アルゴリズムとデータ構造

第24週目

担当　情報システム部門　徳光政弘
2025年12月9日

今日の内容

- 多項式の表現
- 多項式の計算
- 行列計算の最適化

多項式の定義

- n次の多項式を考える

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

多項式の定義

- xと多項式が与えられると、値を計算できる

アルゴリズム 11.1　多項式の計算を行う基本的なアルゴリズム

入力：多項式の係数を表す A[0], A[1], ..., A[n], および値 x

```
sum=A[0];
for (i=1; i<=n; i=i+1) {
  xp=1;
  for (j=1; j<=i; j=j+1) { xp=xp*x; }
  sum=sum+A[i]*xp;
}
sum を出力;
```

次数部の計算

多項式の定義

- 次数ごとに素直に計算している点が改善点

アルゴリズム 11.1　多項式の計算を行う基本的なアルゴリズム

入力：多項式の係数を表す A[0], A[1], ..., A[n], および値 x

```
sum=A[0];
for (i=1; i<=n; i=i+1) {
  xp=1;
  for (j=1; j<=i; j=j+1) { xp=xp*x; }
  sum=sum+A[i]*xp;
}
sum を出力;
```

次数ごとの計算

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

動的計画法による改善

- 次数ごとの計算結果を利用する

$$x^i = x^{i-1} \times x$$

アルゴリズム 11.2　多項式の計算を行う動的計画法を用いたアルゴリズム

入力：多項式の係数を表す A[0], A[1], . . ., A[n]，および値 x

```
sum=A[0];
xp=1;
for (i=1; i<=n; i=i+1) {
  xp=xp*x;
  sum=sum+A[i]*xp;
}
sum を出力;
```

ホーナーの方法

- 多項式の性質を用いて、計算を効率化

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

-

$3x^3+2x^2-x+5$ → $((3x+2)x-1)x+5$

① $3x+2$ を計算する.

② ①の結果に x を掛けて -1 を足す.

③ ②の結果に x を掛けて 5 を足す.

ホーナーの方法

アルゴリズム 11.3　ホーナーの方法

入力：多項式の係数を表す A[0], A[1], . . ., A[n]，および値 x

```
sum=A[n];
for (i=n-1; i>=0; i=i-1) {
  sum=sum*x+A[i];
}
sum を出力;
```

N=10,000,000の場合、動的計画法に対してホーナーの方法は25%高速に実行できる

行列積のアルゴリズム

- 科学技術計算で頻繁に登場
  - 3DCG、ニューラルネットワーク、物理計算

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} = \begin{pmatrix} 5 & 9 & 2 \\ -1 & 7 & 5 \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix} = \begin{pmatrix} -3 & 2 \\ 5 & 8 \\ -6 & -3 \end{pmatrix}$$

$$c_{ij} = \sum_{k=1}^{q} a_{ik} b_{kj}$$

行列積のアルゴリズム

$$
\begin{aligned}
C &= \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} \\
&= \begin{pmatrix} a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31} & a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32} \\ a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31} & a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32} \end{pmatrix} \\
&= \begin{pmatrix} 5\times(-3)+9\times5+2\times(-6) & 5\times2+9\times8+2\times(-3) \\ (-1)\times(-3)+7\times5+5\times(-6) & (-1)\times2+7\times8+5\times(-3) \end{pmatrix} \\
&= \begin{pmatrix} 18 & 76 \\ 8 & 39 \end{pmatrix}
\end{aligned}
$$

行列積のアルゴリズム

アルゴリズム 11.4　行列積を求める基本的なアルゴリズム

入力：$p \times q$ の 2 次元行列 A と $q \times r$ の 2 次元行列 B を表す 2 次元配列 A, B

```
for (i=1; i<=p; i=i+1) {
  for (j=1; j<=r; j=j+1) {
    C[i][j]=0;
    for (k=1; k<=q; k=k+1) { C[i][j]=C[i][j]+A[i][k]*B[k][j]; }
  }
}
```

行列積の連続積

- 複数の行列の積を求めるときに、積を実行する順番で全体の演算回数が違ってくる

$$((A_1 A_2) A_3) A_4, \quad (A_1 A_2)(A_3 A_4), \quad (A_1 (A_2 A_3)) A_4,$$

$$A_1 ((A_2 A_3)) A_4), \quad A_1 (A_2 (A_3 A_4))$$

行列積の連続積

$$((A_1A_2)A_3)A_4: \underbrace{20\times 2\times 30}_{A_1A_2}+\underbrace{20\times 30\times 5}_{(A_1A_2)A_3}+\underbrace{20\times 5\times 25}_{(A_1A_2A_3)A_4}=6700$$

$$(A_1A_2)(A_3A_4): \underbrace{20\times 2\times 30}_{A_1A_2}+\underbrace{30\times 5\times 25}_{A_3A_4}+\underbrace{20\times 30\times 25}_{(A_1A_2)(A_3A_4)}=19950$$

$$(A_1(A_2A_3))A_4: \underbrace{2\times 30\times 5}_{A_2A_3}+\underbrace{20\times 2\times 5}_{A_1(A_2A_3)}+\underbrace{20\times 5\times 25}_{(A_1A_2A_3)A_4)}=3000$$

$$A_1((A_2A_3)A_4): \underbrace{2\times 30\times 5}_{A_2A_3}+\underbrace{2\times 5\times 25}_{(A_2A_3)A_4}+\underbrace{20\times 2\times 25}_{A_1(A_2A_3A_4)}=1550$$

$$A_1(A_2(A_3A_4)): \underbrace{30\times 5\times 25}_{A_3A_4}+\underbrace{2\times 30\times 25}_{A_2(A_3A_4)}+\underbrace{20\times 2\times 25}_{A_1(A_2A_3A_4)}=6250$$

行列の連続積の問題

【問題 11.1】 行列の連続積

n 個の行列 $A_1, A_2, \ldots, A_n$ が与えられ，各行列 A_i は $r_i \times c_i$ 行列であるとする．また，$1 \le i \le n-1$ の範囲で $c_i = r_{i+1}$ が成り立つものとする．このとき，この行列の連続積 $A_1 A_2 \cdots A_n$ を求める場合に，もっとも計算時間が短くなる積の計算順序を求めよ．

行列の連続積の問題

- 行列の積の定義

  行列 A_i から行列 A_j までの行列 $A_i, A_{i+1}, \ldots, A_j$
- 行列のiからjまでの計算時間　m(i, j)
- 行列の連続積を求める　m(1, n)を求めること

行列の連続積の問題

- m(i, j)の定義

$$m(i,j) = \underbrace{m(i,k)}_{A_i A_{i+1} \cdots A_k \text{の計算}} + \underbrace{m(k+1,j)}_{A_{k+1} A_{k+2} \cdots A_j \text{の計算}} + \underbrace{r_i c_k c_j}_{\text{アルゴリズム 11.4 による行列積計算}}$$

- 最小時間の定義

$$m(i,j) = \begin{cases} \min_{i \le k \le j-1} \{m(i,k) + m(k+1,j) + r_i c_k c_j\} & (i < j \text{ の場合}) \\ 0 & (i = j \text{ の場合}) \end{cases}$$

行列の連続積の問題

アルゴリズム 11.5

行列の連続積を求める分割統治法を用いたアルゴリズム

入力：入力行列の行数と列数を表す配列 R と C

```
Matrix_Chain(i,j) {
  if (i==j) return 0;
  else {
    min=+∞;
    for (k=i; k<=j-1; k=k+1) {
      m=Matrix_Chain(i,k)+Matrix_Chain(k+1,j)+R[i]*C[k]*C[j];
      if (m<min) min=m;
    }
  }
  return min;
}
//Matrix_Chain(1,n)と指定して実行する.
```

さまざまな積の組み合わせを再帰的に求めるために実行効率が悪い

動的計画法による計算

- m(i, j)の計算結果を保持することで、計算を効率化する（つまり動的計画法の考え方）

m(1, 1)、m(2, 2)、m(3, 3)、m(4, 4)を求める

—	1	2	3	4
1	0			
2	—	0		
3	—	—	0	
4	—	—	—	0

間隔 0 （a）

m(1, 2)、m(2, 3)、m(3, 4)を求める

—	1	2	3	4
1	0	1200		
2	—	0	300	
3	—	—	0	3750
4	—	—	—	0

間隔 1 （b）

動的計画法による計算

求まっている計算結果を使うことで、計算を効率的に実行できる

—	1	2	3	4
1	0	1200	500	
2	—	0	300	550
3	—	—	0	3750
4	—	—	—	0

(c)

間隔 2

—	1	2	3	4
1	0	1200	500	1550
2	—	0	300	550
3	—	—	0	3750
4	—	—	—	0

(d)

間隔 3

動的計画法による計算

実際の計算

$$m(1,2) = m(1,1) + m(2,2) + r_1c_1c_2 = 0 + 0 + 20 \times 2 \times 30 = 1200$$

$$m(2,3) = m(2,2) + m(3,3) + r_2c_2c_3 = 0 + 0 + 2 \times 30 \times 5 = 300$$

$$m(3,4) = m(3,3) + m(4,4) + r_3c_3c_4 = 0 + 0 + 30 \times 5 \times 25 = 3750$$

動的計画法による計算

実際の計算（最小の組み合わせを選ぶ）

$$
\begin{aligned}
m(1,3) &= \min\{m(1,1) + m(2,3) + r_1c_1c_3,\ m(1,2) + m(3,3) + r_1c_2c_3\} \\
&= \min\{0 + 300 + 20 \times 2 \times 5,\ 1200 + 0 + 20 \times 30 \times 5\} \\
&= 500 \\
m(2,4) &= \min\{m(2,2) + m(3,4) + r_2c_2c_4,\ m(2,3) + m(4,4) + r_2c_3c_4\} \\
&= \min\{0 + 3750 + 2 \times 30 \times 25;\ 300 + 0 + 2 \times 5 \times 25\} \\
&= 550
\end{aligned}
$$

動的計画法による計算

実際の計算（最小の組み合わせを選ぶ）

$$
\begin{aligned}
m(1,4) = \min\{&m(1,1)+m(2,4)+r_1c_1c_4,\ m(1,2)+m(3,4)+r_1c_2c_4,\\
&m(1,3)+m(4,4)+r_1c_3c_4\}\\
= \min\{&0+550+20\times 2\times 25,\ 1200+3750+20\times 30\times 25,\\
&500+0+20\times 5\times 25\} = 1550
\end{aligned}
$$

動的計画法による計算

アルゴリズム 11.6
行列の連続積を求める動的計画法を用いたアルゴリズム

入力：入力行列の行数と列数を表す配列 R と C

```
for (i=1; i<=n; i=i+1) { M[i][i]=0; }
for (w=1; w<=n-1; w=w+1) {              //w は i と j の間隔を表す
  for (i=1; i<=n-w; i=i+1) {
    j=i+w; M[i][j]=+∞;
    for (k=i; k<=j-1; k=k+1) {
      m=M[i][k]+M[k+1][j]+r[i]*c[k]*c[j];
      if (m<M[i][j]) { M[i][j]=m; }
    }
  }
}
M[1][n]を出力;
```

ストラッセンの行列積

- n×nに限ってのアルゴリズム　O(n^2.81)
- 行列の積はO(n^3)なので速い

行列の基本形

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \qquad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$C = AB = \begin{pmatrix} a_{11} \times b_{11} + a_{12} \times b_{21} & a_{11} \times b_{12} + a_{12} \times b_{22} \\ a_{21} \times b_{11} + a_{22} \times b_{21} & a_{21} \times b_{12} + a_{22} \times b_{22} \end{pmatrix}$$

行列の積の定義

- ここがストラッセンの行列積の肝心なところ
- なぜこのような計算式が成立するかは別の話題
- 固有値を使って成立することが確認されている

$$x_1=(a_{11}+a_{22})\times(b_{11}+b_{22}), \qquad x_2=(a_{21}+a_{22})\times b_{11},$$

$$x_3=a_{11}\times(b_{12}-b_{22}), \qquad x_4=a_{22}\times(b_{21}-b_{11}),$$

$$x_5=(a_{11}+a_{12})\times b_{22}, \qquad x_6=(a_{21}-a_{11})\times(b_{11}+b_{12}),$$

$$x_7=(a_{12}-a_{22})\times(b_{21}+b_{22})$$

$$C=AB=\begin{pmatrix} x_1+x_4-x_5+x_7 & x_3+x_5 \\ x_2+x_4 & x_1+x_3-x_2+x_6 \end{pmatrix}$$

行列の積の定義

- 基本形は2×2の行列だが、次数が大きい行列は分割して元の定義で計算する
- n×nの行列の場合

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \qquad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$C = AB = \begin{pmatrix} A_{11} \times B_{11} + A_{12} \times B_{21} & A_{11} \times B_{12} + A_{12} \times B_{22} \\ A_{21} \times B_{11} + A_{22} \times B_{21} & A_{21} \times B_{12} + A_{22} \times B_{22} \end{pmatrix}$$

行列の積の定義

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \qquad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$C = AB = \begin{pmatrix} A_{11} \times B_{11} + A_{12} \times B_{21} & A_{11} \times B_{12} + A_{12} \times B_{22} \\ A_{21} \times B_{11} + A_{22} \times B_{21} & A_{21} \times B_{12} + A_{22} \times B_{22} \end{pmatrix}$$

$$\begin{aligned} X_1 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}), & X_2 &= (A_{21} + A_{22}) \times B_{11}, \\ X_3 &= A_{11} \times (B_{12} - B_{22}), & X_4 &= A_{22} \times (B_{21} - B_{11}), \\ X_5 &= (A_{11} + A_{12}) \times B_{22}, & X_6 &= (A_{21} - A_{11}) \times (B_{11} + B_{12}), \\ X_7 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \end{aligned}$$

$$C = AB = \begin{pmatrix} X_1 + X_4 - X_5 + X_7 & X_3 + X_5 \\ X_2 + X_4 & X_1 + X_3 - X_2 + X_6 \end{pmatrix}$$

アルゴリズム 11.7　ストラッセンの行列積アルゴリズム

入力：2 つの $n \times n$ 行列 A, B を表す 2 次元配列 A, B

```
Matrix_Multiplay_Strassen(A,B) {
  if (A, B が 1x1 の行列) return AxB;
  else {
    A, B をそれぞれ A11, A12, A21, A22 と B11, B12, B21, B22 に分割;
    X1=Matrix_Multiplay_Strassen(A11+A22,B11+B22);
    X2=Matrix_Multiplay_Strassen(A21+A22,B11);
    X3=Matrix_Multiplay_Strassen(A11,B12-B22);
    X4=Matrix_Multiplay_Strassen(A22,B21-B11);
    X5=Matrix_Multiplay_Strassen(A11+A12,B22);
    X6=Matrix_Multiplay_Strassen(A21-A11,B11+B12);
    X7=Matrix_Multiplay_Strassen(A12-A22,B21+B22);
    C1=X1+X4-X5+X7; C2=X3+X5; C3=X2+X4; C4=X1+X3-X2+X6;
    C1, C2, C3, C4 を 1 つの配列 C に結合;
    return C;
  }
}
```

代入で実現

計算量の考え方

$\frac{n}{2} \times \frac{n}{2}$ 行列に分割

4つの $\frac{n}{2} \times \frac{n}{2}$ 行列を1つの $n \times n$ 行列に結合 ⇒ $O(n^2)$

18回の行列の足し算(引き算) ⇒ $\frac{n}{2} \times \frac{n}{2} \fallingdotseq O(n^2)$

7回の再帰呼出し以外はO(n^2)になっている

$$T(n) = 7 \times \underbrace{T\left(\frac{n}{2}\right)}_{\text{再帰呼び出し}} + \underbrace{O(n^2)}_{\text{その他}}$$

$$\fallingdotseq O(n^{\log_2 7})$$

$$\fallingdotseq O(n^{2.81}) \qquad (\because \log_2 7 \cong 2.81)$$