**Algorytmy macierzowe**

**Laboratorium 1**

**Sprawozdanie**

**Łukasz Stępień, Szymon Urbański**

1. **Temat zadania**

Laboratorium polegało na zaimplementowaniu i przetestowaniu rekurencyjnych algorytmów mnożenia macierzy metodą Binét’a oraz Strassena. Należało również narysować wykres zależności czasu i ilości obliczeń od rozmiarów macierzy oraz określić złożoność obliczeniową tych algorytmów.

1. **Rozwiązanie**

**2.1. Metoda Binéta**

Korzystając z informacji przedstawionych na wykładzie zaimplementowano rekurencyjne mnożenie macierzy dla dowolnego, możliwego przypadku.

* **Pseudokod algorytmu:**

recursvie\_matrix(A, B):

if B jest puste Then Return pusta macierz (1)

if A ma jeden wiersz Or B ma jedną kolumnę Then

Return A \* B (według definicji)

Podziel macierz A na macierze A11, A12, A21, A22 (2)

Podziel macierz B na macierze B11, B12, B21, B22

C11 = dodaj recursvie\_matrix(A11, B11) i recursvie\_matrix(A12, B21) (3)

C12 = dodaj recursvie\_matrix(A11, B12) i recursvie\_matrix(A12, B22)

C21 = dodaj recursvie\_matrix(A21, B11) i recursvie\_matrix(A22, B21)

C22 = dodaj recursvie\_matrix(A21, B12) i recursvie\_matrix(A22, B22)

Połącz C11, C12, C21, C22 (4)

Return C

* **Złożoność obliczeniowa:**

Niech T(n) oznacza czas procedury *recursvie\_matrix* dla dwóch macierzy *n*x*n*.

1. W przypadku bazowym (*n=1*), wykonujemy jedno skalarne mnożenie, więc:
2. Zakładamy, że podział wykonuje się w stałym czasie .
3. Wykonujemy łącznie 8 wywołań rekurencyjnych procedury *recursvie\_matrix*. Ponieważ w każdym tym wywołaniu mnożymy macierze x , co wnosi do łącznego czasu działania, czas tych ośmiu wywołań to . Następnie uwzględniamy 4 dodawania macierzy zawierających elementów. Każde z czterech dodawań wymaga . Ponieważ liczba dodawań macierzy jest stała, łączny czas ich wykonywania to .
4. Zakładamy, że łączenie wykonuje się w stałym czasie .

Rekurencja opisująca czas działania algorytmu Bineta przedstawia się równaniem:

Rozwiązaniem tej rekurencji jest T(n) =

* **Najważniejsze fragmenty kodu:**

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Fragment kodu odpowiedzialny za dodanie do siebie dwóch macierzy.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Fragment kodu odpowiedzialny za wykonanie mnożenia w przypadkach granicznych.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Fragment kodu odpowiedzialny za podzielenie macierzy na 4 części.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Fragment kodu odpowiedzialny za wyznaczenie wyniku mnożenia i połączenie składowych wynikowej macierzy.

**2.2. Metoda Strassena**

Zaimplementowano także rekurencyjne mnożenie macierzy o wymiarach NxN, gdzie N jest potęgą liczby 2 z wykorzystaniem metody Strassena.

* **Pseudokod algorytmu:**

recursvie\_matrix\_strassen(A, B):

if B jest puste Then Return pusta macierz (1)

if A ma jeden wiersz Or B ma jedną kolumnę Then

Return A \* B (według definicji)

Podziel macierz A na macierze A11, A12, A21, A22 (2)

Podziel macierz B na macierze B11, B12, B21, B22

Oblicz rekurencyjnie P1, P2, ..., P7 (3)

Oblicz C11, C12, C21, C22 (4)

Połącz C11, C12, C21, C22 (5)

Return C

* **Złożoność obliczeniowa:**

Niech T(n) oznacza czas procedury *recursvie\_matrix\_strassen* dla dwóch macierzy *n*x*n*.

1. W przypadku bazowym (*n=1*), wykonujemy jedno skalarne mnożenie, więc:
2. Zakładamy, że podział wykonuje się w stałym czasie .
3. Wykonujemy łącznie 7 wywołań rekurencyjnych procedury *recursvie\_matrix*. Ponieważ w każdym tym wywołaniu mnożymy macierze x , co wnosi do łącznego czasu działania, czas tych siedmiu wywołań to .
4. Uwzględniamy dodawanie i odejmowanie macierzy zawierających elementów. Każde z tych działań wymaga , a ich liczba jest stała. Oznacza to, że łączny czas ich wykonywania to .
5. Zakładamy, że łączenie wykonuje się w stałym czasie .

Rekurencja opisująca czas działania algorytmu Strassena przedstawia się równaniem:

Rozwiązaniem tej rekurencji jest .

* **Najważniejsze fragmenty kodu:**

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Fragment kodu odpowiedzialny za odejmowanie dwóch macierzy.

Obraz zawierający tekst, Czcionka, zrzut ekranu, algebra

Opis wygenerowany automatycznie

Fragment kodu odpowiedzialny za kolejne

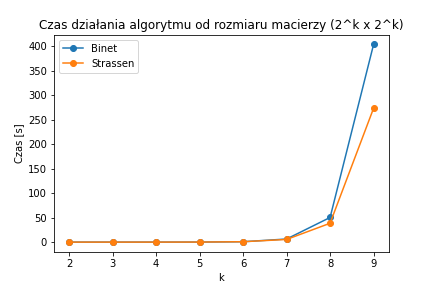
Obraz zawierający tekst, Czcionka, zrzut ekranu

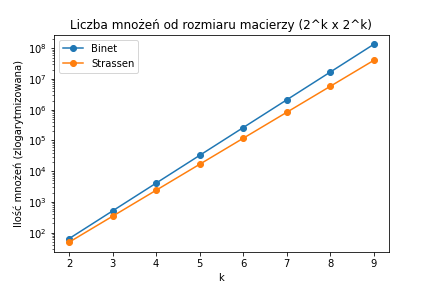
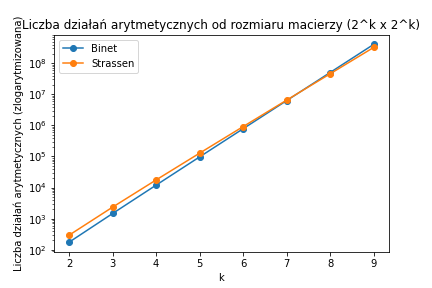
Opis wygenerowany automatycznie

Fragment kodu odpowiedzialny za wyznaczenie wyniku mnożenia.

W implementacji tego algorytmu fragmenty kodu odpowiedzialne za dodawanie macierzy, warunki brzegowe rekurencji, podział macierzy na cztery części oraz łączenie wyniku zaimplementowano tak samo jak w algorytmie Binét’a.

1. **Wyniki**

Na poniższym wykresie przedstawiono zależność czasu wyznaczania wyniku mnożenia macierzy w zależności od ilości elementów tej macierzy.

Obraz zawierający tekst, zrzut ekranu, linia, Wykres

Opis wygenerowany automatycznieAnalogicznie wygenerowano wykres przedstawiający liczbę wykonanych operacji arytmetycznych w zależności od wielkości macierzy.

Dodatkowo dla dwóch macierzy 2x2:

zaimplementowane algorytmy zwróciły następujące wyniki:

* Binet:
* Strassen:

Iloczyn ten przeliczono również w programie *MATLAB,* który zwrócił wynik:

1. **Wnioski**

Patrząc na wykresy można stwierdzić, że szybszy czas wykonania mnożenia macierzy charakteryzuje metoda Strassena. Liczba operacji arytmetycznych w miarę zwiększania rozmiaru macierzy ma większą tendencję spadkową niż w przypadku algorytmu Binéta.

* liczba dodawań: dla małych k mniej operacji wykonuje algorytm Binéta, lecz przy k>=9 tę cechę wykazuje algorytm Strassena
* liczba mnożeń: dla wszystkich k algorytm Strassena wykonuje mniej operacji niż algorytm Binéta
* liczba wszystkich działań arytmetycznych: dla k<8 mniej działań wykonuje algorytm Binéta, lecz dla k>=8 algorytm Strassena wychodzi na prowadzenie

Złożoność obliczeniowa w zależności od rozmiaru macierzy (*nxn*) dla obu algorytmów przedstawia się następująco:

* Binét: ,
* Strassen: .

Przy wyprowadzaniu tych złożoności w celu ułatwienia rachunków założono, że dzielenie oraz łączenie macierzy odbywa się w czasie stałym. Jednak zaimplementowane tutaj algorytmy nie posiadają tej cechy, gdyż w każdym wywołaniu rekurencyjnym tworzymy nowe macierze. Fakt ten może wpływać na nieefektywne wykonywanie obu algorytmów, ponieważ już przy macierzach o rozmiarach czas ten wynosi powyżej czterech minut. Dodatkowo czas ten może wydłużać implementacja algorytmów na listach w języku Python, które cechują się wolnym czasem operacji.

Porównując iloczyny dwóch macierzy o wymiarach 2x2 wyliczonymi przez oba algorytmy oraz program MATLAB można stwierdzić, że z dokładnością do błędów numerycznych wszystkie trzy iloczyny są identyczne. Można z tego wywnioskować, że zaimplementowane algorytmy działają poprawnie.

1. **Bibliografia**

* wykład z przedmiotu „Algorytmy macierzowe” przygotowany przez prof. dr hab. Macieja Paszyńskiego
* <https://en.wikipedia.org/wiki/Strassen_algorithm>
* Thomas H. Cormen - „Wprowadzenie do algorytmów”