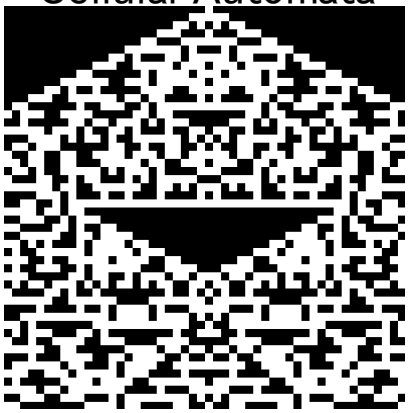
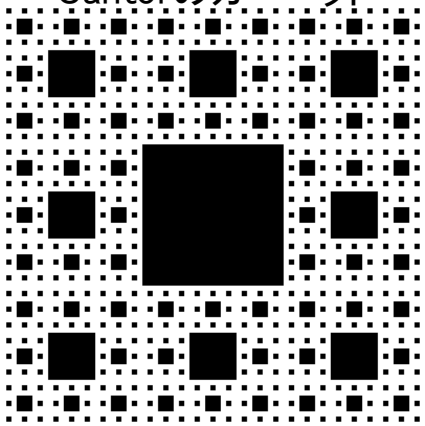
第4章 関数から計算へ

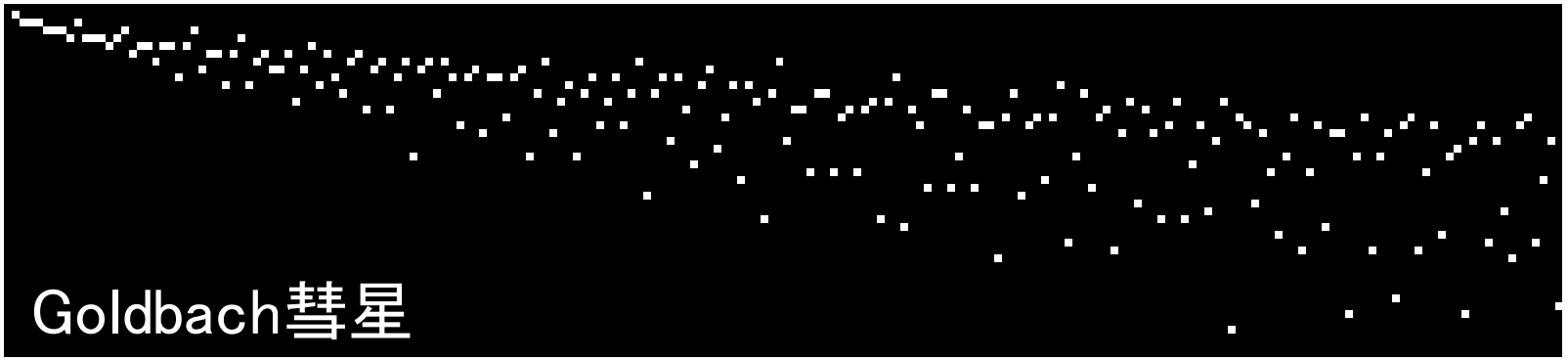
- 再帰による反復計算(補足)
- 繰り返し命令による反復計算
- 配列(3.4)

Cellular Automata



Cantorのカーペット





Collatz予想 (p.67)

- 予想: どんな正の整数でも「偶数ならば2で割る，奇数ならば3倍して1加える」を繰り返せば1になる

pp.37-38の関数tnpo(n)

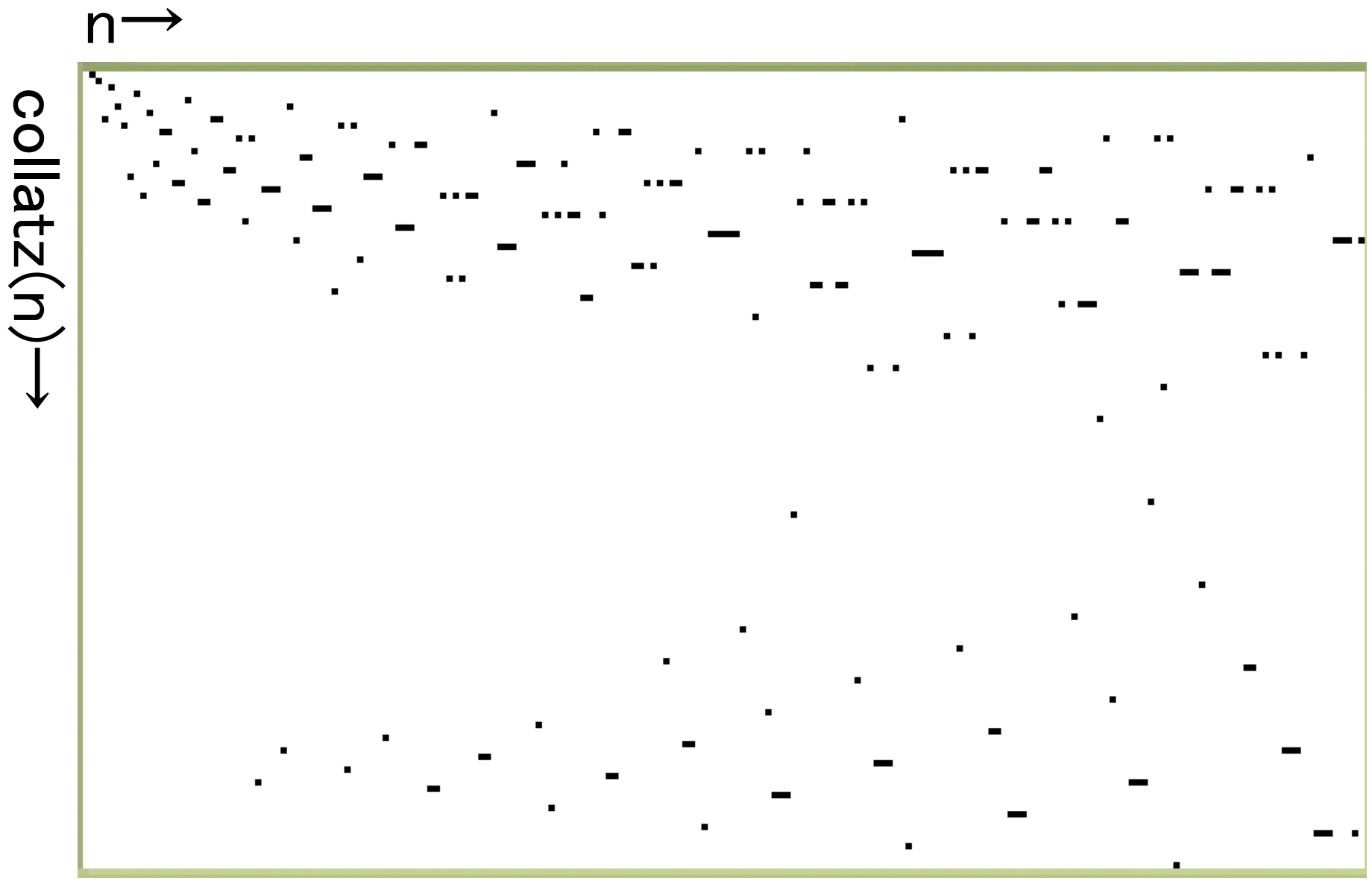
- 問題: collatz(n) 1になるまでの変化の回数
  - collatz(3)=7, colltaz(5)=5, collatz(2)=1
- クイズ: collatz(7)はいくつか? (手計算で)

ターミナルで

```
cd Downloads
ruby c.rb 回数
```

解答したらcollatz(n)を
collatz(tnpo(n))で表わす式を
考え、さらに関数を定義してみよ

1に達する回数の不思議な振舞



補足: 再帰による反復計算のパターン化

- f(n)がf(n−1)を使って定義できる場合
  - 1からnまでの和、積、最大、最小など
- f(n)がf(g(n))で定義できる場合
  - collatz(n)など
- 補助関数が必要な場合
  - 「約数の個数」など

補足: f(n)がf(n−1)を使って定義できる場合

- 例: 1からnまでの和sum(n)
  - sum(n)=sum(n−1) + n
- あとはn=0の場合を考えて

$$\mathrm{sum}(n) = \begin{cases} \mathrm{sum}(n-1)+n & (n>0) \\ 0 & (n=0) \end{cases}$$

- Rubyの関数にする→

```
def sum(n)
  if n > 0
    sum(n−1)+n
  else
    0
  end
end
```

補足: f(n)がf(g(n))で定義できる場合

- tnpo(n) = 偶数ならば2で割る,
  奇数ならば3倍して1加える
- collatz(n) = 1になるまでの変化の回数

$$\mathbf{collatz}(n) = \begin{cases} \mathbf{0} & \mathbf{(???)} \\ \mathbf{1 + collatz(tnpo}(n)\mathbf{)} & \mathbf{(otherwise)} \end{cases}$$

補足: 補助関数が必要な再帰関数

- 例:「nの約数の個数」 f(n)とする
  - f(n)とf(n−1)の間に関係はなさそう
  - f(n)とf(n−k)であっても関係なさそう
  - 「nの約数の個数」=「1…nにあるnの約数」
=「x…nにあるnの約数(x=1の場合)」
- 補助関数:「x以上のnの約数の個数」g(x,n)
  - g(x,n)をg(x+1,n)で表わすことができる!
  - g(x,n)が定義できれば f(n)  g(1,n)

補足: 補助関数が必要な再帰関数

- 例:「nが素数かを判定する関数」 p(n)
  - p(n)とp(n−1)の間に関係はなさそう
  - 「nが素数か」=「2…(n−1)にnの約数はあるか」
    =「x…(n−1)にnの約数はあるか(x=2のとき)」
- 補助関数:「x以上のnの約数があるか」
  q(x,n)
  - q(x,n)をq(x+1,n)で表わすことができる!
  - q(x,n)が定義できれば p(n)  q(2,n)

少し練習

- 練習4.8cのcollatz(n)を定義せよ
  - ヒント: f(n)がf(g(n))で定義される場合
- 完成したら collatz(学生証番号) を求めその結果を投票せよ
- 時間が余る場合は
  - 課題をやる (CFIVE教材)
  - max_collatz(n)=「nを1まで変化させたときに現われる数の最大値」を定義

ターミナルで
cd Downloads
ruby c.rb 自分のCollatz数

4.1 繰り返し命令による反復計算 (pp.51−)

1～10までの和を求める

- 箱の中を0にしておき
- i=1,2,3,4,5,6,7,8,9,10について
  箱の数をiだけ増やす
- 箱の値 = 1～10までの和

4.1 繰り返し命令による反復計算 (pp.51–)

1～10までの和を求める

- 箱の中を0にしておき
- i=1,2,3,4,5,6,7,8,9,10について
  箱の数をiだけ増やす
- 箱の値 = 1～10までの和

```
def sum_loop(n)
    s = 0
    for i in 1..n
        s = s + i
    end
    s
end
```

条件を満たしている間の繰り返し (p.53)

kの最大の約数を求める

- nをk−1にしておき
- nがkの約数でない間
  n を 1 減らす
- nはkの最大の約数

```
def gd_loop(k)
    n = k-1
    while !divisible(k,n)
        n = n-1
    end
    n
end
```

練習: Collatz予想を繰り返しで解く (p.67)

- 「偶数ならば2で割る，奇数ならば3倍して1加える」
- 問題: collatz_loop(n) 1になるまでの変化の回数
- 求めたい値: 回数(cとする)
- 変化させる値: 現在のn

```
def collatz_loop(n)
    c=0
    while ここの条件は?
        n = tnpo(n)
        c = c + 1
    end
    ここは?
end
```

最初の回数はゼロ

p.37参照
回数を増やす

完成したら:
collatz(n)と
collatz_loop(n)の答が
一致することを確めよ

4.3.1 配列の作成と操作（pp.60–）

配列とは何か

- データの列(一次元)
- データの表(二次元)
- 三次元、四次元…

```
> make1d(3)
[0, 0, 0]
> make2d(2,3)
[ [0, 0, 0],
  [0, 0, 0]]
```

与えられた大きさの 1 次元配列を作る

```
def make1d(n)
  a = Array.new(n)
  for i in 0..(n-1)
    a[i] = 0
  end
  a
end
```

大きさがnで
中身が「空」の
1次元配列を作る
Rubyの式

中身を全て「0」にする

作った配列を返す
(使う側:
x = make1d(10))

2 次元配列を作る

```
load("./make1d.rb")

def make2d(h,w)
  a = Array.new(h)
  for i in 0..(h-1)
    a[i] = make1d(w)
  end
  a
end
```

大きさがhで
中身が「空」の
1次元配列を作る

中身を全て
「大きさwの配列」
にする

注意: 配列のある場所

```
load("./make1d.rb")

def make2d(h,w)
  a = Array.new(h)
  for i in 0..(h-1)
    a[i] = make1d(w)
  end
  a
end
```



```
load("./make1d.rb")

def make2dNG(h,w)
  a = Array.new(h)
  b = make1d(w)
  for i in 0..(h-1)
    a[i] = b
  end
  a
end
```

?

注意: 配列のある場所

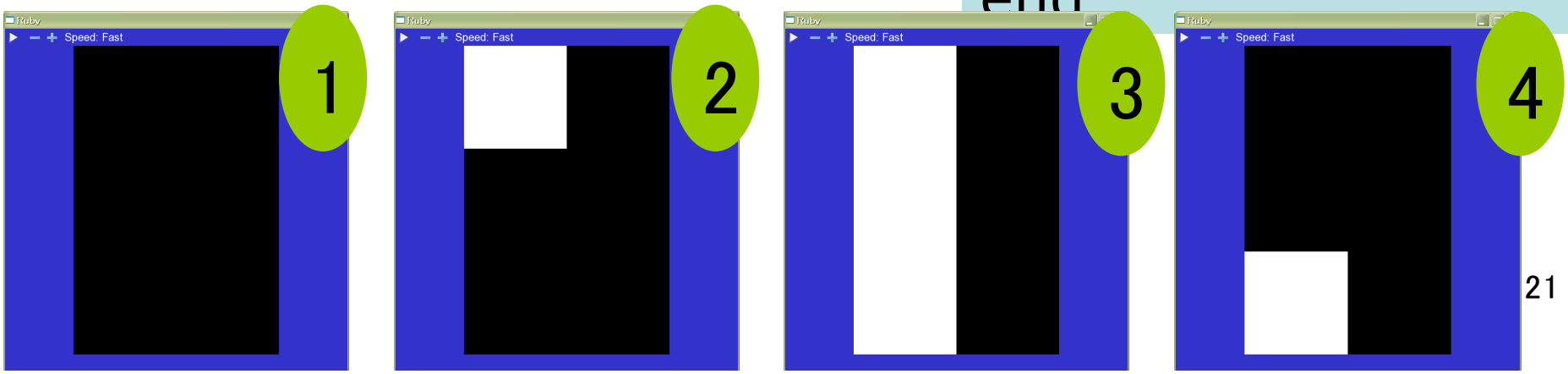
```
> a = make2dNG(3,2)
=> [[0, 0], [0, 0], [0, 0]]
> show(a)
=> nil
> a[0][0]=1
=> 1
> show(a)
```

ターミナルで

```
cd Downloads
ruby c.rb 番号
```

```
load("./make1d.rb")

def make2dNG(h,w)
  a = Array.new(h)
  b = make1d(w)
  for i in 0..(h-1)
    a[i] = b
  end
  a
end
```



3.3 文字列 (pp.39–)

```
irb(main):003:0> s = "abra"
=> "abra"
irb(main):004:0> t = "cadabra"
=> "cadabra"

irb(main):006:0> u = s + t
=> "abracadabra"
irb(main):007:0> "123" + "456"
=> "123456"
```

```
def l(name)
  load("./"+name+".rb")
end

irb(main):001:0> l("max")
```

s = "abra"
t = "cadabra"

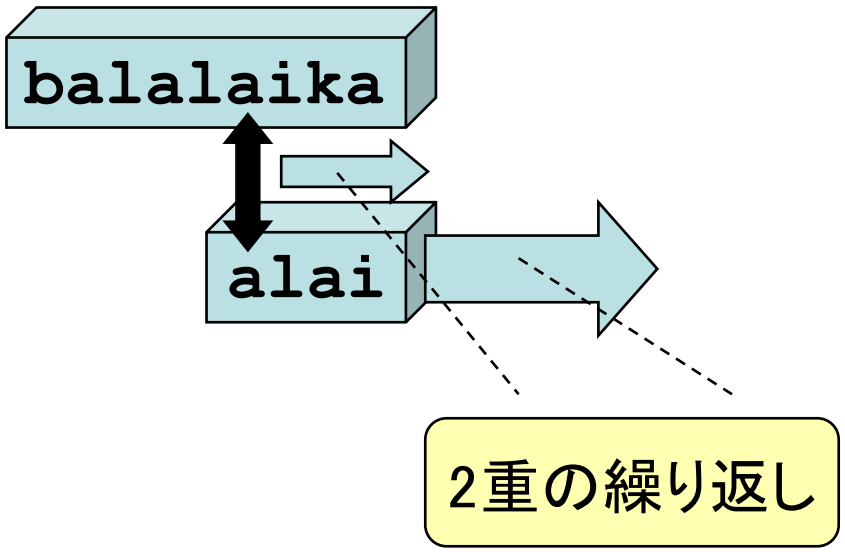
```
irb(main):009:0> s.length()
=> 4
irb(main):010:0> (s + t).length()
=> 11

irb(main):012:0> s[0..0]
=> "a"
irb(main):013:0> s[1..2]
=> "br"
irb(main):014:0> t[1..(t.length()-1)]
=> "adabra"
```

4.3.3 繰り返しの例: 言葉探し (pp.63–)

- 例: 配列sの中で配列pと一致する部分は何番目?

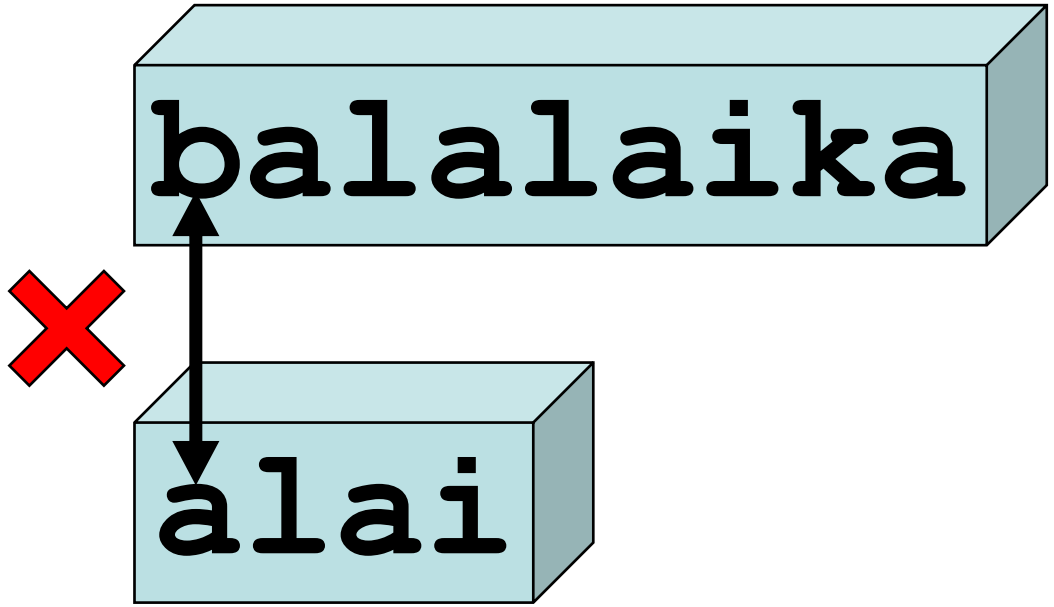
match("balalaika", "alai") → 3



繰り返しの例: 文字列の検索

- 例: 配列sの中で配列pと一致する部分は何番目?

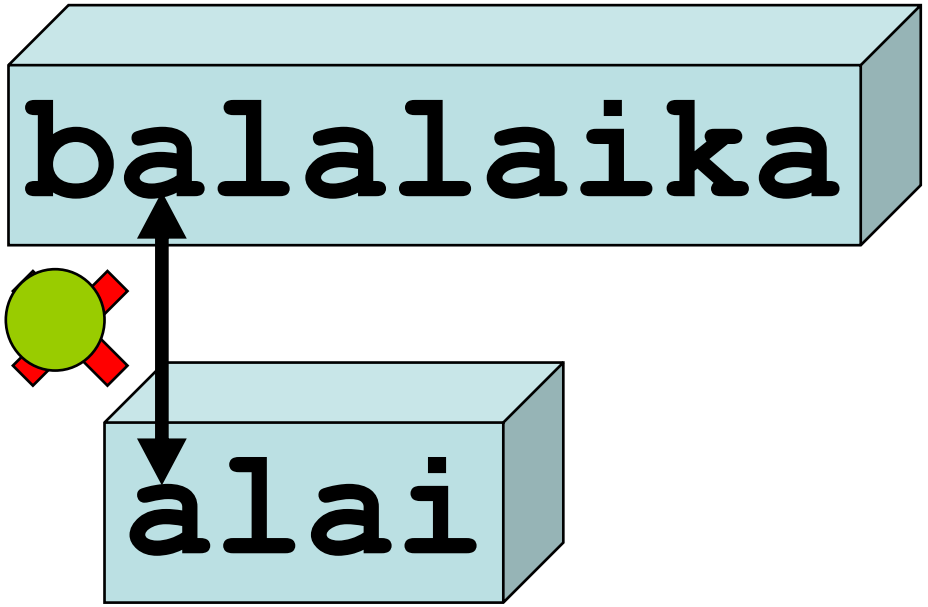
match("balalaika", "alai") → 3



繰り返しの例: 文字列の検索

- 例: 配列sの中で配列pと一致する部分は何番目?

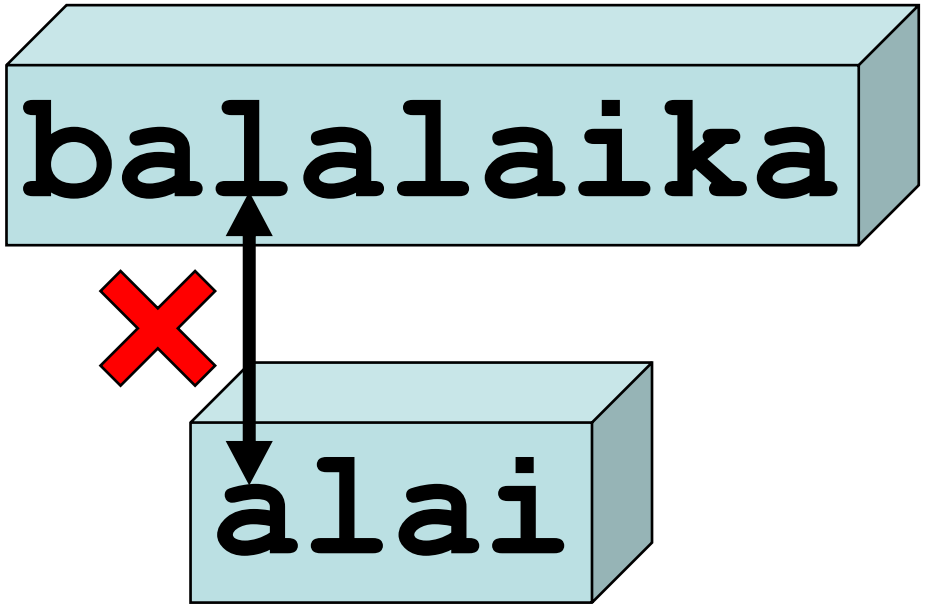
match("balalaika", "alai") → 3



繰り返しの例: 文字列の検索

- 例: 配列sの中で配列pと一致する部分は何番目?

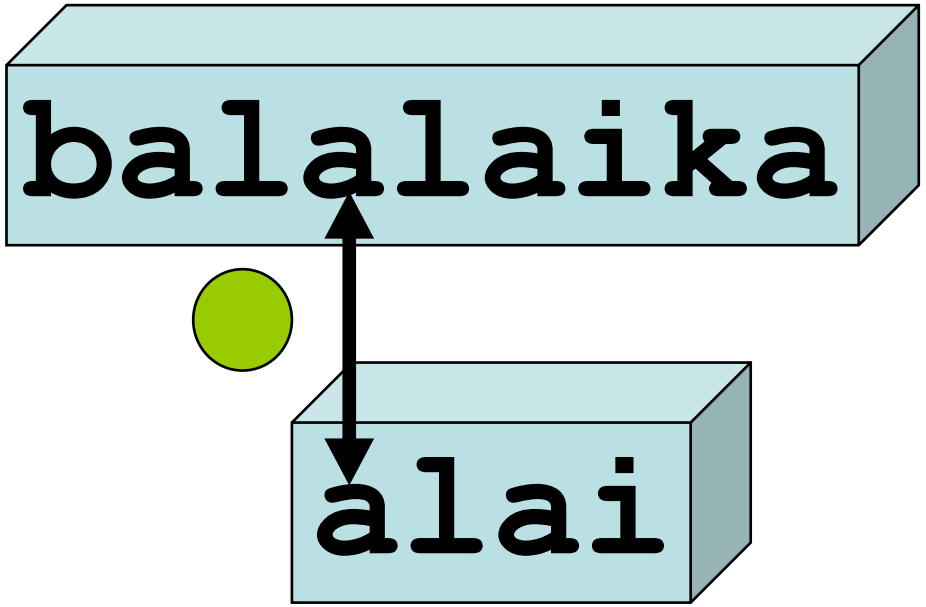
match("balalaika", "alai") → 3



繰り返しの例: 文字列の検索

- 例: 配列sの中で配列pと一致する部分は何番目?

match("balalaika", "alai") → 3



繰り返しの例: 文字列の検索

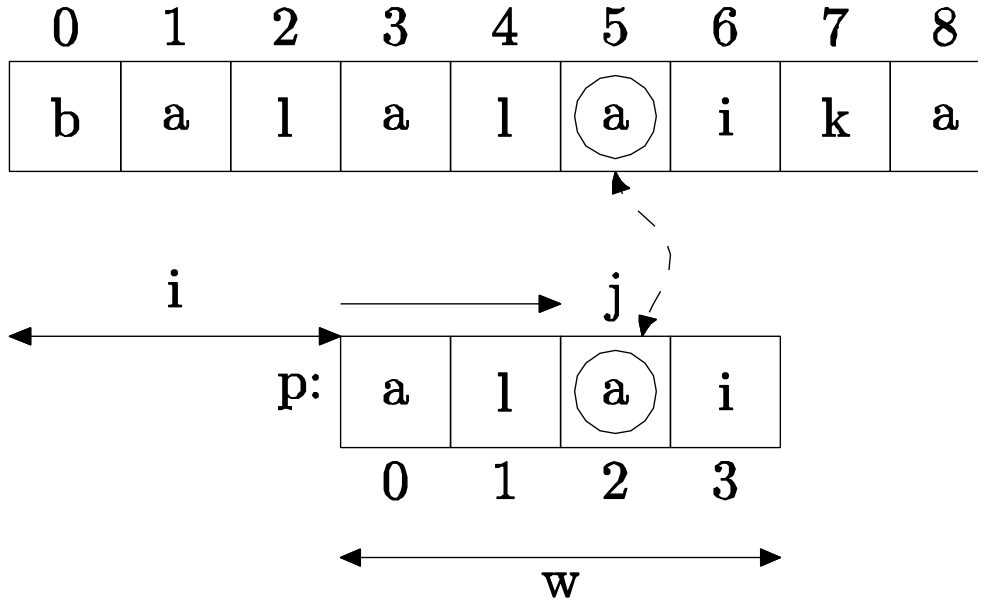
- 例: 配列sの中で配列pと一致する部分は何番目?

sのi番目からとpの一致数

```
def submatch(s,i,p,w)
  j = 0
  while j < w && s[(i+j)..(i+j)] == p[j..j]
    j = j + 1
  end
  j
end

def match(s,p)
  i=0
  w=p.length()
  while submatch(s,i,p,w) < w
    i = i + 1
  end
  i
end
```

pはsの何文字目から出現するか?




```
match("balalaika",
      "alai")
```

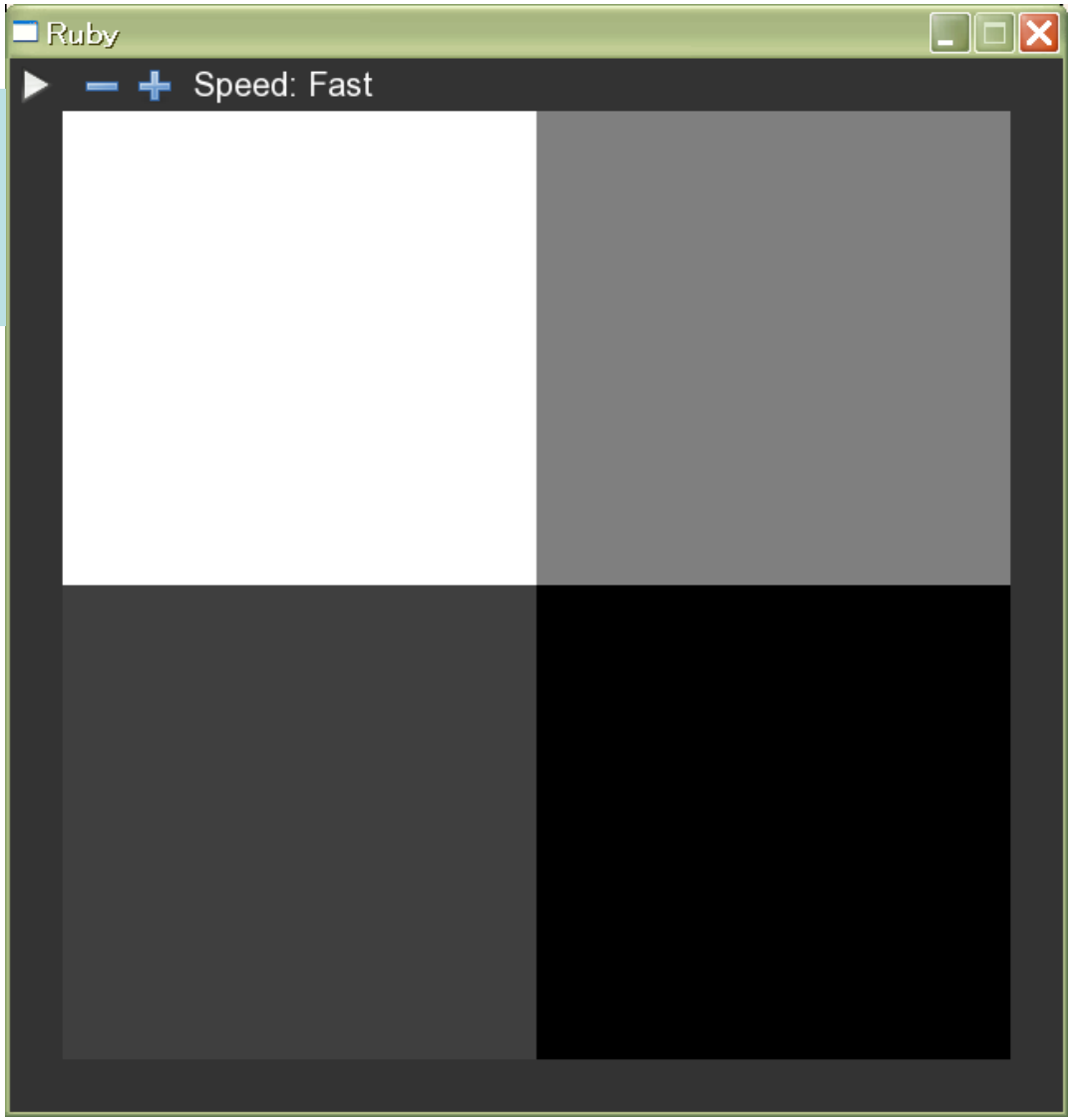
配列の実際

(isrbで)

```
> show([[1,0.5],
[0.25,0]])
```

とやってみよう

- 画像をクリックしてみよう



配列の実際

(isrbで)

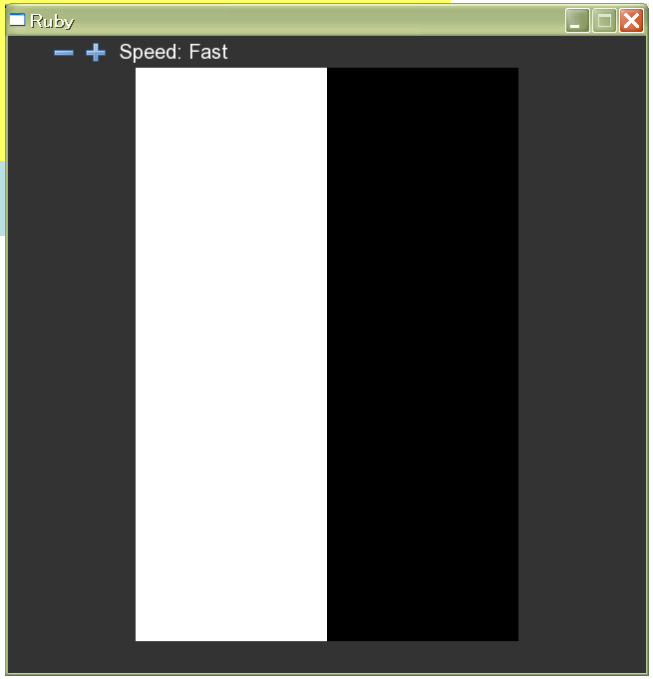
```
> show([[1,0.5],[0.25,0]])
```

とやってみよう

- 画像をクリックしてみよう

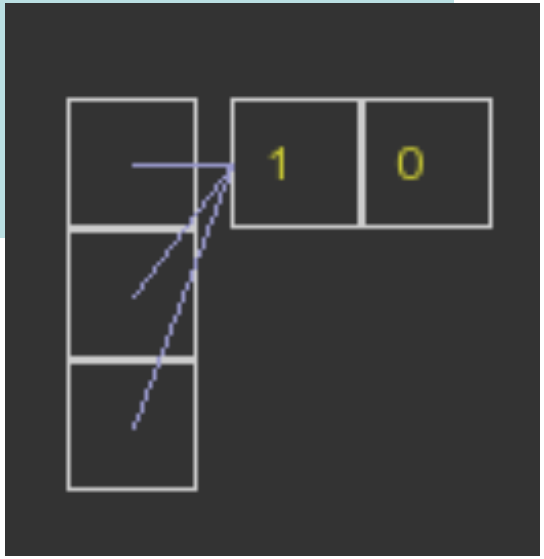
注意: 配列のある場所

```
> a = make2d(3,2)
=> [[0, 0], [0, 0], [0, 0]]
> show(a)
=> nil
> a[0][0]=1
=> 1
> show(a)
=> nil
 end
```




```
load("./make1d.rb")

def make2d(h,w)
  a = Array.new(h)
  b = make1d(w)
  for i in 0..(h-1)
    a[i] = b
  end
  a
end
```



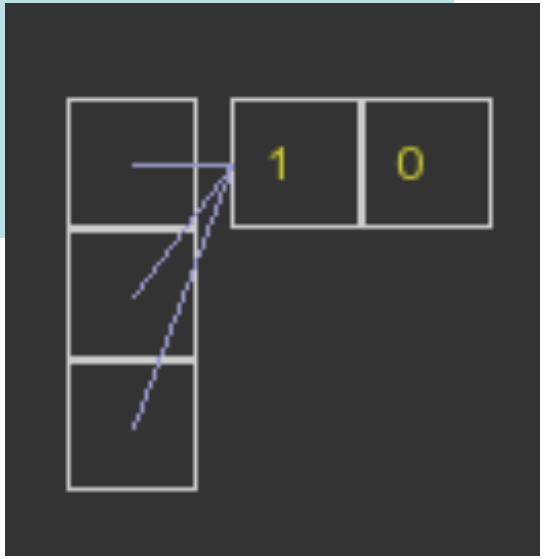
注意: 配列のある場所

```
load("./make1d.rb")

def make2d(h,w)
  a = Array.new(h)
  for i in 0..(h-1)
    a[i] = make1d(w)
  end
  a
end
```

```
load("./make1d.rb")

def make2d(h,w)
  a = Array.new(h)
  b = make1d(w)
  for i in 0..(h-1)
    a[i] = b
  end
  a
end
```



配列と再帰

- 大きさが$\underbrace{n \times n \times \cdots \times n}_{m}$であるm次元配列を作る

- makeMd(m,n) = { 1次元配列を作る (m=1)
  (m−1)次元配列を n個作って並べる (m>1)

```
def make2d(h,w)
  a = Array.new(h)
  for i in 0..(h-1)
    a[i] = make1d(w)
  end
  a
end     (参考:2次元配列)
```

配列と再帰

- 大きさが $\underbrace{n \times n \times \cdots \times n}_{m}$ であるm次元配列を作る
- makeMd(m,n) =
  - 1次元配列を作る （m=1）
  - (m−1)次元配列をn個作って並べる （m>1）

```
def makeMd(m,n)
  if m==1
    make1d(n)
  else
    a = Array.new(n)
    for i in 0..(n-1)
      a[i] = makeMd(m-1,n)
    end
    a
  end
end
```

```
def make2d(h,w)
  a = Array.new(h)
  for i in 0..(h-1)
    a[i] = make1d(w)
  end
  a
end
```

(参考:2次元配列)