

行列演算

```

let rec list_transpose ll =
  match ll with
  | [] -> [] (* 行が残らないと長さが分からない *)
  | [l] -> List.map (fun x -> [x]) l
  | l :: ll ->
      let ll = list_transpose ll in
      List.map2 (fun x l -> x :: l) l ll
val list_transpose : 'a list list -> 'a list list = <fun>
# list_transpose [[1;2;3]; [4;5;6]];;
- : int list list = [[1; 4]; [2; 5]; [3; 6]]

```

```

let array_transpose mn =
  let m = Array.length mn in
  let n = Array.length mn.(0) in
  Array.init n (fun i -> Array.init m (fun j -> mn.(j).(i)))
val array_transpose : 'a array array -> 'a array array = <fun>
# array_transpose [| [1;2;3] |; [4;5;6] | ];;
- : int array array = [| [1; 4] |; [2; 5] |; [3; 6] | ]

```

```

let rec sum l =
  match l with
  | [] -> 0.
  | a :: l -> a +. sum l
val sum : float list -> float = <fun>

```

```

let rec list_prod ll1 ll2 =
  let ll2 = list_transpose ll2 in
  List.map (fun l1 ->
    List.map (fun l2 -> sum (List.map2 ( *. ) l1 l2)) ll2)
    ll1
val list_prod : float list list -> float list list -> float list list

```

```

let array_prod mn np =
  let m = Array.length mn in
  let n = Array.length np in
  let p = Array.length np.(0) in
  Array.init m (fun i -> Array.init p (fun j ->
    let r = ref 0. in
    Array.iteri (fun k b -> r := !r +. mn.(i).(k) *. b.(j)) np;
    !r))
val array_prod : float array array -> float array array -> float array array

```

4 整列

ある集合を順序によって整列する .

挿入による整列

既に整列された集合に新しい要素を正しい所に入れば , 整列が保たれる .

```
let rec insert a l = (* 挿入する関数 *)
  match l with
  | [] -> [a]
  | b :: l' ->
    if b < a then b :: insert a l' else a :: l
val insert : 'a -> 'a list -> 'a list = <fun>
```

```
let rec insertion_sort l = (* 後から一個ずつ挿入 *)
  match l with
  | [] -> []
  | a :: l -> insert a (insertion_sort l)
val insert : 'a -> 'a list -> 'a list = <fun>
```

選択による整列

```
let selection_sort arr =
  let len = Array.length arr in
  for i = 0 to len - 2 do
    let arri = arr.(i) in
    let p = ref i in (* 最小値の位置 *)
    for j = i+1 to len - 1 do
      if arr.(j) < arr.(i) then begin
        arr.(i) <- arr.(j);
        p := j;
      end
    done;
    arr.(!p) <- arri; (* 最小値のあった所に入れる *)
  done
val selection_sort : 'a array -> unit = <fun>
```

アルゴリズムの計算量

測る対象 まず何の量に注目するかを決める

- 結果を計算するのにどれだけ時間がかかるか
- 結果を計算するのにどれだけメモリが必要か

基本演算 時間の大部分を占めるプログラムの中の演算を探す . 時間がそれに比例するだけでいいので , 比較か代入が便利 . 演算ごとに呼び出しがある場合 , 関数呼び出しの数でもいい .

最悪か平均 安定したアルゴリズムを求めていれば、最悪の入力の場合どれだけ時間とメモリが要るかを知らない。実際の効率が問題であれば、平均的な計算量に興味がある。当然ながら後者が難しい。

入力の大きさ 計算量は入力に関数であるが、異なる入力に対する動作を比較するために、入力に大きさの概念がなければならない。「平均」は同じ大きさの平均的な分布の入力を指すことになる。

計算量の正確さ 基本演算を元に、計算量を計算しつくすこともできるが、通常知りたいのは入力が大きくなったときの計算量の概算 $A(n)$ 。これを漸近的な計算量と言ひ、実際の計算量が $E(n)$ ならば、ある C が存在し、 $E(n) = C \cdot A(n) + o(A(n))$ であるとき、 $E(n) = O(A(n))$ と書く。

計算量の例

挿入整列の最悪な入力 関数呼び出しを数える。入力の大きさはリストの長さを n とする。

`insertion_sort` は入力の長さにかよわないので、違いは `insert` から来る。a が 1 の全ての要素より大きければ、リストの最後まで見なければならぬ。

すなわち、入力が逆順になっていれば、毎回 `insert` が最後までリストを見ないといけない。

この場合の計算量は

$$\sum_{k=1}^n 1+k = \frac{(n+1)(n+2)}{2} = O(n^2)$$

選択整列の計算量 比較の回数を数える。入力の大きさは配列の長さを n とする。

最悪と平均が同じで、

$$\sum_{i=0}^{n-2} (n-i-1) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = O(n^2)$$

挿入整列の平均の計算量 平坦な分布を考える。ここで重要なのは入力の中の要素の順序なので、値の重複がなければ $n!$ とおりある。

別の見方をすれば、長さ k のリストに新しい要素を挿入したとき、最終的な位置は $k+1$ とおりあって、その確立は皆同率である。

平均的な計算量は

$$\sum_{k=0}^{n-1} 1 + \sum_{i=1}^{k+1} \frac{i}{k+1} = \sum_{k=0}^{n-1} 1 + \frac{k+2}{2} = n + \frac{(n+1)(n+2)}{4} = O(n^2)$$

分割による整列

```
let rec partition pivot l =
  match l with
  | [] -> ([], [])
  | a :: l ->
    let (l1,l2) = partition pivot l in
    if a < pivot then (a::l1, l2) else (l1, a::l2)
val partition : 'a -> 'a list -> 'a list * 'a list = <fun>
```

```

let rec quicksort l =
  match l with
  | [] -> []
  | a :: l ->
    let (l1, l2) = partition a l in
    quicksort l1 @ a :: quicksort l2
val quicksort : 'a list -> 'a list = <fun>

```

最悪の計算量 リストが既に整列されていれば, $l1$ は常に空リストになる.

$$\sum_{k=1}^n 2 + k = 2 + \frac{n(n+1)}{2} = O(n^2)$$

平均の計算量 $l1$ と $l2$ の長さの分布が平坦と考えていい.

$$\begin{aligned}
A(n+1) &= 1 + n + \sum_{k=0}^n \frac{A(k) + A(n-k)}{n+1} = 1 + n + \sum_{k=0}^n \frac{2}{n+1} A(k) \\
&= 1 + n + \frac{n}{n+1} A(n) - \frac{n(1+n)}{n+1} + \frac{2}{n+1} A(n) = 1 + \frac{n+2}{n+1} A(n) \\
\frac{A(n+1)}{n+2} &= \frac{1}{n+2} + \frac{A(n)}{n+1} & \frac{A(n)}{n+1} &= \sum_{k=1}^{n-1} \frac{1}{k+1} = O(\log n) \\
A(n) &= O(n \cdot \log n)
\end{aligned}$$

実習課題 (4回目)

1. `Random.int : int -> int` が乱数を生成する. `Random.int n` は $[0 \dots n-1]$ の整数になる. それを使って長さ m の整数のリストや配列を作る関数を定義せよ.
2. `quicksort` を前述の定義のように定義する. ただし, 以下の変化を加える.

```

let counter = ref 0 ;;
let rec partition pivot l =
  incr counter;
  match l with ...

```

それを使えば, `partition` が何回呼ばれたかが測れる. (1) の関数を使って乱数のリストを作り, `quicksort` の計算量の変化を実際に見よ.

3. (2) の操作の流れを関数で行い, 実験的に n に対する計算量の平均値を求めよ.
4. (3) で得られる平均値が $A(n)$ の具体的な値だとすると, $A(n) = C \cdot n \log n + o(n \log n)$ となるはずである. C を実験で求めよ. 収束は早いですか?
5. `insertion_sort` についても同じことをせよ.

来週までに感想や課題のできた分だけを `computer-lecture-2005-aw-4@math.nagoya-u.ac.jp`宛に送って下さい.