アルゴリズム論

2010年度

三つの問題を Python で解く[1]

行列の積

- 行列の積
  - 入力：$n \times n$ の実数値行列
    $\boldsymbol{A} = A[i, j]$ および $\boldsymbol{B} = B[i, j]$
  - 出力：$\boldsymbol{C} = C[i, j] = \boldsymbol{A} \cdot \boldsymbol{B}$

ソーティング

- ソーティング
  - 入力：n 個の実数列 $A = (a_1, a_2, ..., a_n)$
  - 出力：これらの数を非減少順に並べ替えた列

「非減少」＝「減少しない」＝「同じ or 増加」

「非減少順」：「小さいものから並べた順」

ascending order: 増加順　descending order: 減少順

最大と２番目

- 最大と２番目に大きい要素
  - 入力：n 個の正整数の集合 $A = \{a_1, a_2, ..., a_n\}$
  - 出力：最大の要素 $a_{\max}$ および２番目に大きい要素 $a_{\max 2}$

行列の積

- 行列の積
  - 入力：$n \times n$ の実数値行列

    $\boldsymbol{A} = A[i, j]$ および $\boldsymbol{B} = B[i, j]$
  - 出力：$\boldsymbol{C} = C[i, j] = \boldsymbol{A} \cdot \boldsymbol{B}$

行列の積 Python

```
def multiple(A, B):  # A と B は同じサイズの正方行列
    """ A と B の積を返す """
    size = len(A)    # len(A) はリスト A の長さ
    C = []           # C はリスト（初期値は空）と宣言
    for i in range(size):   # range(size) は [0,1,...,size-1]
        row = []     # row はリスト。C の i 行目を作る準備
        for j in range(size):  # j = 0, 1,..., size-1
            val = 0  # C の i 行 j 列目の計算準備
            for k in range(size):
                val += A[i][k] * B[k][j]
            row.append(val)   # リスト row の最後に val を追加
        C.append(row)         # リスト C の最後に row を追加
    return C
```

実行例：２次正方行列

$$\begin{bmatrix} 1 & 2 \\ -3 & 0 \end{bmatrix}\begin{bmatrix} -2 & 1 \\ 3 & -1 \end{bmatrix}=\begin{bmatrix} 4 & -1 \\ 6 & -3 \end{bmatrix}$$

```
>>> A = [[1,2],[-3,0]]  # リスト A は２行２列
>>> print A             # A を出力せよ
[[1, 2], [-3, 0]]
>>> B = [[-2,1],[3,-1]]
>>> pro = multiple(A, B)  # A と B の積をリスト pro へ代入
>>> pro                   # pro を出力せよ
[[4, -1], [6, -3]]
>>>
```

実行例：３次正方行列

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & -2 & 0 \\ 2 & 0 & -1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 3 \\ -1 & 2 & 2 \\ 3 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 6 & 7 & 7 \\ -2 & -4 & 8 \\ -5 & -1 & 6 \end{bmatrix}$$

```
>>> A = [[1,2,3],[4,-2,0],[2,0,-1]]  # リスト A は３行３列
>>> B = [[-1,0,3],[-1,2,2],[3,1,0]]
>>> pro = multiple(A, B)
>>> pro
[[6, 7, 7], [-2, -4, 8], [-5, -1, 6]]
>>>
```

計算時間の見積もり

- n 行 n 列の正方行列同士の積を考える
- 計算時間は，問題のサイズ（n の値）の関数になる
- $n = 100$ のとき，計算時間が約 1 秒であったとき，$n = 1000$ ならその計算時間は？

行列の積

```
def multiple(A, B):
    size = len(A)
    C = []                          ⇐あわせて a
    for i in range(size):
        row = []                    ⇐C.append(row) とあわせて b
        for j in range(size):
            val = 0                 ⇐row.append(val) とあわせて c
            for k in range(size):
                val += A[i][k] * B[k][j]    ⇐ d
            row.append(val)
        C.append(row)
    return C
```

$$T(n) = a + n(b + n(c + nd)) = dn^3 + cn^2 + bn + a$$

計算時間の見積もり

$$T(n) = dn^3 + cn^2 + bn + a$$

- $n = 100$ のとき，計算時間が約１秒であったとき，$n = 1000$ ならその計算時間は？

  $T(n) \approx dn^3$ と考えたらよい

  $$T(100) = d \times 100^3 = 1\,[\mathrm{sec}] \Rightarrow d = 10^{-6}\,[\mathrm{sec}]$$
  $$T(1000) = 10^{-6} \times 1000^3 = 10^3\,[\mathrm{sec}] \approx 17\,[\mathrm{min}]$$

- 計算時間はサイズの３乗に比例するので，サイズが 10倍なら時間は $10^3 = 1000$倍

計算時間の測定

```
import datetime   # モジュールをインポート

def multipleT(a, b, times):
    """ multiple を times 回実行する """
    start = datetime.datetime.now()
    for i in range(times):
        ans = multiple(a, b)
    print datetime.datetime.now() - start
    return ans
```

正方行列でなくても

```
>>> A = [[1,2,3],[4,-2,0]]
>>> B = [[-1,0],[-1,2],[1,0]]
>>> print multipleAdv(A, B)
[[0, 4], [-2, -4]]
>>>
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & -2 & 0 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ -1 & 2 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 4 \\ -2 & -4 \end{bmatrix}$$

課題 p1：行列の積

行列 A の列数と行列 B の行数が同じであるとき，その積 AB を計算するプログラムを書け。また，その計算時間を行列のサイズの関数として表せ。

ソーティング

- ソーティング
  - 入力：n 個の実数列 $A = (a_1, a_2, ..., a_n)$
  - 出力：これらの数を非減少順に並べ替えた列

バブルソート

```
def bubbleSort(A):
    for i in range(len(A) - 1):
        for j in range(len(A) - 1, i, -1):
            if A[j - 1] > A[j]:
                A[j - 1], A[j] = (A[j], A[j - 1])
    return
```

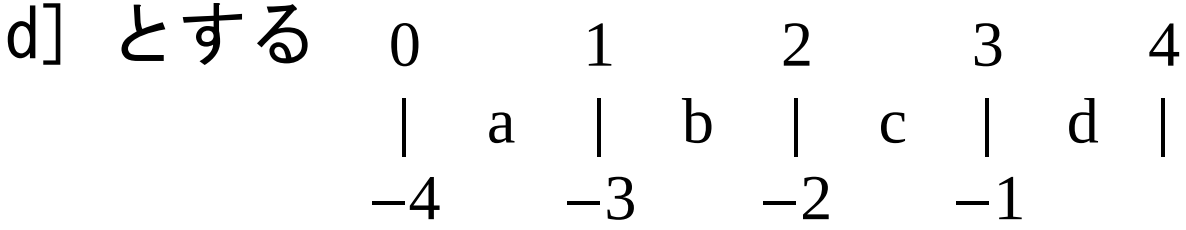
len(A) が 5 のとき：
  range(len(A)) は, リスト [0, 1, 2, 3, 4]
  range(len(A) - 1) は [0, 1, 2, 3] ← i の範囲
  range(len(A) - 1, 0, -1) は [4, 3, 2, 1] ← i = 0 の j の範囲
  range(len(A) - 1, 1, -1) は [4, 3, 2] ← i = 1 の j の範囲
  range(len(A) - 1, 3, -1) は [4] ← i = 3 の j の範囲

組み込み関数 range()

- range(([開始,]終了[,ステップ])
  - 「開始」が省略されたら 0
  - 「ステップ」が省略されたら 1
  - 引数が２個のときは range(開始, 終了)
- 「開始」から始まり，「終了」直前の整数までの要素をもつリストを返す
- ステップが 1 なら，range(s, t) の要素は t－s 個

スライス表記（slicing）

- x = [a, b, c, d] とする



- x[1:3], x[−3:−1] はいずれも [b, c]
- x[:3] は x[0:3] と同じで [a, b, c]
- x[−1] は x[3] や x[3:4] と同じで，要素 d のこと。x の長さに関係なく x[−1] は最後の要素
- x の最後の２要素 [c, d] は，x[−2:]
- x, x[0:4], x[:] はすべて [a, b, c, d]

バブルソート

```
def bubbleSort(A):
    for i in range(len(A) - 1):
        for j in range(len(A) - 1, i, -1):
            if A[j - 1] > A[j]:
                A[j - 1], A[j] = (A[j], A[j - 1])
    return
```

len(A) が 5 のとき：

```
  range(len(A)) は, リスト [0, 1, 2, 3, 4]
  range(len(A) - 1) は [0, 1, 2, 3] ← i の範囲
  range(len(A) - 1, 0, -1) は [4, 3, 2, 1] ← i = 0 の j の範囲
  range(len(A) - 1, 1, -1) は [4, 3, 2]    ← i = 1 の j の範囲
  range(len(A) - 1, 3, -1) は [4]          ← i = 3 の j の範囲
```

実は . . . Python

```
>>> A = [3,4,7,1,3,6,8,5,9]
>>> A.sort()
>>> A
[1, 3, 3, 4, 5, 6, 7, 8, 9]
```

- Python では，リストオブジェクトに用意されたメソッド sort() がある。

```
>>> A = [3,4,7,1,3,6,8,5,9]
>>> A.sort()
>>> A.reverse()    # 並び順を反転するメソッド
>>> A
[9, 8, 7, 6, 5, 4, 3, 3, 1]
```

最大と２番目

- 最大と２番目に大きい要素
  - 入力：n 個の正整数の集合 $A = \{a_1, a_2, ..., a_n\}$
  - 出力：最大の要素 $a_{\max}$ および２番目に大きい要素 $a_{\max 2}$

バブル＝泡

```
def bubbleSort(A):
    for i in range(len(A) - 1):
        for j in range(len(A) - 1, i, -1):
            if A[j - 1] > A[j]:
                A[j - 1], A[j] = (A[j], A[j - 1])
        print A
    return

>>> A = [4,3,2,1,5]
>>> bubbleSort(A)
[1, 4, 3, 2, 5]
[1, 2, 4, 3, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
```

最大と２番目

```
def bubble2(A):
    for i in [0, 1]:
        for j in range(len(A) - 1, i, -1):
            if A[j - 1] < A[j]:
                A[j - 1], A[j] = (A[j], A[j - 1])
    print "max =", A[0], ".. max2 =", A[1]
    return

>>> A = [4,3,2,1,5]
>>> bubble2(A)
max = 5 .. max2 = 4
```

課題 s1:バブルソートの計算時間

スワップ操作に要する時間を c とし，その他の演算時間は考えなくてよい。

(1) $T_{\mathrm{worst}}(n)$：与えられた数列が最悪の場合。
どのような数列が最悪かも答よ。

(2) $T_{\mathrm{average}}(n)$：数列がランダムであって，
if 文が成立する確率が 1/2 である場合。

(3) bubble2 の最悪計算時間：$T_{\mathrm{bubble2}}(n)$

実は . . . Python

```
>>> A = [3,4,7,1,3,6,8,5]
>>> max1 = max(A)    # max は組み込み関数
>>> max1
8
>>> A.remove(max1)   # remove はリストのメソッド
>>> A
[3, 4, 7, 1, 3, 6, 5]   # 要素 max1 が削除されている
>>> max2 = max(A)
>>> max2
7
>>> A.append(max1)   # 要素 max1 を追加
>>> A
[3, 4, 7, 1, 3, 6, 5, 8]   # 当初とは順序は違うが . . .
```