

IV zestaw zadań - Mnożenie macierzy hierarchicznych

Kacper Kozubowski, Mateusz Podmokły
III rok Informatyka WI

11 grudnia 2024

1 Treść zadania

Należy wygenerować macierz o rozmiarze $2^{3k} = 2^k \cdot 2^k \cdot 2^k$ dla $k \in \{2, 3, 4\}$ o strukturze opisującej topologię trójwymiarowej siatki zbudowanej z elementów sześciennych. Następnie wykonać polecenia:

1. użyć rekurencyjną procedurę kompresji macierzy z Zadania 3,
2. narysować macierz skompresowaną używając funkcji z Zadania 3,
3. przemnożyć macierz skompresowaną przez wektor,
4. przemnożyć macierz skompresowaną przez samą siebie.

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:

```

1  import numpy as np
2  import cv2
3  import matplotlib.pyplot as plt
4  import matplotlib.patches as patches

```

3.2 Kod funkcji

W zadaniu wykorzystane zostały funkcje `truncated_SVD`, `compress_matrix`, `create_tree` oraz klasa `Node` z Zadania 3.

```

1  def matrix_vector_mult(node, x, tmin, tmax, smin, smax):
2      if not node.children:
3          if node.rank == 0:
4              return np.zeros(tmax - tmin, dtype=float)
5
6          U = np.array(node.U)
7          V = np.array(node.V)
8          x_sub = x[smin:smax]
9
10         if node.size == (1, 1):
11             return node.singular_values * x_sub
12
13         tmp = V @ x_sub
14         y = U @ tmp
15         return y
16
17     tmid = (tmin + tmax) // 2
18     smid = (smin + smax) // 2
19     y_tl = matrix_vector_mult(node.children[0], x, tmin, tmid, smin, smid)
20     y_tr = matrix_vector_mult(node.children[1], x, tmin, tmid, smid, smax)
21     y_bl = matrix_vector_mult(node.children[2], x, tmid, tmax, smin, smid)
22     y_br = matrix_vector_mult(node.children[3], x, tmid, tmax, smid, smax)
23
24     top = y_tl + y_tr
25     bottom = y_bl + y_br
26     return np.concatenate([top, bottom])

```

Rysunek 1: Mnożenie macierzy przez wektor.

```

1  def matrix_matrix_mult(v: Node, w: Node, rmax=10, eps=1e-8) -> Node:
2      if v.rank == 0 and not v.children:
3          result = Node()
4          result.size = (v.size[0], w.size[1])
5          result.rank = 0
6          return result
7      if w.rank == 0 and not w.children:
8          result = Node()
9          result.size = (v.size[0], w.size[1])
10         result.rank = 0
11         return result
12
13     rowsA, colsA = v.size
14     rowsB, colsB = w.size
15     if colsA != rowsB:
16         raise ValueError("Niezgodne rozmiary macierzy do mnożenia!")
17
18     # 1) liść x liść
19     if not v.children and not w.children:
20         if v.rank == 0 or w.rank == 0:
21             node = Node()
22             node.size = (rowsA, colsB)
23             node.rank = 0
24             return node
25
26     Uv = np.array(v.U)
27     Vv = np.array(v.V)
28     Uw = np.array(w.U)
29     Vw = np.array(w.V)
30     block = Uv @ (Vv @ Uw) @ Vw
31
32     rank_new, U_r, V_r = truncated_SVD_block(block, rmax, eps)
33     node = Node()
34     node.size = (rowsA, colsB)
35     node.rank = rank_new
36     if rank_new > 0:
37         node.U = U_r
38         node.V = V_r
39     return node

```

Rysunek 2: Mnożenie dwóch macierzy część 1.

```

1  # 2) liść x węzeł
2  if not v.children and w.children:
3      A1 ,A2, A3, A4 = split_leaf_into_4_subleaves(v)
4      B1, B2, B3, B4 = w.children
5      C1 = matrix_matrix_add(matrix_matrix_mult(A1, B1, rmax, eps),
6                             matrix_matrix_mult(A2, B3, rmax, eps),
7                             rmax, eps)
8      C2 = matrix_matrix_add(matrix_matrix_mult(A1, B2, rmax, eps),
9                             matrix_matrix_mult(A2, B4, rmax, eps),
10                             rmax, eps)
11     C3 = matrix_matrix_add(matrix_matrix_mult(A3, B1, rmax, eps),
12                             matrix_matrix_mult(A4, B3, rmax, eps),
13                             rmax, eps)
14     C4 = matrix_matrix_add(matrix_matrix_mult(A3, B2, rmax, eps),
15                             matrix_matrix_mult(A4, B4, rmax, eps),
16                             rmax, eps)
17     node = Node()
18     node.size = (rowsA, colsB)
19     node.children = [C1, C2, C3, C4]
20     return node
21
22 # 3) węzeł x liść
23 if v.children and not w.children:
24     B1, B2, B3, B4 = split_leaf_into_4_subleaves(w)
25     A1, A2, A3, A4 = v.children
26     C1 = matrix_matrix_add(matrix_matrix_mult(A1, B1, rmax, eps),
27                             matrix_matrix_mult(A2, B3, rmax, eps),
28                             rmax, eps)
29     C2 = matrix_matrix_add(matrix_matrix_mult(A1, B2, rmax, eps),
30                             matrix_matrix_mult(A2, B4, rmax, eps),
31                             rmax, eps)
32     C3 = matrix_matrix_add(matrix_matrix_mult(A3, B1, rmax, eps),
33                             matrix_matrix_mult(A4, B3, rmax, eps),
34                             rmax, eps)
35     C4 = matrix_matrix_add(matrix_matrix_mult(A3, B2, rmax, eps),
36                             matrix_matrix_mult(A4, B4, rmax, eps),
37                             rmax, eps)
38     node = Node()
39     node.size = (rowsA, colsB)
40     node.children = [C1, C2, C3, C4]
41     return node

```

Rysunek 3: Mnożenie dwóch macierzy część 2.

```

1  # 4) węzeł x węzeł
2  A1, A2, A3, A4 = v.children
3  B1, B2, B3, B4 = w.children
4  C1 = matrix_matrix_add(matrix_matrix_mult(A1, B1, rmax, eps),
5                          matrix_matrix_mult(A2, B3, rmax, eps),
6                          rmax, eps)
7  C2 = matrix_matrix_add(matrix_matrix_mult(A1, B2, rmax, eps),
8                          matrix_matrix_mult(A2, B4, rmax, eps),
9                          rmax, eps)
10 C3 = matrix_matrix_add(matrix_matrix_mult(A3, B1, rmax, eps),
11                        matrix_matrix_mult(A4, B3, rmax, eps),
12                        rmax, eps)
13 C4 = matrix_matrix_add(matrix_matrix_mult(A3, B2, rmax, eps),
14                        matrix_matrix_mult(A4, B4, rmax, eps),
15                        rmax, eps)
16 node = Node()
17 node.size = (rowsA, colsB)
18 node.children = [C1, C2, C3, C4]
19 return node

```

Rysunek 4: Mnożenie dwóch macierzy część 3.

Funkcja pomocnicza do mnożenia macierzy:

```

1 def matrix_matrix_add(v: Node, w: Node, rmax=10, eps=1e-8) -> Node:
2     if v.rank==0 and not v.children: return copy_node(w)
3     if w.rank==0 and not w.children: return copy_node(v)
4     if v.size != w.size:
5         raise ValueError("Niezgodne rozmiary macierzy w add!")
6
7     # 1) liść + liść
8     if not v.children and not w.children:
9         if v.rank == 0 and w.rank == 0:
10             node = Node()
11             node.size = v.size
12             node.rank = 0
13             return node
14
15         if v.rank == 0 and w.rank > 0:
16             return copy_node(w)
17         if w.rank == 0 and v.rank > 0:
18             return copy_node(v)
19         return add_leaf_leaf(v, w, rmax, eps)
20
21     # 2) węzeł + węzeł
22     if v.children and w.children:
23         node = Node()
24         node.size = v.size
25         A1, A2, A3, A4 = v.children
26         B1, B2, B3, B4 = w.children
27
28         C1 = matrix_matrix_add(A1, B1, rmax, eps)
29         C2 = matrix_matrix_add(A2, B2, rmax, eps)
30         C3 = matrix_matrix_add(A3, B3, rmax, eps)
31         C4 = matrix_matrix_add(A4, B4, rmax, eps)
32         node.children = [C1,C2,C3,C4]
33         return node

```

Rysunek 5: Dodawanie część 1.

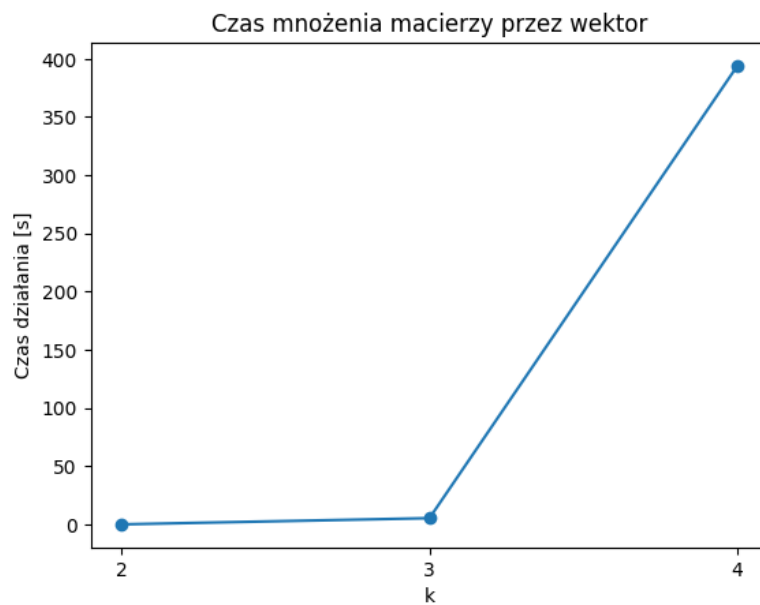
```

1  # 3) liść + węzeł
2  if not v.children and w.children:
3      sub_v = split_leaf_into_4_subleaves(v)
4      node = Node()
5      node.size = v.size
6      node.children = [matrix_matrix_add(sub_v[0], w.children[0], rmax, eps),
7                        matrix_matrix_add(sub_v[1], w.children[1], rmax, eps),
8                        matrix_matrix_add(sub_v[2], w.children[2], rmax, eps),
9                        matrix_matrix_add(sub_v[3], w.children[3], rmax, eps)]
10     return node
11
12 # 4) węzeł + liść
13 if v.children and not w.children:
14     sub_w = split_leaf_into_4_subleaves(w)
15     node = Node()
16     node.size = v.size
17     node.children = [matrix_matrix_add(v.children[0], sub_w[0], rmax, eps),
18                     matrix_matrix_add(v.children[1], sub_w[1], rmax, eps),
19                     matrix_matrix_add(v.children[2], sub_w[2], rmax, eps),
20                     matrix_matrix_add(v.children[3], sub_w[3], rmax, eps)]
21     return node
22
23 raise RuntimeError("Nieobsłużona kombinacja w matrix_matrix_add.")

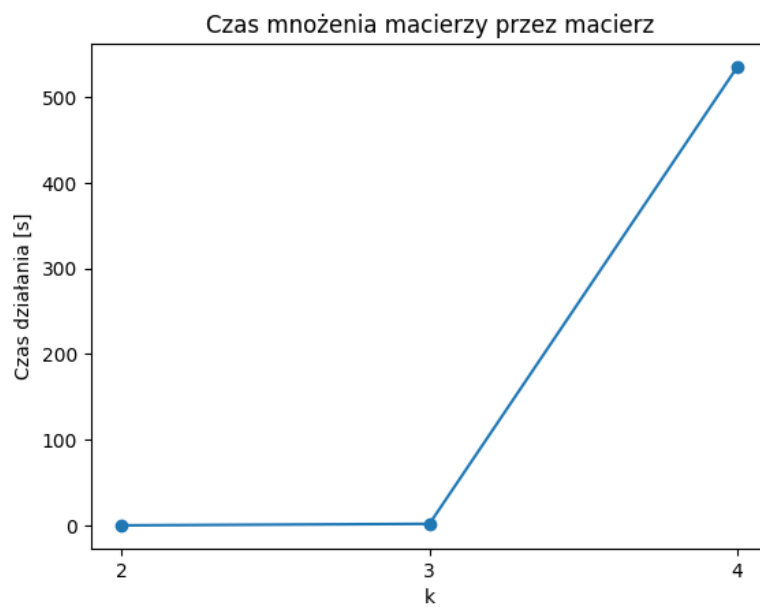
```

Rysunek 6: Dodawanie część 2.

4 Czas działania



Rysunek 7: Wykres czasu działania.



Rysunek 8: Wykres czasu działania.

5 Oszacowanie złożoności obliczeniowej

Do wyników pomiarów, za pomocą funkcji `curve_fit` z biblioteki `scipy.optimize`, dopasowana została krzywa o równaniu

$$y = \alpha N^\beta$$

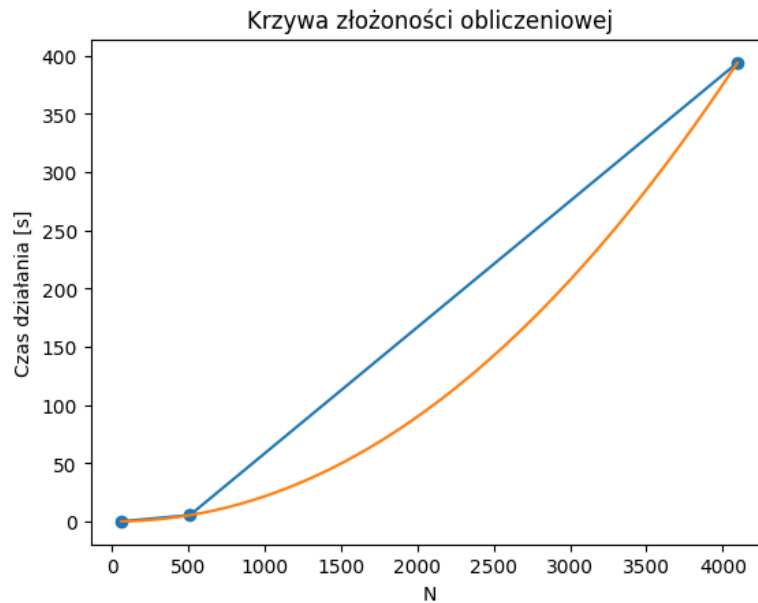
5.1 Mnożenie macierzy i wektora

Dopasowana krzywa:

$$y = 1,41 \cdot 10^{-5} \cdot N^{2,06}$$

Zatem oszacowana złożoność obliczeniowa wynosi:

$$O(n) = n^{2,06}$$



Rysunek 9: Krzywa na wykresie.

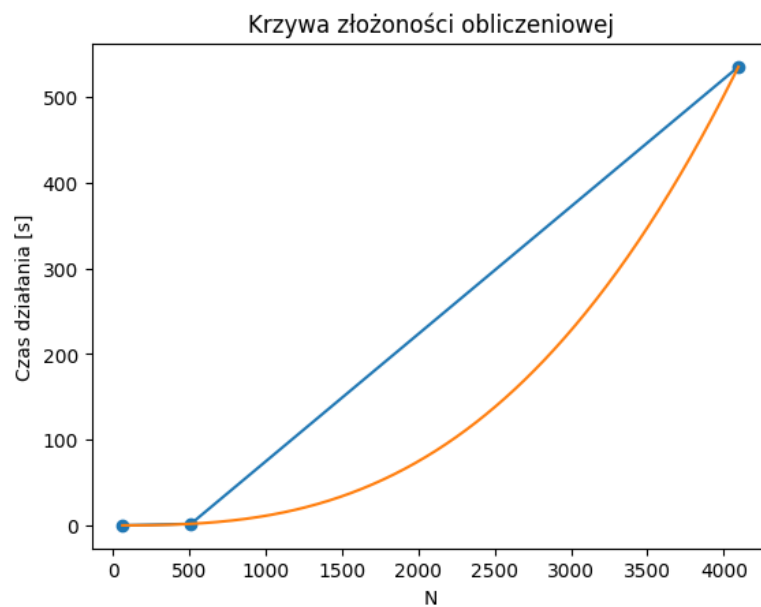
5.2 Mnożenie dwóch macierzy

Dopasowana krzywa:

$$y = 6,4 \cdot 10^{-8} \cdot N^{2,75}$$

Zatem oszacowana złożoność obliczeniowa wynosi:

$$O(n) = n^{2,75}$$



Rysunek 10: Krzywa na wykresie.

6