The Pigeonhole Principle

Friday Four Square
Today at 4:15PM, Outside Gates
(Weather Permitting)

Announcements

- Problem Set 2 due right now.
- Problem Set 3 goes out.
  - Checkpoint due **Monday, October 15**.
  - Remainder due **Friday, October 19**.
- Play around with graphs, relations, functions, cardinality, and the pigeonhole principle!

The Pigeonhole Principle

The **pigeonhole principle** is the following:

If m objects are placed into n bins, where $m > n$, then some bin contains at least two objects.

(We sketched a proof in Lecture #02)

Why This Matters

- The pigeonhole principle can be used to show results must be true because they are "too big to fail."
- Given a large enough number of objects with a bounded number of properties, eventually at least two of them will share a property.
- The applications are interesting, surprising, and thought-provoking.

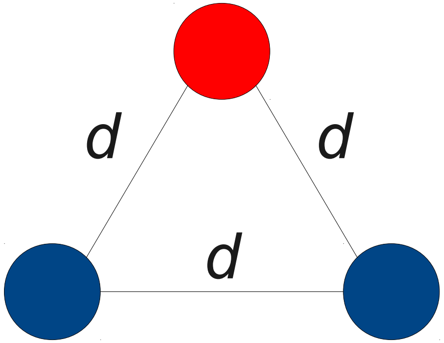
Using the Pigeonhole Principle

- To use the pigeonhole principle:
  - Find the *m* objects to distribute.
  - Find the $n < m$ buckets into which to distribute them.
  - Conclude by the pigeonhole principle that there must be two objects in some bucket.
- The details of how to proceeds from there are specific to the particular proof you're doing.

A Surprising Application

Theorem: Suppose that every point in the real plane is colored either red or blue. Then for any distance $d > 0$, there are two points exactly distance d from one another that are the same color.

Any pair of these points is at distance d from one another. Since two must be the same color, there is a pair of points of the same color at distance d!



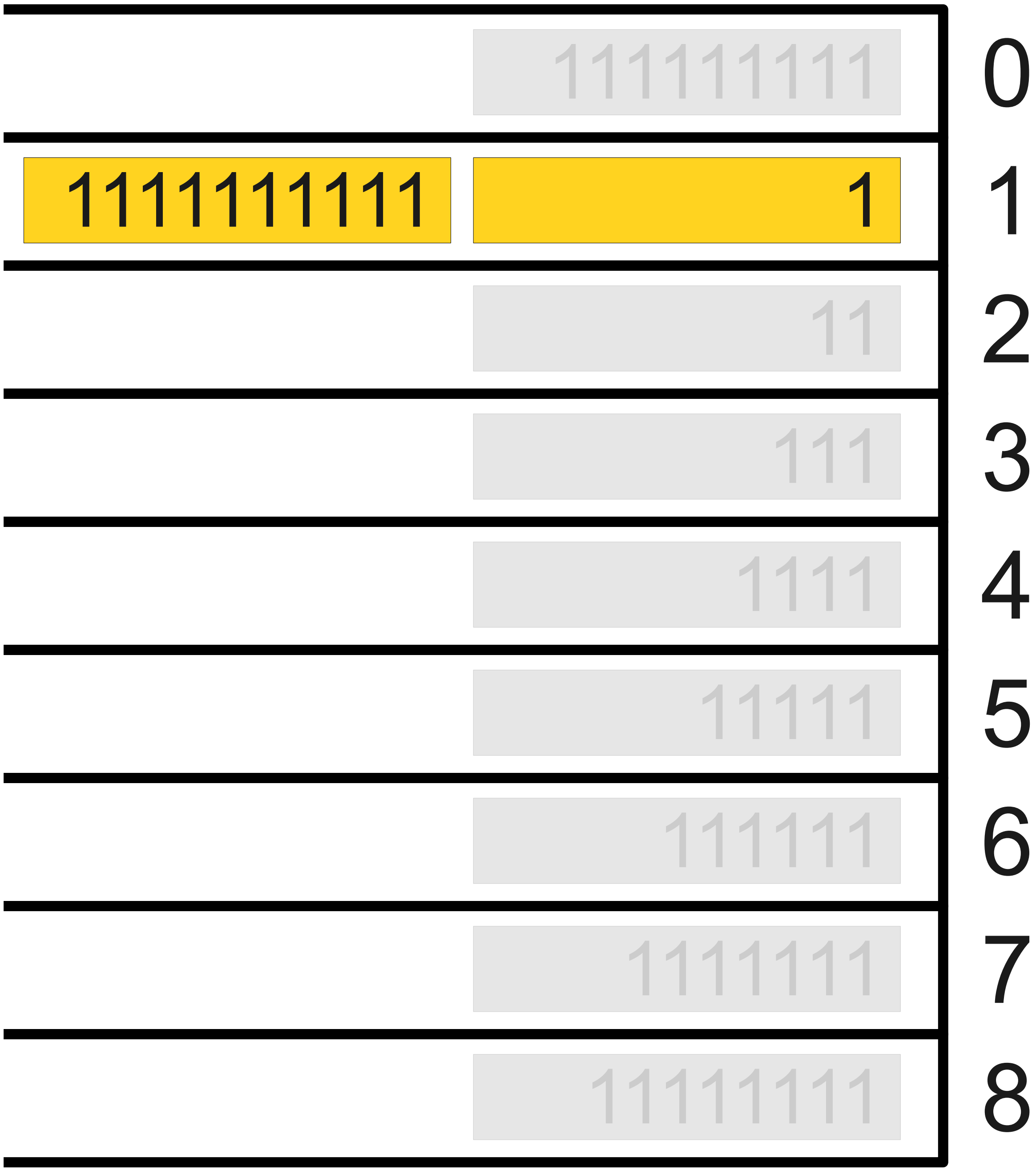
A Surprising Application

Theorem: Suppose that every point in the real plane is colored either red or blue. Then for any distance $d > 0$, there are two points exactly distance d from one another that are the same color.

Proof: Consider any equilateral triangle whose side lengths are d. Put this triangle anywhere in the plane. By the pigeonhole principle, because there are three vertices, two of the vertices must have the same color. These vertices are at distance d from each other, as required. ■

Theorem: For any natural number n, there is a nonzero multiple of n whose digits are all 0s and 1s.

```
  1111111111
-          1
  1111111110
```



	111111111	0
1111111111	1	1
	11	2
	111	3
	1111	4
	11111	5
	111111	6
	1111111	7
	11111111	8

Proof Idea

- For any natural number $n \geq 2$ generate the numbers 1, 11, 111, … until $n + 1$ numbers are generated.
- There are n possible remainders modulo n, so two of these numbers have the same remainder.
- Their difference is a multiple of n.
- Their difference consists of 1s and 0s.

Theorem: For any natural number n, there is a nonzero multiple of n whose digits are all 0s and 1s.

Proof: For any $k \in \mathbb{N}$ in the range $0 \leq k \leq n$, consider S_k defined as

$$S_k = \sum_{i=0}^{k} 10^i$$

Now, consider the remainders of the S_k's modulo n. Since there are $n + 1$ S_k's and n remainders modulo n, by the pigeonhole principle there must be at least two S_k's that leave the same remainder modulo n. Let two of these S_k's be S_x and S_y, with $x > y$, and let the remainder be r.

Since $S_x \equiv_n r$, there exists $q_{\mathrm{x}} \in \mathbb{Z}$ such that $S_{\mathrm{x}} = nq_x + r$. Similarly, since $S_y \equiv_n r$, there exists $q_y \in \mathbb{Z}$ such that $S_y = nq_y + r$. Then $S_x - S_y = (nq_x + r) - (nq_y + r) = nq_x - nq_y = n(q_{\mathrm{x}} - q_{\mathrm{y}})$. Thus $S_x - S_y$ is a multiple of n. Moreover, we have that

$$n(q_x - q_y) = S_x - S_y = \sum_{i=0}^{x} 10^i - \sum_{i=0}^{y} 10^i = \sum_{i=y+1}^{x} 10^i$$

So $S_x - S_y$ is a sum of distinct powers of ten, so its digits are zeros and ones. Since $x > y$, we know that $x \geq y + 1$ and so the sum is nonzero. Therefore $S_x - S_y$ is a nonzero multiple of n consisting of 0s and 1s. ■

The Limits of Data Compression

Pigeonholing Injective Functions

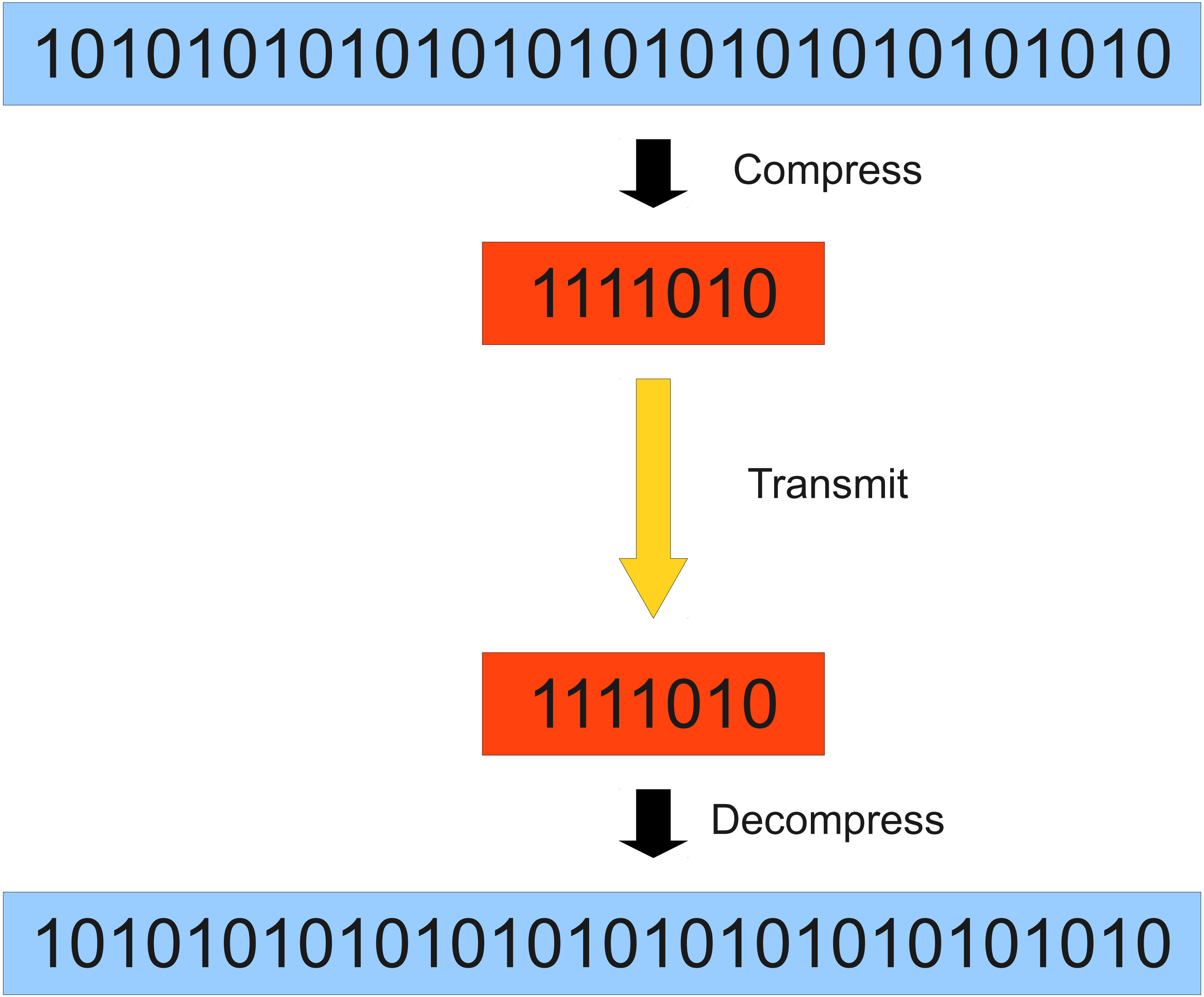
- Consider a function $f : A \to B$ for finite sets A and B.
- If $|A| > |B|$, then by the pigeonhole principle some element of B has at least two elements of A that map to it.
- Thus f cannot be injective.

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 (across a network, on DVDs, on a flash drive, etc.), it is advantageous to try 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**.



Lossless Data Compression

- In order to losslessly compress data, we need two functions:
    - A **compression function** C, and
    - A **decompression function** D.
- These functions must be inverses of one another: $D(C(x)) = x$.
    - Otherwise, we can't uniquely encode or decode some bitstring.
- Therefore, C must be injective.
    - Otherwise, $C(x_0) = y = C(x_1)$ for some x_0 and x_1, and so we can't tell whether $D(y) = x_0$ or $D(y) = x_1$.

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:** $2^0 + 2^1 + \ldots + 2^{n-1} = 2^n - 1$
- Using our earlier result, by the pigeonhole principle, there cannot be an injection from $\mathbb{B}^n$ to $\mathbb{B}^{<n}$.
- 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!**

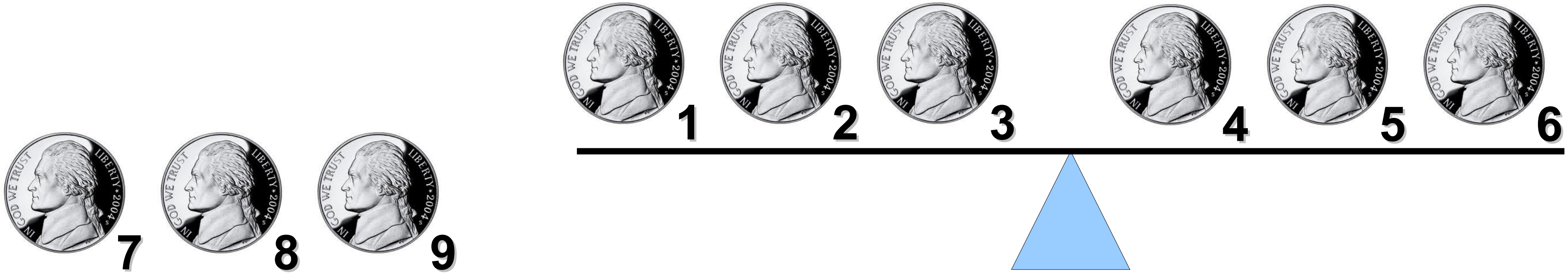
Why this Result is Interesting

- 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!

The Limits of Counterfeit Detection

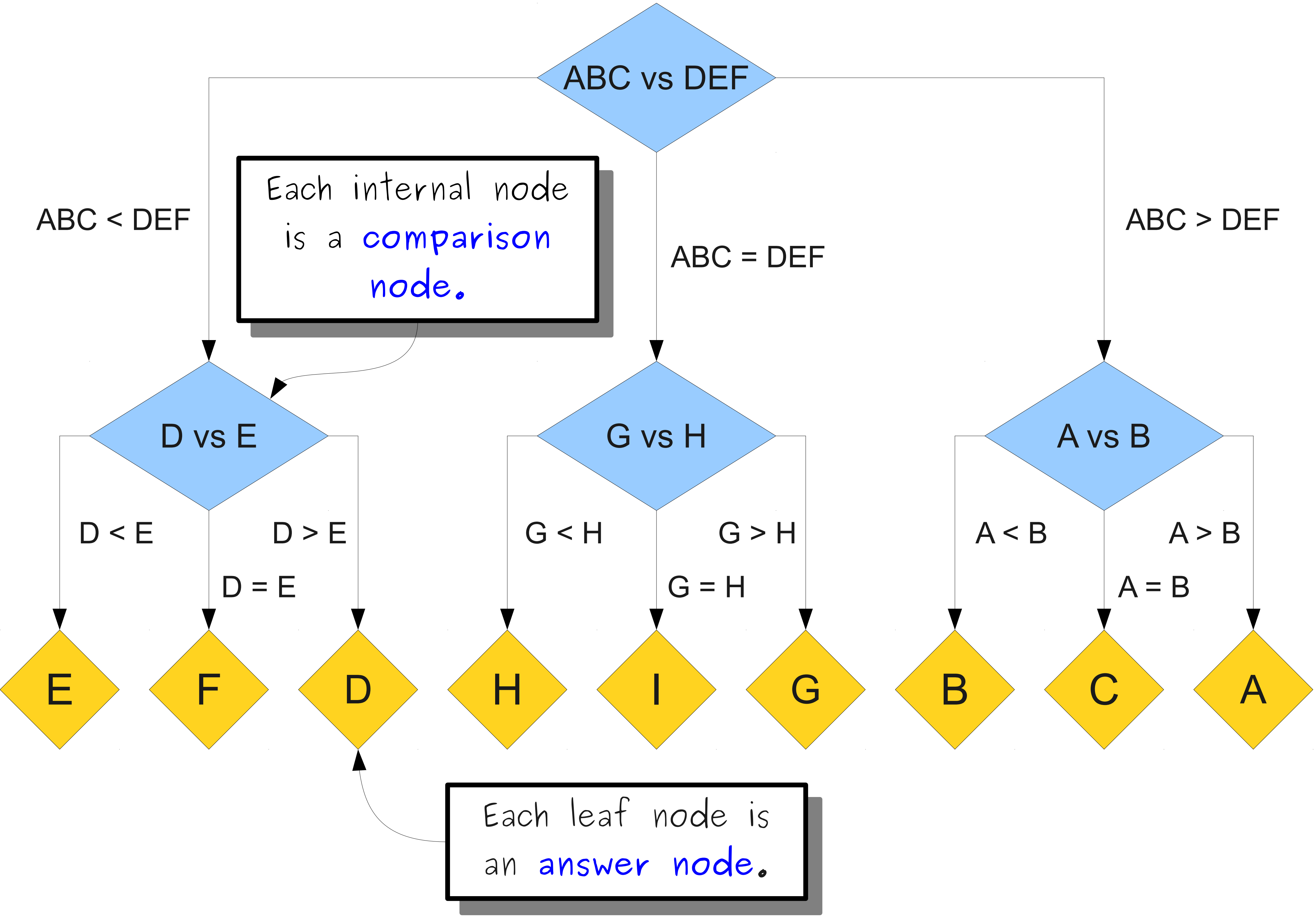
The Counterfeit Coin Problem

- Given 3^n coins, one of which weighs more than the rest, find that coin with at most n weighings on a balance.



Modeling an Algorithm

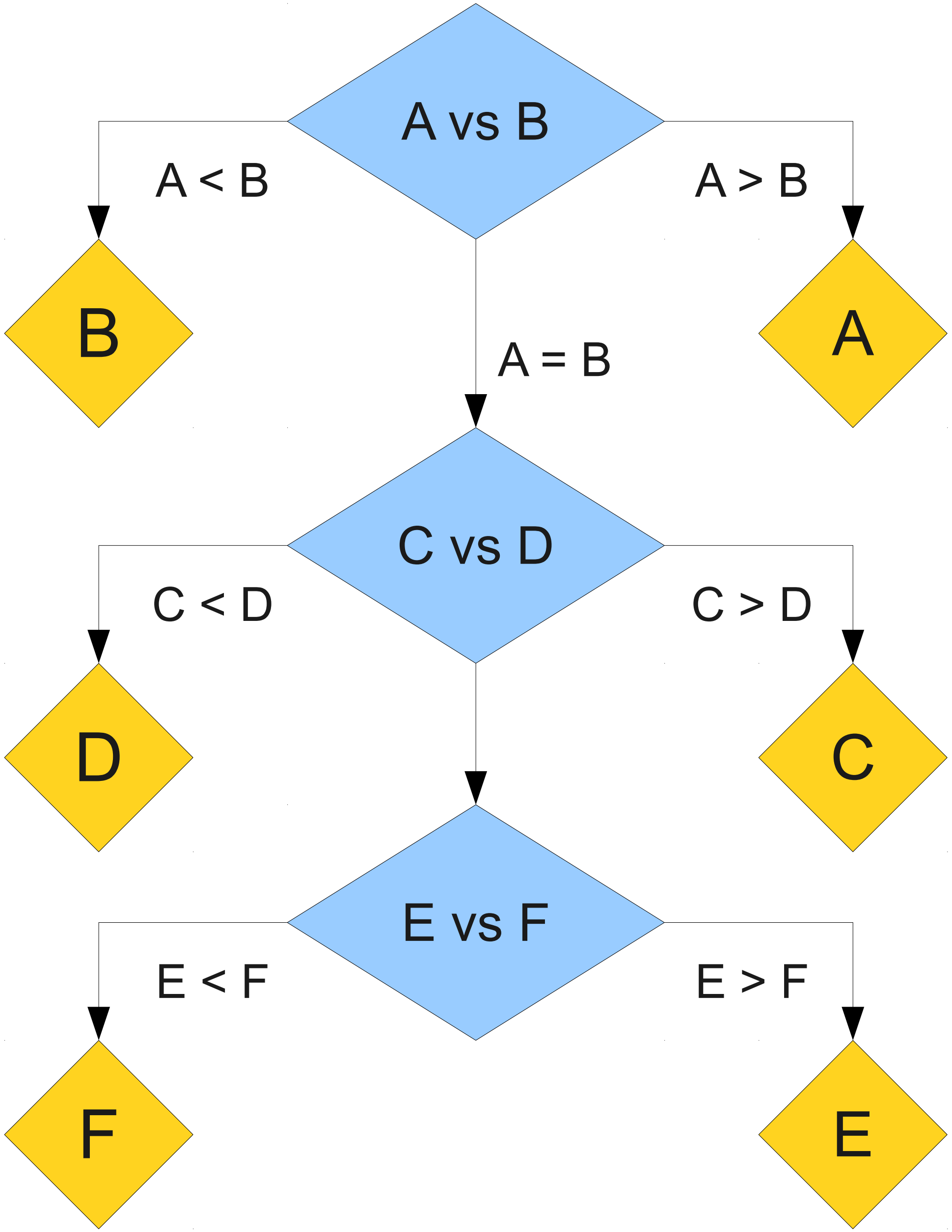
- In order to reason about the maximum number of coins, we need to find some way to reason about all possible algorithms for finding the coin.
- Main assumption: The only operation we can perform on the coins is weighing them on the scale.
  - We can't test their density, give them to the Secret Service, etc.
- We'll call such an algorithm a **comparison-based algorithm**, since the only way of distinguishing coins is through comparisons.



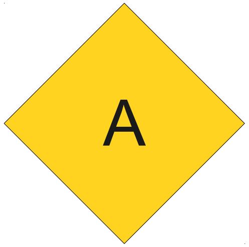
Each internal node is a **comparison node.**

Each leaf node is an **answer node.**

An Algorithm for Six Coins



An Algorithm for One Coin



Reasoning about Algorithms

- In this setup, every algorithm corresponds to a tree structure consisting of **comparisons** and **answers**.
- Each **comparison node** produces one of three outputs.
- Each **answer node** immediately ends the algorithm with the answer.
- Reasoning about these structures will tell us about the counterfeit coin problem.

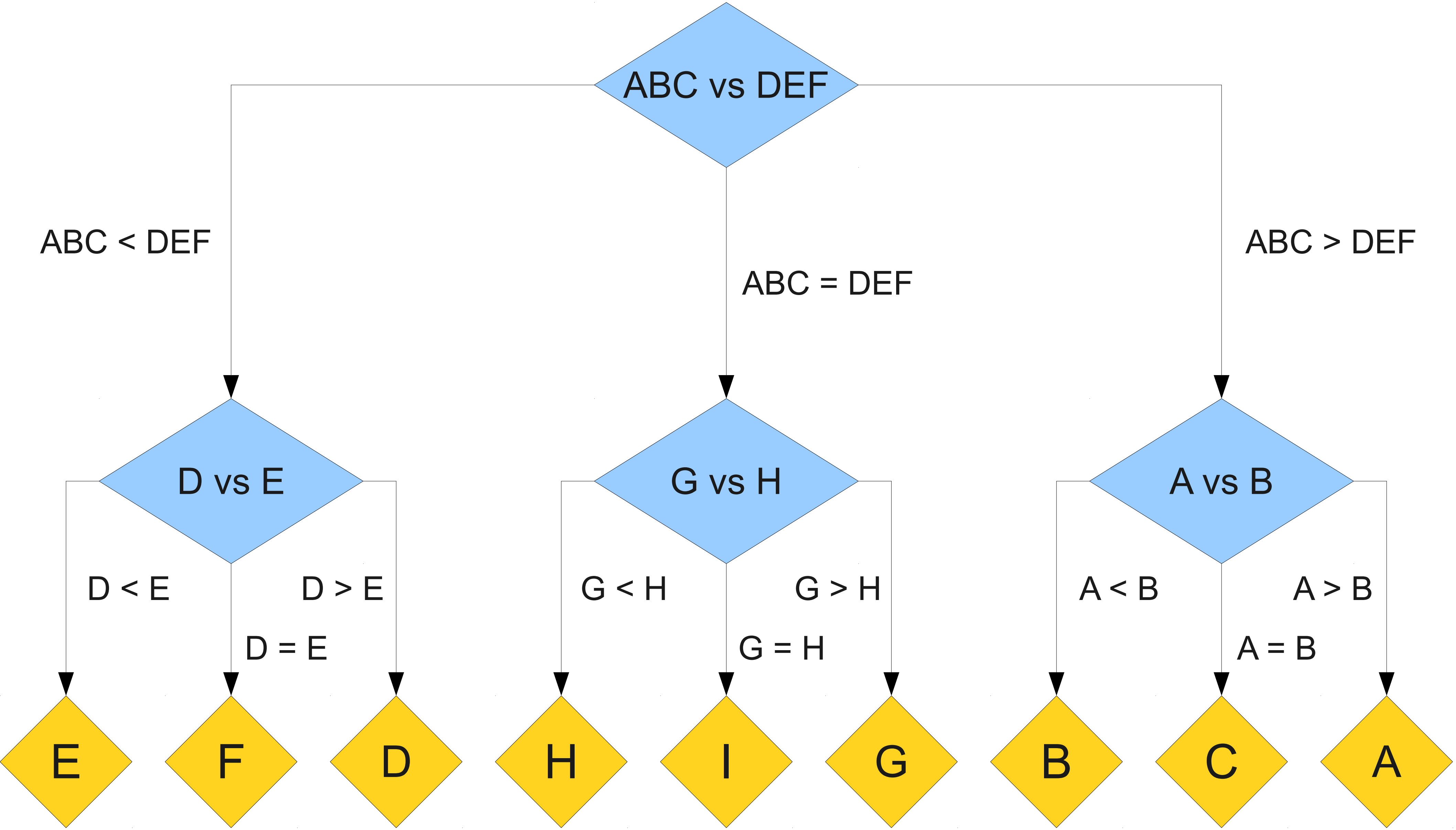
Reasoning about Inputs

- To be precise, we need to reason about the inputs to our algorithm.
- An input is a collection of *k* coins, exactly one of which is heavier than the rest.
  - It doesn't matter how much heavier it is; just that it weighs more than the rest.
- This means that there are exactly *k* possible inputs to the algorithm – one in which the first coin is counterfeit, one in which the second coin is counterfeit, etc.

A Critical Observation

- Suppose that we have an algorithm for finding which of k coins is counterfeit.
- There must be at least k answer nodes in the tree.
- Reasoning from the pigeonhole principle:
    - Run the algorithm on all k possible inputs.
    - Consider the set of counterfeit coins that arrive at each answer node in the tree.
    - If there are more coins than answer nodes, there must be two different coins that arrive at the same answer node.
    - The algorithm has to be wrong at least one of the two inputs.

How many answer nodes
can there be in the tree?


ABC vs DEF
ABC < DEF
ABC = DEF
ABC > DEF
D vs E
G vs H
A vs B
D < E
D = E
D > E
G < H
G = H
G > H
A < B
A = B
A > B
E
F
D
H
I
G
B
C
A

Theorem: In a comparison-based algorithm that makes at most n weighings on any input, there are at most 3^n answer nodes.

Proof: By induction on n. As our base case, consider an algorithm that makes zero comparisons. Then the algorithm is a single answer node. Since $1 = 3^0$, the claim holds for $n = 0$.

For the inductive hypothesis, assume that for any algorithm that makes at most n weighings, there are at most 3^n answer nodes. Consider any algorithm that makes at most $n + 1$ weighings. If the first step of the algorithm is an answer, then the algorithm is just a single answer node. In this case, it has $1 \leq 3^{n+1}$ answer nodes, so the claim holds.

Otherwise, the first step of the algorithm is a comparison. For each of the three outcomes, the algorithm can then make up to n more comparisons. By the inductive hypothesis, this means that in each of the subtrees corresponding to these outcomes, there can be up to 3^n answer nodes. Consequently, the overall algorithm can have at most $3(3^n) = 3^{n+1}$ answer nodes, completing the induction. ■

Theorem: Any comparison-based algorithm for finding which of k coins is heavier must perform at least $\log_3 k$ weighings on some input.

Proof: By contradiction; assume there is a comparison-based algorithm for this problem that makes strictly fewer than $\log_3 k$ comparisons on all inputs. By our previous result, this means that the algorithm must have strictly fewer than $3^{\log_3 k} = k$ answer nodes.

Now, consider the answer nodes arrived in when we run the algorithm on all k possible inputs. By the pigeonhole principle, since there are k inputs and fewer than k answer nodes, at least two inputs must arrive in the same answer node. Since these two inputs have different answers, this means that the algorithm must be incorrect on at least one of these two inputs, contradicting the fact that the algorithm finds which of the coins is heavier.

We have reached a contradiction, so our assumption must have been wrong. Thus any comparison-based algorithm for finding the heavier coin out of k coins must make at least $\log_3 k$ weighings. ■

Thm: There is no comparison-based algorithm for finding which of $3^n + k$ coins is counterfeit in n weighings for any $k > 0$.

Proof: By our previous result, we need at least $\log_3 (3^n + k)$ comparisons to determine which of $3^n + k$ coins is heaviest. If $k > 0$, then

$$\log_3 (3^n + k) > \log_3 3^n = n.$$

Thus we need strictly more than n weighings to find which of the coins is counterfeit. ■

***Corollary:* The comparison-based algorithm we developed in class is optimal.**

What Just Happened?

- **This is our first theorem about the difficulty of a specific problem!**
- Our procedure was as follows:
    - Build a mathematical model of computation for finding the counterfeit coin.
    - Given the model, reason about the behavior of that model on various inputs.
    - Write a proof that formalizes our reasoning about that behavior.
- We will build many more models like this one later in the quarter.

Suppose you have a set of coins. There is a counterfeit coin among them that weighs more than the rest of the coins.

If you have n weighings, what is the largest number of coins for which you can solve this problem?

Suppose you have a set of coins. There **may be** a counterfeit coin among them **(though there doesn't have to be)**. If there is a counterfeit, it will weigh more than the rest of the coins.

If you have *n* weighings, what is the largest number of coins for which you can solve this problem?

The Possibly Counterfeit Coin Problem

- With n weighings, we have a strategy for finding which of $3^n - 1$ coins, if any, is heavier.
- If $n = 0$, then we can check $3^0 - 1 = 0$ coins and determine they are all real.
- Otherwise:
  - Split the coins into groups of size 3^{n-1}, 3^{n-1}, and $3^{n-1} - 1$. Call them A, B, and C.
  - Weigh A vs. B.
  - If A or B is heavier than the other, one of the $3^{\mathrm{n}-1}$ coins in it is counterfeit. We can find it in $n - 1$ weighings.
  - Otherwise, nothing in A or B is counterfeit. Recursively check the $3^{n-1} - 1$ coins using $n - 1$ weighings.

Is this solution optimal?

Using Our Model

- Let's use the same reasoning as before.
- How many different inputs are there if there are *k* coins?
  - **Answer:** $k + 1$: one for each coin, plus one for “no coin is counterfeit.”
- How many answer nodes are in an algorithm that makes *n* comparisons?
  - **Answer:** 3^n
- Solving $k + 1 = 3^n$, we get $k = 3^n - 1$.
- **Our algorithm has to be optimal!**

Why All This Matters

- We've spent the last few weeks exploring proof techniques and defining mathematical structures.
- These techniques make it possible to reason about fundamental questions in computing.
- In about a week, we'll begin exploring more elaborate models of computation using similar techniques.

Next Time

- **Mathematical Logic**
  - How do we start formalizing our intuitions about mathematical truth?
  - How do we justify proofs by contradiction and contrapositive?