再帰

計算機アルゴリズム特論：2015年度

只木進一

再帰
recursion

- 関数や手続きが、その関数・手続きそのもので記述される
- 参考：数学的帰納法
  - 自然数nに関する命題$S(n)$
  - $S(1)$は正しい（多くの場合自明）
  - $S(n)$を仮定して$S(n+1)$が成立を導出

- 関数$f(n)$の値は$f(n-1)$が分かると計算できる

例：階乗

$$n! = \prod_{k=1}^{n} k$$

```
function factorial(n){
  int k=1;
  for(int i=1;i<=n;i++)k *= I;
  return k;
}
```

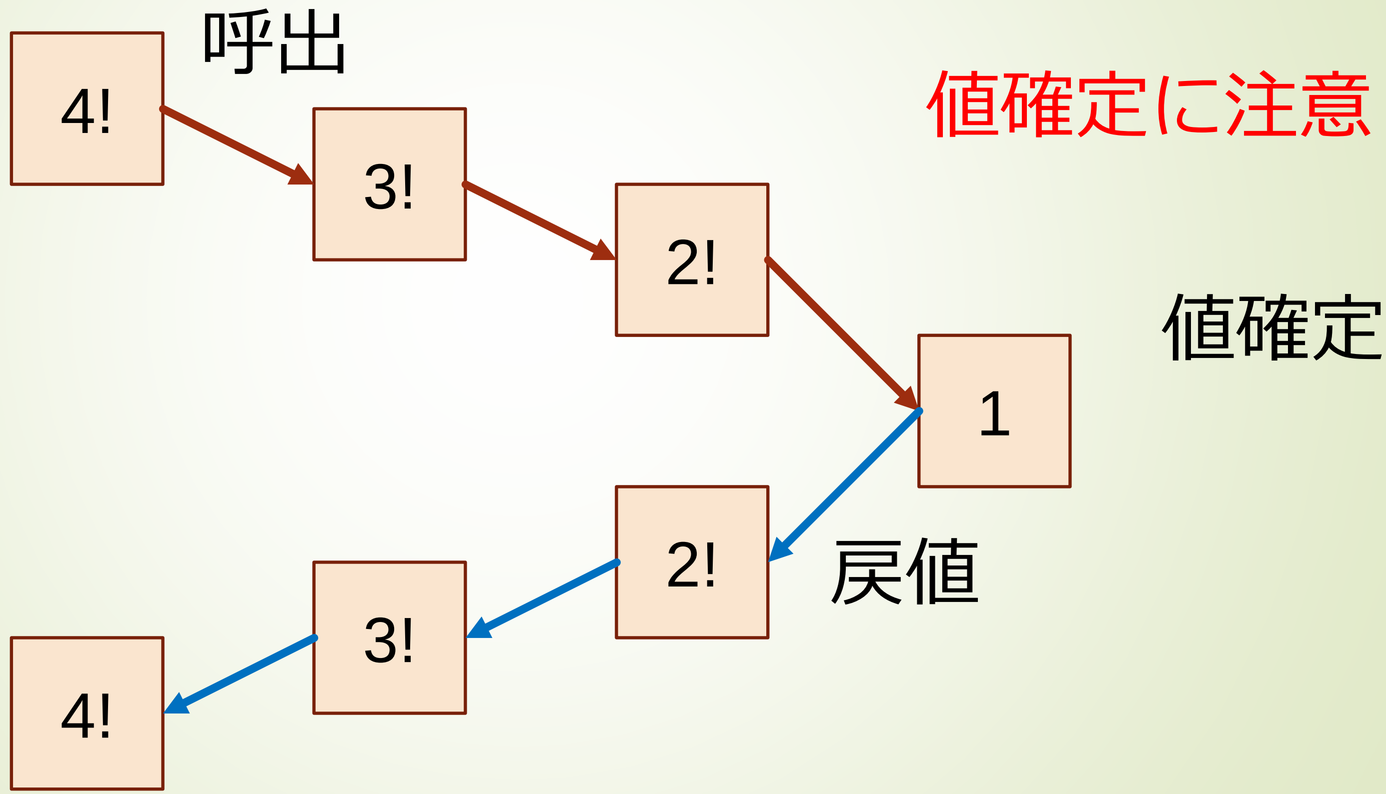
例：階乗

$$n! = \prod_{k=1}^{n} k$$

$$n! = n \times (n-1)!$$

```
function factorial(n){
  if ( n==1) return 1;
  return n * factorial (n-1);
}
```

再帰の動作



値確定に注意

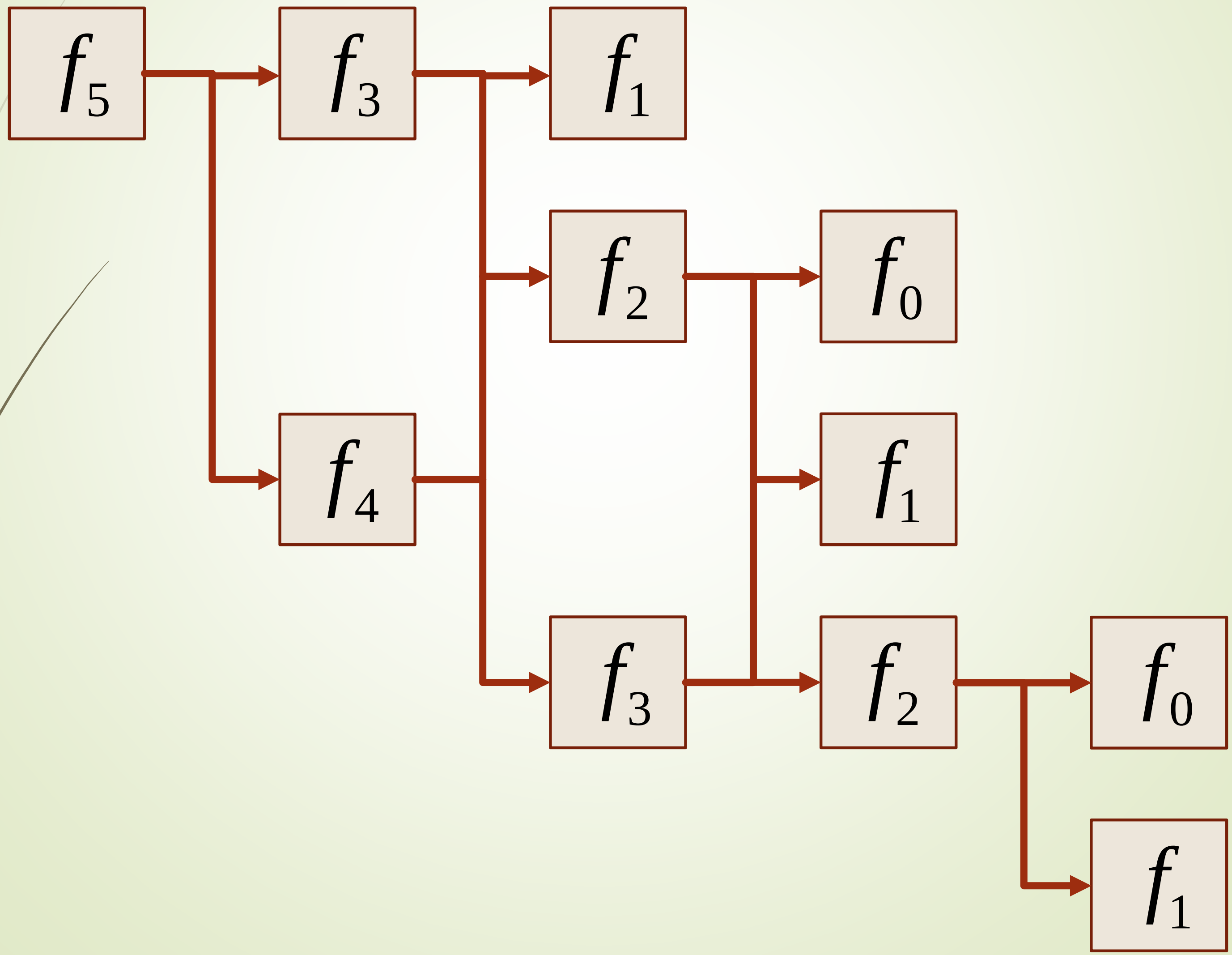
戻りの部分に留意

例：Fibonacci数

$$f_0 = 0$$

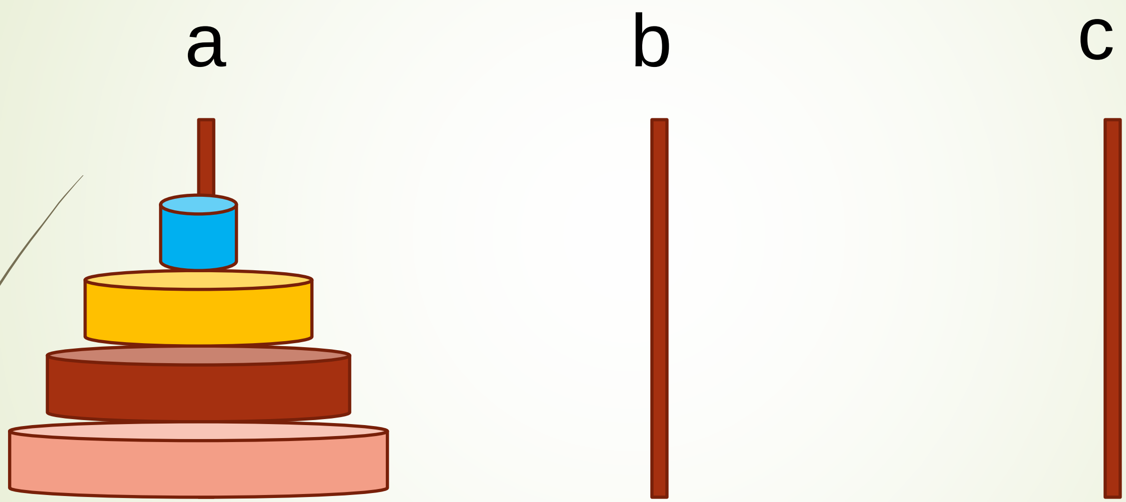
$$f_1 = 1$$

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


f5
f3
f1
f2
f0
f4
f1
f3
f2
f0
f1

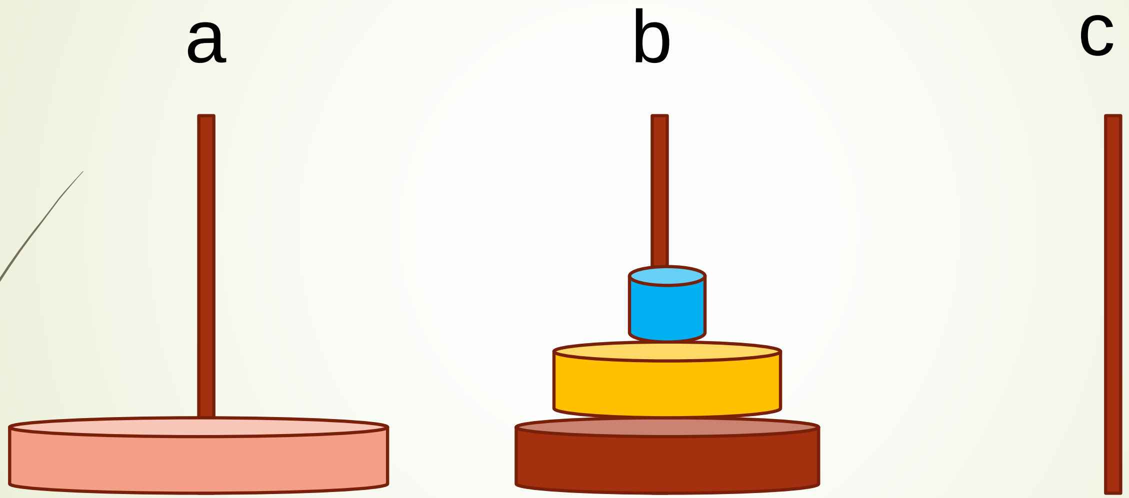
- 単純な再帰では、下位の二つの値が必要
  - f_nの計算には、f_{n-1}とf_{n-2}の二つが必要
  - 一つの再帰毎に、必要な計算が倍になってしまう
- しかし、下位の値さへ分かれば良いはず。
  - f_nの計算には、$f_m\,(m < n)$だけが必要なはず。

例：ハノイの塔



- 円盤をaからcへ移動させる
- 一度に一つの円盤を移動
- 小さい円盤が常に上になければならない

仮に3枚を動かすことが可能ならば



- 一番大きな円盤をcへ
- 残り3枚をcへ

ハノイの塔の再帰表現

- N枚の円盤をaからcへ移動させる
  - $N-1$枚の円盤をaからbへ移動させる
  - 最後の一枚をcへ移動させる
  - $N-1$枚の円盤をbからcへ移動させる

```
moveDisks(始点、終点、枚数) {
   if (枚数== 1) {
      始点から終点に1枚移動;
      return;
   }
   o = 空いている棒;
  moveDisks(始点, o, 枚数 - 1);
  残った一枚を終点へ移動;
  moveDisks(o, 終点, 枚数 - 1);
}
```

```
void moveDisks(int from, int to, int number) {
    if (number == 1) {
        moveSingleDisk(from, to);
        return;
    }
    int o = 3 - (from + to); //other pillars
    moveDisks(from, o, number - 1);
    moveSingleDisk(from, to);
    moveDisks(o, to, number - 1);
}
```

計算量
円盤の移動回数

- n枚の円盤の移動回数$N(n)$

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

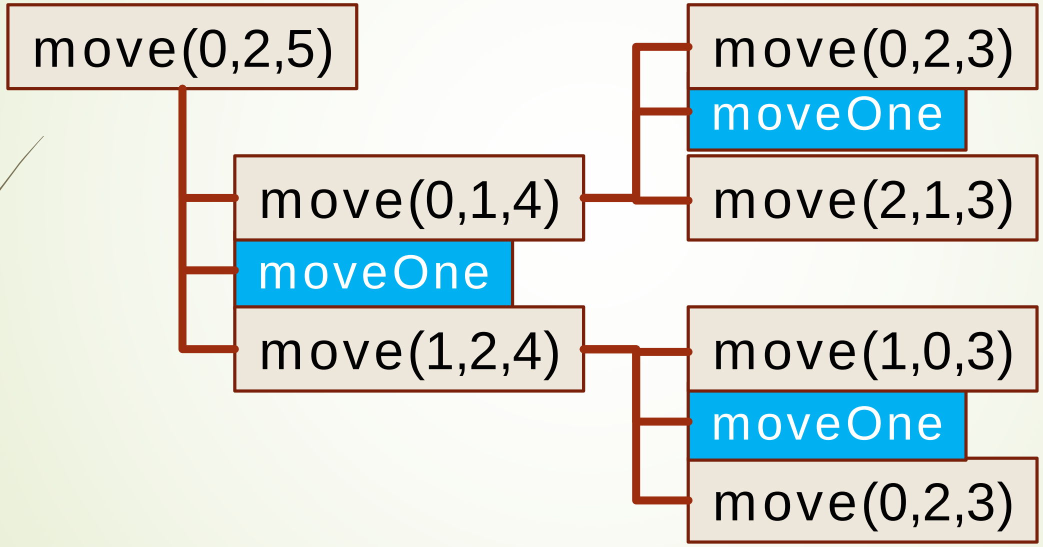
- 解

$$N(n) = 2^n - 1$$

数学的帰納法により証明せよ

- $O(2^n)$の計算時間を要する

計算量
なぜ、そんなに時間がかかる



移動する枚数を1枚減らす度に、2回の移動に増える

再帰計算の注意

- 停止条件
  - 無限ループにならないように注意
- 計算時間
  - 複数の自関数を呼ぶ場合に注意