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
1.2.2 ~1.3

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/09/IntroAlgDs/
okuno@i.kyoto-u.ac.jp

12月8日3階大会議室で中間試験
TAの居室は10号館4階奥乃1研，2研
http://winnie.kuis.kyoto-u.ac.jp/~fukubaya/AlgDsWiki/

11月10日・本日のメニュー

- 1.2.5 Great Common Divisors
- 1.2.6 Examples: Testing for Primality
- 1.3 Formulating Abstractions with Higher-Order Procedures
- 1.3.1 Procedures as Arguments
- 1.3.2 Constructing Procedures Using Lambda
- 1.3.3 Procedures as General Methods

Ex.1.10改 Ackermann Function

$$Ack(m,n)=\begin{cases} n+1, & \text{if } m=0 \\ Ack(m-1,1), & \text{if } n=0 \\ Ack(m-1,Ack(m,n-1)), & \text{otherwise} \end{cases}$$

Ackermann関数は線形再帰ではない

```
(define (ackermann m n)
  (if (= m 0)
      (+ n 1)
      (if (= n 0)
          (ackermann (- m 1) 1)
          (ackermann
             (- m 1)
             (ackermann m (- n 1)) ))))
```

宿題1:アッカーマン関数の値

手計算で求める. 計算過程を書くこと.

(trace ackermann)で得られる結果も提出.

1. (ackermann 0 2)
2. (ackermann 1 2)
3. (ackermann 2 2)
4. (ackermann 3 2)

以下随意課題

5. (ackermann 0 n) ≡ n+1 (理由も)
6. (ackermann 1 n) ≡ ? (理由も)
7. (ackermann 2 n) ≡ ? (理由も)
8. (ackermann 3 n) ≡ ? (理由も)

4

宿題1:アッカーマン関数の値

```
(define (ackermann m n)
     (if (= m 0)
         (+ n 1)
         (if (= n 0)
             (ackermann (- m 1) 1)
             (ackermann (- m 1)
                        (ackermann m (- n 1))))))
```

1. (ackermann 0 n) ≡
2. (ackermann 1 n) ≡
3. (ackermann 2 n) ≡
4. (ackermann 3 n) ≡
5. (ackermann 4 n) ≡

5

Order of Growth: まとめ

手続き	ステップ数	スペース
factorial	$\Theta(n)$	$\Theta(n)$
fact-iter	$\Theta(n)$	$\Theta(1)$
テーブル参照型fact	$\Theta(1)$	$\Theta(n)$
fib	$\Theta(\phi^n)$	$\Theta(n)$
fib-iter	$\Theta(n)$	$\Theta(1)$
テーブル参照型fib	$\Theta(1)$	$\Theta(n)$

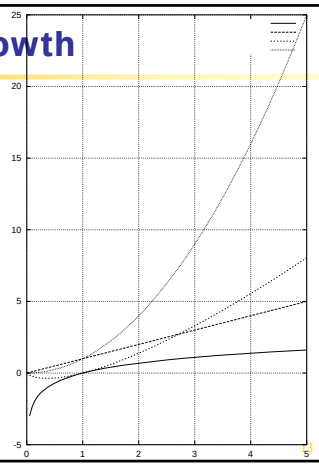
ステップ数(時間計算量)とスペース(空間計算量)が交換可能な場合がある!

12

Order of Growth

次の式はどの曲線・直線に対応するか

- $y = x$
- $y = x^2$
- $y = \log x$
- $y = x \log x$



11月10日・本日のメニュー

- 1.2.5 Great Common Divisors
- 1.2.6 Examples: Testing for Primality
- 1.3 Formulating Abstractions with Higher-Order Procedures
- 1.3.1 Procedures as Arguments
- 1.3.2 Constructing Procedures Using Lambda

14

小学校の復習

1. 143と546の最大公約数を求めよ.

15

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)) ))
```

16

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*) ≒ $\Phi^k / \sqrt{5}$

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

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

17

小学校の復習

1. 143と546の最大公約数を求めよ.
2. 3で割った余りが2 $(x \bmod 3 \equiv 2)$,
   5で割った余りが3 $(x \bmod 5 \equiv 3)$,
   7で割った余りが1 $(x \bmod 7 \equiv 1)$,
   となるような最小正整数 x を求めよ.
3. 3で割った余りが1 $(x \bmod 3 \equiv 1)$,
   5で割った余りが2 $(x \bmod 5 \equiv 2)$,
   7で割った余りが3 $(x \bmod 7 \equiv 3)$,
   となるような最小正整数 x を求めよ.

18

Modularity Calculus（合同式）

- $a \equiv b \bmod n$（congruent modulo n）
  「a mod n と b mod n が等しい」
- (reminder of) x modulo n は剰余
- $a+b \bmod n$
  $\equiv (a \bmod n + b \bmod n) \bmod n$
- $a*b \bmod n$
  $\equiv (a \bmod n * b \bmod n) \bmod n$
- $a \bmod (m*n)$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）

19

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。

20

Chinese Remainder Theoremの例

$x \bmod 105$ 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、
- $x \bmod 105$
  $\equiv 1*35*2 + 2*21*1 + 3*15*1 \bmod 105$
  $= 157 \bmod 105 \equiv 52 \bmod 105$

21

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

22

test: プログラムを作成せよ

1. GCDの手続きを書け
2. （GCD 120 36） でプロセス実行を書け.
3. (GCD m n) の時間計算量は？
4. GCDの実行プロセスは？

23

11月10日・本日のメニュー

- 1.2.5 Great Common Divisors
- 1.2.6 Examples: Testing for Primality
- 1.3 Formulating Abstractions with Higher-Order Procedures
- 1.3.1 Procedures as Arguments
- 1.3.2 Constructing Procedures Using Lambda

24

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)) )
```

25

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)) ))
```

26

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

Fermat's test は十分条件ではない

28

Fermat's Last Theorem

$$x^n + y^n = z^n$$

I have discovered a truly remarkable proof which this margin is too small to contain.

n>2 で *x*, *y*, *z* を満たす非負整

Euler's Conjecture

$$a^4 + b^4 + c^4 \neq d^4$$

が成立するだろう。

$$95800^4 + 217519^4 + 414560^4 = 422481^4$$

1987年発見

29

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)) ))
```

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

31

1.3.1 sumを抽象化

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

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

```
(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) )
```

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

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

32

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) )
```

33

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 ))
```

34

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) )
```

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

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

35

Ex.1.32 Accumulation（さらなる抽象化）

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

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

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

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

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

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) )
```

37

11月10日・本日のメニュー

- 1.2.5 Great Common Divisors
- 1.2.6 Examples: Testing for Primality
- 1.3 Formulating Abstractions with Higher-Order Procedures
- 1.3.1 Procedures as Arguments
- *Intermission*
- 1.3.2 Constructing Procedures Using Lambda
- 1.3.3 Procedures as General Methods

39

lambda: 無名(匿名)手続き

```
(define (plus4 x) (+ x 4))
```

は次式と等価

```
(define plus4 (lambda (x) (+ x 4)))
```

ラムダ式の読み方

本体 (body)

```
(lambda                    (x)      (+   x     4))
   |                        |        |   |     |
the procedure of an argument x that adds x and 4
```

仮引数 (formal parameters)

ラムダ式の適用

```
((lambda (x y z) (+ x y (square z))) 1 2 3)
```

x = 1, y = 2, z = 3 を代入(置換)

```
12
```

48

Lambda as anonymous procedure

```
(lambda (x) (+ x 4))          無名(匿名)手続き
((lambda (x) (+ x 4)) 5)      手続き適用
```

```
(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) )
```

局所的な無駄な名前 Pi-term, Pi-next をなくす

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

49

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))) )))
```

50

Using let to create local variables

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

補助変数 a, b を使いたい

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

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

51

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))) )))
```

52

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 ))
```

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 ((<v_1> <e_1>)
(<v_2> <e_2>)
…
(<v_n> <e_n>))
<*body*>)

シンタックス・シュガー

54

scope of variables

lambda式への procedure application を Substitution model で考える

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

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

55

宿題:11月17日正午締切

宿題は、次の4題:

Ex.1.29～1.32.

（Ex.1.30~1.32 は復習）

随意課題: **Ex.1.33**

DON'T PANIC!