— INSERT EA goes here —

— INSERT EE goes here —
(from 10/2/08 lecture notes)

— INSERT EG goes here —

~~Chapter 12~~

~~Communication Networks~~

6.3 ~~12.1~~ Communication Networks

— INSERT EH goes here —

~~Modeling communication networks is an important application of digraphs in computer science. In this such models,~~ vertices represent computers, processors, or ~~and~~ switches; and 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~~ such as ~~complete binary trees~~ arrays or butterfly, by

INSERT EA

CHAPTER 6

Directed Graphs

6.1 Definitions

So far, we have been working with graphs with undirected edges. ~~i.e., where~~ ~~Indeed~~ A *directed edge* is an edge where the endpoints are distinguished. ~~— one is the head and one is the tail~~ — one is the *head* and one is the *tail*. In particular, ~~an~~ a directed edge is ~~pe~~ specified as an ordered pair of vertices ~~(u,v)~~ ~~or~~ u, v ~~where~~ and is denoted by (u,v) or $u \to v$. In this case, u is the tail of the edge and v is the head. For example see Figure EA.

Figure EA: A directed edge $e = (u,v)$. u is the tail of e and v is the head of e.

A graph with directed edges is called a directed graph or digraph. ~~Such graphs can have~~ ~~Definition~~

Definition A directed graph $G = (V, E)$ consists of a nonempty set of nodes V and a set of directed edges E. Each edge e of E is specified by an ordered pair of vertices $u, v \in V$ where $u \neq v$. A directed graph is simple if it has no loops (i.e. edges of the form $u \to u$) and no multiple edges. ~~(i.e., two or more edges of the ... with the same ...~~ ~~Note that ... In this text we will concentrate on simple directed graphs and will henceforth omit the word ...~~ ~~Note that a directed graph~~

Since we will focus on the case of simple directed graphs in this chapter, we will generally omit the word simple when referring to them. Note that such a graph can contain the edge $u \to v$ as well the edge $v \to u$ since these are different edges (e.g., they have a different tail).

Directed graphs arise in applications where the ~~edc~~ relationship represented by an edge is 1-way, or asymmetric. Examples include: ~~For example,~~ ~~one entity can~~ a 1-way street, one person likes another but the feelings are not necessarily reciprocated, a communication channel such as a cable modem that has more capacity for downloading ~~as~~ than uploading, one entity ~~person~~ is larger ~~stronger~~ than another, and one ~~pro~~ job needs to be completed before another can begin. We'll see several such examples in this chapter and also in Chapter 7.

Most all of the definitions for undirected graphs from Chapter 5 carry over in a natural way for directed graphs. ~~We'll cover some of the most important definitions in the rest of this section.~~ For example, two directed graphs $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ are isomorphic if there ~~as~~ exists a bijection $f: V_1 \to V_2$ such that for every pair of vertices $u, v, \in V_1$

$$u \to v \in E_1 \text{ iff } f(u) \to f(v) \in E_2.$$

~~In other words, the bijection f must preserve the direction of each edge as we~~

In a weighted directed graph $G = (V, E)$, there ~~every~~ is a weight function ~~f: E~~ $w: E \to R$ that specifies a weight for each edge. Note that the weight of $u \to v$ may be different than the weight of $v \to u$ since they are different edges.

Directed ~~Weighted~~ graphs have adjacency matrices just like undirected graphs. In the case of a directed graph $G = (V, E)$, the adjacency matrix $A_G = \{a_{ij}\}$ is defined ~~as fol~~ so that

$$a_{ij} = \begin{cases} 1 & \text{if } i \to j \in E \\ 0 & \text{otherwise} \end{cases}.$$

The only difference with undirected graphs is that the adjacency matrix for an ~~undirected~~ graph is not necessarily symmetric (i.e., it may be that $A_G^T \neq A_G$).

~~dele~~

6.1.1 Degrees

With directed graphs, the notion of degree splits into *indegree* and *outdegree*. For example, indegree (c) = 2 and outdegree (c) = 1 for the graph in Figure EB. ~~A node with~~ If a node has outdegree 0, it is called a *sink*; ~~and~~ if it has ~~a node with degree~~ indegree 0, it is called a *source*. ~~There are no sources or~~ The graph in Figure EB has one source (node a) and no sinks.

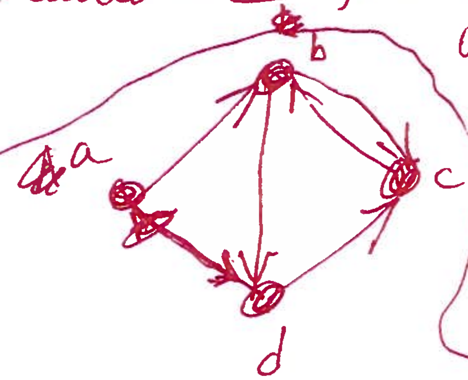


Figure EB: A 4-node directed graph with 6 edges.

~~Most/all of the undefinitions for ordinary undirected graphs from Chapter 5 carry over in the natural way for directed graphs.~~

6.1.2 Directed Walks, Paths and Cycles

The definitions for (directed) walks, paths and cycles in a directed graph are similar to those for undirected graphs except that the direction of the edges need to be consistent with the order in which the walk is traversed.

~~In some cases, there are more~~

Definition: A ~~...~~ directed walk (or more simply, a walk) in a directed graph G is a sequence of vertices $v_0, v_1, \ldots, v_k$ and edges

$$v_0 \to v_1,\ v_1 \to v_2, \ldots, v_{k-1} \to v_k$$

such that $v_i \to v_{i-1}$ is an edge of G for all i where $0 \le i < k$. A directed path (or path) in a directed graph is a walk where the nodes on the walk are all different. A directed closed walk (or closed walk) in a directed graph is a walk where $v_0 = v_k$. A directed cycle (or cycle) in a directed graph is a closed walk where all the vertices v_i are different ~~for the~~ for $0 \le i < k$.

~~For example,~~

As with undirected graphs, we will typically refer to a ~~directed~~ walk in a directed graph by a sequence of vertices. For example, for the graph in Figure 6B,

a, b, ~~d~~ c, b, d is a walk,

a, b, d is a path,

~~a~~ d, c, b, c, b, d is a closed walk, and b, d, c, b is a cycle.

\#Note that b, c, b is also a cycle for the graph in Figure EB. This is a cycle of length 2. Such cycles are not possible with undirected graphs.

\#Also note that

c, b, a, d

is not a walk in the graph shown in Figure EB, since b → a is ~~not~~ not an edge in this graph. (You are not allowed to traverse edges in the wrong direction as part of a walk.)

A path or cycle in a directed graph is said to be Hamiltonian if it visits every node in the graph. For example, a, b, d, c is the only Hamiltonian path for the graph in Figure EB. The graph in Figure EB does not have a Hamiltonian cycle.

A ~~closed walk~~ walk in a directed graph is said to be Eulerian if it contains every edge. The ~~For example, a, b, d, c, b, c is an Eulerian walk in the graph shown~~ graph shown in Figure EB does not have an Eulerian walk. Can you see why not? (Hint: look at node a.)

6.1.3 ~~DAGs~~ Strong Connectivity

~~If an undirected graph does~~

~~Conne~~ The notion of being connected is a little more complicated for a directed graph that it is for an undirected graph. For example, ~~is the graph~~ should we consider the graph in Figure EB to be connected? There is a path ~~from~~ node a to every other node so, on that basis, we might ~~say~~ answer "Yes." But there is no path from ~~any~~ nodes b, c, or d to node a, and so on that basis, we might answer "No." a

For this reason, ~~a~~ graph theorists have

~~to~~ come up with ~~to~~ the notion of strong connectivity for directed graphs.

Definition: A directed graph $G=(V,E)$ is said to be strongly connected if for every ~~e~~ pair of nodes $u, v \in V$, there is a directed path from u to v (and vice-versa) in G.

For example, the graph in Figure E B is not strongly connected since there is no directed ~~e~~ path from node b to node a. But if node a is removed, ~~from the gr~~ the resulting graph would be strongly connected.

A directed graph is said to be ~~con~~ weakly connected (or, more simply, connected) if the ~~underlying graph when the edge~~ corresponding undirected graph (where directed edges $u \to v$ and/or $v \to u$ are replaced with a single undirected edge $u - v$) is connected. For example, the graph in

Figure EB is weakly connected.

6.1.4 DAGs

If an undirected graph does not have any cycles, then it is a tree, or a ~~col~~ forest. But what ~~about a~~ is a directed graph look like if it has no cycles? For example, consider the graph in Figure ED. This graph is (weakly) connected and has no directed cycles but it ~~does~~ certainly does not look like a tree.

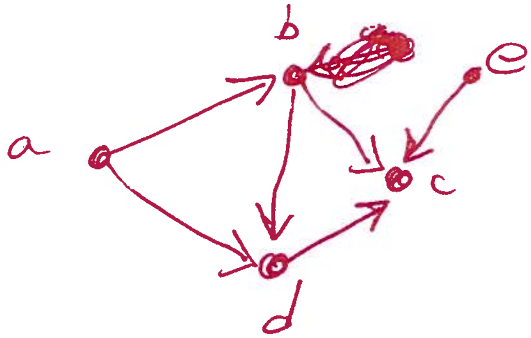


Figure ED: A ~~direct~~ 4-node directed ~~cycl~~ acyclic graph (DAG).

Definition: A directed graph is ~~a~~ called a directed acyclic graph (or, DAG) if it does not contain any directed cycles.

At first glance, DAGs don't appear to be particularly interesting. ~~A~~ But first ~~glan~~ impressions are not always accurate. In fact, DAGs arise in many scheduling and optimization problems and they have several interesting properties. We will study them extensively in chapter 7.

FROM 10/2/08 lecture notes EE-1

Then $W^T\vec{P} = \vec{P}$, where W^T denotes the transpose of W. Note that this follows from the fact that the equations have the form $\sum_{1 \le i \le n} W_{i,j} PR(x_i) = PR(x_j)$ for each $1 \le j \le n$.

If you have taken a linear algebra or numerical analysis course, you realize that the vector of page ranks is just the principle eigenvector of the weighted adjacency matrix of the web graph! Once you've had such a course, these values are easy to compute. Of course, when you are dealing with matrices of this size, the problem gets a little more interesting. Just keeping track of the digraph whose nodes are billions of web pages is a daunting task. That's why Google is building power plants. Indeed, Larry and Sergey named their system Google after the number 10^{100}, called "googol", to reflect the fact that the web graph is so enormous.

Anyway, now you can see how 6.042 ranked fourth out of 4 million matches. Lots of other universities use our notes and probably have links to the 6.042 open courseware site and they are themselves legitimate, which ultimately leads 6.042 to get a high page rank in the web graph.

The moral of this story is that you too can become a billionaire if you study your graph theory!

INSERT EE

6.2 Tournament Graphs

~~5 Tournament Rankings~~

Suppose that n players compete in a round-robin tournament. Thus, for every pair of players u and v, either u beats v or else v beats u. Interpreting the results of a round-robin tournament can be problematic. There might be all sorts of cycles where x beat y, y beat z, yet z beat x. Graph theory provides at least a partial solution to this problem.

The results of a round-robin tournament can be represented with a ***tournament graph***. not bold This is a directed graph in which the vertices represent players and the edges indicate the outcomes of games. In particular, an edge from u to v indicates that player u defeated player v. In a round-robin tournament, every pair of players has a match. Thus, in a tournament graph there is either an edge from u to v or an edge from v to u (but not both) for *every* pair of distinct vertices u and v. ~~Here is an~~ An example of a tournament graph is shown in Figure EE1.

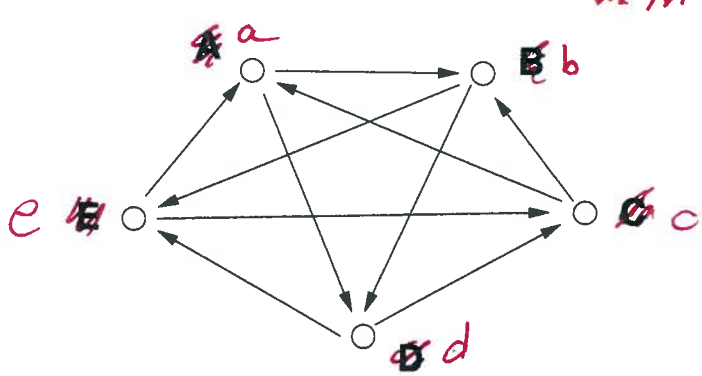


Figure EE1: A 5-node tournament graph.

6.2.1 Finding a Hamiltonian Path in a Tournament Graph (EE-2)

Its adjacency matrix is

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

(Notice that if M is the adjacency matrix of a tournament graph, then $I + M + M^T$, the addition of the identity matrix, the adjacency matrix, and its transpose, is equal to the all-one matrix.)

The notions of walks, Euler tours, and Hamiltonian cycles all carry over naturally to directed graphs. A ***directed walk*** is an alternating sequence of vertices and directed edges:

$$v_0, v_0 \longrightarrow v_1, v_1, v_1 \longrightarrow v_2, v_2, \ldots, v_{n-1}, v_{n-1} \longrightarrow v_n, v_n$$

~~A **directed Hamiltonian path** is a directed walk that visits every vertex exactly once.~~

We're going to prove that in every round-robin tournament, there exists a ranking of the players such that each player lost to the player ranked one position higher. For example, in the tournament ~~above~~ corresponding to Figure EE1, the ranking

$$\cancel{A} > \cancel{B} > \cancel{D} > \cancel{E} > \cancel{C}$$
$$a \quad b \quad d \quad e \quad c$$

satisfies this criterion, because ~~B~~ b lost to ~~A~~ a, ~~D~~ d lost to ~~B~~ b, ~~E~~ e lost to ~~D~~ d and ~~C~~ c lost to ~~E~~ e. In graph terms, proving the existence of such a ranking amounts to proving that every tournament graph has a Hamiltonian path.

Theorem 2. *Every tournament graph contains a directed Hamiltonian path.*

Proof. We use strong induction. Let $P(n)$ be the proposition that every tournament graph with n vertices contains a directed Hamiltonian path.

Base case. $P(1)$ is trivially true; every graph with a single vertex has a Hamiltonian path consisting of only that vertex.

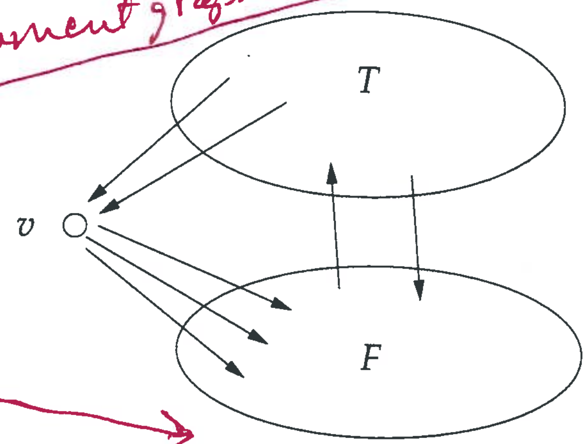
Inductive step. For $n \geq 1$, we assume that $P(1), \ldots, P(n)$ are all true and prove $P(n+1)$. Consider a tournament graph $G = (V, E)$ with $n+1$ players. Select one vertex v arbitrarily. Every other vertex in the tournament either has an edge *to* vertex v or an edge *from* vertex v. Thus, we can partition the remaining vertices into two corresponding sets, T and F, each containing at most n vertices, where

$T = \{u \mid u \to v \in E\}$ and $F = \{u \mid v \to u \in E\}$.

For example, see Figure EE2.

EE-3

Figure EE2: The sets T and F in a tournament graph.



~~Such an approach works well in a tournament for every player v, the players with the best record either defeats v directly or defeats another player that defeats v.~~

The vertices in T together with the edges that join them form a smaller tournament. Thus, by strong induction, there is a Hamiltonian path within T. Similarly, there is a Hamiltonian path within the tournament on the vertices in F. Joining the path in T to the vertex v followed by the path in F gives a Hamiltonian path through the whole tournament. (As special cases, if T or F is empty, then so is the corresponding portion of the path.) □

For example, in the tournament associated with Figure EE1,

The ranking defined by a Hamiltonian path is not entirely satisfactory. ~~In the example tournament~~, notice that the lowest-ranked player (~~f~~) actually defeated the highest-ranked ~~player (f)!~~ ~~Maybe it would be better to order the players according to the number of victories they attained as is usually done in practice.~~

~~May be there is another way to analyze tournaments. As the following section explains, there is always a "best" player who is able to defeat any given player or some other player who defeats the given player.~~

— INSERT EF Goes here —

6.2.2 ~~6~~ The King Chicken Theorem

Suppose that there ~~Consider the following situation. There~~ are n chickens in a farmyard. For each pair of distinct chickens, either the first pecks the second or the second pecks the first, but not both. We say that chicken u *virtually pecks* chicken v if either:

- Chicken u pecks chicken v, or
- Chicken u pecks some other chicken w who in turn pecks chicken v.

A chicken that virtually pecks every other chicken is called a *king chicken*.

We can model this situation with a tournament digraph. The vertices are chickens, and an edge $u \to v$ indicates that chicken u pecks chicken v. In the tournament ~~below~~ shown in Figure EE3, three of the four chickens are kings.

Chicken c is not a king in this example since it does not peck chicken b and it does not peck any chicken that pecks chicken b. Chicken a is a king since it pecks chicken d, who in turn pecks chickens b and c.

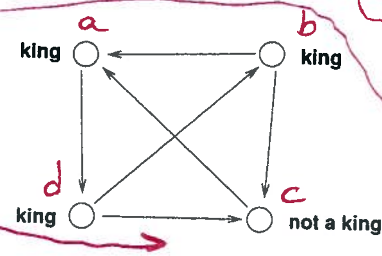
INSERT E~~&~~F

~~In practice, players are typically ranked~~

player (a). ~~&~~ &

In practice, players are typically ranked according to how many victories they achieve. This makes sense for several reasons. One not-so-obvious reason is that ~~the~~ ~~because~~ if the player with the most victories ~~will always is guaranteed to have either~~ ~~beaten~~ or ~~every other player directly or~~ ~~to have be~~ does not beat some other player v, he is guaranteed to have at least beaten a third player ~~that~~ who beat v. We prove this fact in the next section.

Figure EE3: A 4-chicken tournament in which ~~there are 3 king chickens~~ chickens a, b, and d are kings.



~~Now we're going to prove that a chicken tournament always results in at least one chicken king.~~

Theorem 3 (King Chicken Theorem). *The chicken with highest outdegree in an n-chicken tournament with $n \geq 1$ is king.*

$G = (V, E)$

Proof. (By contradiction.) Let u be the node in the tournament graph with highest outdegree and suppose that u is not king. ~~Let Y be the set of nodes to which u has directed edges. Then, the number of nodes in Y is equal to the outdegree of u; $v \in Y$ iff $u \to v$. Since u is not king, there exists a node x such that~~

- there is not a directed edge from u to x, and
- for all nodes v with a directed edge $u \to v$ (that is, $v \in Y$), there is not a directed edge from v to x.

Since in a tournament there exists exactly one directed edge between any two nodes, these two conditions are equivalent to

- there is a directed edge from x to u, and
- for all nodes $v \in Y$, there is a directed edge from x to v.

In other words, the outdegree of x is at least one more than the number of nodes in Y. So, the outdegree of x is at least one more than the outdegree of u. But u is a node with highest outdegree. Contradiction! □

~~EASE~~

~~Let $Y = \{v \mid u \to v \in E\}$~~

Let $Y = \{v \mid u \to v \in E\}$ be the set of chickens that chicken u pecked. Then $\deg(u) = |Y|$.

Since u is not a king, there is a ~~node~~ chicken ~~such that~~ ~~that defeated chicken u and for which~~ $x \notin Y$ (i.e., that ~~defeated~~ was not pecked by chicken u) and ~~which did not lose to any~~ which was not pecked by any chicken in Y. Since ~~then~~ for any pair of chickens, one pecks the other, this means that x pecks u as well as every chicken in Y. This means that ~~$\deg(x) = |Y|+1$~~

$$
\begin{aligned}
\deg(x) &= |Y| + 1 \\
&= \deg(u) + 1 \\
&> \deg(u).
\end{aligned}
$$

But u was assumed to be the node with the highest degree in the tournament, so we have a contradiction. Hence, u must be a king. ∎

Theorem 3 means that if the player with the most victories is defeated by another player x, then at least he/she defeats some third player that defeats x. In this sense, the player with the most victories has some sort of bragging rights over every other player. Unfortunately, as Figure EE3 illustrates, there can be many other players with ~~bra~~ such bragging rights, even some with fewer victories. Indeed, for some tournaments, it is possible that every player is a "king." ~~Fora~~ example, consider the tournament illustrated in Figure EE4.

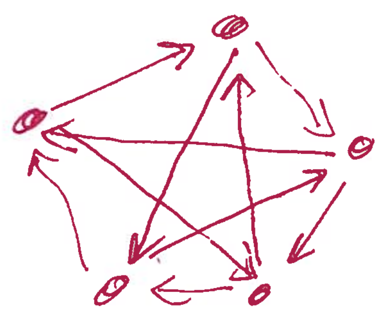


Figure EE4: A 5-chicken tournament in ~~a~~ ~~Dia~~ which every chicken is a king.

INSERT EH

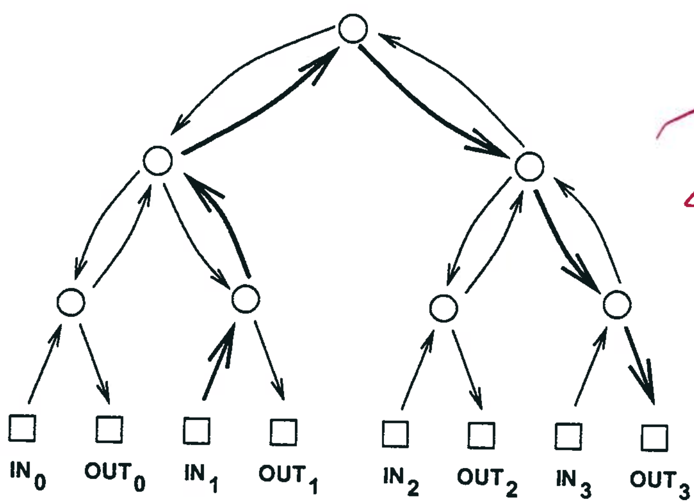
~~Ro~~ ~~Rea~~ While reasoning about chickens pecking each other may be amusing (to mathematicians, at least), ~~directed~~ the use of directed graphs to model communication networks ~~problems~~ is very serious business. ~~In~~ the context of communication problems, contrast, find application in telephone switching systems and the communication hardware inside parallel computers. ~~In this chapter~~ section, ~~we'll look at some of the nicest and most~~ commonly used structured networks ~~and we'll see how~~ ~~For communication. analyze how good they are at supporting communications.~~

— INSERT EJ goes here —

~~12.2~~ 6.3.2 The Complete Binary Tree

subsection

One of the simplest structured communications networks is a ~~Let's start with a~~ *complete binary tree*. ~~Here is an example~~ A complete binary tree with 4 inputs and 4 outputs is shown in Figure EH.



David: remove bold from the edge

Figure EH: A 4-input, 4-output complete binary tree. The squares represent terminals (input/output registers) and the circles represent switches. Directed edges represent communication channels in the network through which data packets can move. The unique path from input 1 to output 3 is shown in bold.

~~The kinds of communication networks we consider aim to transmit packets of data between computers, processors, telephones, or other devices. The term *packet*~~

~~this goes to footnote 1 on the next page.~~

INSERT EJ

6.3.1 ~~The~~ Packet Routing

Whatever ~~the~~ architecture ~~a~~ is chosen, the goal of a communication network is to get data from inputs to outputs. In this ~~section~~ text, we will focus on a model in which ~~packets of data~~ the data to be communicated is in the form of a packet. In practice, a packet would consist of a fixed amount of data and a message (such as a web page or a movie) would consist of many packets.

For simplicity, we will restrict our attention to the scenario where there is just one packet at every input and where there is just one packet destined for each output. We will denote the number of inputs and outputs by N and we will ~~usu~~ often assume that N is a power of two.

We will ~~~~ ~~also~~ specify the desired destinations of the packets — INSERT EK goes here (it is text on p 526

~~Let's see how to solve a packet~~

~~all this works by means of an~~

~~ek~~

Of course, the goal is to get all the packets to their destinations as quickly as possible using as little hardware as possible. The time needed to get the packets to their destinations ~~will~~ depends on several factors, such as how many switches they need to go through, and how ~~much~~ many packets will ~~need to~~ ~~want~~ need to cross the same ~~edge~~ wire. ~~Edges~~ (we will assume that only one packet can cross a wire at a time.) The complexity of the hardware depends on factors such as the number of switches needed and the size of the switches.

Let's see how all this works with an example — routing packets on a complete binary tree.

this is footnote 1

~~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* (i.e., the inputs and outputs) and the ~~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 in Figure E.H.

~~12.3 Routing Problems~~

~~Communication networks are supposed to get packets of data 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 net-work designs, where each network has N inputs and N outputs; for convenience, we'll assume N is a power of two.~~

This is insert E K & goes on insert E I

~~Which input is supposed to go where is specified~~ by a permutation[1] of $\{0, 1, \ldots, N-1\}$. So a permutation, π, defines a *routing problem*: get a packet that starts at input i to output $\pi(i)$ for $0 \le i < N$. 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 $\pi(i)$.

6.2.3 ~~12.4~~ Network Diameter

The delay between the time that a packets arrives at an input and arrives at its designated output is known as latency and it is a critical issue in communication networks. ~~Generally this~~ If congestion is not a factor, then this delay is generally proportional to the length of the path a packet follows. Assuming it takes one time unit to travel across a wire, ~~EDITING NOTE:~~ and that there are no additional delays at switches,

David: insert into text

1 A permutation of a sequence is a reordering of the sequence.

the delay of a packet will be the number of wires it crosses going from input to output.

David: ~~keep~~ put the footnote in the text

~~EDITING NOTE~~

[1]

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[2] is the maximum length of any shortest path between an input and an output. For example, in the complete

[1] Latency ~~is often~~ can also be measured as the number of switches that a packet must pass through when traveling between the most distant input and output, since switches usually have the biggest impact on network speed. For example, in the complete binary tree example, the packet traveling from input 1 to output 3 crosses 5 switches, which is 1 less than the number of edges traversed.

[2] 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.

shown in Figure EH,

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, $2\log(2^{10}) + 2 = 22$.

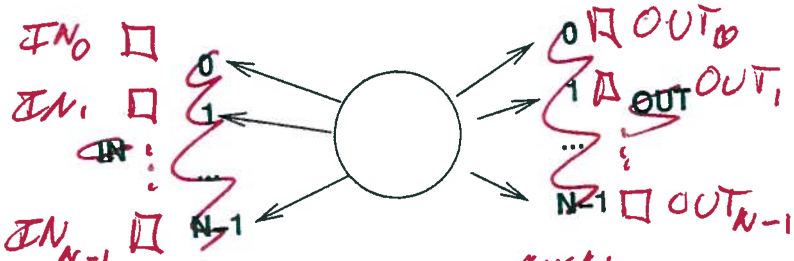
~~make~~ monitor subsection

6.2.4 ~~12.4.1~~ Switch Size

(and hence the latency ~~to rou~~ needed to route packets)

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, ~~as~~ as shown in Figure EI.

In this case, the "network" would consist of a single ~~node~~ switch and the latency would be 2.



~~EDITING NOTE:~~

Figure EI: A 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.

6.2.5 ~~12.5~~ Switch Count ⇐ subsection

A Key ~~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 ~~$1 + 2 + 4 + 8 + \cdots + N$,~~ $1 + 2 + 4 + \cdots + N = 2N - 1$, since there is 1 switch at the top (the "root switch"), 2 below it, 4 below those, and

This is nearly the

so forth. ~~By the formula (??) for geometric sums, the total number of switches is~~

~~$2N - 1$, which is nearly the~~ best possible with 3×3 switches, since at least one switch will be needed for each pair of inputs and outputs.

~~12.6 Network Latency~~ subsection 12.6.6

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.~~

6.2.6 ~~12.7~~ Congestion subsection

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, $\text{Id}(i) ::= i$, 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.~~ in Figure EJ.

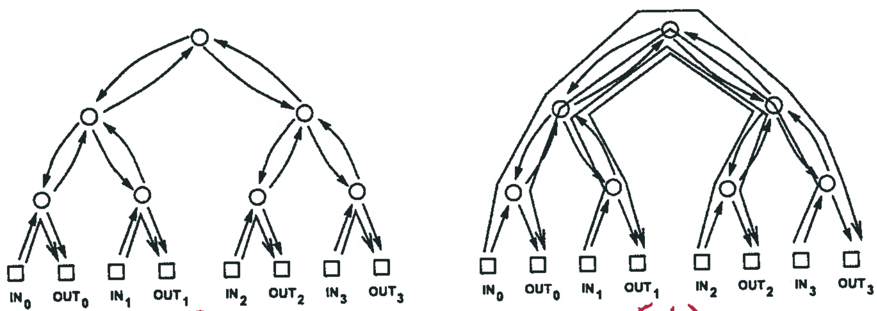


Figure EJ: Paths for the routing problem given by $\pi(i) = i$ (a) and $\pi(i) = N-1-i$ (b). The root is a bottleneck in the latter scenario.

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~~ in Figure EJ(a) is 1, since at most 1 path passes through each switch. However, the congestion of the routing ~~on the right~~ in Figure EJ(b) 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 "maxi-min" ~~"maximin"~~ congestion is called the *congestion of the network*.

~~EDITING NOTE:~~ put this in the text

You may find it helpful to think about max congestion in terms of a value game. You design your spiffy, new communication network; this defines the game. Your opponent makes the first move in the game: she inspects your network and specifies a permutation routing problem that will strain your network.You move second: given her specification, you choose the precise paths that the packets should take through your network; you're trying to avoid overloading any one switch. Then her next move is to pick a switch with as large as possible a number of packets passing through it; this number is her score in the competition. The max congestion of your network is the largest score she can ensure; in other words, it is precisely the max-value of this game.

For example, if your enemy were trying to defeat the complete binary tree, she would choose a permutation like $\pi(i) = (N-1) - i$. Then for *every* packet i, you would be forced to select a path $P_{i,\pi(i)}$ passing through the root switch. ~~Thus, the~~ Then, your enemy would choose the root switch and achieve a score of N. In other words, the max congestion of the complete binary tree is N— which is horrible!

We have summarized the results of our analysis for the complete binary tree in Figure EK.

~~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!~~

Overall, the complete binary tree does well in every category except the last—congestion, and that is a killer in practice. Next, will look at a network that solves the congestion problem, but at a very high cost.

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

~~How~~

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

Figure EK: A summary of the attributes of the complete binary tree.

6.2.7 ~~12.8~~ The 2-~~D~~d Array

An illustration of the N×N 2-d array (aka, grid or crossbar) is shown in Figure EL for the case when N=4.

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

~~EDITING NOTE: or crossbar.~~

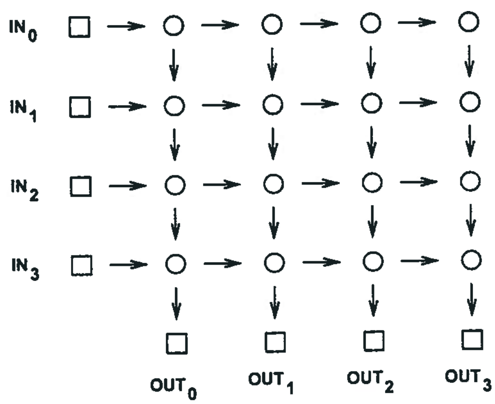


Figure EL: A 4x4 2-dimensional array.

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

The diameter of the 4x4 2-d array ~~in this example~~ is 8, which is the number of edges between input 0 and output 3. More generally, the diameter of ~~an~~ a 2-d 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 ($2\log N + 2$). On the other hand, replacing a complete binary tree with ~~an~~ a 2-d array almost eliminates congestion.

Theorem 12.8.1. *The congestion of an N-input* 2-d *array is 2.*

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~~ In this solution, 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. ■

The characteristics of the 2-d array are

~~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.~~ recorded in Figure EM.

~~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

Figure EM: Comparing the N-input 2-d array to the N-input complete binary tree.

no new # (continue from previous sentence)

The crucial entry ~~here~~ in this table is the number of switches, which is N^2. This is a major defect of the 2-~~D~~ d array; a network ~~of size~~ with $N = 1000$ inputs 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.

6.2.8 ~~12.9~~ The Butterfly

← subsection

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 butterfly network with $N = 8$ inputs is shown in Figure EN.

~~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 12.1.~~

INSERT BP goes here
(this is text from ~~latest~~ lecture notes) ~~from~~ on communication networks (date is wrong on notes)

INSERT EP

EP-1

for applications where N is small, the simplicity and low congestion of the array make it an attractive choice.

3 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. ~~Here is a~~ A butterfly network with $N = 8$ inputs and outputs. is shown in Figure EN.

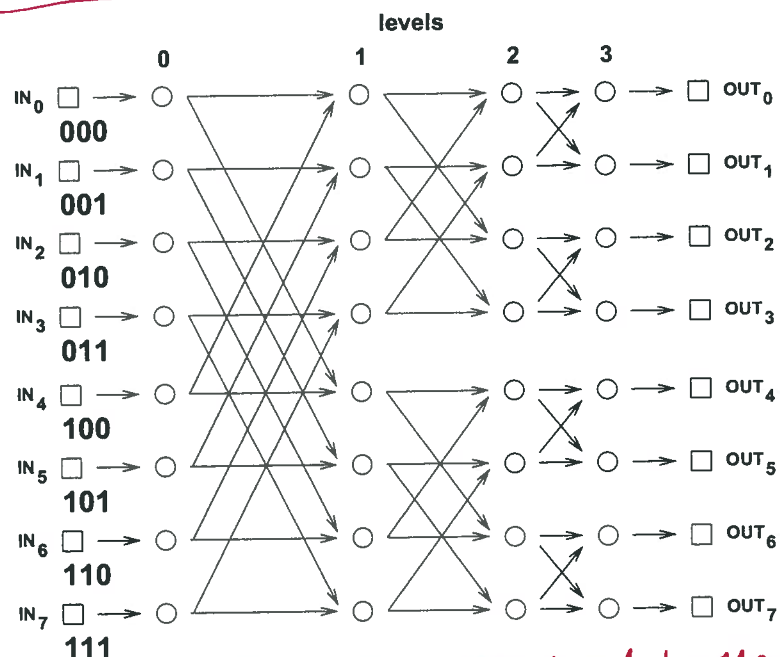


Figure EN: An 8-input/output butterfly.

The structure of the butterfly is certainly more complicated than that of the complete binary tree or 2-~~D~~d array. Let's ~~work through the various parts of the butterfly.~~ see how it is constructed.

All the terminals and switches in the network are arranged in N rows. In particular, input i is at the left end of row i, and output i is at the right end of row i. Now let's label the rows in *binary*~~; thus,~~ so that the label on row i is the binary number $b_1 b_2 \ldots b_{\log N}$ that represents the integer i.

Between the inputs and the outputs, there are $\log(N)+1$ levels of switches, numbered from 0 to $\log N$. Each level consists of a column of N switches, one per row. Thus, each switch in the network is uniquely identified by a sequence $(b_1, b_2, \ldots, b_{\log N}, l)$, where $b_1 b_2 \ldots b_{\log N}$ is the switch's row in binary and l is the switch's level.

All that remains is to describe how the switches are connected up. The basic connection

EP-2

level 0, row 0 in

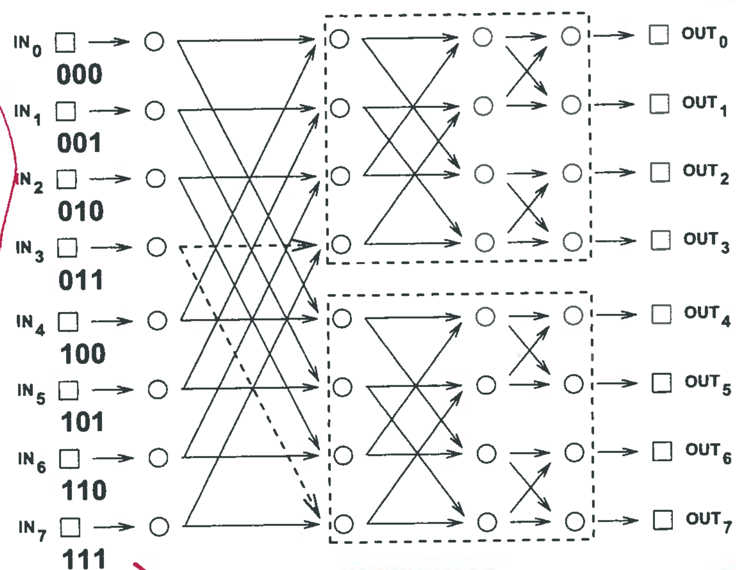
pattern is expressed below in a compact notation:

$$(b_1, b_2, \ldots, b_{l+1}, \ldots, b_{\log N}, l) \begin{array}{l} \nearrow (b_1, b_2, \ldots, b_{l+1}, \ldots, b_{\log N}, l+1) \\ \searrow (b_1, b_2, \ldots, \overline{b_{l+1}}, \ldots, b_{\log N}, l+1) \end{array}$$

This says that there are directed edges from switch $(b_1, b_2, \ldots, b_{\log N}, l)$ to two switches in the next level. One edge leads to the switch in the *same* row, and the other edge leads to the switch in the row obtained by *inverting* bit $l+1$. For example, referring back to the illustration of the size $N = 8$ butterfly, there is an edge from switch $(0, 0, 0, 0)$ to switch $(0, 0, 0, 1)$, which is in the same row of level 1, and to switch $(1, 0, 0, 1)$, which is in the row obtained by inverting bit $l + 1 = 1$.

The butterfly network has a recursive structure; specifically, a butterfly of size $2N$ consists of two butterflies of size N, which are shown in dashed boxes ~~below~~ in Figure EP, and one additional level of switches. Each switch in the new level has directed edges to a pair of corresponding switches in the smaller butterflies; one example is dashed in the figure.

Figure EP: An N-input butterfly contains two $\frac{N}{2}$-input butterflies.



~~through its switches~~

Despite the relatively complicated structure of the butterfly, there is a simple way to route packets. In particular, suppose that we want to send a packet from input $x_1 x_2 \ldots x_{\log N}$ to output $y_1 y_2 \ldots y_{\log N}$. (Here we are specifying the input and output numbers in binary.) Roughly, the plan is to "correct" the first bit ~~by~~ on level 1, correct the second bit ~~by~~ on level 2, and so forth. Thus, the sequence of switches visited by the packet is:

$$\begin{aligned}(x_1, x_2, x_3, \ldots, x_{\log N}, 0) &\rightarrow (y_1, x_2, x_3, \ldots, x_{\log N}, 1) \\ &\rightarrow (y_1, y_2, x_3, \ldots, x_{\log N}, 2) \\ &\rightarrow (y_1, y_2, y_3, \ldots, x_{\log N}, 3) \\ &\rightarrow \quad \ldots \\ &\rightarrow (y_1, y_2, y_3, \ldots, y_{\log N}, \log N)\end{aligned}$$

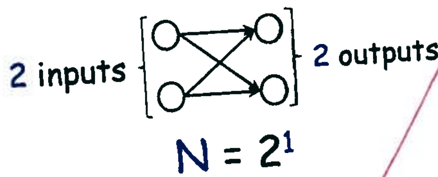
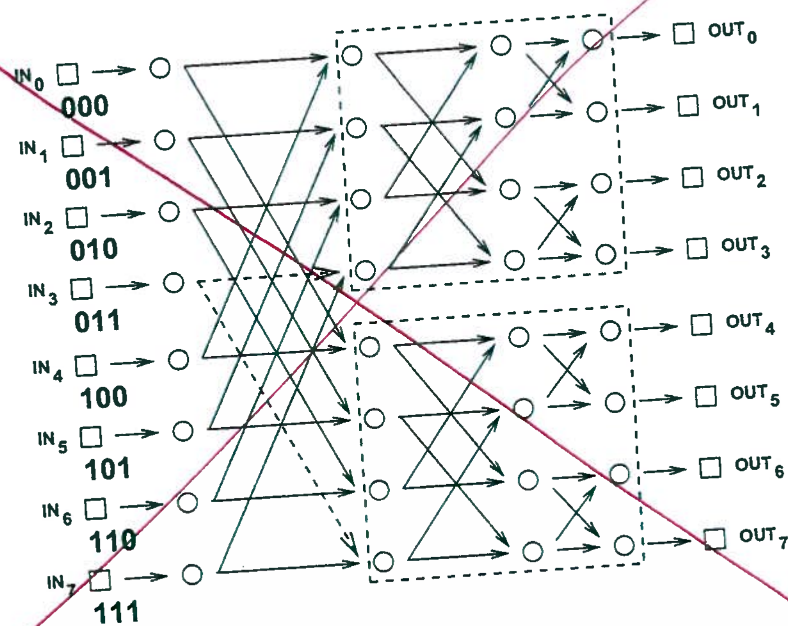


Figure 12.1: F_1, the Butterfly Net switches with $N = 2^1$.

EDITING NOTE:

The butterfly of size $2N$ consists of two butterflies of size N, which are shown in dashed boxes below, and one additional level of switches. Each switch in the new level has directed edges to a pair of corresponding switches in the smaller butterflies; one example is dashed in the figure.



David – keep this text ↓

Despite the relatively complicated structure of the butterfly, there is a simple way to route packets through its switches. In particular, suppose that we want to send a packet from input $x_1x_2\ldots x_{\log N}$ to output $y_1y_2\ldots y_{\log N}$. (Here we are specifying the input and output numbers in binary.) Roughly, the plan is to "correct" the first bit ~~by~~ on level 1, correct the second bit ~~by~~ on level 2, and so forth. Thus, the sequence of switches visited by the packet is:

$$
\begin{aligned}
(x_1, x_2, x_3, \ldots, x_{\log N}, 0) &\to (y_1, x_2, x_3, \ldots, x_{\log N}, 1) \\
&\to (y_1, y_2, x_3, \ldots, x_{\log N}, 2) \\
&\to (y_1, y_2, y_3, \ldots, x_{\log N}, 3) \\
&\to \quad \ldots \\
&\to (y_1, y_2, y_3, \ldots, y_{\log N}, \log N)
\end{aligned}
$$

In fact, this is the *only* path from the input to the output!

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 12.2. 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 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.

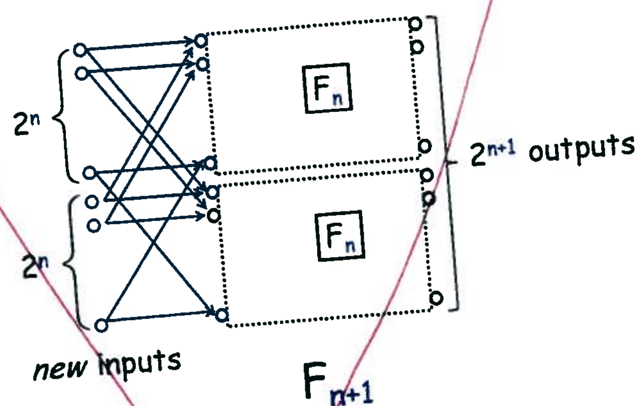


Figure 12.2: F_{n+1}, the Butterfly Net switches with 2^{n+1} inputs and outputs.

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 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~~ The task of proving this fact has been left to the ~~simple proof of this appears in Problem ??.~~ ~~Section~~ problem section.

A comparison of the butterfly with the complete binary tree and 2-d array is provided in Figure EQ. As you can see, the ~~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 $\sqrt{N/2}$

Figure EQ: A comparison of the N-input butterfly with the N-input complete binary tree and the N-input 2-d array.

~~The~~ 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.

6.2.9 The ~~12.10~~ Beneš Network ← subsection

In the 1960's, a researcher at Bell Labs named Václav 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 12.1.~~

~~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 12.3.~~

~~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~~

For example, the 8-input Beneš network is shown in Figure ER.

output switches are connected to the same two switches, namely, to the ith output switches of each of two B_n components.

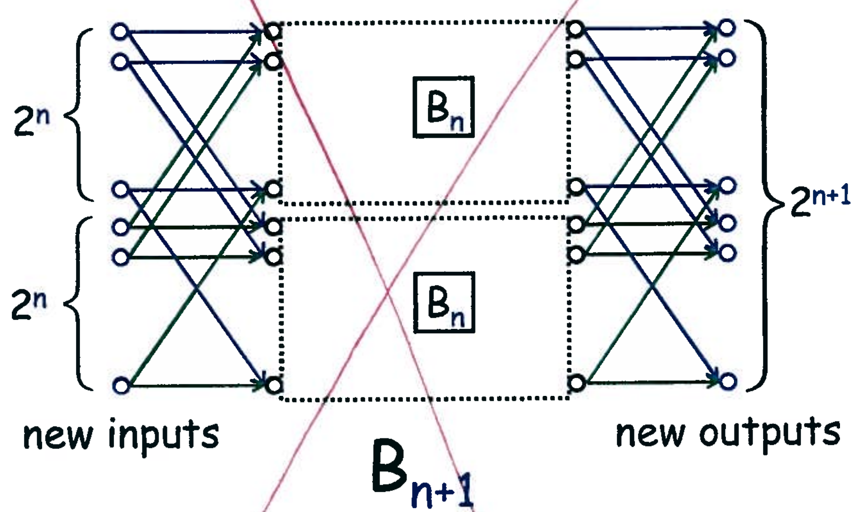


Figure 12.3: B_{n+1}, the Beneš Net switches with 2^{n+1} inputs and outputs.

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.

EDITING NOTE:

DAVID ✓ Keep this here

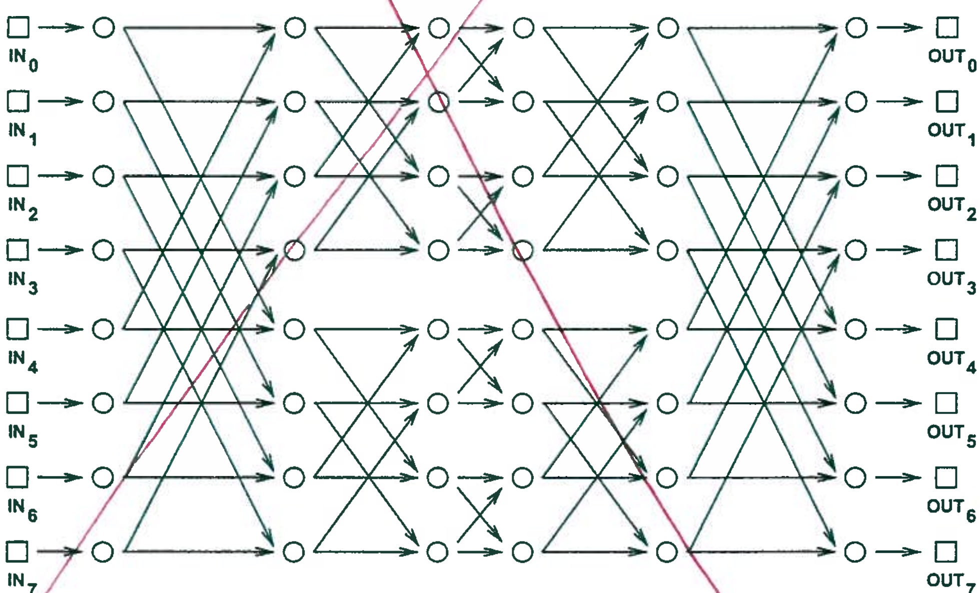


Figure ER: The 8-input Beneš network.

This network now has levels labeled $0, \ldots, 2\log N + 1$. For $1 \leq k \leq \log N$, the connections from level $k-1$ to level k are just as in the Butterfly network,

~~the connections based on bit k. The conections from level $2\log N - k + 1$ to level $2\log N - k + 2$ are also the ones based on bit k. (Informally, to make the connections from level 0 to level $2\log N + 1$ one level at a time, use the connections based on bits $1, 2, 3, \ldots, \log N - 1, \log N, \log N - 1, \log N - 2, \ldots, 3, 2, 1$ in that order.)~~

Putting two butterflies back-to-back doubles

~~So Beneš has doubled~~ the number of switches and the diameter, ~~of course, but~~ of a single butterfly, but it 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 against the other networks we have been studying. As you can see in Figure ES, the

Figure ES: A comparison of the N-input Beneš network with the N-input complete binary tree, 2-d array, and butterfly.

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

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

INSERT EV goes here
(text from some lecture as before)

INSERT EV

~~EV-?~~

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 is small, compact, and completely eliminates congestion. The Holy Grail of routing networks is in hand!

Theorem 2. *The congestion of the N-input Beneš network is 1,* ~~for any a =~~ ~~*where $N = 2^a$ for some $a \geq 1$*~~ for any N that is a power of 2.

Proof. We use induction. Let $P(a)$ be the propositon that the congestion of the ~~size 2^a~~ 2^a-input Beneš network is 1.

Base case $(a = 1)$. We must show that the congestion of the size ~~$N = 2^1 = 2$~~ 2^1-input Beneš network is 1. This network is shown ~~below:~~ in Figure ET.

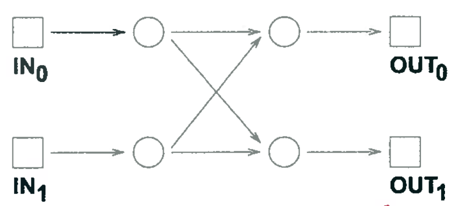


Figure ET: The 2-input Beneš network.

There are only two possible permutation routing problems for a 2-input network. If $\pi(0) = 0$ and $\pi(1) = 1$, then we can route both packets along the straight edges. On the other hand, if $\pi(0) = 1$ and $\pi(1) = 0$, then we can route both packets along the diagonal edges. In both cases, a single packet passes through each switch.

Inductive step. We must show that $P(a)$ implies $P(a + 1)$, where $a \geq 1$. Thus, we assume that the congestion of ~~an N-~~ a 2^a-input Beneš network is 1 in order to prove that the congestion of a ~~$2N$~~ 2^{a+1}-input Beneš network is also 1.

Digression. Time out! Let's work through an example, develop some intuition, and then complete the proof. Notice that inside a Beneš network of size $2N$ lurk two Beneš subnetworks of size N. (This follows from our earlier observation that a butterfly of size $2N$ contains two butterflies of size N.) ~~In the Beneš network shown below, the two subnetworks are in dashed boxes.~~ ~~In Figure ET~~

David – back to p548, continue in same paragraph

~~$P(n)$ ::= the congestion of B_n 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~~ in Figure 6ll with $N = 8$ inputs and outputs, the two 4-input/output subnetworks are in dashed boxes.

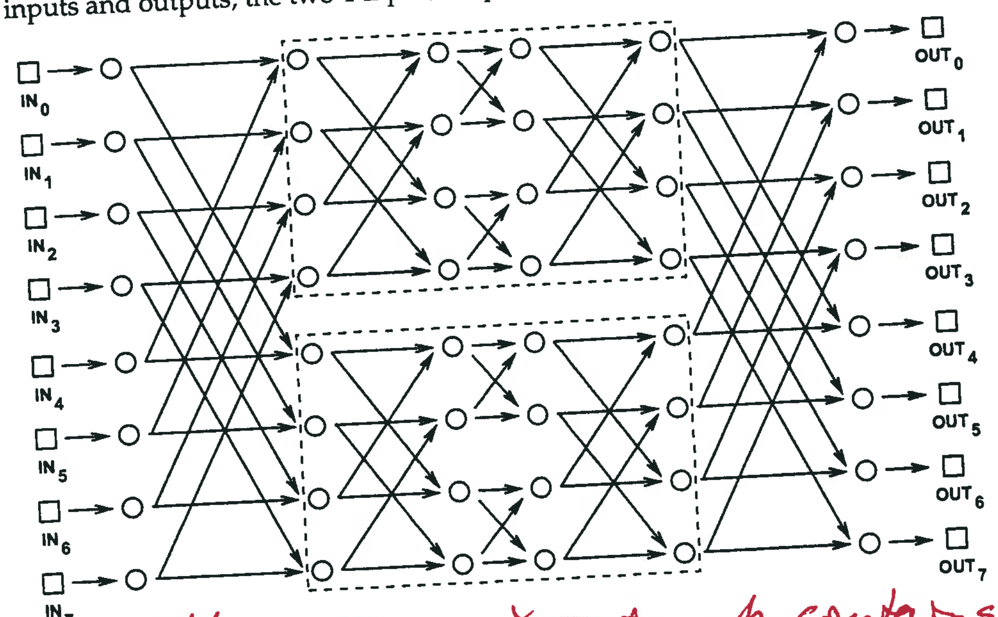


Figure 6ll: An ~~8~~ 2N-input Beneš network contains two N-input Beneš networks — shown here for N=8.

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:

$$\pi(0) = 1 \qquad \pi(4) = 3$$
$$\pi(1) = 5 \qquad \pi(5) = 6$$
$$\pi(2) = 4 \qquad \pi(6) = 0$$
$$\pi(3) = 7 \qquad \pi(7) = 2$$

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:~~ The resulting constraint graph is illustrated in Figure EV.

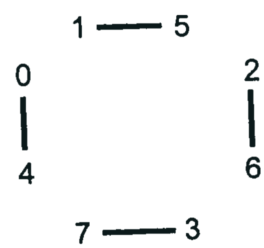


Figure EV: A constraint graph for our ~~graph~~ packet routing problem. Adjacent packets cannot be routed using the same sub-Beneš network.

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 constraint graph with gray edges, as is illustrated in Figure EW.

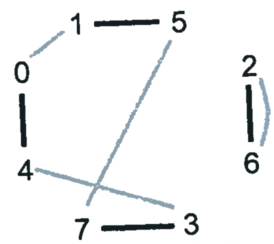


Figure EW: the updated constraint graph.

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: *a 2-coloring of the graph corresponds to a solution to the routing problem*. In particular, 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.

The only remaining question is whether the constraint graph is 2-colorable.

Fortunately, this ~~which~~ is easy to verify:

Lemma 12.10.2. ~~*Prove that if*~~ If *the edges of a graph can be grouped into two sets such that every vertex* ~~*has*~~ is incident to *at most 1 edge from each set* ~~*incident to it*~~*, then the graph is 2-colorable.*

Proof. Since the two sets of edges may overlap, let's call an edge that is in both sets a *doubled edge*.

David – I will add this into chapter 5 on the next pass

We know from Theorem 9.7.2 that all we have to do is show that every ~~cycle~~ closed walk. has even length. There are two cases:

Case 1: [The ~~cycle~~ closed walk 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~~ closed walk 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~~ closed walk 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~~ closed walk must traverse a path of alternating edges that begins and ends with edges from different sets. This means the ~~cycle~~ closed walk has to be of even length. ■

For example, ~~here is~~ a 2-coloring of the constraint graph: In Figure EW is shown in Figure EX.

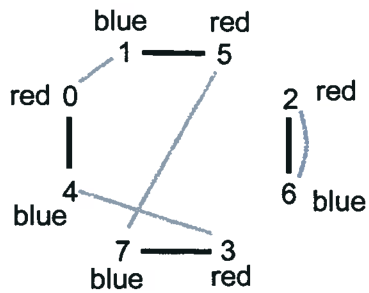


Figure EX: A 2-coloring of the constraint graph in Figure EW.

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 12.10.2, 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 $P(n)$. ■

~~12.10.1~~ 6.4 Problems

~~Exam Problems~~

~~Class Problems~~

~~Homework Problems~~