Algorytmy Macierzowe
Sprawozdanie 1
Rekurencyjne mnożenie macierzy

Szymon Paszkiewicz
Przemysław Węglik

6 listopada 2022

Spis treści

1 Algorytm Binét'a

1.1 Opis algorytmu

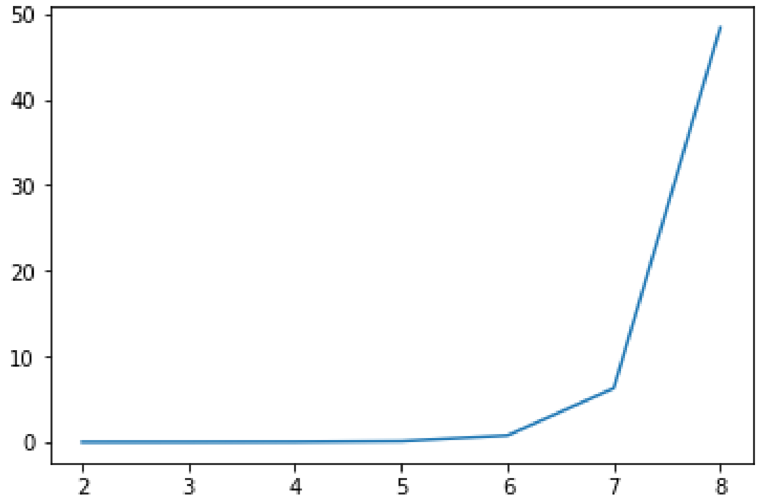
Polega na rekurencyjnym rozbijaniu macierzy na 4 mniejsze i obliczaniu wyników dla tych podproblemów. Tam ponownie będziemy musieli użyć mnożenia i wykorzystamy procedurę. To klasyczne podejście nosi nazwę "divide and conquer". Stosujemy wzór:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} (A_{11}B_{11} + A_{12}B_{21}) & (A_{11}B_{21} + A_{12}B_{22}) \\ (A_{21}B_{11} + A_{22}B_{21}) & (A_{21}B_{12} + A_{22}B_{22}) \end{bmatrix}$$

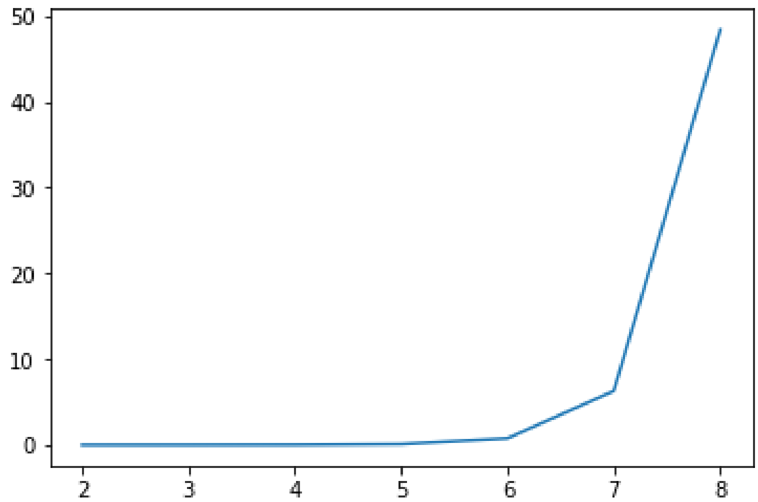
1.2 Kod algorytmu

```
def binet_core_algorithm(
                          A: np.ndarray,
                          B: np.ndarray,
                          counter: Counter
                      ) -> np.ndarray:
    add = counter.add
    sub = counter.sub
    mul = counter.mul
    if A.size > 1:
        split_at = A.shape[0]//2
        A11, A12, A21, A22 = split(A, split_at, split_at)
        B11, B12, B21, B22 = split(B, split_at, split_at)

        C11 = add(binet_core_algorithm(A11, B11, counter),
            binet_core_algorithm(A12, B21, counter))
        C12 = add(binet_core_algorithm(A11, B12, counter),
            binet_core_algorithm(A12, B22, counter))
        C21 = add(binet_core_algorithm(A21, B11, counter),
            binet_core_algorithm(A22, B21, counter))
        C22 = add(binet_core_algorithm(A21, B12, counter),
            binet_core_algorithm(A22, B22, counter))

        return np.concatenate(
            [
                np.concatenate([C11, C12], axis = 1),
                np.concatenate([C21, C22], axis = 1)
            ],
            axis = 0)
    else:
        return mul(A, B)
```

1.3 Benchmarki



Rysunek 1: Wykres czasu od rozmiaru macierzy



Rysunek 2: Wykres ilości operacji od rozmiaru macierzy

1.4 Analiza złożoności

1.5 Porównanie z matlabem

Matlab:

```
A = [1 3 5; 2 4 6; 3 5 7];
B = [1 2 3; 4 5 6; 7 8 9];

disp(A * B);

% output
>> matrix_mul
48      57      66
60      72      84
72      87     102
```

Python:

```python
A = np.array([[1, 3, 5], [2, 4, 6], [3, 5, 7]])
B = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print(binet_algorithm(A, B, Counter()))

# output
[[ 48  57  66]
 [ 60  72  84]
 [ 72  87 102]]
```