**Temat: Programowanie dynamiczne. Przykłady zastosowania.**

## 1. Programowanie dynamiczne

Użycie strategii programowania dynamicznego polega na zapamiętaniu w odpowiedniej strukturze (najczęściej tablicy) wyników rozwiązania podproblemów, na które został podzielony problem zasadniczy, unikając w ten sposób wielokrotnych obliczeń dla tego samego podproblemu. Programowanie dynamiczne prowadzi do całkowitej bądź częściowej eliminacji rekurencji.
Problem może być rozwiązany techniką programowania dynamicznego gdy ma własność optymalnej podstruktury czyli jego optymalne rozwiązanie jest funkcją optymalnych rozwiązań podproblemów. Innymi słowy problem ma własność optymalnej podstruktury gdy ma naturę rekurencyjną. Jednak rozwiązanie problemu nie powinno być zapisywane w postaci kodu rekurencyjnego, ale poprzez spamiętywanie wyników lokalnych.

## 2. Przykłady zastosowania

### a) Ciąg Fibonacciego

Funkcja obliczająca wartości elementów ciągu Fibonacciego
Ciąg Fibbonaciego: $F(0)=0$
$F(1)=1$
$F(n)=F(n-1)+F(n-2)$ dla $n>2$

Algorytm I (rozwiązanie rekurencyjne)

```
F(n)
{
    if (n==0) return 0;
    else if (n==1) return 1;
         else return F(n-1)+F(n-2);
}
```

Koszt czasowy: $O(2^n)$
Koszt pamięciowy: $O(n)$

Algorytm II (programowanie dynamiczne)

```
F(n)
{
    if (n==1) return 0;
    else if (n==2) return 1;
    for (i=2; i<=n; i++)
       {
          c = a+b;
          a = b;
          b = c;
        }
     return c;
}
```

Koszt czasowy: O(*n*)
Złożoność pamięciowa: O(*1*)

**b) Problem optymalnego mnożenia macierzy**

**WP** : Ciąg rozmiarów macierzy: $M_1, M_2, ..., M_n$, gdzie $M_i$ jest macierzą o $r_{i-1}$ wierszach i $r_i$ kolumnach; wektor rozmiarów $R$ : [$r_0$, $r_1$, …, $r_n$]

**WK:** Takie rozstawienie nawiasów w iloczynie macierzowym: $M_1 \circ M_2 \circ ... \circ M_n$, aby liczba wykonanych operacji mnożenia wykonanych przy obliczaniu iloczynu macierzy była minimalna.

Przykład 1

$$M_1 = [5 \times 10], M_2 = [10 \times 20], M_3 = [20 \times 1], M_4 = [1 \times 10]$$

$$R = [5 \quad 10 \quad 20 \quad 1 \quad 10],$$

Jeżeli mnożymy dwie macierze: $A = [p \times q], B = [q \times r]$, to wykonywanych jest *pqr* mnożeń.

I kolejność: $((M_1 \circ M_2) \circ (M_3 \circ M_4)) = M_{12} \circ M_{34} = M_{1234}$
Liczba mnożeń:

$$\underbrace{5 \cdot 10 \cdot 20}_{\text{koszt mnożenia macierzy } M_1 \circ M_2} + \underbrace{20 \cdot 1 \cdot 10}_{\text{koszt mnożenia macierzy } M_3 \circ M_4} + \underbrace{5 \cdot 20 \cdot 10}_{\text{koszt mnożenia macierzy } M_{12} \circ M_{34}} = 2200$$

II kolejność: $(M_1 \circ (M_2 \circ (M_3 \circ M_4))) = M_1 \circ (M_2 \circ M_{34}) = M_1 \circ M_{234} = M_{1234}$
Liczba mnożeń:

$$\underbrace{20 \cdot 1 \cdot 10}_{\text{koszt mnożenia macierzy } M_3 \circ M_4} + \underbrace{10 \cdot 20 \cdot 10}_{\text{koszt mnożenia macierzy } M_2 \circ M_{34}} + \underbrace{5 \cdot 10 \cdot 10}_{\text{koszt mnożenia macierzy } M_1 \circ M_{234}} = 2700$$

Algorytm naiwny

Rozważyć wszystkie możliwe ustawienia nawiasów, obliczając dla każdej możliwości liczbę wykonanych operacji mnożenia.

Liczba możliwych ustawień nawiasów $X(n)$ w iloczynie $M_1 \circ M_2 \circ ... \circ M_n$ wyraża się następującą zależnością rekurencyjną:

$$(*) \quad X(n) = \begin{cases} 1 \text{ dla } n = 1 \\ \sum_{i=1}^{n-1} X(i) \cdot X(n-i-1) \text{ dla } n > 1 \end{cases}$$

Można pokazać, że postać jawna równania (*) jest następująca:

$$X(n) = \frac{1}{n}\binom{2n-1}{n-1} = \frac{1}{n} \cdot \frac{n \cdot ... \cdot (2n-1)}{(n-1)!} = \frac{(n+1) \cdot ... \cdot (2n-1)}{(n-1)!}$$

Można udowodnić, że:

$$\frac{(n+1) \cdot (n+3) \cdot ... \cdot (2n-1)}{(n-1)!} \geq 2^{n-3}$$

Algorytm oparty na programowaniu dynamicznym

Oznaczenie:

$m_{ij}$ - minimalny koszt obliczenia fragmentu iloczynu macierzy od macierzy *i* do macierzy *j*

$$M_i \circ M_{i+1} \circ \ldots \circ M_j \text{ dla } 1 \le i \le j \le n$$

$$m_{ij} = \begin{cases} 0 \text{ dla } i = j \\ \min\limits_{i \le k < j} \left( m_{ik} + m_{k+1\,j} + r_{i-1} r_k r_j \right) \text{ dla } i < j \end{cases}$$

$m_{ik}$ – minimalny koszt mnożenia macierzy $M_i \circ M_{i+1} \circ \ldots \circ M_k$ o rozmiarze $r_{i-1} \times r_k$

$m_{k+1\,j}$ – minimalny koszt mnożenia macierzy $M_{k+1} \circ M_{k+2} \circ \ldots \circ M_j$ o rozmiarze $r_k \times r_j$

$r_{i-1} \cdot r_k \cdot r_j$ – koszt mnożenia macierzy $M_{ik} \circ M_{k+1\,j}$ o rozmiarze $r_{i-1} \times r_j$

Algorytm polega na wyznaczeniu wartości funkcji $m_{ij}$, które zapamiętuje w tablicy dwuwymiarowej.

Własność optymalnej podstruktury dla problemu optymalnego nawiasowania wynika z faktu, że optymalne nawiasowanie całego ciągu jest zależne od optymalnego nawiasowania fragmentów:

$$M_1 \circ M_{i+1} \circ \ldots \circ M_k$$
$$M_{k+1} \circ M_{k+2} \circ \ldots \circ M_n$$

dla $k$ =1, 2, …, $n$-1

Przykład 2

$$M_1 = [5 \times 10], M_2 = [10 \times 20], M_3 = [20 \times 1], M_4 = [1 \times 10]$$

$$R : [5, 10, 20, 1, 10]$$

$$M = \begin{bmatrix} m_{11} & m_{22} & m_{33} & m_{44} \\ m_{12} & m_{23} & m_{34} & \\ m_{13} & m_{24} & & \\ m_{14} & & & \end{bmatrix}$$

Proces wyznaczania wartości macierzy *M*

Elementy pierwszego wiersza macierzy *M*

$$m_{11} = 0, m_{22} = 0, m_{33} = 0, m_{44} = 0$$

Elementy drugiego wiersza macierzy *M* (koszty fragmentów o długości dwóch macierzy), wyrazy $m_{12}$, $m_{23}$, $m_{34}$

$$m_{12} = \min_{1 \le k < 2} \left( m_{1k} + m_{k+12} + r_0 r_k r_2 \right) =$$

$$= \min \left\{ \underbrace{m_{11} + m_{22} + r_0 r_1 r_2}_{k=1} \right\} = 0 + 0 + 5 \cdot 10 \cdot 20 = 1000$$

$$m_{23} = \min_{2 \le k < 3} \left( m_{2k} + m_{k+13} + r_1 r_k r_3 \right) =$$

$$= \min \left\{ \underbrace{m_{22} + m_{33} + r_1 r_2 r_3}_{k=2} \right\} = 0 + 0 + 10 \cdot 20 \cdot 1 = 200$$

$$m_{34} = \min_{3\le k<4}\left(m_{3k} + m_{k+14} + r_2 r_k r_4\right) =$$

$$= \min\left\{\underbrace{m_{33} + m_{44} + r_2 r_3 r_4}_{k=3}\right\} = 0 + 0 + 20\cdot 1\cdot 10 = 200$$

Elementy trzeciego wiersza macierzy *M* (koszty fragmentów o długości trzech macierzy), wyrazy $m_{13}$, $m_{24}$.

$$m_{13} = \min_{1\le k<3}\left(m_{1k} + m_{k+13} + r_0 r_k r_3\right) =$$

$$= \min\left\{\underbrace{m_{11} + m_{23} + r_0 r_1 r_3}_{k=1}, \underbrace{m_{12} + m_{33} + r_0 r_2 r_3}_{k=2}\right\} =$$

$$= \min\left\{0 + 200 + 5\cdot 10\cdot 1, 1000 + 0 + 5\cdot 20\cdot 1\right\} = 250$$

$$m_{24} = \min_{2\le k<4}\left(m_{2k} + m_{k+14} + r_1 r_k r_4\right) =$$

$$= \min\left\{\underbrace{m_{22} + m_{34} + r_1 r_2 r_4}_{k=2}, \underbrace{m_{23} + m_{44} + r_1 r_3 r_4}_{k=3}\right\} =$$

$$= \min\left\{0 + 200 + 10\cdot 20\cdot 10, 200 + 0 + 10\cdot 1\cdot 10\right\} = 300$$

Elementy czwartego wiersza macierzy *M* (koszty fragmentów o długości czterech macierzy), wyrazy $m_{14}$

$$m_{14} = \min_{1\le k<4}\left(m_{1k} + m_{k+14} + r_0 r_k r_4\right) =$$

$$= \min\left\{\underbrace{m_{11} + m_{24} + r_0 r_1 r_4}_{k=1}, \underbrace{m_{12} + m_{34} + r_0 r_2 r_4}_{k=2}, \underbrace{m_{13} + m_{44} + r_0 r_3 r_4}_{k=3}\right\} =$$

$$= \min\left\{0 + 300 + 5\cdot 10\cdot 10, 1000 + 200 + 5\cdot 20\cdot 10, 250 + 0 + 5\cdot 1\cdot 10\right\} = 300$$

Ostatecznie macierz *M* ma następującą postać:

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1000 & 200 & 200 & \\ 250 & 300 & & \\ 300 & & & \end{bmatrix}$$

Optymalny koszt optymalnego mnożenia wynosi 300 i zachodzi przy następującym rozstawieniu nawiasów w iloczynie $M_1 \circ M_2 \circ M_3 \circ M_4$:

$$\left(M_1 \circ \left(M_2 \circ M_3\right)\right) \circ M_4$$

Algorytm

for (*i*=1; *i*<=*n*; *i*++) $m_{ii} = 0$;

for (*t*=1; *t*<=*n*-1; *t*++)

for (*i*=1; *i*<= *n*-*t*; *i*++)

{

*j*=*i*+*t*;

$m_{ij} = \min_{i \le k < j} \left(m_{ik} + m_{k+1\,j} + r_{i-1} r_k r_j\right)$;

};

Złożoność pesymistyczna algorytmu rozwiązującego problem optymalnego nawiasowania stosującego programowanie dynamiczne:

Rozmiar problemu: *n* – liczba macierzy w iloczynie
Operacja elementarna: wyznaczanie iloczynu rozmiarów macierzy $r_{i-1} r_k r_j$

$$T_{\max}(n) = 1 \cdot (n-1) + 2 \cdot (n-2) + \ldots + (n-2) \cdot 2 + (n-1) \cdot 1$$

Można pokazać, że $T_{\max}(n) = \Theta\left(n^3\right)$.

## c) Problem optymalnej triangulacji wielokąta wypukłego

**WP:** $v_0$, $v_2$, ... , $v_{n-1}$ – ciąg wierzchołków wielokąta wypukłego $W$

**WK:** $T$ – zbiór nie przecinających się przekątnych wielokąta, dzielących go na rozłączne trójkąty tak, że suma obwodów powstały trójkątów jest najmniejsza.

Algorytm wykorzystujący programowanie dynamiczne

Problem optymalnego mnożenia macierzy jest szczególnym przypadkiem problemu optymalnej triangulacji. Każdy egzemplarz problemu nawiasowania iloczynu macierzy można sformułować jako równoważny egzemplarz problemu optymalnej triangulacji.

Ciągowi rozmiarów macierzy $r_0, r_1, r_2, ..., r_{n-1}$

odpowiada ciąg wierzchołków wielokąta $v_0, v_1, v_2, ..., v_{n-1}$.

Wystarczy w algorytmie ustalającym optymalne nawiasowanie iloczyn rozmiarów macierzy: $r_{i-1} \cdot r_k \cdot r_j$ zastąpić wywołaniem funkcji obliczającej obwód trójkąta $v_{i-1}, v_k, v_j$. Wartość $m_{ij}$ macierzy wyznaczonej w algorytmie optymalnego nawiasowania opowiada wynikowi optymalnej triangulacji wielokąta: $v_{i-1}, v_i, ..., v_j$.

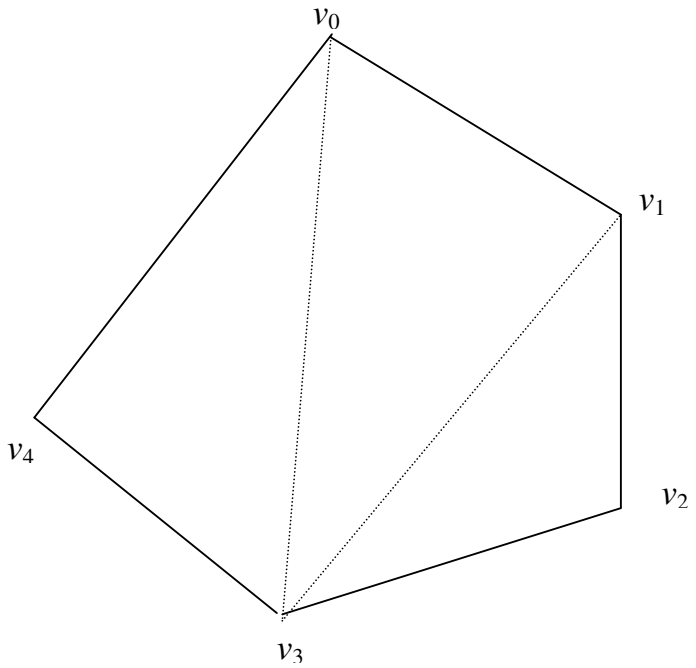
Odpowiednikiem rozstawienia nawiasów w iloczynie macierzowym

$$(M_1 \circ (M_2 \circ M_3)) \circ M_4$$

jest następująca triangulacja:

$(M_1 \circ (M_2 \circ M_3)) \longrightarrow v_0, v_1, v_2, v_3$

$(M_2 \circ M_3) \longrightarrow v_1, v_2, v_3$


$v_0$
$v_1$
$v_2$
$v_3$
$v_4$