Algorytmy Macierzowe

Sprawozdanie nr 1

23.10.2024

Mateusz Król, Natalia Bratek

gr. 3

Spis treści

1. Polecenie

Proszę wybrać ulubiony język programowania, wygenerować macierze losowe o wartościach z przedziału otwartego (0.00000001, 1.0) i zaimplementować:

- Rekurencyjne mnożenie macierzy metodą *Binét*'a
- Rekurencyjne mnożenie macierzy metodą *Strassen*'a
- Mnożenie macierzy metodą AI na podstawie artykułu w *Nature*

Proszę zliczać liczbę operacji zmiennoprzecinkowych (+-*/ liczb) wykonywanych podczas mnożenia macierzy.

2. Algorytm Binet'a

2.1 Opis algorytmu

Zakładamy, że mamy dwie macierze A i B o wymiarach n x n. Chcemy wyznaczyć macierz C, będącą iloczynem macierzy A i B, czyli C = A ·B. Algorytm Binet'a polega na rozbiciu macierzy na mniejsze bloki, a następnie mnożeniu tych bloków i sumowaniu ich zgodnie z zasadami mnożenia macierzy. Po wykonaniu obliczeń dla wszystkich bloków, wyniki są łączone w macierz C. W

naszej implementacji wykorzystujemy mechanizm *dynamic peeling*:

Another approach, called **dynamic peeling**, deals with odd dimensions by stripping off the extra row and/or column as needed, and adding their contributions to the final result in a later round of fixup work. More specifically, let A be an $m \times k$ matrix and B be a $k \times n$ matrix. Assuming that m, k, and n are all odd, A and B are partitioned into the block matrices

$$A = \left(\begin{array}{c|c} A_{11} & a_{12} \\ \hline a_{21} & a_{22} \end{array}\right) \quad \text{and} \quad B = \left(\begin{array}{c|c} B_{11} & b_{12} \\ \hline b_{21} & b_{22} \end{array}\right),$$

where A_{11} is an $(m-1) \times (k-1)$ matrix, a_{12} is a $(m-1) \times 1$ matrix, a_{21} is a $1 \times (k-1)$ matrix, a_{22} is a 1×1 matrix and B_{11} is an $(k-1) \times (n-1)$ matrix, b_{12} is a $(k-1) \times 1$ matrix, b_{21} is a $1 \times (n-1)$ matrix, b_{22} is a 1×1 matrix. The product $C = AB$ is computed as

$$\left(\begin{array}{c|c} C_{11} & c_{12} \\ \hline c_{21} & c_{22} \end{array}\right) = \left(\begin{array}{c|c} A_{11}B_{11} + a_{12}b_{21} & A_{11}b_{12} + a_{12}b_{22} \\ \hline a_{21}B_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{array}\right), \tag{9}$$

where $A_{11}B_{11}$ is computed using Strassen's algorithm, and the other computations constitute

2.2 Pseudokod

```
    binet(A, B)
        Jeśli macierz A ma rozmiar 1
            zwróć wynik A * B
        Jeśli rozmiar macierzy A jest nieparzysty
            wykonaj podział macierzy A i B na podmacierze dynamiczne A11, A12, A21, A22
oraz B11, B12, B21, B22 o rozmiarach (n-1) x (n-1), (n-1) x 1, 1 x (n-1), 1 x 1
odpowiednio, gdzie n to rozmiar macierzy A i B
            oblicz pomocnicze macierze:
                C11 = Binet(A11, B11) + standardowe_mnożenie_macierzy(A12, B21)
                C12 = standardowe_mnożenie_macierzy(A11, B12) +
standardowe_mnożenie_macierzy(A12, B22)
                C21 = dodaj(standardowe_mnożenie_macierzy(A21, B11) +
standardowe_mnożenie_macierzy(A22, B21)
                C22 = dodaj(standardowe_mnożenie_macierzy(A21, B12) +
standardowe_mnożenie_macierzy(A22, B22)
            zwróć połączone macierze C11, C12, C21, C22

        W przeciwnym przypadku (n jest parzyste):
            wykonaj podział macierzy A i B na równe 4 bloki: A11, A12, A21, A22 oraz B11,
B12, B21, B22

            oblicz pomocnicze macierze:
            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)

            zwróć połączone bloki (C11, C12, C21, C22)
```

2.3 Fragment kodu

```
def __binet(self, A, B):
    n = len(A)
    if n == 1:
        return self.calculator.standard_matrix_multiplication(A, B)
    if n % 2 == 1:
        A11, A12, A21, A22 = self.calculator.split_into_block_matrices_dynamic_peeling(A)
        B11, B12, B21, B22 = self.calculator.split_into_block_matrices_dynamic_peeling(B)

        C11 = self.calculator.add(self.__binet(A11, B11),
self.calculator.standard_matrix_multiplication(A12, B21))
        C12 = self.calculator.add(self.calculator.standard_matrix_multiplication(A11,
B12), self.calculator.standard_matrix_multiplication(A12, B22))
        C21 = self.calculator.add(self.calculator.standard_matrix_multiplication(A21,
B11), self.calculator.standard_matrix_multiplication(A22, B21))
        C22 = self.calculator.add(self.calculator.standard_matrix_multiplication(A21,
B12), self.calculator.standard_matrix_multiplication(A22, B22))

        return self.calculator.connect_block_matrices_dynamic_peeling(C11, C12, C21, C22)
    else:
        A11, A12, A21, A22 = self.calculator.split_into_block_matrices(A)
        B11, B12, B21, B22 = self.calculator.split_into_block_matrices(B)

        C11 = self.calculator.add(self.__binet(A11, B11), self.__binet(A12, B21))
        C12 = self.calculator.add(self.__binet(A11, B12), self.__binet(A12, B22))
        C21 = self.calculator.add(self.__binet(A21, B11), self.__binet(A22, B21))
        C22 = self.calculator.add(self.__binet(A21, B12), self.__binet(A22, B22))

        return self.calculator.connect_block_matrices(C11, C12, C21, C22)
```

3. Algorytm Strassen'a

3.1 Opis algorytmu

Algorytm Strassena polega na podzieleniu macierzy A i B na 4 podmacierze. Tworzone są pomocnicze macierze P1,P2, P3, P4, P5, P6, P7 poprzez dodawanie, odejmowanie i mnożenie odpowiednich podmacierzy. Wynikowa macierz C jest konstruowana na podstawie pomocniczych macierzy P1, P2, P3, P4, P5, P6, P7. Te macierze pośrednie są wykorzystywane do obliczenia poszczególnych bloków macierzy wynikowej C11, C12, C21, C22. Po obliczeniu tych bloków, są one łączone w pełną macierz wynikową C.

3.2 Pseudokod

```
    strassen(matrix_1, matrix_2)
        Jeśli macierz matrix_1 ma rozmiar 1
            to zwróć matrix_1 * matrix_2

        W przeciwnym przypadku:
            wykonaj podział macierzy matrix_1 i matrix_2 na równe 4 bloki:
            matrix_1_11, matrix_1_12, matrix_1_21, matrix_1_22 oraz matrix_2_11,
matrix_2_12, matrix_2_21, matrix_2_22

            oblicz pomocnicze macierze:
            p_1 = strassen(matrix_1_11+matrix_1_22 , matrix_2_11+matrix_2_22)
            p_2 = strassen(matrix_1_21 + matrix_1_22, matrix_2_11)
            p_3 = strassen(matrix_1_11,matrix_2_12 - matrix_2_22)
            p_4 = strassen(matrix_1_22,matrix_2_21 - matrix_2_11)
            p_5 = strassen(matrix_1_11+matrix_1_12, matrix_2_22)
            p_6 = strassen(matrix_1_21 - matrix_1_1, matrix_2_11+matrix_2_12)
            p_7 = strassen(matrix_1_12 - matrix_1_22, matrix_2_21 + matrix_2_22)
```

```
            oblicz bloki wynikowej macierzy:
            matrix_3_11 = p_1 + p_4 -  p_5 + p_7
            matrix_3_12 = p_3 + p_5
            matrix_3_21 = p_2 + p_4
            matrix_3_22 = p_1 + p_3 - p_2 + p_6

            połącz bloki (matrix_3_11, matrix_3_12, matrix_3_21, matrix_3_22) w jedną
macierz matrix_3

            zwróć matrix_3
```

3.3 Fragment kodu

```python
def __rec(self, matrix_1, matrix_2):
    if len(matrix_1) == 1:
        return self.calculator.multiply_one_by_one_matrices(matrix_1, matrix_2)

    matrix_1_11, matrix_1_12, matrix_1_21, matrix_1_22 =
self.calculator.split_into_block_matrices(matrix_1)
    matrix_2_11, matrix_2_12, matrix_2_21, matrix_2_22 =
self.calculator.split_into_block_matrices(matrix_2)

    p_1 = self.__rec(self.calculator.add(matrix_1_11, matrix_1_22),
self.calculator.add(matrix_2_11, matrix_2_22))
    p_2 = self.__rec(self.calculator.add(matrix_1_21, matrix_1_22), matrix_2_11)
    p_3 = self.__rec(matrix_1_11, self.calculator.subtract(matrix_2_12, matrix_2_22))
    p_4 = self.__rec(matrix_1_22, self.calculator.subtract(matrix_2_21, matrix_2_11))
    p_5 = self.__rec(self.calculator.add(matrix_1_11, matrix_1_12), matrix_2_22)
    p_6 = self.__rec(self.calculator.subtract(matrix_1_21, matrix_1_11),
self.calculator.add(matrix_2_11, matrix_2_12))
    p_7 = self.__rec(self.calculator.subtract(matrix_1_12, matrix_1_22),
self.calculator.add(matrix_2_21, matrix_2_22))

    matrix_3_11 = self.calculator.add(self.calculator.subtract(self.calculator.add(p_1,
p_4), p_5), p_7)
    matrix_3_12 = self.calculator.add(p_3, p_5)
    matrix_3_21 = self.calculator.add(p_2, p_4)
    matrix_3_22 = self.calculator.add(self.calculator.add(self.calculator.subtract(p_1,
p_2), p_3), p_6)

    matrix_3 = self.calculator.connect_block_matrices(matrix_3_11, matrix_3_12,
matrix_3_21, matrix_3_22)
    return matrix_3
```

4. Algorytm zaproponowany przez sztuczną inteligencję

4.1 Opis algorytmu

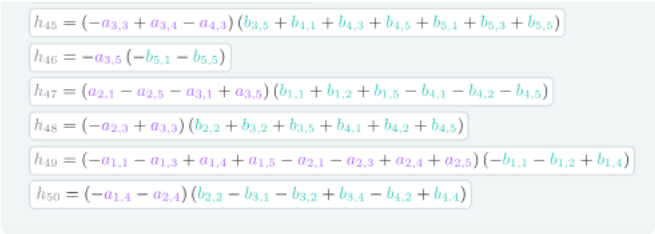
Algorytm działa w oparciu o uczenie maszynowe, gdzie proces odkrywania nowych, wydajnych algorytmów mnożenia macierzy został przekształcony w grę jednoosobową. W tej grze stan początkowy to trójwymiarowy tensor, który reprezentuje aktualny stan algorytmu. Gracz (czyli model) wykonuje dozwolone ruchy, które odpowiadają instrukcjom algorytmicznym, aby doprowadzić tensor do stanu zerowego. Ostatecznym celem jest znalezienie poprawnego algorytmu mnożenia macierzy, który jest oceniany na podstawie liczby wykonanych kroków. Grę tę rozwiązywał agent AlphaTensor, który na początku nie posiadał wiedzy o istniejących algorytmach. Poprzez uczenie wzmacniające, AlphaTensor odkrywał coraz lepsze algorytmy, w tym ponownie odkrywał znane algorytmy, takie jak Strassen. Tradycyjny algorytm mnoży macierz 4x5 przez 5x5 za pomocą 100

mnożeń. Liczbę tę udało się zmniejszyć do 80 za pomocą metody Strassena, a AlphaTensor znalazł algorytmy, które wykonują tę samą operację, wykonując 76 mnożeń.

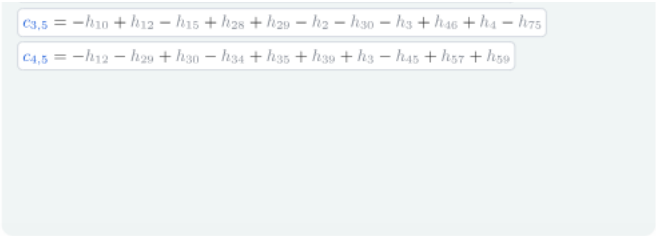
$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} \end{bmatrix} \times \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} \\ b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} & b_{4,5} \\ b_{5,1} & b_{5,2} & b_{5,3} & b_{5,4} & b_{5,5} \end{bmatrix} = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} & c_{1,5} \\ c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} & c_{2,5} \\ c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} & c_{3,5} \\ c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} & c_{4,5} \end{bmatrix}$$

$$h_{1} = a_{3,2}\left(-b_{2,1} - b_{2,5} - b_{3,1}\right)$$
$$h_{2} = \left(a_{2,2} + a_{2,5} - a_{3,5}\right)\left(-b_{2,5} - b_{5,1}\right)$$
$$h_{3} = \left(-a_{3,1} - a_{4,1} + a_{4,2}\right)\left(-b_{1,1} + b_{2,5}\right)$$
$$h_{4} = \left(a_{1,2} + a_{1,4} + a_{3,4}\right)\left(-b_{2,5} - b_{4,1}\right)$$
$$h_{5} = \left(a_{1,5} + a_{2,2} + a_{2,5}\right)\left(-b_{2,4} + b_{5,1}\right)$$
$$h_{6} = \left(-a_{2,2} - a_{2,5} - a_{4,5}\right)\left(b_{2,3} + b_{5,1}\right)$$
$$h_{7} = \left(-a_{1,1} + a_{4,1} - a_{4,2}\right)\left(b_{1,1} + b_{2,4}\right)$$
$$h_{8} = \left(a_{3,2} - a_{3,3} - a_{4,3}\right)\left(-b_{2,3} + b_{3,1}\right)$$
$$h_{9} = \left(-a_{1,2} - a_{1,4} + a_{4,4}\right)\left(b_{2,3} + b_{4,1}\right)$$
$$h_{10} = \left(a_{2,2} + a_{2,5}\right)b_{5,1}$$
$$h_{11} = \left(-a_{2,1} - a_{4,1} + a_{4,2}\right)\left(-b_{1,1} + b_{2,2}\right)$$
$$h_{12} = \left(a_{4,1} - a_{4,2}\right)b_{1,1}$$
$$h_{13} = \left(a_{1,2} + a_{1,4} + a_{2,4}\right)\left(b_{2,2} + b_{4,1}\right)$$
$$h_{14} = \left(a_{1,3} - a_{3,2} + a_{3,3}\right)\left(b_{2,4} + b_{3,1}\right)$$
$$h_{15} = \left(-a_{1,2} - a_{1,4}\right)b_{4,1}$$
$$h_{16} = \left(-a_{3,2} + a_{3,3}\right)b_{3,1}$$
$$h_{17} = \left(a_{1,2} + a_{1,4} - a_{2,1} + a_{2,2} - a_{2,3} + a_{2,4} - a_{3,2} + a_{3,3} - a_{4,1} + a_{4,2}\right)b_{2,2}$$
$$h_{18} = a_{2,1}\left(b_{1,1} + b_{1,2} + b_{5,2}\right)$$
$$h_{19} = -a_{2,3}\left(b_{3,1} + b_{3,2} + b_{5,2}\right)$$
$$h_{20} = \left(-a_{1,5} + a_{2,1} + a_{2,3} - a_{2,5}\right)\left(-b_{1,1} - b_{1,2} + b_{1,4} - b_{5,2}\right)$$
$$h_{21} = \left(a_{2,1} + a_{2,3} - a_{2,5}\right)b_{5,2}$$
$$h_{22} = \left(a_{1,3} - a_{1,4} - a_{2,4}\right)\left(b_{1,1} + b_{1,2} - b_{1,4} - b_{3,1} - b_{3,2} + b_{3,4} + b_{4,4}\right)$$
$$h_{23} = a_{1,3}\left(-b_{3,1} + b_{3,4} + b_{4,4}\right)$$
$$h_{24} = a_{1,5}\left(-b_{4,4} - b_{5,1} + b_{5,4}\right)$$
$$h_{25} = -a_{1,1}\left(b_{1,1} - b_{1,4}\right)$$
$$h_{26} = \left(-a_{1,3} + a_{1,4} + a_{1,5}\right)b_{4,4}$$
$$h_{27} = \left(a_{1,3} - a_{3,1} + a_{3,3}\right)\left(b_{1,1} - b_{1,4} + b_{1,5} + b_{3,5}\right)$$
$$h_{28} = -a_{3,4}\left(-b_{3,5} - b_{4,1} - b_{4,5}\right)$$
$$h_{29} = a_{3,1}\left(b_{1,1} + b_{1,5} + b_{3,5}\right)$$
$$h_{30} = \left(a_{3,1} - a_{3,3} + a_{3,4}\right)b_{3,5}$$
$$h_{31} = \left(-a_{1,4} - a_{1,5} - a_{3,4}\right)\left(-b_{4,4} - b_{5,1} + b_{5,4} - b_{5,5}\right)$$
$$h_{32} = \left(a_{2,1} + a_{4,1} + a_{4,4}\right)\left(b_{1,3} - b_{4,1} - b_{4,2} - b_{4,3}\right)$$
$$h_{33} = a_{4,3}\left(-b_{3,1} - b_{3,3}\right)$$
$$h_{34} = a_{4,4}\left(-b_{1,3} + b_{4,1} + b_{4,3}\right)$$
$$h_{35} = -a_{4,5}\left(b_{1,3} + b_{5,1} + b_{5,3}\right)$$
$$h_{36} = \left(a_{2,3} - a_{2,5} - a_{4,5}\right)\left(b_{3,1} + b_{3,2} + b_{3,3} + b_{5,2}\right)$$
$$h_{37} = \left(-a_{4,1} - a_{4,4} + a_{4,5}\right)b_{1,3}$$
$$h_{38} = \left(-a_{2,3} - a_{3,1} + a_{3,3} - a_{3,4}\right)\left(b_{3,5} + b_{4,1} + b_{4,2} + b_{4,5}\right)$$
$$h_{39} = \left(-a_{3,1} - a_{4,1} - a_{4,4} + a_{4,5}\right)\left(b_{1,3} + b_{5,1} + b_{5,3} + b_{5,5}\right)$$
$$h_{40} = \left(-a_{1,3} + a_{1,4} + a_{1,5} - a_{4,4}\right)\left(-b_{3,1} - b_{3,3} + b_{3,4} + b_{4,4}\right)$$
$$h_{41} = \left(-a_{1,1} + a_{4,1} - a_{4,5}\right)\left(b_{1,3} + b_{3,1} + b_{3,3} - b_{3,4} + b_{5,1} + b_{5,3} - b_{5,4}\right)$$
$$h_{42} = \left(-a_{2,1} + a_{2,5} - a_{3,5}\right)\left(-b_{1,1} - b_{1,2} - b_{1,5} + b_{4,1} + b_{4,2} + b_{4,5} - b_{5,2}\right)$$
$$h_{43} = a_{2,4}\left(b_{4,1} + b_{4,2}\right)$$
$$h_{44} = \left(a_{2,3} + a_{3,2} - a_{3,3}\right)\left(b_{2,2} - b_{3,1}\right)$$

$$h_{51} = a_{2,2}\left(b_{2,1} + b_{2,2} - b_{5,1}\right)$$
$$h_{52} = a_{4,2}\left(b_{1,1} + b_{2,1} + b_{2,3}\right)$$
$$h_{53} = -a_{1,2}\left(-b_{2,1} + b_{2,4} + b_{4,1}\right)$$
$$h_{54} = \left(a_{1,2} + a_{1,4} - a_{2,2} - a_{2,5} - a_{3,2} + a_{3,3} - a_{4,2} + a_{4,3} - a_{4,4} - a_{4,5}\right)b_{2,3}$$
$$h_{55} = \left(a_{1,4} - a_{4,4}\right)\left(-b_{2,3} + b_{3,1} + b_{3,3} - b_{3,4} + b_{4,3} - b_{4,4}\right)$$
$$h_{56} = \left(a_{1,1} - a_{1,5} - a_{4,1} + a_{4,5}\right)\left(b_{3,1} + b_{3,3} - b_{3,4} + b_{5,1} + b_{5,3} - b_{5,4}\right)$$
$$h_{57} = \left(-a_{3,1} - a_{4,1}\right)\left(-b_{1,3} - b_{1,5} - b_{2,5} - b_{5,1} - b_{5,3} - b_{5,5}\right)$$
$$h_{58} = \left(-a_{1,4} - a_{1,5} - a_{3,4} - a_{3,5}\right)\left(-b_{5,1} + b_{5,4} - b_{5,5}\right)$$
$$h_{59} = \left(-a_{3,3} + a_{3,4} - a_{4,3} + a_{4,4}\right)\left(b_{4,1} + b_{4,3} + b_{4,5} + b_{5,1} + b_{5,3} + b_{5,5}\right)$$
$$h_{60} = \left(a_{2,5} + a_{4,5}\right)\left(b_{2,3} - b_{3,1} - b_{3,2} - b_{3,3} - b_{5,2} - b_{5,3}\right)$$
$$h_{61} = \left(a_{1,4} + a_{3,4}\right)\left(b_{1,1} - b_{1,4} + b_{1,5} - b_{2,5} - b_{4,4} + b_{4,5} - b_{5,1} + b_{5,4} - b_{5,5}\right)$$
$$h_{62} = \left(a_{2,1} + a_{4,1}\right)\left(b_{1,2} + b_{1,3} + b_{2,2} - b_{4,1} - b_{4,2} - b_{4,3}\right)$$
$$h_{63} = \left(-a_{3,3} - a_{4,3}\right)\left(-b_{2,3} - b_{3,3} - b_{3,5} - b_{4,1} - b_{4,3} - b_{4,5}\right)$$
$$h_{64} = \left(a_{1,1} - a_{1,3} - a_{1,4} + a_{3,1} - a_{3,3} - a_{3,4}\right)\left(b_{1,1} - b_{1,4} + b_{1,5}\right)$$
$$h_{65} = \left(-a_{1,1} + a_{4,1}\right)\left(-b_{1,3} + b_{1,4} + b_{2,4} - b_{5,1} - b_{5,3} + b_{5,4}\right)$$
$$h_{66} = \left(a_{1,1} - a_{1,2} + a_{1,3} - a_{1,5} - a_{2,2} - a_{2,5} - a_{3,2} + a_{3,3} - a_{4,1} + a_{4,2}\right)b_{2,4}$$
$$h_{67} = \left(a_{2,5} - a_{3,5}\right)\left(b_{1,1} + b_{1,2} + b_{1,5} - b_{2,5} - b_{4,1} - b_{4,2} - b_{4,5} + b_{5,2} + b_{5,5}\right)$$
$$h_{68} = \left(a_{1,1} + a_{1,3} - a_{1,4} - a_{1,5} - a_{4,1} - a_{4,3} + a_{4,4} + a_{4,5}\right)\left(-b_{3,1} - b_{3,3} + b_{3,4}\right)$$
$$h_{69} = \left(-a_{1,3} + a_{1,4} - a_{2,3} + a_{2,4}\right)\left(-b_{2,4} - b_{3,1} - b_{3,2} + b_{3,4} - b_{5,2} + b_{5,4}\right)$$
$$h_{70} = \left(a_{2,3} - a_{2,5} + a_{4,3} - a_{4,5}\right)\left(-b_{3,1} - b_{3,2} - b_{3,3}\right)$$
$$h_{71} = \left(-a_{3,1} + a_{3,3} - a_{3,4} + a_{3,5} - a_{4,1} + a_{4,3} - a_{4,4} + a_{4,5}\right)\left(-b_{5,1} - b_{5,3} - b_{5,5}\right)$$
$$h_{72} = \left(-a_{2,1} - a_{2,4} - a_{4,1} - a_{4,4}\right)\left(b_{4,1} + b_{4,2} + b_{4,3}\right)$$
$$h_{73} = \left(a_{1,3} - a_{1,4} - a_{1,5} + a_{2,3} - a_{2,4} - a_{2,5}\right)\left(b_{1,1} + b_{1,2} - b_{1,4} + b_{2,4} + b_{5,2} - b_{5,4}\right)$$
$$h_{74} = \left(a_{2,1} - a_{2,3} + a_{2,4} - a_{3,1} + a_{3,3} - a_{3,4}\right)\left(b_{4,1} + b_{4,2} + b_{4,5}\right)$$
$$h_{75} = -\left(a_{1,2} + a_{1,4} - a_{2,2} - a_{2,5} - a_{3,1} + a_{3,2} + a_{3,4} + a_{3,5} - a_{4,1} + a_{4,2}\right)b_{2,5}$$
$$h_{76} = \left(a_{1,3} + a_{3,3}\right)\left(-b_{1,1} + b_{1,4} - b_{1,5} + b_{2,4} + b_{3,4} - b_{3,5}\right)$$
$$c_{1,1} = -h_{10} + h_{12} + h_{14} - h_{15} - h_{16} + h_{53} + h_{5} - h_{66} - h_{7}$$
$$c_{2,1} = h_{10} + h_{11} - h_{12} + h_{13} + h_{15} + h_{16} - h_{17} - h_{44} + h_{51}$$
$$c_{3,1} = h_{10} - h_{12} + h_{15} + h_{16} - h_{1} + h_{2} + h_{3} - h_{4} + h_{75}$$
$$c_{4,1} = -h_{10} + h_{12} - h_{15} - h_{16} + h_{52} + h_{54} - h_{6} - h_{8} + h_{9}$$
$$c_{1,2} = h_{13} + h_{15} + h_{20} + h_{21} - h_{22} + h_{23} + h_{25} - h_{43} + h_{49} + h_{50}$$
$$c_{2,2} = -h_{11} + h_{12} - h_{13} - h_{15} - h_{16} + h_{17} + h_{18} - h_{19} - h_{21} + h_{43} + h_{44}$$
$$c_{3,2} = -h_{16} - h_{19} - h_{21} - h_{28} - h_{29} - h_{38} + h_{42} + h_{44} - h_{47} + h_{48}$$
$$c_{4,2} = h_{11} - h_{12} - h_{18} + h_{21} - h_{32} + h_{33} - h_{34} - h_{36} + h_{62} - h_{70}$$
$$c_{1,3} = h_{15} + h_{23} + h_{24} + h_{34} - h_{37} + h_{40} - h_{41} + h_{55} - h_{56} - h_{9}$$
$$c_{2,3} = -h_{10} + h_{19} + h_{32} + h_{35} + h_{36} + h_{37} - h_{43} - h_{60} - h_{6} - h_{72}$$
$$c_{3,3} = -h_{16} - h_{28} + h_{33} + h_{37} - h_{39} + h_{45} - h_{46} + h_{63} - h_{71} - h_{8}$$
$$c_{4,3} = h_{10} + h_{15} + h_{16} - h_{33} + h_{34} - h_{35} - h_{37} - h_{54} + h_{6} + h_{8} - h_{9}$$
$$c_{1,4} = -h_{10} + h_{12} + h_{14} - h_{16} + h_{23} + h_{24} + h_{25} + h_{26} + h_{5} - h_{66} - h_{7}$$
$$c_{2,4} = h_{10} + h_{18} - h_{19} + h_{20} - h_{22} - h_{24} - h_{26} - h_{5} - h_{69} + h_{73}$$
$$c_{3,4} = -h_{14} + h_{16} - h_{23} - h_{26} + h_{27} + h_{29} + h_{31} + h_{46} - h_{58} + h_{76}$$
$$c_{4,4} = h_{12} + h_{25} + h_{26} - h_{33} - h_{35} - h_{40} + h_{41} + h_{65} - h_{68} - h_{7}$$
$$c_{1,5} = h_{15} + h_{24} + h_{25} + h_{27} - h_{28} + h_{30} + h_{31} - h_{4} + h_{61} + h_{64}$$
$$c_{2,5} = -h_{10} - h_{18} - h_{2} - h_{30} - h_{38} + h_{42} - h_{43} + h_{46} + h_{67} + h_{74}$$



$$h_{45} = (-a_{3,3} + a_{3,4} - a_{4,3})(b_{3,5} + b_{4,1} + b_{4,3} + b_{4,5} + b_{5,1} + b_{5,3} + b_{5,5})$$
$$h_{46} = -a_{3,5}(-b_{5,1} - b_{5,5})$$
$$h_{47} = (a_{2,1} - a_{2,5} - a_{3,1} + a_{3,5})(b_{1,1} + b_{1,2} + b_{1,5} - b_{4,1} - b_{4,2} - b_{4,5})$$
$$h_{48} = (-a_{2,3} + a_{3,3})(b_{2,2} + b_{3,2} + b_{3,5} + b_{4,1} + b_{4,2} + b_{4,5})$$
$$h_{49} = (-a_{1,1} - a_{1,3} + a_{1,4} + a_{1,5} - a_{2,1} - a_{2,3} + a_{2,4} + a_{2,5})(-b_{1,1} - b_{1,2} + b_{1,4})$$
$$h_{50} = (-a_{1,4} - a_{2,4})(b_{2,2} - b_{3,1} - b_{3,2} + b_{3,4} - b_{4,2} + b_{4,4})$$



$$c_{3,5} = -h_{10} + h_{12} - h_{15} + h_{28} + h_{29} - h_2 - h_{30} - h_3 + h_{46} + h_4 - h_{75}$$
$$c_{4,5} = -h_{12} - h_{29} + h_{30} - h_{34} + h_{35} + h_{39} + h_3 - h_{45} + h_{57} + h_{59}$$

4.2 Pseudokod

```
ai(A, B)
    Sprawdź, czy wymiary macierzy są odpowiednie do mnożenia.
    Jeśli rozmiary macierzy są mniejsze niż wymagane ( mniejsze niż 4x5 i 5x5), wykonaj
standardowe mnożenie macierzy.
    Jeśli rozmiary są większe, przeskaluj macierze do potęg liczb (4 i 5) poprzez dodanie
dodatkowych wierszy i kolumn z zerami.

    Podziel macierze A i B na mniejsze bloki, tak aby można było  wykonywać na nich
operacje.
    Oblicz zestaw pomocniczych wartości h na podstawie elementów macierzy A i B.
    Wykorzystaj pomocnicze wartości h do obliczenia wynikowych bloków macierzy C.
    Po obliczeniu bloków wynikowej macierzy, połącz je w jedną macierz wynikową C.
    Zwróć wynikową macierz C po zakończeniu wszystkich operacji.
```

4.3 Fragmenty kodu

```
def ai_compute_c(self, h, multiplication):
    c = Matrix.empty(4, 5, self.calculator, multiplication=multiplication)

    c[1, 1] = -h[10] + h[12] + h[14] - h[15] - h[16] + h[53] + h[5] - h[66] - h[7]
    c[2, 1] = h[10] + h[11] - h[12] + h[13] + h[15] + h[16] - h[17] - h[44] + h[51]
    c[3, 1] = h[10] - h[12] + h[15] + h[16] - h[1] + h[2] + h[3] - h[4] + h[75]
    c[4, 1] = -h[10] + h[12] - h[15] - h[16] + h[52] + h[54] - h[6] - h[8] + h[9]
    c[1, 2] = h[13] + h[15] + h[20] + h[21] - h[22] + h[23] + h[25] - h[43] + h[49] +
h[50]
    c[2, 2] = -h[11] + h[12] - h[13] - h[15] - h[16] + h[17] + h[18] - h[19] - h[21] +
h[43] + h[44]
    c[3, 2] = -h[16] - h[19] - h[21] - h[28] - h[29] - h[38] + h[42] + h[44] - h[47] +
h[48]
    c[4, 2] = h[11] - h[12] - h[18] + h[21] - h[32] + h[33] - h[34] - h[36] + h[62] -
h[70]
    c[1, 3] = h[15] + h[23] + h[24] + h[34] - h[37] + h[40] - h[41] + h[55] - h[56] -
h[9]
    c[2, 3] = -h[10] + h[19] + h[32] + h[35] + h[36] + h[37] - h[43] - h[60] - h[6] -
h[72]
    c[3, 3] = -h[16] - h[28] + h[33] + h[37] - h[39] + h[45] - h[46] + h[63] - h[71] -
h[8]
    c[4, 3] = h[10] + h[15] + h[16] - h[33] + h[34] - h[35] - h[37] - h[54] + h[6] + h[8]
- h[9]
    c[1, 4] = -h[10] + h[12] + h[14] - h[16] + h[23] + h[24] + h[25] + h[26] + h[5] -
h[66] - h[7]
    c[2, 4] = h[10] + h[18] - h[19] + h[20] - h[22] - h[24] - h[26] - h[5] - h[69] +
h[73]
    c[3, 4] = -h[14] + h[16] - h[23] - h[26] + h[27] + h[29] + h[31] + h[46] - h[58] +
h[76]
    c[4, 4] = h[12] + h[25] + h[26] - h[33] - h[35] - h[40] + h[41] + h[65] - h[68] -
h[7]
    c[1, 5] = h[15] + h[24] + h[25] + h[27] - h[28] + h[30] + h[31] - h[4] + h[61] +
```

```
h[64]
    c[2, 5] = -h[10] - h[18] - h[2] - h[30] - h[38] + h[42] - h[43] + h[46] + h[67] +
h[74]
    c[3, 5] = -h[10] + h[12] - h[15] + h[28] + h[29] - h[2] - h[30] - h[3] + h[46] + h[4]
- h[75]
    c[4, 5] = -h[12] - h[29] + h[30] - h[34] + h[35] + h[39] + h[3] - h[45] + h[57] +
h[59]

    return c
```

```
@staticmethod
def ai_compute_h(a, b):
    h = [None] * 77
    h[1] = (a[3, 2]) * (-b[2, 1] - b[2, 5] - b[3, 1])
    h[2] = (a[2, 2] + a[2, 5] - a[3, 5]) * (-b[2, 5] - b[5, 1])
    h[3] = (-a[3, 1] - a[4, 1] + a[4, 2]) * (-b[1, 1] + b[2, 5])
    h[4] = (a[1, 2] + a[1, 4] + a[3, 4]) * (-b[2, 5] - b[4, 1])
    h[5] = (a[1, 5] + a[2, 2] + a[2, 5]) * (-b[2, 4] + b[5, 1])
    h[6] = (-a[2, 2] - a[2, 5] - a[4, 5]) * (b[2, 3] + b[5, 1])
    h[7] = (-a[1, 1] + a[4, 1] - a[4, 2]) * (b[1, 1] + b[2, 4])
    h[8] = (a[3, 2] - a[3, 3] - a[4, 3]) * (-b[2, 3] + b[3, 1])
    h[9] = (-a[1, 2] - a[1, 4] + a[4, 4]) * (b[2, 3] + b[4, 1])
    h[10] = (a[2, 2] + a[2, 5]) * (b[5, 1])
    h[11] = (-a[2, 1] - a[4, 1] + a[4, 2]) * (-b[1, 1] + b[2, 2])
    h[12] = (a[4, 1] - a[4, 2]) * (b[1, 1])
    h[13] = (a[1, 2] + a[1, 4] + a[2, 4]) * (b[2, 2] + b[4, 1])
    h[14] = (a[1, 3] - a[3, 2] + a[3, 3]) * (b[2, 4] + b[3, 1])
    h[15] = (-a[1, 2] - a[1, 4]) * (b[4, 1])
    h[16] = (-a[3, 2] + a[3, 3]) * (b[3, 1])
    h[17] = (a[1, 2] + a[1, 4] - a[2, 1] + a[2, 2] - a[2, 3] + a[2, 4] - a[3, 2] + a[3,
3] - a[4, 1] + a[4, 2]) * (
        b[2, 2])
    h[18] = (a[2, 1]) * (b[1, 1] + b[1, 2] + b[5, 2])
    h[19] = (-a[2, 3]) * (b[3, 1] + b[3, 2] + b[5, 2])
    h[20] = (-a[1, 5] + a[2, 1] + a[2, 3] - a[2, 5]) * (-b[1, 1] - b[1, 2] + b[1, 4] -
b[5, 2])
    h[21] = (a[2, 1] + a[2, 3] - a[2, 5]) * (b[5, 2])
    h[22] = (a[1, 3] - a[1, 4] - a[2, 4]) * (b[1, 1] + b[1, 2] - b[1, 4] - b[3, 1] - b[3,
2] + b[3, 4] + b[4, 4])
    h[23] = (a[1, 3]) * (-b[3, 1] + b[3, 4] + b[4, 4])
    h[24] = (a[1, 5]) * (-b[4, 4] - b[5, 1] + b[5, 4])
    h[25] = (-a[1, 1]) * (b[1, 1] - b[1, 4])
    h[26] = (-a[1, 3] + a[1, 4] + a[1, 5]) * (b[4, 4])
    h[27] = (a[1, 3] - a[3, 1] + a[3, 3]) * (b[1, 1] - b[1, 4] + b[1, 5] + b[3, 5])
    h[28] = (-a[3, 4]) * (-b[3, 5] - b[4, 1] - b[4, 5])
    h[29] = (a[3, 1]) * (b[1, 1] + b[1, 5] + b[3, 5])
    h[30] = (a[3, 1] - a[3, 3] + a[3, 4]) * (b[3, 5])
    h[31] = (-a[1, 4] - a[1, 5] - a[3, 4]) * (-b[4, 4] - b[5, 1] + b[5, 4] - b[5, 5])
    h[32] = (a[2, 1] + a[4, 1] + a[4, 4]) * (b[1, 3] - b[4, 1] - b[4, 2] - b[4, 3])
    h[33] = (a[4, 3]) * (-b[3, 1] - b[3, 3])
    h[34] = (a[4, 4]) * (-b[1, 3] + b[4, 1] + b[4, 3])
    h[35] = (-a[4, 5]) * (b[1, 3] + b[5, 1] + b[5, 3])
    h[36] = (a[2, 3] - a[2, 5] - a[4, 5]) * (b[3, 1] + b[3, 2] + b[3, 3] + b[5, 2])
    h[37] = (-a[4, 1] - a[4, 4] + a[4, 5]) * (b[1, 3])
    h[38] = (-a[2, 3] - a[3, 1] + a[3, 3] - a[3, 4]) * (b[3, 5] + b[4, 1] + b[4, 2] +
b[4, 5])
    h[39] = (-a[3, 1] - a[4, 1] - a[4, 4] + a[4, 5]) * (b[1, 3] + b[5, 1] + b[5, 3] +
b[5, 5])
    h[40] = (-a[1, 3] + a[1, 4] + a[1, 5] - a[4, 4]) * (-b[3, 1] - b[3, 3] + b[3, 4] +
b[4, 4])
    h[41] = (-a[1, 1] + a[4, 1] - a[4, 5]) * (b[1, 3] + b[3, 1] + b[3, 3] - b[3, 4] +
```

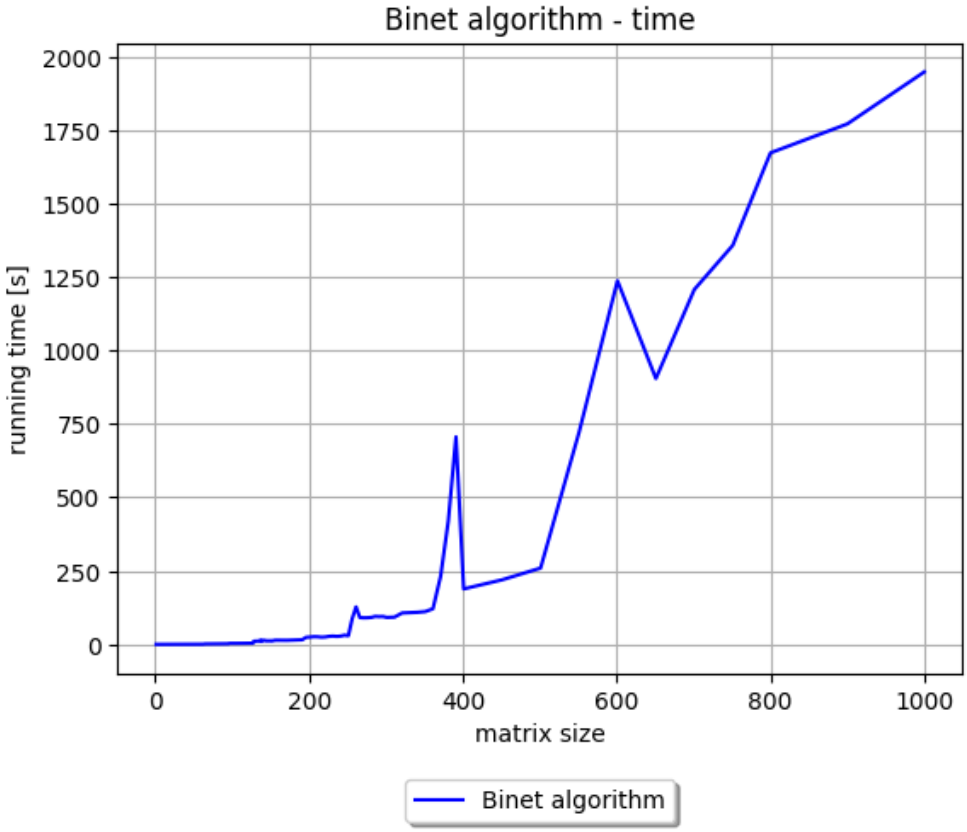
```
b[5, 1] + b[5, 3] - b[5, 4])
    h[42] = (-a[2, 1] + a[2, 5] - a[3, 5]) * (-b[1, 1] - b[1, 2] - b[1, 5] + b[4, 1] +
b[4, 2] + b[4, 5] - b[5, 2])
    h[43] = (a[2, 4]) * (b[4, 1] + b[4, 2])
    h[44] = (a[2, 3] + a[3, 2] - a[3, 3]) * (b[2, 2] - b[3, 1])
    h[45] = (-a[3, 3] + a[3, 4] - a[4, 3]) * (b[3, 5] + b[4, 1] + b[4, 3] + b[4, 5] +
b[5, 1] + b[5, 3] + b[5, 5])
    h[46] = (-a[3, 5]) * (-b[5, 1] - b[5, 5])
    h[47] = (a[2, 1] - a[2, 5] - a[3, 1] + a[3, 5]) * (b[1, 1] + b[1, 2] + b[1, 5] - b[4,
1] - b[4, 2] - b[4, 5])
    h[48] = (-a[2, 3] + a[3, 3]) * (b[2, 2] + b[3, 2] + b[3, 5] + b[4, 1] + b[4, 2] +
b[4, 5])
    h[49] = (-a[1, 1] - a[1, 3] + a[1, 4] + a[1, 5] - a[2, 1] - a[2, 3] + a[2, 4] + a[2,
5]) * (
            -b[1, 1] - b[1, 2] + b[1, 4])
    h[50] = (-a[1, 4] - a[2, 4]) * (b[2, 2] - b[3, 1] - b[3, 2] + b[3, 4] - b[4, 2] +
b[4, 4])
    h[51] = (a[2, 2]) * (b[2, 1] + b[2, 2] - b[5, 1])
    h[52] = (a[4, 2]) * (b[1, 1] + b[2, 1] + b[2, 3])
    h[53] = (-a[1, 2]) * (-b[2, 1] + b[2, 4] + b[4, 1])
    h[54] = (a[1, 2] + a[1, 4] - a[2, 2] - a[2, 5] - a[3, 2] + a[3, 3] - a[4, 2] + a[4,
3] - a[4, 4] - a[4, 5]) * (
        b[2, 3])
    h[55] = (a[1, 4] - a[4, 4]) * (-b[2, 3] + b[3, 1] + b[3, 3] - b[3, 4] + b[4, 3] -
b[4, 4])
    h[56] = (a[1, 1] - a[1, 5] - a[4, 1] + a[4, 5]) * (b[3, 1] + b[3, 3] - b[3, 4] + b[5,
1] + b[5, 3] - b[5, 4])
    h[57] = (-a[3, 1] - a[4, 1]) * (-b[1, 3] - b[1, 5] - b[2, 5] - b[5, 1] - b[5, 3] -
b[5, 5])
    h[58] = (-a[1, 4] - a[1, 5] - a[3, 4] - a[3, 5]) * (-b[5, 1] + b[5, 4] - b[5, 5])
    h[59] = (-a[3, 3] + a[3, 4] - a[4, 3] + a[4, 4]) * (b[4, 1] + b[4, 3] + b[4, 5] +
b[5, 1] + b[5, 3] + b[5, 5])
    h[60] = (a[2, 5] + a[4, 5]) * (b[2, 3] - b[3, 1] - b[3, 2] - b[3, 3] - b[5, 2] - b[5,
3])
    h[61] = (a[1, 4] + a[3, 4]) * (
            b[1, 1] - b[1, 4] + b[1, 5] - b[2, 5] - b[4, 4] + b[4, 5] - b[5, 1] + b[5, 4]
- b[5, 5])
    h[62] = (a[2, 1] + a[4, 1]) * (b[1, 2] + b[1, 3] + b[2, 2] - b[4, 1] - b[4, 2] - b[4,
3])
    h[63] = (-a[3, 3] - a[4, 3]) * (-b[2, 3] - b[3, 3] - b[3, 5] - b[4, 1] - b[4, 3] -
b[4, 5])
    h[64] = (a[1, 1] - a[1, 3] - a[1, 4] + a[3, 1] - a[3, 3] - a[3, 4]) * (b[1, 1] - b[1,
4] + b[1, 5])
    h[65] = (-a[1, 1] + a[4, 1]) * (-b[1, 3] + b[1, 4] + b[2, 4] - b[5, 1] - b[5, 3] +
b[5, 4])
    h[66] = (a[1, 1] - a[1, 2] + a[1, 3] - a[1, 5] - a[2, 2] - a[2, 5] - a[3, 2] + a[3,
3] - a[4, 1] + a[4, 2]) * (
        b[2, 4])
    h[67] = (a[2, 5] - a[3, 5]) * (
            b[1, 1] + b[1, 2] + b[1, 5] - b[2, 5] - b[4, 1] - b[4, 2] - b[4, 5] + b[5, 2]
+ b[5, 5])
    h[68] = (a[1, 1] + a[1, 3] - a[1, 4] - a[1, 5] - a[4, 1] - a[4, 3] + a[4, 4] + a[4,
5]) * (
            -b[3, 1] - b[3, 3] + b[3, 4])
    h[69] = (-a[1, 3] + a[1, 4] - a[2, 3] + a[2, 4]) * (-b[2, 4] - b[3, 1] - b[3, 2] +
b[3, 4] - b[5, 2] + b[5, 4])
    h[70] = (a[2, 3] - a[2, 5] + a[4, 3] - a[4, 5]) * (-b[3, 1] - b[3, 2] - b[3, 3])
    h[71] = (-a[3, 1] + a[3, 3] - a[3, 4] + a[3, 5] - a[4, 1] + a[4, 3] - a[4, 4] + a[4,
5]) * (
            -b[5, 1] - b[5, 3] - b[5, 5])
    h[72] = (-a[2, 1] - a[2, 4] - a[4, 1] - a[4, 4]) * (b[4, 1] + b[4, 2] + b[4, 3])
    h[73] = (a[1, 3] - a[1, 4] - a[1, 5] + a[2, 3] - a[2, 4] - a[2, 5]) * (
            b[1, 1] + b[1, 2] - b[1, 4] + b[2, 4] + b[5, 2] - b[5, 4])
```

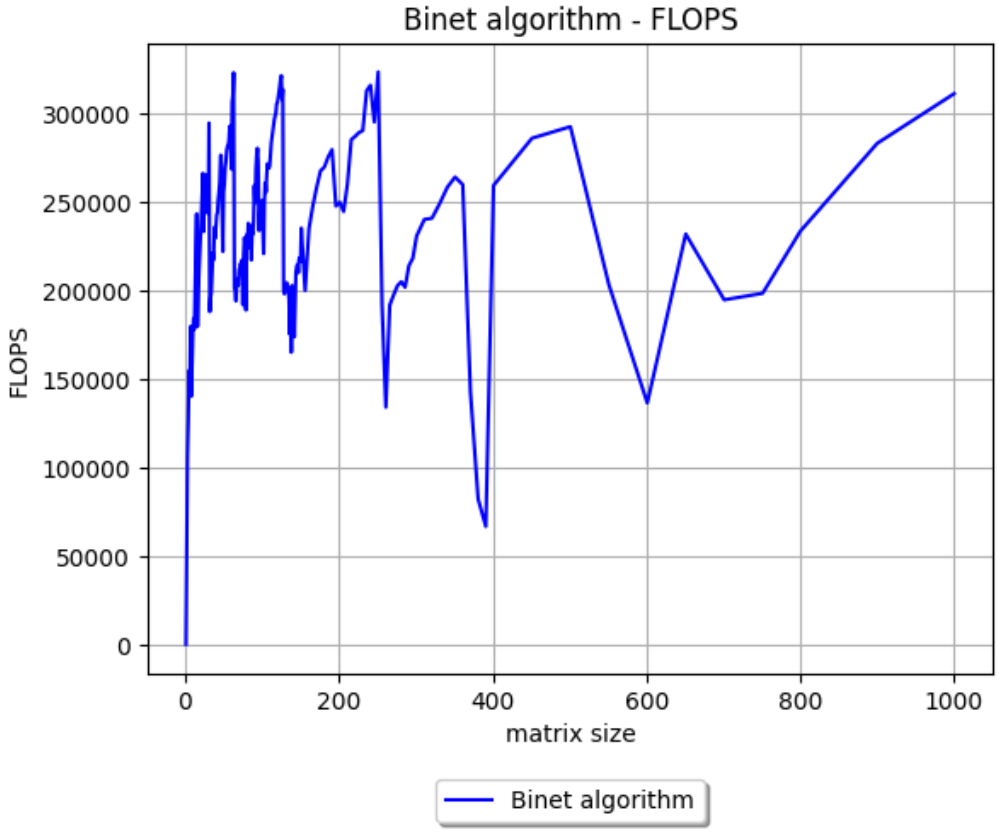
```
    h[74] = (a[2, 1] - a[2, 3] + a[2, 4] - a[3, 1] + a[3, 3] - a[3, 4]) * (b[4, 1] + b[4,
2] + b[4, 5])
    h[75] = -(a[1, 2] + a[1, 4] - a[2, 2] - a[2, 5] - a[3, 1] + a[3, 2] + a[3, 4] + a[3,
5] - a[4, 1] + a[4, 2]) * (
        b[2, 5])
    h[76] = (a[1, 3] + a[3, 3]) * (-b[1, 1] + b[1, 4] - b[1, 5] + b[2, 4] + b[3, 4] -
b[3, 5])

    return h
```

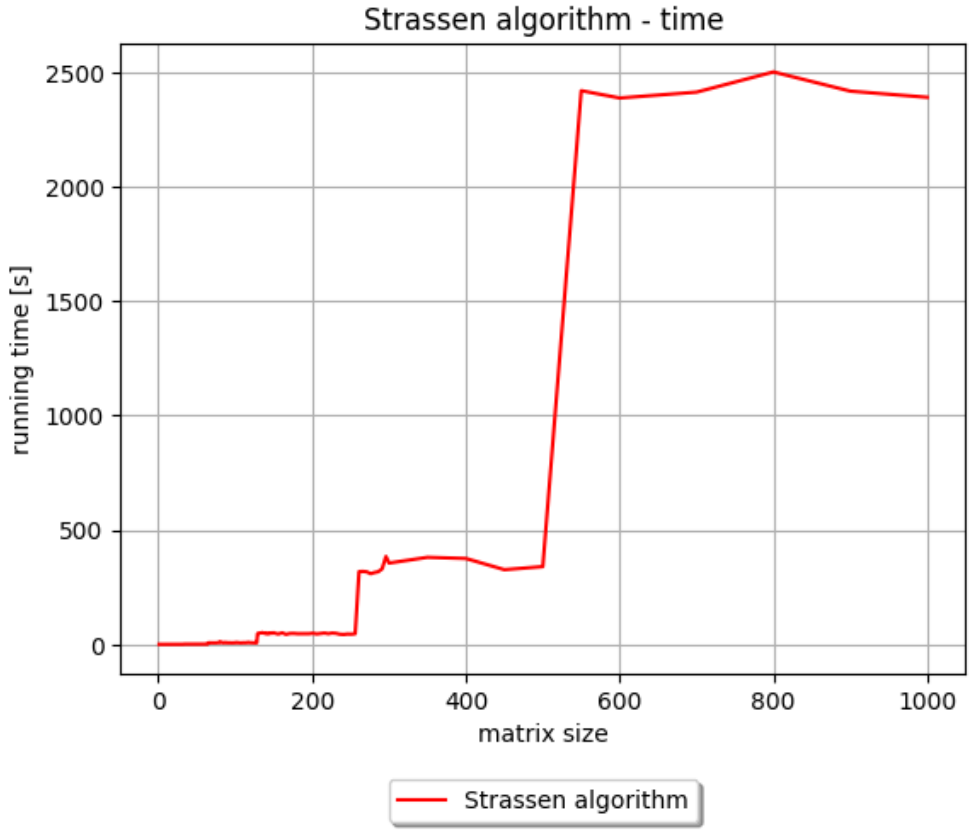
```
def recursive_ai(self, a, b):
    # Scales up matrices to powers of 4 and 5 by appending zeroes
    if a.shape[0] < 4 or a.shape[1] < 5 or b.shape[0] < 5 or b.shape[1] < 5:
        return Matrix(self.calculator.standard_matrix_multiplication(a.data, b.data),
self.calculator, multiplication=a.multiplication)
    n, k = a.shape
    k, m = b.shape
    a = self.rescale(a, shape=(self.power(n, 4), self.power(k, 5)))
    b = self.rescale(b, shape=(self.power(k, 5), self.power(m, 5)))
    a.multiplication = self.recursive_ai_rec
    b.multiplication = self.recursive_ai_rec
    retval = self.recursive_ai_rec(a, b)

    return self.submatrix(retval, yrange=(1, n + 1), xrange=(1, m + 1))
```

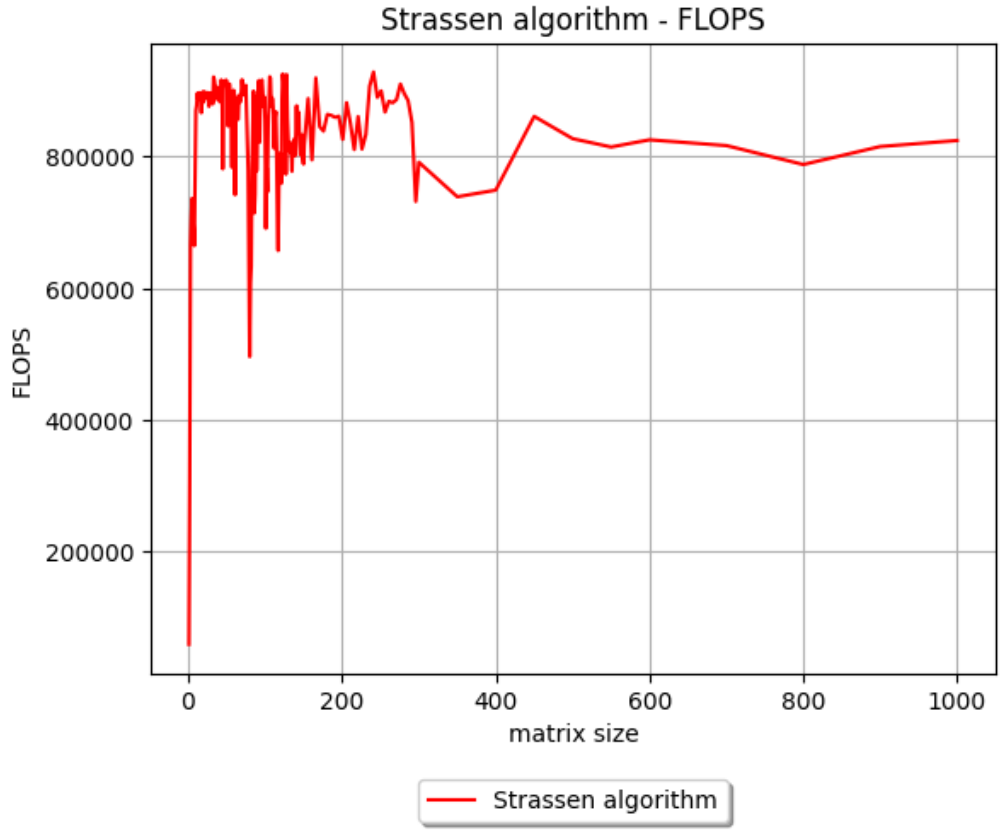
```
def recursive_ai_rec(self, a, b):
    # Works only for powers of 4 and 5
    if a.shape[0] < 4 or a.shape[1] < 5 or b.shape[0] < 5 or b.shape[1] < 5:
        return Matrix(self.calculator.standard_matrix_multiplication(a.data, b.data),
self.calculator, multiplication=a.multiplication)

    A = self.block(a, shape=(4, 5))
    B = self.block(b, shape=(5, 5))
    return self.expand(self.ai_multiplication(A, B))
```

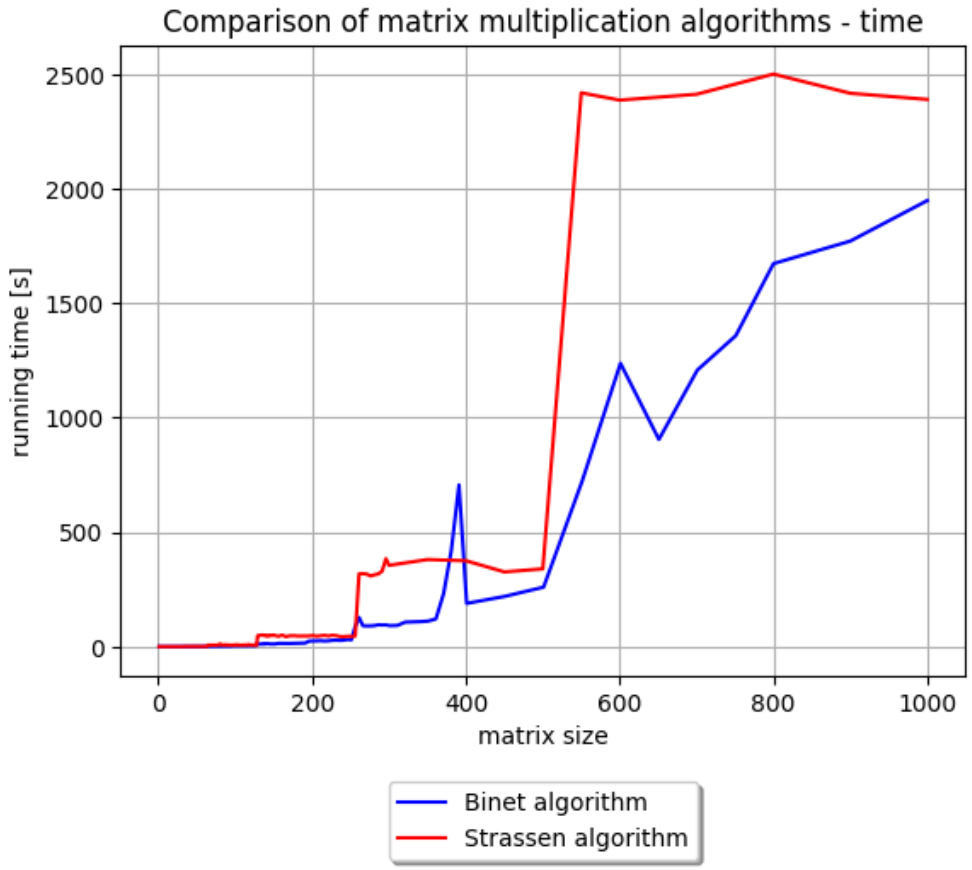
5. Wykresy

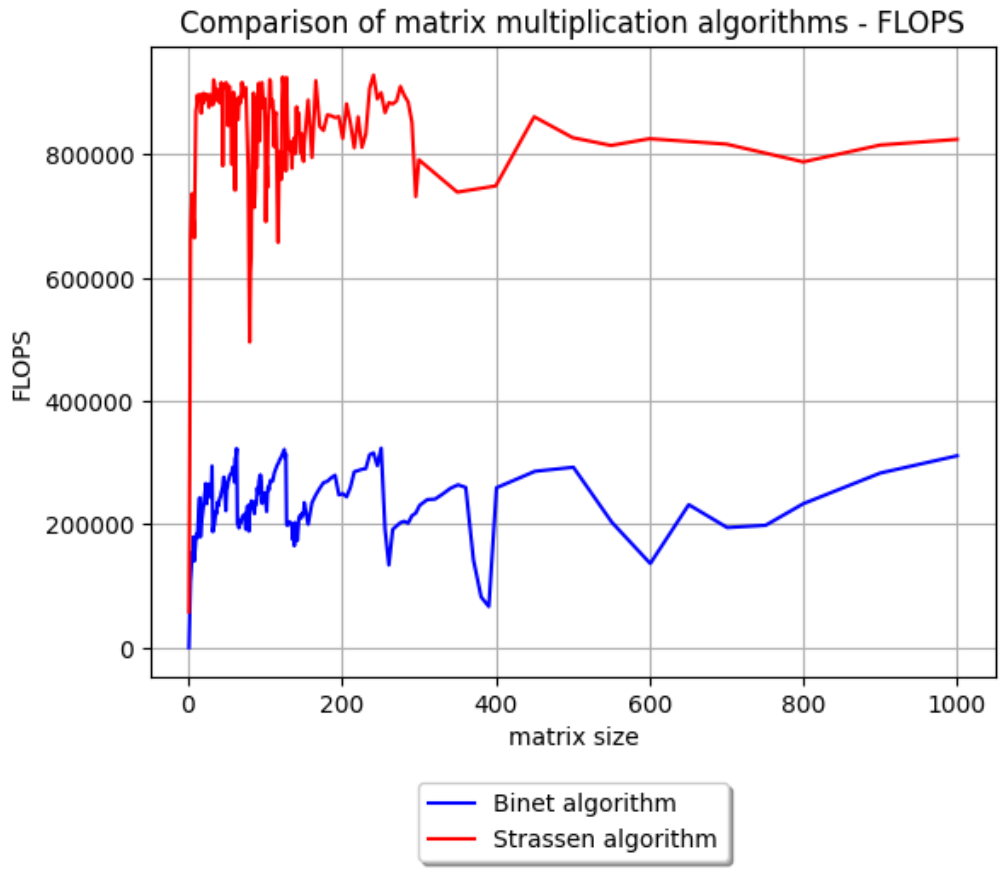

Binet algorithm - time
running time [s]
matrix size
Binet algorithm


Binet algorithm - FLOPS
FLOPS
matrix size
Binet algorithm


Strassen algorithm - time
running time [s]
matrix size
Strassen algorithm


Strassen algorithm - FLOPS
FLOPS
matrix size
Strassen algorithm





6. Szacowanie złożoności obliczeniowej

Szacowana złożoność obliczeniowa rekurencyjnego algorytmu mnożenia macierzy Bineta wynosi $O(n^3)$, ponieważ każde mnożenie bloków wymaga 8 rekurencyjnych mnożeń mniejszych macierzy. W wyniku rekurencji, rozmiar macierzy jest zmniejszany o połowę (do $n/2$) na każdym poziomie. Liczba poziomów rekurencji wynosi $log_2 n$ Dlatego całkowita liczba operacji

mnożenia to $8^{log_2 n)}$, co upraszcza się do $O(n^3)$. W przypadku algorytmu Strassena liczba mnożeń na każdym poziomie rekurencji została zredukowana do 7 zamiast 8, co prowadzi do złożoności $O(n^{log_2 7})$, czyli około $O(n^{2.81})$. Złożoność algorytmu AI będzie mieściła się w przedziale między $O(n^3)$ a $O(n^{2.81})$, bliżej dolnej granicy dzięki wykorzystaniu bloków o określonych rozmiarach i optymalizacji operacji na blokach.

7. Porównanie wyników z Matlabem

Aby zweryfikować poprawność zaimplementowanych algorytmów, porównaliśmy wyniki mnożenia dwóch macierzy w środowisku Matlab z wynikami uzyskanymi przy użyciu implementacji algorytmu Binet'a i Strassen'a.

Porównanie pozwoliło upewnić się, że zaimplementowane algorytmy dają poprawne wyniki, zgodne z otrzymanymi w Matlabie. Dodatkowo, po sprawdzeniu czasu wykonania, okazało się, że mnożenie macierzy w Matlabie działa szybciej w porównaniu z naszymi implementacjami.

```
A = [0.1, 0.7, 0.2, 0.9;
    0.1, 0.5, 0.2, 0.7;
    0.9, 0.3, 0.5, 0.6;
    0.4, 0.6, 0.2, 0.9];


B = [0.2, 0.1, 0.8, 0.3;
    0.3, 0.9, 0.4, 0.7;
    0.5, 0.3, 0.1, 0.7;
    0.1, 0.6, 0.4, 0.8];

tic;
C = A * B;
czasWykonania = toc;

disp('Wynik mnożenia macierzy A i B:');
disp(C);

disp(['Czas wykonywania mnożenia: ', num2str(czasWykonania), ' sekund']);
```

8. Wnioski

- Nasza implementacja algorytmu Strassena okazała się być gorsza pod względem wydajnościowym niż nasza implementacja algorytmu Bineta
- Klasyczny algorytm Bineta jest najprostszy, ale przy dużych macierzach jest niepraktyczny
- Algorytm stworzony przez sztuczną inteligencję posiada optymalizacje, takie jak dzielenie macierzy na bloki i skalowanie ich, aby dopasować ich rozmiar do efektywnych operacji blokowych
- Przy większych macierzach rekurencyjne podejście może znacząco spowolnić obliczenia, w praktyce, kroki rekurencyjne wykorzystuje się tylko do pewnego opłacalnego pułapu, a dalej korzysta się z prostszych algorytmów mnożenia macierzy.

9. Bibliografia

- Wykłady prof. dr hab. Macieja Paszyńskiego (https://home.agh.edu.pl/~paszynsk/RM/RachunekMacierzowy1.pdf)
- https://www.researchgate.net/publication/2779622_Implementation_of_Strassen's_Algorithm_for_Matrix_Multiplication
- https://deepmind.google/discover/blog/discovering-novel-algorithms-with-alphatensor/#:~:text=In%20our%20paper,%20published%20today%20in%20Nature,%20we