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!"

INDUCTION

Lecture 23
CS2110 – Spring 2015

We may not cover all slides!

2

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

Overview: Reasoning about programs

3

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

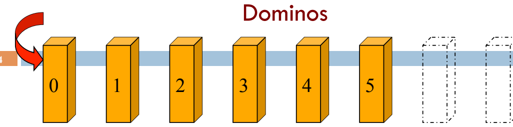
- We need clear problem statements
- We need a rigorous way to convince ourselves that what we wrote solves the problem

But reasoning about programs can be hard

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

Dominos

4



Assume equally spaced dominos, with the spacing between dominos less than domino length

How would you argue that all dominos would fall if we push the first one over??

- 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 mathematical induction

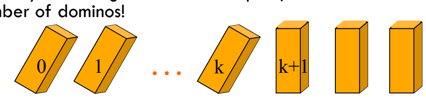
5

- Domino 0 falls because we push it over (Base Case)
- Assume induction hypotheses P(0), …, P(k), for arbitrary $k \geq 0$:

  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!



(Strong) Induction over integers

6

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), …, P(k) for an *arbitrary* integer k >= 0, prove P(k+1)

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

Alternative Inductive Case: Assume inductive hypotheses P(0), … P(k-1) for an *arbitrary* integer k > 0, prove P(k)

Can define a function in various ways: Example

7

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   S(3) = 0+1+2+3 = 6

- Definition, iterative form: S(n) = 0+1+ … + n

  = $\text{sum}_{i=0}^{n}\ i$

- Definition, recursive form:

  S(0) = 0   S(n) = n + S(n-1) for n > 0

- Definition: closed form (doesn't use recursion or iteration):

  S(n) = n(n+1)/2

How do we know these three definitions are equivalent?

8

- Definition, iterative form: S(n) = 0+1+ … + n

  = $\text{sum}_{i=0}^{n}\ i$

- Definition, recursive form:

  S(0) = 0   S(n) = n + S(n-1) for n > 0

- Definition: closed form: S(n) = n(n+1)/2

How can we **prove** they are equivalent?

Example proof by mathematical induction (2 slides)

9

- Definition, recursive form:

  S1(0) = 0   S1(n) = n + S1(n-1) for n > 0

- Definition: closed form:

  S2(n) = n(n+1)/2

Theorem: P(n) holds for 0 ≤ n, where

P(n) is: S1(n) = S2(n)

We made P(n) explicit, we wrote it down

Base case: n = 0, so
P(0) is S1(0) = S2(0)

S2(0)
= <def of S2>
0(1)/2
= <arith>
0
= <def of S1>
S1(0)

Example proof by mathematical induction

10

Exposed inductive hypothesis

- Definition, recursive form:

  S1(0) = 0   S1(n) = n + S1(n-1) for n > 0

- Definition: closed form:

  S2(n) = n(n+1)/2

Theorem: P(n) holds for 0 ≤ n,

where P(n) is: S1(n) = S2(n)

Inductive case:
For arbitrary k,
assume P(0), … P(k)
and prove P(k+1)

S1(k+1)
= <def of S1>
k+1 + S1(k)
= <ind hyp P1(k)>
k + 1 + S2(k)
= <def of S2>
k + 1 + k(k+1)/2
= <arith>
(k+1)(k+2)/2
= <def of S2>
S2(k+1)

Write S1 as a Java function

11

S1(0) = 0
S1(n) = n + S1(n-1) for n > 0
S2(n) = n(n+1)/2
Prove P(n) for all n: S1(n) = S2(n)

```
/** Return S2(n): n(n+1)/2 */
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>
n + (n-1)(n)/2
= <arithmetic>
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!

Proving a recursive function correct using induction

12

```
/** Return S2(n)*/
public int S1(int n) {
    …
}
```

Write P(k): S1(k) = S2(k)

1. Make sure you there is a good specification

2. Prove the base cases –those values of n that do not involve recursion.

3. Prove the recursive cases.
A. Assume P(k) for 0 ≤ k < n (recursive calls do what spec says)
B. Prove that all recursive calls have argument < n.
C. Knowing that each recursive call S1(k) returns S2(k), prove that the body does its job.

(Strong) Induction over integers

13

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

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

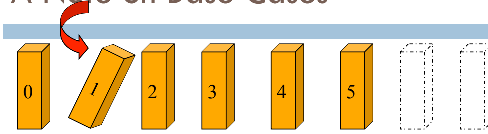
2. Inductive Step: Assume inductive hypotheses P(0), … P(k) for an *arbitrary* integer k >= 0, prove P(k+1).

- Conclusion: P(n) holds for all integers n ≥ 0

Alternative Induction Step: Assume inductive hypotheses P(0), … P(k-1) for an *arbitrary* integer k > 0, prove P(k)

A Note on Base Cases

14



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

Math induction nonzero base case: stamp problem

15

Claim: Can make any amount of postage above 7¢ using 3¢ and 5¢ stamps.

Theorem: For n ≥ 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

16

Theorem: For n ≥ 8, P(n) holds:

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

Induction Hypothesis: P(8), …, P(k) hold
for arbitrary k ≥ 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)

Sum of squares: more complex example

17

Let SQ : int → 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)

18

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?

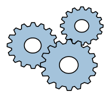
Idea: Use any 4 values of n to generate 4 linear equations, and then solve

Finding coefficients

19

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

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



Solve these 4 equations to get

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

Is the formula correct?

20

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 correct for other values of n

One approach

21

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: 5·6·11/6 = 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?

22

SQR (R for recursive)

$SQR(0) = 0$

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

SQC (C for closed-form)

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

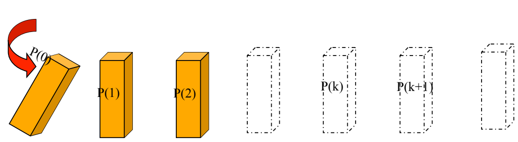
Theorem? SQR(n) = SQC(n) for all n?

23

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

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

Define P(n) as SQR(n)= SQC(n)



- Prove P(0)
- Assume P(0), …, P(k) for arbitrary k ≥ 0, prove P(k+1) under these assumptions

Proof (by Induction)

24

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

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

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 ≥ 0
2. Prove the base case P(0).
3. Assume the inductive hypotheses P(0), …, P(k) for arbitrary k ≥ 0 and prove P(k+1).
4. When attempting to prove P(k+1), expose 1 or more inductive hypotheses P(0), …, P(k).

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)

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)

Proof of P(k+1)

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

SQR(k+1)
= <def of SQR(k+1)>
SQR(k) + $(k+1)^2$
= <Ind Hyp P(k)>
SQC(k) + $(k+1)^2$
= <def of SQC(k)>
k(k+1)(2k+1)/6 + $(k+1)^2$
= <algebra ---we leave this to you>
(k+1)(k+2)(2k+3)/6
= <def of SQC(k+1)>
SQC(k+1)

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

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

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

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*c2*c1 (for some c1 and c2)

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

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!

Strong versus weak induction

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

- Weak induction
  - Base case: Prove P(0)
  - Inductive case:
    Assume P(k) for arbitrary k ≥ 0 and prove P(k+1)
- Strong induction
  - Base case: Prove P(0)
  - Inductive case:
    Assume P(0), …, P(k) for arbitrary k ≥ 0 and prove P(k+1)

The two proof techniques are equally powerful.
Somebody proved that.

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

31

Induction hypotheses P(0), …, 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.

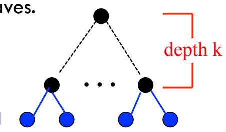
The number of nodes is now

$2^{k+1}–1 + 2^{k+1}$

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

2^k leaves

2^{k+1} nodes added



What are the "dominos"?

32

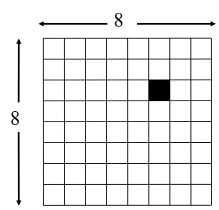
- In some problems, it can be tricky to determine how to set up the induction
- This is particularly true for geometric problems that can be attacked using induction

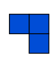
Tiling Elaine's kitchen

33

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.





Proof outline

34

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

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!

Base case

35

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

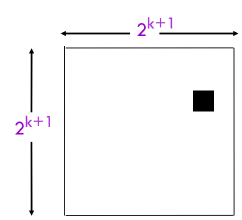
The refrigerator sits on the single square

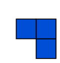
Inductive case

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

36

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?



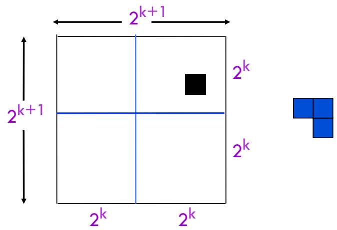


Recursive case

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

37

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

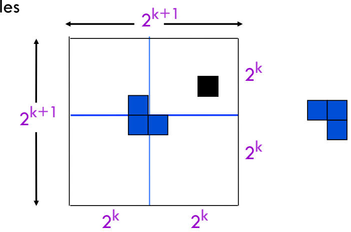


Recursive case

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

38

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

39

- 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)

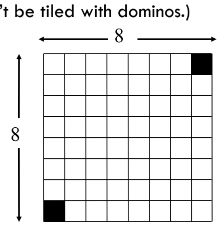
40

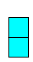
Try a different induction strategy
- Proposition
  - Any n x n board with one square covered can be tiled
- Problem
  - A 3 x 3 board with one square covered has 8 remaining squares, but the tiles have 3 squares; tiling is impossible
- Thus, any attempt to give an inductive proof of this proposition *must fail*
- Note that this failed proof does not tell us anything about the 8x8 case

A seemingly similar tiling problem

41

- 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

42

```
/** 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

43

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.

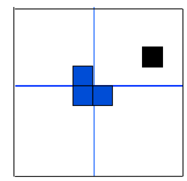
```
/** Tile a kitchen of size 2^k x 2^k .
    Precondition: k >= 0 and one square is covered */
public static void tile(int k, Positions p) {
    if (k == 0) return;
    …
}
```

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)

44

```
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.

Proving a recursive function correct

45

```
/** = 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

46

```
/** = 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.

Conclusion

47

- 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