A well-known scientist (Bertrand Russell?) once gave a public lecture on astronomy. He described how the earth orbits the sun and how the sun, in turn, orbits the center of a vast collection of stars called our galaxy.

Afterward, a little old lady at the back of the room got up and said: "You told us 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 is 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!"

INDUCTION

Lecture 21
CS2110 – Spring 2014

We won'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 and in closed form
2. Weak induction over the integers
3. Proving recursive methods correct using induction
4. Strong induction

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

Overview: Reasoning about Programs

Iteration (loops)

- Loop invariants are the main tool in proving correctness of loops

Recursion

- A programming strategy that solves a problem by reducing it to simpler or smaller instance(s) of the same problem

Induction

- A mathematical strategy for proving statements about natural numbers 0,1,2, ... (or more generally, about inductively defined objects)

Recursion and induction are closely related

Induction can be used to establish the *correctness* and *complexity* of programs

Defining Functions

It is often useful to describe a function in different ways

- Let S : int → 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:
  $S(n) = 0+1+ \ldots + n$
  $= \text{sum}_{i=0}^{n}\ i$
- Another characterization: closed form
  $S(n) = n(n+1)/2$

Sum of Squares: more complex example

Let SQ : int → int be the function that gives the sum of the squares of integers from 0 to n:

$SQ(0) = 0$
$SQ(3) = 0^2 + 1^2 + 2^2 + 3^2 = 14$

Definition (iterative form):

$SQ(n) = 0^2 + 1^2 + \ldots + n^2$

Is there an equivalent closed-form expression?

Closed-Form Expression for SQ(n)

7

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

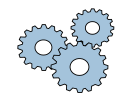
Finding Coefficients

8

$SQ(n) = 0^2 + 1^2 + \ldots + n^2 = an^3 + bn^2 + cn + d$

Use n = 0, 1, 2, 3

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



Solve these 4 equations to get

$a = 1/3 \quad b = 1/2 \quad c = 1/6 \quad d = 0$

Is the Formula Correct?

9

This suggests

$$\begin{aligned} SQ(n) &= 0^2 + 1^2 + \ldots + n^2 \\ &= n^3/3 + n^2/2 + n/6 \\ &= (2n^3 + 3n^2 + n)/6 \\ &= n(n+1)(2n+1)/6 \end{aligned}$$

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 valid for other values of n

One Approach

10

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: $5{\cdot}6{\cdot}11/6 = 55$
- 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

A Recursive Definition

11

To solve this problem, let's express SQ(n) differently:

- $SQ(n) = \boxed{0^2 + 1^2 + \ldots + (n-1)^2} + n^2$
- The part in the box is just SQ(n-1)

This leads to the following *recursive* definition

- $SQ(0) = 0$ ← Base Case
- $SQ(n) = SQ(n-1) + n^2, \; n > 0$ ← Recursive Case

$$\begin{aligned} \text{Thus, } SQ(4) &= SQ(3) + 4^2 \\ &= SQ(2) + 3^2 + 4^2 \\ &= SQ(1) + 2^2 + 3^2 + 4^2 \\ &= SQ(0) + 1^2 + 2^2 + 3^2 + 4^2 \\ &= 0 + 1^2 + 2^2 + 3^2 + 4^2 \end{aligned}$$

Are These Two Functions Equal?

12

SQ_r (r = recursive)

$SQ_r(0) = 0$

$SQ_r(n) = SQ_r(n-1) + n^2, \quad n > 0$

SQ_c (c = closed-form)

$SQ_c(n) = n(n+1)(2n+1)/6$

Induction over Integers

13

To prove that some property P(n) holds for all integers n ≥ 0,

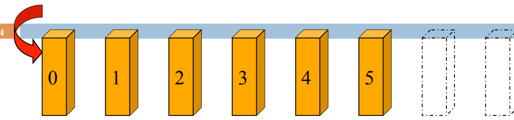
1. Base case: Prove that P(0) is true
2. Induction Step: Assume inductive hypothesis P(k) for an *unspecified* integer k >= 0, prove P(k+1).

- Conclusion: Because we could have picked any k, we conclude that P(n) holds for all integers n ≥ 0

Alternative Induction Step: Assume inductive hypothesis P(k-1) for an integer k > 0, prove P(k)

Dominos

14



Assume equally spaced dominos, and assume that spacing between dominos is less than domino length

How would you argue that all dominos would fall?

- Dumb argument:
  - Domino 0 falls because we push it over
  - Domino 0 hits domino 1, therefore domino 1 falls
  - Domino 1 hits domino 2, therefore domino 2 falls
  - Domino 2 hits domino 3, therefore domino 3 falls
  - ... Go on forever …
- Can we make a more compact argument?

Better Argument, Using Induction

15

Argument:

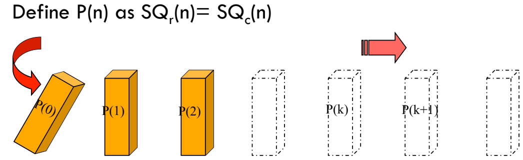
- Domino 0 falls because we push it over (Base Case)
- Assume induction hypothesis P(k), for k ≥ 0: P(k): domino k falls over
- Because domino k's length is larger than inter-domino spacing, it will knock over domino k+1 (Inductive Step)
- Because we could have picked any domino to be domino k, we conclude that all dominos will fall over (Conclusion)

- This is an inductive argument
- This version is called *weak induction*
  - *Strong induction* is discussed later
- Not only is this argument more compact, it works for an arbitrary number of dominoes!

$SQ_r(n) = SQ_c(n)$ for all n?

16

Define P(n) as $SQ_r(n) = SQ_c(n)$



- Prove P(0)
- Assume P(k) for unspecified k ≥ 0 and prove P(k+1) under this assumption

Proof (by Induction)

17

$SQ_r(0) = 0$
$SQ_r(n) = SQ_r(n-1) + n^2, \quad n > 0$

$SQ_c(n) = n(n+1)(2n+1)/6$

Here is P(n): $SQ_r(n) = SQ_c(n)$

In doing such proofs, it is important to

1. State carefully what you are trying to prove: P(n) for n ≥ 0
2. Prove the base case P(0).
3. Assume the inductive hypothesis P(k) for arbitrary k ≥ 0 and prove P(k+1).

The proof of P(k+1) can usually be easily developed. We see this on next slide

Other variations. E.g. Can use more base cases. E.g. Can prove P(n) for n ≥ 5

Proof (by Induction)

18

$SQ_r(0) = 0$
$SQ_r(n) = SQ_r(n-1) + n^2, \quad n > 0$

$SQ_c(n) = n(n+1)(2n+1)/6$

Here is P(n) : $SQ_r(n) = SQ_c(n)$

Base case: P(0) holds because $SQ_r(0) = 0 = SQ_c(0)$ by definition

Induction Hypothesis: P(k), k ≥ 0: $SQ_r(k) = SQ_c(k)$

Proof of P(k+1)

19

Induction Hypothesis: P(k), $k \ge 0$: $SQ_r(k) = SQ_c(k)$

$SQ_r(k+1)$

<definition of $SQ_r(k+1)$>

$= SQ_r(k) + (k+1)^2$

<Induction Hypothesis>

$= SQ_c(k) + (k+1)^2$

<definition of $SQ_c(k)$>

$= k(k+1)(2k+1)/6 + (k+1)^2$

<algebra ---we leave this to you>

$= (k+1)(k+2)(2k+3)/6$

<definition of $SQ_c(k+1)$>

$= SQ_c(k+1)$

$SQ_r(0) = 0$
$SQ_r(n) = SQ_r(n-1) + n^2, \quad n > 0$

$SQ_c(n) = n(n+1)(2n+1)/6$

Don't just flounder around.
Opportunity directed.
Expose induction hypothesis

Complete binary trees (cbtrees)

20

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.

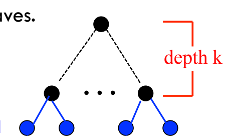
Proof of P(k+1) for cbtrees

21

Induction hypothesis P(k), for $k \ge 0$.

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

$2^{k+1}-1 + 2^{k+1}$

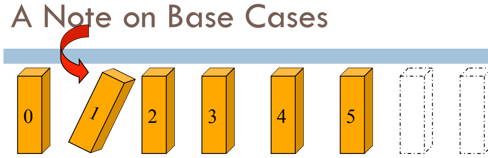
$= 2^{k+2} - 1$

2^k leaves

2^{k+1} nodes added

A Note on Base Cases

22



Sometimes we are interested in showing some proposition is true for integers ≥ 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, …
  - Does not say anything about n in 0..b-1

Weak Induction: Nonzero Base Case

23

Claim: You can make any amount of postage above 8¢ with some combination of 3¢ and 5¢ stamps.

Theorem: For $n \ge 8$, 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

Weak Induction: Nonzero Base Case

24

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

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

Induction Hypothesis: P(k) holds for arbitrary $k \ge 8$: 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 >= 8, k >= 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)

What are the "Dominos"?

25

- 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

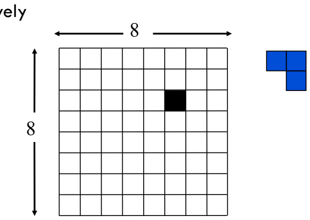
Tiling Elaine's kitchen

26

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.

Gries does is inductively



Proof Outline

27

Consider kitchens of size 2^n x 2^n for n = 0, 1, 2,…

P(n): A 2^n x 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 ≥ 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!

Base case

28

The 1 x 1 kitchen can be tiled by putting 0 tiles down.

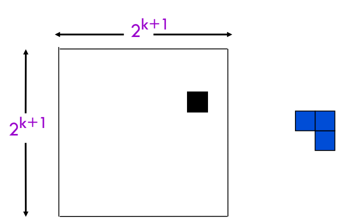
The refrigerator sits on the single square

Inductive case

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

29

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

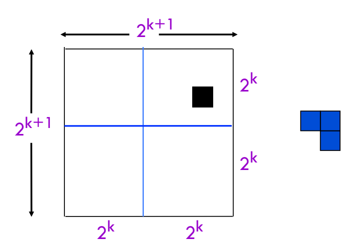


Recursive case

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

30

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

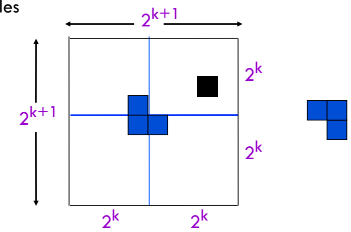


Recursive case

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

31

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



When Induction Fails

32

- 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)

33

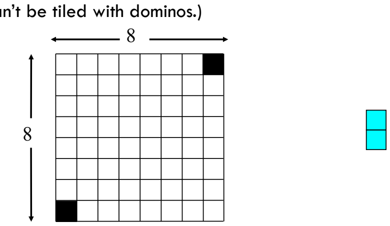
Try a different induction strategy

- Proposition
  - Any n x n board with one square covered can be tiled
- Problem
  - A 3 x 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 8x8 case

A Seemingly Similar Tiling Problem

34

- 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.)



Proving a recursive function correct

35

```
/** = 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.

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

36

```
/** = 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..])
}
```

Inductive case: Assume P(k), $k \geq 0$, 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.

Procedure to tile a kitchen

Use abstraction to help focus attention

37

```
/** 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;
    View the kitchen as 4 kitchens of size 2^(k-1) x 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

38

Theorem. For all $n \geq 0$, 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.

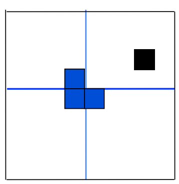
```
/** Tile a kitchen of size 2^k x 2^k .
    Precondition: k >= 0 and one square is covered */
public static void tile(int k, Positions p) {
    if (k == 0) return;
    …
}
```

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

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

39

```
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) ;
    Place one tile so that all 4 kitchens have one tile covered.
    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.

Strong Induction

40

We want to prove that some property P holds for all n

- Weak induction
  - P(0): Show that property P is true for 0
  - P(k) ⇒ P(k+1): Show that if property P is true for k >= 0, it is true for k+1
  - Conclude that P(n) holds for all n
- Strong induction
  - P(0): Show that property P is true for 0
  - P(0) and P(1) && … && P(k) ⇒ P(k+1): show that if P is true for numbers at most k, it is true for k+1
  - Conclude that P(n) holds for all n

The two proof techniques are equally powerful

Proof using strong induction

41

Theorem. Every integer greater than 1 is divisible by a prime.

Restatement. For all n >= 2, P(n) holds:
P(n): n is divisible by a prime.

Inductive case required not P(k) but P(d)

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.

Conclusion

42

- 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