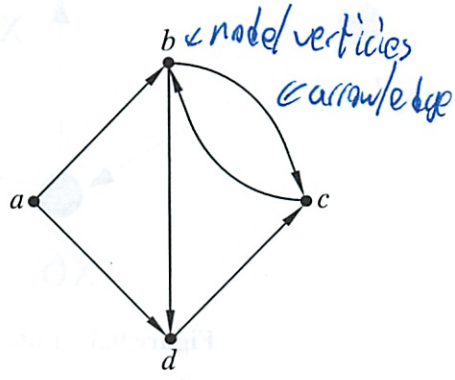
9 Directed graphs & Partial Orders

digraph = directed graph (w/ arrows 1 way)

Directed graphs, called *digraphs* for short, provide a handy way to represent how things are connected together and how to get from one thing to another by following the connections. They are usually pictured as a bunch of dots or circles with arrows between some of the dots as in Figure 9.1. The dots are called *nodes* (or *vertices*), and the lines are called *directed edges* or *arrows*, so the digraph in Figure 9.1 has 4 nodes and 6 directed edges.

Digraphs appear everywhere in computer science. In Chapter 10, we'll use digraphs are used to describe communication nets for routing data packets. The digraph in Figure 9.2 has three "in" nodes (pictured as little squares) representing locations where packets may arrive at the net, the three "out" nodes representing destination locations for packets, and the remaining six nodes (pictured with little circles) represent switches. The 16 edges indicate paths that packets can take through the router.

Another digraph example, is the hyperlink structure of the World Wide Web. Letting the vertices $x_1, \ldots, x_n$ correspond to web pages and using arrows to indicate when one page has a hyperlink to another, yields a digraph like the one Figure 9.3. In the graph of the real World Wide Web, n would be a number in the billions and probably even the trillions. At first glance, this graph wouldn't seem to be very interesting. But in 1995, two students at Stanford, Larry Page and Sergey Brin ultimately became multibillionaires from the realization of how useful the structure of this graph could be in building a search engine. So pay attention to graph theory, and who knows what might happen!



b < node/vertices
< arrow/edge

Figure 9.1 A 4-node directed graph with 6 edges.

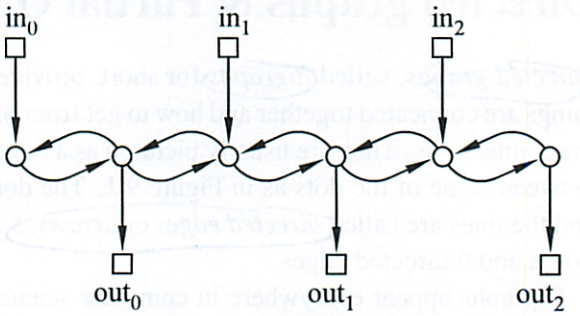


Figure 9.2 A 6-switch packet routing digraph.

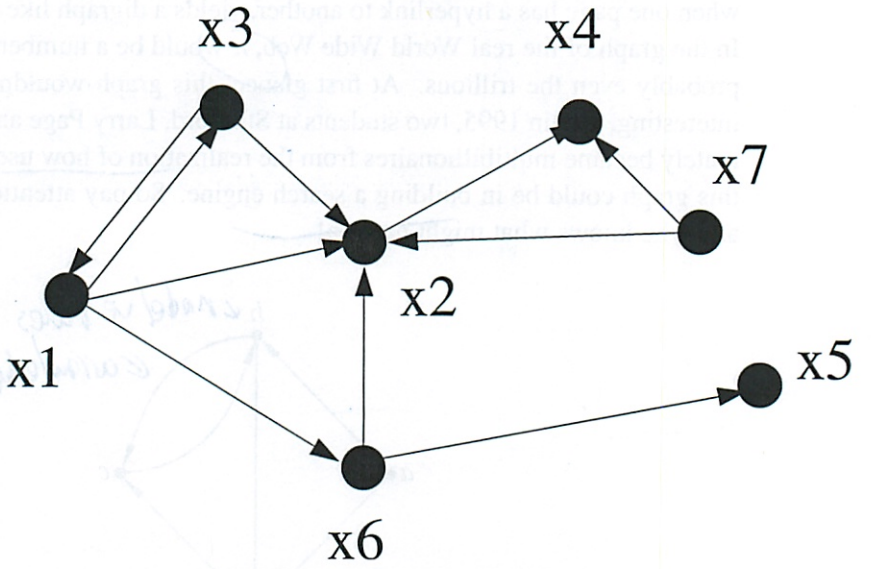


Figure 9.3 Links among Web Pages

Google Page Rank

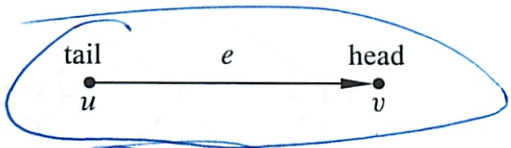


Figure 9.4 A directed edge $e = \langle u \to v \rangle$. The edge e starts at the tail vertex, u, and ends at the head vertex, v.

9.1 Vertex Degrees

The *in-degree* of a vertex in a digraph is the number of arrows coming into it and similarly its *out-degree* is the number of arrows out of it. More precisely,

Definition 9.1.1. If G is a digraph and $v \in V(G)$, then

$$\text{indeg}(v) ::= |\{e \in E(G) \mid \text{head}(e) = v\}|$$
$$\text{outdeg}(v) ::= |\{e \in E(G) \mid \text{tail}(e) = v\}|$$

An immediate consequence of this definition is

Lemma 9.1.2.

$$\sum_{v \in V(G)} \text{indeg}(v) = \sum_{v \in V(G)} \text{outdeg}(v).$$

Proof. Both sums are obviously equal to $|E(G)|$. ■

9.2 Digraph Walks and Paths

Definition 9.2.1. A *directed graph*, G, consists of a nonempty set, $V(G)$, called the *vertices* of G, and a set, $E(G)$, called the *edges* of G. An element of $V(G)$ is called a *vertex*. A vertex is also called a *node*; the words "vertex" and "node" are used interchangeably. An element of $E(G)$ is called a *directed edge*. A directed edge is also called an "arrow" or simply an "edge." A directed edge *starts* at some vertex, u, called the *tail* of the edge, and *ends* at some vertex, v, called the *head* of the edge, as in Figure 9.4. Such an edge can be represented by the ordered pair (u, v). The notation $\langle u \to v \rangle$ denotes this edge.

There is nothing new in Definition 9.2.1 except for a lot of vocabulary. Formally, a digraph G is the same as a binary relation on the set $V = V(G)$—that is, a

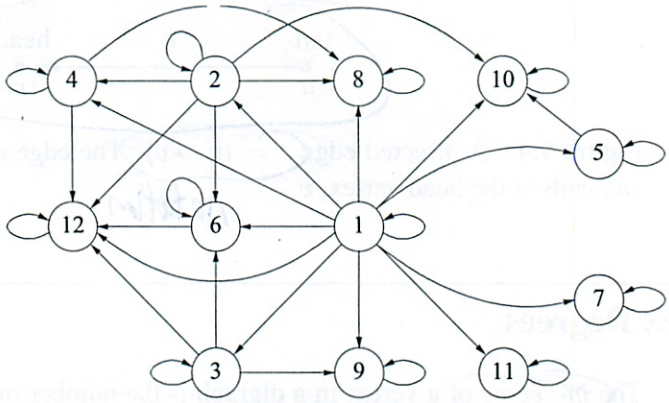


Figure 9.5 The Digraph for Divisibility on $\{1, 2, \ldots, 12\}$.

digraph is just a binary relation whose domain and codomain are the same set, V. In fact we've already referred to the arrows in a relation G as the "graph" of G. For example, the divisibility relation on the integers in the interval $[1, 12]$ could be pictured by the digraph in Figure 9.5.

Picturing digraphs with points and arrows makes it natural to talk about following successive edges through the graph. For example, in the digraph of Figure 9.5, you might start at vertex 1, successively follow the edges from vertex 1 to vertex 2, from 2 to 4, from 4 to 12, and then from 12 to 12 twice (or as many times as you like). The sequence of edges followed in this way is called a *walk* through the graph.

The obvious way to represent a walk is with the sequence of sucessive vertices it went through, in this case:

$$1\ 2\ 4\ 12\ 12\ 12.$$

However, it is conventional to represent a walk by an alternating sequence of successive vertices and edges, so this walk would formally be

$$1\ \langle 1 \to 2\rangle\ 2\ \langle 2 \to 4\rangle\ 4\ \langle 4 \to 12\rangle\ 12\ \langle 12 \to 12\rangle\ 12\ \langle 12 \to 12\rangle\ 12. \tag{9.1}$$

The redundancy of this definition is enough to make any computer scientist cringe, but it does make it easy to talk about how many times vertices and edges occur on the walk. Here is a formal definition:

Definition 9.2.2. A *walk in a digraph*, G, is an alternating sequence of vertices and edges that begins with a vertex, ends with a vertex, and such that for every edge $\langle u \to v\rangle$ in the walk, vertex u is the element just before the edge, and vertex v is the next element after the edge.

So a walk, $\mathbf{v}$, is a sequence of the form

$$\mathbf{v} ::= v_0\ \langle v_0 \to v_1\rangle\ v_1\ \langle v_1 \to v_2\rangle\ v_2\ \ldots\ \langle v_{k-1} \to v_k\rangle\ v_k$$

where $\langle v_i \to v_{i+1}\rangle \in V(G)$ for $i \in [0, k)$. The walk is said to *start* at v_0, to ~~*end* at~~ v_k, and the *length*, $|\mathbf{v}|$, of the walk is defined to be k. The walk is a *path* iff all the v_i's are different, that is, if $i \neq j$, then $v_i \neq v_j$.

Note that a single vertex counts as length zero path that begins and ends at itself.

If you walk for a while, stop for a rest at some vertex, and then continue walking, you have broken a walk into two parts. For example, stopping to rest after following two edges in the walk (9.1) through the divisibility graph breaks the walk into the first part of the walk

$$1\ \langle 1 \to 2\rangle\ 2\ \langle 2 \to 4\rangle\ 4 \tag{9.2}$$

from 1 to 4, and the rest of the walk

$$4\ \langle 4 \to 12\rangle\ 12\ \langle 12 \to 12\rangle\ 12\ \langle 12 \to 12\rangle\ 12. \tag{9.3}$$

from 4 to 12, and we'll say the whole walk (9.1) is the *merge* of the walks (9.2) and (9.3). In general, if a walk $\mathbf{f}$ ends with a vertex, v, and a walk $\mathbf{r}$ starts with the same vertex, v, we'll say that their *merge*, $\mathbf{f}\,\hat{}\,\mathbf{r}$, is the walk that starts with $\mathbf{f}$ and continues with $\mathbf{r}$.[1] Two walks can only be merged if the first ends with the same vertex, v, that the second one starts with. Sometimes it's useful to name the node v where the walks merge; we'll use the notation $\mathbf{f}\,\hat{v}\,\mathbf{r}$ to describe the merge of a walk $\mathbf{f}$ that ends at v with a walk $\mathbf{r}$ that begins at v.

A consequence of this definition is that

Lemma 9.2.3.

$$|\mathbf{f}\,\hat{}\,\mathbf{r}| = |\mathbf{f}| + |\mathbf{r}|.$$

In the next section we'll get mileage out of walking this way.

9.2.1 Finding a Path

If you were trying to walk somewhere quickly, you'd know you were in trouble if you came to the same place twice. This is actually a basic theorem of graph theory.

Theorem 9.2.4. *The shortest walk between a pair of vertices is a path.*

[1] It's tempting say the *merge* is the concatenation of the two walks, but that wouldn't quite be right because if the walks were concatenated, the vertex v would appear twice in a row where the walks meet.

Proof. If there is a walk from vertex u to v, there must, by the Well Ordering Principle, be a minimum length walk $\mathbf{w}$ from u to v. We claim $\mathbf{w}$ is a path.

To prove the claim, suppose to the contrary that $\mathbf{w}$ is not a path, namely, some vertex x occurs twice on this walk. That is,

$$\mathbf{w} = \mathbf{e}\,\widehat{x}\,\mathbf{f}\,\widehat{x}\,\mathbf{g}$$

for some walks $\mathbf{e}, \mathbf{f}, \mathbf{g}$ where the length of $\mathbf{f}$ is positive. But then deleting $\mathbf{f}$ yields is a strictly shorter walk

$$\mathbf{e}\,\widehat{x}\,\mathbf{g}$$

from u to v, contradicting the minimality of $\mathbf{w}$. ■

Definition 9.2.5. The *distance* dist (u, v), in a graph from vertex u to vertex v is the length of a shortest path from u to v.

As would be expected, this definition of distance satisfies:

Lemma 9.2.6. *[The Triangle Inequality]*

$$dist\,(u, v) \leq dist\,(u, x) + dist\,(x, v)$$

for all vertices u, v, x with equality holding iff x is on a shortest path from u to v.

Of course you may expect this property to be true, but distance has a technical definition and its properties can't be taken for granted. For example, unlike ordinary distance in space, the distance from u to v is typically different from the the distance from v to u. So let's prove the Triangle Inequality:

Proof. To prove the inequality, suppose $\mathbf{f}$ is a shortest path from u to x and $\mathbf{r}$ is a shortest path from x to v. Then by Lemma 9.2.3, $\mathbf{f}\,\widehat{x}\,\mathbf{r}$ is a path of length dist (u, x) + dist (x, v) from u to v, so this sum is an upper bound on the length of the shortest path from u to v.

To prove the "iff" from left to right, suppose dist (u, v) = dist (u, x)+dist (x, v). Then taking a shortest path from u to x followed by a shortest path from x to w yields a path of whose length is dist (u, x)+dist (x, v) which by assumption equals dist (u, v). So this is a shortest path containing x.

To prove the "iff" from right to left, suppose vertex x is on a shortest path $\mathbf{w}$ from u to v, namely, $\mathbf{w}$ is a shortest path of the form $\mathbf{f}\,\widehat{x}\,\mathbf{r}$. The path $\mathbf{f}$ must be a shortest path from u to x; otherwise replacing $\mathbf{f}$ by a shorter path from u to x would yield a shorter path from u to v than $\mathbf{w}$. Likewise from $\mathbf{r}$ must be a shortest path from x to v. So dist $(u, v) = |\mathbf{w}| = |\mathbf{f}| + |\mathbf{r}|$ = dist (u, x) + dist (x, v). ■

9.3 Adjacency Matrices

If a graph, G, has n vertices, $v_0, v_1, \ldots, v_{n-1}$, a useful way to represent it is with a $n \times n$ matrix of zeroes and ones called its *adjacency matrix*, A_G. The ijth entry, $(A_G)_{ij}$, of the adjacency matrix is 1 if there is an edge from vertex v_i to vertex v_j, and 0 otherwise. That is,

$$(A_G)_{ij} ::= \begin{cases} 1 & \text{if } \langle v_i \to v_j \rangle \in V(G), \\ 0 & \text{otherwise.} \end{cases}$$

For example, let H be the 4-node the graph shown in Figure 9.1. Then its adjacency matrix A_H is the 4×4 matrix:

$$A_H = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 1 & 0 & 1 \\ b & 0 & 0 & 1 & 1 \\ c & 0 & 1 & 0 & 0 \\ d & 0 & 0 & 1 & 0 \end{array}$$

A payoff of this representation is that we can use matrix powers to count numbers of walks between vertices. For example, there are two length-2 walks between verices a and c in the graph H, namely

$$a \ \langle a \to b \rangle \ b \ \langle b \to c \rangle \ c$$
$$a \ \langle a \to d \rangle \ d \ \langle d \to c \rangle \ c$$

and these are the only length-2 walks from a to c. Also, there is exactly one length-2 walk from b to c and exactly one length-2 walk from c to c and from d to b, and these are the only length-2 walks in H. It turns out we could have read these counts from the entries in the matrix $(A_H)^2$:

$$(A_H)^2 = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 0 & 2 & 0 \\ b & 0 & 0 & 1 & 0 \\ c & 0 & 0 & 1 & 0 \\ d & 0 & 1 & 0 & 0 \end{array}$$

More generally, the matrix $(A_G)^k$ provides a count of the number of length k walks between vertices in any digraph, G, as we'll now explain.

Definition 9.3.1. The length-k *walk counting matrix* for an n-vertex graph G is the $n \times n$ matrix C such that

$$C_{uv} ::= \text{the number of length-}k\text{ walks from } u \text{ to } v. \tag{9.4}$$

Notice that the adjacency matrix A_G is the length-1 walk counting matrix for G, and that $(A_G)^0$, which by convention is the identity matrix, is the length-0 walk counting matrix.

Theorem 9.3.2. *If C is the length-k walk counting matrix for a graph G, and D is the length-m walk counting matrix, then CD is the length $k+m$ walk counting matrix for G.*

According to this theorem, the square $(A_G)^2$ of the adjacency matrix is the length-2 walk counting matrix for G. Applying the theorem again to $(A_G)^2 A_G$, shows that the length-3 walk counting matrix is $(A_G)^3$. More generally, it follows by induction that

Corollary 9.3.3. *The length-k counting matrix of a digraph, G, is $(A_G)^k$, for all $k \in \mathbb{N}$.*

In other words, you can determine the number of length k walks between any pair of vertices simply by computing the kth power of the adjacency matrix!

That may seem amazing, but the proof uncovers this simple relationship between matrix multiplication and numbers of walks.

Proof of Theorem 9.3.2. Any length-$(k+m)$ walk between vertices u and v begins with a length-k walk starting at u and ending at some vertex, w, followed by a length-m walk starting at w and ending at v. So the number of length-$(k+m)$ walks from u to v that go through w at the kth step equals the number C_{uw} of length-k walks from u to w, times the number D_{wv} of length-m walks from w to v. We can get the total number of length-$(k+m)$ walks from u to v by summing, over all possible vertices w, the number of such walks that go through w at the kth step. In other words,

$$\#\text{length-}(k+m)\text{ walks from } u \text{ to } v = \sum_{w \in V(G)} C_{uw} \cdot D_{wv} \tag{9.5}$$

But the right hand side of (9.5) is precisely the definition of $(CD)_{uv}$. Thus, CD is indeed the length-$(k+m)$ walk counting matrix. ∎

9.3.1 Shortest Paths

The relation between powers of the adjacency matrix and numbers of walks is cool (to us math nerds at least), but a much more important problem is finding shortest paths between pairs of nodes in a graph. For example, when you drive home for vacation, you generally want to take the shortest-time route.

One simple way to find the lengths of all the shortests paths in an n-vertix graph, G, is to compute the successive powers of A_G one by one up to the $n-1$st, watching for the first power at which each entry becomes positive. That's because Theorem 9.3.2 implies that the length of the shortest path, if any, between u and v, that is, the distance from u to v, will be the smallest value k for which $(A_G)^k_{uv}$ is nonzero, and if there is a shortest path, its length will be $\leq n-1$. Refinements of this idea lead to methods that find shortest paths in reasonably efficient ways. The methods apply as well to weighted graphs, where edges are labelled with weights or costs and the objective is to find least weight, cheapest paths. These refinements are typically covered in introductory algorithm courses, and we won't go into them here any further.

9.4 Path Relations

A basic question about a digraph is whether there is a path from one particular vertex to another. So for any digraph, G, we interested in a binary relation, G^*, called the *path relation* on $V(G)$ where

$$u\ G^*\ v ::= \text{there is a path in } G \text{ from } u \text{ to } v. \tag{9.6}$$

Similarly, there is a *positive path relation*

$$a\ G^+\ b ::= \text{there is a positive length path in } G \text{ from } u \text{ to } v. \tag{9.7}$$

Since merging a path from u to v with a path from v to w gives a path from u to w, both path relations have a relational property called *transitivity*:

Definition 9.4.1. A binary relation, R, on a set, A, is *transitive* iff

$$(a\ R\ b \text{ AND } b\ R\ c) \text{ IMPLIES } a\ R\ c$$

for every $a, b, c \in A$.

Since there is a length-0 path from any vertex to itself, the path relation has another relational property called *reflexivity*:

Definition 9.4.2. A binary relation, R, on a set, A, is *reflexive* iff $a\ R\ a$ for all $a \in A$.

9.4.1 Composition of Relations

There is a simple way to extend composition of functions to composition of relations, and this gives another way to talk about paths in digraphs.

Let $R : B \to C$ and $S : A \to B$ be binary relations. Then the composition of R with S is the binary relation $(R \circ S) : A \to C$ defined by the rule

$$a\ (R \circ S)\ c ::= \exists b \in B.\ (a\ S\ b) \text{ AND } (b\ R\ c). \tag{9.8}$$

This agrees with the Definition 4.3.1 of composition in the special case when R and S are functions.[2]

Remembering that a digraph is a binary relation on its vertices, it makes sense to compose a digraph G with itself. Then if we let G^n denote the composition of G with itself n times, it's easy to check (see Problem 9.4) that G^n is the *length-n path relation*:

$$a\ G^n\ b \quad \text{iff} \quad \text{there is a length-}n\text{ path in } G \text{ from } a \text{ to } b.$$

This even works for $n = 0$, with the usual convention that G^0 is the identity relation $\mathrm{Id}_{V(G)}$ on the set of vertices.[3] So now we have[4]

$$G^* = G^0 \cup G^1 \cup G^2 \cup \ldots \cup G^{|V(G)|-1} = (G \cup G^0)^{|V(G)|-1}. \tag{9.9}$$

The final equality points to the use of repeated squaring as a way to compute G^* with $\log n$ rather than $n - 1$ compositions of relations.

9.5 Directed Acyclic Graphs & Partial Orders

Definition 9.5.1. A *closed walk* in a digraph is a walk that begins and ends at the same vertex. A *cycle* in a digraph is a closed walk whose vertices are distinct except for the beginning and end vertices. This includes the path of length 0 that begins and ends at the same vertex. A *directed acyclic graph (DAG)* is a directed graph with no *positive* length cycles.

[2] The reversal of the order of R and S in (9.8) is not a typo.

[3] The identity relation, Id_A, on a set, A, is the equality relation:

$$a\ \mathrm{Id}_A\ b \quad \text{iff} \quad a = b.$$

[4] Equation (9.9) involves a harmless abuse of notation: we should have written

$$\mathrm{graph}(G^*) = \mathrm{graph}(G^0) \cup \mathrm{graph}(G^1) \ldots.$$

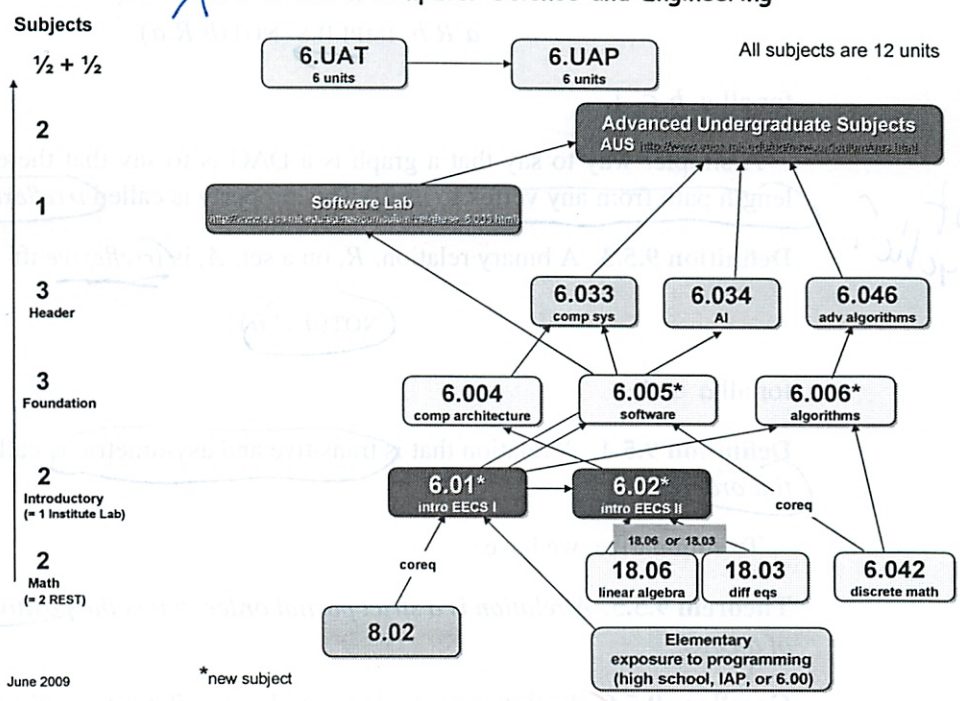


Figure 9.6 Subject prerequisites for MIT Computer Science (6-3) Majors

DAG's come up constantly because, among other things, they model task scheduling problems, where nodes represent tasks to be completed and arrows indicate which tasks must be completed before others can begin. For example, Figure 9.6 shows the prerequisite structure among MIT computer science subjects.

A positive length cycle in a prerequisite graph like this would have a dire effect on the time it takes to graduate.

The edges in subject prerequisite DAG of Figure 9.6 show the *direct* prerequisites listed for each subject, but to enroll for a subject you must of course have taken the prerequisites of the prerequisites and their prerequisites, and so on. In other words, if D is the direct prerequisite relation, then subject u has to be completed before taking subject v iff $u\ D^+\ v$.

The condition that D is a DAG is equivalent to saying that if there is a positive length path from vertex u to vertex v, there can't be one from v back to u. This relational property is called *asymmetry*.

Definition 9.5.2. A binary relation, R, on a set, A, is *asymmetric* iff

$$a\ R\ b \text{ IMPLIES } \text{NOT}(b\ R\ a)$$

for all $a, b \in A$.

A simpler way to say that a graph is a DAG is to say that there is no positive length path from any vertex to itself. This property is called *irreflexivity*.

Definition 9.5.3. A binary relation, R, on a set, A, is *irreflexive* iff

$$\text{NOT}(a\ R\ a)$$

for all $a \in A$.

Definition 9.5.4. A relation that is transitive and asymmetric is called a *strict partial order*.

To summarize, we have

Theorem 9.5.5. *A relation is a strict partial order iff it is the positive path relation of a DAG.*

Corollary 9.5.6. *A relation is a strict partial order iff it is transitive and irreflexive.*

A strict partial may be the positive path relation of different DAG's. This raises the question of finding a DAG with the *smallest* number of edges that determines a given strict partial order. For *finite* strict partial orders, the smallest such DAG turns out to be unique and easy to find (see Problem 9.2).

9.6 Weak Partial Orders

Partial orders come up in many situations which on the face of it have nothing to do with digraphs. For example, the less-than order, $<$, on numbers is a partial order:

- if $x < y$ and $y < z$ then $x < z$, so less-than is transitive, and
- if $x < y$ than $y \not< x$ so less-than is asymmetric.

The proper containment relation $\subset$ is also a partial order:

- if $A \subset B$ and $B \subset C$ then $A \subset C$, so containment is transitive, and

- $A \not\subset A$, so proper containment is irreflexive!

Partial orders have particular importance in computer science because, besides modeling task scheduling problems, they capture key concepts used, for example, in analyzing concurrency control, as illustrated in Section 9.10.

The less-than-or-equal relation, $\leq$, is at least as familiar as the less-than strict partial order, and the ordinary containment relation, $\subseteq$, is even more common than the proper containment relation. These are examples of *weak partial orders*.

Definition 9.6.1. A relation R on a set, A, is a weak partial order iff there a there is a strict partial order, S on A such that

$$a \; R \; b \quad \text{iff} \quad (a \; S \; b \text{ OR } a = b),$$

for all $a, b \in A$.

Weak partial orders can also be defined in terms of relational properties. We just have to relax the asymmetry property to allow each element to be related to itself; this property is called *antisymmetry*.

Definition 9.6.2. A binary relation, R, on a set A, is *antisymmetric* iff

$$a \; R \; b \text{ IMPLIES NOT}(b \; R \; a)$$

for all $a \neq b \in A$.

A relation is *weak partial order*[5] iff it is transitive, reflexive, and antisymmetric.

For weak partial orders in general, we often write an ordering-style symbol like $\preceq$ or $\sqsubseteq$ instead of a letter symbol like R.[6] Likewise, we generally use $\prec$ or $\sqsubset$ to indicate a strict partial order.

Two more examples of partial orders are worth mentioning:

Example 9.6.3. Let A be some family of sets and define $a \; R \; b$ iff $a \supset b$. Then R is a strict partial order.

For integers, m, n we write $m \mid n$ to mean that m *divides* n, namely, there is an integer, k, such that $n = km$.

Example 9.6.4. The divides relation is a weak partial order on the nonnegative integers.

[5] Some authors define partial orders to be what we call weak partial orders, but we'll use the phrase "partial order" to mean either a weak or strict one.

[6] General relations are usually denoted by a letter like R instead of a cryptic squiggly symbol, so $\preceq$ is kind of like the musical performer/composer Prince, who redefined the spelling of his name to be his own squiggly symbol. A few years ago he gave up and went back to the spelling "Prince."

9.7 Representing Partial Orders by Set Containment

Axioms can be a great way to abstract and reason about important properties of objects, but it helps to have a clear picture of the things that satisfy the axioms. We'll show that every partial order can be pictured as a collection of sets related by containment. That is, every partial order has the "same shape" as such a collection. The technical word for "same shape" is "isomorphic."

Definition 9.7.1. A binary relation, R, on a set, A, is *isomorphic* to a relation, S, on a set D iff there is a relation-preserving bijection from A to D. That is, there is bijection $f : A \to D$, such that for all $a, a' \in A$,

$$a \; R \; a' \quad \text{iff} \quad f(a) \; S \; f(a').$$

To picture a partial order, $\preceq$, on a set, A, as a collection of sets, we simply represent each element A by the set of elements that are $\preceq$ to that element, that is,

$$a \longleftrightarrow \{b \in A \mid b \preceq a\}.$$

For example, if $\preceq$ is the divisibility relation on the set of integers, $\{1, 3, 4, 6, 8, 12\}$, then we represent each of these integers by the set of integers in A that divides it. So

$$\begin{aligned} 1 &\longleftrightarrow \{1\} \\ 3 &\longleftrightarrow \{1, 3\} \\ 4 &\longleftrightarrow \{1, 4\} \\ 6 &\longleftrightarrow \{1, 3, 6\} \\ 8 &\longleftrightarrow \{1, 4, 8\} \\ 12 &\longleftrightarrow \{1, 3, 4, 6, 12\} \end{aligned}$$

So, the fact that $3 \mid 12$ corresponds to the fact that $\{1, 3\} \subseteq \{1, 3, 4, 6, 12\}$.

In this way we have completely captured the weak partial order $\preceq$ by the subset relation on the corresponding sets. Formally, we have

Lemma 9.7.2. *Let $\preceq$ be a weak partial order on a set, A. Then $\preceq$ is isomorphic to the subset relation, $\subseteq$, on the collection of inverse images under the $\preceq$ relation of elements $a \in A$.*

We leave the proof to Problem 9.10. Essentially the same construction shows that strict partial orders can be represented by set under the proper subset relation, $\subset$. To summarize

Theorem 9.7.3. *Every weak partial order, $\preceq$, is isomorphic to the subset relation, $\subseteq$, on a collection of sets.*

Every strict partial order, $\prec$, is isomorphic to the proper subset relation, $\subset$, on a collection of sets.

I don't get this section!

9.8 Total Orders

The familiar order relations on numbers have an important additional property: given two different numbers, one will be bigger than the other. Partial orders with this property are said to be *total*[7] *orders*.

Definition 9.8.1. Let R be a binary relation on a set, A, and let a, b be elements of A. Then a and b are *comparable* with respect to R iff $[a\ R\ b$ OR $b\ R\ a]$. A partial order for which every two different elements are comparable is called a *total order*.

So $<$ and $\leq$ are total orders on $\mathbb{R}$. On the other hand, the subset relation is *not* total, since, for example, any two different finite sets of the same size will be incomparable under $\subseteq$. The prerequisite relation on Course 6 required subjects is also not total because, for example, neither 8.01 nor 6.042 is a prerequisite of the other.

9.9 Product Orders

Taking the product of two relations is a useful way to construct new relations from old ones.

Definition 9.9.1. The product, $R_1 \times R_2$, of relations R_1 and R_2 is defined to be the relation with

$$\begin{aligned} \text{domain}(R_1 \times R_2) &::= \text{domain}(R_1) \times \text{domain}(R_2), \\ \text{codomain}(R_1 \times R_2) &::= \text{codomain}(R_1) \times \text{codomain}(R_2), \\ (a_1, a_2)\,(R_1 \times R_2)\,(b_1, b_2) &\quad \text{iff} \quad [a_1\ R_1\ b_1 \text{ and } a_2\ R_2\ b_2]. \end{aligned}$$

Example 9.9.2. Define a relation, Y, on age-height pairs of being younger *and* shorter. This is the relation on the set of pairs (y, h) where y is a nonnegative

[7] "Total" is an overloaded term when talking about partial orders: being a total order is a much stronger condition than being a partial order that is a total relation. For example, any weak partial order such as $\subseteq$ is a total relation.

integer ≤ 2400 which we interpret as an age in months, and h is a nonnegative integer ≤ 120 describing height in inches. We define Y by the rule

$$(y_1, h_1)\, Y\, (y_2, h_2) \quad \text{iff} \quad y_1 \leq y_2 \text{ AND } h_1 \leq h_2.$$

That is, Y is the product of the $\leq$-relation on ages and the $\leq$-relation on heights.

It follows directly from the definitions that products preserve the properties of transitivity, reflexivity, irreflexivity, and antisymmetry, as shown in Problem 9.19. That is, if R_1 and R_2 both have one of these properties, then so does $R_1 \times R_2$. This implies that if R_1 and R_2 are both partial orders, then so is $R_1 \times R_2$.

On the other hand, the property of being a total order is not preserved. For example, the age-height relation Y is the product of two total orders, but it is not total: the age 240 months, height 68 inches pair, (240,68), and the pair (228,72) are incomparable under Y.

wait for class

9.10 Scheduling

Scheduling problems are a common source of partial orders: there is a set, A, of tasks and a set of constraints specifying that starting a certain task depends on other tasks being completed beforehand. We can picture the constraints by drawing labelled boxes corresponding to different tasks, with an arrow from one box to another if the first box corresponds to a task that must be completed before starting the second one.

For example, here is a drawing describing the order in which you could put on clothes. The tasks are the clothes to be put on, and the arrows indicate what should be put on directly before what.

When we have a partial order of tasks to be performed, it can be useful to have an order in which to perform all the tasks, one at a time, while respecting the dependency constraints. This amounts to finding a total order that is consistent with the partial order. This task of finding a total ordering that is consistent with a partial order is known as *topological sorting*.

"like I possible solution"

Definition 9.10.1. A *topological sort* of a partial order, $\prec$, on a set, A, is a total ordering, $\sqsubset$, on A such that

$$a \prec b \text{ IMPLIES } a \sqsubset b.$$

For example,

$$\text{shirt} \sqsubset \text{sweater} \sqsubset \text{underwear} \sqsubset \text{leftsock} \sqsubset \text{rightsock} \sqsubset \text{pants}$$
$$\sqsubset \text{leftshoe} \sqsubset \text{rightshoe} \sqsubset \text{belt} \sqsubset \text{jacket},$$

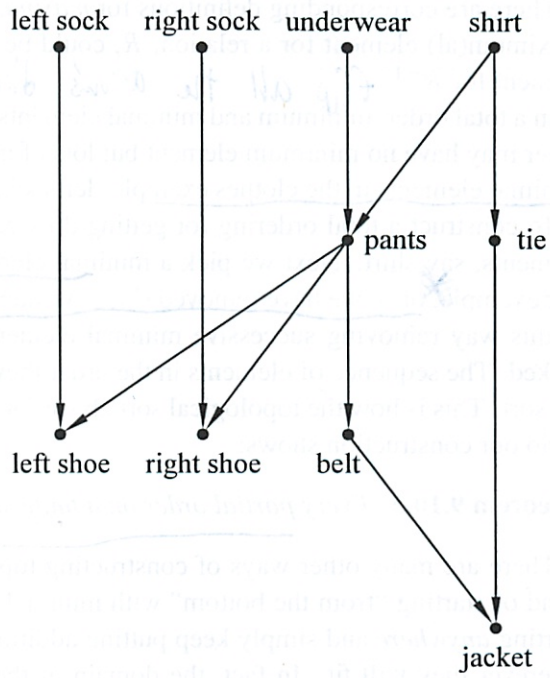


Figure 9.7 Partial order describing which clothing items have to be put on before others.

is one topological sort of the partial order of dressing tasks given by Figure 9.7; there are several other possible sorts as well.

Topological sorts for partial orders on finite sets are easy to construct by starting from *minimal* elements:

Definition 9.10.2. Let $\preceq$ be a partial order on a set, A. An element $a_0 \in A$ is *minimum* iff it is $\preceq$ every other element of A, that is, $a_0 \preceq b$ for all $b \neq a_0$.

The element a_0 is *minimal* iff no other element is $\preceq a_0$, that is, NOT$(b \preceq a_0)$ for all $b \neq a_0$.

There are corresponding definitions for *maximum* and *maximal*. Alternatively, a maximum(al) element for a relation, R, could be defined to be as a minimum(al) element for R^{-1}.

In a total order, minimum and minimal elements are the same thing. But a partial order may have no minimum element but lots of minimal elements. There are four minimal elements in the clothes example: leftsock, rightsock, underwear, and shirt.

To construct a total ordering for getting dressed, we pick one of these minimal elements, say shirt. Next we pick a minimal element among the remaining ones. For example, once we have removed shirt, sweater becomes minimal. We continue in this way removing successive minimal elements until all elements have been picked. The sequence of elements in the order they were picked will be a topological sort. This is how the topological sort above for getting dressed was constructed.

So our construction shows:

Theorem 9.10.3. *Every partial order on a finite set has a topological sort.*

There are many other ways of constructing topological sorts. For example, instead of starting "from the bottom" with minimal elements, we could build a total starting *anywhere* and simply keep putting additional elements into the total order wherever they will fit. In fact, the domain of the partial order need not even be finite: we won't prove it, but *all* partial orders, even infinite ones, have topological sorts.

9.10.1 Parallel Task Scheduling

For a partial order of task dependencies, topological sorting provides a way to execute tasks one after another while respecting the dependencies. But what if we have the ability to execute more than one task at the same time? For example, say tasks are programs, the partial order indicates data dependence, and we have a parallel machine with lots of processors instead of a sequential machine with only one. How should we schedule the tasks? Our goal should be to minimize the total *time*

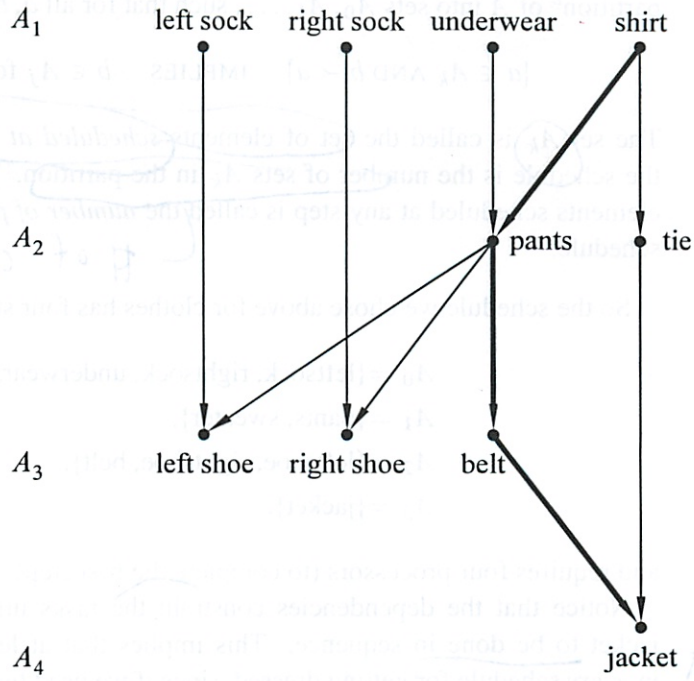


Figure 9.8 A parallel schedule for the tasks-in-getting-dressed poset in Figure 9.7. The tasks in A_i can be performed in step i for $1 \leq i \leq 4$. A chain of length 4 (the critical path in this example) is shown with bold edges.

to complete all the tasks. For simplicity, let's say all the tasks take the same amount of time and all the processors are identical.

So, given a finite partially ordered set of tasks, how long does it take to do them all, in an optimal parallel schedule? We can also use partial order concepts to analyze this problem.

In the clothes example, we could do all the minimal elements first (leftsock, rightsock, underwear, shirt), remove them and repeat. We'd need lots of hands, or maybe dressing servants. We can do pants and sweater next, and then leftshoe, rightshoe, and belt, and finally jacket. This schedule is illustrated in Figure 9.8.

In general, a *schedule* for performing tasks specifies which tasks to do at successive steps. Every task, a, has be scheduled at some step, and all the tasks that have to be completed before task a must be scheduled for an earlier step.

Definition 9.10.4. A *parallel schedule* for a strict partial order, $\prec$, on a set, A, is a

restrictions on what can't be done w/ when?
- like shared hard resources
- this ignores I believe

partition[8] of A into sets $A_0, A_1, \ldots$, such that for all $a, b \in A, k \in \mathbb{N}$,

$$[a \in A_k \text{ AND } b \prec a] \quad \text{IMPLIES} \quad b \in A_j \text{ for some } j < k.$$

The set A_k is called the set of elements *scheduled at step* k, and the *length* of the schedule is the number of sets A_k in the partition. The maximum number of elements scheduled at any step is called the *number of processors* required by the schedule.

So the schedule we chose above for clothes has four steps

$$\begin{aligned} A_0 &= \{\text{leftsock, rightsock, underwear, shirt}\}, \\ A_1 &= \{\text{pants, sweater}\}, \\ A_2 &= \{\text{leftshoe, rightshoe, belt}\}, \\ A_3 &= \{\text{jacket}\}. \end{aligned}$$

and requires four processors (to complete the first step).

Notice that the dependencies constrain the tasks underwear, pants, belt, and jacket to be done in sequence. This implies that at least four steps are needed in *every* schedule for getting dressed, since if we used fewer than four steps, two of these tasks would have to be scheduled at the same time. A set of tasks that must be done in sequence like this is called a *chain*.

Definition 9.10.5. A *chain* in a partial order is a set of elements such that any two different elements in the set are comparable. A chain is said to *end at* an its maximum element.

In general, the earliest step at which an element a can ever be scheduled must be at least as large as any chain that ends at a. A *largest* chain ending at a is called a *critical path* to a, and the size of the critical path is called the *depth* of a. So in any possible parallel schedule, it takes at least depth (a) steps to complete task a.

There is a very simple schedule that completes every task in this minimum number of steps. Just use a "greedy" strategy of performing tasks as soon as possible. Namely, schedule all the elements of depth k at step k. That's how we found the schedule for getting dressed given above.

[8] Partitioning a set, A, means "cutting it up" into non-overlapping, nonempty pieces. The pieces are called the blocks of the partition. More precisely, a *partition* of A is a set $\mathcal{B}$ whose elements are nonempty subsets of A such that

- if $B, B' \in \mathcal{B}$ are different sets, then $B \cap B' = \emptyset$, and
- $\bigcup_{B \in \mathcal{B}} B = A$.

Theorem 9.10.6. *Let $\prec$ be a strict partial order on a set, A. A minimum length schedule for $\prec$ consists of the sets $A_0, A_1, \ldots$, where*

$$A_k ::= \{a \mid \text{depth}(a) = k\}.$$

We'll leave to Problem 9.27 the proof that the sets A_k are a parallel schedule according to Definition 9.10.4.

The minimum number of steps needed to schedule a partial order, $\prec$, is called the *parallel time* required by $\prec$, and a largest possible chain in $\prec$ is called a *critical path* for $\prec$. So we can summarize the story above by this way: with an unlimited number of processors, the minimum parallel time to complete all tasks is simply the size of a critical path:

Corollary 9.10.7. *Parallel time = length of critical path.*

9.10.2 Dilworth's Lemma

Definition 9.10.8. An *antichain* in a partial order is a set of elements such that any two elements in the set are incomparable.

Our conclusions about scheduling also tell us something about antichains.

Corollary 9.10.9. *If the largest chain in a partial order on a set, A, is of size t, then A can be partitioned into t antichains.*

Proof. Let the antichains be the sets $A_k ::= \{a \mid \text{depth}(a) = k\}$. It is an easy exercise to verify that each A_k is an antichain (Problem 9.27) ■

Corollary 9.10.9 implies a famous result[9] about partially ordered sets:

Lemma 9.10.10 (Dilworth). *For all $t > 0$, every partially ordered set with n elements must have either a chain of size greater than t or an antichain of size at least n/t.*

Proof. Assume there is no chain of size greater than t, that is, the largest chain is of size $\leq t$. Then by Corollary 9.10.9, the n elements can be partitioned into at most t antichains. Let ℓ be the size of the largest antichain. Since every element belongs to exactly one antichain, and there are at most t antichains, there can't be more than ℓt elements, namely, $\ell t \geq n$. So there is an antichain with at least $\ell \geq n/t$ elements. ■

[9] Lemma 9.10.10 also follows from a more general result known as Dilworth's Theorem which we will not discuss.

Corollary 9.10.11. *Every partially ordered set with n elements has a chain of size greater than $\sqrt{n}$ or an antichain of size at least $\sqrt{n}$.*

Proof. Set $t = \sqrt{n}$ in Lemma 9.10.10. ■

Example 9.10.12. In the dressing partially ordered set, $n = 10$.

Try $t = 3$. There is a chain of size 4.

Try $t = 4$. There is no chain of size 5, but there is an antichain of size $4 \geq 10/4$.

Example 9.10.13. Suppose we have a class of 101 students. Then using the product partial order, Y, from Example 9.9.2, we can apply Dilworth's Lemma to conclude that there is a chain of 11 students who get taller as they get older, or an antichain of 11 students who get taller as they get younger, which makes for an amusing in-class demo.

9.11 Equivalence Relations

Skipping this semester

A relation is an *equivalence relation* if it is reflexive, symmetric, and transitive. Congruence modulo n is an excellent example of an equivalence relation:

exactly the same,

- It is reflexive because $x \equiv x \pmod{n}$.
- It is symmetric because $x \equiv y \pmod{n}$ implies $y \equiv x \pmod{n}$.
- It is transitive because $x \equiv y \pmod{n}$ and $y \equiv z \pmod{n}$ imply that $x \equiv z \pmod{n}$.

There is an even more well-known example of an equivalence relation: equality itself. Thus, an equivalence relation is a relation that shares some key properties with "=". wow (sarcasm)

9.11.1 Partitions

There is another way to think about equivalence relations, but we'll need a couple of definitions to understand this alternative perspective.

Definition 9.11.1. Given an equivalence relation $R : A \to A$, the *equivalence class* of an element $x \in A$ is the set of all elements of A related to x by R. The equivalence class of x is denoted $[x]_R$. Thus, in symbols:

$$[x]_R ::= \{y \mid x \, R \, y\}.$$

For example, suppose that $A = \mathbb{Z}$ and $x \; R \; y$ means that $x \equiv y \pmod 5$. Then

$$[7]_R = \{\ldots, -3, 2, 7, 12, 22, \ldots\}.$$

Notice that 7, 12, 17, etc., all have the same equivalence class; that is, $[7]_R = [12]_R = [17]_R = \cdots$.

Definition 9.11.2. A *partition* of a finite set A is a collection of disjoint, nonempty subsets $A_1, A_2, \ldots, A_n$ whose union is all of A. The subsets are usually called the *blocks* of the partition.[10] For example, one possible partition of $A = \{a, b, c, d, e\}$ is

$$A_1 = \{a, c\} \qquad A_2 = \{b, e\} \qquad A_3 = \{d\}.$$

Here's the connection between all this stuff: there is an exact correspondence between *equivalence relations on A* and *partitions of A*. We can state this as a theorem:

Theorem 9.11.3. *The equivalence classes of an equivalence relation on a set A form a partition of A.*

We won't prove this theorem (too dull even for us!), but let's look at an example. The congruent-mod-5 relation partitions the integers into five equivalence classes:

$$\{\ldots, -5, 0, 5, 10, 15, 20, \ldots\}$$
$$\{\ldots, -4, 1, 6, 11, 16, 21, \ldots\}$$
$$\{\ldots, -3, 2, 7, 12, 17, 22, \ldots\}$$
$$\{\ldots, -2, 3, 8, 13, 18, 23, \ldots\}$$
$$\{\ldots, -1, 4, 9, 14, 19, 24, \ldots\}$$

In these terms, $x \equiv y \pmod 5$ is equivalent to the assertion that x and y are both in the same block of this partition. For example, $6 \equiv 16 \pmod 5$, because they're both in the second block, but $2 \not\equiv 9 \pmod 5$ because 2 is in the third block while 9 is in the last block.

In social terms, if "likes" were an equivalence relation, then everyone would be partitioned into cliques of friends who all like each other and no one else.

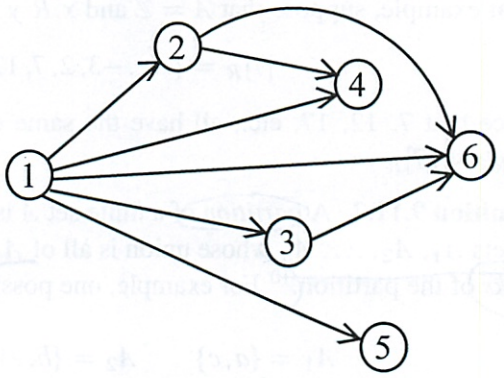
Problems for Section 9.5

Practice Problems

Problem 9.1.

Why is every strict partial order a DAG?

[10] We think they should be called the *parts* of the partition. Don't you think that makes a lot more sense?



Class Problems

Problem 9.2.

If a and b are distinct nodes of a digraph, then a is said to *cover* b if there is an edge from a to b and every path from a to b traverses this edge. If a covers b, the edge from a to b is called a *covering edge*.

(a) What are the covering edges in the following DAG?

(b) Let covering (D) be the subgraph of D consisting of only the covering edges. Suppose D is a finite DAG. Explain why covering (D) has the same positive path relation as D.

Hint: Consider *longest* paths between a pair of vertices.

(c) Show that if two *DAG*'s have the same positive path relation, then they have the same set of covering edges.

(d) Conclude that covering (D) is the *unique* DAG with the smallest number of edges among all digraphs with the same positive path relation as D.

The following examples show that the above results don't work in general for digraphs with cycles.

(e) Describe two graphs with vertices $\{1, 2\}$ which have the same set of covering edges, but not the same positive path relation (*Hint:* Self-loops.)

(f) (i) The *complete digraph* without self-loops on vertices $1, 2, 3$ has edges between every two distinct vertices. What are its covering edges?

(ii) What are the covering edges of the graph with vertices $1, 2, 3$ and edges $\langle 1 \to 2\rangle$, $\langle 2 \to 3\rangle$, $\langle 3 \to 1\rangle$?

(iii) What about their positive path relations?

Problem 9.3.

In a *round-robin* tournament, every pair of distinct players play against each other just once. For a round-robin tournament with with no tied games, a record of who beat whom can be described with a *tournament digraph*, where the vertices correspond to players and there is an edge $x \to y$ if x beat y in their game.

A *ranking* is a directed simple path that includes all the players.

(a) Give an example of a tournament digraph with more than one ranking.

(b) If a tournament digraph is a DAG, then it has a unique ranking. Explain.

(c) Prove that every tournament digraph has a ranking. *Hint:* Induction on the size of the tournament.

Homework Problems

Problem 9.4.

Let R be a binary relation on a set A. Then R^n denotes the composition of R with itself n times. Regarding R as a digraph, let $R^{(n)}$ denote the length n path relation R, that is,

$$a\ R^{(n)}\ b ::= \text{there is a length } n \text{ path from } a \text{ to } b \text{ in } R.$$

Prove that

$$R^n = R^{(n)} \tag{9.10}$$

for all $n \in \mathbb{N}$.

Problem 9.5.

If R is a binary relation on a set, A, then R^k denotes the relational composition of R with itself k times.

(a) Prove that if R is a relation on a finite set, A, then

$$a\ (R \cup I_A)^n\ b \quad \text{iff} \quad \text{there is a path in } R \text{ of length length} \leq n \text{ from } a \text{ to } b.$$

(b) Conclude that if A is a finite set, then

$$R^* = (R \cup I_A)^{|A|-1}. \tag{9.11}$$

Problem 9.6.

In this problem we'll consider some special cycles in graphs called *Euler tours*, named after the famous mathematician Leonhard Euler. (Same Euler as for the constant $e \approx 2.718$—he did a lot of stuff.)

Definition 9.11.4. An Euler tour of a graph is a closed walk that includes every edge exactly once.

So how do you tell in general whether a graph has an Euler tour? At first glance this may seem like a daunting problem (the similar sounding problem of finding a cycle that touches every vertex exactly once is one of those million dollar NP-complete problems known as the *Traveling Salesman Problem*)—but it turns out to be easy.

(a) Show that if a graph has an Euler tour, then the in-degree of each vertex equals its out-degree.

A digraph is *weakly connected* if there is a "path" between any two vertices that may follow edges backwards or forwards. More precisely, a digraph, G is weakly connected iff there is a path from each vertex to every other vertex in the digraph $G \cup G^{-1}$.

In the remaining parts, we'll work out the converse: if a graph is weakly connected and if the in-degree of every vertex equals its out-degree, then the graph has an Euler tour. To do this, let's define an Euler *walk* to be a walk that traverses each edge *at most* once.

(b) Suppose that an Euler path in a connected graph does not include every edge. Explain why there must be an edge not on the path whose head or tail is on the path.

In the remaining parts, let W be the *longest* Euler path in the graph.

(c) Show that if W is a cycle, then it must be an Euler tour.

Hint: part (b)

(d) Explain why all the edges incident to the end of W must be on W.

(e) Show that if the end of W was not equal to the start of W, then the degree of the end would be odd.

Hint: part (d)

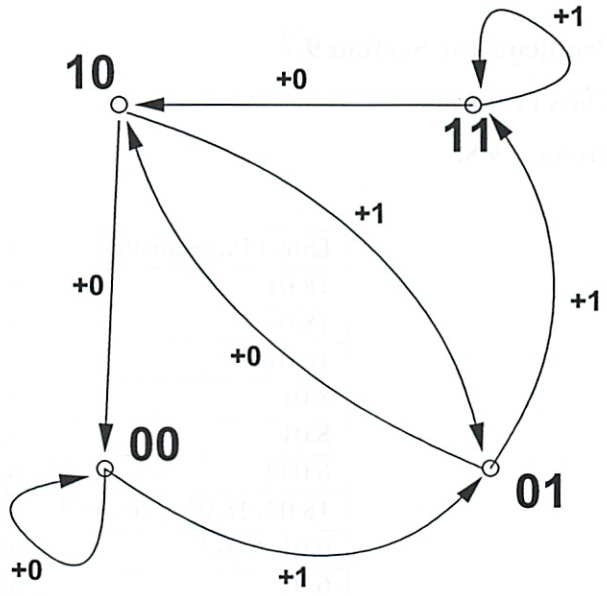
(f) Conclude that if every vertex of a finite, connected graph has even degree, then it has an Euler tour.

Problem 9.7.

A 3-bit string is a string made up of 3 characters, each a 0 or a 1. Suppose you'd like to write out, in one string, all eight of the 3-bit strings in any convenient order. For example, if you wrote out the 3-bit strings in the usual order starting with 000 001 010..., you could concatenate them together to get a length $3 \cdot 8 = 24$ string that started 000001010....

But you can get a shorter string containing all eight 3-bit strings by starting with 00010.... Now 000 is present as bits 1 through 3, 001 is present as bits 2 through 4, 010 is present as bits 3 through 5, ...

(a) Take a few moments to see how short a string you can make that contains every 3-bit string as 3 consecutive bits somewhere in it. Can you see why 10 bits is the absolute minimum length for such a string?



(b) Imagine that the labels on the vertices of the graph above represent the last two digits in a string you build by adding one bit at a time. Convince yourself that the graph completely describes how the last two digits of your string can change throughout this process.

(c) Find a directed path in this graph starting at some vertex, v, that contains every edge exactly once. Note that vertices will have to be used more than once and the path will have to end in v.

(d) Explain how such a path provides a shortest possible solution to the original

problem.

(e) What about k-bit substrings, $k = 4, 5, \ldots$? Can you define the appropriate generalization of the useful graph above? (They're called *de Bruijn graphs*.) If you do it sucessfully, you should be able to see that the in-degree (as well as the out-degree) of every vertex is 2.

Problem 9.6 shows that if the in-degree of every vertex is equal to its out-degree then a directed path can be drawn in that digraph that uses every edge exactly once.[11] You might want to think about why this should be true or how you might find such a path.

But if you do believe it, you should be able to see why all 2^k k-bit strings can be written as substrings of a string of length $2^k + k - 1$. (These strings are essentially de Bruijn strings.)

Problems for Section 9.7

Class Problems

Problem 9.8.

Direct Prerequisites	Subject
18.01	6.042
18.01	18.02
18.01	18.03
8.01	8.02
8.01	6.01
6.042	6.046
18.02, 18.03, 8.02, 6.01	6.02
6.01, 6.042	6.006
6.01	6.034
6.02	6.004

(a) For the above table of MIT subject prerequisites, draw a diagram showing the subject numbers with a line going down to every subject from each of its (direct) prerequisites.

(b) Give an example of a collection of sets partially ordered by the proper subset relation, $\subset$, that is isomorphic to ("same shape as") the prerequisite relation among MIT subjects from part (a).

[11] The graph must also be weakly connected, see Problem 9.6.

(c) Explain why the empty relation is a strict partial order and describe a collection of sets partially ordered by the proper subset relation that is isomorphic to the empty relation on five elements—that is, the relation under which none of the five elements is related to anything.

(d) Describe a *simple* collection of sets partially ordered by the proper subset relation that is isomorphic to the "properly contains" relation, $\supset$, on $\mathcal{P}\{1,2,3,4\}$.

Problem 9.9.
Consider the proper subset partial order, $\subset$, on the power set $\mathcal{P}\{1,2,\ldots,6\}$.

(a) What is the size of a maximal chain in this partial order? Describe one.

(b) Describe the largest antichain you can find in this partial order.

(c) What are the maximal and minimal elements? Are they maximum and minimum?

(d) Answer the previous part for the $\subset$ partial order on the set $\mathcal{P}\{1,2,\ldots,6\} - \emptyset$.

Homework Problems

Problem 9.10.
This problem asks for a proof of Lemma 9.7.2 showing that every weak partial order can be represented by (is isomorphic to) a collection of sets partially ordered under set inclusion ($\subseteq$). Namely,

Lemma. *Let $\preceq$ be a weak partial order on a set, A. For any element $a \in A$, let*

$$L(a) ::= \{b \in A \mid b \preceq a\},$$
$$\mathcal{L} ::= \{L(a) \mid a \in A\}.$$

Then the function $L : A \to \mathcal{L}$ is an isomorphism from the $\preceq$ relation on A, to the subset relation on $\mathcal{L}$.

(a) Prove that the function $\mathrm{L} : A \to \mathcal{L}$ is a bijection.

(b) Complete the proof by showing that

$$a \preceq b \quad \text{iff} \quad \mathrm{L}(a) \subseteq \mathrm{L}(b) \tag{9.12}$$

for all $a, b \in A$.

Problems for Section 9.8

Practice Problems

Problem 9.11.
For each of the binary relations below, state whether it is a strict partial order, a weak partial order, or neither. If it is not a partial order, indicate which of the axioms for partial order it violates. If it is a partial order, state whether it is a total order and identify its maximal and minimal elements, if any.

(a) The superset relation, $\supseteq$ on the power set $\mathcal{P}\{1, 2, 3, 4, 5\}$.

(b) The relation between any two nonegative integers, a, b that the remainder of a divided by 8 equals the remainder of b divided by 8.

(c) The relation between propositional formulas, G, H, that G IMPLIES H is valid.

(d) The relation 'beats' on Rock, Paper and Scissor (for those who don't know the game Rock, Paper, Scissors, Rock beats Scissors, Scissors beats Paper and Paper beats Rock).

(e) The empty relation on the set of real numbers.

(f) The identity relation on the set of integers.

(g) The divisibility relation on the integers, $\mathbb{Z}$.

Class Problems

Problem 9.12. **(a)** Verify that the divisibility relation on the set of nonnegative integers is a weak partial order.

(b) What about the divisibility relation on the set of integers?

Problem 9.13.
Consider the nonnegative numbers partially ordered by divisibility.

(a) Show that this partial order has a unique minimal element.

(b) Show that this partial order has a unique maximal element.

(c) Prove that this partial order has an infinite chain.

(d) An *antichain* in a partial order is a set of elements such that any two elements in the set are incomparable. Prove that this partial order has an infinite antichain. *Hint:* The primes.

(e) What are the minimal elements of divisibility on the integers greater than 1? What are the maximal elements?

Problem 9.14.
How many binary relations are there on the set $\{0, 1\}$?

How many are there that are transitive?, ... asymmetric?, ... reflexive?, ... irreflexive?, ... strict partial orders?, ... weak partial orders?

Hint: There are easier ways to find these numbers than listing all the relations and checking which properties each one has.

Problem 9.15.
Prove that if a binary relation on a set is transitive and irreflexive, then it is asymmetric.

Problem 9.16.
Prove that if R is a partial order, then so is R^{-1}

Homework Problems

Problem 9.17.
Let R and S be binary relations on the same set, A.

Definition 9.11.5. The *composition*, $S \circ R$, of R and S is the binary relation on A defined by the rule:[12]

$$a\ (S \circ R)\ c \quad \text{iff} \quad \exists b\,[a\ R\ b\ \text{AND}\ b\ S\ c].$$

Suppose both R and S are transitive. Which of the following new relations must also be transitive? For each part, justify your answer with a brief argument if the new relation is transitive and a counterexample if it is not.

[12]Note the reversal in the order of R and S. This is so that relational composition generalizes function composition, Composing the functions f and g means that f is applied first, and then g is applied to the result. That is, the value of the composition of f and g applied to an argument, x, is $g(f(x))$. To reflect this, the notation $g \circ f$ is commonly used for the composition of f and g. Some texts do define $g \circ f$ the other way around.

(a) R^{-1}

(b) $R \cap S$

(c) $R \circ R$

(d) $R \circ S$

Exam Problems

Problem 9.18.

(a) For each row in the following table, indicate whether the binary relation, R, on the set, A, is a weak partial order or a total order by filling in the appropriate entries with either Y = YES or N = NO. In addition, list the minimal and maximal elements for each relation.

A	a R b	weak partial order	total order	minimal(s)	maximal(s)
$\mathbb{R} - \mathbb{R}^+$	$a \mid b$				
$\mathcal{P}(\{1,2,3\})$	$a \subseteq b$				
$\mathbb{N} \cup \{i\}$	$a > b$				

0.5in

(b) What is the longest *chain* on the subset relation, $\subseteq$, on $P(\{1,2,3\})$? (If there is more than one, provide ONE of them.)

1.5in

(c) What is the longest *antichain* on the subset relation, $\subseteq$, on $P(\{1,2,3\})$? (If there is more than one, provide *one* of them.)

Problems for Section 9.9

Class Problems

Problem 9.19.

Let R_1, R_2 be binary relations on the same set, A. A relational property is preserved under product, if $R_1 \times R_2$ has the property whenever both R_1 and R_2 have the property.

(a) Verify that each of the following properties are preserved under product.

1. reflexivity,
2. antisymmetry,
3. transitivity.

(b) Verify that if *either* of R_1 or R_2 is irreflexive, then so is $R_1 \times R_2$.

Note that it now follows immediately that if if R_1 and R_2 are partial orders and at least one of them is strict, then $R_1 \times R_2$ is a strict partial order.

Problems for Section 9.10

Practice Problems

Problem 9.20.
What is the size of the longest chain that is guaranteed to exist in any partially ordered set of n elements? What about the largest antichain?

Problem 9.21.
Describe a sequence consisting of the integers from 1 to 10,000 in some order so that there is no increasing or decreasing subsequence of size 101.

Problem 9.22.
What is the smallest number of partially ordered tasks for which there can be more than one minimum time schedule? Explain.

Class Problems

Problem 9.23.
The table below lists some prerequisite information for some subjects in the MIT Computer Science program (in 2006). This defines an indirect prerequisite relation,

$\prec$, that is a strict partial order on these subjects.

$$\begin{aligned}
18.01 &\to 6.042 & 18.01 &\to 18.02\\
18.01 &\to 18.03 & 6.046 &\to 6.840\\
8.01 &\to 8.02 & 6.001 &\to 6.034\\
6.042 &\to 6.046 & 18.03, 8.02 &\to 6.002\\
6.001, 6.002 &\to 6.003 & 6.001, 6.002 &\to 6.004\\
6.004 &\to 6.033 & 6.033 &\to 6.857
\end{aligned}$$

(a) Explain why exactly six terms are required to finish all these subjects, if you can take as many subjects as you want per term. Using a *greedy* subject selection strategy, you should take as many subjects as possible each term. Exhibit your complete class schedule each term using a greedy strategy.

(b) In the second term of the greedy schedule, you took five subjects including 18.03. Identify a set of five subjects not including 18.03 such that it would be possible to take them in any one term (using some nongreedy schedule). Can you figure out how many such sets there are?

(c) Exhibit a schedule for taking all the courses—but only one per term.

(d) Suppose that you want to take all of the subjects, but can handle only two per term. Exactly how many terms are required to graduate? Explain why.

(e) What if you could take three subjects per term?

Problem 9.24.

A pair of Math for Computer Science Teaching Assistants, Jay and Oscar, have decided to devote some of their spare time this term to establishing dominion over the entire galaxy. Recognizing this as an ambitious project, they worked out the following table of tasks on the back of Oscar's copy of the lecture notes.

1. **Devise a logo** and cool imperial theme music - 8 days.
2. **Build a fleet** of Hyperwarp Stardestroyers out of eating paraphernalia swiped from Lobdell - 18 days.
3. **Seize control** of the United Nations - 9 days, after task #1.
4. **Get shots** for Jay's cat, Tailspin - 11 days, after task #1.

5. **Open a Starbucks chain** for the army to get their caffeine - 10 days, after task #3.

6. **Train an army** of elite interstellar warriors by dragging people to see *The Phantom Menace* dozens of times - 4 days, after tasks #3, #4, and #5.

7. **Launch the fleet** of Stardestroyers, crush all sentient alien species, and establish a Galactic Empire - 6 days, after tasks #2 and #6.

8. **Defeat Microsoft** - 8 days, after tasks #2 and #6.

We picture this information in Figure 9.9 below by drawing a point for each task, and labelling it with the name and weight of the task. An edge between two points indicates that the task for the higher point must be completed before beginning the task for the lower one.

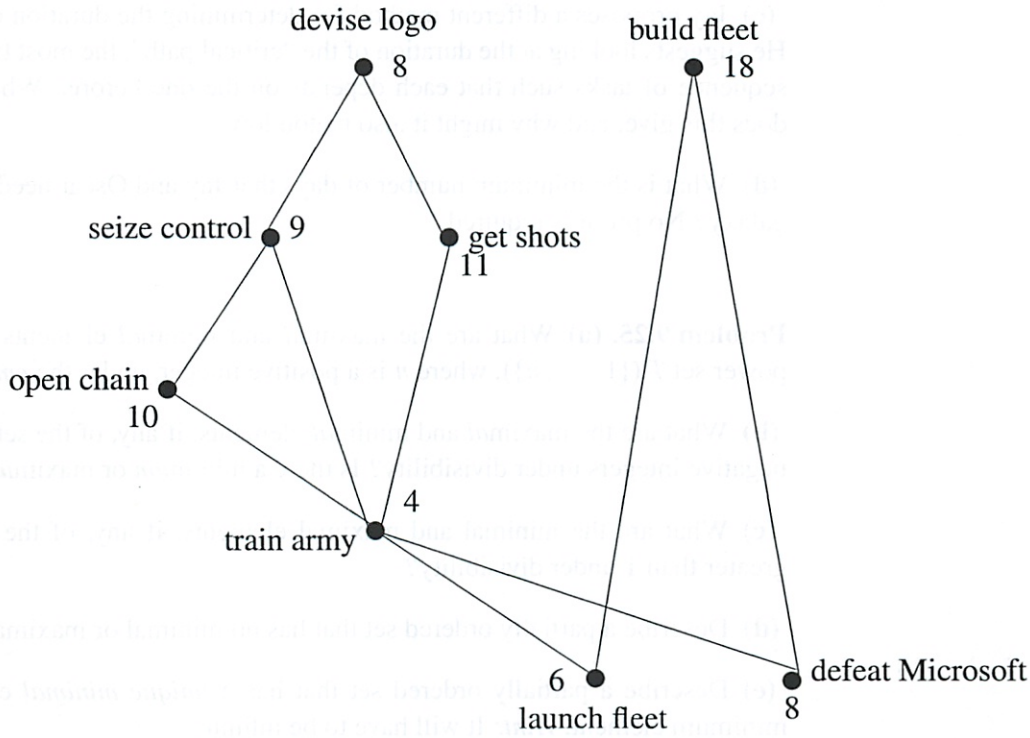


Figure 9.9 Graph representing the task precedence constraints.

(a) Give some valid order in which the tasks might be completed.

Jay and Oscar want to complete all these tasks in the shortest possible time. However, they have agreed on some constraining work rules.

- Only one person can be assigned to a particular task; they can not work together on a single task.
- Once a person is assigned to a task, that person must work exclusively on the assignment until it is completed. So, for example, Jay cannot work on building a fleet for a few days, run to get shots for Tailspin, and then return to building the fleet.

(b) Jay and Oscar want to know how long conquering the galaxy will take. Oscar suggests dividing the total number of days of work by the number of workers, which is two. What lower bound on the time to conquer the galaxy does this give, and why might the actual time required be greater?

(c) Jay proposes a different method for determining the duration of their project. He suggests looking at the duration of the "critical path", the most time-consuming sequence of tasks such that each depends on the one before. What lower bound does this give, and why might it also be too low?

(d) What is the minimum number of days that Jay and Oscar need to conquer the galaxy? No proof is required.

Problem 9.25. **(a)** What are the maxim*al* and minim*al* elements, if any, of the power set $\mathcal{P}(\{1,\ldots,n\})$, where n is a positive integer, under the *empty relation*?

(b) What are the maxim*al* and minim*al* elements, if any, of the set, $\mathbb{N}$, of all non-negative integers under divisibility? Is there a minim*um* or maxim*um* element?

(c) What are the minimal and maximal elements, if any, of the set of integers greater than 1 under divisibility?

(d) Describe a partially ordered set that has no minimal or maximal elements.

(e) Describe a partially ordered set that has a *unique minimal* element, but no minimum element. *Hint:* It will have to be infinite.

Homework Problems

Problem 9.26.
The following procedure can be applied to any digraph, G:

1. Delete an edge that is in a cycle.
2. Delete edge $\langle u \to v \rangle$ if there is a path from vertex u to vertex v that does not include $\langle u \to v \rangle$.
3. Add edge $\langle u \to v \rangle$ if there is no path in either direction between vertex u and vertex v.

Repeat these operations until none of them are applicable.

This procedure can be modeled as a state machine. The start state is G, and the states are all possible digraphs with the same vertices as G.

(a) Let G be the graph with vertices $\{1, 2, 3, 4\}$ and edges

$$\{\langle 1 \to 2 \rangle, \langle 2 \to 3 \rangle, \langle 3 \to 4 \rangle, \langle 3 \to 2 \rangle, \langle 1 \to 4 \rangle\}$$

What are the possible final states reachable from G?

A *line graph* is a graph whose edges are all on one path. All the final graphs in part (a) are line graphs.

(b) Prove that if the procedure terminates with a digraph, H, then H is a line graph with the same vertices as G.

Hint: Show that if H is *not* a line graph, then some operation must be applicable.

(c) Prove that being a DAG is a preserved invariant of the procedure.

(d) Prove that if G is a DAG and the procedure terminates, then the path relation of the final line graph is a topological sort of G.

Hint: Verify that the predicate

$$P(u, v) ::= \text{there is a directed path from } u \text{ to } v$$

is a preserved invariant of the procedure, for any two vertices u, v of a DAG.

(e) Prove that if G is finite, then the procedure terminates.

Hint: Let s be the number of cycles, e be the number of edges, and p be the number of pairs of vertices with a directed path (in either direction) between them. Note that $p \leq n^2$ where n is the number of vertices of G. Find coefficients a, b, c such that $as + bp + e + c$ is nonnegative integer valued and decreases at each transition.

Problem 9.27.
Let $\prec$ be a partial order on a set, A, and let

$$A_k ::= \{a \mid \text{depth}(a) = k\}$$

where $k \in \mathbb{N}$.

(a) Prove that $A_0, A_1, \ldots$ is a parallel schedule for $\prec$ according to Definition 9.10.4.

(b) Prove that A_k is an antichain.

Problem 9.28.
Let S be a sequence of n different numbers. A *subsequence* of S is a sequence that can be obtained by deleting elements of S.

For example, if

$$S = (6, 4, 7, 9, 1, 2, 5, 3, 8)$$

Then 647 and 7253 are both subsequences of S (for readability, we have dropped the parentheses and commas in sequences, so 647 abbreviates $(6, 4, 7)$, for example).

An *increasing subsequence* of S is a subsequence of whose successive elements get larger. For example, 1238 is an increasing subsequence of S. Decreasing subsequences are defined similarly; 641 is a decreasing subsequence of S.

(a) List all the maximum length increasing subsequences of S, and all the maximum length decreasing subsequences.

Now let A be the *set* of numbers in S. (So $A = \{1, 2, 3, \ldots, 9\}$ for the example above.) There are two straightforward ways to totally order A. The first is to order its elements numerically, that is, to order A with the $<$ relation. The second is to order the elements by which comes first in S; call this order $<_S$. So for the example above, we would have

$$6 <_S 4 <_S 7 <_S 9 <_S 1 <_S 2 <_S 5 <_S 3 <_S 8$$

Next, define the partial order $\prec$ on A defined by the rule

$$a \prec a' \quad ::= \quad a < a' \text{ and } a <_S a'.$$

(It's not hard to prove that $\prec$ is strict partial order, but you may assume it.)

(b) Draw a diagram of the partial order, $\prec$, on A. What are the maximal elements,... the minimal elements?

(c) Explain the connection between increasing and decreasing subsequences of S, and chains and anti-chains under $\prec$.

(d) Prove that every sequence, S, of length n has an increasing subsequence of length greater than $\sqrt{n}$ or a decreasing subsequence of length at least $\sqrt{n}$.

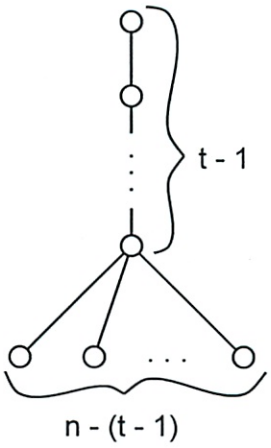
(e) (Optional, tricky) Devise an efficient procedure for finding the longest increasing and the longest decreasing subsequence in any given sequence of integers. (There is a nice one.)

Problem 9.29.

We want to schedule n partially ordered tasks.

(a) Explain why any schedule that requires only p processors must take time at least $\lceil n/p \rceil$.

(b) Let $D_{n,t}$ be the strict partial order with n elements that consists of a chain of $t-1$ elements, with the bottom element in the chain being a prerequisite of all the remaining elements as in the following figure:



What is the minimum time schedule for $D_{n,t}$? Explain why it is unique. How many processors does it require?

(c) Write a simple formula, $M(n,t,p)$, for the minimum time of a p-processor schedule to complete $D_{n,t}$.

(d) Show that *every* partial order with n vertices and maximum chain size, t, has a p-processor schedule that runs in time $M(n, t, p)$.

Hint: Induction on t.

Problems for Section 9.11

Homework Problems

Problem 9.30.

For any total function $f : A \to B$ define a relation $\equiv_f$ by the rule:

$$a \equiv_f a' \quad \text{iff} \quad f(a) = f(a'). \tag{9.13}$$

(a) Prove that $\equiv_f$ is an equivalence relation on A.

(b) Prove that every equivalence relation on a set A is equal to $\equiv_f$ for some total function $f : A \to B$.

Skip chap

10 Communication Networks

Modeling communication networks is an important application of digraphs in computer science. In this such models, vertices represent computers, processors, and switches; edges will represent wires, fiber, or other transmission lines through which data flows. For some communication networks, like the internet, the corresponding graph is enormous and largely chaotic. Highly structured networks, by contrast, find application in telephone switching systems and the communication hardware inside parallel computers. In this chapter, we'll look at some of the nicest and most commonly used structured networks.

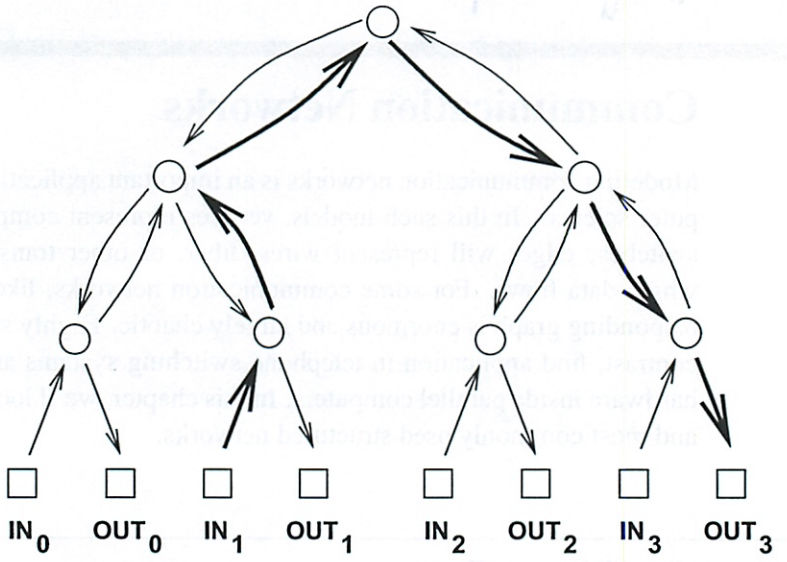
10.1 Complete Binary Tree

Let's start with a *complete binary tree*. Here is an example with 4 inputs and 4 outputs. The kinds of communication networks we consider aim to transmit packets of data between computers, processors, telephones, or other devices. The term *packet* refers to some roughly fixed-size quantity of data— 256 bytes or 4096 bytes or whatever. In this diagram and many that follow, the squares represent *terminals*, sources and destinations for packets of data. The circles represent *switches*, which direct packets through the network. A switch receives packets on incoming edges and relays them forward along the outgoing edges. Thus, you can imagine a data packet hopping through the network from an input terminal, through a sequence of switches joined by directed edges, to an output terminal.

Recall that there is a unique simple path between every pair of vertices in a tree. So the natural way to route a packet of data from an input terminal to an output in the complete binary tree is along the corresponding directed path. For example, the route of a packet traveling from input 1 to output 3 is shown in bold.

10.2 Routing Problems

Communication networks are supposed to get packets from inputs to outputs, with each packet entering the network at its own input switch and arriving at its own output switch. We're going to consider several different communication network designs, where each network has N inputs and N outputs; for convenience, we'll



assume N is a power of two.

Which input is supposed to go where is specified by a permutation of $\{0, 1, \ldots, N-1\}$. So a permutation, π, defines a *routing problem*: get a packet that starts at input i to output $\pi(i)$. A *routing*, P, that *solves* a routing problem, π, is a set of paths from each input to its specified output. That is, P is a set of n paths, P_i, for $i = 0 \ldots, N-1$, where P_i goes from input i to output $\pi(i)$.

10.3 Network Diameter

The delay between the time that a packets arrives at an input and arrives at its designated output is a critical issue in communication networks. Generally this delay is proportional to the length of the path a packet follows. Assuming it takes one time unit to travel across a wire, the delay of a packet will be the number of wires it crosses going from input to output.

Generally packets are routed to go from input to output by the shortest path possible. With a shortest path routing, the worst case delay is the distance between the input and output that are farthest apart. This is called the *diameter* of the network. In other words, the diameter of a network[1] is the maximum length of any shortest

[1]The usual definition of *diameter* for a general *graph* (simple or directed) is the largest distance between *any* two vertices, but in the context of a communication network we're only interested in the distance between inputs and outputs, not between arbitrary pairs of vertices.

path between an input and an output. For example, in the complete binary tree above, the distance from input 1 to output 3 is six. No input and output are farther apart than this, so the diameter of this tree is also six.

More generally, the diameter of a complete binary tree with N inputs and outputs is $2\log N + 2$. (All logarithms in this lecture— and in most of computer science — are base 2.) This is quite good, because the logarithm function grows very slowly. We could connect up $2^{10} = 1024$ inputs and outputs using a complete binary tree and the worst input-output delay for any packet would be this diameter, namely, $2\log(2^{10}) + 2 = 22$.

10.3.1 Switch Size

One way to reduce the diameter of a network is to use larger switches. For example, in the complete binary tree, most of the switches have three incoming edges and three outgoing edges, which makes them 3×3 switches. If we had 4×4 switches, then we could construct a complete *ternary* tree with an even smaller diameter. In principle, we could even connect up all the inputs and outputs via a single monster $N \times N$ switch.

This isn't very productive, however, since we've just concealed the original network design problem inside this abstract switch. Eventually, we'll have to design the internals of the monster switch using simpler components, and then we're right back where we started. So the challenge in designing a communication network is figuring out how to get the functionality of an $N \times N$ switch using fixed size, elementary devices, like 3×3 switches.

10.4 Switch Count

Another goal in designing a communication network is to use as few switches as possible. The number of switches in a complete binary tree is $1+2+4+8+\cdots+N$, since there is 1 switch at the top (the "root switch"), 2 below it, 4 below those, and so forth. By the formula (6.7) for geometric sums, the total number of switches is $2N - 1$, which is nearly the best possible with 3×3 switches.

10.5 Network Latency

We'll sometimes be choosing routings through a network that optimize some quantity besides delay. For example, in the next section we'll be trying to minimize packet congestion. When we're not minimizing delay, shortest routings are not always the best, and in general, the delay of a packet will depend on how it is routed. For any routing, the most delayed packet will be the one that follows the longest path in the routing. The length of the longest path in a routing is called its *latency*.

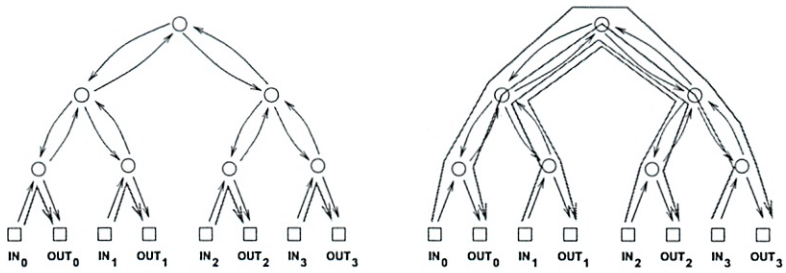
The latency of a *network* depends on what's being optimized. It is measured by assuming that optimal routings are always chosen in getting inputs to their specified outputs. That is, for each routing problem, π, we choose an optimal routing that solves π. Then *network latency* is defined to be the largest routing latency among these optimal routings. Network latency will equal network diameter if routings are always chosen to optimize delay, but it may be significantly larger if routings are chosen to optimize something else.

For the networks we consider below, paths from input to output are uniquely determined (in the case of the tree) or all paths are the same length, so network latency will always equal network diameter.

10.6 Congestion

The complete binary tree has a fatal drawback: the root switch is a bottleneck. At best, this switch must handle an enormous amount of traffic: every packet traveling from the left side of the network to the right or vice-versa. Passing all these packets through a single switch could take a long time. At worst, if this switch fails, the network is broken into two equal-sized pieces.

For example, if the routing problem is given by the identity permutation, $\mathrm{Id}(i) ::= i$, then there is an easy routing, P, that solves the problem: let P_i be the path from input i up through one switch and back down to output i. On the other hand, if the problem was given by $\pi(i) ::= (N-1) - i$, then in *any* solution, Q, for π, each path Q_i beginning at input i must eventually loop all the way up through the root switch and then travel back down to output $(N-1) - i$. These two situations are illustrated below. We can distinguish between a "good" set of paths and a "bad" set based on congestion. The *congestion* of a routing, P, is equal to the largest number of paths in P that pass through a single switch. For example, the congestion of the routing on the left is 1, since at most 1 path passes through each switch. However,



the congestion of the routing on the right is 4, since 4 paths pass through the root switch (and the two switches directly below the root). Generally, lower congestion is better since packets can be delayed at an overloaded switch.

By extending the notion of congestion to networks, we can also distinguish between "good" and "bad" networks with respect to bottleneck problems. For each routing problem, π, for the network, we assume a routing is chosen that optimizes congestion, that is, that has the minimum congestion among all routings that solve π. Then the largest congestion that will ever be suffered by a switch will be the maximum congestion among these optimal routings. This "maximin" congestion is called the *congestion of the network.*

So for the complete binary tree, the worst permutation would be $\pi(i) ::= (N - 1) - i$. Then in every possible solution for π, *every* packet, would have to follow a path passing through the root switch. Thus, the max congestion of the complete binary tree is N —which is horrible!

Let's tally the results of our analysis so far:

network	diameter	switch size	# switches	congestion
complete binary tree	$2\log N + 2$	3×3	$2N - 1$	N

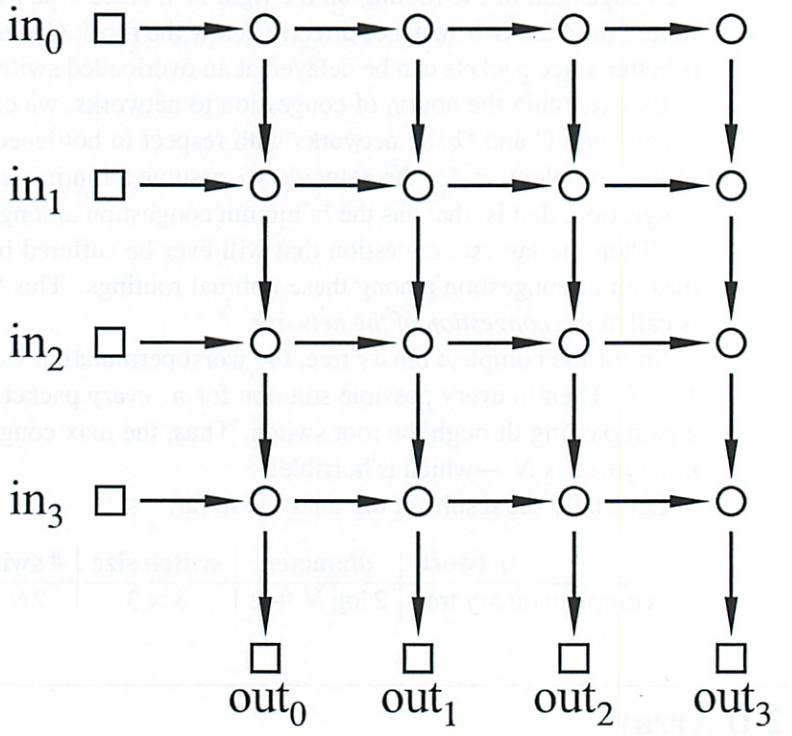
10.7 2-D Array

Let's look at an another communication network. This one is called a *2-dimensional array* or *grid.*

Here there are four inputs and four outputs, so $N = 4$.

The diameter in this example is 8, which is the number of edges between input 0 and output 3. More generally, the diameter of an array with N inputs and outputs is $2N$, which is much worse than the diameter of $2\log N + 2$ in the complete binary tree. On the other hand, replacing a complete binary tree with an array almost eliminates congestion.

Theorem 10.7.1. *The congestion of an N-input array is 2.*


in0
in1
in2
in3
out0
out1
out2
out3

Proof. First, we show that the congestion is at most 2. Let π be any permutation. Define a solution, P, for π to be the set of paths, P_i, where P_i goes to the right from input i to column $\pi(i)$ and then goes down to output $\pi(i)$. Thus, the switch in row i and column j transmits at most two packets: the packet originating at input i and the packet destined for output j.

Next, we show that the congestion is at least 2. This follows because in any routing problem, π, where $\pi(0) = 0$ and $\pi(N-1) = N-1$, two packets must pass through the lower left switch. ■

As with the tree, the network latency when minimizing congestion is the same as the diameter. That's because all the paths between a given input and output are the same length.

Now we can record the characteristics of the 2-D array.

network	diameter	switch size	# switches	congestion
complete binary tree	$2\log N + 2$	3×3	$2N - 1$	N
2-D array	$2N$	2×2	N^2	2

The crucial entry here is the number of switches, which is N^2. This is a major defect of the 2-D array; a network of size $N = 1000$ would require a *million* 2×2 switches! Still, for applications where N is small, the simplicity and low congestion of the array make it an attractive choice.

10.8 Butterfly

The Holy Grail of switching networks would combine the best properties of the complete binary tree (low diameter, few switches) and of the array (low congestion). The *butterfly* is a widely-used compromise between the two.

A good way to understand butterfly networks is as a recursive data type. The recursive definition works better if we define just the switches and their connections, omitting the terminals. So we recursively define F_n to be the switches and connections of the butterfly net with $N ::= 2^n$ input and output switches.

The base case is F_1 with 2 input switches and 2 output switches connected as in Figure 10.1.

In the constructor step, we construct F_{n+1} with 2^{n+1} inputs and outputs out of two F_n nets connected to a new set of 2^{n+1} input switches, as shown in as in Figure 10.2. That is, the ith and $2^n + i$th new input switches are each connected to the same two switches, namely, to the ith input switches of each of two F_n

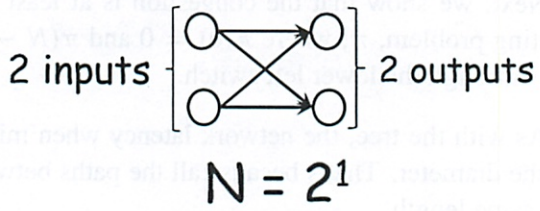


Figure 10.1 F_1, the Butterfly Net switches with $N = 2^1$.

components for $i = 1, \ldots, 2^n$. The output switches of F_{n+1} are simply the output switches of each of the F_n copies.

So F_{n+1} is laid out in columns of height 2^{n+1} by adding one more column of switches to the columns in F_n. Since the construction starts with two columns when $n = 1$, the F_{n+1} switches are arrayed in $n + 1$ columns. The total number of switches is the height of the columns times the number of columns, namely, $2^{n+1}(n + 1)$. Remembering that $n = \log N$, we conclude that the Butterfly Net with N inputs has $N(\log N + 1)$ switches.

Since every path in F_{n+1} from an input switch to an output is the same length, namely, $n + 1$, the diameter of the Butterfly net with 2^{n+1} inputs is this length plus two because of the two edges connecting to the terminals (square boxes) — one edge from input terminal to input switch (circle) and one from output switch to output terminal.

There is an easy recursive procedure to route a packet through the Butterfly Net. In the base case, there is obviously only one way to route a packet from one of the two inputs to one of the two outputs. Now suppose we want to route a packet from an input switch to an output switch in F_{n+1}. If the output switch is in the "top" copy of F_n, then the first step in the route must be from the input switch to the unique switch it is connected to in the top copy; the rest of the route is determined by recursively routing the rest of the way in the top copy of F_n. Likewise, if the

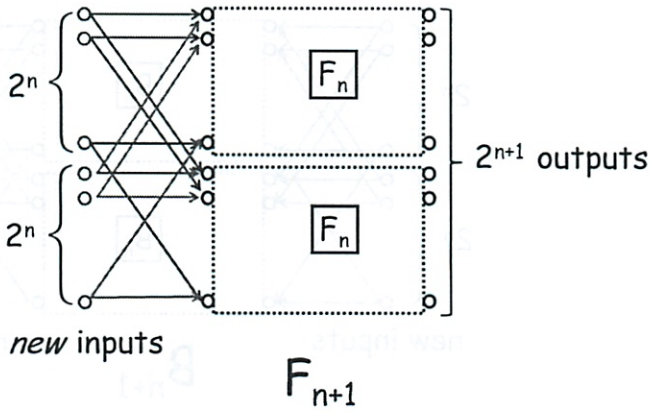


Figure 10.2 F_{n+1}, the Butterfly Net switches with 2^{n+1} inputs and outputs.

output switch is in the "bottom" copy of F_n, then the first step in the route must be to the switch in the bottom copy, and the rest of the route is determined by recursively routing in the bottom copy of F_n. In fact, this argument shows that the routing is *unique*: there is exactly one path in the Butterfly Net from each input to each output, which implies that the network latency when minimizing congestion is the same as the diameter.

The congestion of the butterfly network is about $\sqrt{N}$, more precisely, the congestion is $\sqrt{N}$ if N is an even power of 2 and $\sqrt{N/2}$ if N is an odd power of 2. A simple proof of this appears in Problem10.8.

Let's add the butterfly data to our comparison table:

network	diameter	switch size	# switches	congestion
complete binary tree	$2\log N + 2$	3×3	$2N - 1$	N
2-D array	$2N$	2×2	N^2	2
butterfly	$\log N + 2$	2×2	$N(\log(N) + 1)$	$\sqrt{N}$ or $\sqrt{N/2}$

The butterfly has lower congestion than the complete binary tree. And it uses fewer switches and has lower diameter than the array. However, the butterfly does not capture the best qualities of each network, but rather is a compromise somewhere between the two. So our quest for the Holy Grail of routing networks goes on.

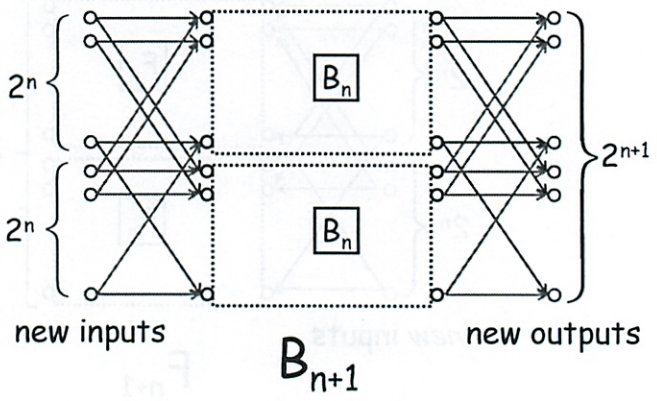


Figure 10.3 B_{n+1}, the Beneš Net switches with 2^{n+1} inputs and outputs.

10.9 Beneš Network

In the 1960's, a researcher at Bell Labs named Beneš had a remarkable idea. He obtained a marvelous communication network with congestion 1 by placing *two* butterflies back-to-back. This amounts to recursively growing *Beneš nets* by adding both inputs and outputs at each stage. Now we recursively define B_n to be the switches and connections (without the terminals) of the Beneš net with $N ::= 2^n$ input and output switches.

The base case, B_1, with 2 input switches and 2 output switches is exactly the same as F_1 in Figure 10.1.

In the constructor step, we construct B_{n+1} out of two B_n nets connected to a new set of 2^{n+1} input switches *and also* a new set of 2^{n+1} output switches. This is illustrated in Figure 10.3.

Namely, the ith and $2^n + i$th new input switches are each connected to the same two switches, namely, to the ith input switches of each of two B_n components for $i = 1, \ldots, 2^n$, exactly as in the Butterfly net. In addition, the ith and $2^n + i$th new *output* switches are connected to the same two switches, namely, to the ith output switches of each of two B_n components.

Now B_{n+1} is laid out in columns of height 2^{n+1} by adding two more columns of switches to the columns in B_n. So the B_{n+1} switches are arrayed in $2(n+1)$ columns. The total number of switches is the number of columns times the height of the columns, namely, $2(n+1)2^{n+1}$.

All paths in B_{n+1} from an input switch to an output are the same length, namely, $2(n+1)-1$, and the diameter of the Beneš net with 2^{n+1} inputs is this length plus two because of the two edges connecting to the terminals.

So Beneš has doubled the number of switches and the diameter, of course, but completely eliminates congestion problems! The proof of this fact relies on a clever induction argument that we'll come to in a moment. Let's first see how the Beneš network stacks up:

network	diameter	switch size	# switches	congestion
complete binary tree	$2\log N + 2$	3×3	$2N - 1$	N
2-D array	$2N$	2×2	N^2	2
butterfly	$\log N + 2$	2×2	$N(\log(N) + 1)$	$\sqrt{N}$ or $\sqrt{N/2}$
Beneš	$2\log N + 1$	2×2	$2N \log N$	1

The Beneš network has small size and diameter, and completely eliminates congestion. The Holy Grail of routing networks is in hand!

Theorem 10.9.1. *The congestion of the N-input Beneš network is 1.*

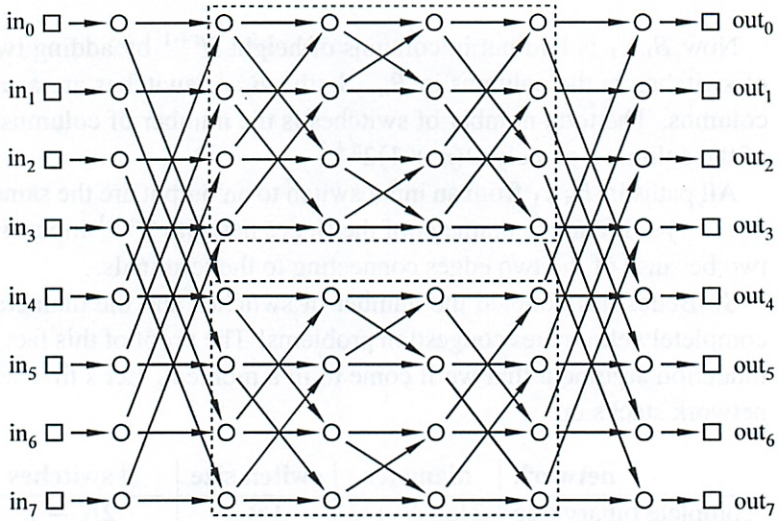
Proof. By induction on n where $N = 2^n$. So the induction hypothesis is

$$P(n) ::= \text{the congestion of } B_n \text{ is 1.}$$

Base case ($n = 1$): $B_1 = F_1$ and the unique routings in F_1 have congestion 1.

Inductive step: We assume that the congestion of an $N = 2^n$-input Beneš network is 1 and prove that the congestion of a $2N$-input Beneš network is also 1.

Digression. Time out! Let's work through an example, develop some intuition, and then complete the proof. In the Beneš network shown below with $N = 8$ inputs and outputs, the two 4-input/output subnetworks are in dashed boxes.



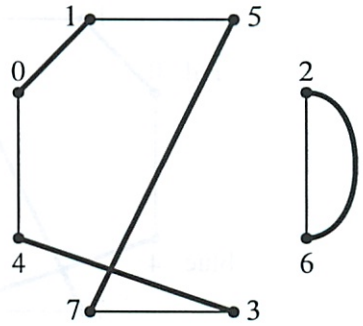
By the inductive assumption, the subnetworks can each route an arbitrary permutation with congestion 1. So if we can guide packets safely through just the first and last levels, then we can rely on induction for the rest! Let's see how this works in an example. Consider the following permutation routing problem:

$$\begin{aligned} \pi(0) &= 1 & \pi(4) &= 3 \\ \pi(1) &= 5 & \pi(5) &= 6 \\ \pi(2) &= 4 & \pi(6) &= 0 \\ \pi(3) &= 7 & \pi(7) &= 2 \end{aligned}$$

We can route each packet to its destination through either the upper subnetwork or the lower subnetwork. However, the choice for one packet may constrain the choice for another. For example, we can not route both packet 0 *and* packet 4 through the same network since that would cause two packets to collide at a single switch, resulting in congestion. So one packet must go through the upper network and the other through the lower network. Similarly, packets 1 and 5, 2 and 6, and 3 and 7 must be routed through different networks. Let's record these constraints in a graph. The vertices are the 8 packets. If two packets must pass through different networks, then there is an edge between them. Thus, our constraint graph looks like this:

Notice that at most one edge is incident to each vertex.

The output side of the network imposes some further constraints. For example, the packet destined for output 0 (which is packet 6) and the packet destined for output 4 (which is packet 2) can not both pass through the same network; that would require both packets to arrive from the same switch. Similarly, the packets destined for outputs 1 and 5, 2 and 6, and 3 and 7 must also pass through different switches. We can record these additional constraints in our graph with gray edges:



Notice that at most one new edge is incident to each vertex. The two lines drawn between vertices 2 and 6 reflect the two different reasons why these packets must be routed through different networks. However, we intend this to be a simple graph; the two lines still signify a single edge.

Now here's the key insight: suppose that we could color each vertex either red or blue so that adjacent vertices are colored differently. Then all constraints are satisfied if we send the red packets through the upper network and the blue packets through the lower network. Such a *2-coloring of the graph corresponds to a solution to the routing problem.* The only remaining question is whether the constraint graph is 2-colorable, which is easy to verify:

Lemma 10.9.2. *Prove that if the edges of a graph can be grouped into two sets such that every vertex has at most 1 edge from each set incident to it, then the graph is 2-colorable.*

Proof. It is not hard to show that A graph is 2-colorable iff every cycle in it has even length (see Problem 10.9). We'll take this for granted here.

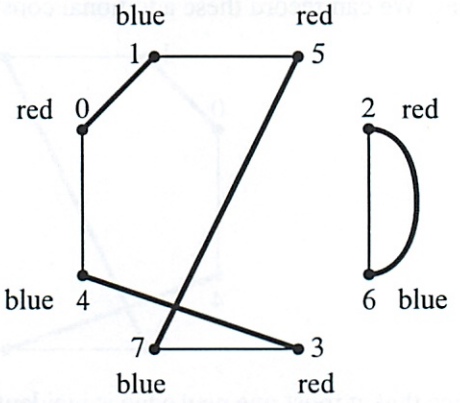
So all we have to do is show that every cycle has even length. Since the two sets of edges may overlap, let's call an edge that is in both sets a *doubled edge*.

There are two cases:

Case 1: [The cycle contains a doubled edge.] No other edge can be incident to either of the endpoints of a doubled edge, since that endpoint would then be incident to two edges from the same set. So a cycle traversing a doubled edge has nowhere to go but back and forth along the edge an even number of times.

Case 2: [No edge on the cycle is doubled.] Since each vertex is incident to at most one edge from each set, any path with no doubled edges must traverse successive edges that alternate from one set to the other. In particular, a cycle must traverse a path of alternating edges that begins and ends with edges from different sets. This means the cycle has to be of even length. ■

For example, here is a 2-coloring of the constraint graph:



The solution to this graph-coloring problem provides a start on the packet routing problem:

We can complete the routing in the two smaller Beneš networks by induction! Back to the proof. **End of Digression.**

Let π be an arbitrary permutation of $\{0, 1, \ldots, N-1\}$. Let G be the graph whose vertices are packet numbers $0, 1, \ldots, N-1$ and whose edges come from the union of these two sets:

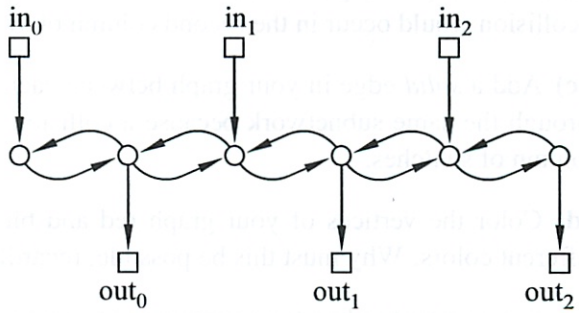
$$E_1 ::= \{u — v \mid |u - v| = N/2\}, \text{ and}$$
$$E_2 ::= \{u — w \mid |\pi(u) - \pi(w)| = N/2\}.$$

Now any vertex, u, is incident to at most two edges: a unique edge u—$v \in E_1$ and a unique edge u—$w \in E_2$. So according to Lemma 10.9.2, there is a 2-coloring for the vertices of G. Now route packets of one color through the upper subnetwork and packets of the other color through the lower subnetwork. Since for each edge in E_1, one vertex goes to the upper subnetwork and the other to the lower subnetwork, there will not be any conflicts in the first level. Since for each edge in E_2, one vertex comes from the upper subnetwork and the other from the lower subnetwork, there will not be any conflicts in the last level. We can complete the routing within each subnetwork by the induction hypothesis $P(n)$. ■

Problems for Section 10.9

Exam Problems

Problem 10.1.
Consider the following communication network:



(a) What is the max congestion? 0.5in

(b) Give an input/output permutation, π_0, that forces maximum congestion:

$$\pi_0(0) = \_\_\_\_ \qquad \pi_0(1) = \_\_\_\_ \qquad \pi_0(2) = \_\_\_\_$$

(c) Give an input/output permutation, π_1, that allows *minimum* congestion:

$$\pi_1(0) = \_\_\_\_ \qquad \pi_1(1) = \_\_\_\_ \qquad \pi_1(2) = \_\_\_\_$$

(d) What is the latency for the permutation π_1? (If you could not find π_1, just choose a permutation and find its latency.) 0.5in

Class Problems

Problem 10.2.
The Beneš network has a max congestion of 1; that is, every permutation can be

routed in such a way that a single packet passes through each switch. Let's work through an example. A Beneš network of size $N = 8$ is attached.

(a) Within the Beneš network of size $N = 8$, there are two subnetworks of size $N = 4$. Put boxes around these. Hereafter, we'll refer to these as the *upper* and *lower* subnetworks.

(b) Now consider the following permutation routing problem:

$$\begin{aligned} \pi(0) &= 3 & \pi(4) &= 2 \\ \pi(1) &= 1 & \pi(5) &= 0 \\ \pi(2) &= 6 & \pi(6) &= 7 \\ \pi(3) &= 5 & \pi(7) &= 4 \end{aligned}$$

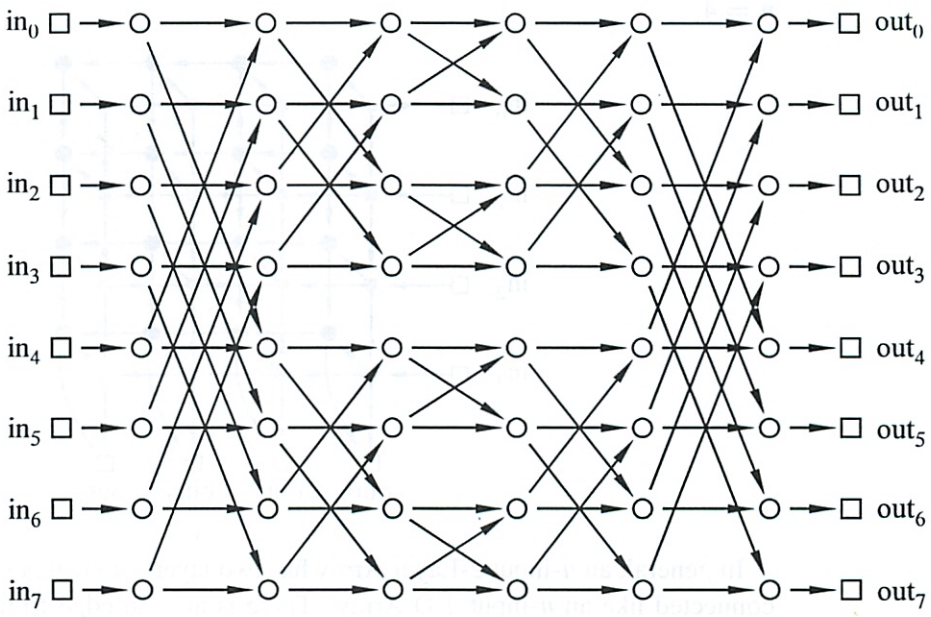
Each packet must be routed through either the upper subnetwork or the lower subnetwork. Construct a graph with vertices $0, 1, \ldots, 7$ and draw a *dashed* edge between each pair of packets that can not go through the same subnetwork because a collision would occur in the second column of switches.

(c) Add a *solid* edge in your graph between each pair of packets that can not go through the same subnetwork because a collision would occur in the next-to-last column of switches.

(d) Color the vertices of your graph red and blue so that adjacent vertices get different colors. Why must this be possible, regardless of the permutation π?

(e) Suppose that red vertices correspond to packets routed through the upper subnetwork and blue vertices correspond to packets routed through the lower subnetwork. On the attached copy of the Beneš network, highlight the first and last edge traversed by each packet.

(f) All that remains is to route packets through the upper and lower subnetworks. One way to do this is by applying the procedure described above recursively on each subnetwork. However, since the remaining problems are small, see if you can complete all the paths on your own.



Problem 10.3.

A *multiple binary-tree network* has n inputs and n outputs, where n is a power of 2. Each input is connected to the root of a binary tree with $n/2$ leaves and with edges pointing away from the root. Likewise, each output is connected to the root of a binary tree with $n/2$ leaves and with edges pointing toward the root.

Two edges point from each leaf of an input tree, and each of these edges points to a leaf of an output tree. The matching of leaf edges is arranged so that for every input and output tree, there is an edge from a leaf of the input tree to a leaf of the output tree, and every output tree leaf has exactly two edges pointing to it.

(a) Draw such a multiple binary-tree net for $n = 4$.

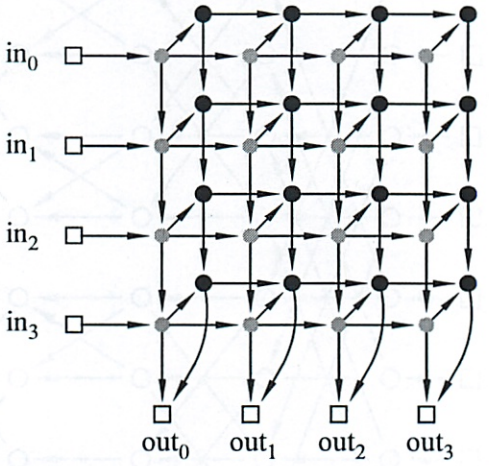
(b) Fill in the table, and explain your entries.

# switches	switch size	diameter	max congestion

Problem 10.4.

The n-input *2-D Array* network was shown to have congestion 2. An n-input *2-Layer Array* consisting of two n-input 2-D Arrays connected as pictured below for

$n = 4$.



In general, an n-input 2-Layer Array has two layers of switches, with each layer connected like an n-input 2-D Array. There is also an edge from each switch in the first layer to the corresponding switch in the second layer. The inputs of the 2-Layer Array enter the left side of the first layer, and the n outputs leave from the bottom row of either layer.

(a) For any given input-output permutation, there is a way to route packets that achieves congestion 1. Describe how to route the packets in this way.

(b) What is the latency of a routing designed to minimize latency?

(c) Explain why the congestion of any minimum latency (CML) routing of packets through this network is greater than the network's congestion.

Problem 10.5.

A *5-path* communication network is shown below. From this, it's easy to see what an n-path network would be. Fill in the table of properties below, and be prepared to justify your answers.

network	# switches	switch size	diameter	max congestion
5-path				
n-path				

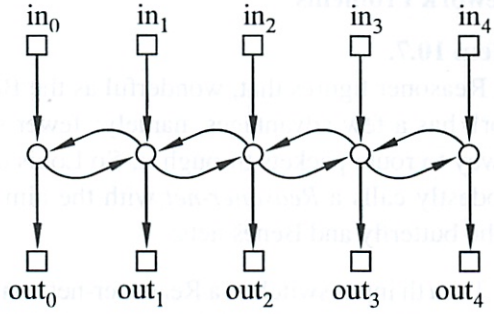
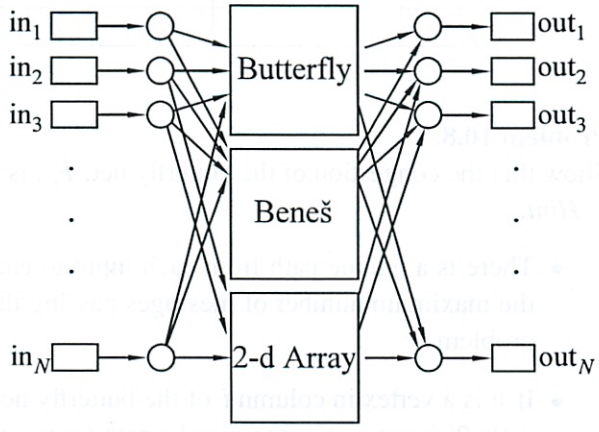


Figure 10.4 5-Path

Problem 10.6.

Tired of being a TA, Megumi has decided to become famous by coming up with a new, better communication network design. Her network has the following specifications: every input node will be sent to a butterfly network, a Beneš network and a 2-d array network. At the end, the outputs of all three networks will converge on the new output.

In the Megumi-net a minimum latency routing does not have minimum congestion. The *latency for min-congestion* (*LMC*) of a net is the best bound on latency achievable using routings that minimize congestion. Likewise, the *congestion for min-latency* (*CML*) is the best bound on congestion achievable using routings that minimize latency.



Fill in the following chart for Megumi's new net and explain your answers.

network	diameter	# switches	congestion	LMC	CML
Megumi's net					

Homework Problems

Problem 10.7.

Louis Reasoner figures that, wonderful as the Beneš network may be, the butterfly network has a few advantages, namely: fewer switches, smaller diameter, and an easy way to route packets through it. So Louis designs an N-input/output network he modestly calls a *Reasoner-net* with the aim of combining the best features of both the butterfly and Beneš nets:

> The ith input switch in a Reasoner-net connects to two switches, a_i and b_i, and likewise, the jth output switch has two switches, y_j and z_j, connected to it. Then the Reasoner-net has an N-input Beneš network connected using the a_i switches as input switches and the y_j switches as its output switches. The Reasoner-net also has an N-input butterfly net connected using the b_i switches as inputs and the z_j switches as outputs.

In the Reasoner-net a minimum latency routing does not have minimum congestion. The *latency for min-congestion* (*LMC*) of a net is the best bound on latency achievable using routings that minimize congestion. Likewise, the *congestion for min-latency* (*CML*) is the best bound on congestion achievable using routings that minimize latency.

Fill in the following chart for the Reasoner-net and briefly explain your answers.

diameter	switch size(s)	# switches	congestion	LMC	CML

Problem 10.8.

Show that the congestion of the butterfly net, F_n, is exactly $\sqrt{N}$ when n is even.

Hint:

- There is a unique path from each input to each output, so the congestion is the maximum number of messages passing through a vertex for any routing problem.
- If v is a vertex in column i of the butterfly network, there is a path from exactly 2^i input vertices to v and a path from v to exactly 2^{n-i} output vertices.
- At which column of the butterfly network must the congestion be worst? What is the congestion of the topmost switch in that column of the network?

Problem 10.9.
In this problem you will prove:

Theorem. *A graph G is 2-colorable iff it contains no odd length cycle.*

As usual with "iff" assertions, the proof splits into two proofs: part (a) asks you to prove that the left side of the "iff" implies the right side. The other problem parts prove that the right side implies the left.

(a) Assume the left side and prove the right side. Three to five sentences should suffice.

(b) Now assume the right side. As a first step toward proving the left side, explain why we can focus on a single connected component H within G.

(c) As a second step, explain how to 2-color any tree.

(d) Choose any 2-coloring of a spanning tree, T, of H. Prove that H is 2-colorable by showing that any edge *not* in T must also connect different-colored vertices.

Index