Chapter 5 / 4: induction and recursion

pages 307/263 and 339/295

two-step approach to problem solving:

1. solve smallest problem instance
   "basis" of induction, "bottom" of recursion
2. show either how
   - solution of instance of size k leads to solution of instance of size $k+1$ ←induction

   or how
   - instance of size k can be solved by solving instance(s) of size $< k$ ←recursion

without basis or bottom:

step 2 useless and worthless

Section 5.1 / 4.1: mathematical induction

pages 307-325 /263-279

let $P(n)$ be propositional function for $n \in \mathbf{Z}_{\geq 1}$ (thus $\forall\, n \in \mathbf{Z}_{\geq 1}\; P(n) = \text{true} \vee P(n) = \text{false}$)

to prove the statement

$$\forall\, n \in \mathbf{Z}_{\geq 1}\; P(n) = \text{true}$$

it suffices to prove that:

1. $P(1) = \text{true}$
   (*basis of the induction*, or *basis step*)
2. $\forall\, k \in \mathbf{Z}_{\geq 1}$: if $P(k) = \text{true}$
   then $P(k+1) = \text{true}$ (*inductive step*)

(if $\exists\, n$ with $P(n) = \text{false}$, let s be the smallest, then $s \neq 1$, so $s > 1$, so $P(s-1) = \text{true}$, so $P(s) = \text{true}$)

Section 5.1 / 4.1: mathematical induction

pages 307-325 /263-279

let $P(n)$ be propositional function for $n \in \mathbf{Z}_{\geq b}$

(thus $\forall\, n \in \mathbf{Z}_{\geq b}\; P(n) = \text{true} \vee P(n) = \text{false}$)

to prove the statement

$$\forall\, n \in \mathbf{Z}_{\geq b}\; P(n) = \text{true}$$

it suffices to prove that:

1. $P(b) = \text{true}$
   (*basis of the induction*, or *basis step*)
2. $\forall\, k \in \mathbf{Z}_{\geq b}$: if $P(k) = \text{true}$
   then $P(k+1) = \text{true}$ (*inductive step*)

(if $\exists\, n$ with $P(n) = \text{false}$, let s be the smallest, then $s \neq b$, so $s > b$, so $P(s-1) = \text{true}$, so $P(s) = \text{true}$)

Mathematical induction (MI), examples

pages 307-325 /263-279

mathematical induction great way to prove known results, hardly useful to derive them

page 312/267

- seen that $\sum_{i=0}^{k} i = k(k+1)/2$,
  now use MI to prove it (again)

page 314/270

- seen that $\sum_{i=0}^{k} r^i = \dfrac{r^{k+1}-1}{r-1}$ $(r \neq 1)$,
  now use MI to prove it (again)

page 315/271

- not just equalities: $n < 2^n$, prove with MI

page 166/157

- if suspect that $\sum_{i=0}^{k} i^2 = k(k+1)(2k+1)/6$,
  we can use MI to prove it
  but how does one find that formula?

page 281 exerc 39-41 / 47-49

- careful with buggy proofs (all horses have same color) not in book

Finding $S(k) = \sum_{i=0}^{k} i^2 = k(k+1)(2k+1)/6$

- from $\int_0^k x^2 dx = k^3/3$ we suspect that $S(k) = k^3/3 + ak^2 + bk + c$ for $a, b, c \in \mathbf{R}$
- $S(0) = 0 \Rightarrow c = 0$
- $S(1) = 1 \Rightarrow 1/3 + a + b = 1$
- $S(2) = 5 \Rightarrow 8/3 + 4a + 2b = 5$

subtract $1/3+a+b=1$ twice from $8/3+4a+2b=5$

$\Rightarrow 6/3+2a = 3 \Rightarrow a = ½ \Rightarrow b = 1/6$

$\Rightarrow$ we suspect $S(k) = k^3/3 + k^2/2+k/6$

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

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

now we still need to prove it…

page
325/280
exerc 5

MI proof of $S(k)=\sum_{i=0}^{k} i^2 = k(k+1)(2k+1)/6$

use familiar two step induction approach and
$P(n)$: "$S(n)=n(n+1)(2n+1)/6$" (for $n \geq 0$)

1. basis of induction, proof that $P(0)$ = true:
   because $S(0) = 0$ and $0(0+1)(2*0+1) = 0$
   it follows that $P(0)$ = true
2. inductive step: assume $P(k)$ = true
   $S(k+1) = S(k) + (k+1)^2$ (from definition)
   $= k(k+1)(2k+1)/6 + (k+1)^2$

(we use $P(k)$ = true: $S(k)=k(k+1)(2k+1)/6$)

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

$= (k+1)(2k^2+7k+6)/6 = (k+1)(k+2)(2k+3)/6$

which shows that $P(k+1)$ = true

Section 5.2 / 4.2: strong induction

pages 328-336 /283-291

let $P(n)$ be propositional function for $n \in \mathbf{Z}_{\geq b}$
(thus $\forall\ n \in \mathbf{Z}_{\geq b}\ P(n) = \text{true} \vee P(n) = \text{false}$)

to prove the statement

$$\forall\ n \in \mathbf{Z}_{\geq b}\ P(n) = \text{true}$$

it suffices to prove that:

1. $P(b) = \text{true}$
   (the *basis of the induction*, or *basis step*)
2. $\forall\ k \in \mathbf{Z}_{\geq b}$: if $P(b)=P(b+1)= \ldots =P(k) = \text{true}$
   then $P(k+1) = \text{true}$ (the *inductive step*)

(if $\exists\ n$ with $P(n) = \text{false}$, let s be the smallest, then
$s \neq b$, so $s > b$, so $\forall i<s\ P(i) = \text{true}$, so $P(s) = \text{true}$)

Strong induction, example

pages 329-336 /284-291

strong induction equally powerful as mathematical induction

page 331/286

example: for $n \in \mathbf{Z}_{>1}$, let $P(n)$ be "n can be written as a product of one or more primes"

- $P(2)$ = true, since 2 is prime
- assume $\forall\ k \in \mathbf{Z}$, $2 \le k < n$, $P(k)$ = true, what about $P(n)$?
  - if n prime, then $P(n)$ = true
  - if n composite: $\exists\ n_1, n_2 \in \mathbf{Z}$ s.t. $n = n_1 n_2$ $2 \le n_1, n_2 < n \Rightarrow P(n_1) = P(n_2) =$ true $\Rightarrow$ n is product of products of primes $\Rightarrow P(n)$ = true (uniqueness: page 270/233)

Strong induction, another example

page 337/292
exerc 10/14

given pile of n stones; for any k a pile of k stones can be split into two non-empty piles of $r > 0$ and $k-r > 0$ stones at cost $r(k-r)$

how to split a single pile of n stones into n "piles" of one stone at lowest total cost $C(n)$?

- $n = 1$: nothing to do at cost $C(1) = 0$
- $n = 2$: one way to split ($2 \rightarrow 1+1$) at cost $C(2) = 1(2-1) = 1$
- $n = 3$: one way to split ($3 \rightarrow 2+1$) at cost $2(3-2)=2$;
  next $2 \rightarrow 1+1$ at cost 1; total cost $C(3) = 2+1 = 3$
- $n = 4$: **either** $4 \rightarrow 2+2$ at cost 4, plus $2C(2) = 2$: $4+2 = 6$;
  or $4 \rightarrow 3+1$ at cost 3 plus $C(3)$: $3+3 = 6$; $\Rightarrow C(4) = 6$
- $n = 5$: **either** $5 \rightarrow 3+2$ at cost 6, plus $C(3)+C(2)=4$, thus 10
  or $5 \rightarrow 4+1$ at cost 4 plus $C(4)=6$: $4+6=10$; $\Rightarrow C(5) = 10$

Splitting piles of stones, continued

page 337/292 exerc 10/14

- $n = 1$: nothing to do at cost $C(1) = 0$
- $n = 2$: one way to split ($2 \rightarrow 1+1$) at cost $C(2) = 1(2-1) = 1$
- $n = 3$: one way to split ($3 \rightarrow 2+1$) at cost $2(3-2)=2$;
  next $2 \rightarrow 1+1$ at cost 1; total cost $C(3) = 2+1 = 3$
- $n = 4$: **either** $4 \rightarrow 2+2$ at cost 4, plus $2C(2) = 2$: $4+2 = 6$;
  or $4 \rightarrow 3+1$ at cost 3 plus $C(3)$: $3+3 = 6$; $\Rightarrow C(4) = 6$
- $n = 5$: **either** $5 \rightarrow 3+2$ at cost 6, plus $C(3)+C(2)=4$, thus 10
  or $5 \rightarrow 4+1$ at cost 4 plus $C(4)=6$: $4+6=10$; $\Rightarrow C(5) = 10$

we suspect that $C(n) = n(n-1)/2$ why so simple?

proof with strong induction: (page 400/359: find combinatorial proof)

- correct for $n = 1$
- split n in $r > 0$ and $n-r > 0$: cost $r(n-r)$ plus

(induction hypothesis) $r(r-1)/2+(n-r)(n-r-1)/2$;

$r(n-r)+r(r-1)/2+(n-r)(n-r-1)/2 = n(n-1)/2$

Common recursive definitions & algorithms

Pages 339-362 /294-321

- factorial function:

  $n! = n * (n-1)!$ for $n > 0$

  with $0! = 1$ this defines $n!$ for $n \geq 0$

  leads to recursive implementation:

  $factorial(n) =$ if $n < 1$ then 1 ← *bottom of recursion*

  else $n*factorial(n-1)$

- fibonacci numbers:

  $f_n = f_{n-1} + f_{n-2}$ for $n > 1$

  with $f_0 = 0$, $f_1 = 1$ this defines f_n for $n \geq 0$

  $fib(n) =$ if $n < 2$ then n ← *bottom of recursion*

  else $fib(n-1)+fib(n-2)$ ← *very bad idea*

More recursive algorithms

pages 353-362 /311-321

recursion often great for lazy programmer
(with proper *bottom of recursion*):

$\Rightarrow$ useful to quickly get working prototypes

- gcd(a,b): (b == 0) ? a : gcd(b,a **mod** b)

(refer to slide 35 of March 27 lecture for division-free method)

pages 355/313

- exponentiation: a^e **mod** m ($e \geq 0$, $m > 1$)

power(a,e,m):

if e equals 0: return(1)

else $t = (power(a,[e/2],m))^2$ **mod** m

if (e is even): return(t)

else return (ta **mod** m)

same computation as before?

- mostly less efficient than iterative version

Exponentiation: a^e **mod** m ($e \geq 0$, $m > 1$)

power(a,e,m):
if e equals 0: return(1)
else $t = (power(a,[e/2],m))^2 \textbf{ mod } m$
if (e is even): return(t)
else return (ta **mod** m)

to calculate 3^5 **mod** 7 (using different colors for variables at different recursion levels)

call *power*(3,5,7): $a = 3$, $e = 5$, $m = 7$
$e = 5 \neq 0$, thus $t = (power(3,[5/2]=2,7))^2 \textbf{ mod } 7$
generates recursive call *power*(3,2,7): $a = 3$, $e = 2$, $m = 7$
$e = 2 \neq 0$, thus $t = (power(3,[2/2]=1,7))^2 \textbf{ mod } 7$
generates recursive call *power*(3,1,7): $a = 3$, $e = 1$, $m = 7$
$e = 1 \neq 0$, thus $t = (power(3,[1/2]=0,7))^2 \textbf{ mod } 7$
generates recursive call *power*(3,0,7): $a = 3$, $e = 0$, $m = 7$
$e = 0$: return 1 (as the value of *power*(3,0,7))
we find $t = (power(3,0,7))^2 \textbf{ mod } 7 = (1)^2 \textbf{ mod } 7 = 1$
$e = 1$ is not even: return ta **mod** $7 = 1*3$ **mod** $7 = 3$ (as the value of *power*(3,1,7))
we find $t = (power(3,1,7))^2 \textbf{ mod } 7 = 3^2 \textbf{ mod } 7 = 2$
$e = 2$ is even: return $t = 2$ (as the value of *power*(3,2,7))
we find $t = (power(3,2,7))^2 \textbf{ mod } 7 = 2^2 \textbf{ mod } 7 = 4$
$e = 5$ is not even: return ta **mod** $7 = 4*3$ **mod** $7 = 5$ (as the value of *power*(3,5,7))

More recursive algorithms

pages 353-362 /311-321

recursion often great for lazy programmer (with proper *bottom of recursion*):

$\Rightarrow$ useful to quickly get working prototypes

- gcd(a,b): (b == 0) ? a : gcd(b,a **mod** b)

(refer to slide 35 of March 27 lecture for division-free method)

pages 355/313

- exponentiation: a^e **mod** m ($e \geq 0$, $m > 1$)

  power(a,e,m):

  if e equals 0: return(1)

  else $t = (power(a,[e/2],m))^2 \textbf{ mod } m$

  if (e is even): return(t)

  else return (ta **mod** m)

same computation as before

- mostly less efficient than iterative version

Exponentiation: a^e **mod** m ($e \geq 0$, $m > 1$)

power(a,e,m):
if e equals 0: return(1)
else $t = (power(a,[e/2],m))^2 \textbf{ mod } m$
    if (e is even): return(t)
    else return (ta **mod** m)

to calculate 3^5 **mod** 7 (using different colors for variables at different recursion levels)

call *power*(3,5,7): $a = 3$, $e = 5$, $m = 7$
$e = 5 \neq 0$, thus $t = (power(3,[5/2]=2,7))^2 \textbf{ mod } 7$
  recursive call *power*(3,2,7): $a = 3$, $e = 2$, $m = 7$
  $e = 2 \neq 0$, thus $t = (power(3,[2/2]=1,7))^2 \textbf{ mod } 7$
    recursive call *power*(3,1,7): $a = 3$, $e = 1$, $m = 7$
    $e = 1 \neq 0$, thus $t = (power(3,[1/2]=0,7))^2 \textbf{ mod } 7$
      recursive call *power*(3,0,7): $a = 3$, $e = 0$, $m = 7$
      $e = 0$: return 1 (as the value of *power*(3,0,7))
    we find $t = (power(3,0,7))^2 \textbf{ mod } 7 = (1)^2 \textbf{ mod } 7 = 1$
    $e = 1$ is not even: return ta **mod** $7 = 1*3$ **mod** $7 = 3$ (as the value of *power*(3,1,7))
  we find $t = (power(3,1,7))^2 \textbf{ mod } 7 = 3^2 \textbf{ mod } 7 = 2$
  $e = 2$ is even: return $t = 2$ (as the value of *power*(3,2,7))
we find $t = (power(3,2,7))^2 \textbf{ mod } 7 = 2^2 \textbf{ mod } 7 = 4$
$e = 5$ is not even: return ta **mod** $7 = 4*3$ **mod** $7 = 5$ (as the value of *power*(3,5,7))

Another quick and dirty recursion example

print all permutations of $1, 2, 3, \ldots, n$

initialize $a_i = i, 1 \leq i \leq n$

permute(b): /* $a_1, a_2, \ldots, a_b$ still need to be permuted */

if ($b \leq 1$)

print $a_1, a_2, \ldots, a_n$

else

for $i = 1$ to b

swap a_i and a_b

permute(b–1)

swap a_i and a_b

permute(n)

Recursive sorting

generic recursive sorting of list L of n items:

if $n \le 1$: list sorted already, done
else

1. create smaller subproblems:
   form disjoint sublists L_1, L_2 of L
2. recurse:
   sort L_1 and L_2
3. combine:
   sort L using sorted L_1 and L_2

pages 359-362 /317-319

two common realizations of this idea:

- merge sort

page 364/322 exercises 34-39/50-55

- quick sort

Merge sort

pages 359-362 /317-319

sort list L of n items: cost $M(n)$:

if $n \leq 1$: list sorted already, done 0

else

1. create smaller subproblems by splitting L in the middle: 0
   L_1 first half, L_2 second half of L
2. recurse: sort L_1 and sort L_2 $2M(n/2)$
3. combine: merge sorted lists L_1 and L_2 into single sorted list L n
   (as seen in homework 5.3c)

total: $n + 2M(n/2)$

Solving $M(n) = n + 2M(n/2)$

$$
\begin{aligned}
M(n) &= n + 2M(n/2) \\
&= n + 2(n/2 + 2M(n/4)) \\
&= 2n + 4M(n/4) \\
&= 2n + 4(n/4 + 2M(n/8)) \\
&= 3n + 8M(n/8) \\
&\dots \\
&= kn + 2^k M(n/2^k) \\
&\text{(when } k \text{ reaches } \log_2(n)\text{: } M(n/2^k) = 0 \text{)} \\
&= n\log_2(n)
\end{aligned}
$$

Proof that $M(n) = n\log_2(n)$
using strong induction:

1. $M(1) = 0$ follows from the algorithm
   $1*\log_2(1) = 0$
   $\Rightarrow M(n) = n\log_2(n)$ holds for $n = 1$
2. induction hypothesis: $M(k) = k\log_2(k)$ if $k<n$
   $$\begin{aligned} M(n) &= n + 2M(n/2) \text{ (from algorithm)} \\ &= n + 2((n/2)\log_2(n/2)) \text{ (Induction hypothesis)} \\ &= n + n\log_2(n/2) \\ &= n + n(\log_2(n) - \log_2(2)) \\ &= n + n(\log_2(n) - 1) \\ &= n\log_2(n) \end{aligned}$$
   (note the cheating: this proof requires n to be a power of 2; for general n the result $M(n) = O(n\log(n))$ is valid though)

Quick sort

page 364/322 exercises 34-39/50-55

sort list L of n items $l_0, l_1, \ldots, l_{n-1}$: cost $Q(n)$:

if $n \le 1$: list sorted already, done 0

else

1. create smaller subproblems:

   $L_1 = \{l_i\text{: } 0<i<n,\ l_i \le l_0\}$, $r = |L_1|$ $n-1$

   $L_2 = \{l_i\text{: } 0<i<n,\ l_i > l_0\}$

2. recurse: sort L_1 and sort L_2 $Q(r)+Q(n-1-r)$

3. combine: concatenate sorted lists L_1, l_0 and L_2 into single sorted list L:

   $L = L_1, l_0, L_2$ 0

total: $n-1 + Q(r)+Q(n-1-r)$

Solving $Q(n) = n-1 + Q(r) + Q(n-1-r)$

depends on r (and cardinalities in recursion)

- worst case: $r = 0$ or $r = n-1$:

$$
\begin{aligned}
Q(n) &= n-1 + Q(n-1) \\
&= n-1+n-2+Q(n-2) \quad \text{(if bad luck again)} \\
&= n-1+n-2+n-3+Q(n-3) \quad \text{(same)} \\
&= \ldots = n(n-1)/2
\end{aligned}
$$

(as bad as bubble or insertion sort)

- optimal case: always $r \approx n/2 \approx n-r$:

$$
\begin{aligned}
Q(n) &\approx n + 2Q(n/2) \\
&= n\log_2(n) \text{ (same as } M(n))
\end{aligned}
$$

A final recursion example: heapsort

given a_i, $0 \le i < n$

- $i \le [(n-1)/2]$: a_i's children are a_{2i+1} and a_{2i+2}
- to make a_i largest among ordered offspring:

  insert(i,n): if a_i has larger child, then

  - swap a_i with largest child a_k
  - *insert*(k,n)
- sort a_i in increasing order:

$O(n)$ for $i = [(n-1)/2]$ downto 0 do *insert*(i,n)

  for $i = n-1$ downto 1 do

  - swap a_0 and a_i
  - *insert*$(0,i)$ $O(\log(i))$ per insertion

overall $O(n\log(n))$