アルゴリズム論

2010年度

問題のクラス P と NP [1]

離散対数 [4]

離散対数問題：
生成元 g で生成される巡回群 G において，
$a \in G$ であるとき，$g^k = a$ となる k を求めよ

g^k の計算（N を法とする）にどんな工夫ができますか？

```
# normal version
ans = 1
for i in range(k):
    ans = (ans * g) % N
```

```
# binary version
ans = 1
while k != 0:
    if k % 2 == 1:
        ans = (ans * g) % N
    k /= 2
    g = (g * g) % N
```

$$g^5 = g^4 \times g^1$$
$$g^{10} = g^8 \times g^2$$
$$g^{11} = g^8 \times g^2 \times g^1$$

課題 b2：べき乗計算

- N を法とする正整数の乗算において，g の k 乗を計算する２種類の関数 powBinary(g, k, N) と powNormal(g, k, N) をまず作る。さらに，これらを times 回計算してその計算時間を表示する。
  [PC にできるだけ他の負荷をかけないで計測]
- さまざまな値に対して両者の計算時間を計測してその結果をグラフや表にまとめ，powBinary の優位性をアピールせよ。
  - g を固定して k を変えるとどうなるか。逆は？
  - 計算時間に N は影響しない？
  - powNormal の方が早い場合もあるだろう

多項式時間アルゴリズム

- 個別問題のサイズの多項式オーダーの時間計算量をもつアルゴリズム
  - 易しい問題には多項式時間アルゴリズムがある
  - 多項式は乗算や合成で閉じている
  - 計算機械のモデルに依存しない

- 多項式時間アルゴリズムで解くことのできる問題のクラス（集まり，集合）を **P** と表す
  → クラス **P** に属する問題は易しい問題

多項式の多項式は多項式

$f(n)$ および $g(n)$ が n の多項式なら

$$f(n)g(n),\ f(g(n)),\ g(f(n))\ \text{は多項式}$$

［例］ $f(n)=an^2+b,\ g(n)=n^3$ のとき

$$f(n)g(n)=n^3(an^2+b)=an^5+bn^3$$

$$f(g(n))=a(n^3)^2+b=an^6+b$$

$$g(f(n))=(an^2+b)^3=a^3n^6+3a^2bn^4+3ab^2n^2+b^3$$

判定問題と最適化問題

- 判定問題
  - “yes” か “no” かを問う問題
  - 例：このグラフは５色で彩色できるか？

- 最適化問題
  - 条件を満たす解の中で，目的の値が最小（あるいは最大）となる解［最適解］を求める
  - 例：このグラフを彩色する場合，最小で何色必要か？
    ［彩色数を求めよ］

判定問題と最適化問題

- 判定問題を高速に解く方法があれば，多くの場合，最適化問題も高速に解ける
- 判定問題がクラス **P** に属す問題であれば，多くの場合，最適化問題もクラス **P** に属す

- 従って，問題の難しさ(易しさ)を考えるときは，判定問題だけを議論する

グラフ彩色問題の場合

- グラフ彩色判定問題 (k-Color)
  - 入力：グラフ G および正整数 k
  - 性質： G は k–彩色可能
- グラフ彩色最適化問題 (Chromatic)
  - 入力：グラフ G
  - 出力： G の彩色数

- 判定問題を解くアルゴリズム colorable(G, k)
- 最適化問題を解くアルゴリズム chromatic(G)

colorable と chromatic

- グラフの頂点数 n を問題のサイズと考える
- colorable(G, k) の計算時間：$f(n)$
  chromatic(G) の計算時間：$g(n)$ とする

f は n と k の関数かもしれないが,
$k \le n$ だから（多分）問題ない

$$f(n,k) = (n+k)^3 \le (n+n)^3 = 8n^3 \leftarrow OK$$

$$f(n,k) = n^k \le n^n \leftarrow n \text{ の多項式ではない！}$$

- 課題 b1 から $g(n) = \lceil \log_2 n \rceil f(n)$ であるから,
  『$f(n)$ が n の多項式ならば, $g(n)$ も多項式』

$$\text{k-Color} \in \mathbf{P} \Rightarrow \text{Chromatic} \in \mathbf{P}$$

colorable と chromatic

```
def colorable(G, k):
    """ chromatic(G) が使えるならば ... """
    return (k >= chromatic(G))
```

- colorable(G, k) の計算時間：$f(n)$
  chromatic(G) の計算時間：$g(n)$ とすると
- $f(n) = g(n) + c$ である（c は比較の時間）から,
  『$g(n)$ が n の多項式ならば, $f(n)$ も多項式』

$$\text{k-Color} \in \mathbf{P} \Leftarrow \text{Chromatic} \in \mathbf{P}$$

クラス P に属する問題

- オイラーグラフ判定問題：O(m)　m は枝数
- ソーティング問題：O($n \log n$)　n は要素数
- 二つの n 次正方行列の積の計算：O($n^{2.81}$)
- 全頂点間の最短経路を見つける問題：O(n^3)
- ２頂点間の最短経路を見つける問題：O(n^2)

　［オーダー記法 O(m) についてはあとで説明］

◎ データ構造とアルゴリズム

Wirth 著，Prentice-Hall (1976)

Algorithm + Data Structures = Programs

多項式の値

n 次多項式

$$p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0 \quad (a_n \neq 0)$$

次のように変形できる

$$p(x) = ((\cdots((a_n x + a_{n-1})x + a_{n-2})x + \cdots + a_2)x + a_1)x + a_0$$

```
def Horner(A, x):
    """ A[0] + A[1]*x + A[2]*x**2 + ... を計算する """
    val = A[-1]                     # A[-1] = A[len(A) - 1]
    for k in range(len(A) - 2, -1, -1):
        val *= x                    # 乗算 n 回：n は len(A) - 1
        val += A[k]                 # 加算 n 回
    return val
```

Todd の方法

4次多項式 $p(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \quad (a_4 \neq 0)$

$$= a_4[\{(x+\lambda_1)x+\lambda_2\}\{(x+\lambda_1)x+x+\lambda_3\}+\lambda_4]$$

$$\lambda_1 = \frac{a_3 - a_4}{2a_4},\ \ \lambda_2 = \frac{a_1}{a_4} - \lambda_1\frac{a_2}{a_4} + \lambda_1{}^2(\lambda_1+1),$$

$$\lambda_3 = \frac{a_2}{a_4} - \lambda_1(\lambda_1+1) - \lambda_2, \qquad \lambda_4 = \frac{a_0}{a_4} - \lambda_2\lambda_3$$

$$p_1 = x+\lambda_1,\ \ p_2 = p_1 \times x,\ \ p_3 = p_2 + \lambda_2,\ \ p_4 = p_2 + x,$$

$$p_5 = p_4 + \lambda_3,\ \ p_6 = p_3 \times p_5,\ \ p_7 = p_6 + \lambda_4,\ \ p_8 = a_4 \times p_7$$

λ_1〜λ_4 を計算する前処理（preconditioning）を別にすると
乗算３回と加算５回で多項式の値 $p(x) = p_8$ が計算できる

高速な行列積

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} \quad (i, j = 1, 2)$$

通常の計算では, 乗算８回と加算４回

$\rightarrow$ 乗算７回と加算18回で計算する方法

(Strassen, 1969)

$\rightarrow$ N 次正方行列の積を $\mathrm{O}(N^{2.81})$ で計算できる

[$\mathrm{O}(N^{2.81})$ は “ビッグオー N の 2.81 乗” と読む]

オーダー記法

- 計算量の上界値を評価する： $\mathrm{O}(\bullet)$ “ビッグオー”
- 計算量の下界値を評価する： $\Omega(\bullet)$ “ビッグオメガ”

$T(N) = \mathrm{O}(\varphi(N))$ とは：

ある正定数 c と正整数 N_0 が存在して,
$N \geq N_0$ に対し $T(N) \leq c\varphi(N)$ が成立する

$N^2,\ 100N^2,\ 5N^2 + \log N,\ N^2 + 1000N + 5$
などはすべて $\mathrm{O}(N^2)$ と書ける

$\mathrm{O}(1)$ 定数オーダー（N に独立）
$\mathrm{O}(N)$ 線形オーダー（リニアオーダー）

オーダー記法

- 計算量の上界値を評価する： $O(\bullet)$ “ビッグオー”
- 計算量の下界値を評価する： $\Omega(\bullet)$ “ビッグオメガ”

$T(N)=\Omega(\varphi(N))$ とは：

ある正定数 c が存在して,
無限個の N に対し $T(N)\geq c\varphi(N)$ が成立する

$T(N)=\begin{cases} N^2, & N:\text{奇数} \\ N^3, & N:\text{偶数} \end{cases}$ のとき, $T(N)=\Omega(N^3)$ である

N 次多項式の値を求める計算量は $O(N)$ かつ $\Omega(N)$

行列を（１次元）リストで記憶

n 行 m 列の行列 $A = [a_{ij}] \quad 0 \le i < n,\ 0 \le j < m$

3 行 4 列の行列 $A = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \end{bmatrix}$

リストでは

$$A = [a_{00}, a_{01}, a_{02}, a_{03},\ a_{10}, a_{11}, a_{12}, a_{13},\ a_{20}, a_{21}, a_{22}, a_{23}]$$

$\mathrm{A[pos]} = a_{ij}$

i ← pos / m,　j ← pos % m

pos ← i * m + j

$$\mathrm{A[6]} = a_{12} \iff \begin{cases} i = 6 / 4 = 1,\quad j = 6 \% 4 = 2 \\ pos = 1 * 4 + 2 = 6 \end{cases}$$

リストを行列として出力

```
def prntMatrix(A, n, m):
    """ リスト A を n 行 m 列の行列として出力 """
    if len(A) < n * m:
        print "impossible!"
        return
    for k in range(n):
        for j in range(m):
            print "%3d" % A[k * m + j],
        print
    return
```

print 文の最後のコンマで改行しない

ここは，強制的に改行

出力のフォーマット指定

Strassen の方法

$$\begin{bmatrix} c_0 & c_1 \\ c_2 & c_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_1 \\ a_2 & a_3 \end{bmatrix} \begin{bmatrix} b_0 & b_1 \\ b_2 & b_3 \end{bmatrix} \qquad \begin{array}{l} c_0 = a_0 b_0 + a_1 b_2,\ c_1 = a_0 b_1 + a_1 b_3, \\ c_2 = a_2 b_0 + a_3 b_2,\ c_3 = a_2 b_1 + a_3 b_3 \end{array}$$

$$p_1 = (a_1 - a_3) \times (b_2 + b_3),\ \ p_2 = (a_0 + a_3) \times (b_0 + b_3),$$

$$p_3 = (a_0 - a_2) \times (b_0 + b_1),\ \ p_4 = (a_0 + a_1) \times b_3,$$

$$p_5 = a_0 \times (b_1 - b_3),\ \ p_6 = a_3 \times (b_2 - b_0),\ \ p_7 = (a_2 + a_3) \times b_0$$

$$c_0 = p_1 + p_2 - p_4 + p_6,\ \ c_1 = p_4 + p_5,$$

$$c_2 = p_6 + p_7,\ \ c_3 = p_2 - p_3 + p_5 - p_7$$

乗算７回と加（減）算18回

関数 strassen

$$p_1 = (a_1 - a_3) \times (b_2 + b_3),\ p_2 = (a_0 + a_3) \times (b_0 + b_3),$$
$$p_3 = (a_0 - a_2) \times (b_0 + b_1),\ p_4 = (a_0 + a_1) \times b_3,$$
$$p_5 = a_0 \times (b_1 - b_3),\ p_6 = a_3 \times (b_2 - b_0),\ p_7 = (a_2 + a_3) \times b_0$$

```
def strassen(A, B):
    """ Strassen's の方法，A と B のサイズは２ """
    p1 = (A[1] - A[3]) * (B[2] + B[3])
    p2 = (A[0] + A[3]) * (B[0] + B[3])
    p3 = (A[0] - A[2]) * (B[0] + B[1])
    p4 = (A[0] + A[1]) *     B[3]
    p5 =      A[0]     * (B[1] - B[3])
    p6 =      A[3]     * (B[2] - B[0])
    p7 = (A[2] + A[3]) *     B[0]
    return [p1 + p2 - p4 + p6, p4 + p5,
            p6 + p7, p2 - p3 + p5 - p7]
```

$$c_0 = p_1 + p_2 - p_4 + p_6$$
$$c_1 = p_4 + p_5$$
$$c_2 = p_6 + p_7$$
$$c_3 = p_2 - p_3 + p_5 - p_7$$

高速な行列積

N 次正方行列, $N = 2^n$ とする

$$\begin{bmatrix} C_0 & C_1 \\ C_2 & C_3 \end{bmatrix} = \begin{bmatrix} A_0 & A_1 \\ A_2 & A_3 \end{bmatrix} \begin{bmatrix} B_0 & B_1 \\ B_2 & B_3 \end{bmatrix}$$

A_0 などは 2^{n-1} 次正方行列

$$P_1 = (A_1 - A_3) \times (B_2 + B_3),\ P_2 = (A_0 + A_3) \times (B_0 + B_3),$$
$$P_3 = (A_0 - A_2) \times (B_0 + B_1),\ P_4 = (A_0 + A_1) \times B_3,$$
$$P_5 = A_0 \times (B_1 - B_3),\ P_6 = A_3 \times (B_2 - B_0),\ P_7 = (A_2 + A_3) \times B_0$$

$$C_0 = P_1 + P_2 - P_4 + P_6,\ C_1 = P_4 + P_5,$$
$$C_2 = P_6 + P_7,\ C_3 = P_2 - P_3 + P_5 - P_7$$

$P_1 \sim P_7, C_0 \sim C_3$ はすべて行列

関数 product

```
def product(A, B, N):
    """ サイズ N（＝ 2 ** n）の行列 A と B の積を出力 """
    if N == 2:
        return strassen(A, B)
    A0, A1, A2, A3 = divide4(A, N)     # 行列 A を４分割
    B0, B1, B2, B3 = divide4(B, N)     # 行列 B を４分割
    p1 = product(difM(A1, A3), sumM(B2, B3), N / 2)
       .....
       .....
    C0 = sumM(difM(sumM(p1, p2), p4), p6)
       .....
       .....
    return conquer(C0, C1, C2, C3, N)  # C0～C3 をまとめる
```

補助関数

```
def sumM(A, B):
    """ 行列の足し算 A + B を出力 """
    return [a + b for a, b in zip(A, B)]

def divide4(A, N)
    """ サイズ N の行列 A を４つの部分行列に分ける """
    A0, A1, A2, A3 = ([], [], [], [])
      .....   A0.extend(A[・・ : ・・])
    return (A0, A1, A2, A3)

def conquer(C0, C1, C2, C3, N):
    """ ４つの部分行列からサイズ N の行列 C を作る """
    C = []
      .....   C.extend(C0[・・ : ・・])
    return C
```

課題 p2：高速な行列積

- Strassen の方法を用いた行列積を計算する関数 product(A, B, N) を完成せよ。
- 関数 product を使うためのユーザ向け関数 fastProduct(A, B) を作れ。

```
def fastProduct(A, B):
    if len(A) != len(B):
        print "impossible! [len(A) != len(B)]
        return
    if ..... :   # N == 2 ** n か？ 等
       .....
    return product(A, B, N)
```

高速な行列積の計算量

- 2^n 次正方行列の積に必要な乗算の回数：$f(n)$ とする
- 関数 product は自分自身を７回呼び出す
  呼び出し１回につき乗算は $f(n-1)$ 回

$$f(n) = 7f(n-1),\ \ f(1) = 7 \ \Rightarrow \ f(n) = 7^n$$

- $2^n = N$ 次行列の積に必要な乗算回数は 7^n

$$7^n = \boxed{7^{\log_2 N} = N^{\log_2 7}} = N^{2.80735\cdots} = \mathrm{O}(N^{2.81})$$

$$\begin{aligned} x = a^{\log b} \Rightarrow \log x &= \log(a^{\log b}) = (\log b)(\log a) \\ &= (\log a)(\log b) = \log(b^{\log a}) \end{aligned}$$

$$\Rightarrow x = b^{\log a} \Rightarrow \boxed{a^{\log b} = b^{\log a}}$$

加減算の回数

- 加減算の回数：$g(n) = \mathrm{O}(N^{2.81})$

$$g(n) = 7g(n-1) + 18\,(2^{n-1} \times 2^{n-1}) = 7g(n-1) + \frac{9}{2}4^n$$
$$= \cdots < 6 \times 7^n = \mathrm{O}(7^n) = \mathrm{O}(N^{2.81})$$

その後, 多くの研究者により指数は
2.81 から 2.5 以下にまで下げられている

1階差分方程式

$g(n) = 7g(n-1) + \frac{9}{2}4^n \quad (n \geq 1),\ g(0) = 0$ を解いてみる

$g(n) = 7^n y_n \quad (n \geq 0)$ と変数変換する。$y_0 = 0$ である。

元の式に代入すると,

$$7^n y_n = 7 \times 7^{n-1} y_{n-1} + \frac{9}{2}4^n \Rightarrow y_n = y_{n-1} + \frac{9}{2}\left(\frac{4}{7}\right)^n \quad (n \geq 1)$$

$a = \frac{9}{2},\ b = \frac{4}{7}$ とおけば $y_n = y_{n-1} + ab^n$

1階差分方程式

$$y_n = y_{n-1} + ab^n \quad \left(a = (9/2),\ b = (4/7)\right)$$

$$y_n = y_{n-1} + ab^n$$
$$y_{n-1} = y_{n-2} + ab^{n-1}$$
$$\cdots$$
$$y_2 = y_1 + ab^2$$
$$y_1 = y_0 + ab = ab$$

よって $y_n = a\sum_{k=1}^{n} b^k$

$b = (4/7) < 1$ だから収束する

$$y_n = a\sum_{k=1}^{n} b^k = \frac{9}{2}\sum_{k=1}^{n}\left(\frac{4}{7}\right)^k < \frac{9}{2}\sum_{k=1}^{\infty}\left(\frac{4}{7}\right)^k = \frac{9}{2}\times\frac{4/7}{1-(4/7)} = 6$$

$$g(n) = 7^n y_n = 7^n \frac{9}{2}\sum_{k=1}^{n}\left(\frac{4}{7}\right)^k < 6\times 7^n = \mathrm{O}(7^n) = \mathrm{O}(N^{2.81})$$