I zestaw zadań - Mnożenie macierzy

Kacper Kozubowski, Mateusz Podmokły
III rok Informatyka WI

2 październik 2024

1 Treść zadania

Należy wygenerować macierze losowe o wartościach z przedziału otwartego $(10^{-8}, 1.0)$ i zaimplementować

1. Rekurencyjne mnożenie macierzy metodą Binét'a
2. Rekurencyjne mnożenie macierzy metodą Strassena
3. Mnożenie macierzy metodą AI na podstawie artykułu w Nature

Proszę zliczać liczbę operacji zmienno-przecinkowych wykonywanych podczas mnożenia macierzy.

2 Specyfikacja użytego środowiska

Specyfikacja:

- Środowisko: Jupyter Notebook,
- Język programowania: Python,
- System operacyjny: Microsoft Windows 11,
- Architektura systemu: x64.

3 Działanie algorytmów

3.1 Wykorzystane biblioteki

W realizacji rozwiązania wykorzystane zostały następujące biblioteki:

```
import numpy as np
import matplotlib.pyplot as plt
import time
```

3.2 Pseudokod

Algorithm 1 Algorytm Binét'a

```
Input: A, B
Output: C
function BINET(A, B)
    n = size(A)
    if n = 1 then
        return A * B
    end if

    mid = n div 2
    A11 = A[1:n, 1:n] // Górny lewy blok
    A12 = A[1:n, n+1:2n] // Górny prawy blok
    A21 = A[n+1:2n, 1:n] // Dolny lewy blok
    A22 = A[n+1:2n, n+1:2n] // Dolny prawy blok
    B11 = B[1:n, 1:n]
    B12 = B[1:n, n+1:2n]
    B21 = B[n+1:2n, 1:n]
    B22 = B[n+1:2n, n+1:2n]

    C11 = BINET(A11, B11) + BINET(A12, B21)
    C12 = BINET(A11, B12) + BINET(A12, B22)
    C21 = BINET(A21, B11) + BINET(A22, B21)
    C22 = BINET(A21, B12) + BINET(A22, B22)

    C[1:n/2, 1:n/2] = C1 // Górny lewy blok
    C[1:n/2, n/2+1:n] = C2 // Górny prawy blok
    C[n/2+1:n, 1:n/2] = C3 // Dolny lewy blok
    C[n/2+1:n, n/2+1:n] = C4 // Dolny prawy blok
    return C
end function
```

Algorithm 2 Algorytm Strassen'a

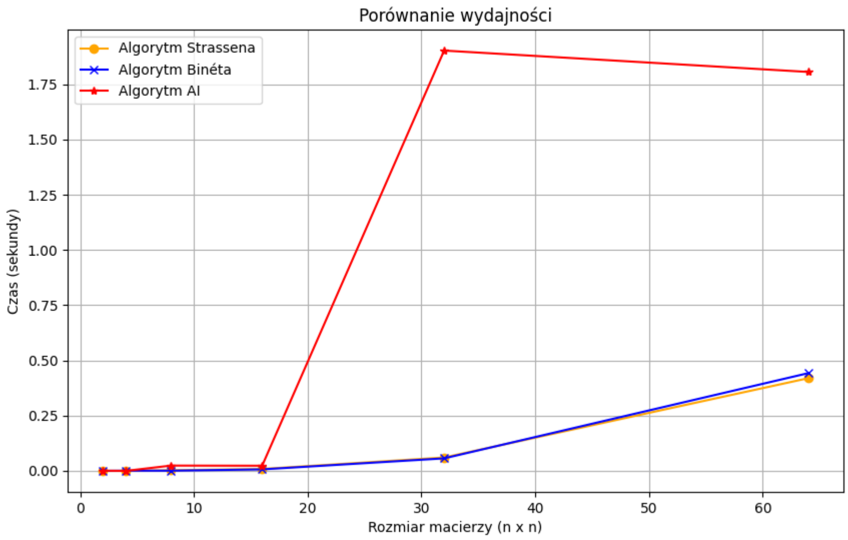
```
Input: A, B
Output: C
function STRASSEN(A, B)
    n = size(A)
    if n = 1 then
        return A * B
    end if

    mid = n div 2
    A11 = A[1:n, 1:n] // Górny lewy blok
    A12 = A[1:n, n+1:2n] // Górny prawy blok
    A21 = A[n+1:2n, 1:n] // Dolny lewy blok
    A22 = A[n+1:2n, n+1:2n] // Dolny prawy blok
    B11 = B[1:n, 1:n]
    B12 = B[1:n, n+1:2n]
    B21 = B[n+1:2n, 1:n]
    B22 = B[n+1:2n, n+1:2n]

    P1 = STRASSEN(A11 + A22, B11 + B22)
    P2 = STRASSEN(A21 + A22, B11)
    P3 = STRASSEN(A11, B12 - B22)
    P4 = STRASSEN(A22, B21 - B11)
    P5 = STRASSEN(A11 + A12, B22)
    P6 = STRASSEN(A21 - A11, B11 + B12)
    P7 = STRASSEN(A12 - A22, B21 + B22)

    C11 = P1 + P4 - P5 + P7
    C12 = P3 + P5
    C21 = P2 + P4
    C22 = P1 + P3 - P2 + P6

    C[1:n/2, 1:n/2] = C1 // Górny lewy blok
    C[1:n/2, n/2+1:n] = C2 // Górny prawy blok
    C[n/2+1:n, 1:n/2] = C3 // Dolny lewy blok
    C[n/2+1:n, n/2+1:n] = C4 // Dolny prawy blok
    return C
end function
```

4 Porównanie działania algorytmów

4.1 Czas działania

Czas działania algorytmów dla $n = 2^m$, gdzie

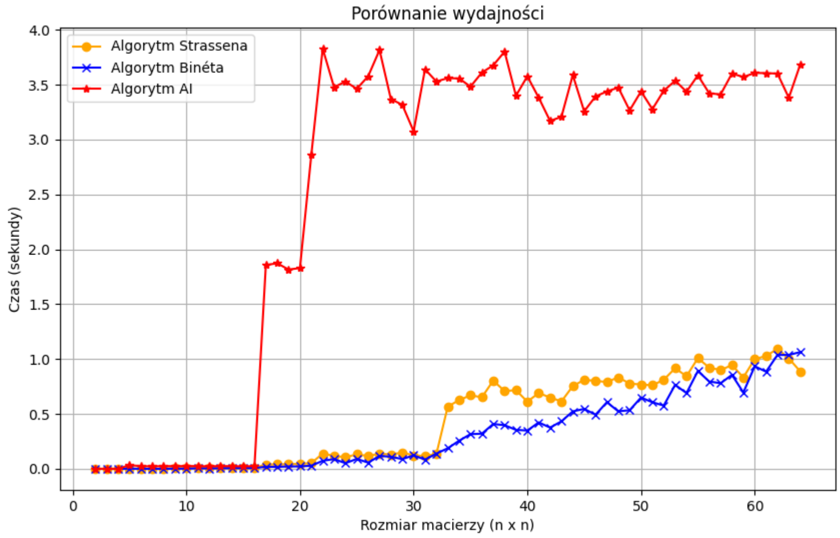
$$m \in \{2, 4, 8, 16, 32, 64\}$$



Rysunek 1: Porównanie czasu działania.

Czas działania algorytmów dla $n = 2^m$, gdzie

$$m \in [2, 64]$$

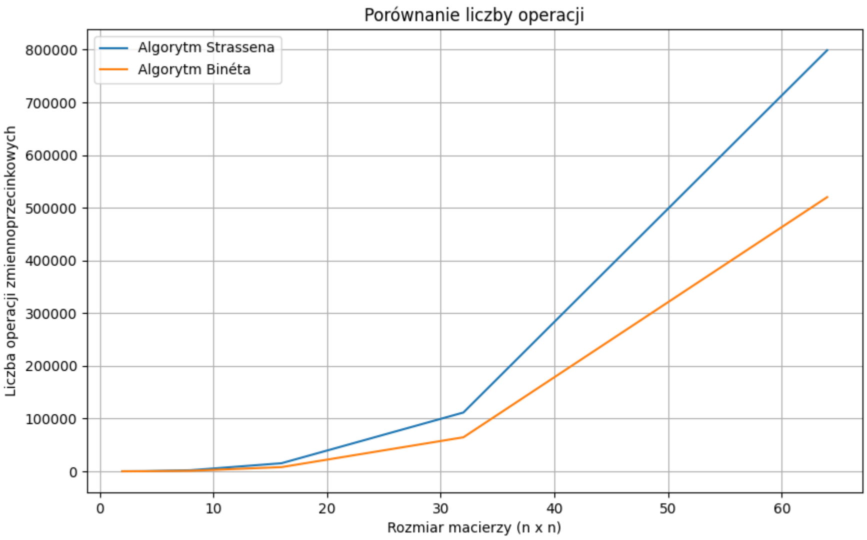


Rysunek 2: Porównanie czasu działania.

4.2 Liczba operacji zmienno-przecinkowych

Liczba operacji zmienno-przecinkowych wykonanych podczas działania algorytmów dla $n = 2^m$, gdzie

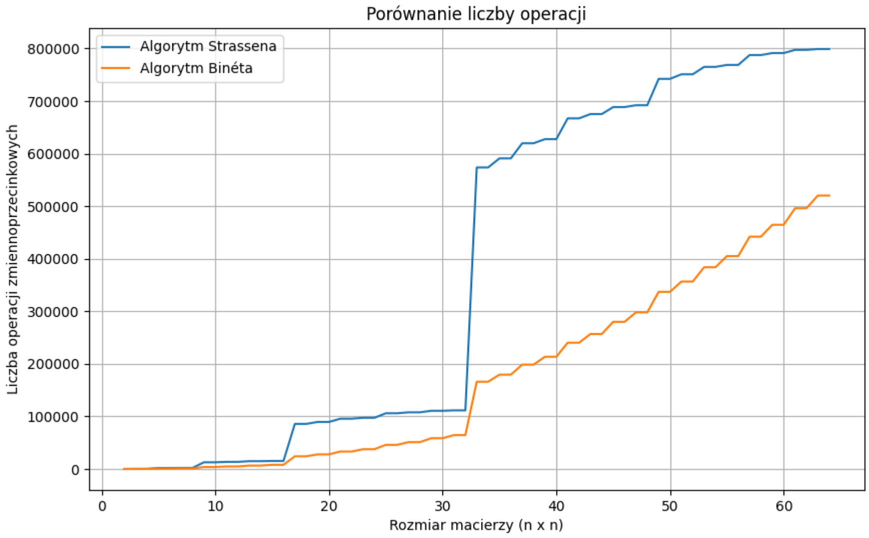
$$m \in \{2, 4, 8, 16, 32, 64\}$$



Rysunek 3: Porównanie liczby operacji zmienno-przecinkowych.

Liczba operacji zmienno-przecinkowych wykonanych podczas działania algorytmów dla $n = 2^m$, gdzie

$$m \in [2, 64]$$



Rysunek 4: Porównanie liczby operacji zmienno-przecinkowych.

5 Oszacowanie złożoności obliczeniowej

Dla każdego algorytmu zmierzony został czas obliczeń dla następujących wielkości macierzy:

$$n \in \{2, 4, 8, 16, 32, 64, 128\}$$

oraz dopasowana krzywa złożoności obliczeniowej.

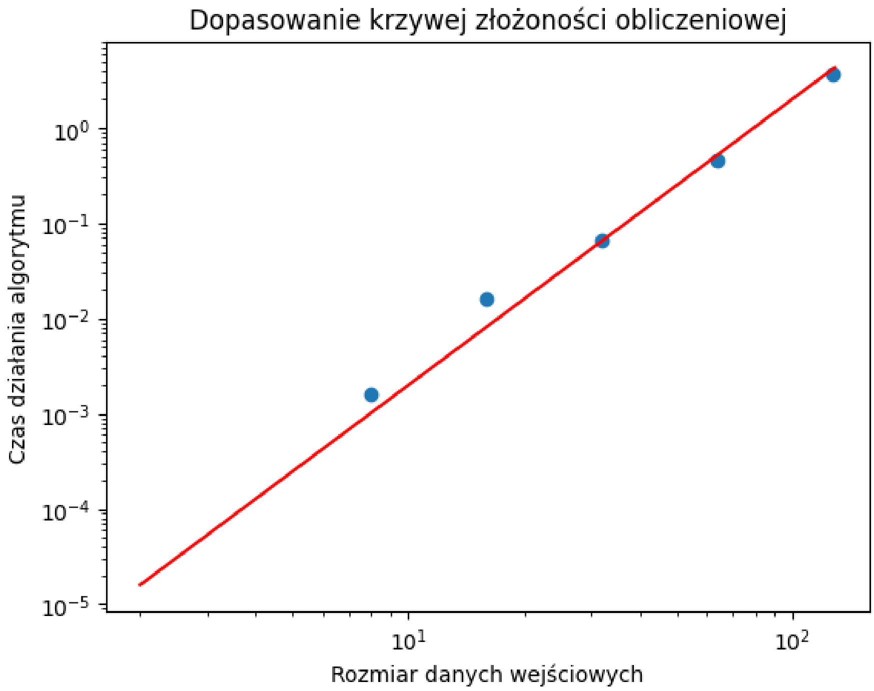
5.1 Algorytm Binét'a

Dopasowana krzywa:

$$y = 2 * 10^{-6} * x^3$$

Zatem złożoność wynosi

$$O(n) \approx n^3$$



Rysunek 5: Krzywa dopasowana w skali logarytmicznej dla algorytmu Binét'a.

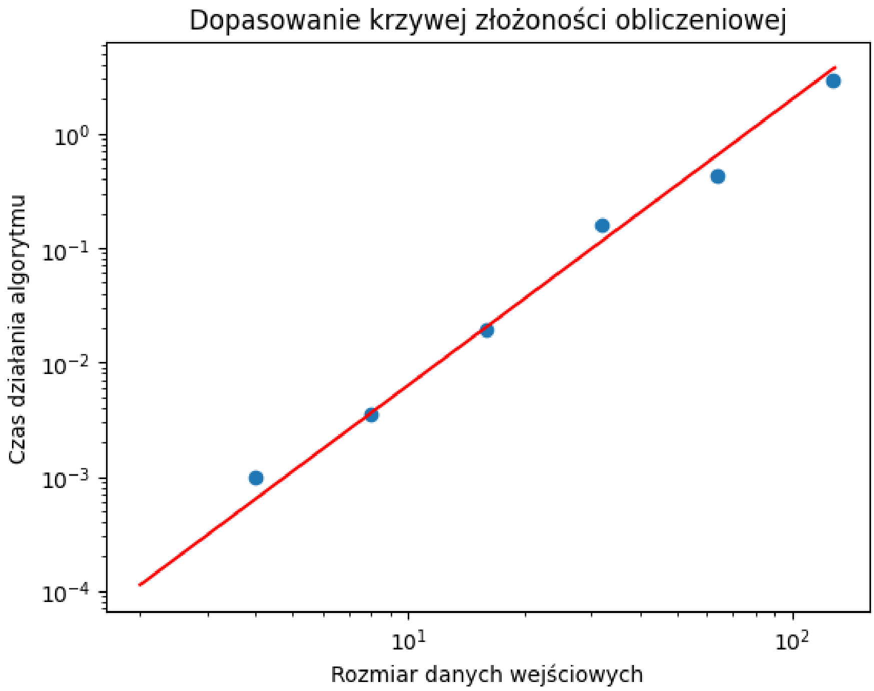
5.2 Algorytm Strassena

Dopasowana krzywa:

$$y = 2 * 10^{-5} * x^{2.5}$$

Zatem złożoność wynosi

$$O(n) \approx n^{2.5}$$



Rysunek 6: Krzywa dopasowana w skali logarytmicznej dla algorytmu Strassena.

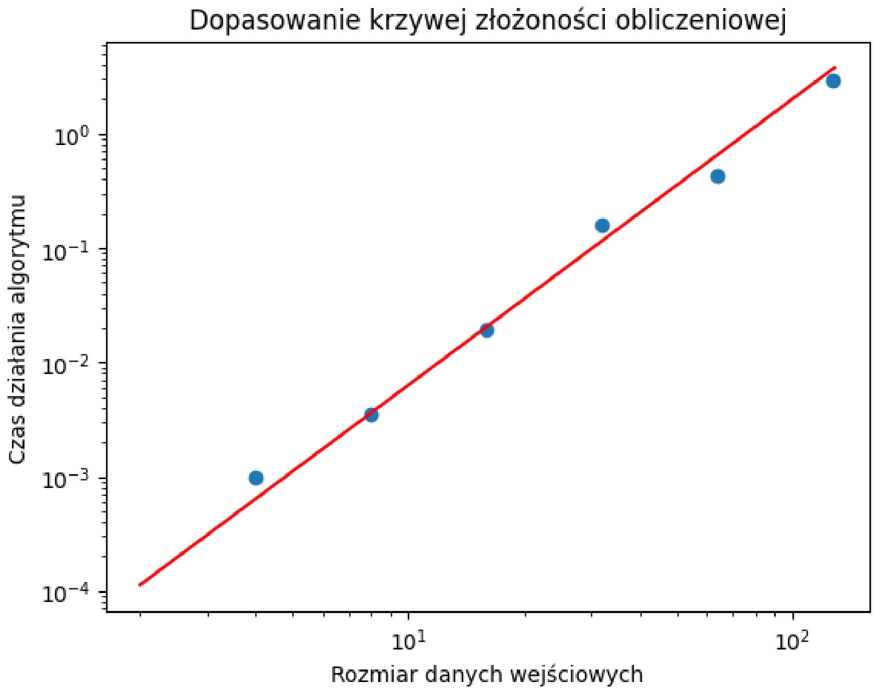
5.3 Algorytm AI

Dopasowana krzywa:

$$y = 1.8 * 10^{-5} * x^3$$

Zatem złożoność wynosi

$$O(n) \approx n^3$$



Rysunek 7: Krzywa dopasowana w skali logarytmicznej dla algorytmu AI.