

2. アルゴリズムの高速化

アルゴリズムにおける 大幅な性能アップ

- 多項式時間アルゴリズム VS
対数時間アルゴリズム
(最大公約数の問題)
- 指数時間アルゴリズム VS
多項式時間アルゴリズム
(フィボナッチ数列を求める問題)

最大公約数問題

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、

$$\max\{a, b\}$$

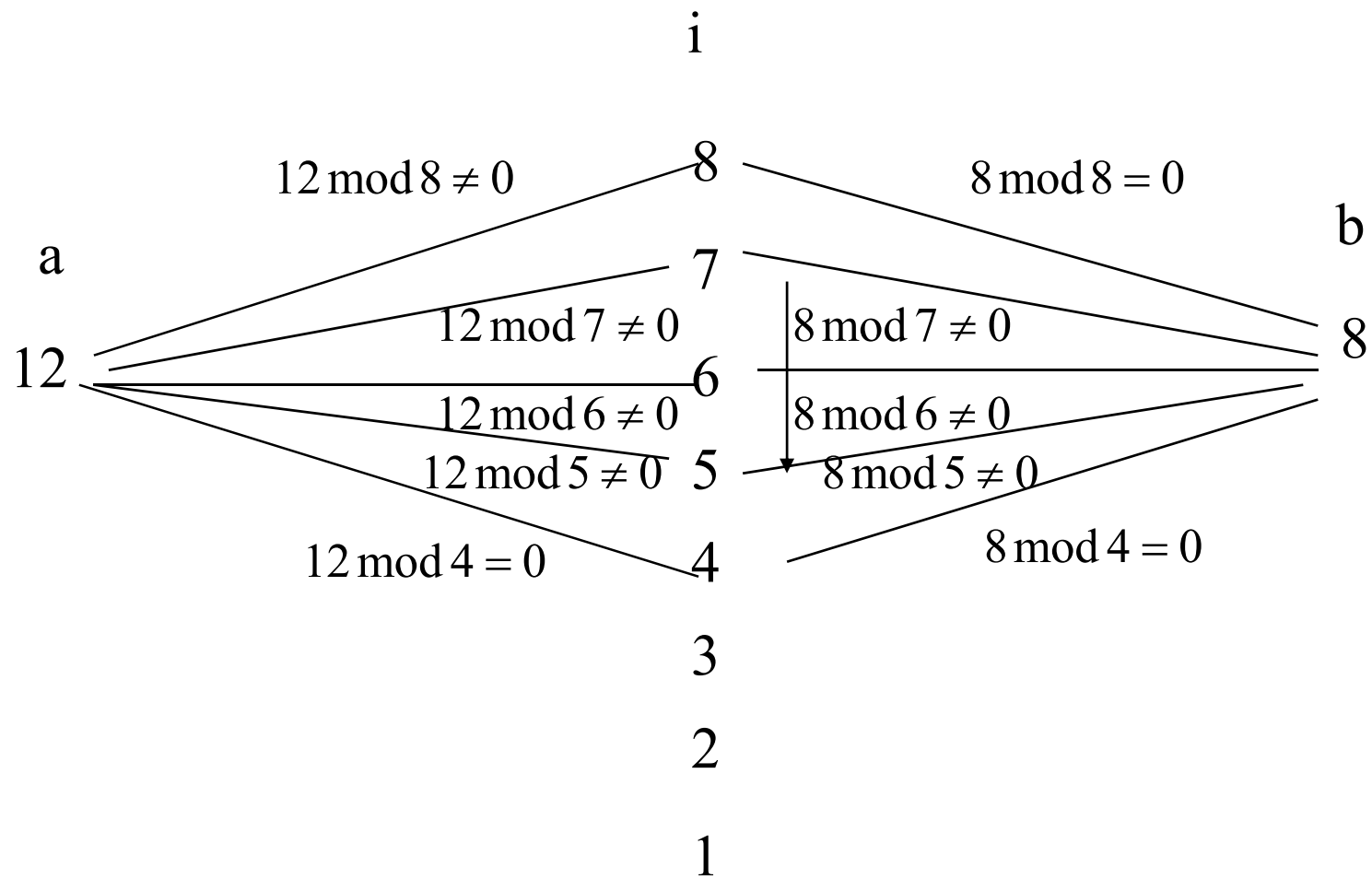
とします。)

出力: a, b の最大公約数

$$\gcd(a, b)$$

素朴な最大公約数発見法

- 注目点
 - すべて整数
- アイディア
 - $1 \sim \min\{a, b\}$ 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。



アルゴリズム $\text{naïve_gcd}(a,b)$

入力: a, b

出力: $\text{gcd}(a,b)$

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

- 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。

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。

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 ( $r \neq 0$ ) ) {
5.     big=small;
6.     small=r;
7.     r=big % small;
8. }
9. return small;
```

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

練習

$$a = 126, b = 70$$

に対して、ユークリッドの互除法を用いて、
最大公約数を求めよ。

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 (割り算の性質)

2つの自然数 $a, b (a \geq b > 0)$ に対して、
2つの自然数 q, r を用いて、

$$a = bq + r \quad (0 \leq r < b)$$

と表せる。

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。

$$1 = b * 1 + 0$$

このとき、 $q=1$ 、 $r=0$

qを商、rを余りだと
考える。

場合2: a=1, $b > 1$ のとき

$$1 = b * 0 + 1$$

このとき、 $q=0$ 、 $r=1$

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、

$$n=bq+r \ (0 \leq r < b) \quad \text{—————} \quad \textcircled{1}$$

が成り立つ。

r は自然数なので、

$$0 \leq r < b$$

は、

$$1 \leq r+1 \leq b \quad \text{—————} \quad \textcircled{2}$$

と同じである。

①の両辺に1を加える。

$$n+1=bq+r+1$$

これも ②の等号の有無で2つの場合に分ける。

場合1: $b=r+1$ のとき。

$$n+1=bq+r+1$$

$$\therefore n+1=bq+b$$

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

商 $q+1$ 、余り 0

場合2: $b>r+1$ のとき。

$$n+1=bq+r+1$$

$$\therefore n+1=bq+(r+1)$$

商 q 、余り $r+1$

QED

命題E2(割り算の一意性)

2つの自然数 $a, b (a \geq b > 0)$ に対して、
2つの自然数 q, r を用いて、

$$a = bq + r \quad (0 \leq r < b)$$

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、

$$a = bq + r \quad (0 \leq r < b)$$

$$a = bq' + r' \quad (0 \leq r' < b')$$

と表現できる。ここで、 q と q' か、 r と r' のどちらかは、異なってなければならい。

両辺を減算する。

$$\begin{array}{l} a=bq+r \quad (0 \leq r < b) \\ \text{—) } a=bq'+r' \quad (0 \leq r' < b) \end{array}$$

$$0=(q-q')b+r-r' \quad (-b < r-r' < b)$$

$$\therefore (q'-q)b=r-r'$$

これは、 $r-r'$ が b の倍数であることを意味する。

一方、 $(-b < r-r' < b)$ であり、**全て整数**なので、

$$r-r'=0$$

$$\therefore r=r'$$

これより、 $q'=q$ 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

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

例えば、

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

Common Divisor
(公約数)

命題E3 (約数集合の普遍性)

$a, b (a \geq b)$ に対して、先の命題E1, E2で定まる表現を

$$a = bq + r \quad (0 < r < b)$$

とする。このとき、

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

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、

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

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

また、

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

より、

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

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。

$$a = bq_1 + r_1 \quad (0 < r_1 < b)$$

$$b = r_1q_2 + r_2 \quad (0 < r_2 < r_1)$$

$$r_1 = r_2q_3 + r_3 \quad (0 < r_3 < r_2)$$

⋮

このとき、余りの系列は、単調減少する。すなわち、

$$b > r_1 > r_2 > \dots$$

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。

²²
QED

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）

$a, b (a \geq b)$ に対して、先の命題E1, E2で定まる表現を

$$a = bq + r \quad (0 < r < b)$$

とする。このとき、

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

証明の前に、具体例で確認する。

$$a = 36, b = 21$$

$$36 = 21 \times 1 + 15$$

$$< \frac{36}{2}$$

$$21 = 15 \times 1 + 6$$

$$< \frac{21}{2}$$

$$15 = 6 \times 2 + 3$$

$$< \frac{15}{2}$$

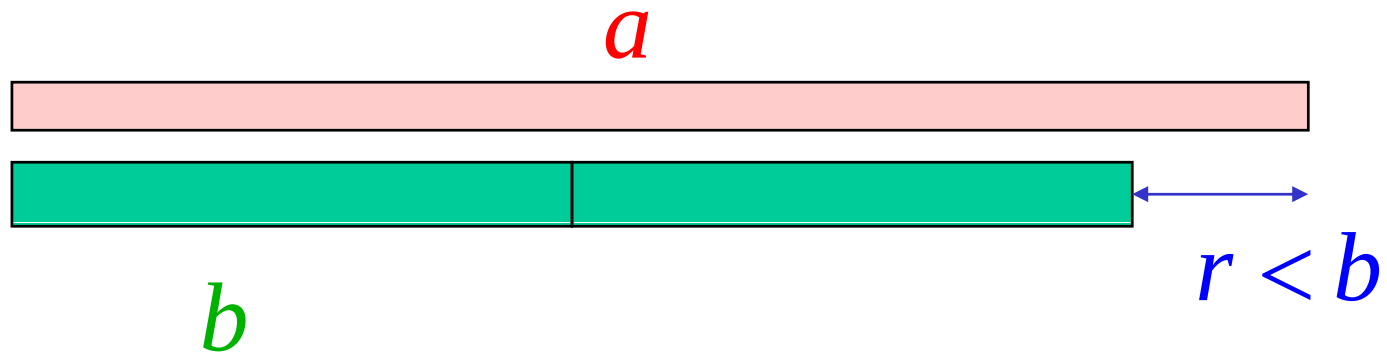
$$6 = 3 \times 2 + 0$$

証明

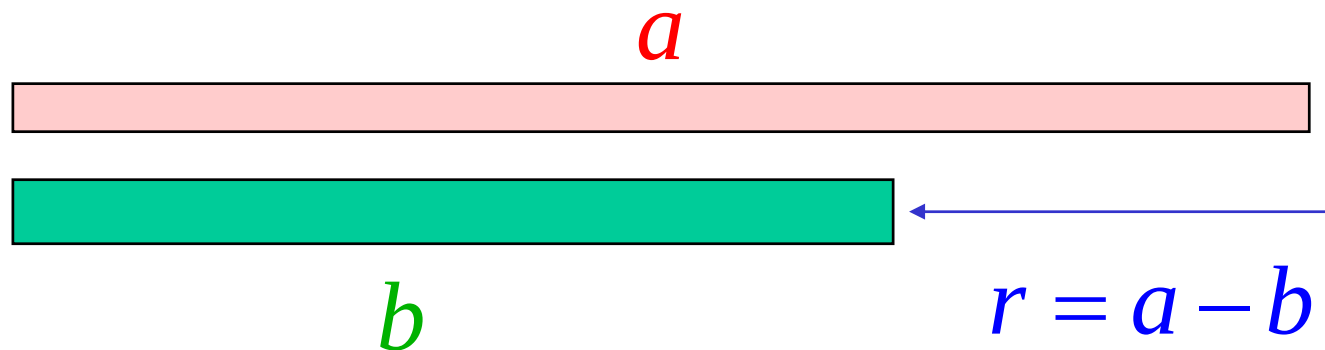
2つの場合に分けて、 $r < \frac{a}{2}$ を示す。

場合1: $b \leq \frac{a}{2}$ のとき、

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



場合2: $b > \frac{a}{2}$ のとき、
このとき、 $a - b < \frac{a}{2}$ に注意する。

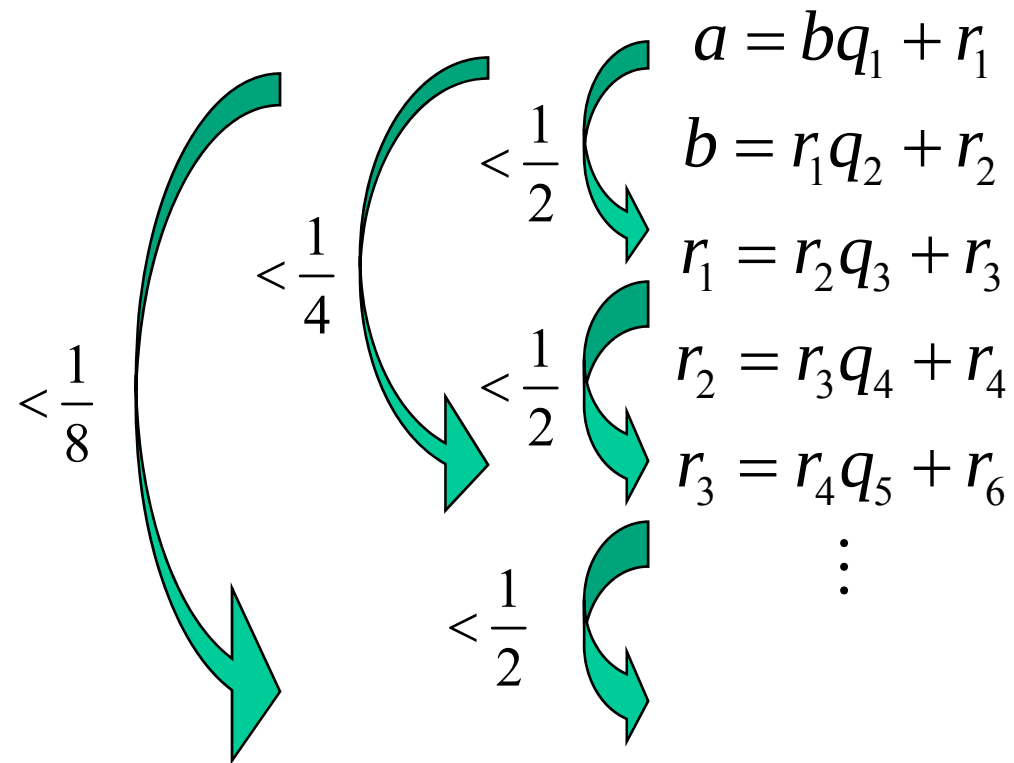


$$a = bq + r$$

において、商として $q = 1$ にしかない。
したがって、

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

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。

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

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。

$$a = 36, b = 21$$

$$36 = 21 \times 1 + 15 \longleftarrow \text{gcd}(36, 21)$$

$$21 = 15 \times 1 + 6 \longleftarrow \text{gcd}(21, 15)$$

$$15 = 6 \times 2 + 3$$

$$6 = 3 \times 2 + 0 \longleftarrow \text{割り切れるときが基礎}$$

サイズが小さい同じ
問題を解くことに注意する。

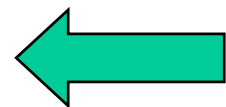
アルゴリズム Δ recursive_gcd(a,b)

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

フィボナッチ数列問題

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

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

漸化式と再帰アルゴリズム

アルゴリズム fibo_rec

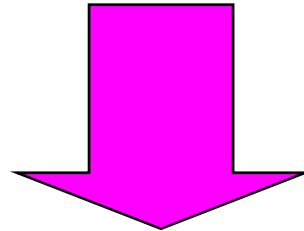
入力: N

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. }
```

$$\left\{ \begin{array}{ll} f(0) = 0 & n=0 \\ f(1) = 1 & n=1 \\ f(n) = f(n-1) + f(n-2) & n \geq 2 \end{array} \right.$$

- 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、元の問題の解放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。

- 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。

アルゴリズムfibonacci_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.}
```

$$T(0) = c_1$$

$$T(1) = c_2$$

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

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

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。

この漸化式を解く。

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

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。

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。

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

漸化式は、その項より前の項番号を持つ式を組み合わせで定義される。

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。

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. }
```

$$f(0) = 0$$

$$f(1) = 1$$

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

● アルゴリズムfibonacci_arrayの時間計算量

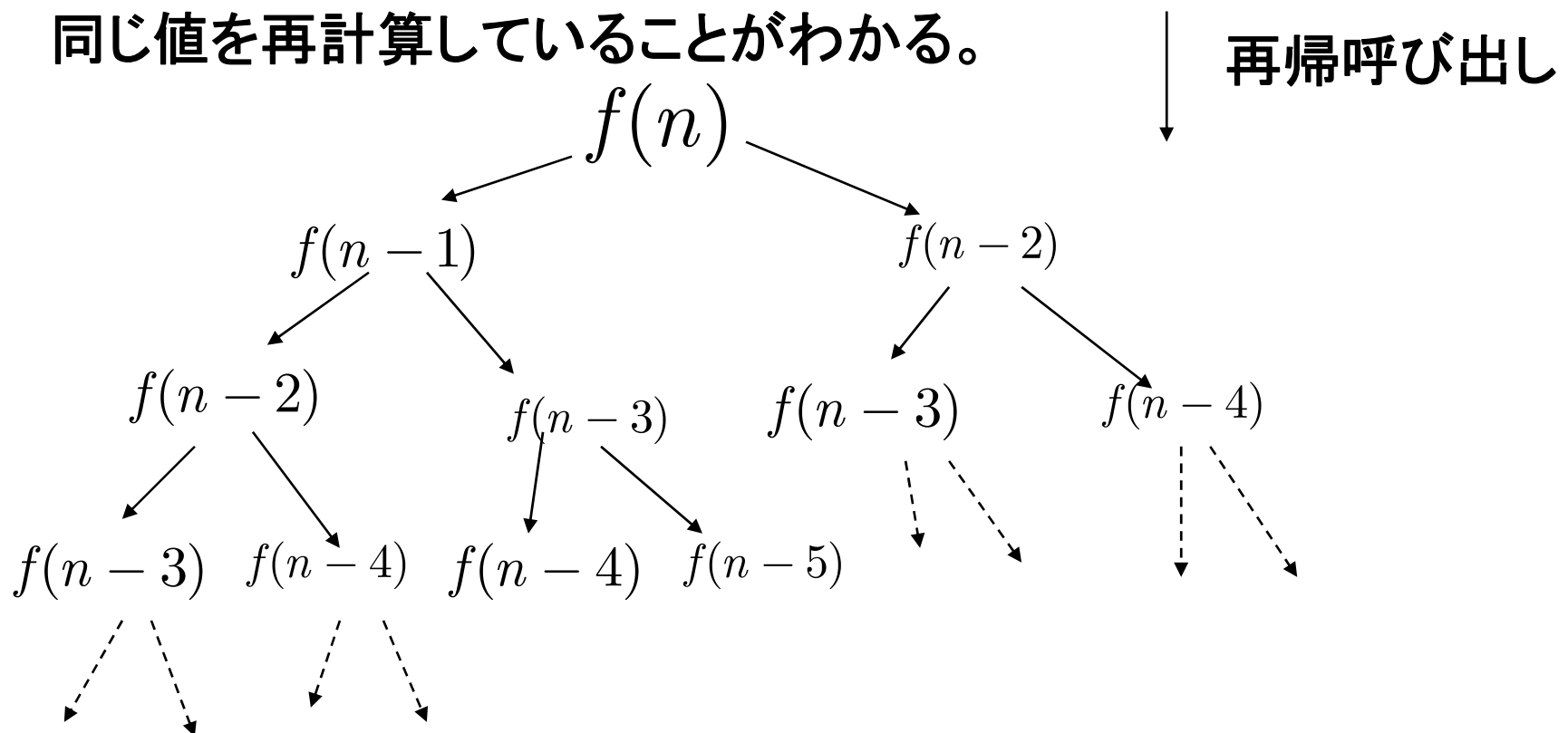
forループをn-1回繰り返しているだけなので、
 $O(n)$ 時間

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。)

fibo_recが低速な理由

- 主に、 unnecessary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。



$f(1)f(0)$ $f(1)f(0)$ $f(1)f(0)$

更なる高速化

○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。