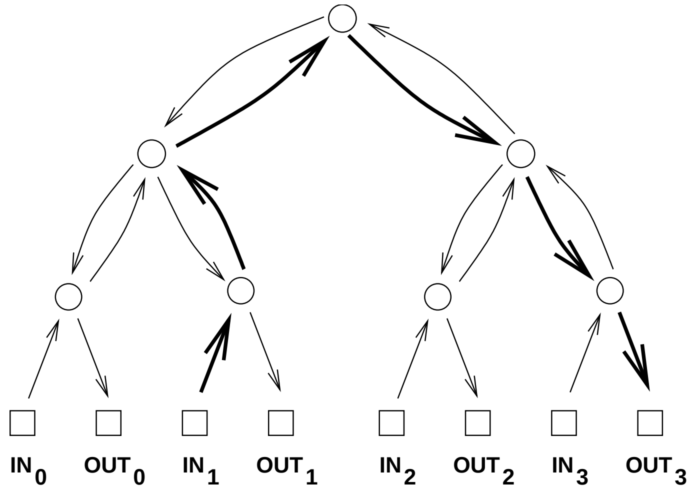
Communication Networks

Today we'll explore an important application of graphs in computer science: modelling communication networks. Generally, vertices will represent computers, processors, 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. However, there do exist more organized networks, such as certain telephone switching networks and the communication networks inside parallel computers. For these, the corresponding graphs are highly structured. In this lecture, we'll look at some of the nicest and most commonly used communication networks.

1 Complete Binary Tree

Let's start with a ***complete binary tree***. Here is an example with 4 inputs and 4 outputs.



The basic function of the communication networks we consider today is 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

[1] This is a relatively new topic in 6.042, so these lecture notes are more likely to contain errors. If you suspect you've found an error or just find something particularly confusing, send email to 6042staff@mit.edu and we'll try to correct the problem.

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 outoing 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 path between every pair of vertices in an undirected 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 analogous directed path. For example, the route of a packet traveling from input 1 to output 3 is shown in bold.

1.1 Latency and Diameter

Latency is a critical issue in communication networks. This is the time required for a packet to travel from an input to an output. In 6.042, we will measure latency by summing the number of wires that need to be crossed.[2]

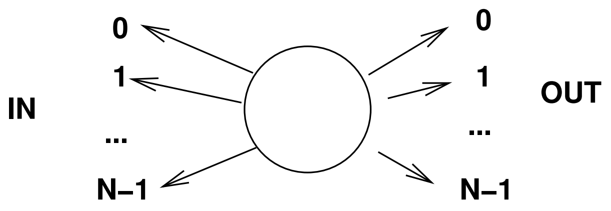
As in an undirected graph, the ***diameter*** of a network is the length of the shortest path between the input and output that are farthest apart. Thus, diameter is an approximate measure of worst-case latency. Notice that the diameter of a *communications network* is defined somewhat differently from the diameter of an *undirected graph*, since in the context of a communication network we're only interested in the distance between inputs and outputs, not between arbitrary pairs of switches. Since input 1 and output 3 are as far apart as possible in the complete binary tree, the diameter is 6.

We're going to consider several different communication networks today. For a fair comparison, let's assume that each network has N inputs and N outputs, where N is a power of two. For example, 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 still have a latency of only $2 \log(2^{10}) + 2 = 22$.

1.2 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 switch:

[2]In non-6.042 settings, latency is often 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.



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 elementary devices, like 3×3 switches. Following this approach, we can build arbitrarily large networks just by adding in more building blocks.

1.3 Switch Count

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

1.4 Congestion

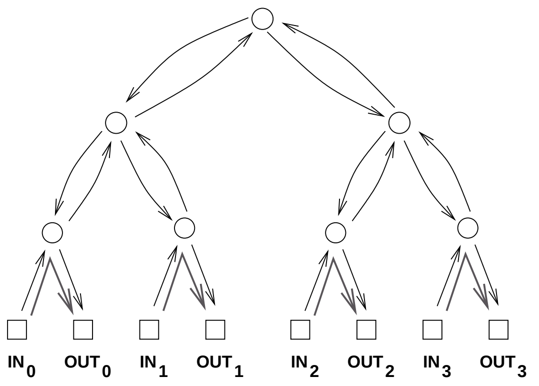
The complete binary tree has a fatal drawback: the root switch is a bottleneck. At best, this switch must handle an enormous amount of traffic: every packet traveling from the left side of the network to the right or vice-versa. Passing all these packets through a single switch could take a long time. At worst, if this switch fails, the network is broken into two equal-sized pieces. We're going to develop a single statistic called "max congestion" that quantifies bottleneck problems in communication networks. But we'll need some preliminary definitions.

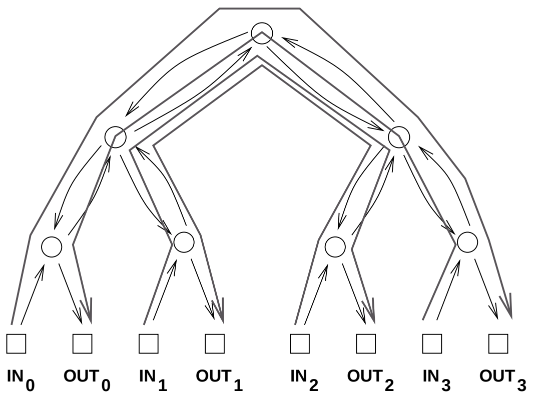
A ***permutation*** is a function π that maps each number in the set $\{0, 1, \ldots, N-1\}$ to another number in the set such that no two numbers are mapped to the same value. In other words, $\pi(i) = \pi(j)$ if and only if $i = j$. For example, $\pi(i) = i$ is one permutation (called the ***identity permutation***) and $\pi(i) = (N-1) - i$ is another.

For each permutation π, there is a corresponding ***permutation routing problem***. In this problem, one packet starts out at each input; in particular, the packet starting at input i is called packet i. The challenge is to direct each packet i through the network from input i to output $\pi(i)$.

A solution to a permutation routing problem is a specification of the path taken by each of the N packets. In particular, the path taken by packet i from input i to output $\pi(i)$

is denoted $P_{i,\pi(i)}$. For example, if $\pi(i) = i$, then there is an easy solution: let $P_{i,\pi(i)}$ be the path from input i up through one switch and back down to output i. On the other hand, if $\pi(i) = (N-1) - i$, then each path $P_{i,\pi(i)}$ must begin at input i, 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 set of paths $P_{0,\pi(0)}, \ldots, P_{N-1,\pi(N-1)}$ is equal to the largest number of paths that pass through a single switch. For example, the congestion of the set of paths in the diagram at left is 1, since at most 1 path passes through each switch. However, the congestion of the paths 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, we can also distinguish between "good" and "bad" networks with respect to bottleneck problems. The ***max congestion*** of a network is that *maximum* over all permutations π of the *minimum* over all paths $P_{i,\pi(i)}$ of the congestion of the paths.

You may find it easier to think about max congestion in terms of a competition. Imagine that you've designed a spiffy, new communication network. Your worst enemy devises a permutation routing problem; that is, she decides which input terminal sends a packet to which output terminal. However, you are free to choose the precise path that each packet takes through your network so as to avoid overloading any one switch. Assuming that you both do your absolute best, the largest number of packets that end up passing through any switch is the max congestion of the network.

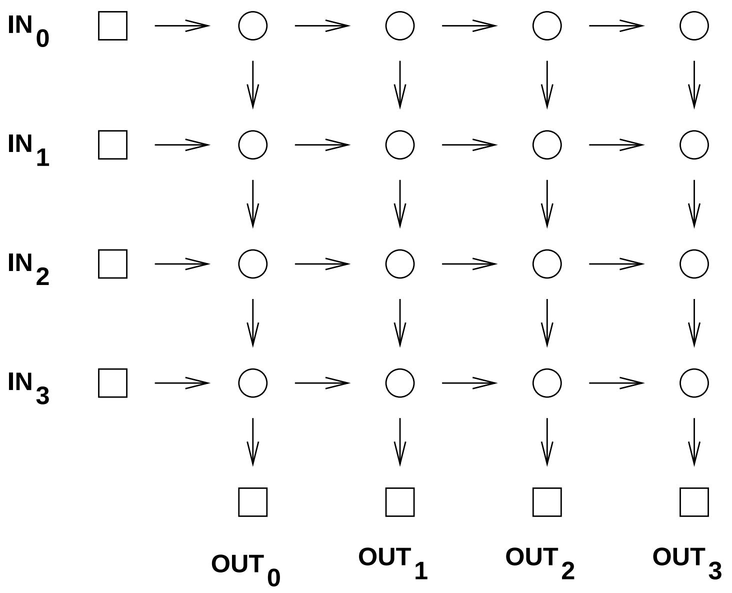
For example, if your enemy were trying to defeat the complete binary tree, she would choose a permutation like $\pi(i) = (N-1) - i$. Then for *every* packet i, you would be forced to select a path $P_{i,\pi(i)}$ passing through the root switch. Thus, the max congestion of the complete binary tree is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

2 2-D Array

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



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

Proof. First, we show that the congestion is at most 2. Let π be any permutation. Define $P_{i,\pi(i)}$ to be the path extending from input i rightward to column $\pi(i)$ and then downward to output $\pi(i)$. Thus, the switch in row i and column $\pi(i)$ transmits at most two packets: the packet originating at input i and the packet destined for column $\pi(i)$.

Next, we show that the congestion is at least 2. In any permutation routing problem where $\pi(0) = 0$ and $\pi(N-1) = N-1$, two packets must pass through the lower left switch. $\square$

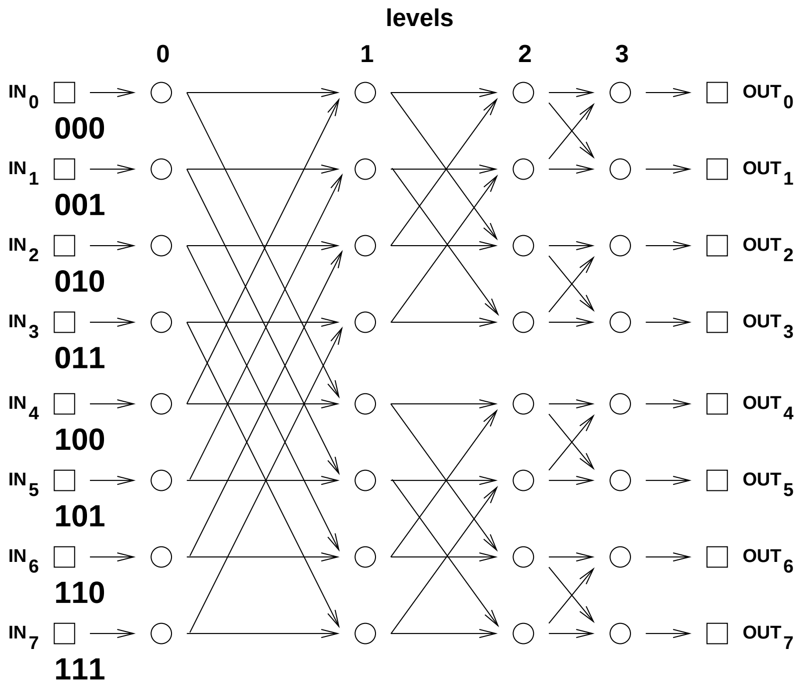
Now we can record the characteristics of the 2-D array.

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.

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 butterfly network with $N = 8$ inputs and outputs.



The structure of the butterfly is certainly more complicated than that of the complete binary tree or 2-D array! Let's work through the various parts of the butterfly.

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, the label on row i is the binary number $b_1 b_2 \ldots b_{\log N}$ that represents the integer i.

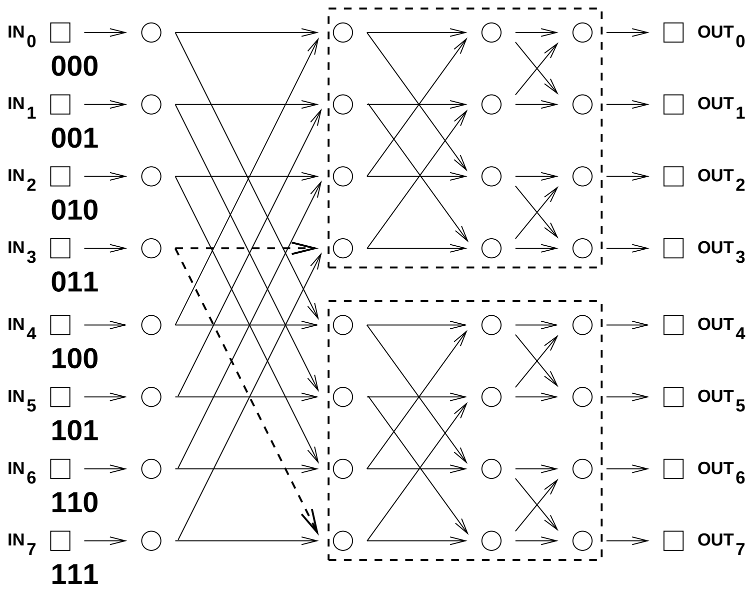
Between the inputs and the outputs, there are $\log(N) + 1$ levels of switches, numbered from 0 to $\log N$. Each level consists of a column of N switches, one per row. Thus, each switch in the network is uniquely identified by a sequence $(b_1, b_2, \ldots, b_{\log N}, l)$, where $b_1 b_2 \ldots b_{\log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

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

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, and to switch $(1, 0, 0, 1)$, which 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, 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.



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 level 1, correct the second bit by level 2, and so forth. Thus, the sequence of switches visited by the packet is:

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

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

The congestion of the butterfly network turns out to be around $\sqrt{N}$; 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. (You'll prove this fact for homework.)

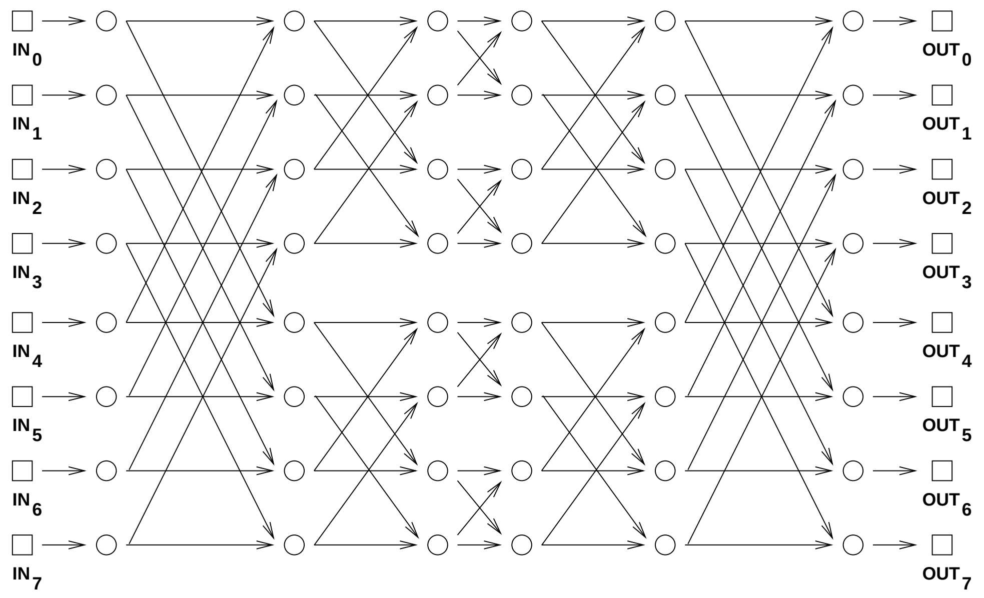
Let's add the butterfly data to our comparison table:

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 compromise somewhere between the two. So our quest for the Holy Grail of routing networks goes on.

4 Beneš Network

In the 1960's, a researcher at Bell Labs named Beneš had a remarkable idea. He noticed that by placing *two* butterflies back-to-back, he obtained a marvelous communication network:



This doubles 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:

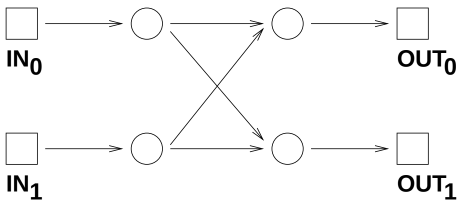
network	**diameter**	**switch size**	**# switches**	**congestion**
complete binary tree	$2\log N + 2$	3×3	$2N - 1$	N
2-D array	$2N$	2×2	N^2	2
butterfly	$\log N + 2$	2×2	$N(\log(N) + 1)$	$\sqrt{N}$ or $\sqrt{N/2}$
Beneš	$2\log N + 1$	2×2	$2N \log N$	1

The Beneš network 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, where $N = 2^a$ for some $a \geq 1$.*

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

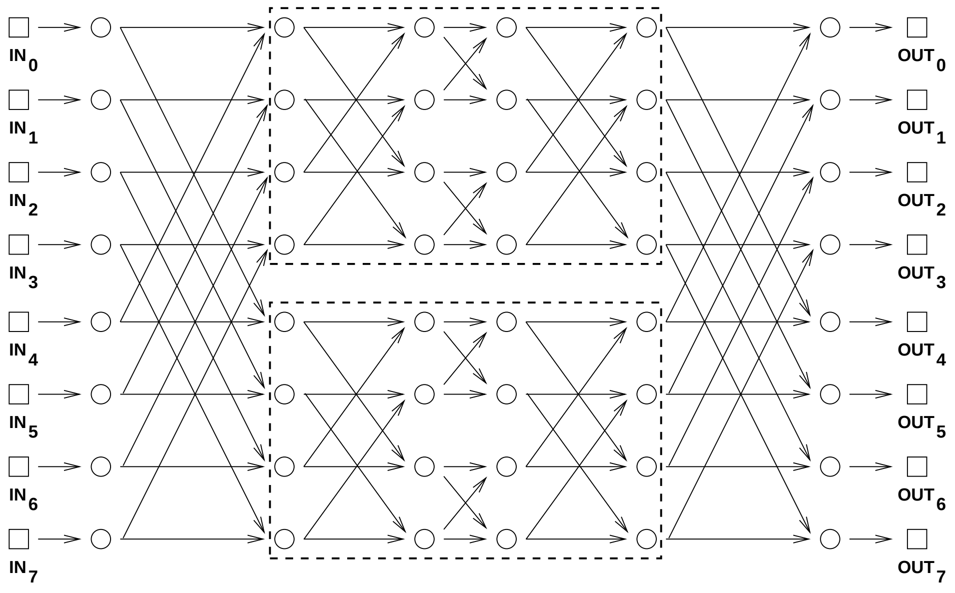
Base case. We must show that the congestion of the size $N = 2^1 = 2$ Beneš network is 1. This network is shown below:



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-input Beneš network is 1 in order to prove that the congestion of a $2N$-input Beneš network is also 1.

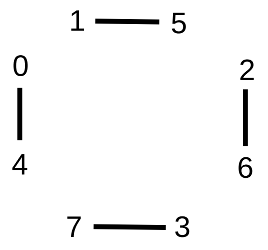
Digression. Time out! Let's work through an example, develop some intuition, and then complete the proof. Notice that inside a Beneš network of size $2N$ lurk two Beneš subnetworks of size N. (This follows from our earlier observation that a butterfly of size $2N$ contains two butterflies of size N.)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



By the inductive assumption, the subnetworks can each route an arbitrary permution 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:

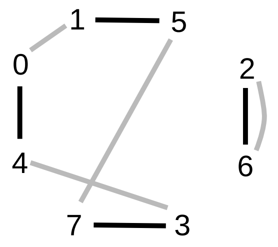
$$\begin{aligned} \pi(0) &= 1 & \pi(4) &= 3 \\ \pi(1) &= 5 & \pi(5) &= 6 \\ \pi(2) &= 4 & \pi(6) &= 0 \\ \pi(3) &= 7 & \pi(7) &= 2 \end{aligned}$$

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:



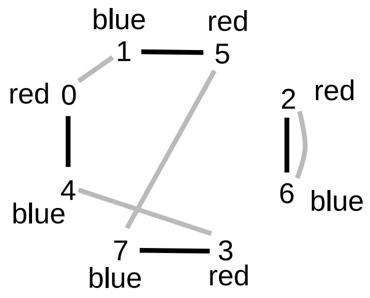
Notice that at most one new edge is incident to each vertex. The two lines drawn between vertices 2 and 6 reflect the two different reasons why these packets must be routed through different networks. However, we intend this to be a simple graph; the two lines still signify a single edge.

Now here's the key insight: *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. This follows from a fact proved in homework:

Theorem 3. *If the graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ both have maximum degree 1, then the graph $G = (V, E_1 \cup E_2)$ is 2-colorable.*

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 $\{0, 1, \ldots, 2N-1\}$. Let $G_1 = (V, E_1)$ be a graph where the vertices are packets 0, 1, . . . , $2N-1$ and there is an edge u—v if $|u-v| = N$. Let $G_2 = (V, E_2)$ be a graph with the same vertices and an edge u—v if $|\pi(u) - \pi(v)| = N$. By Theorem 3, the graph $G = (V, E_1 \cup E_2)$ is 2-colorable, so color the vertices red and blue. Route red packets through the upper subnetwork and blue packets through the lower subnetwork. Since for each edges 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 edges 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 $P(a)$. $\square$

In recitation, you will work through an example in which you route packets using this recursive idea!