2.式計算のアルゴリズム

1

式計算の問題

- 最大公約数問題
- フィボナッチ数列計算問題
- べき乗計算の問題
- 多項式評価の問題

2

最大公約数問題

- 素朴なアルゴリズム(多項式時間のアルゴリズム)
- ユークリッドの互除法(対数時間アルゴリズム)

3

最大公約数問題

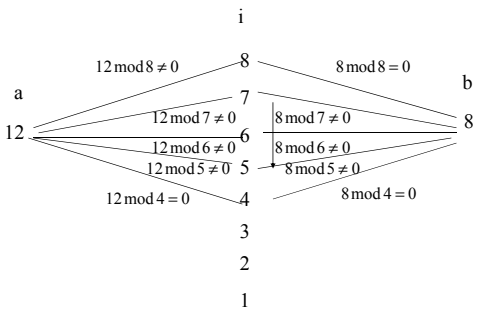
入力:2つの整数a,b
(ここで、入力サイズは、
$\max\{a,b\}$
とします。)
出力:a,bの最大公約数
$\gcd(a,b)$

4

素朴な最大公約数発見法

- 注目点
  - すべて整数
- アィディア
  - $1 \sim \min\{a,b\}$ の整数をすべて調べる。
  - $\min\{a,b\}$ から初めてカウンタを減らしながら繰り返す。

5



6

アルゴリズムnaïve_gcd(a,b)
入力:a,b
出力:gcd(a,b)

```
1. for(i=min{a,b}  ;i>0;i--){
2.   if(iがaとbの約数){
3.      return(i);
4.   }
5. }
```

7

- アルゴリズムnaïve_gcdの正当性は明らかなので省略する。(帰納法で証明もできる。)
- アルゴリズムnaïve_gcdの時間計算量
  高々 $\min\{a,b\}$ の繰り返し回数

$O(\min\{a,b\})$ の時間計算量

$n \equiv a \approx b$ とすると、

$O(n)$ 時間

よって、naïve_gcdは線形時間アルゴリズムである。

8

練習

$a = 9, b = 6$

に対して、素朴なアルゴリズムnaïve_gcd(a,b)を動作させ、最大公約数をもとめよ。

9

ユークリッドの互除法

- 注意点
  - 全ての整数を調べる必要はない。
  - 整数における性質を利用(整数論)
- アイディア
  - 余りに注意して、互いに除算を繰り返す。
  - 前回の小さい方の数と余りを、新たな2数に置き換えて繰り返す。
  - 割り切れた時の小さい数が、最大公約数。
    →繰り返し回数を大幅に削減できる。

10

アルゴリズムeuclid_gcd(a,b)
入力:a,b(a>bとする。)
出力:gcd(a,b)

```
1. big=a;
2. small=b;
3. r=big % small;
4. While(割り切れない(r≠0) ){
5.      big=small;
6.      small=r;
7.      r=big % small;
8. }
9. return small;
```

11

ユークリッドの互除法の動作

$a = 36, b = 21$ の最大公約数を求める。

$$36 = 21\times 1 + 15$$
$$21 = 15\times 1 + 6$$
$$15 = 6\times 2 + 3$$
$$6 = 3\times 2 + 0$$

繰り返し回数

割り切れたときの除数が最大公約数

余りの系列が重要

12

練習

1.

$a=126, b=70$

に対して、ユークリッドの互除法euclid_gcd(a,b)を動作させ、最大公約数を求めよ。

2.

$a=126, b=70$

に対して、素朴なアルゴリズムnaïve_gcd(a,b)を動作させ、最大公約数を求めよ。

13

ユークリッド互除法の正当性

整数論の初歩を用いる。ここでは、必要なものの証明を与える。

命題E1（割り算における関係式）

2つの自然数 $a,b\in\mathbb{N},(a\geq b\geq 1)$ に対して、
2つの非負整数 $q,r\in\mathbb{Z}^+$ を用いて、
$a=q\times b+r,\quad (0\leq r<b)$
と表せる。

$\mathbb{Z}^+=\{0,1,2,\cdots\}$

q を商(quotient)
r を余り(remainder)
だと考える。

14

証明 b を固定し、 a に関する帰納法で証明する。

基礎:
$a=1$ のとき。

さらに、2つの場合に分ける。
場合1: $a=1,b=1$ のとき。

$a=$
$1=1$
$=1\times b+0$

このとき、
$q=1,r=0$

場合2: $a=1,b>1$ のとき

$a=$
$1=1$
$=0\times b+1$

このとき、
$q=1,r=0$

15

帰納:
$a=n$のとき命題が成り立つと仮定する。(帰納法の仮定)
帰納法の仮定より、
$(a=)$
$n=q\times b+r,\quad 0\leq r<b$ ———— ①
が成り立つ。
r,b は自然数なので、
$0\leq r<b$
は、
$1\leq r+1\leq b$ ———— ②
と等価。

ここで、$a=n+1$ のときを考える。

① $n=q\times b+r$の両辺に1を加える。
$n+1=q\times b+r+1$

16

②の右の不等号において等号の有無で2つの場合に分ける。

場合1: $r+1=b$ のとき。

$n+1=q\times b+r+1$
$\therefore n+1=q\times b+b$
$\therefore n+1=(q+1)\times b+0$
$\therefore a=(q+1)\times b+0$

商 $q+1\in\mathbb{Z}^+$ 、
余り $r=0\in\mathbb{Z}^+$

$q'=q+1, r'=r$
$a=q'\times b+r',\quad 0\leq r'<b$

場合2: $r+1<b$ のとき。

$n+1=q\times b+r+1$
$\therefore n+1=q\times b+(r+1)$
$\therefore a=q\times b+(r+1)$

商 $q\in\mathbb{Z}^+$ 、
余り $r+1\in\mathbb{Z}^+$

$q'=q, r'=r+1$
$a=q'\times b+r',\quad 0\leq r'<b$

QED

17

命題E2（割り算式の一意性）

2つの自然数 $a,b\in\mathbb{N},(a\geq b\geq 1)$ に対して、
2つの非負整数 $q,r\in\mathbb{Z}^+$ を用いて、
$a=q\times b+r,\quad (0\leq r<b)$
と表すとき、その表現法は一意である。

証明 背理法による。

2通りに表せると仮定する。(背理法の仮定)

背理法の仮定より、
$a=q\times b+r,\quad (0\leq r<b)$
$a=q'\times b+r',\quad (0\leq r'<b)$
と表現できる。ここで、$q\neq q'$ か $r\neq r'$ のいずれかは成り立つ。

18

両辺を減算する。

$$\begin{array}{lll} & a=qb+r & ,0\le r<b \\ -) & a=q'b+r' & ,0\le r'<b \\ \hline & 0=(q-q')b+(r-r') & ,-b<r-r'<b \end{array}$$

$0=(q-q')b+(r-r')$ より $(r-r')=(q'-q)b$

これは、$(r-r')$ が b の倍数であることを意味する。
一方、$-b<r-r'<b$ より、

$$-b<(q-q')b<b$$

$$\therefore -1<q-q'<1 \quad (\because b\neq 0)$$

$(q-q')$ は整数なので、

$$(q-q')=0$$

$$\therefore q=q'$$

これより、$r=r'$ が導けるが、背理法の仮定と矛盾する。
よって、命題が成り立つ。

QED

19

$a,b(a\ge b)$ の公約数を $\{\mathrm{cd}(a,b)\}$ と書く。

例えば、

$$\{\mathrm{cd}(32,20)\}=\{4,2,1\}$$

Common Divisor
(公約数)

命題E3 （約数集合の普遍性）

$a,b(a\ge b)$ に対して、先の命題E1，E2で定まる表現を

$a=q\times b+r \quad (0\le r<b)$

とする。このとき、

$$\{\mathrm{cd}(a,b)\}=\{\mathrm{cd}(b,r)\}$$

20

証明の前に、具体例で調べる。

$a=32,b=20$ とする。

$32=20\times 1+12$

より、$q=1,r=12$

公約数をみると、

$$\{\mathrm{cd}(32,20)\}=\{4,2,1\}$$

$$\{\mathrm{cd}(20,12)\}=\{4,2,1\}$$

また、

$$20=12\times 1+8$$

より、

$$\{\mathrm{cd}(12,8)\}=\{4,2,1\}$$

21

証明

任意の $e\in\{cd(a,b)\}$ に対して、$e\in\{cd(b,r)\}$ を示す。

$e\in\{cd(a,b)\}$ より、自然数 l,m を用いて、

$$\begin{cases} a=le \\ b=me \end{cases}$$

と書ける。$a=bq+r$ を用いると、次式が成り立つ。

$$bq+r=le$$

$b=me$ を代入してまとめる。

$$r=(l-mq)e$$

この式は、e が r の約数であることを示している。

QED

ユークリッドの互除法の停止性

命題E4 （ユークリッド互除法の停止性）

ユークリッドの互除法は停止する。

証明

ユークリッドの互除法によって次のような系列が得られたする。

$$a=bq_1+r_1 \quad (0<r_1<b)$$
$$b=r_1q_2+r_2 \quad (0<r_2<r_1)$$
$$r_1=r_2q_2+r_3 \quad (0<r_3<r_2)$$
$$\vdots$$

このとき、余りの系列は、単調減少する。すなわち、

$$b>r_1>r_2>\cdots$$

余りは、非負整数なので、ある繰り返し回数で必ず0になる。
したがって、停止する。

QED

23

ユークリッドの互除法の時間計算量

ユークリッド互除法の時間計算量を見積もるために、
次の命題が成り立つことに注意する。

命題E5 （余りの性質）

$a,b(a\ge b)$ に対して、先の命題E1，E2で定まる表現を

$a=q\times b+r \quad (0\le r<b)$

とする。このとき、

$$r<\frac{a}{2}$$

24

証明の前に、具体例で確認する。

$a=36, b=21$

$36 = 21\times1+15$ $< \frac{36}{2}$

$21 = 15\times1+6$ $< \frac{21}{2}$

$15 = 6\times2+3$ $< \frac{15}{2}$

$6 = 3\times2+0$

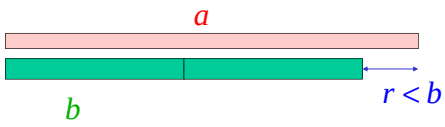
25

証明

2つの場合に分けて、$r<\frac{a}{2}$ を示す。

場合1: $b\le\frac{a}{2}$ のとき、

$r<b\le\frac{a}{2}$

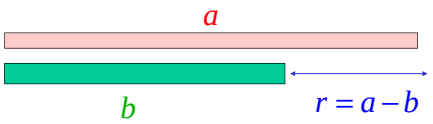


26

QED

場合2: $b>\frac{a}{2}$ のとき、

このとき、$a-b<\frac{a}{2}$ に注意する。



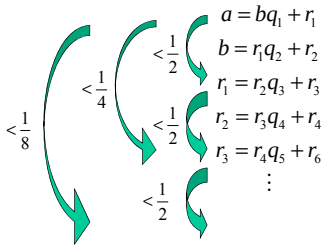
$a=bq+r$

において、商として $q=1$ にしかならない。
したがって、

$r=a-b<\frac{a}{2}$

27

命題E5より、ユークリッド互除法の計算量を求められる。



繰り返し回数は、高々 $O(\log_2 a)$ 回である。
したがって、ユークリッドの互除法は、
時間計算量 $O(\log_2 a)$ のアルゴリズムである。

28

最大公約数問題のまとめ

- 素朴な方法
  - $O(n)$ 時間のアルゴリズム
- ユークリッドの互除法
  - $O(\log n)$ 時間のアルゴリズム

n: 入力サイズ

29

補足:ユークリッドの互除法の再帰的な実現

ユークリッドの互除法は、再帰的にも実現できる。

$a=36, b=21$

$36 = 21\times1+15$ ← $\gcd(36,21)$

$21 = 15\times1+6$ ← $\gcd(21,15)$

$15 = 6\times2+3$

$6 = 3\times2+0$ ← 割り切れるときが基礎

サイズが小さい同じ問題を解くことに注意する。

30

アルゴリズムrecucive_gcd(a,b)
入力:a,b
出力:gcd(a,b)

```
1.int gcd(a,b){
2. r=a%b;
2. if(r==0){
3.    return b;
4.   }
5.  else{
6.     return gcd(b,r);
7.  }
8.}
```

基礎

帰納

31

フィボナッチ数列計算の問題

- 定義に基づく計算方法
- 再計算を防ぐ方法(動的計画法)
- 数学的考察による方法(参考)

32

フィボナッチ数列問題

入力:整数N
出力:フィボナッチ数列の第N項

$$\begin{cases} f(0) &= 0 & n = 0 \\ f(1) &= 1 & n = 1 \\ f(n) &= f(n-1) + f(n-2) & n \geq 2 \end{cases}$$

33

漸化式と再帰アルゴリズム

アルゴリズムfibo_rec
入力:N
出力:fibo(N)

```
1.int f(n){
2.  if(n==0){
3.    return 0;
4.  }else if(n==1){
5.    return 1;
6.  }else {
7.   return f(n-1)+f(n-1);
8.  }
9.}
```

$$\begin{cases} f(0) = 0 & n=0 \\ f(1) = 1 & n=1 \\ f(n) = f(n-1)+f(n-2) & n \geq 2 \end{cases}$$

34

- **再帰アルゴリズムの利点の一つに、漸化式で表された数列を直接プログラムにすることができることがある。**
- **ある問題を解くときに、同じ問題でよりサイズが小さい問題インスタンスの解が、元の問題の解法に大きな役割を果たせることが多い。**

再帰アルゴリズムが便利

ただし、再帰アルゴリズムは、性能に大幅な差異があるので、最悪時間計算量をきちんと見積もる必要がある。

35

- アルゴリズムfibo_recの正当性は明らかなので省略する。
- アルゴリズムfibo_recの時間計算量T(n)は漸化式で表される。この漸化式を解くことで、時間計算量が求まる。

36

アルゴリズムfibo_rec(N)　　の最悪時間量

求める時間量をT(N)とすると次式が成り立つ。

```
1.int f(n){
2.  if(n==0){
3.    return 0;
4.  }else if(n==1){
5.    return 1;
6.  }else {
7.   return f(n-1)+f(n-2);
8.  }
9.}
```

$T(0) = c_1$

$T(1) = c_2$

$T(n) = T(n-1) + T(n-2) + c_3$

37

$$\begin{cases} T(0) = c_0 & n=0 \\ T(1) = c_1 & n=1 \\ T(n) = T(n-1)+T(n-2)+c_2 & n \geq 2 \end{cases}$$

このように、再帰アルゴリズムの時間計算量は、始め漸化式で導かれることが多い。

漸化式を簡単に解くには、時間の関数は単調増加であることを利用する。

$T(n-1) \geq T(n-2)$

を利用すると、与式の帰納部分は、

$T(n) \leq 2T(n-1) + c_2$

とかける。

38

この漸化式を解く。

$$\begin{aligned} T(n) &\leq 2T(n-1) + c_2 \\ &\leq 2(2T(n-2) + c_2) + c_2 = 4T(n-2) + 3c_2 \\ &\leq 2(4T(n-3) + 3c_2) + c_2 = 8T(n-3) + 7c_2 \\ &\leq \cdots \\ &\leq 2^{n-1}T(1) + (2^{n-1})c_2 \\ &\leq c_1 2^{n-1} + c_2 2^{n-1} \\ &\leq c2^n \qquad (c \equiv \max\{c_1, c_2\}) \\ &\therefore T(n) = O(2^n) \end{aligned}$$

厳密に解くには、帰納法を用いるか、あるいは差分方程式の解を求める必要がある。

しかし、オーダー記法による漸近的評価では、不等式で計算していった方が、簡単に時間計算量が求まることが多い。

39

問題の考察による高速化

- フィボナッチ数列のような漸化式で表される数列は、N以下の全ての項が計算されていれば、定数時間で計算することができる。

$a_0, a_1, \cdots, a_{n-1} \longrightarrow a_n$

漸化式は、その項より前の項番号を持つ式を組み合わせて定義される。

小さい項番号から、全ての数列を保持していれば、高速化が図れる。

40

配列を用いたアルゴリズム

アルゴリズムfibo_array
入力：N
出力：fibo(N)

```
1.int f(n){
2.  A[0]=0;
3.  A[1]=1;
4.  For(i=2;i<=n;i++){
5.     A[i]=A[i-1]+A[i-2];
6.  }
7.   return A[n];
8.}
```

$f(0) = 0$

$f(1) = 1$

$f(n) = f(n-1) + f(n-2)$

41

- **アルゴリズムfibo_arrayの時間計算量**

forループをn-1回繰り返しているだけなので、O(n)時間

この問題の場合は、再帰を用いると性能が悪くなる。（問題によっては、高性能の再帰アルゴリズムもあるので、十分な考察が必要となる。）

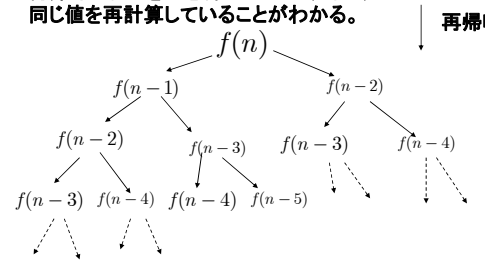
42

fibo_recが低速な理由

- 主に、不必要な再計算を行っていることが原因。

再帰呼び出しを注意深くトレースすると、
同じ値を再計算していることがわかる。

再帰呼び出し



$f(1)f(0)$ $f(1)f(0)$ $f(1)f(0)$ ……

43

更なる高速化(数学的考察)

○近似値でよければ、さらに高速化できる。

数学的には、フィボナッチ数列の一般項は次式である。

$$\phi_1 = \frac{1+\sqrt{5}}{2}, \phi_2 = \frac{1-\sqrt{5}}{2}$$

$$f(n) = \frac{1}{\sqrt{5}}{\phi_1}^{n+1} - \frac{1}{\sqrt{5}}{\phi_2}^{n+1}$$

$x^2 + x + 1 = 0$ の解。
黄金比

高速なべき乗アルゴリズムを用いれば、

$$O(\log_2 n)$$

の時間計算量で解くことができる。

44

べき乗計算の問題

- 素朴な方法
- 高速アルゴリズム(ロシアの農民算法)

45

3−1:べき乗問題

入力: x, n

(ここで、入力サイズは、n とします。)

出力: $f(x) = x^n$

46

素朴なべき乗の求め方

- アィディア
  - x を繰り返し、n 回乗算する。

$$x^n = \prod_{i=1}^{n} x = \underbrace{x \bullet x \bullet \cdots \bullet x}_{n}$$

47

アルゴリズムnaïve_pow(x,n)
入力:x,n
出力:xのn乗

初期化が重要

```
1. f=1.0;
2. for(i=0;i<N;i++){
3.   f=f*x;
4. }
5. return f;
```

48

- アルゴリズムnaïve_powの正当性。
（ほとんど明らかだが、証明することもできる。）

練習
（1）
アルゴリズムnaïve_powの正当性に関する命題を設定せよ。（不変条件を定めよ。）
（2）（1）で設定した命題を数学的帰納法で証明せよ。

**(1)のような命題（条件）を、不変条件（invariant）という。
ループ内の条件等は、特にループ不変条件という。
アサーション(assertion、表明)ともいう。**

49

アルゴリズムnaïve_powの時間計算量

- 高々 n 回の繰り返し。
- 各繰り返し中では、1回の乗算と、1回の代入が行われているだけである。

$\therefore O(n)$ の時間計算量である。

50

数学の記号とプログラム

$\prod_{i=0}^{n-1} x_i$ ⇔

```
1. for(i=0;i<n;i++){
2.    f=f*x[i] ;
3. }
```

$\sum_{i=0}^{n-1} x_i$ ⇔

```
1. for(i=0;i<n;i++){
2.    f=f+x[i] ;
3. }
```

51

高速なべき乗の求め方。

- 注意
  - とりあえず、入力に制限を加える
  - n が2のべき乗と仮定する。すなわち、ある整数 k が存在して、 $n = 2^k$ と表せる。
- アィディア
  - 倍、倍に計算した方が高速に値を求められる。

$$x^i \bullet x^i = x^{2i}$$

  - （一方、 $x^i \bullet x = x^{i+1}$ ）

52

例

3^8 を求める。

素朴な方法

$3^0 = 1 \rightarrow 3^1 = 1 \bullet 3 = 3 \rightarrow 3^2 = 3 \bullet 3 = 9 \rightarrow 3^3 = 9 \bullet 3 = 27$

$\rightarrow 3^4 = 27 \bullet 3 = 81 \rightarrow 3^5 = 81 \bullet 3 = 243 \rightarrow 3^6 = 243 \bullet 3 = 729$

$\rightarrow 3^7 = 729 \bullet 3 = 2187 \rightarrow 3^8 = 2187 \bullet 3 = 6561$

高速な方法

$3^1 = 3 \rightarrow 3^2 = 3 \bullet 3 = 9 \rightarrow 3^4 = 9 \bullet 9 = 81 \rightarrow 3^8 = 81 \bullet 81 = 6561$

53

アルゴリズムfast_pow(x,n)
入力：x,n（ただし、 $n = 2^k$ ）
出力：xのn乗

```
1. f=x;
2. for(i=1;i<k;i++){
3.    f=f*f;
4. }
5. return f;
```

54

命題FP1（fast_powの正当性）

forループが k回繰り返されたとき、
fの値は、

$$f = x^{2^k}$$

である。

証明　繰り返し回数kによる帰納法による。
繰り返し回数kのときのfの値を f_k と表す。

基礎: $k = 0$

アルゴリズム中のステップ1より、 $f_0 = x$ である。
一方、右辺= $x^{2^0} = x^1 = x$
よって、命題は成り立つ。

55

帰納: $0 < k$

$0 \le k' < k$ なる全ての k' に対して、
命題が成り立つと仮定する（帰納法の仮定）
$k' = k - 1$ として、 $k-1$ 回の繰り返しで、

$$f_{k-1} = x^{2^{k-1}}$$

である。よって、 k 回めの繰り返しでは、

$$f_k = f_{k-1} \bullet f_{k-1} = x^{2^{k-1}} \bullet x^{2^{k-1}} = x^{2 \bullet \left(2^{k-1}\right)} = x^{2^k}$$

QED

56

fast_powの時間計算量

アルゴリズムは、明らかに、k回繰り返す。
繰り返し中は、1回の乗算しか行っていない。
したがって、

$O(k)$

の時間計算量である。

$$n = 2^k$$
$$\therefore k = \log_2 n$$

なので、結局

$O(\log n)$

の時間計算量である。

57

一般の自然数に対する高速なべき乗アルゴリズム

n **が2のべき乗でないときを考える。**

このとき、前の高速なべき乗アルゴリズムをサブルーチンとして用いることができる。
n **の2進数表現を次のように表す。**

$$(n)_{10} = \left(b_m b_{m-1} \cdots b_1 b_0\right)_2$$
$$= 2^m \times b_m + 2^{m-1} \times b_{m-1} + \cdots + 2^0 \times b_0$$

58

このとき、 x^n **は次のように表せる。**

$$x^n = x^{2^m b_m + 2^{m-1} b_{m-1} + \cdots + 2^0 b_0}$$
$$= \left(x^{2^m}\right)^{b_m} \bullet \left(x^{2^{m-1}}\right)^{b_{m-1}} \bullet \cdots \bullet \left(x^1\right)^{b_0}$$
$$= \prod_{i=0}^{m} \left(x^{2^i}\right)^{b_i}$$

これより、一般のnに対する高速なアルゴリズムが得られる。

59

アルゴリズムgeneral_fast_pow(x,n)
入力:x,n(nは一般の数)
出力:xのn乗

1. f=1.0;
2. nを2進数 $\left(b_m b_{m-1} \cdots b_1 b_0\right)_2$ に変換する。
3. for(i=0;i<m;i++){
4. if($b_i == 1$){
5. f=f*fast_pow(x, 2^i);
6. }
7. }
8. return f;

60

general_fast_pow(x,n)の時間計算量

general_fast_pow(x, n)の時間計算量を$T_G(n)$と表し、fast_pow(x, 2^i)の時間計算量を $T_F(i)$ と表す。

3−7のループの繰り返しは、$m = \log n$ 回繰り返される。

ループの各繰り返しにおける実行時間の総和により $T_G(n)$を求める。

61

$$T_G(n) \le \underbrace{T_F(0) + T_F(1) + \cdots + T_F(m)}_{\log n} + c_1$$
$$\le \underbrace{\log 1 + \log 2 + \cdots + \log n}_{\log n} + c_1$$
$$\le \underbrace{\log n + \log n + \cdots + \log n}_{\log n} + c_1$$
$$\le \log^2 n + c_1$$
$$\therefore T_G(n) = O(\log^2 n)$$

62

さらなる高速化

$$x^n = x^{2^m b_m + 2^{m-1} b_{m-1} + \cdots + 2^0 b_0}$$
$$= \left(x^{2^m}\right)^{b_m} \bullet \left(x^{2^{m-1}}\right)^{b_{m-1}} \bullet \cdots \bullet \left(x^1\right)^{b_0}$$

であるが、x^{2^m} を求めるための高速べき乗アルゴリズム fast_pow(x, 2^m)の途中段階で全てべき乗

$$x^{2^{m-1}}, x^{2m-2}, \cdots, x^1$$

が出現していることに注意する。

63

アルゴリズムsuper_fast_pow(x,n)
入力:x,n(nは一般の整数)
出力:xのn乗

```
1. f=1.0;
2. tmp=x;
3. nを2進数 (b_m b_{m-1} ... b_1 b_0)_2 に変換する。
4. for(i=0;i<=m;i++){
5.   if(b_i == 1){
6.       f=f*tmp;
7.   }
8.   tmp=tmp*tmp;
9. }
10.return f;
```

64

super_fast_pow(x,n)の時間計算量

super_fast_pow(x, n)の時間計算量を $T_S(n)$と表す。

3−7のループの繰り返しは、$m+1 = \log n + 1$ 回繰り返される。

ループの各繰り返しは、$O(1)$ 時間で実現可能である。

よって、

$$T_S(n) = O(\log n)$$

である。

65

多項式評価の問題

- 素朴な方法
- 高速べき乗アルゴリズムの利用
- ホーナー法

66

多項式の値を評価する問題

- 入力: $x, n, a_0, a_1, \cdots, a_n$
- （ここで、入力サイズは、n とします。）
- 出力:

$$f(x) = a_0 + a_1 x + \cdots + a_n x^n$$

$$= \sum_{i=0}^{n} a_i x^i$$

67

素朴な多項式の求め方

- アィディア
  - 各項を素朴な乗算で計算し、総和を求める。

$$f(x) = a_0 + a_1 x + \cdots + a_n x^n$$

$$= \sum_{i=0}^{n} a_i x^i$$

$$= \sum_{i=0}^{n} a_i \left(\prod_{j=0}^{i-1} x \right)$$

68

アルゴリズムnaive_poly()
入力:x,次数n,係数a[n]
出力: $f(x) = \sum_{i=0}^{n} a[i] x^i$

```
1. fx=0.0;
2. for(i=0;i<=N;i++){
3.    tmp=a[i];
4.    for(j=0;j<i;j++){
5.        tmp=tmp*x;
6.    }
7.   fx=fx+tmp;
8. }
9. return fx;
```

3-6は、第i項の計算

69

素朴な多項式計算アルゴリズムの正当性

命題NP1（naive_polyの正当性1）

2.のforループが i回繰り返されたとき、tmpの値は、

$$a_i x^i$$

である。

70

証明

アルゴリズム中のステップ3より、$tmp = a_i$に設定される。

また、4の繰り返しは明らかにi回である。

したがって、 x はi回乗算される。
よって、

$$tmp = a_i x^i$$

である。

より厳密な帰納法でも証明できる。

QED

71

命題NP2（naive_polyの正当性2）

naïve_polyは

$$f(x) = \sum_{i=0}^{n} a_i x^i$$

を計算する。

証明　次数nに関する帰納法による。

基礎 $n = 0$

$tmp = a_0 = f(x)$ **であり正しい。**

帰納 $n > 0$

$0 \le n' < n$ **の時正しいと仮定する。**

72

$n' = n-1$ とする。
n-1の繰り返しのときのfxの値を $f_{n-1}(x)$ と書く。
このとき、帰納法の仮定より、

$$f_{n-1}(x) = \sum_{i=0}^{n-1} a_i x^i$$

が成り立つ。n 回目の繰り返しでは、$i = n$ なので、

$$tmp = a_n x^n \qquad \text{（命題PL1より）}$$

また、ステップ7より、

$$f(x) = f_{n-1}(x) + tmp$$
$$= \left(\sum_{i=0}^{n-1} a_i x^i \right) + a_n x^n$$
$$= \sum_{i=0}^{n} a_i x^i \qquad QED$$

73

素朴な多項式計算アルゴリズムの計算時間

- ステップ5の部分が最も時間を多く繰り返される。

ステップ4のforループは、i の値にしたがって、
i 回繰り返される。
よって、時間計算量を $T(n)$ とすると、
$T(n)$ は、次のように計算できる。

$$T(n) = \sum_{i=0}^{n} i$$
$$= 1 + 2 + 3 + \cdots + n$$
$$= \frac{n(n+1)}{2}$$
$$= O(n^2)$$

74

以上から、naive_polyの最悪時間計算量は、

$$O(n^2)$$

であることがわかる。
また、繰り返し回数は、入力サイズ(n)だけに依存し、
問題例(係数配列の状態)に依存しない。
よって、

$$\Theta(n^2)$$

の時間計算量を持つこともわかる。

75

高速なべき乗計算アルゴリズムを利用した多項式評価

$$T(n) = \sum_{i=0}^{n} \log i$$
$$\leq \log 1 + \log 2 + \cdots + \log n$$
$$< \log n + \log n + \cdots + \log n$$
$$= n \log n$$
$$\therefore T(n) = O(n \log n)$$

と計算できるので、$O(n \log n)$ のアルゴリズムといえる。
（べき乗の計算に、mathライブラリのpowを用いた場合、
この計算量になると考えられる。）

76

アルゴリズム：lib_pow_poly()
入力：x,次数n,係数a[n]
出力：

$$f(x) = \sum_{i=0}^{n} a[i] x^i$$

```
1. fx=0.0;
2. for(i=0;i<=N;i++){
3.    fx=fx+a[i]*pow(x,(double)i);
4.    }
5. return fx;
```

77

ホーナーの方法

- アィディア
  - xをできるだけくくりだしながら計算する。

$$f(x) = a_0 + a_1 x + \cdots + a_n x^n$$
$$= (a_0 + (a_1 + (\cdots (a_{n-1} + (a_n) * x) * x) \cdots) * x * x)$$

78

アルゴリズムを示すまえに、等式の正当性を証明する。

命題H1（hornerの正当性1）

$$f(x) = a_0 + a_1 x + \cdots + a_n x^n$$
$$= (a_0 + (a_1 + (\cdots (a_{n-1} + (a_n) * x) * x) \cdots) * x * x)$$

証明 $a_0, a_1, \cdots, a_n$ に対して、

$$f_k(x) \equiv a_{n-k} + a_{n-k+1} x + a_{n-k+2} x^2 + \cdots + a_n x^k$$
$$= \sum_{i=0}^{k} a_{n-k+i} x^i$$

と定義する。

79

このとき、各関数は次のような系列になる。

$$f_0(x) = a_n$$
$$f_1(x) = a_{n-1} + a_n x$$
$$f_2(x) = a_{n-2} + a_{n-1} x + a_n x^2$$
$$\vdots$$
$$f_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = f(x)$$

この $f_k(x)$ に関して、
次式を k に関する帰納法で命題を証明する。

$$f_k(x) = (a_{n-k} + (a_{n-k+1} + (\cdots (a_{n-1} + (a_n) * x) \cdots) * x) * x)$$

80

基礎 $k = 0$

$$f_0(x) = a_n = (a_n)$$

であり、満たされる。

帰納 $k > 0$

$0 \le k' < k$ なる全ての k' で

$$f_{k'}(x) = a_{n-k'} + a_{n-k'+1} x + \cdots + a_n x^{k'}$$
$$= (a_{n-k'} + (a_{n-k'+1} + (\cdots (a_{n-1} + (a_n) * x) \cdots) * x) * x)$$

と仮定する。（帰納法の仮定）

$k' = k - 1$ として

$$f_{k-1}(x) = a_{n-k+1} + a_{n-k+2} x + \cdots + a_n x^{k-1}$$
$$= (a_{n-k+1} + (a_{n-k+2} + (\cdots (a_{n-1} + (a_n) * x) \cdots) * x) * x)$$

81

このとき、

$$a_{n-k} + a_{n-k+1} x + \cdots + a_n x^k$$
$$= a_{n-k} + (a_{n-k+1} + \cdots + a_n x^{k-1}) * x$$
$$= a_{n-k} + (a_{n-k+1} + (\cdots (a_{n-1} + (a_n) * x) \cdots) * x) * x$$
$$= (a_{n-k} + (a_{n-k+1} + (\cdots (a_{n-1} + (a_n) * x) \cdots) * x) * x)$$

帰納法の仮定を用いている。

よって、命題は成り立つ。

QED

82

ホーナーの方法の計算手順

$$(a_0 + (a_1 + (\cdots (a_{n-1} + (a_n) * x) * x) \cdots) * x) * x)$$

83

練習

次の多項式を素朴な計算法と、ホーナーの方法により計算せよ。

$$f(x) = x^5 - 12x^4 + 3x^3 - 8x^2 + x - 5$$

(1) $f(2)$ (2) $f(-3)$

84

アルゴリズムhorner_poly()
入力:x,次数n,係数a[n]
出力: $f(x)=\sum_{i=0}^{n} a[i]x^i$

```
1. horner(int k,double x){
2.   if(k==0){
3.       return a[n];
4.   }else{
5.       f_k=a[n-k]+horner(k-1,x)*x;
6.       return f_k;
7.    }
8. }
```

85

ホーナーの方法の計算時間

- 計算時間を $T(n)$ とすると次の漸化式を満たす。

$$T(n)=\begin{cases} c_0 & (n=0) \\ c_1+T(n-1)+c_2 & (n>0) \end{cases}$$

ここで、 c_1 を加算に必要な計算時間とし、
c_2 を乗算に必要な計算時間としている。

$c=\max\{c_1,c_2\}$ とすると、次のようにかける。

$$T(n)\le\begin{cases} c_0 & (n=0) \\ T(n-1)+2c & (n>0) \end{cases}$$

86

$$\begin{aligned} T(n) &\le T(n-1)+2c \\ &\le \big(T(n-2)+2c\big)+2c &&= T(n-2)+4c \\ &\vdots \\ &\le T(0)+2nc &&\le (c_0+2c)n \end{aligned}$$

これより、 $T(n)=O(n)$

よって、ホーナーの方法による多項式の計算は、
線形時間アルゴリズムである。

87

ホーナーの方法の繰り返し版

ホーナーの方法は、次のように、繰り返しを用いても
実現できる。

```
1. horner(int k,double x){
2.   double f_k=a[n];
3.   for(k=1;k<=n;k++){
4.      f_k=a[N-k]+( f_k )*x;
5.   }
6.    return f_k;
7. }
```

このアルゴリズムも明らかに線形時間で実行される。

88

練習

(1)ループによるホーナー法において、ループ不変条件を設定せよ。

(2)(1)を数学的帰納法で証明せよ。

89