Mathematical Induction

Part Two

Recap from Last Time

Let P be some predicate. The ***principle of mathematical induction*** states that if

If it starts true…

$$P(0) \text{ is true}$$

…and it stays true…

and

$$\forall k \in \mathbb{N}.\ (P(k) \to P(k+1))$$

then

$$\forall n \in \mathbb{N}.\ P(n)$$

…then it's always true.

New Stuff!

Theorem: The sum of the first n powers of two is $2^n - 1$.

Proof: Let $P(n)$ be the statement "the sum of the first n powers of two is $2^n - 1$." We will prove, by induction, that $P(n)$ is true for all $n \in \mathbb{N}$, from which the theorem follows.

For our base case, we need to show $P(0)$ is true, meaning that the sum of the first zero powers of two is $2^0 - 1$. Since the sum of the first zero powers of two is zero and $2^0 - 1$ is zero as well, we see that $P(0)$ is true.

For the inductive step, assume that for some arbitrary $k \in \mathbb{N}$ that $P(k)$ holds, meaning that

$$2^0 + 2^1 + \ldots + 2^{k-1} = 2^k - 1. \qquad (1)$$

We need to show that $P(k + 1)$ holds, meaning that the sum of the first $k + 1$ powers of two is $2^{k+1} - 1$. To see this, notice that

$$\begin{aligned} 2^0 + 2^1 + \ldots + 2^{k-1} + 2^k &= (2^0 + 2^1 + \ldots + 2^{k-1}) + 2^k \\ &= 2^k - 1 + 2^k \qquad \textit{(via (1))} \\ &= 2(2^k) - 1 \\ &= 2^{k+1} - 1. \end{aligned}$$

Therefore, $P(k + 1)$ is true, completing the induction. ■

Induction in Practice

- Typically, a proof by induction will not explicitly state $P(n)$.
- Rather, the proof will describe $P(n)$ implicitly and leave it to the reader to fill in the details.
- Provided that there is sufficient detail to determine
  - what $P(n)$ is;
  - that $P(0)$ is true; and that
  - whenever $P(k)$ is true, $P(k+1)$ is true,

  the proof is usually valid.

Theorem: The sum of the first n powers of two is $2^n - 1$.

***Proof*:** By induction.

For our base case, we'll prove the theorem is true when $n = 0$. The sum of the first zero powers of two is zero, and $2^0 - 1 = 0$, so the theorem is true in this case.

For the inductive step, assume the theorem holds when $n = k$ for some arbitrary $k \in \mathbb{N}$. Then

$$\begin{aligned} 2^0 + 2^1 + \ldots + 2^{k-1} + 2^k &= (2^0 + 2^1 + \ldots + 2^{k-1}) + 2^k \\ &= 2^k - 1 + 2^k \\ &= 2(2^k) - 1 \\ &= 2^{k+1} - 1. \end{aligned}$$

So the theorem is true when $n = k+1$, completing the induction. ■

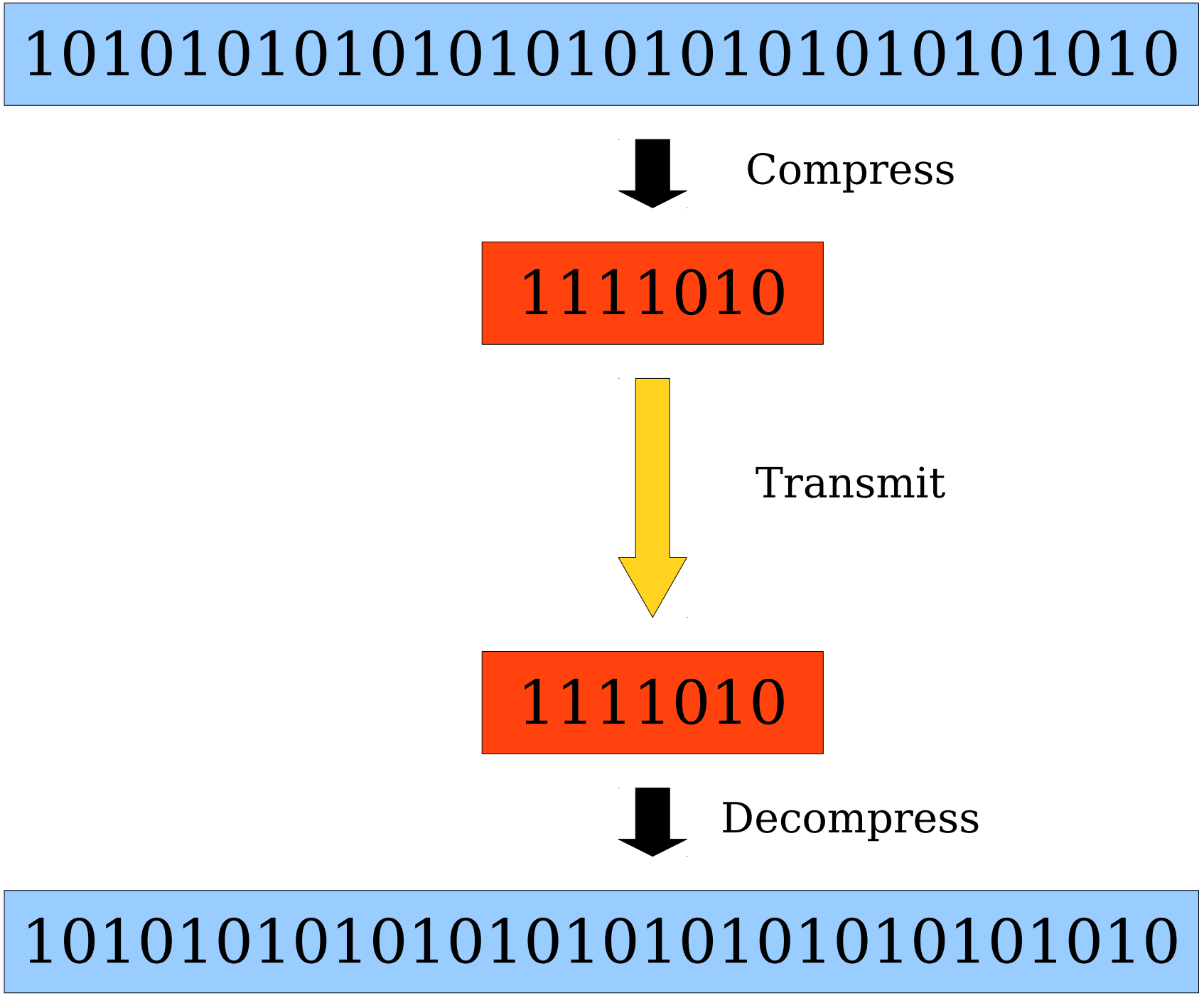
A Fun Application:
The Limits of Data Compression

Bitstrings

- A ***bitstring*** is a finite sequence of 0s and 1s.
- Examples:
  - 11011100
  - 010101010101
  - 0000
  - ε (the ***empty string***)
- There are 2^n bitstrings of length n.

Data Compression

- Inside a computer, all data are represented as sequences of 0s and 1s (bitstrings)
- To transfer data over a network (or on a flash drive, if you're still into that), it is useful to reduce the number of 0s and 1s before transferring it.
- Most real-world data can be compressed by exploiting redundancies.
  - Text repeats common patterns (“the”, “and”, etc.)
  - Bitmap images use similar colors throughout the image.
- ***Idea:*** Replace each bitstring with a *shorter* bitstring that contains all the original information.
  - This is called ***lossless data compression***.


10101010101010101010101010101010
Compress
1111010
Transmit
1111010
Decompress
10101010101010101010101010101010

Lossless Data Compression

- In order to losslessly compress data, we need two functions:
  - A ***compression function*** C, and
  - A ***decompression function*** D.
- We need to have $D(C(x)) = x$.
  - Otherwise, we can't uniquely encode or decode some bitstring.

How many of the following must be true about C and D?

C must be injective.
C must be surjective.
D must be injective.
D must be surjective.

Answer at **PollEv.com/cs103** or
text **CS103** to **22333** once to join, then a number.

Lossless Data Compression

- In order to losslessly compress data, we need two functions:
  - A ***compression function*** C, and
  - A ***decompression function*** D.
- We need to have $D(C(x)) = x$.
  - Otherwise, we can't uniquely encode or decode some bitstring.
- This means that D must be a left inverse of C, so (as you proved in PS3!) C must be injective.

A Perfect Compression Function

- Ideally, the compressed version of a bitstring would always be shorter than the original bitstring.
- ***Question***: Can we find a lossless compression algorithm that always compresses a string into a shorter string?
- To handle the issue of the empty string (which can't get any shorter), let's assume we only care about strings of length at least 10.

A Counting Argument

- Let $\mathbb{B}^n$ be the set of bitstrings of length n, and $\mathbb{B}^{<n}$ be the set of bitstrings of length less than n.
- How many bitstrings of length n are there?
  - ***Answer***: 2^n
- How many bitstrings of length *less than* n are there?
  - ***Answer***: $2^0 + 2^1 + \ldots + 2^{n-1} = 2^n - 1$
- By the pigeonhole principle, no function from $\mathbb{B}^n$ to $\mathbb{B}^{<n}$ can be injective – at least two elements must collide!
- Since a perfect compression function would have to be an injection from $\mathbb{B}^n$ to $\mathbb{B}^{<n}$, ***there is no perfect compression function!***

Why this Result is Interesting

- Our result says that no matter how hard we try, it is ***impossible*** to compress every string into a shorter string.
- No matter how clever you are, you cannot write a lossless compression algorithm that always makes strings shorter.
- In practice, only highly redundant data can be compressed.
- The fields of ***information theory*** and ***Kolmogorov complexity*** explore the limits of compression; if you're interested, go explore!

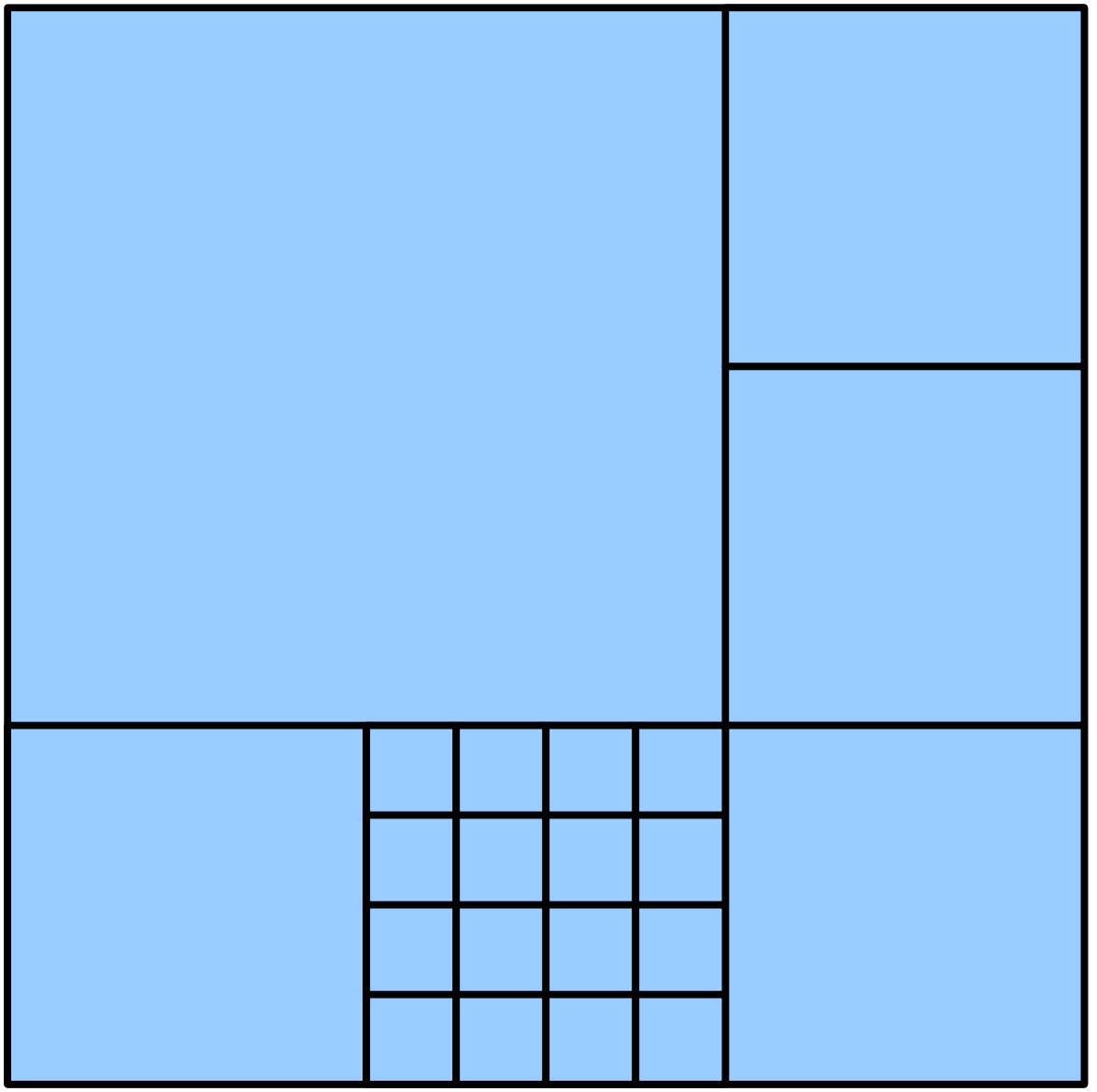
Variations on Induction: ***Starting Later***

Induction Starting at ***m***

- To prove that $P(n)$ is true for all natural numbers greater than or equal to ***m***:
  - Show that $P($***m***$)$ is true.
  - Show that for any $k \geq$ ***m***, that if $P(k)$ is true, then $P(k+1)$ is true.
  - Conclude $P(n)$ holds for all natural numbers greater than or equal to ***m***.

Variations on Induction: ***Bigger Steps***

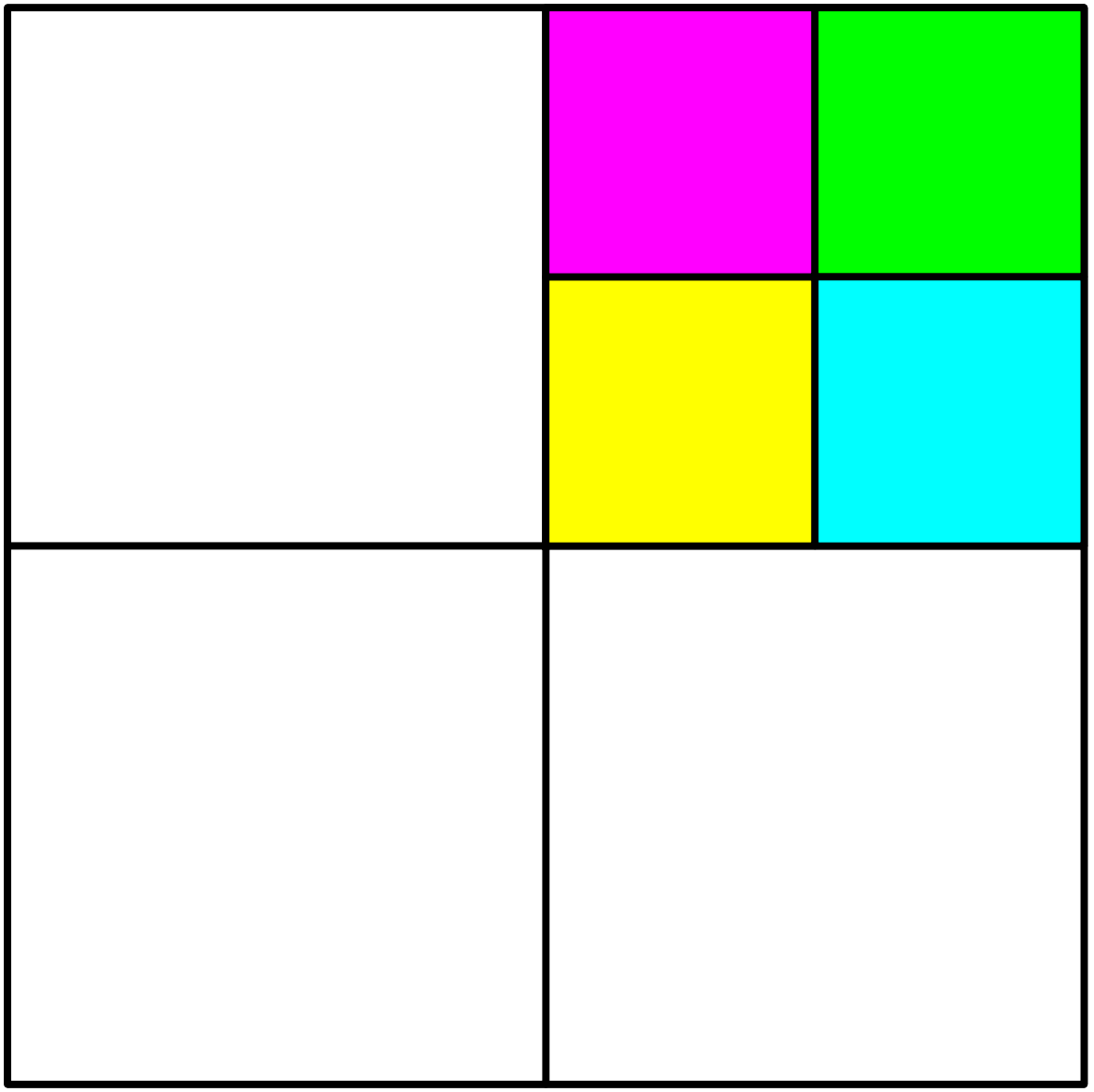
Subdividing a Square



For what values of n can a square be subdivided into n squares?

1 ~~2~~ ~~3~~ 4 ~~5~~ 6 7 8 9 10 11 12

The Key Insight



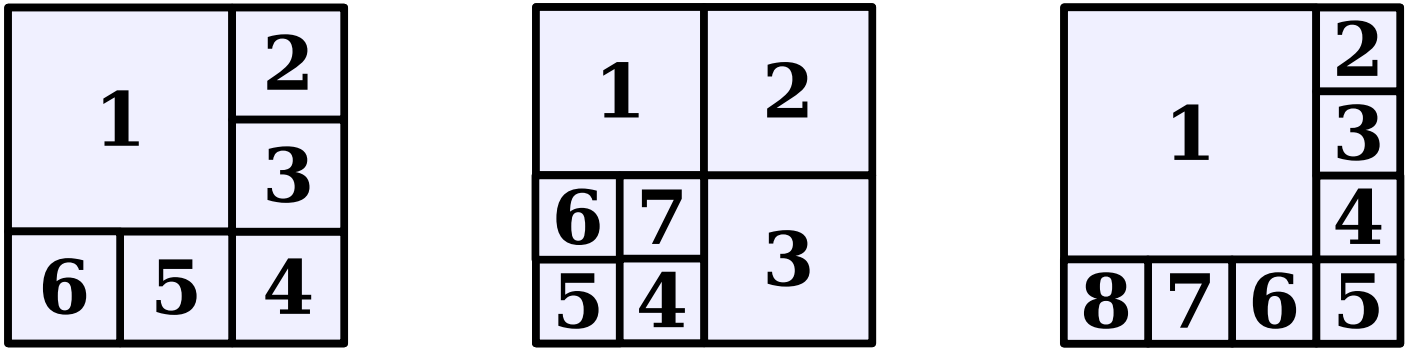
The Key Insight

- If we can subdivide a square into n squares, we can also subdivide it into $n + 3$ squares.
- Since we can subdivide a bigger square into 6, 7, and 8 squares, we can subdivide a square into n squares for any $n \geq 6$:
  - For multiples of three, start with 6 and keep adding three squares until n is reached.
  - For numbers congruent to one modulo three, start with 7 and keep adding three squares until n is reached.
  - For numbers congruent to two modulo three, start with 8 and keep adding three squares until n is reached.

Theorem: For any $n \geq 6$, it is possible to subdivide a square into n smaller squares.

Proof: Let $P(n)$ be the statement "a square can be subdivided into n smaller squares." We will prove by induction that $P(n)$ holds for all $n \geq 6$, from which the theorem follows.

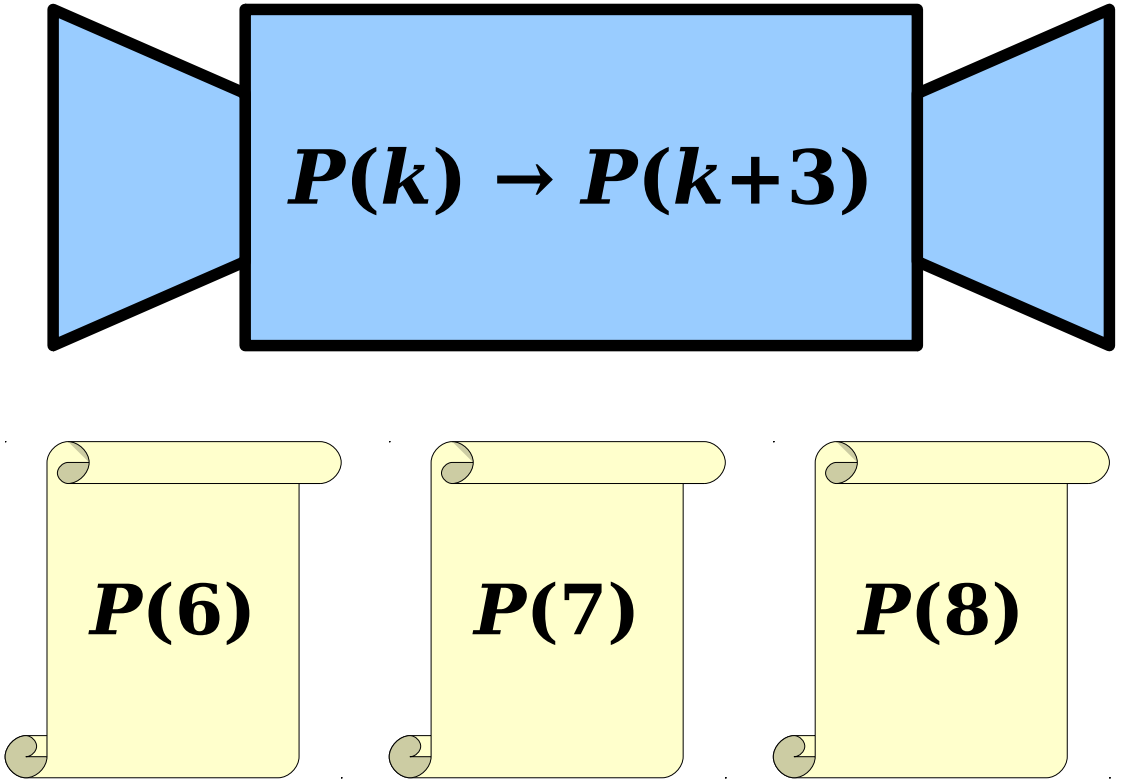
As our base cases, we prove $P(6)$, $P(7)$, and $P(8)$, that a square can be subdivided into 6, 7, and 8 squares. This is shown here:



For the inductive step, assume that for some arbitrary $k \geq 6$ that $P(k)$ is true and that a square can be subdivided into k squares. We prove $P(k+3)$, that a square can be subdivided into $k+3$ squares. To see this, start by obtaining (via the inductive hypothesis) a subdivision of a square into k squares. Then, choose any of the squares and split it into four equal squares. This removes one of the k squares and adds four more, so there will be a net total of $k+3$ squares. Thus $P(k+3)$ holds, completing the induction. ■

Why This Works

- This induction has three consecutive base cases and takes steps of size three.
- Thinking back to our “induction machine” analogy:



Generalizing Induction

- When doing a proof by induction,
  - feel free to use multiple base cases, and
  - feel free to take steps of sizes other than one.
- Just be careful to make sure you cover all the numbers you think that you're covering!
  - We won't require that you prove you've covered everything, but it doesn't hurt to double-check!

More on Square Subdivisions

- There are a ton of interesting questions that come up when trying to subdivide a rectangle or square into smaller squares.
- In fact, one of the major players in early graph theory (William Tutte) got his start playing around with these problems.
- Good starting resource: this Numberphile video on ***Squaring the Square***.

Time-Out for Announcements!

CS+SOCIAL GOOD

WINTER **MIXER**

MONDAY, FEBRUARY 12 **5-6 PM**

OLD UNION **200**

Come meet fellow students and teachers who are passionate about using technology for good!

Everyone is welcome, and there will be free boba and snacks!

Problem Set Five

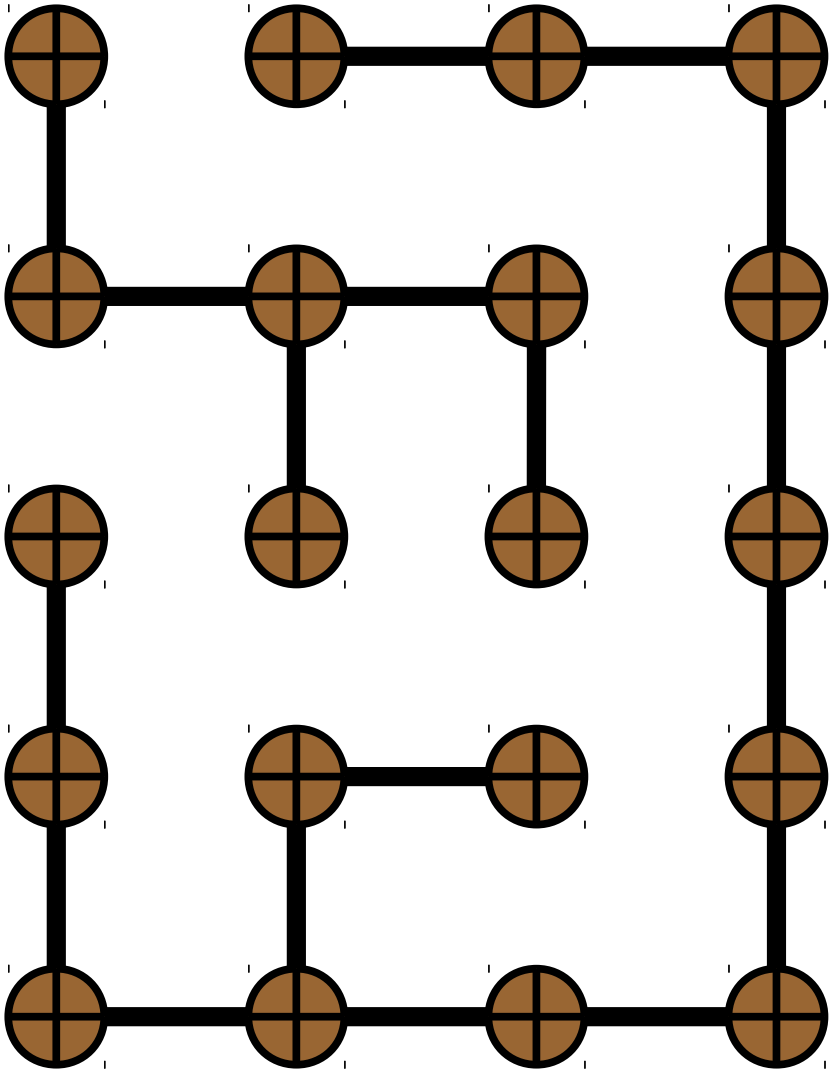
- Problem Set Four was due at 2:30PM today.
- Problem Set Five goes out today. It's due next Friday at 2:30PM.
  - Play around with everything we've covered so far, plus a healthy dose of induction and inductive problem-solving.
  - There is no checkpoint problem, and there are no checkpoints from here out out.

Back to CS103!

A Motivating Question: ***Rat Mazes***

Rat Mazes

- Suppose you want to make a rat maze consisting of an $n \times m$ grid of pegs with slats between them.
- The maze should have these properties:
  - There is one entrance and one exit in the border.
  - Every spot in the maze is reachable from every other spot.
  - There is exactly one path from each spot in the maze to each other spot.

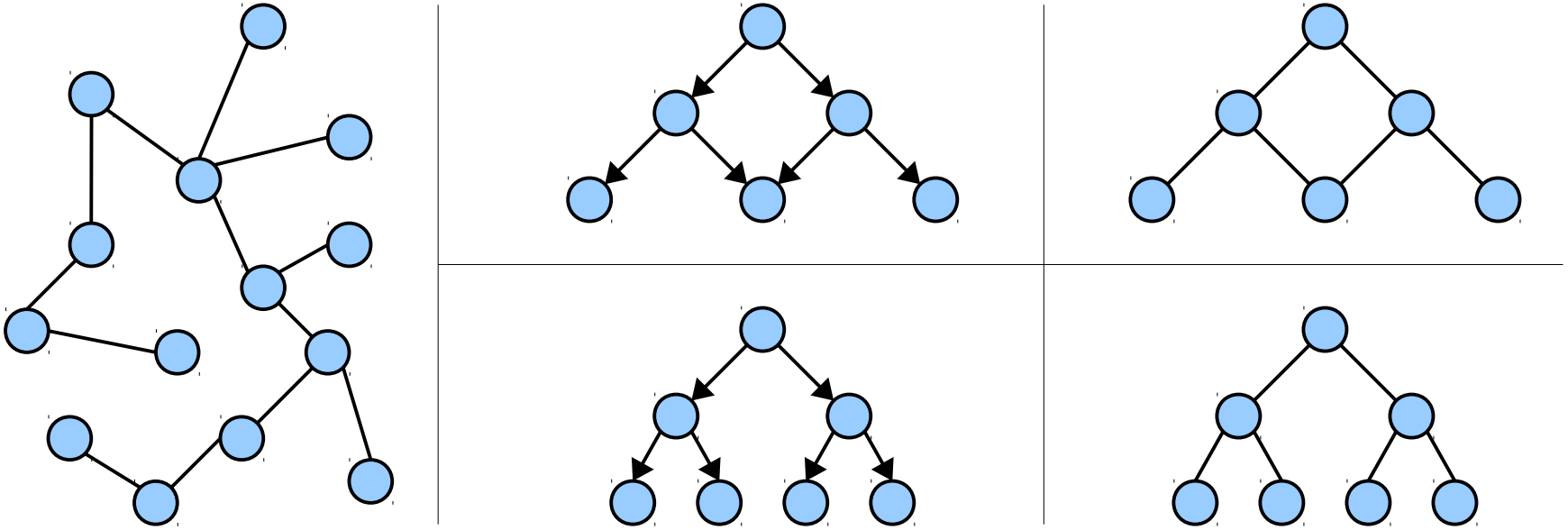


Question: If you have an $n \times m$ grid of pegs, how many slats do you need to make?

A Special Type of Graph: ***Trees***

- A ***tree*** is a connected, nonempty graph with no simple cycles.

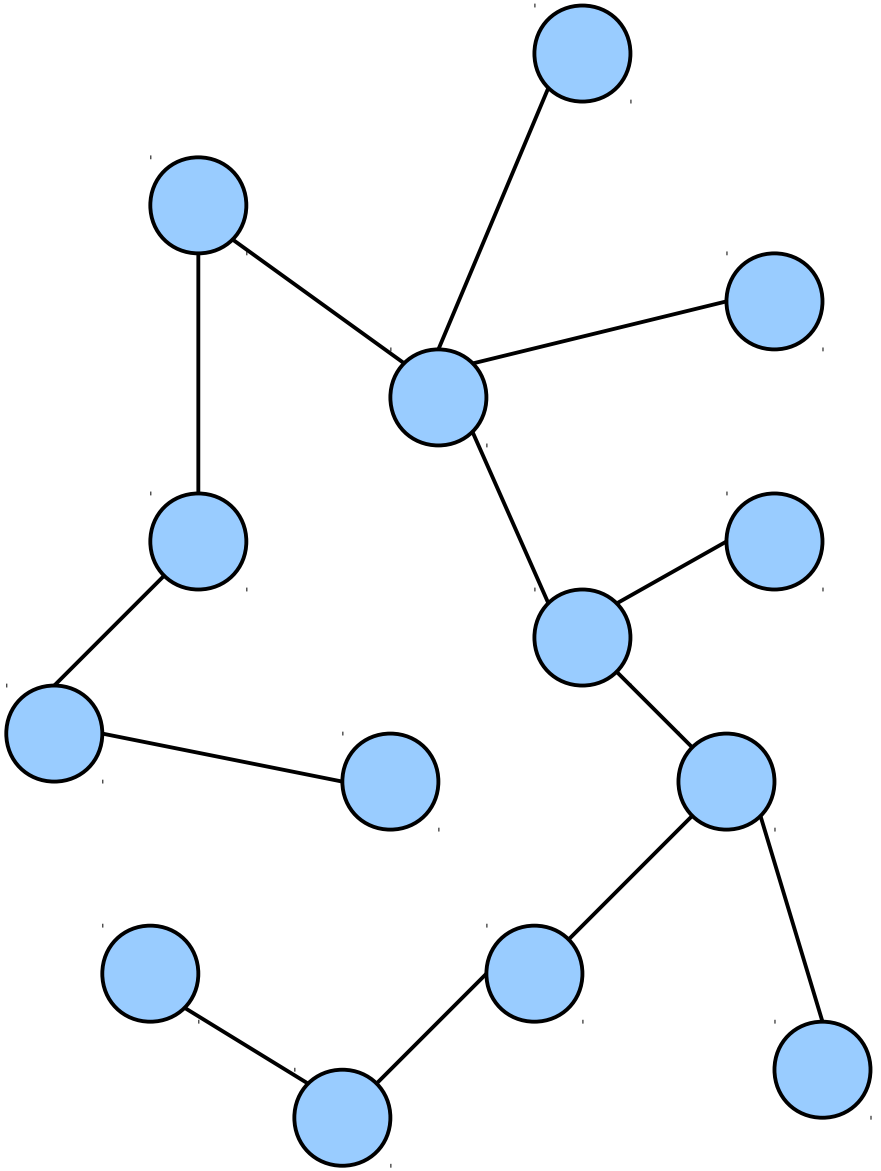
According to the above definition of trees, how many of these graphs are trees?



Answer at **PollEv.com/cs103** or
text **CS103** to **22333** once to join, then a number.

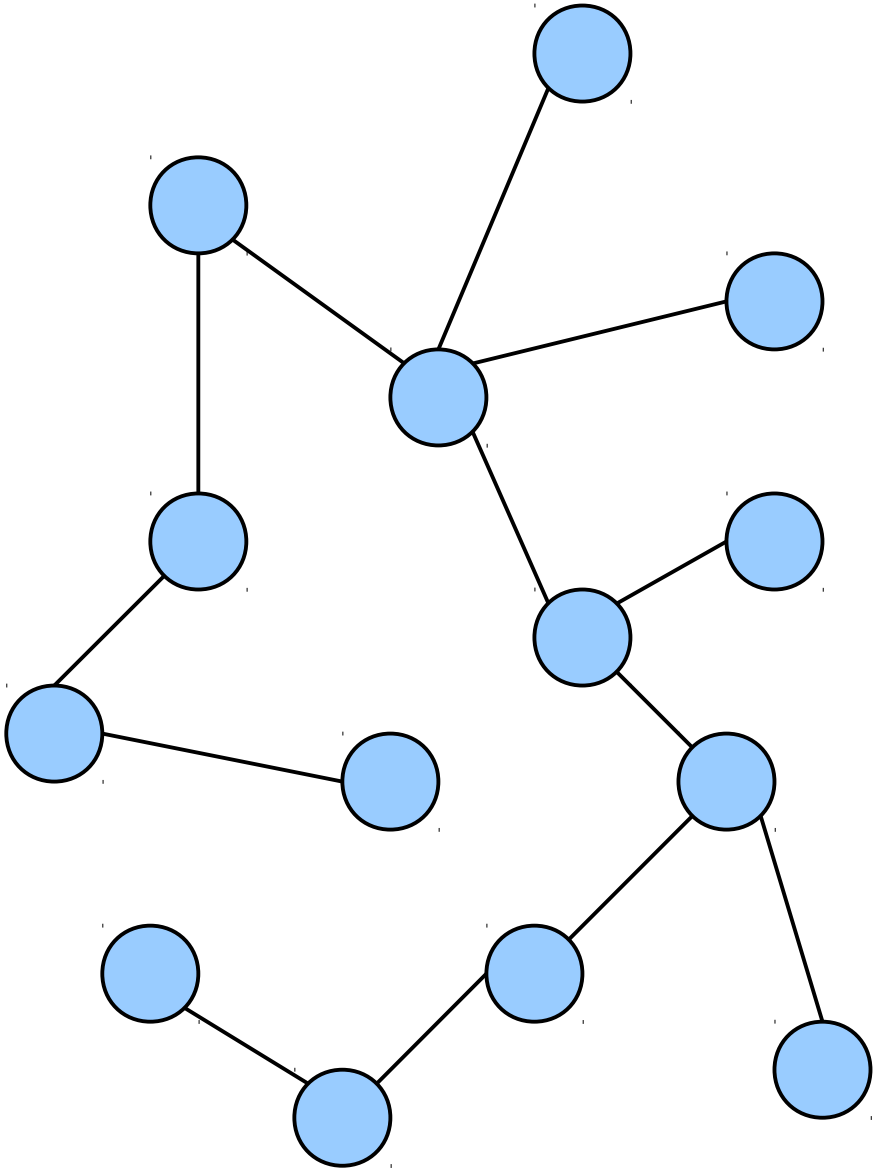
Trees

- A ***tree*** is a connected, nonempty graph with no simple cycles.
- Trees have tons of nice properties:
  - They're ***maximally acyclic*** (adding any missing edge creates a simple cycle)
  - They're ***minimally connected*** (deleting any edge disconnects the graph)
- Proofs of these results are in the course reader if you're interested. They're also great exercises.



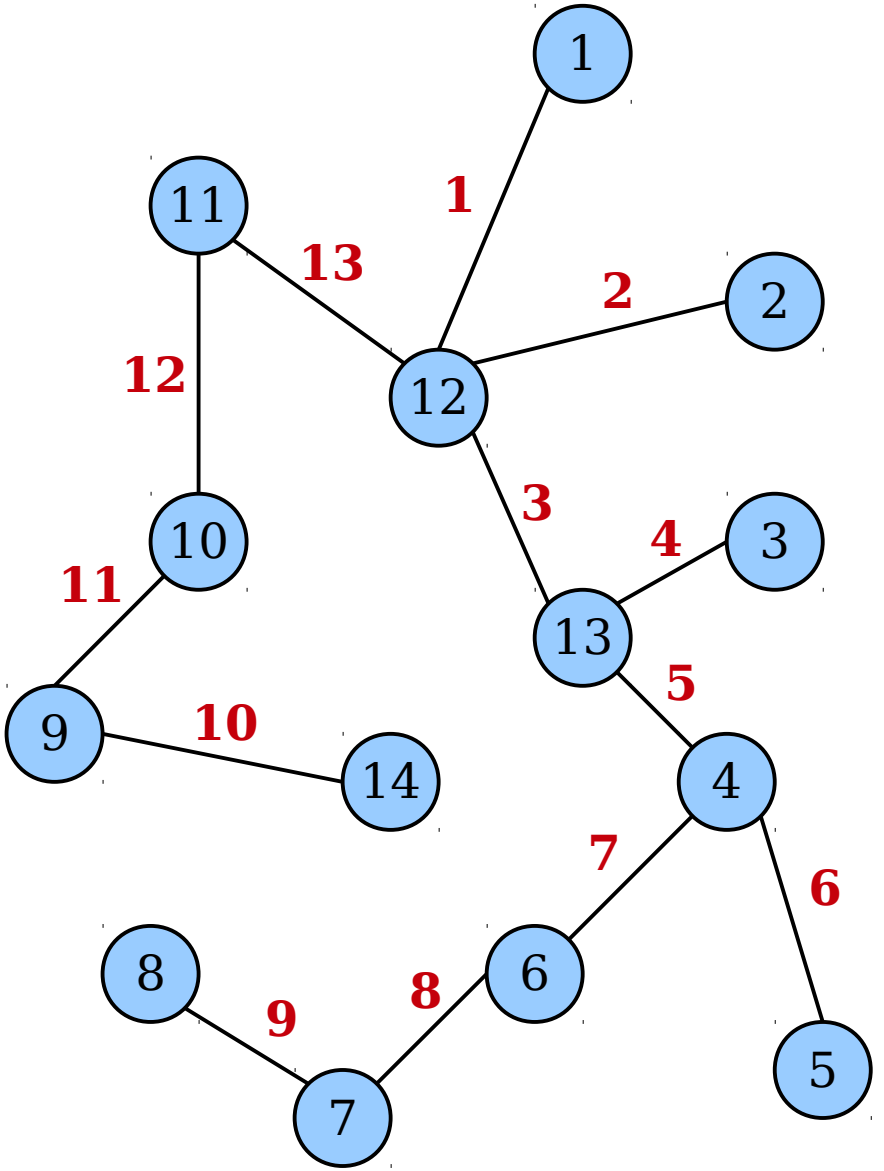
Trees

- ***Theorem:*** If T is a tree with at least two nodes, then deleting any edge from T splits T into two nonempty trees T_1 and T_2.
- ***Proof:*** Left as an exercise to the reader. ☺



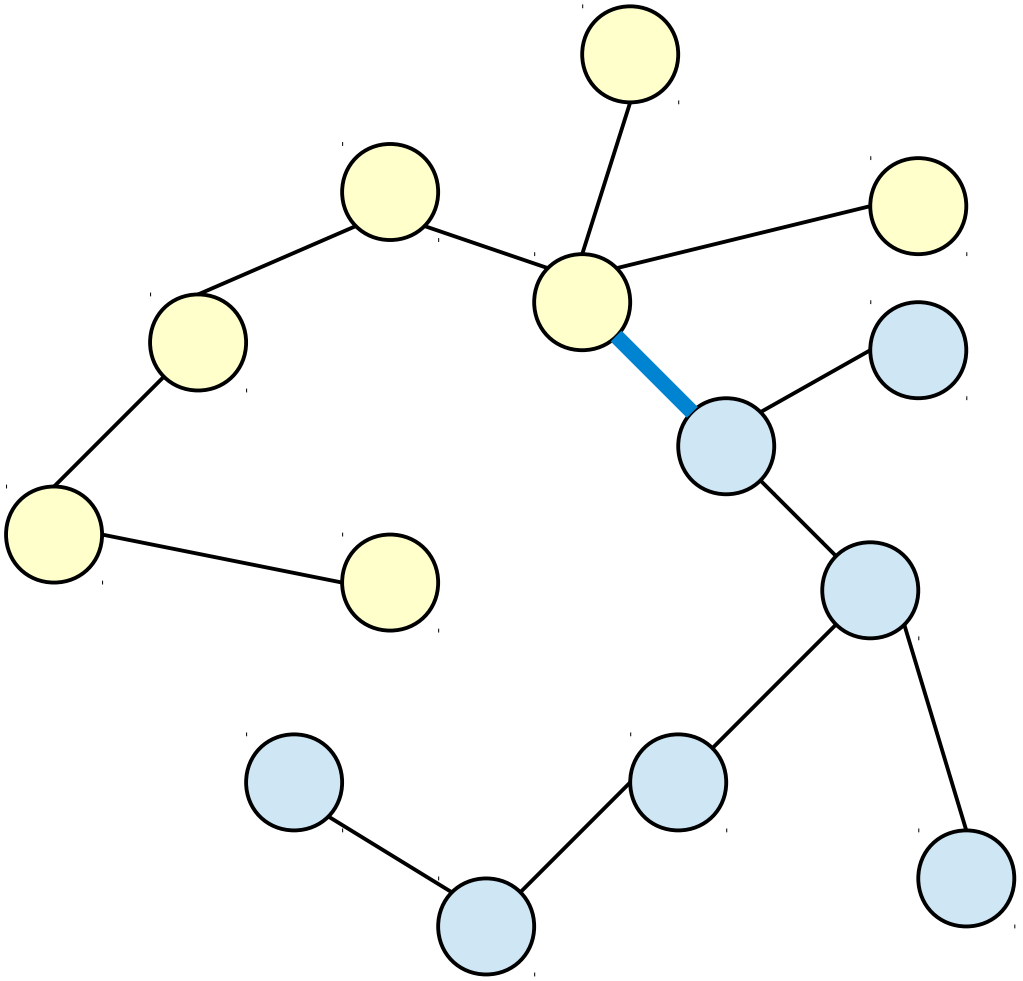
Trees

- ***Theorem:*** If T is a tree with $n \geq 1$ nodes, then T has exactly n-1 edges.
- ***Proof:*** Up next!



Our Base Case

Assume any tree with at most k nodes has one more node than edge.



Consider an arbitrary tree with $k+1$ nodes.

Suppose there are r nodes in the yellow tree.

Then there are $(k+1)-r$ nodes in the blue tree.

There are $r-1$ edges in the yellow tree and $k-r$ edges in the blue tree.

Adding in the initial edge we cut, there are $r-1 + k-r + 1 = k$ edges in the original tree.

Theorem: If T is a tree with $n \geq 1$ nodes, then T has n-1 edges.

Proof: Let $P(n)$ be the statement "any tree with n nodes has n-1 edges." We will prove by induction that $P(n)$ holds for all $n \geq 1$, from which the theorem follows.

As a base case, we will prove $P(1)$, that any tree with 1 node has 0 edges. Any such tree has single node, so it cannot have any edges.

Now, assume for some arbitrary $k \geq 1$ that $P(1)$, $P(2)$, …, and $P(k)$ are true, so any tree with between 1 and k nodes has one more node than edge. We will prove $P(k+1)$, that any tree with $k+1$ nodes has k edges.

Consider any tree T with $k+1$ nodes. Since T has at least two nodes and is connected, it must contain at least one edge. Choose any edge in T and delete it. This splits T into two nonempty trees T_1 and T_2. Every edge in T is part of T_1, is part of T_2, or is the initial edge we deleted.

Let r be the number of nodes in T_1. Since every node in T belongs to either T_1 or T_2, we see that T_2 has $(k+1)$-r nodes. Additionally, since T_1 and T_2 are nonempty, neither T_1 nor T_2 contains all the nodes from T. Therefore, T_1 and T_2 each have between 1 and k nodes. We can then apply our inductive hypothesis to see that T_1 has r-1 edges and T_2 has k-r edges. Thus the total number of edges in T is $1 + (r\text{-}1) + (k\text{-}r) = k$, as required. Therefore, $P(k+1)$ is true, completing the induction. ■

Theorem: If T is a tree with $n \geq 1$ nodes, then T has n-1 edges.

Proof: Let $P(n)$ be the statement "any tree with n nodes has n-1 edges." We will prove by induction that $P(n)$ holds for all $n \geq 1$, from which the theorem follows.

As a base case, we will prove $P(1)$, that any tree with 1 node has 0 edges. Any such tree has single node, so it cannot have any edges.

Now, assume for some arbitrary $k \geq 1$ that $P(1)$, $P(2)$, ..., and $P(k)$ are true, so any tree with between 1 and k nodes has one more node than edge. We will prove $P(k+1)$, that any tree with $k+1$ nodes has k edges.

Which of the following best describes the structure of the inductive step in this proof?

A. Assume $P(1)$, then prove $P(k+1)$.
B. Assume $P(k)$, then prove $P(k+1)$.
C. Assume $P(1)$, then prove $P(1)$, ..., $P(k)$, and $P(k+1)$.
D. Assume $P(1)$, ..., and $P(k)$, then prove $P(k+1)$.
E. None of these, or more than one of these.

Answer at **PollEv.com/cs103** or text **CS103** to **22333** once to join, then ***A***, ..., or ***E***.

T_1

Therefore, T_1 and T_2 each have between 1 and k nodes. We can then apply our inductive hypothesis to see that T_1 has r-1 edges and T_2 has k-r edg $r) = k$, as req n. ■

Complete Induction

- If the following are true:
  - $P(0)$ is true, and
  - If $P(0)$, $P(1)$, $P(2)$, …, $P(k)$ are true, then $P(k+1)$ is true as well.

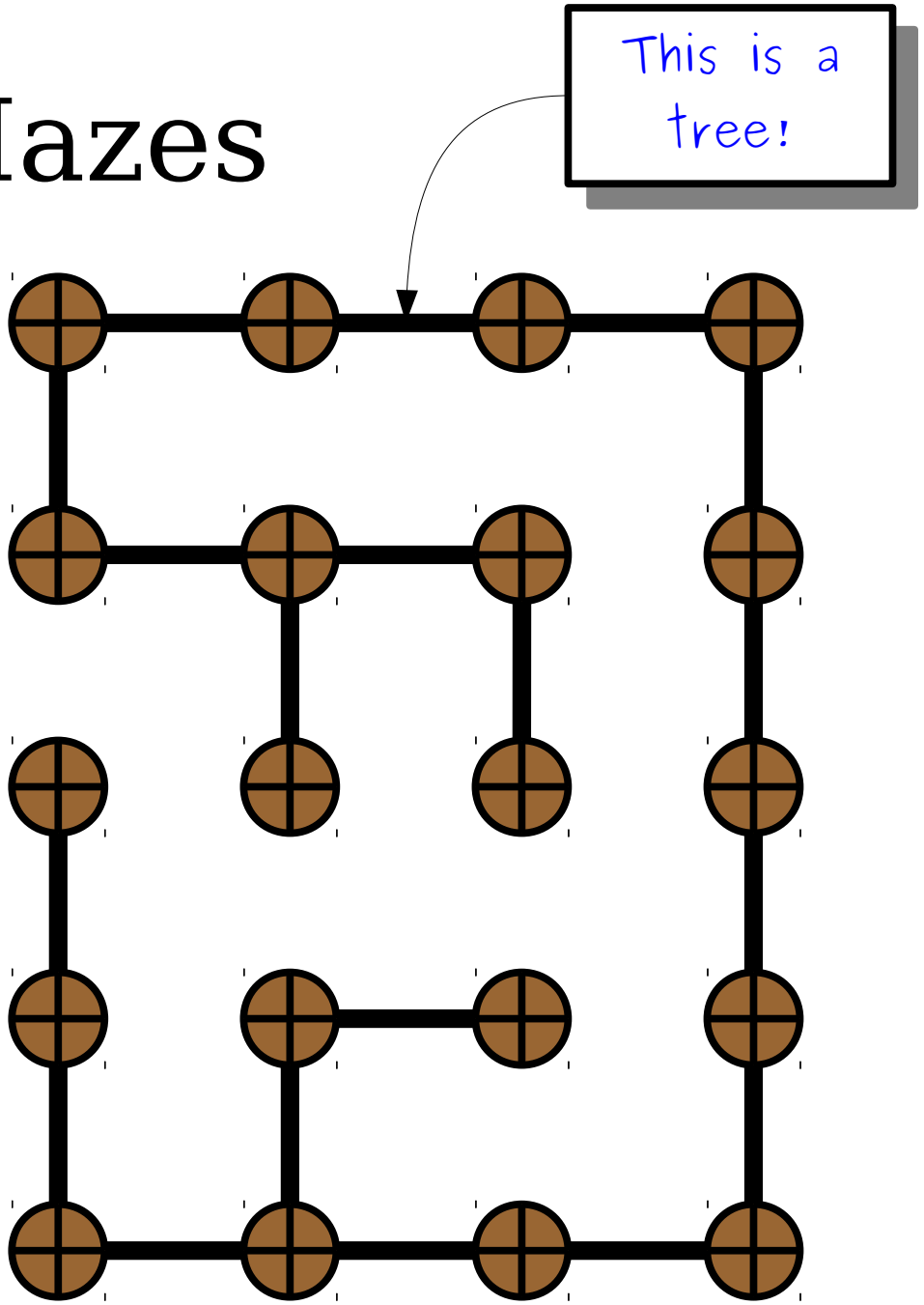
  then $P(n)$ is true for all $n \in \mathbb{N}$.
- This is called the ***principle of complete induction*** or the ***principle of strong induction***.
  - (This also works starting from a number other than 0; just modify what you're assuming appropriately.)

When Use Complete Induction?

- Normal induction is good for when you are shrinking the problem size by exactly one.
  - Peeling one final term off a sum.
  - Making one weighing on a scale.
  - Considering one more action on a string.
- Complete induction is good when you are shrinking the problem, but you can't be sure by how much.
  - In the previous example, if we delete a random edge, we can't know in advance how big the resulting trees will be.

Rat Mazes

- Suppose you want to make a rat maze consisting of an $n \times m$ grid of pegs with slats between them.
- ***Question:*** How many slats do you need to create?
- ***Answer:*** $mn - 2$.



For more on trees, take CS161 / 261 / 267!

An Important Milestone

Recap: ***Discrete Mathematics***

- The past five weeks have focused exclusively on discrete mathematics:

Induction	Functions
Graphs	The Pigeonhole Principle
Relations	Mathematical Logic
Set Theory	Cardinality

- These are building blocks we will use throughout the rest of the quarter.
- These are building blocks you will use throughout the rest of your CS career.

Next Up: ***Computability Theory***

- It's time to switch gears and address the limits of what can be computed.
- We'll explore these questions:
  - How do we model computation itself?
  - What exactly is a computing device?
  - What problems can be solved by computers?
  - What problems *can't* be solved by computers?
- ***Get ready to explore the boundaries of what computers could ever be made to do.***

Next Time

- ***Formal Language Theory***
  - How are we going to formally model computation?
- ***Finite Automata***
  - A simple but powerful computing device made entirely of math!
- ***DFAs***
  - A fundamental building block in computing.