効率：簡約計算モデル

胡　振江

簡約モデル

- 次のような定義を行ったとしよう。

```
pyth x y = sqr x + sqr y
sqr x = x * x
```

- 簡約例

pyth 3 4 ➔ sqr 3 + sqr 4
➔ (3*3) + sqr 4
➔ 9 + sqr 4
➔ 9 + (4*4)
➔ 9 + 16
➔ 25

簡約項

漸近的性質

- 効率の異なるプログラム例
  - プログラム1（リストを反転させる）：

```
reverse [] = []
reverse (x:xs) = reverse xs ++ [x]
[] ++ ys = ys
(x:xs) ++ ys = x : (xs++ys)
```

簡約ステップ数：リストの長さの二乗に比例する

プログラム2：

```
rev xs =  shunt [] xs
shunt ys [] = ys
shunt ys (x:xs) = shunt (x:ys) xs
```

簡約ステップ数：リストの長さに比例する

漸近的解析

- O-記法
  g(n) = O(h(n))
  ← どの正の数nについても
  |g(n)| <= M |h(n)|
  であるような定数Mが存在する
- 例
  T_reverse(x) = O(n^2)
  T_rev(x) = O(n)

- T_f(x): f xの計算に要する簡約ステップ数
- リストxの長さをnとする

練習問題

1. Init/lastの計算時間をO-記法を用いて与えよ.

```
init [x] = []
init (x:x':xs) = x : init (x':xs)

last [x] = x
last (x:x':xs) = last (x':xs)
```

2. g(n) = O(n^2) -O (n^2)ならば，g(n) = 0?

簡約順序

- 簡約段数は簡約項が簡約順序に依存

```
fst (x, y) = x
fst (sqr 4, sqr 2) → fst (4*4, sqr 2)
                   → fst (16, sqr 2)
                   → fst (16, 2*2)        最内簡約法
                   → fst (16, 4)
                   → 16
fst (sqr 4, sqr 2) → sqr 4
                   → 4*4                  最外簡約法
                   → 16
```

簡約の停止性

- 簡約順序によって簡約過程が停止しないことがある。

```
answer = fst (42, loop)
loop = tail loop
answer → fst (42, loop)                 最内簡約法
       → fst (42, tail loop)
       → fst (42. tail (tail loop))
       → …
answer → fst (42, loop)                 最外簡約法
       → 42
```

ただし，どちらの簡約法も停止する場合には同じ階が得られる.

簡約法

- 正規簡約法(normal order reduction)
  - 最外簡約法
  - 性質1：正規形を持つ項は必ず正規簡約法によって正規形に簡約することができる。
  - 性質2：解をもとめるために本質的に必要でなければ、簡約を行わない。
  - → 遅延評価
- 作用的簡約法(applicative order reduction)
  - 最内簡約法
  - → 先行評価 (f ⊥ = ⊥)

グラフ簡約計算モデル

- 最外簡約に要する簡約段数が最内簡約の段数を越えることはない? → ×

```
sqr (4+2)  → sqr 6
           → 6*6
           → 36
sqr (4+2)  → (4+2) * (4+2)
           → 6 * (4+2)
           → 6 * 6
           → 36
```

- グラフ簡約を用いると、… → ○

共有

グラフ簡約法

```
sqr (4+2)  →  ( . x . ) (4+2)

           →  ( . x . ) 6

           →  36
```

共有部分項の指定: where

```
roots a b c = ((-b-d) / e, (-b+d) / e)
    where d = sqrt (sqr b - 4 * a * c))
          e = 2 * a
```

項 roots 1 5 3の簡約:

```
((-5- . ) / . , (-b+ . )/ . )   sqrt (sqr 5 - 4*1*3)   2*1
```

頭部正規形

- ときには、式の全体を正規形するのではなく、ある部分項だけを簡約してよい.

```
head (map sqr [1..7])
    ➔ head (map sqr (1:[2..7]))
    ➔ head (sqr 1 : map sqr [2..7])
    ➔ sqr 1
    ➔ 1*1
    ➔ 1
```

- 定義：簡約項でなく、その部分項のどれを簡約しても簡約項にはならない項は頭部正規形
  - 例： e1 : e2, (e1, e2)

簡約順序と所要領域

```
sum = foldl (+) 0
    sum [1..1000]
        ➔ foldl (+) 0 [1..1000]
        ➔ foldl (+) (0+1) [2..1000]
        ➔ foldl (+) ((0+1)+2) [3..1000]
        ➔ …
最外簡約 ➔ foldl (+) (… ((0+1)+2)+…+1000) []        O(n)領域
        ➔ (… ((0+1)+2)+…+1000)
        ➔ …
        ➔ 500500
    sum [1..1000]
        ➔ foldl (+) 0 [1..1000]
最内簡約 ➔ foldl (+) (0+1) [2..1000]
        ➔ foldl (+) 1 [2..1000]                      O(1)領域
        ➔ …
        ➔ 500500
```

簡約順序の制御

- 計算モデル：最外簡約
- strictを用いて簡約順序を制御する
  - strict f eの簡約
    - まずはじめにeを頭部正規形に簡約する
    - 次にfを適用する

```
strict sqr (4+2)
    ➔ sqr 6
    ➔ 6*6
    ➔ 36
```

strict f x = seq x (f x)

- 例

```
foldl' f a [] = a
foldl' f a (x:xs) = strict (foldl' f) (f a x) xs
```

```
sum [1..1000]
    ➔ foldl' (+) 0 [1..1000]
    ➔ strict (foldl' (+)) (0+1) [2..1000]
    ➔ foldl' (+) 1 [2..1000]
    ➔ …
```

分割統治法

- 効率のよいアルゴリズムの設計の有用な方法の一つ

問題Pを幾つかの部分問題（それぞれはPと同類の問題であるが、入力の大きさが小さいもの）に分割し、部分問題の解を集めてもとの問題の解とする手法

整列：挿入整列法

- リストを第一要素と残りの部分に分割して処理を行う。

3 | 1 4 1 5 9 6 2 5 |

```
isort = foldr insert []
insert x xs = takeWhile (<=x) xs ++ [x] ++
                dropWhile (<=x) xs
```

T_insert(n) = O(n)
T_isort(n) = T_insert(0) + T_insert(1) + ... + T_insert(n)
= O(n^2)

insertの効率化

```
insert x xs = takeWhile (<=x) xs ++ [x] ++
              dropWhile (<=x) xs
```

⇩ xsが整列されている

```
insert x [] = [x]
Insert x (y:ys) | x <= y = x : y : ys
                | otherwise =  y : insert x ys
```

宿題：　導出（合成）過程をしめせ.

併合整列法

- リストをほぼ同じ大きさの2つの部分に分割し、それぞれの部分を整列したあとで併合する。

```
msort xs | n <= 1    = xs
         | otherwise = merge (msort us) (msort vs)
   where n = length xs
         us = take (n `div` 2) xs
         vs = drop (n `div` 2) xs

merge [] ys = ys
merge (x:xs) [] = x:xs
merge (x:xs) (y:ys) | x<=y      = x : merge xs (y:ys)
                    | otherwise = y : merge (x:xs) ys
```

T_sort(n)
= O(1), if n<=1
= 2T_sort(n/2)+T_merge(n), otherwise

T_merge(n) = O(n)

クイック整列法

- 複雑な分割＋簡単な統合

```
qsort [] = []
qsort (x:xs) = qsort [u|u<-xs,u<x] ++
               [x] ++
               qsort [u|u<-xs,u>=x]
```

T_qsort(n) = O(n^2)（平均的に O(n log n)）

乗算

- n個の数字の列で表現される2つの正の整数x, yの乗算を考える.

```
x = x1 * 10^(n/2) + x0
y = y1 * 10^(n/2) + y0

x * y = z2 * 10^n + z1 * 10^(n/2) + z0
  where z2 = x1 * y1
        z1 = x1 * y0 + x0 * y1
        z0 = x0 * y0
```

nが2のベキであると仮定する.

乗算

- n個の数字の列で表現される2つの正の整数x, yの乗算を考える.

```
x = x1 * 10^(n/2) + x0
y = y1 * 10^(n/2) + y0

x * y = z2 * 10^n + z1 * 10^(n/2) + z0
  where z2 = x1 * y1
        z1 = (x1+x0) * (y1+y0) - z0 - z2
        z0 = x0 * y0
```

nが2のベキであると仮定する.

二分探索法

- 問題
  - 整数aと整数bと述語pが与えられたとき、区間[a..b]内でp xが成立するような最小のxを求める。
- 仕様

```
find p a b = min [ x | x <- [a..b], p x]
```

  - 正しい：問題の翻訳
  - 効率が悪い　← b-a+1回のpの計算が必要

効率化1

- [a..b]の単調性質の利用

find p a b = min [x | x <- [a..b], p x]
（pはb-a+1回評価される）
➔
find p a b = head [x | x <- [a..b], p x]
（pは1からb-a+1回評価される）

効率化2

- 述語pが単調の場合

p x = True ∧ x < y ならば p y = True

a m b

```
find p a b     | a==b && p a          = a
               | a<b && p m           = find  p a m
               | a<b && not (p m)     = find p (m+1) b
          where
               m = (a+b) `div` 2
```

T(n) = O(log n)

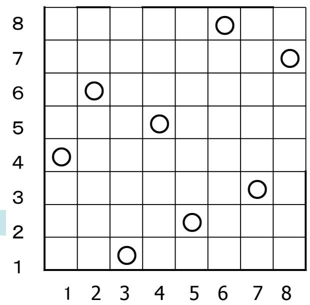
探索と数え上げ

- 組合せ論的問題：ある性質を満たす対象の組合せを探す。
- 手法：
  標準的な手法：逆戻り探索法
  （成功リストによって実現する手法を紹介する）
- 例題：
  – Eight-queen 問題
  – Instant insanity 問題

エイトクイーン問題

- チェス盤と8個のクイーンが与えられたとき、どの2つのクイーンも互いに効き筋にはならないように盤面に置く。

盤面：[4,6,1,5,2,8,3,7]

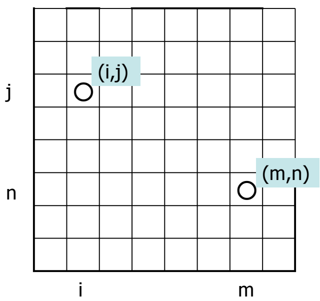


じの列に二つ以上のクイーンがない。

安全性チェック

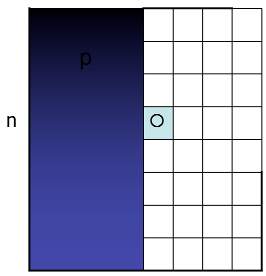
- Check: 座標(i,j)と(m,n)にある2つのクイーンはそれらが同一の行、あるいは2つの対角線上のいずれかにあるか

```
check (i,j) (m,n)
   = j == n  ||
     i+j == m+n ||
     i-j == m-j
```



盤面にクイーンの追加

```
safe p n
   = and [ not (check (i,j) (m,n))
          | (i,j) <- zip [1..length p] p ]
     where m = length p + 1
```



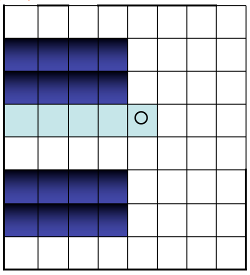
p′ = p ++ [n]

メーンプログラム

後戻しのため、最初のm個のクイーンの
さまざまな方法を考える必要がある。

```
queens 0 = [[]]
queens (m+1)
  = [ p ++ [n]
     | p <- queens m,
       n <- [1..8],
       safe p n]

queens1 m = head (queens m)
```



p′ = p ++ [n]

Crypt-arithmetic puzzle

10個までの文字を用いた3個の単語を次のように配置するような問題を解くプログラムを書け。

```
    F O U R
+   F I V E
-------------------
    N I N E
```

(解の例:2970+2483=5453)

```
puzzle = [ (e,f,i,n,o,r,u,v)
           | e<-[0..9],
             f<-[1..9],
             …,
             v<-[0..9],
             vplus [f,o,u,r] [f,i,v,e] == [n,i,n,e],
             allDifferent [e,f,I,n,0,r,u,v] ]
where allDifferent xs = rmdup (sort xs) == sort xs
```

期末試験

- 日時: 2月6日
- 場所: 63号室
- 教科書、ノートを持ち込み可
- 内容:教科書1-7章

来週はアンケートを取ります.
ご協力をよろしくお願いします.