A scientist gave a lecture on astronomy. He described how the earth orbits the sun, which, in turn, orbits the center of a vast collection of stars called our galaxy.

Afterward, a lady got up and said, “That’s rubbish. The world is really a flat plate supported on the back of a giant turtle.”

The scientist gave a superior smile before replying, “What’s the turtle standing on?” “Another turtle,” was the reply.

“And that turtle?” “You’re very clever, young man, very clever”, said the old lady. “But it’s turtles all the way down!”

---

**Overview: Reasoning about programs**

Our broad problem: code is unlikely to be correct if we don’t have good reasons for believing it works

- We need clear problem statements
- We need a rigorous way to convince ourselves that what we wrote solves the problem

But reasoning about programs can be hard

- Especially with recursion, concurrency
- Today: focus on induction and recursion

---

**Better argument, using mathematical induction**

- Domino 0 falls because we push it over (Base Case)
- Assume induction hypotheses P(0), ..., P(k), for arbitrary k ≥ 0: P(i): domino i falls over
- Because domino k’s length is larger than inter-domino spacing, it will knock over domino k+1 (Inductive Step)
- Because k is arbitrary, conclude that all dominos will fall over (Conclusion)

This is an inductive argument

- Not only is this argument more compact, it works for an arbitrary number of dominos!

---

**Can define a function in various ways: Example**

Let $S : \text{int} \rightarrow \text{int}$ be the function where $S(n)$ is the sum of the integers from 0 to n. For example,

\[
S(0) = 0 \quad S(3) = 0 + 1 + 2 + 3 = 6
\]

- Definition, iterative form: $S(n) = 0 + 1 + \ldots + n = \sum_{i=0}^{n} i$
- Definition, recursive form: $S(0) = 0 \quad S(n) = n + S(n-1)$ for $n > 0$
- Definition: closed form (doesn’t use recursion or iteration): $S(n) = n(n+1)/2$

---

**Don’t cover all slides!**

This 50-minute lecture cannot cover all the material. But you are responsible for it. Please study it all.

1. Defining functions recursively, iteratively, and in closed form
2. Induction over the integers
3. Proving recursive methods correct using induction
4. Weak versus strong induction

Prelim 2 is coming up: 20 November. Review handout is on the course website (exam page), along with past prelims and statement about conflicts
How do we know these three definitions are equivalent?

- Definition, iterative form: \[ S(n) = 0 + 1 + \ldots + n = \sum_{i=0}^{n} i \]
- Definition, recursive form: \[ S(0) = 0 \quad S(n) = n + S(n-1) \text{ for } n > 0 \]
- Definition: closed form: \[ S(n) = \frac{n(n+1)}{2} \]

How can we prove they are equivalent?

(Strong) Induction over integers

To prove that property \( P(n) \) holds for all integers \( n \geq 0 \),
1. Base case: Prove that \( P(0) \) is true
2. Inductive Case: Assume inductive hypotheses \( P(0), \ldots, P(k) \) for an arbitrary integer \( k \geq 0 \), prove \( P(k+1) \)
3. Conclusion: \( P(n) \) holds for all integers \( n \geq 0 \)

Example proof by mathematical induction (2 slides)

Example proof by mathematical induction

Write \( S1 \) as a Java function

```java
/** Return S2(n): n(n+1)/2 */
public int S1(int n) {
    if (n == 0) return 0;
    return n + S1(n-1);
}
```

Proving a recursive function correct using induction

```java
/** Return S2(n) */
public int S1(int n) {
    if (n == 0) return 0;
    return n + S1(n-1);
    
    // Base case: The call S1(0) returns 0, which is S(0).
    
    // Recursive case, n > 0: Assume that the inner call does what the spec says: return (n-1)(n)/2.
    // Arithmetic then shows that correct value is returned:
    // n + S1(n-1) = <assumption> + S2(n-1) = <arithmetics> = n(n+1)/2.

    // This is how we explained how to understand a recursive method.
    // You see now that we understand it (prove it correct) using induction!
```
(Strong) Induction over integers

1. Base case: Prove that $P(0)$ is true.

2. Inductive Step: Assume inductive hypotheses $P(0), \ldots, P(k)$ for an arbitrary integer $k \geq 0$, prove $P(k+1)$.

Conclusion: $P(n)$ holds for all integers $n \geq 0$

Alternative Inductive Step: Assume inductive hypotheses $P(0), \ldots, P(k-1)$ for an arbitrary integer $k > 0$, prove $P(k)

A Note on Base Cases

Sometimes we are interested in showing some proposition is true for integers $\geq b$

Intuition: we knock over domino $b$, and dominoes in front get knocked over; not interested in dominoes $0..b-1$

In general, the base case in induction does not have to be 0

If base case is an integer $b$

Induction proves the proposition for $n = b, b+1, b+2, \ldots$

Does not say anything about $n$ in $0..b-1$

Math induction nonzero base case: stamp problem

Claim: Can make any amount of postage above 8¢ using 3¢ and 5¢ stamps.

Theorem: For $n \geq 8$, $P(n)$ holds:

$P(n)$: There exist non-negative ints $b, c$ such that $n = 3b + 5c$

Base case: True for $n=8$: $8 = 3 + 5$.

Choose $b = 1$ and $c = 1$.

i.e. one 3¢ stamp and one 5¢ stamp

Math induction nonzero base case: stamp problem

Theorem: For $n \geq 8$, $P(n)$ holds:

$P(n)$: There exist non-negative ints $b, c$ such that $n = 3b + 5c$

Induction Hypothesis: $P(8), \ldots, P(k)$ hold for arbitrary $k \geq 8$: $k = 3b + 5c$

Inductive Step: Two cases: $c > 0$ and $c = 0$

Case $c > 0$

There is 5¢ stamp. Replace it by two 3¢ stamps. Get $k+1$.

Formally $k+1 = 3b + 5c + 1 = 3(b+2) + 5(c-1)$

Case $c = 0$, i.e. $k = 3b$. Since $k \geq 8$, $k \geq 9$ also, i.e. there are at least 3 3¢ stamps. Replace them by two 5¢ stamps. Get $k+1$.

Formally, $k+1 = 3b + 1 = 3(b-3) + 5(2)$

Sum of squares: more complex example

Let $SQ : \text{int} \rightarrow \text{int}$ be the function that gives the sum of the squares of integers from 0 to $n$:

- Definition (recursive):
  $SQ(0) = 0$
  $SQ(n) = n^2 + SQ(n-1)$ for $n > 0$

- Definition (iterative form):
  $SQ(n) = 0^2 + 1^2 + \ldots + n^2$

- Equivalent closed-form expression?
  (neither iterative nor recursive)

Closed-Form Expression for $SQ(n)$

Sum of integers in $0..n$ was $n(n+1)/2$ which is a quadratic in $n$, i.e. $O(n^2)$

Inspired guess: perhaps sum of squares of integers between 0 through $n$ is a cubic in $n$

Conjecture: $SQ(n) = an^3 + bn^2 + cn + d$ where $a, b, c, d$ are unknown coefficients

How can we find the four unknowns?

Idea: Use any 4 values of $n$ to generate 4 linear equations, and then solve
**Finding coefficients**

Use \( n = 0, 1, 2, 3 \)

- \( SQ(0) = 0 = a \cdot \cdot 0 + b \cdot \cdot 0 + c \cdot \cdot 0 + d \)
- \( SQ(1) = 1 = a \cdot \cdot 1 + b \cdot \cdot 1 + c \cdot \cdot 1 + d \)
- \( SQ(2) = 5 = a \cdot \cdot 8 + b \cdot \cdot 4 + c \cdot \cdot 2 + d \)
- \( SQ(3) = 14 = a \cdot \cdot 27 + b \cdot \cdot 9 + c \cdot \cdot 3 + d \)

Solve these 4 equations to get:

- \( a = 1/3 \)
- \( b = 1/2 \)
- \( c = 1/6 \)
- \( d = 0 \)

**Is the formula correct?**

This suggests:

- \( SQ(n) = 0^2 + 1^2 + \ldots + n^2 = n^3/3 + n^2/2 + n/6 \)
- \( = (2n^3 + 3n^2 + n)/6 \)
- \( = n(n+1)(2n+1)/6 \)

Question: Is this closed-form solution true for all \( n \)?

- Remember, we used only \( n = 0, 1, 2, 3 \) to determine these coefficients.
- We do not know that the closed-form expression is correct for other values of \( n \).

**One approach**

Try a few other values of \( n \) to see if they work.

- Try \( n = 5 \): \( SQ(n) = 0 + 1 + 4 + 9 + 16 + 25 = 55 \)
- Closed-form expression: \( 5 \cdot \cdot 6 + 11/6 = 55 \)
- Works!

Try some more values...

We can never prove validity of the closed-form solution for all values of \( n \) this way, since there are an infinite number of values of \( n \).

**Are these two functions equal?**

**SQR (R for recursive)**

- \( SQR(0) = 0 \)
- \( SQR(n) = SQR(n-1) + n^2, \ n > 0 \)

**SQC (C for closed-form)**

- \( SQC(n) = n(n+1)(2n+1)/6 \)

**Proof (by Induction)**

- \( SQR(0) = 0 \)
- \( SQR(n) = SQR(n-1) + n^2, \ n > 0 \)
- \( SQC(n) = n(n+1)(2n+1)/6 \)

Here is \( P(n): SQR(n) = SQC(n) \).

In doing such proofs, it is important to:

1. State carefully what you are trying to prove: \( P(n) \) for \( n \geq 0 \).
2. Prove the base case \( P(0) \).
3. Assume the inductive hypotheses \( P(0), \ldots, P(k) \) for unspecified \( k \geq 0 \) and prove \( P(k+1) \).
4. When attempting to prove \( P(k+1) \), expose 1 or more inductive hypotheses \( P(0), \ldots, P(k) \).
Proof (by Induction)

**Base case:** P(0) holds because SQ_R(0) = 0 = SQ_C(0), by definition

**Inductive case:**
Inductive Hypotheses: P(0), …, P(k), k ≥ 0:
Using them, prove P(k+1)

Here is P(n): SQR(n) = SQC(n)

---

**Proof of P(k+1)**

Inductive Hypotheses: P(k), k ≥ 0: 
SQR(k) = SQC(k)

\[
\begin{align*}
\text{SQR}(k+1) &= \text{def of SQR}(k+1) \\
&= \text{SQR}(k) + (k+1)^2 \\
&= \text{Ind Hyp P(k)} \\
&= \text{SQC}(k) + (k+1)^2 \\
&= \text{def of SQC}(k) \\
&= k(k+1)(2k+1)/6 + (k+1)^2 \\
&= \text{def of SQC}(k+1) \\
&= \text{SQC}(k+1)
\end{align*}
\]

Don’t just flounder around.
Opportunity directed.

---

**Theorem.** Every integer > 1 is divisible by a prime.

**Restatement.** For all n >= 2, P(n) holds:

P(n): n is divisible by a prime.

**Proof**

**Base case:** P(2): 2 is a prime, and it divides itself.

**Inductive case:** Assume P(2), …, P(k) and prove P(k+1).

Case 1. k+1 is prime, so it is divisible by itself.
Case 2. k+1 is composite – it has a divisor d in 2..k.
P(d) holds, so some prime p divides d.
Since p divides d and d divides k+1, p divides k+1.
So k+1 is divisible by a prime.

\[k+1 = d*c1 = p*e2*c2\] (for some c1 and c2)

---

**Strong versus weak induction**

In our first proofs, in inductive case, we assumed P(0), …, P(k) but used only P(k) in the proof. Didn’t have to assume P(0), …, P(k-1).

That’s using weak induction.

In the last proof, in inductive case, we assumed P(0), …, P(k) and actually used P(d), where d < k, in the proof.

That’s strong induction.

Strong induction and weak induction are equally powerful — one can turn a strong-induction proof into a weak-induction proof with an appropriate change in what P(n) is.

Don’t be concerned about this difference!

---

**Complete binary trees (cbtrees)**

**Theorem:**

A depth-d cbtree has \(2^d\) leaves and \(2^{d+1} - 1\) nodes.

**Proof by induction on d.**
P(d): A depth-d cbtree has \(2^d\) leaves and \(2^{d+1} - 1\) nodes.

**Base case:** d = 0. A cbtree of depth 0 consists of one node. It is a leaf. There are \(2^0 = 1\) leaves and \(2^1 - 1 = 1\) nodes.
Proof of $P(k+1)$ for cbtrees

Induction hypotheses $P(0), \ldots, P(k)$, for $k \geq 0$.

$P(k)$: A depth-$k$ cbtree has $2^k$ leaves and $2^{k+1} - 1$ nodes.

Proof of $P(k+1)$. A cbtree of depth $k+1$ arises by adding 2 children to each of the leaves of a depth-$k$ cbtree. Thus, the depth $k+1$ tree has $2^{k+1}$ leaves.

The number of nodes is now $2^{k+1} - 2^{k+2} - 1 + 2^{k+1} = 2^{k+2} - 1$.

What are the “dominos”?

- In some problems, it can be tricky to determine how to set up the induction.
- This is particularly true for geometric problems that can be attacked using induction.

Tiling Elaine’s kitchen

Kitchen in Gries’s house is 8 x 8. A refrigerator sits on one of the 1 x 1 squares.

His wife, Elaine, wants the kitchen tiled with el-shaped tiles – every square except where the refrigerator sits should be tiled.

Base case

The 1 x 1 kitchen can be tiled by putting 0 tiles down. The refrigerator sits on the single square.

Inductive case

$P(k)$: A $2^k \times 2^k$ kitchen with one square covered can be tiled.

In order to use the inductive hypothesis $P(k)$, we have to expose kitchens of size $2^k \times 2^k$. How do we draw them?

Proof outline

Consider kitchens of size $2^n \times 2^n$ for $n = 0, 1, 2, \ldots$

$P(n)$: A $2^n \times 2^n$ kitchen with one square covered can be tiled.

- Base case: Show that tiling is possible for 1 x 1 board.
- Induction Hypothesis: for some $k \geq 0$, $P(k)$ holds.
- Prove $P(k+1)$ assuming $P(k)$.

The 8 x 8 kitchen is a special case of this argument. We will have proven the 8 x 8 special case by solving a more general problem!
Recursive case

P(k): A $2^k \times 2^k$ kitchen with one square covered can be tiled.

By P(k), the upper right kitchen can be tiled.
What about the other 3?

Recursive case

P(k): A $2^k \times 2^k$ kitchen with one square covered can be tiled.

Put in one tile so that each $2^k \times 2^k$ kitchen has one square covered. Now, by P(k), all four $2^k \times 2^k$ kitchens can be tiled.

When induction fails

- Sometimes an inductive proof strategy for some proposition may fail.
- This does not necessarily mean that the proposition is wrong.
  - It may just mean that the particular inductive strategy you are using is the wrong choice.
  - A different induction hypothesis (or a different proof strategy altogether) may succeed.

Tiling example (poor strategy)

Try a different induction strategy:

- Proposition: Any $n \times n$ board with one square covered can be tiled.
- Problem: A $3 \times 3$ board with one square covered has 8 remaining squares, but the tiles have 3 squares; tiling is impossible.
- Thus, any attempt to give an inductive proof of this proposition must fail.

- Note that this failed proof does not tell us anything about the $8 \times 8$ case.

A seemingly similar tiling problem

- A chessboard has opposite corners cut out of it. Can the remaining board be tiled using tiles of the shape shown in the picture (rotation allowed)?
- Induction fails here. Why? (Well...for one thing, this board can't be tiled with dominos.)

Procedure to tile a kitchen

Use abstraction to help focus attention.

```java
/** Tile a kitchen of size $2^k \times 2^k$.
 * Precondition: $k \geq 0$ and one square is covered */
public static void tile(int k, Positions p) {
    if (k == 0) return;
    View the kitchen as 4 kitchens of size $2^{k-1} \times 2^{k-1}$;
    Place one tile so that all 4 kitchens have one tile covered.
    tile(k-1, positions for upper left kitchen);
    tile(k-1, positions for upper right kitchen);
    tile(k-1, positions for lower left kitchen);
    tile(k-1, positions for lower right kitchen);
}
```

p gives 2 things:
1. Position of top left corner of kitchen
2. Position of covered square
Procedure to tile a kitchen

Theorem. For all n ≥ 0, P(n) holds:

P(n): The call tile(n, p) tiles the kitchen given by n and p.

Proof by induction on n.

Base case, n = 0. It’s a 1 x 1 covered square. No tiles need to be laid, and the procedure doesn’t lay any.

** Tile a kitchen of size 2^k x 2^k. 
Precondition: k >= 0 and one square is covered */

public static void tile(int k, Positions p) {
    if (k == 0)
        return;
    ...
}

Theorem. For all n ≥ 0, P(n) holds:

P(n): The call tile(n, p) tiles the kitchen given by n and p.

Proof by induction on n.

Base case, n = 0. It’s a 1 x 1 covered square. No tiles need to be laid, and the procedure doesn’t lay any.

** Tile a kitchen of size 2^k x 2^k. 
Precondition: k >= 0 and one square is covered */

public static void tile(int k, Positions p) {
    if (k == 0)
        return;
    .... }

Proving a recursive function correct

/** = the number of ‘e’s in s */

public static int nE(String s) { 
    if (s.length == 0) 
        return 0; // base case 
    // {s has at least 1 char} 
    return (s[0] == ’e’ ? 1 : 0)  + nE(s[1..])
}

Theorem. For all n, n ≥ 0, P(n) holds:

P(n): For s a string of length n, nE(s) = number of ‘e’s in s.

Proof by induction on n.

Base case. If n = 0, the call nE(s) returns 0, which is the number of ‘e’s in s, the empty string. So P(0) holds.

/** = the number of ’e’ s in s */

public static int nE(String s) { 
    if (s.length == 0) 
        return 0; // base case 
    // {s has at least 1 char} 
    return (s[0] == ’e’ ? 1 : 0)  + nE(s[1..])
}

Theorem. For all n, n ≥ 0, P(n) holds:

P(n): For s a string of length n, nE(s) = number of ‘e’s in s.

Proof by induction on n.

Base case. If n = 0, the call nE(s) returns 0, which is the number of ‘e’s in s, the empty string. So P(0) holds.

Inductive case. Assume P(k-1) for k > 0, Prove P(k)

public static void tile(int k, Positions p) {
    if (k == 0)
        return;
    // View the kitchen as 4 kitchens of size 2^(k-1) x 2^(k-1) ;
    tile(k-1, p for upper left kitchen);
    tile(k-1, p for upper right kitchen);
    tile(k-1, p for lower left kitchen);
    tile(k-1, p for lower right kitchen);
}

There are four recursive calls. Each, by the inductive hypothesis P(k-1), tiles a kitchen … etc.

P(k): For s a string of length k, nE(s) = number of ’e’s in s

/** = the number of ’e’s in s */

public static int nE(String s) { 
    if (s.length == 0) 
        return 0; // base case 
    // {s has at least 1 char} 
    return (s[0] == ’e’ ? 1 : 0)  + nE(s[1..])
}

Theorem. For all n, n ≥ 0, P(n) holds:

P(n): For s a string of length n, nE(s) = number of ’e’s in s.

Proof by induction on n.

Base case. If n = 0, the call nE(s) returns 0, which is the number of ‘e’s in s, the empty string. So P(0) holds.

Inductive case: Assume P(k), k ≥ 0, and prove P(k+1).

Suppose s has length k+1. Then s[1..] has length k. By the inductive hypothesis P(k),

nE(s[1..]) = number of ’e’s in s[1..].

Thus, the statement returns the number of ’e’s in s.

Conclusion

- Induction is a powerful proof technique
- Recursion is a powerful programming technique
- Induction and recursion are closely related
  - We can use induction to prove correctness and complexity results about recursive methods