WYDZIAŁ
INFORMATYKI
I TELEKOMUNIKACJI

optymalne mnożenie macierzy

Veronika Tronchuk

I rok Informatyki

Nr albumu: 30019

Celem ćwiczenia było zrozumienie problemu optymalnego nawiasowania ciągu macierzy i implementacja algorytmów rekurencyjnych oraz z memoizacją. W ramach zadania sprawdzono przykłady podane w instrukcji oraz dodatkowe przypadki dla większej liczby macierzy

Zaimplementowano dwie wersje algorytmu: czystą rekurencję oraz wersję z memoizacją. Sprawdzono działanie dla różnych zestawów danych. Wyniki zostały porównane i pokazane poniżej.

Zadanie 1-2

```
Liczba macierzy: 3
Rozmiary macierzy:
Macierz 1: 2 x 1
Macierz 2: 1 x 3
Macierz 3: 3 x 4
Wynik (rekurencja): 20
Wynik (memoizacja): 20
```

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;
```

dołączamy biblioteki: wejście-wyjście, wektor i stała INT_MAX.

```
int minMultRec(vector<int>& arr, int i, int j) {
```

początek definicji funkcji minMultRec — rekurencyjnej.

```
if (i + 1 == j)
    return 0;
```

jeśli podciąg zawiera tylko jedną macierz, zwraca 0 — brak mnożenia.

```
for (int k = i + 1; k < j; k++) {
    int curr = minMultRec(arr, i, k) + minMultRec(arr, k, j) + arr[i] * arr[k] * arr[j];

    res = min(curr, res);
}

return res;
```

pętla po wszystkich miejscach podziału:
dla każdego k liczy lewą i prawą część oraz koszt połączenia;
res zapamiętuje najmniejszy koszt.

```
int matrixMultiplication(vector<int>& arr) {
```

```
 int n = arr.size();
 return minMultRec(arr, 0, n - 1);
```

funkcja pomocnicza:
ustala rozmiar n;
uruchamia minMultRec dla całego ciągu.

```
int minMultRecMemo(vector<int>& arr, int i, int j, vector<vector<int>>& memo) {
```

początek funkcji minMultRecMemo — z memoizacją.

```
if (i + 1 == j)
    return 0;
```

tak samo: jeśli jedna macierz — brak mnożenia.

```
if (memo[i][j] != -1)
    return memo[i][j];
```

jeśli wynik dla i, j już policzony — zwraca z memo.

```
int res = INT_MAX;
```

wartość początkowa dla minimum

```
for (int k = i + 1; k < j; k++) {
    int curr = minMultRecMemo(arr, i, k, memo) + minMultRecMemo(arr, k, j, memo) + arr[i] * arr[k] * arr[j];
    res = min(curr, res);
}
```

pętla po wszystkich podziałach:
 rekurencja z memo dla lewej i prawej strony;
 koszt i minimum.

```
 memo[i][j] = res;
 return res;
```

zapisuje wynik w memo i zwraca go.

```
vector<int> arr = { 2, 1, 3, 4 };
```

ustala przykładowe rozmiary macierzy.

```
int n = arr.size();
```

zapisuje rozmiar wektora w n.

```
cout << "Liczba macierzy: " << n - 1 << endl;
cout << "Rozmiary macierzy:" << endl;
```

wypisuje liczbę macierzy i nagłówek.

```
for (int i = 0; i < n - 1; i++) {
    cout << "Macierz " << i + 1 << ": " << arr[i] << " x " << arr[i + 1] << endl;
}
```

pętla po macierzach — wypisuje rozmiary każdej.

```
int resRec = matrixMultiplication(arr);
```

wywołuje wersję rekurencyjną i pokazuje wynik

```
cout << "Wynik (rekurencja): " << resRec << endl;
```

```
vector<vector<int>> memo(n, vector<int>(n, -1));
```

```
int resMemo = minMultRecMemo(arr, 0, n - 1, memo);
```

tworzy tablicę memo i uruchamia wersję z memoizacją.

```
cout << "Wynik (memoizacja): " << resMemo << endl;
return 0;
```

wypisuje wynik z memoizacji.

Zadnie 3

Dla podanego przykładu mnożenie macierzy zostało wykonane optymalnie i potwierdza, że łańcuch macierzy można obliczyć za pomocą minimalnej liczby mnożeń skalarnych (7500 zamiast 75 000). Wyniki metody rekurencyjnej i z memoizacją są zgodne.

```
 Weryfikacja przykladu z instrukcji
Liczba macierzy: 3
Rozmiary macierzy:
Macierz 1: 10 x 100
Macierz 2: 100 x 5
Macierz 3: 5 x 50
Wynik (rekurencja): 7500
Wynik (memoizacja): 7500
```

```
cout <<  " Weryfikacja przykladu z instrukcji " << endl;
```

```
vector<int> arr = { 10, 100, 5, 50 };
```

Zadnie 4

Dla 7:

Dla zadania 4 przygotowałam przykład dla 7 macierzy. Rozmiary macierzy zostały dobrane ręcznie. Wynik zarówno z metody rekurencji, jak i memoizacji jest zgodny i wynosi 39000, co potwierdza poprawność działania programu i znalezienie optymalnego nawiasowania.

```
vector<int> arr = { 10, 20, 30, 40, 30, 20, 10, 10 };
```

```
 Weryfikacja przykladu z instrukcji
Liczba macierzy: 7
Rozmiary macierzy:
Macierz 1: 10 x 20
Macierz 2: 20 x 30
Macierz 3: 30 x 40
Macierz 4: 40 x 30
Macierz 5: 30 x 20
Macierz 6: 20 x 10
Macierz 7: 10 x 10
Wynik (rekurencja): 39000
Wynik (memoizacja): 39000
```

Dla 15

przykład dla 15 macierzy. Rozmiary macierzy zostały dobrane ręcznie. Wynik zarówno z metody rekurencji, jak i memoizacji jest zgodny i wynosi 39000, co potwierdza poprawność działania programu i znalezienie optymalnego nawiasowania.

```
vector<int> arr = { 10, 15, 20, 25, 30, 35, 40, 45, 30, 35, 20, 25, 10, 15, 10, };
```

```
 Weryfikacja przykladu z instrukcji
Liczba macierzy: 14
Rozmiary macierzy:
Macierz 1: 10 x 15
Macierz 2: 15 x 20
Macierz 3: 20 x 25
Macierz 4: 25 x 30
Macierz 5: 30 x 35
Macierz 6: 35 x 40
Macierz 7: 40 x 45
Macierz 8: 45 x 30
Macierz 9: 30 x 35
Macierz 10: 35 x 20
Macierz 11: 20 x 25
Macierz 12: 25 x 10
Macierz 13: 10 x 15
Macierz 14: 15 x 10
Wynik (rekurencja): 98000
Wynik (memoizacja): 98000
```

Wnioski

Program ma kilka wad: wersja rekurencyjna bez memoizacji jest wolna dla dużej liczby macierzy, nie pokazuje kolejności nawiasowania.