Київський національний університет імені Тараса Шевченка
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем
Алгоритми та складність

Лабораторна робота №1
«Алгоритм Штрассена для множення матриць»
Варіант №1
Виконав студент 2-го курсу
Групи ІПС-21
Міцкевич Костянтин Олександрович

Київ – 2024

Завдання: Реалізувати алгоритм Штрассена для множення матриць, забезпечивши проходження юніт-тестів та виведення часу виконання програми.
Тип представлення матриці: Матриці в стилі C (T**)
Тип даних: Дійснозначні матриці

Теорія:
Квадратна матриця — це матриця з однаковою кількістю рядків і стовпців.
($n \times n$)-матриця — це квадратна матриця порядку n:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix}$$

Числа a_{ij} називаються елементами матриці. Положення кожного елемента в матриці визначається номерами рядка і стовпчика, в яких знаходиться цей елемент.

Блочна матриця — це матриця, яка поділяється на підматриці (блоки), які можуть розглядатися як окремі елементи. Це дозволяє виконувати операції над матрицями на рівні блоків, що зручно при реалізації певних алгоритмів, зокрема рекурсивних методів, як-от алгоритм Штрассена.

Стандартний метод множення матриць — за означенням, якщо $C=AB$ для матриці A розміру $n \times m$ та матриці B розміру $m \times$ p, то C є матрицею розміру $n \times$ p з елементами $c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$
Алгоритм Штрассена — алгоритм розроблений Фолькером Штрассеном у 1969 році, призначений для прискореного множення матриць і є розширенням методу множення Карацуби на матриці. Він забезпечує швидше множення порівняно з традиційним методом, що стає особливо помітним для великих матриць. На відміну від класичного алгоритму множення матриць (за формулою $c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$, який має складність $O(N^3) = O(N^{\log\_2(8)})$, алгоритм Штрассена виконує множення за час $O(N^{\log\_2(7) + o(1)}) \approx O(N^{2.8074})$. Це забезпечує значний виграш на щільних матрицях великого розміру, починаючи приблизно з розмірів 64 × 64.

Множення Карацуби — алгоритмом для швидкого множення двох n-значних чисел, який має обчислювальну складність:
$M(n) = O(n^{\log\_2(3)}\})$, де $\log_2(3) \approx 1{,}5849$.

Алгоритм:

Нехай A і B — дві квадратні матриці над кільцем R, і нам потрібно обчислити матрицю C=A*B, де A, B, C $\in R^{2n\times 2^n}$. Якщо розмір матриць не є ступенем двійки, ми доповнюємо їх нулями до найближчого розміру $2^n\times 2^n$. Це дозволяє зручно застосовувати рекурсивні методи множення, хоча і призводить до додаткових непотрібних обчислень.

Розділимо матриці A, B і C на блоки рівного розміру:

$$A = \begin{matrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{matrix}, B = \begin{matrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{matrix}, \mathrm{C} = \begin{matrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{matrix}$$

де кожен блок A_{ij}, B_{ij}, $C_{ij} \in R^{(2n-1)\times(2n-1)}$.

Звичайне множення матриць дає формули:

$C_{11} = A_{11}B_{11}+A_{12}B_{21}$; $C_{12} = A_{11}B_{12}+A_{12}B_{22}$;
$C_{21} = A_{21}B_{11}+A_{22}B_{21}$; $C_{22} = A_{21}B_{12}+A_{22}B_{22}$;

Це вимагає 8 множень, що так само, як і при звичайному алгоритмі множення. Для оптимізації, визначимо нові матриці:

$M_1 = (A_{11} + A_{22})*(B_{11} + B_{22})$;
$M_2 = (A_{21} + A_{22})*B_{11}$;
$M_3 = A_{11}*(B_{12} - B_{22})$;
$M_4 = A_{22}*(B_{21} - B_{22})$;
$M_5 = (A_{11} + A_{12})*B_{22}$;
$M_6 = (A_{21} - A_{11})*(B_{11} + B_{12})$;
$M_7=(A_{12} – A_{22})(B_{21} + B_{22})$.

За допомогою цих матриць ми можемо обчислити C_{ij} лише з 7 множеннями:

$C_{11}=M_1 + M_4 - M_5 + M_7$; $C_{12} = M_3 + M_5$;
$C_{21} = M_2 + M_4$; $C_{22} = M_1 – M_2 + M_3 + M_6$.

Процес ділення і рекурсивного множення повторюється, доки матриці не стануть достатньо малими, після чого застосовується стандартний метод множення.

Складність алгоритму:

Як вже було на відміну від класичного алгоритму множення матриць (за формулою cij = $\sum_{k=1}^{m} a_{ik} b_{kj}$, який має складність $O(N^3) = O(N^{\log_2(8)})$, алгоритм Штрассена виконує множення за час $O(N^{\log_2(7) + o(1)}) \approx O(N^{2.8074})$.

Кількість додавань і множень в алгоритмі Штрассена можна оцінити таким чином:

Нехай f(n)— це кількість операцій для матриці розміру $2^n \times 2^n$. При рекурсивному застосуванні алгоритму Штрассена маємо співвідношення $f(n)=7f(n-1) + L4^n$, де L - це стала, яка залежить від кількості додавань, виконаних на кожному кроці. Тоді $f(n)=(7+o(1))^n$, і асимптотична складність множення матриць розміру $N = 2^n$ за допомогою алгоритму Штрассена становить $O((7+o(1))^n)=O(Nl^{og\_2(7)+o(1)})\approx O(N^{2.8074})$.

Хоча алгоритм зменшує кількість арифметичних операцій, це частково впливає на числову стабільність. Крім того, він вимагає більше пам'яті порівняно з наївним методом. Обидві початкові матриці повинні бути розширені до найближчого розміру, який є ступенем двійки, що може збільшити обсяг збережених даних до чотирьох разів. Також алгоритм використовує сім допоміжних матриць, кожна з яких містить чверть елементів.

Мова реалізації алгоритму: C++

Модулі програми:

```cpp
//Функція ініціалізує квадратну матрицю розміру size на динамічній пам'яті.
double** matrix_initialize(int size){ ... }

//Функція видаляє матрицю з пам'яті.
void del_matrix(double** matrix, int size){ ... }

//Функція для обчислення суми двох матриць.
double** sum(double** matrix_A, double** matrix_B, int size){ ... }

//Функція для обчислення різниці двох матриць.
double** subtraction(double** matrix_A, double** matrix_B, int size){ ... }

//Функція для виведення матриці на екран.
void print(double** matrix, int size){ ... }

//Функція розширює матрицю до найближчого розміру, що є степенем двійки.
double** expand_matrix(double** matrix, int size){ ... }

//Функція зменшує матрицю до її початкового розміру після обчислень.
double** narrow(double** matrix, int size){ ... }

//Функція реалізує алгоритм Штрассена для множення двох квадратних матриць.
double** Strassen(double** matrix_A, double** matrix_B, int size){ ... }

//Основна функція програми для тестування алгоритму.
int main(){ ... }
```

Інтерфейс користувача:

Введення даних здійснюється у програмному коді.

Вхідні дані:
- розмір квадратної матриці, яку буде виведено у результаті
- матриці, добуток яких потрібно знайти

Вихідні дані:
- Виведення матриці, що є результатом множення двох матриць, у консоль

Тестові приклади:

1) Натуральні числа:

```
A[0][0] = 12; A[0][1] = 7;  A[0][2] = 9;  A[0][3] = 5;
A[1][0] = 15; A[1][1] = 22; A[1][2] = 17; A[1][3] = 14;
A[2][0] = 33; A[2][1] = 21; A[2][2] = 19; A[2][3] = 10;
A[3][0] = 40; A[3][1] = 28; A[3][2] = 35; A[3][3] = 16;

B[0][0] = 4;  B[0][1] = 6;  B[0][2] = 8;  B[0][3] = 9;
B[1][0] = 10; B[1][1] = 12; B[1][2] = 14; B[1][3] = 3;
B[2][0] = 18; B[2][1] = 15; B[2][2] = 2;  B[2][3] = 1;
B[3][0] = 5;  B[3][1] = 7;  B[3][2] = 11; B[3][3] = 13;
```

Вивід:

```
305 326 267 203
656 707 616 400
734 805 706 509
1150 1213 958 687
```

2) Цілі числа:

```
A[0][0] = -23;  A[0][1] = 45;  A[0][2] = 99;   A[0][3] = -47;
A[1][0] = 87;  A[1][1] = -12;  A[1][2] = 64;   A[1][3] = -35;
A[2][0] = -72;  A[2][1] = 19;  A[2][2] = -58;   A[2][3] = 31;
A[3][0] = 44;  A[3][1] = -88;  A[3][2] = 76;   A[3][3] = 53;

B[0][0] = 99;  B[0][1] = -65;  B[0][2] = 25;  B[0][3] = -1;
B[1][0] = 37;  B[1][1] = 12;  B[1][2] = -88;  B[1][3] = 46;
B[2][0] = 63;  B[2][1] = -50;  B[2][2] = 19;  B[2][3] = -9;
B[3][0] = -44;  B[3][1] = 88;  B[3][2] = -100; B[3][3] = 27;
```

Вивід:

```
7693 -7051 2046 -67
13741 -12079 7947 -2160
-11443 10536 -7674 2305
3556 -3052 4988 -3345
```

3) Числа з плаваючую крапкою:

```
A[0][0] = -12.35;  A[0][1] = 7.52;   A[0][2] = -5.22;  A[0][3] = 14;
A[1][0] = 9.89;    A[1][1] = -18.7; A[1][2] = 3.43;   A[1][3] = -4.1;
A[2][0] = -11.51;  A[2][1] = 2.1;    A[2][2] = 12.01;  A[2][3] = -9.68;
A[3][0] = 4.9;    A[3][1] = 7;   A[3][2] = -19.88; A[3][3] = 16.59;

B[0][0] = 15.33;  B[0][1] = -6.28;  B[0][2] = 8;   B[0][3] = -13.52;
B[1][0] = -9.77;  B[1][1] = 12.1;  B[1][2] = -4.88;  B[1][3] = 17.25;
B[2][0] = 6.1;   B[2][1] = -14.44; B[2][2] = 3.92;   B[2][3] = -8.11;
B[3][0] = 19.9;  B[3][1] = -1;  B[3][2] = 7;   B[3][3] = -5.71;
```

Вивід:

```
-16.0379 229.927 -57.96 259.086
273.646 -333.808 155.122 -460.694
-316.336 -66.0516 -123.009 149.712
215.6 324.405 43.2404 121
```

Висновок:

Алгоритм Штрассена є ефективним інструментом для швидкого множення великих матриць. Його використання дозволяє значно прискорити процес обчислень, особливо для матриць розміром у кілька тисяч елементів, наприклад, 2000×2000. Хоча такі великі матриці зустрічаються нечасто в повсякденних задачах, у багатьох прикладних сферах, таких як обробка зображень, наукові обчислення або машинне навчання, прискорене множення є важливим. Використання алгоритму Штрассена в таких випадках дозволяє зменшити час обчислень, що робить його цінним для складних і ресурсоємних задач.

Використані джерела:

- https://en.wikipedia.org/wiki/Strassen_algorithm
- Matrix multiplication - Wikipedia