アルゴリズムとデータ構造入門

1.手続きによる抽象の構築

1.2 プログラムの構築

奥 乃 博
大学院情報学研究科知能情報学専攻
知能メディア講座 音声メディア分野
http://winnie.kuis.kyoto-u.ac.jp/~okuno/Lecture/07/IntroAlgDs/
okuno@i.kyoto-u.ac.jp

10月30日(火)中間試験
(範囲:第1回～第4回)
過去問は上記URLにあります。

10月23日・本日のメニュー

- **10月30日(火)中間試験(範囲:第1回～第4回)**
- 1.2.3 Orders of Growth
- **1.2.4 Exponentiation**
- 1.2.5 Greatest Common Divisors
- 1.2.6 Example: Testing for Primality
- 1.3.1 Procedures as Arguments
- 1.3.2 Constructing Procedures Using `lambda'

32

1.2.4 Exponentiation (冪乗)

```
(define (expt b n)
  (if (= n 0)
      1
      (* b (expt b (- n 1))) ))
```

$b^n = b * \cdots * b$

- Linear recursive process $\Theta(n)$ steps, $\Theta(n)$ space

```
(define (expt b n)
  (define (iter counter product)
    (if (= counter 0)
        product
        (iter (- counter 1) (* b product))
    ))
  (iter n 1) )
```

p = b * p
c = c - 1

- Linear iterative process $\Theta(n)$ steps, $\Theta(1)$ space

33

Exponentiation

```
(define (fast-expt b n)
  (cond ((= n 0) 1)
        ((even? n)
         (square (fast-expt b (/ n 2))))
        (else (* b (fast-expt b
                              (- n 1) )))

(define (even? n)
  (= (remainder n 2) 0) )
```

- recursive process $\Theta(\log n)$ steps, $\Theta(\log n)$ space

34

Exponentiation(べき乗)

- X^{16}
- $16 \equiv 10000_2$ より2進数を4回左シフト

1. まず、1を“SX”、0を“S”で置換し、
2. 次に、先頭の“SX”を除く。
3. 得られたSとXを「Square」「xをかける」と読む。

- 例： X^{23}
- $23 \equiv 10111_2$

1. SX S SX SX SX
2. SSXSXSX
3. X^2 X^4 X^5 X^{10} X^{11} X^{22} X^{23}

35

“Power Tree”

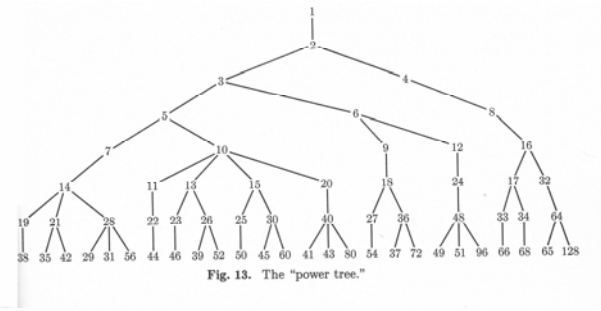

Fig. 13. The “power tree.”

36

10月23日・本日のメニュー

- **10月30日（火）中間試験（範囲：第1回～第4回）**
- 1.2.3 Orders of Growth
- 1.2.4 Exponentiation
- **1.2.5 Greatest Common Divisors**
- 1.2.6 Example: Testing for Primality
- 1.3.1 Procedures as Arguments
- 1.3.2 Constructing Procedures Using `Lambda'

39

Greatest Common Divisors（最大公約数）

- *a* mod *b* ＝ *r*（modulo 剰余）とすると
- GCD(*a*, *b*) = GCD(*b*, *r*) が成立。
- **ユークリッドの互除法**

```
(define (gcd a b)
  (if (= b 0)
      a
      (gcd b (remainder a b)) ))
```

40

Modularity Calculus（合同式）

- ***a*** **≡** ***b*** **mod** ***n***（**congruent modulo** ***n***）
  「***a*** **mod** ***n*** **と** ***b*** **mod** ***n*** が等しい」
- (reminder of) ***x*** modulo ***n*** は剰余
- ***a+b*** **mod** ***n***
  ≡ ***(a*** **mod** ***n*** **+** ***b*** **mod** ***n)*** **mod** ***n***
- ***a**b** **mod** ***n***
  ≡ ***(a*** **mod** ***n*** ***** ***b*** **mod** ***n)*** **mod** ***n***
- ***a*** **mod** ***(m*n)*****は中国人剰余定理で求める**
- $a^{p-1} \equiv 1 \bmod p$ if ***p*** が素数（prime）

41

Chinese Remainder Theorem

連立1次合同式

$x \equiv b_1 \pmod{d_1}$

$x \equiv b_2 \pmod{d_2}$

…

$x \equiv b_t \pmod{d_t}$

の場合、$d_1, d_2, \ldots d_t$ が互いに素であれば、

$n = d_1 d_2 \ldots d_t$

を法として、ただ一つの解がある。

まず、$n / d_i = n_i$ とおけば、d_i と n_i は互いに素であるから、

$n_i x_i \equiv 1 \pmod{d_i}$

の解 x_i を求めることができる。ここで、

$x \equiv b_1 n_1 x_1 + b_2 n_2 x_2 + \ldots + b_t n_t x_t \pmod{n}$

とすれば、この x は明らかにもとの合同式をすべて満足する。

42

Chinese Remainder Theoremの例

x mod *105* は？

- 3 * 5 * 7 = 105
- $x \equiv 1 \pmod 3$
- $x \equiv 2 \pmod 5$
- $x \equiv 3 \pmod 7$
- $35*2 \equiv 1 \pmod 3$
- $21*1 \equiv 1 \pmod 5$
- $15*1 \equiv 1 \pmod 7$ より、
- x mod 105

  $\equiv 1*35*2 + 2*21*1 + 3*15*1 \bmod 105$

  $= 157 \bmod 105 \equiv 52 \bmod 105$

43

Chinese Remainder Theoremの例

2^{90} mod *91* は？

- 91 = 7 * 13
- $2^3 \equiv 1 \pmod 7$ より、$2^{90} \equiv 1 \pmod 7$
- $2^6 \equiv -1 \pmod{13}$ より、

  $2^{84} \equiv 1 \pmod{13} \Rightarrow 2^{90} \equiv -1 \pmod{13}$

- $13*6 \equiv 1 \pmod 7$
- $7*2 \equiv 1 \pmod{13}$ より、

- 2^{90} mod *91* $\equiv 1*13*6 - 1*7*2 = 64$

44

Lameの定理

- GCD(*a*, *b*)（ただし、*b* < *a*）の計算に*k* step 必要なら、*b* ≥ Fib(*k*)
- 例えば、GCD(*m*, *n*)（ただし、*n* < *m*）が *k* step かかるとすると、*n* ≥ Fib(*k*) ≒ Φ^k / √5

$$\phi = \frac{1}{2}\left(1+\sqrt{5}\right) \qquad Fib(n) \cong \frac{\phi^n}{\sqrt{5}}$$

- つまり、ステップ数は、*n* の対数的に増加。
- Θ(log n) steps

45

10月23日・本日のメニュー

- **10月30日（火）中間試験（範囲：第1回～第4回）**
- 1.2.3 Orders of Growth
- 1.2.4 Exponentiation
- 1.2.5 Greatest Common Divisors
- **1.2.6 Example: Testing for Primality**
- 1.3.1 Procedures as Arguments
- 1.3.2 Constructing Procedures Using `lambda'

46

Testing for Primality

```
(define (smallest-divisor n)
  (find-divisor n 2))
(define (find-devisor n test-divisor)
  (cond ((> (square test-divisor) n) n)
        ((divides? test-divisor n) test-divisor)
        (else (find-divisor n
                            (+ test-divisor 1) )) ))
(define (divides? a b)
  (= (remainder b a) 0) )
(define (prime? n)
  (= n (smallest-divisor n)) )
```

47

Improvement by HGO

```
(define (smallest-divisor n)
  (find-divisor n 3))
(define (find-devisor n test-divisor)
  (cond ((> (square test-divisor) n) n)
        ((divides? test-divisor n) test-divisor)
        (else (find-divisor n
                  (+ test-divisor 2)) )) )
(define (divides? a b)
  (= (remainder b a) 0) )
(define (prime? n)
  (if (even? n)
      2
      (= n (smallest-divisor n)) ))
```

The Fermat Test

- $a^{p-1} \equiv 1 \bmod p$ if p が素数(prime)

```
(define (expmod base exp m)
  (cond ((= exp 0) 1)
        ((even? exp)
         (remainder (square (expmod base (/ exp 2) m)) m) )
        (else
         (remainder (* base (expmod base (- exp 1) m)) m) )
  ))
```

```
(define (fermat-test n)
  (define (try-it a)
    (= (expmod a n n) a) )
  (try-it (+ 1 (random (- n 1)))) )
```

```
(define (fast-prime? n times)
  (cond ((= times 0) true)
        ((fermat-test n) (fast-prime? n (- times 1)))
        (else false) ))
```

Probabilistic Algorithms(確率的アルゴリズム)

- Carmichael numbers: 561, 1105, 1072, 2465,
  $a^{560} = a^2\ a^{10}\ a^{16}$
  $a^2 \equiv 1 \bmod 3$, $a^{10} \equiv 1 \bmod 11$, $a^{16} \equiv 1 \bmod 17$
  $\Rightarrow$ $a^{560} \equiv 1 \bmod 561 = 3 * 11 * 17$
- Fermat's testは、エラーの機会を任意に小さくできる。
  → *probabilistic algorithm*
  必要条件のみ満足
- *Algorithm*: Wilson's test
  p is a prime precisely
  when $(p-1)! \equiv -1 \bmod p$
  必要十分条件

50

10月23日・本日のメニュー

- **10月30日（火）中間試験（範囲：第1回～第4回）**
- 1.2.3 Orders of Growth
- 1.2.4 Exponentiation
- 1.2.5 Greatest Common Divisors
- 1.2.6 Example: Testing for Primality
- **1.3.1 Procedures as Arguments**
- 1.3.2 Constructing Procedures Using `lambda'

52

1.3.1 Procedures as Arguments

```
(define (sum-integers a b)
  (if (> a b)
      0
      (+ a (sum-integers (+ a 1) b)) ))
(define (sum-cubes a b)
  (if (> a b)
      0
      (+ (cube a) (sum-cubes (+ a 1) b)) ))
(define (cube x) (* x x x))
(define (pi-sum a b)
  (if (> a b)
      0
      (+ (/ 1.0 (* a (+ a 2))) (pi-sum (+ a 4) b)) ))
```

$$\sum_{i=a}^{b} i$$

$$\sum_{i=a}^{b} i^3$$

```
(define (<name> a b)
  (if (> a b)
      0
      (+ (<term> a)
         (<name> (<next> a) b)) ))
```

$$\sum_{i=a,i+4}^{b} \frac{1}{i(i+2)}$$

53

1.3.1 sumを抽象化

```
(define (<name> a b)
  (if (> a b)
      0
      (+ (<term> a)
         (<name> (<next> a) b)) ))
(define (sum term a next b)
  (if (> a b)
      0
      (+ (term a)
         (sum term (next a)next b)) ))
(define (inc n) (+ n 1))
(define (sum-cubes a b)
  (sum cube a inc b) )
(define (identity x) x)
(define (sum-integers a b)
  (sum identity a inc b) )
```

$$\sum_{i=a,next(i)}^{b} f(i)$$

$$\sum_{i=a,i+1}^{b} cube(i)$$

$$\sum_{i=a,i+1}^{b} i$$

54

Pi-Sum (Pi/8) の計算方法

$$\sum_{i=a,\pi next(i)}^{b} \pi term(i)$$

```
(define (pi-sum a b)
  (define (pi-term x)
    (/ 1.0 (* x (+ x 2))) )
  (define (pi-next x)(+ x 4) )
  (sum pi-term a pi-next b) )
```

55

積分(integral) の計算方法

$$\int_a^b f = \left[f\left(a+\frac{dx}{2}\right) + f\left(a+dx+\frac{dx}{2}\right) + f\left(a+2dx+\frac{dx}{2}\right) + \cdots \right] dx$$

$$\left(\sum_{i=a,i+\Delta x}^{b} f(i) \right) \Delta x$$

```
(define (integral f a b dx)
  (define (add-dx x) (+ x dx))
  (* (sum f (+ a (/ dx 2.0)) add-dx b)
     dx ))
```

56

Ex.1.31 Productを抽象化

$$\prod_{i=a,next(i)}^{b} f(i)$$

```
(define (product term a next b)
  (if (> a b)
      1
      (* (term a)
         (product term (next a)next b))
  ))
```

$$\prod_{i=a,i+1}^{b} i^3$$

```
(define (product-cubes a b)
  (product cube a inc b) )
```

```
(define (product-integers a b)
  (product identity a inc b) )
```

$$\prod_{i=a,i+1}^{b} i$$

57

Ex.1.32 Accumulation（さらなる抽象化）

```
(define (sum term a next b)
  (if (> a b)
      0
      (+ (term a)
         (sum term (next a) next b)) ))
(define (product term a next b)
  (if (> a b)
      1
      (* (term a)
         (product term (next a) next b)) ))

(define (<combiner> <name> <term> a <next> b)
  (if (> a b)
      <null-value>
      (<combiner> (<term> a)
         (<name> <term> (<next> a) <next> b))
 ))
```

$$\sum_{i=a,next(i)}^{b} f(i)$$

$$\prod_{i=a,next(i)}^{b} f(i)$$

Accumulateによる sum, product

```
(define (accumulate combiner null-value
                    term a next b)
  (if (> a b)
      null-value
      (combiner (term a)
         (accumulate combiner null-value
                    term (next a) next b ))))

(define (sum term a next b)
    (accumulate + 0 term a next b) )

(define (product term a next b)
    (accumulate * 1 term a next b) )
```

59

10月23日・本日のメニュー

- **10月30日（火）中間試験（範囲：第1回～第4回）**
- 1.2.3 Orders of Growth
- 1.2.4 Exponentiation
- 1.2.5 Greatest Common Divisors
- 1.2.6 Example: Testing for Primality
- 1.3.1 Procedures as Arguments
- **1.3.2 Constructing Procedures Using `lambda'**

61

lambda: Anonymous procedure

```
(define (fact n)
  (if (= n 0)
      1
      (* n (fact (- n 1))) ))
```

は次の式と等価

```
(define fact
  (lambda (n)
    (if (= n 0)
        1
        (* n (fact (- n 1))) )))
```

62

Lambda as anonymous procedure

```
(lambda (x) (+ x 4))
((lambda (x) (+ x 4)) 5)

(define (pi-sum a b)
  (define (pi-term x)
    (/ 1.0 (* x (+ x 2))) )
  (define (pi-next x)(+ x 4) )
  (sum pi-term a pi-next b) )

(define (pi-sum a b)
  (sum (lambda (x) (/ 1.0 (* x (+ x 2))
    a
    (lambda (x) (+ x 4))
    b ))
```

Using let to create local variables

$$f(x,y) = x(1+xy)^2 + y(1-y) + (1+xy)(1-y)$$

$$a = 1+xy$$
$$b = 1-y$$
$$f(x,y) = xa^2 + yb + ab$$

```
(define (f x y)
  (define (f-helper a b)
    (+ (* x (square a))
       (* y b)
       (* a b) ))
  (f-helper
    (+ 1 (* x y))
    (- 1 y) ))
```

64

1.3.2 Local Variables with `let`

```
(define (f x y)
  (define (f-helper a b)
    (+ (* x (square a))
       (* y b)
       (* a b) ))
  (f-helper
    (+ 1 (* x y))
    (- 1 y) ))
```

```
(define (f x y)
  (let ((a (+ 1 (* x y)))
        (b (- 1 y)) )
    (+ (* x (square a))
       (* y b)
       (* a b) )))
```

```
(define (f x y)
  ((lambda (a b)
     (+ (* x (square a))
        (* y b)
        (* a b) ))
   (+ 1 (* x y))
   (- 1 y) ))
```

```
(let ((<v1> <e1>)
      (<v2> <e2>)
      …
      (<vn> <en>) )
  <body> )
```

シンタックス・シュガー

65

宿題：なし

10月30日（火）中間試験
（範囲：第1回～第4回）

- 過去問はWebにあり。

66

scope of variables

```
(let ((x 7))
  (+ (let ((x 3))
       (+ x (* x 10)) )
     x) )
```

```
(let ((x 5))
  (let ((x 3)
        (y (+ x 2)) )
    (* x y) ))
```

Substition model

67

scope of variables

```
(let ((x 7))
  (+ (let ((x 3))
       (+ x (* x 10)) )
     x) )
```

```
x = 7
  x = 3
    -> 33
x = 7 -> 40
```

```
(let ((x 5))
  (let ((x 3)
        (y (+ x 2)) )
    (* x y) ))
```

```
x = 5
  x = 3
  y = 7
  -> 21
```

68

scope of variables

```
(let ((x 7))
  (+ (let ((x 3))
       (+ x (* x 10)) )
     x)
```

```
x = 7
  x = 3
    -> 33
x = 7 -> 40
```

```
(let ((x 5)
  (let ((x 3)
        (y (+ x 2)) )
    (* x y) ))
```

```
x = 5
  x = 3
  y = 7
  -> 21
```

69