dRe*.

Intransitivity: for some x, y, z, if $x \not R z$, then $x\,R\,y$ and $y\,R\,z$. Relations are either transitive or intransitive. Cycles may exist in transitive relations; but note that a transitive relation with cycles that contains three or more nodes means that there must be a cycle between every pair of nodes that is part of the cycle, resulting in a symmetric relation with loops for the subset of nodes in the cycle. The relation in Figure 5.8 is symmetric but not transitive because *aRb* and *bRa,* but *a* is not related to *a;* the same applies for nodes *b* and *c*. Figure 5.10 shows the transitive version the relation of Figure 5.8; the loops are added at each node.

It should be obvious that it is easier to use a directed graph to visualize the properties of unary relations than the mathematical expressions discussed in

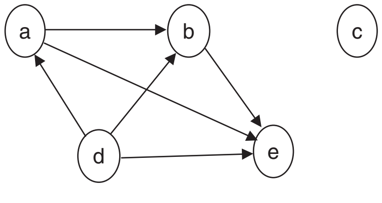


FIGURE 5.9 Digraph of a transitive relation.

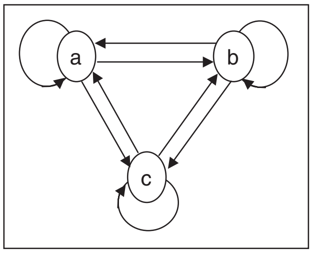


FIGURE 5.10 Transitive version of the digraph in Figure 5.8.

Chapter 4. Likewise, graphical techniques for visualizing functional relationships together with inputs and outputs are much more comprehensible than purely written or tabular methods for most people. "A picture is worth a 1000 words."

5.7 ORDERING RELATIONS*

Relation R is a *partial order* on set A when R is reflexive, antisymmetric, and transitive on the set A. In this case A is called a *partially ordered set*, or *POSET*, written $[A; R]$. *Therefore a* relation *that is a partial order cannot have any cycles.* As discussed in the previous section, a relation that is transitive and has cycles must have pairs of nodes that are symmetric. If any pair of nodes in a relation is symmetric, then the relation cannot be antisymmetric.

Two elements a_1 and a_2 in A are said to be *comparable* under R if either $a_1\ R\ a_2$ or $a_2\ R\ a_1$. Otherwise the elements are incomparable. If every pair of elements is comparable, then $[A; R]$ is *totally ordered.*

A *Hasse diagram* is an undirected graph of the relations between the elements of a partially ordered set. See Figure 5.11. Each element of A is represented as a node. Reflexivity is not represented in the Hasse diagram, thereby eliminating all loops from the graph. Edges that are required by the transitivity property are also omitted; that is, any edge that depicts a shorter path to another node than some other combination of edges is deleted. To draw a Hasse diagram, we place the nodes on a piece of paper such that a_i is below a_j if $a_i\ R\ a_j$. We connect a_i to a_j with an undirected edge if and only if $a_i\ R\ a_j$ and there is no a_k such that $a_i\ R\ a_k$ and $a_k\ R\ a_j$. Figure 5.12 provides a second example of a Hasse diagram and the resulting partial orderings of A.

If there is only one node at the top of the Hasse diagram and only one node at the bottom, then the poset is called a *lattice*. That is, with the transitivity property in force there must be one and only one element, the upper bound or α of A, such that $\alpha\ R\ a_i\ \forall i$, and a second element, the lower bound or ζ of A, such that $a_i\ R\ \zeta\ \forall i$.

* Advanced material.

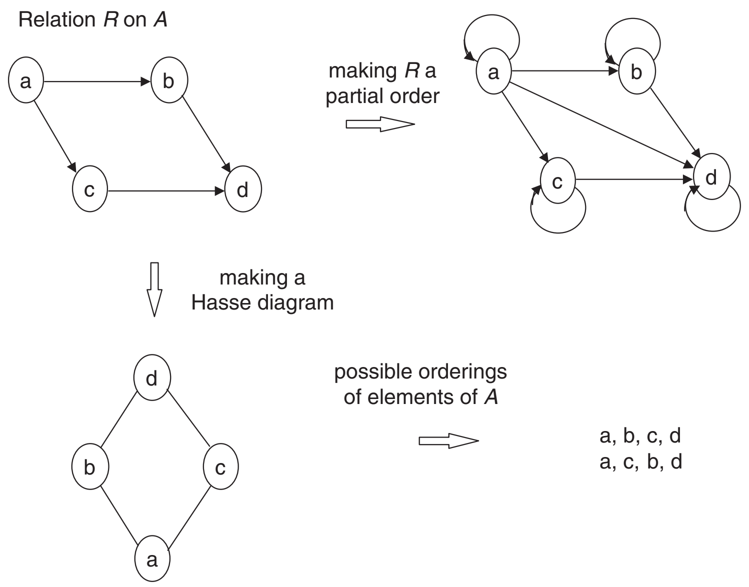
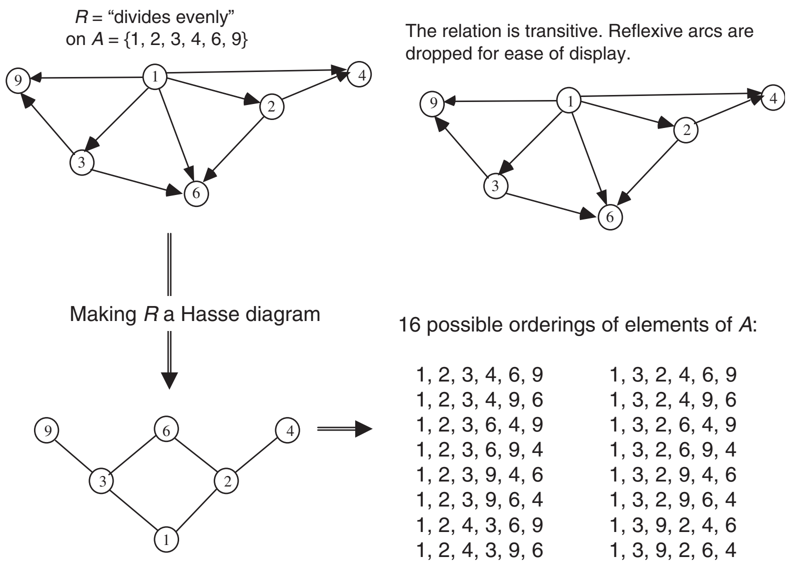

Relation *R* on *A*
making *R* a partial order
making a Hasse diagram
possible orderings of elements of *A*
a, b, c, d
a, c, b, d

FIGURE 5.11 Partial order on a set *A*, Hasse diagram, and partial orderings of *A*.


R = “divides evenly” on *A* = {1, 2, 3, 4, 6, 9}
The relation is transitive. Reflexive arcs are dropped for ease of display.
Making *R* a Hasse diagram
16 possible orderings of elements of *A*:

1, 2, 3, 4, 6, 9	1, 3, 2, 4, 6, 9
1, 2, 3, 4, 9, 6	1, 3, 2, 4, 9, 6
1, 2, 3, 6, 4, 9	1, 3, 2, 6, 4, 9
1, 2, 3, 6, 9, 4	1, 3, 2, 6, 9, 4
1, 2, 3, 9, 4, 6	1, 3, 2, 9, 4, 6
1, 2, 3, 9, 6, 4	1, 3, 2, 9, 6, 4
1, 2, 4, 3, 6, 9	1, 3, 9, 2, 4, 6
1, 2, 4, 3, 9, 6	1, 3, 9, 2, 6, 4

FIGURE 5.12 Second Hasse diagram example.

5.8 ISOMORPHISMS*

Two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, are *isomorphic* if there exists a one-to-one and onto function, f, such that f: $V_1 \rightarrow V_2$ and f preserves adjacency. That is, $E_2 = \{(f(v), f(w)) \mid (v, w) \in E_1\}$. Note that "___ is isomorphic to ___" is an equivalence relation.

An isomorphism f from G_1 to G_2 is not necessarily unique. Some necessary properties for G_1 and G_2 to be isomorphic are: (1) $|V(G_1)| = |V(G_2)|$, (2) $|E(G_1)| = |E(G_2)|$, and (3) if $n_1 \in V(G_1)$, then $\deg^+_{G_1}(n_1) = \deg^+_{G_1}(f(n_1))$ and $\deg^-_{G_1}(n_1) = \deg^-_{G_1}(f(n_1))$.

5.9 TREES

A *tree* is a graph G with no loops in which there is a unique, simple (no loops), nondirected path (or semipath in the case of a digraph) between each pair of nodes. Figure 5.13 shows a graph that is a tree.

A *rooted tree* is a tree in which there is a designated "root" node. In a graph, the root node must have a degree of 1. In Figure 5.13 nodes *a, c,* and *j* could be root nodes. In a directed tree, the root node must have no parents, or an in degree of 0. In Figure 5.14, in the left digraph nodes *a* and *c* could be root nodes; in the right digraph only node *a* can be root node.

A *directed tree* is a rooted tree in which there is a (directed) path from the root to every other node. Note that the tree in Figure 5.13 is not a directed tree because the graph is not a digraph. The right-hand digraph in Figure 5.14 is a directed tree in which node *a* is the root. The left-hand graph is a tree because there exists a semipath from every node to every other node; that is, the graph is weakly connected. The graph is not a directed tree because there is not a path from any root (*a* or *c*) to every other node.

Note the following statements are consistent with the above definitions:

1. A simple nondirected graph G is a tree if and only if G is connected and contains no cycles.
2. A tree with n nodes has exactly $n-1$ edges.
3. A graph G is a tree if and only if G has no cycles and $|E(G)| = |V(G)|-1$.

A directed tree is a graphic representation of a partition, the fundamental construct of our requirements, functional and physical decompositions.

5.9.1 Spanning Trees*

A graph H is a *subgraph* of a graph G if $V(H) \subseteq V(G)$ and $E(H) \subseteq [E(G) \cap (V(H) \times V(H))]$. That is, the nodes in the subgraph must be a subset of the

* Advanced material.

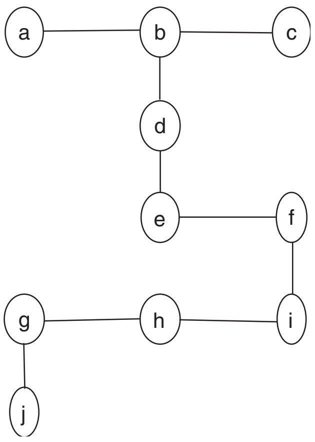


FIGURE 5.13 Sample tree.

nodes in the graph, and the edges in the subgraph must be a subset of those in the graph, with the added stipulation that all of the edges are connected to two nodes, one on each end of the edge.

Graph H is a *proper subgraph* of G if $V(H) \neq V(G)$.

A graph H is a *spanning subgraph* of G if H is a subgraph of G and $V(H) = V(G)$. So a spanning subgraph cannot be a proper subgraph.

Let W be a subgraph of G. The *subgraph induced by W* is the subgraph H of G in which $V(H) = V(W)$ and $E(H) = [E(G) \cap (V(W) \times V(W))]$. That is, H, the subgraph of G induced by W, contains all of the edges of G that are consistent with the nodes of W. A subgraph H of a graph G is called a *spanning*

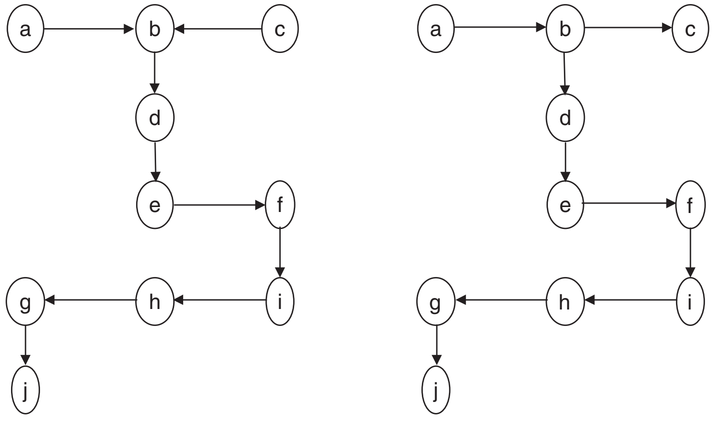


FIGURE 5.14 Sample nondirected and directed trees.

tree of G if (a) H is a tree and (b) $V(H) = V(G)$. A spanning tree that is a directed tree is a *directed spanning tree*.

5.9.2 Directed Trees

Two nodes, n_1 and n_2, in a digraph G are *quasi-strongly connected* if there exists a node n_3 such that there is a path(s) from n_3 to n_1 and from n_3 to n_2. The path from n_3 to n_2 can pass through n_1.

Digraph G is a *quasi-strongly connected digraph* if and only if there is at least one node, r, in G such that there exists a path from r to all of the remaining nodes of G. See Figure 5.15.

Let G be a digraph with $|V(G)| > 1$. Then the following statements are equivalent:

(1) G is a directed tree.
(2) There is a node r in G such that there exists a unique path from r to every node in G.
(3) G is quasi-strongly connected and G – (any edge) is not quasi-strongly connected.
(4) G is quasi-strongly connected and contains a node r such that the in degree of r is 0 and the in degree of every other node in G is 1.

The *height* of a directed tree is the length of the longest path. The height of the directed tree in Figure 5.14 is 8. A directed tree has levels. Level 0 is associated with the root of the directed tree. The first level of the directed tree contains all nodes adjacent to the root, or the children of the root. The second level contains the children of all nodes in level 1, and so on. See Figure 5.16. Note that a directed tree need not be symmetric, that is, reach the same level along every path.

5.9.3 Forest

A *directed forest* is a collection of directed trees. See Figure 5.17. Forests are important in systems engineering as we practice concurrent engineering.

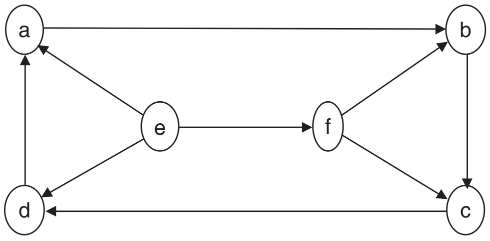


FIGURE 5.15 Quasi-strongly connected digraph.

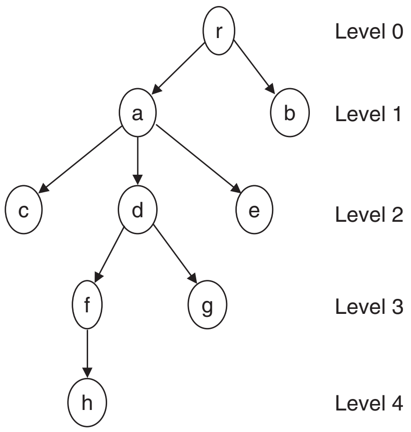


FIGURE 5.16 Levels of a directed tree.

Recall from Chapter 1 that we must be concerned not only with the system that will be used during the operational phase but also with the development, manufacturing, training, deployment, refinement, and retirement systems. The concurrent requirements form a requirements forest.

5.10 FINDING CYCLES AND SEMICYCLES IN A GRAPH

In very large digraphs it will not always be apparent that there are cycles or semicycles. To find the cycles, remove all barren nodes (nodes without children) and border nodes (nodes without parents). Continue this process until there are no remaining barren or border nodes. If there are any nodes remaining, then

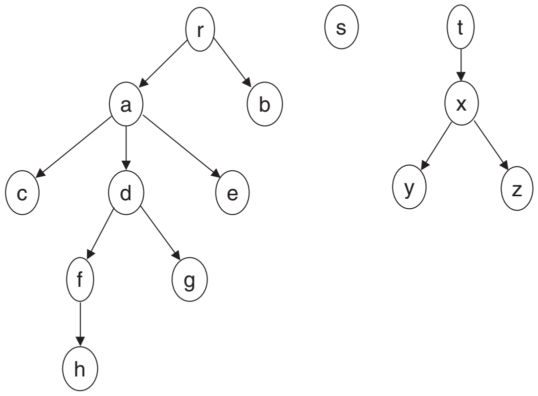


FIGURE 5.17 Sample directed forest.

there are one or more cycles and the remaining nodes are part of at least one of the cycles.

To find the semicycles in a digraph, first replace all of the directed arcs with non-directed arcs. Then remove all nodes of degree 1. Continue this process until there are no remaining nodes of degree 1. If there are any nodes remaining, then there are one or more semicycles, and the remaining nodes are part of at least one of the semicycles.

5.11 REVISITING IDEF0 DIAGRAMS

At a superficial level IDEF0 diagrams resemble the digraphs that we have been discussing. On any IDEF0 page there are nodes, depicted as boxes, and arcs. All of the boxes and edges are labeled as discussed earlier in this chapter. However, we need not look too deep to see some major discrepancies between digraphs and a page of an IDEF0 model. The inputs, controls, outputs, and mechanisms (ICOMs) coming from external sources to the page are not nodes but labels on the edges. These edges, associated with the external ICOMs, do not have a node at one of their ends; this never happened in a digraph since an edge depicted a relation between two elements of a set A and all of the elements of A were shown in the graph.

As mentioned in the previous paragraph, each edge on the IDEF0 diagram is labeled. While there can be labels on the edges in digraphs, all of the digraphs presented in this chapter had none. In a digraph each edge represents the fact that a single relation exists between each pair of connected nodes, *aRb*.

Each node in the IDEF0 diagram is called a function and is named consistently with our understanding of a function, namely a transformation. Yet, digraphs represent a specific relation, which may be a mathematical function if certain conditions are satisfied (see Chapter 4). The relation, or function, in a digraph is represented by the edges, not the nodes.

At an even deeper level, each label on the edge of an IDEF0 arrow actually represents a set of possible items that can become an input, control, or output of the relevant function. All of the possible inputs and controls entering a function must then be represented by *n*-tuple of the Cartesian product across all input and control arrows entering that function. Similarly, the Cartesian product represents all possible outputs of a function across all output arrows exiting a function. So, there are, in fact, many important differences between a digraph and a page of an IDEF0 diagram.

A number of people have attempted to transform an IDEF0 model into a bipartite graph. The first step is to turn the arc labels into nodes of a second type, say circles. The IDEF0 diagram (without mechanisms) in the top of Figure 5.18 is converted into a bipartite graph in the bottom of Figure 5.18. Each label is replaced by a circular node. Each external label is connected by the edge entering or leaving the appropriate function. The new nodes for I12 and C12 are now connected by two edges; one going into the new node and one

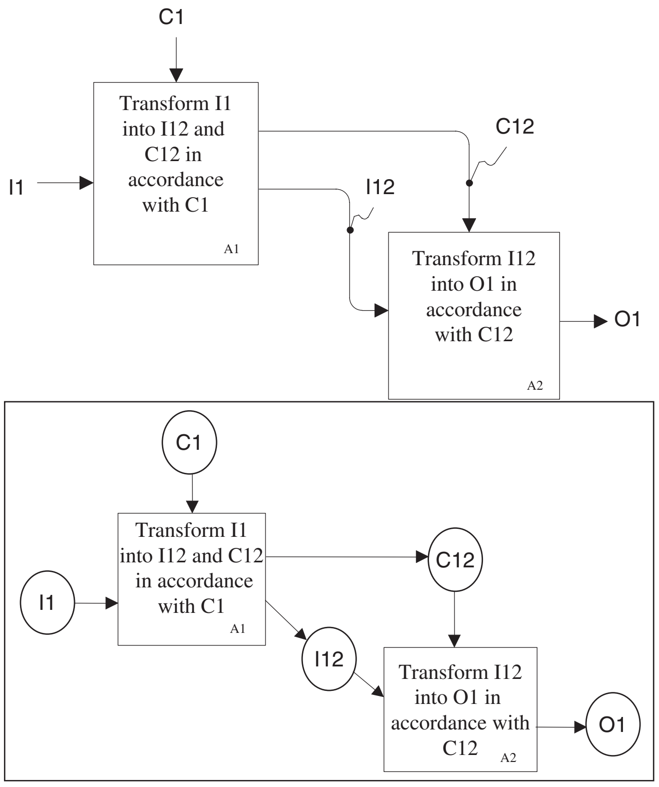

C1
Transform I1 into I12 and C12 in accordance with C1
A1
I1
C12
I12
Transform I12 into O1 in accordance with C12
A2
O1
C1
Transform I1 into I12 and C12 in accordance with C1
A1
C12
I1
I12
Transform I12 into O1 in accordance with C12
A2
O1

FIGURE 5.18 ICOM labels converted to nodes.

coming out of the new node. We have now satisfied the basic requirements of a bipartite graph; there are two types of nodes and no edge connects two nodes of the same type. There are, in essence, two types of edges; those that connect boxes to circles (outputs of the function in the box) and those that connect circles to boxes (inputs to the function in the box).

However, there are two remaining problems. First, IDEF0 differentiates between arcs entering a function from the top and left. There is no provision for such differentiation in digraphs. Other process modeling techniques in Chapter 12 do not differentiate between inputs and controls; it is necessary to drop this distinction between inputs and controls, as is done in Petri nets, which is the only graph-theoretic modeling tool discussed in Chapter 12.

Second, there is a problem with branches and joins. There is no analogous construct in graph theory. To solve this problem a function must be inserted at each branch to accomplish a divide or copy, and at each join to accomplish a paste. See Figure 5.19.

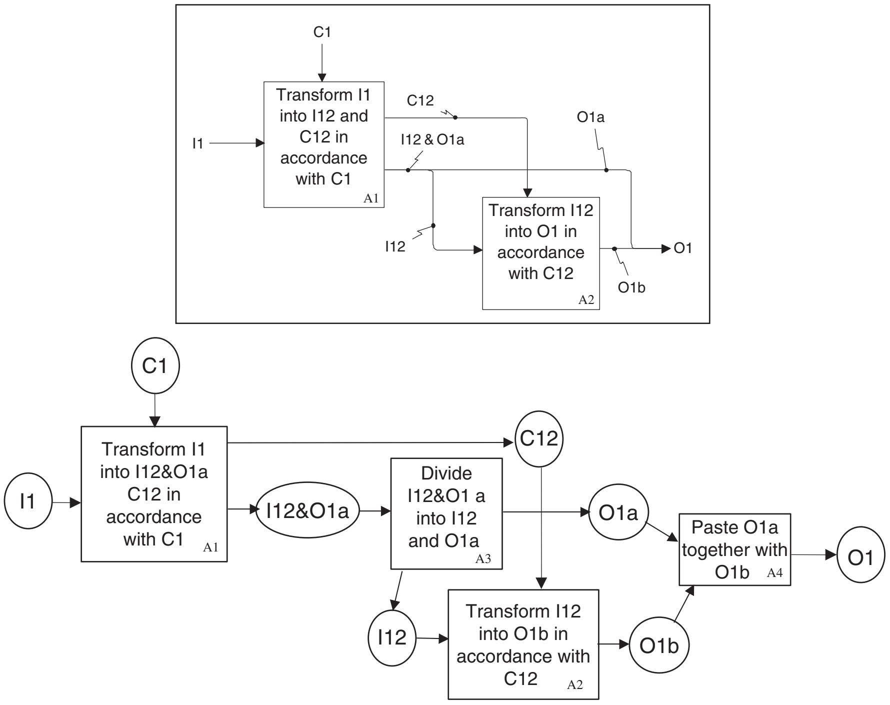

C1
Transform I1 into I12 and C12 in accordance with C1
A1
I1
C12
I12 & O1a
O1a
Transform I12 into O1 in accordance with C12
A2
I12
O1
O1b
C1
Transform I1 into I12&O1a C12 in accordance with C1
A1
I1
C12
I12&O1a
Divide I12&O1 a into I12 and O1a
A3
O1a
Paste O1a together with O1b
A4
O1
I12
Transform I12 into O1b in accordance with C12
A2
O1b

FIGURE 5.19 IDEF0 page with divide and paste functions added.

With all of these workarounds, IDEF0 remains a static snapshot of a dynamic process. There are potentially infinite dynamic models that can be created from each IDEF0 model. The information that separates the proper dynamic model from the rest of the possible dynamic models is not in the IDEF0 model but remains in the mental model of the creator of the IDEF0 model. If a team (which is most common) creates the IDEF0 model, it is possible, even likely, that each team member has a mental model of a different dynamic representation of the static IDEF0 model. This is why creating a dynamic model from the IDEF0 representation is so important; the communication process among the systems engineering team must be carried as far as possible.

5.12 SUMMARY

A graph consists of a set of nodes and a set of edges. The edges define a relation over the set of nodes. The relation can require an order of the nodes in which case the edges are directed; directed graphs are the most applied in the engineering of systems. Bipartite graphs are a special form of a directed graph in which there are two types of nodes, and the edges cannot connect nodes that are the same type.

Sequences of nodes in a graph can be defined by the terms walk, path, trail, circuit, and cycle. Graphs can be connected or disconnected; there are variations of connectedness, ranging from weakly to strongly. Nodes that are not adjacent to each other in a graph can be reachable via a path in the graph. This notion of reachability can be critical if attaining some output requires the execution of a set of functions, but the set of functions is not part of a reachable set.

The properties of reflexivity, irreflexivity, symmetry, antisymmetry, asymmetry, transitivity, and intransitivity were defined in Chapter 4 and then redefined in terms of graphs in this chapter. Visualizing these relations provides a much greater understanding of their meaning and ability to detect their absence or presence in a graph.

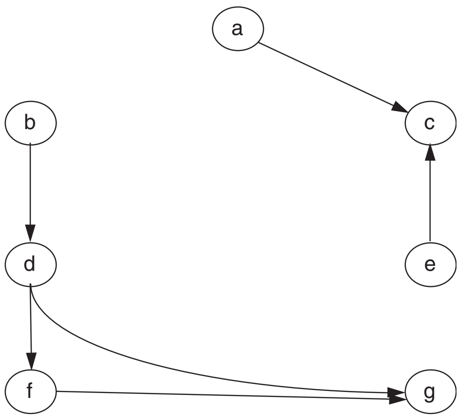
Partial orders of the elements of a set were defined as alternative orders of the nodes based upon the relation defined over the nodes. The Hasse diagram was defined and illustrated for finding the partial order on the set and then enumerating the possible partial orders.

Trees and several variations of trees were introduced as a special form of a graph. A directed tree describes the notion of a hierarchical decomposition. Hierarchies of requirements, functions, and components were discussed in Chapter 2 and will be revisited in Chapters 6 through 11. These hierarchies must be partitions (as defined in Chapter 4) and can be represented as directed trees.

Finally the IDEF0 process modeling technique was revisited and discussed in terms of mathematical graph theory. The reasons why an IDEF0 model is not a directed graph were discussed, as well as the difficulty associated with turning an IDEF0 model into a graph.

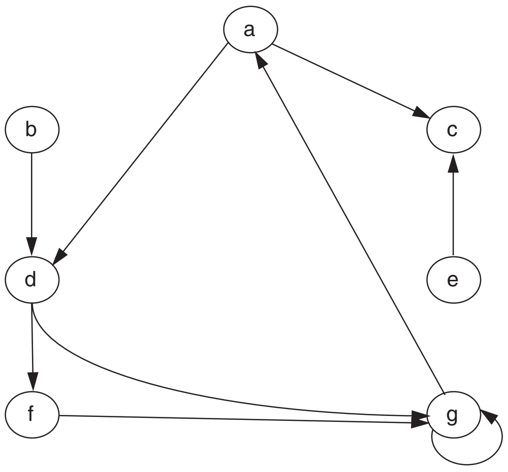
PROBLEMS

5.1 For the following graph, G_1:



a. Find $|V(G_1)|$ and $|E(G_1)|$.
b. Write the relation depicted by G_1 as a set of ordered pairs.
c. Define the adjacency matrix of G_1.
d. What is the out degree of each node of G_1? What is the in degree of each node of G_1?
e. Could G_1 be a bipartite graph? If no, why? If yes, what is the partition into two subsets of nodes that makes this a bipartite graph?
f. Is the relation depicted here reflexive? irreflexive? symmetric? antisymmetric? asymmetric? transitive? intransitive?
g. What arcs (if any) would you have to add to this relation to make it transitive

5.2 For the following graph, G_2:



a. Write the relation depicted by G_2 as a set of ordered pairs.
b. Define the adjacency matrix of G_2.
c. Could G_2 be a bipartite graph? If no, why? If yes, what is the partition into two subsets of nodes that makes this a bipartite graph?
d. *Is there a cycle in G_2? How many?
e. *Is there a semicycle in G_2? Which nodes are included?
f. Is the relation depicted here reflexive? irreflexive? symmetric? antisymmetric? asymmetric? transitive? intransitive?
g. What arcs (if any) would you have to add to this relation to make it transitive?
h. *Delete the arc from g to a and draw a Hasse diagram for G_2. Why must we delete the arc from g to a before we can draw a Hasse

* Advanced assignment.

diagram? Define at least 10 different node orderings consistent with this Hasse diagram.

5.3

a. Develop a directed graph for the relation "_____ has defeated ____." using the following won/lost records of the two 1993 Super Bowl teams. Create a single node for each team and an arc for each defeat. Note this will be a multigraph.

Buffalo Bills (BB) Schedule				Dallas Cowboys (DC) Schedule			
BB	38	NEP	14	DC	16	WR	35
		BB	13	DC	10		
BB	13	MD	22	DC	17	PC	10
BB	17	NYG	14	DC	36	GBP	14
BB	35	HO	7	DC	27	IC	3
BB	19	NYJ	10	DC	26	SFF	17
BB	24	WR	10	DC	23	PE	10
BB	0	PS	23	DC	20	PC	15
BB	13	NEP	10	DC	31	NYG	9
BB	23	IC	9	DC	14	AF	27
BB	7	KCC	23	DC	14	MD	16
BB	24	LAR	25	DC	23	PE	17
BB	10	PE	7	DC	37	MV	20
BB	47	MD	34	DC	28	NYJ	7
BB	16	NYJ	14	DC	38	WR	3
BB	30	IC	10	DC	16	NYG	13
BB	29	LAR	23	DC	27	GBP	17
BB	30	KCC	13	DC	38	SFF	21
	SUPER	BB	13	DC	30		

b. Is this directed graph reflexive? irreflexive? transitive? asymmetric?

c. *There will be cycles in the graph created in part (a). Break these cycles by eliminating arcs in favor of the two Super Bowl teams; that is, if there is a cycle between a Super Bowl team and another team, eliminate the arc showing that the Super Bowl team was defeated by

* Advanced assignment.

the other team. Assume the resulting relation is a partial order and draw a Hasse diagram of the relation.

5.4 For the following adjacency matrix:

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

a. Draw the graphical representation, G_4, that is defined by the adjacency matrix.
b. Find $|V(G_4)|$ and $|E(G_4)|$.
c. Write the relation depicted by G_4 as a set of ordered pairs.
d. What is the out degree of each node of G_4? What is the in degree each node of G_4?
e. Could G_4 be a bipartite graph? If no, why? If yes, what is the partition into two subsets of the nodes that makes G_4 a bipartite graph?
f. Which of the seven properties (reflexive, irreflexive, transitive, intransitive, symmetric, asymmetric, antisymmetric) does this relation satisfy

5.5 *Drop the arc from *b* to *c* in Figure 5.15 and draw a Hasse diagram for the resulting graph. How many orderings of the nodes in the digraph are consistent with this Hasse diagram?

5.6 There are three families defined by the sets *A*, *B,* and *C*; each family has a dad, mom, and three kids:

A = {Dad, Mom, Doris, Bill, Tom}
B = {Dad, Mom, Doris, Daisy, Debbie}
C = {Dad, Mom, Bill, Bob, Biff}

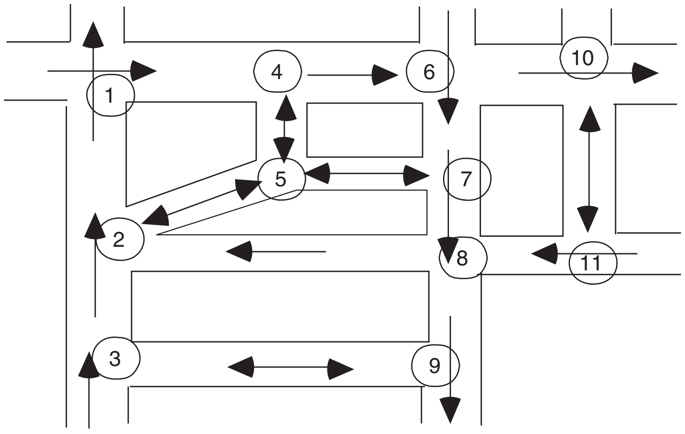
Consider the relations "is the spouse of," "is the brother of," and "is the blood relative of." (Hints: I am not the brother of myself. Two people are blood relatives if they share the blood of a common ancestor, who may or may not be part of sets *A*, *B*, or *C*. I am the blood relative of myself.)

* Advanced assignment.

Create a digraph for each of the three relations on each of the three sets. Identify which of these relations satisfy which of the seven properties of unary relations for each of the three sets by placing a yes or no in the empty cells of the following table.

	Reflexive	Irreflexive	Symmetric	Anti-symmetric	Asymmetric	Transitive	Intransitive
"is the spouse of" on *A*							
"is the brother of" on *A*							
"is the blood relative of" on *A*							
"is the spouse of" on *B*							
"is the brother of" on *B*							
"is the blood relative of" on *B*							
"is the spouse of" on *C*							
"is the brother of" on *C*							
"is the blood relative of" on *C*							

5.7 A city street snapshot is shown in the figure. Note there are streets with arcs on them indicating one-way streets. The streets with double-headed arcs are two-way streets. There are 11 intersections, labeled 1 through 11.



a. Draw a directed graph that represents this street system. (Hint: Use a node to represent street intersections.)

b. Is this digraph quasi-strongly connected? If not, what is the minimum number of arcs that must be added and what nodes must they connect to make it quasi-strongly connected? If yes, why?

c. If you think the digraph in part (a) is quasi-strongly connected, draw a directed spanning tree for it. If you do not think the digraph in part (a) is quasi-strongly connected, add arcs so that it is and then draw a directed spanning tree for it.

d. What is the height of the tree that you have drawn?

5.8 For the set of all possible relations, create a partition using combinations of the properties symmetric, antisymmetric, and asymmetric where each subset in the partition cannot be empty. As an example, a partition of all relations using the properties reflexive and irreflexive would be: (reflexive relations), (irreflexive relations), (relations that are neither reflexive nor irreflexive). Note the subset of relations that are both reflexive and irreflexive is left out because this combination is impossible.

5.9 Consider an IDEF0 model in which the function A0 has two inputs (I_1 and I_2), three controls (C_1, C_2, and C_3) and three outputs (O_1, O_2, and O_3). The IDEF0 function, A0, can be considered a relation that maps elements of $\Delta = (I_1 \text{ X } I_2 \text{ X } C_1 \text{ X } C_2 \text{ X } C_3)$ into elements of $\Pi = (O_1 \text{ X } O_2 \text{ X } O_3)$. The 5-tuple for inputs and controls to A0 and the 3-tuple for outputs are used because each input, control, and output represents a set of possible inputs, controls, or outputs, respectively. The n-tuples define all possible combinations of inputs and outputs, respectively. Under what restrictions is A0 a function? Why?