Imports

In [15]:
```
from queue import PriorityQueue
import string
import numpy as np
from operator import itemgetter
import random
import cv2
import matplotlib.pyplot as plt
import time
```

Zad1 Algorytm znajdowania wzorca 2D

Trie

In [2]:
```
class Node:
    def __init__(self, char="", parent=None):
        self.char = char
        self.children = {}
        self.parent = parent
        self.val = None
        self.fail_link = None
        self.terminal = False

    def add_child(self, char):
        self.children[char] = Node(char, parent=self)

    def has_child(self, char):
        return char in self.children

    def get_child(self, char):
        return self.children.get(char)

    def get_parent(self):
        return self.parent

    def set_fail_link_to(self, other):
        self.fail_link = other

    def get_node_from_fail_link(self):
        return self.fail_link

    def has_fail_link(self):
        return self.fail_link is not None

    def set_terminal(self, val):
        self.output = val
        self.terminal = True

    def is_terminal(self):
        return self.terminal

    def get_terminal_val(self):
        return self.output

class Trie:
    def __init__(self, pattern_mapping):
        self.root = Node()
        self.insert(pattern_mapping)

    def insert(self, pattern_mapping):
        for pattern, mapping in pattern_mapping.items():
            node = self.root
            for c in pattern[:-1]:
                if not node.has_child(c):
                    node.add_child(c)
                node = node.get_child(c)

            if not node.has_child(pattern[-1]):
                node.add_child(pattern[-1])
            node = node.get_child(pattern[-1])
            node.set_terminal(mapping)

    def get_root(self):
        return self.root
```

Aho Corasick Automaton

In [3]:
```
class Aho_Corasick_Automaton:
```

```
    def __init__(self, pattern_mapping):
        self.trie = Trie(pattern_mapping)
        self.create_fail_links()

    def create_fail_links(self):
        i = 0 # rank for priority queue
        queue = PriorityQueue()
        root = self.trie.get_root()

        for key, child in root.children.items():
            child.set_fail_link_to(root)
            queue.put((i, child))
            i += 1

        while not queue.empty():
            _, node = queue.get()
            for key, child in node.children.items():
                queue.put((i, child))
                i += 1
                x = node.get_node_from_fail_link()
                while not x.has_child(key):
                    if x.has_fail_link():
                        x = x.get_node_from_fail_link()
                    else:
                        break
                if not x.has_child(key) and not x.has_fail_link():
                    child.set_fail_link_to(x)
                else:
                    child.set_fail_link_to(x.get_child(key))

    # first loop
    def find_row_patterns(self, matrix):
        m , n = matrix.shape[:2]
        root = self.trie.get_root()

        res_matrix = np.zeros((m, n), dtype=np.int0)
        for i in range(m): # rows
            node = root
            row = matrix[i]
            for j in range(n): # columns
                c = row[j]
                while not node.has_child(c):
                    if node is not root:
                        node = node.get_node_from_fail_link()
                    else:
                        break
                if not node.has_fail_link() and not node.has_child(c):
                    continue
                node = node.get_child(c)

                if node.is_terminal():
                    res_matrix[i,j] = node.get_terminal_val()
        return res_matrix

    # second loop
    def find_col_patterns(self, matrix):
        m, n = matrix.shape[:2]
        root = self.trie.get_root()

        res = []
        for i in range(n): # columns
            node = root
            col = matrix[:, i]
            for j in range(m): # rows
                c = col[j]
                while not node.has_child(c):
                    if node is not root:
                        node = node.get_node_from_fail_link()
                    else:
                        break
                if node is root and not node.has_child(c):
                    continue
                node = node.get_child(c)

                if node.is_terminal():
                    val = node.get_terminal_val()
                    res.append((val, j, i)) # val , row and col idx
        return res
```

Preprocessing text and patterns

In [4]:

```
def text_to_matrix(text):
    lines = text.splitlines(True) # keep linebreaks
    max_row_length = max(len(line) for line in lines) # number of columns

    lines = [line.ljust(max_row_length, "#") for line in lines] # padding
    matrix = np.array(list(map(list, lines)))
```

```
    return matrix

def patterns_mapping(patterns, img=False):
    single_pattern_mapping = dict()
    pattern_2d_mapping = dict()
    patterns_shapes = []

    single_mapping = 0
    mapping_sequence = 0
    for pattern_2d in patterns:
        pattern_2d_mapped_sequence = []
        patterns_shapes.append((len(pattern_2d), len(pattern_2d[0])))
        for pattern in pattern_2d:
            if img:
                pattern = tuple(pattern)
            if pattern not in single_pattern_mapping:
                single_mapping += 1
                single_pattern_mapping[pattern] = single_mapping
            pattern_2d_mapped_sequence.append(single_pattern_mapping[pattern])

        pattern_2d_mapped_sequence = tuple(pattern_2d_mapped_sequence)
        if pattern_2d_mapped_sequence not in pattern_2d_mapping:
            mapping_sequence += 1
            pattern_2d_mapping[pattern_2d_mapped_sequence] = mapping_sequence

    return patterns_shapes, single_pattern_mapping, pattern_2d_mapping

def get_alphabet(text):
    A = set()
    for c in text:
        if c in string.ascii_letters:
            A.add(c)
    return A
```

Multiple 2d patterns matching algorithm

In [58]:
```
def find_2d_matches(text, patterns, img=False, timer=False):
    patterns_shapes, single_pattern_mapping, pattern_2d_mapping = patterns_mapping(patterns, img)

    if img:
        matrix = text.copy()
    else:
        matrix = text_to_matrix(text) # np.array
    m , n = matrix.shape[:2]

    if timer:
        start = time.time()
    automata_rows = Aho_Corasick_Automaton(single_pattern_mapping)
    automata_cols = Aho_Corasick_Automaton(pattern_2d_mapping)
    if timer:
        building_time = time.time() - start
        print("Pattern size : {}".format(patterns_shapes))
        print("Automata building time : {}".format(building_time))


    if timer:
        start = time.time()
    matrix = automata_rows.find_row_patterns(matrix)
    occurences = automata_cols.find_col_patterns(matrix)
    if timer:
        matching_time = time.time() - start
        print("Pattern matching time : {}".format(matching_time))
        print()

    res = []
    for occurence in occurences:
        sequence_mapped_value, row_idx, col_idx = occurence
        pattern_n = sequence_mapped_value - 1
        m_pattern, n_pattern = patterns_shapes[pattern_n]

        # pattern , starting row and col idx
        if img:
            res.append((pattern_n, row_idx - m_pattern + 1, col_idx - n_pattern + 1)) # bcs its big pattern
        else:
            res.append((patterns[pattern_n], row_idx - m_pattern + 1, col_idx - n_pattern + 1))

    if timer:
        return building_time, matching_time
    else:
        return res
```

Example of mapping

In [6]:
```
patterns = [["o", "e"], ["e", "o"]]
patterns_shapes, single_pattern_mapping, pattern_2d_mapping = patterns_mapping(patterns)
```

```
print(patterns_shapes)
print(single_pattern_mapping)
print(pattern_2d_mapping)
```

```
[(2, 1), (2, 1)]
{'o': 1, 'e': 2}
{(1, 2): 1, (2, 1): 2}
```

File reading

In [7]:
```
with open("haystack.txt", "r") as f:
    text = f.read()
```

Zad2

In [8]:
```
def find_all_letters_occurences(text):
    A = get_alphabet(text)
    patterns = []
    for letter in A:
        patterns.append([letter, letter])

    res = find_2d_matches(text, patterns)
    res = sorted(res, key=lambda x : x[0])
    return res
```

In [20]:
```
find_all_letters_occurences(text)
```

Out[20]:
```
[(['a', 'a'], 64, 2),
 (['a', 'a'], 37, 4),
 (['a', 'a'], 20, 6),
 (['a', 'a'], 56, 11),
 (['a', 'a'], 52, 12),
 (['a', 'a'], 53, 12),
 (['a', 'a'], 64, 14),
 (['a', 'a'], 76, 21),
 (['a', 'a'], 64, 22),
 (['a', 'a'], 59, 24),
 (['a', 'a'], 3, 30),
 (['a', 'a'], 65, 35),
 (['a', 'a'], 69, 35),
 (['a', 'a'], 57, 36),
 (['a', 'a'], 58, 36),
 (['a', 'a'], 79, 37),
 (['a', 'a'], 77, 42),
 (['a', 'a'], 53, 48),
 (['a', 'a'], 31, 50),
 (['a', 'a'], 78, 59),
 (['a', 'a'], 5, 60),
 (['a', 'a'], 77, 61),
 (['a', 'a'], 6, 63),
 (['a', 'a'], 33, 66),
 (['a', 'a'], 28, 69),
 (['a', 'a'], 31, 73),
 (['a', 'a'], 76, 74),
 (['a', 'a'], 0, 82),
 (['c', 'c'], 41, 0),
 (['c', 'c'], 68, 0),
 (['c', 'c'], 13, 10),
 (['c', 'c'], 82, 41),
 (['c', 'c'], 10, 45),
 (['c', 'c'], 3, 54),
 (['d', 'd'], 37, 19),
 (['e', 'e'], 10, 1),
 (['e', 'e'], 14, 2),
 (['e', 'e'], 24, 3),
 (['e', 'e'], 17, 6),
 (['e', 'e'], 76, 6),
 (['e', 'e'], 77, 6),
 (['e', 'e'], 80, 6),
 (['e', 'e'], 1, 8),
 (['e', 'e'], 20, 10),
 (['e', 'e'], 40, 11),
 (['e', 'e'], 81, 14),
 (['e', 'e'], 69, 15),
 (['e', 'e'], 67, 17),
 (['e', 'e'], 72, 23),
 (['e', 'e'], 40, 26),
 (['e', 'e'], 18, 27),
```

```
(['e', 'e'], 73, 27),
(['e', 'e'], 51, 31),
(['e', 'e'], 42, 36),
(['e', 'e'], 29, 38),
(['e', 'e'], 71, 38),
(['e', 'e'], 15, 43),
(['e', 'e'], 29, 43),
(['e', 'e'], 68, 46),
(['e', 'e'], 82, 47),
(['e', 'e'], 37, 48),
(['e', 'e'], 42, 48),
(['e', 'e'], 70, 49),
(['e', 'e'], 47, 50),
(['e', 'e'], 58, 50),
(['e', 'e'], 46, 52),
(['e', 'e'], 22, 53),
(['e', 'e'], 57, 54),
(['e', 'e'], 58, 54),
(['e', 'e'], 41, 57),
(['e', 'e'], 21, 61),
(['e', 'e'], 0, 63),
(['e', 'e'], 10, 64),
(['e', 'e'], 7, 65),
(['e', 'e'], 24, 65),
(['e', 'e'], 78, 65),
(['e', 'e'], 63, 66),
(['e', 'e'], 28, 67),
(['e', 'e'], 65, 69),
(['e', 'e'], 66, 72),
(['e', 'e'], 28, 73),
(['e', 'e'], 59, 73),
(['e', 'e'], 4, 77),
(['f', 'f'], 77, 1),
(['f', 'f'], 30, 59),
(['h', 'h'], 27, 2),
(['h', 'h'], 37, 2),
(['h', 'h'], 73, 12),
(['h', 'h'], 56, 31),
(['i', 'i'], 31, 0),
(['i', 'i'], 1, 5),
(['i', 'i'], 73, 13),
(['i', 'i'], 77, 13),
(['i', 'i'], 55, 17),
(['i', 'i'], 31, 31),
(['i', 'i'], 44, 33),
(['i', 'i'], 8, 37),
(['i', 'i'], 60, 45),
(['i', 'i'], 68, 51),
(['i', 'i'], 19, 55),
(['i', 'i'], 9, 60),
(['i', 'i'], 52, 69),
(['l', 'l'], 33, 45),
(['l', 'l'], 53, 45),
(['l', 'l'], 46, 61),
(['l', 'l'], 28, 72),
(['l', 'l'], 41, 77),
(['m', 'm'], 44, 0),
(['m', 'm'], 16, 5),
(['m', 'm'], 34, 40),
(['m', 'm'], 34, 60),
(['m', 'm'], 28, 70),
(['n', 'n'], 31, 1),
(['n', 'n'], 1, 9),
(['n', 'n'], 56, 13),
(['n', 'n'], 35, 18),
(['n', 'n'], 64, 29),
(['n', 'n'], 51, 32),
(['n', 'n'], 54, 33),
(['n', 'n'], 67, 35),
(['n', 'n'], 19, 37),
(['n', 'n'], 67, 40),
(['n', 'n'], 14, 54),
(['n', 'n'], 20, 56),
(['n', 'n'], 67, 57),
(['n', 'n'], 21, 62),
(['n', 'n'], 0, 83),
(['o', 'o'], 41, 1),
(['o', 'o'], 53, 1),
(['o', 'o'], 50, 2),
(['o', 'o'], 52, 8),
(['o', 'o'], 79, 10),
(['o', 'o'], 33, 11),
(['o', 'o'], 27, 17),
```

```
(['o', 'o'], 28, 17),
(['o', 'o'], 33, 26),
(['o', 'o'], 10, 27),
(['o', 'o'], 32, 34),
(['o', 'o'], 6, 38),
(['o', 'o'], 7, 38),
(['o', 'o'], 71, 42),
(['o', 'o'], 58, 45),
(['o', 'o'], 81, 52),
(['o', 'o'], 44, 55),
(['o', 'o'], 30, 58),
(['o', 'o'], 15, 60),
(['o', 'o'], 5, 66),
(['o', 'o'], 4, 75),
(['p', 'p'], 41, 18),
(['p', 'p'], 28, 71),
(['r', 'r'], 1, 4),
(['r', 'r'], 52, 5),
(['r', 'r'], 33, 10),
(['r', 'r'], 7, 13),
(['r', 'r'], 17, 14),
(['r', 'r'], 15, 18),
(['r', 'r'], 69, 22),
(['r', 'r'], 43, 25),
(['r', 'r'], 67, 29),
(['r', 'r'], 60, 30),
(['r', 'r'], 33, 37),
(['r', 'r'], 47, 37),
(['r', 'r'], 6, 39),
(['r', 'r'], 62, 39),
(['r', 'r'], 55, 40),
(['r', 'r'], 46, 42),
(['r', 'r'], 6, 50),
(['r', 'r'], 19, 54),
(['r', 'r'], 20, 54),
(['r', 'r'], 28, 65),
(['r', 'r'], 31, 70),
(['s', 's'], 54, 0),
(['s', 's'], 49, 14),
(['s', 's'], 8, 21),
(['s', 's'], 71, 24),
(['s', 's'], 79, 24),
(['s', 's'], 37, 34),
(['s', 's'], 45, 34),
(['s', 's'], 67, 37),
(['s', 's'], 70, 41),
(['s', 's'], 46, 44),
(['s', 's'], 28, 45),
(['s', 's'], 4, 49),
(['s', 's'], 52, 53),
(['s', 's'], 29, 56),
(['s', 's'], 30, 56),
(['s', 's'], 3, 57),
(['s', 's'], 9, 58),
(['s', 's'], 3, 63),
(['s', 's'], 40, 63),
(['t', 't'], 37, 0),
(['t', 't'], 50, 0),
(['t', 't'], 16, 3),
(['t', 't'], 71, 3),
(['t', 't'], 72, 3),
(['t', 't'], 23, 4),
(['t', 't'], 24, 4),
(['t', 't'], 69, 5),
(['t', 't'], 1, 6),
(['t', 't'], 0, 7),
(['t', 't'], 1, 7),
(['t', 't'], 22, 8),
(['t', 't'], 35, 10),
(['t', 't'], 72, 10),
(['t', 't'], 54, 11),
(['t', 't'], 15, 12),
(['t', 't'], 4, 14),
(['t', 't'], 30, 16),
(['t', 't'], 77, 22),
(['t', 't'], 4, 23),
(['t', 't'], 28, 23),
(['t', 't'], 46, 24),
(['t', 't'], 7, 29),
(['t', 't'], 27, 31),
(['t', 't'], 19, 33),
(['t', 't'], 51, 33),
(['t', 't'], 59, 33),
```

```
(['t', 't'], 3, 37),
(['t', 't'], 41, 45),
(['t', 't'], 58, 49),
(['t', 't'], 28, 52),
(['t', 't'], 55, 54),
(['t', 't'], 13, 55),
(['t', 't'], 61, 56),
(['t', 't'], 72, 59),
(['t', 't'], 52, 61),
(['t', 't'], 67, 71),
(['t', 't'], 41, 73),
(['t', 't'], 8, 75),
(['t', 't'], 59, 75),
(['t', 't'], 58, 78),
(['w', 'w'], 1, 3),
(['w', 'w'], 21, 70),
(['x', 'x'], 28, 68),
(['y', 'y'], 44, 5)]
```

Zad3

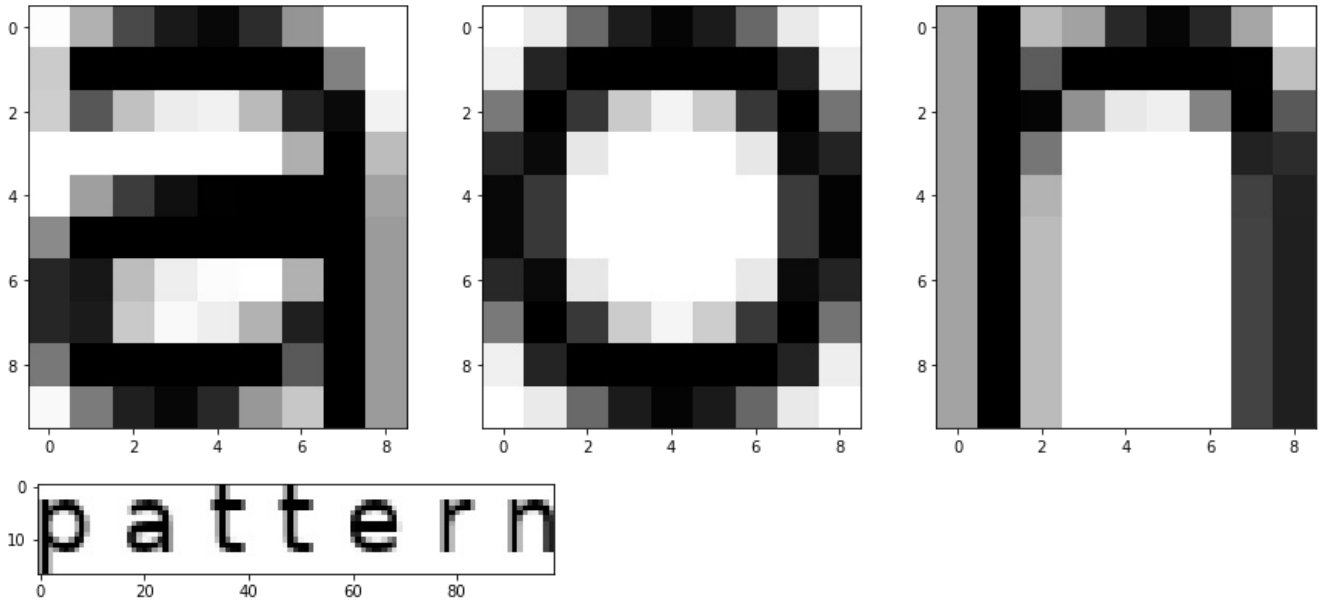
In [10]:
```
patterns = [["th", "th"], ["t h", "t h"]]
find_2d_matches(text, patterns)
```

Out[10]:
```
[(['t h', 't h'], 37, 0)]
```

Zad4

Load all images

In [11]:
```
img = cv2.imread("haystack.png")
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

fig, ax = plt.subplots(1, 3, figsize=(15,15))
img_a = cv2.imread("a.png")
img_a = cv2.cvtColor(img_a, cv2.COLOR_BGR2GRAY)
ax[0].imshow(img_a, cmap="gray")

img_o = cv2.imread("o.png")
img_o = cv2.cvtColor(img_o, cv2.COLOR_BGR2GRAY)
ax[1].imshow(img_o, cmap="gray")

img_n = cv2.imread("n.png")
img_n = cv2.cvtColor(img_n, cv2.COLOR_BGR2GRAY)
ax[2].imshow(img_n, cmap="gray")
plt.show()

img_pattern = cv2.imread("pattern.png")
img_pattern = cv2.cvtColor(img_pattern, cv2.COLOR_BGR2GRAY)
plt.imshow(img_pattern, cmap="gray")
plt.show()
```



Example of mapping img

In [12]:
```
patterns_mapping([img_o], img=True) # letter 'o' is not symmetric in 0-255
```

Out[12]:
```
([(10, 9)],
 {(255, 235, 106, 28, 5, 27, 104, 234, 255): 1,
  (240, 37, 0, 0, 0, 0, 0, 35, 239): 2,
  (122, 0, 54, 202, 244, 203, 55, 0, 116): 3,
  (41, 10, 231, 255, 255, 255, 231, 11, 35): 4,
  (8, 56, 255, 255, 255, 255, 255, 59, 3): 5,
  (8, 57, 255, 255, 255, 255, 255, 59, 3): 6,
  (41, 10, 231, 255, 255, 255, 232, 11, 35): 7,
  (121, 0, 57, 204, 245, 205, 57, 0, 115): 8,
  (240, 37, 0, 0, 0, 0, 0, 35, 238): 9,
  (255, 234, 105, 27, 4, 26, 103, 233, 255): 10},
 {(1, 2, 3, 4, 5, 6, 7, 8, 9, 10): 1})
```

'a', 'o' and 'n' occurences

In [13]:
```
patterns = [img_a, img_o, img_n]
img_positions = find_2d_matches(img, patterns, img=True)
img_positions = sorted(img_positions, key=lambda x : x[0])
img_positions # 0 - img_a, 1 img_o , 2 img_n
```

Out[13]:
```
[(0, 367, 26),
 (0, 411, 26),
 (0, 477, 26),
 (0, 565, 26),
 (0, 675, 26),
 (0, 1379, 26),
 (0, 1423, 26),
 (0, 1775, 26),
 (0, 1863, 26),
 (0, 785, 31),
 (0, 213, 32),
 (0, 1555, 36),
 (0, 301, 37),
 (0, 1049, 37),
 (0, 1643, 37),
 (0, 1841, 43),
 (0, 1005, 44),
 (0, 1313, 45),
 (0, 1445, 47),
 (0, 1335, 49),
 (0, 1181, 50),
 (0, 1467, 53),
 (0, 543, 55),
 (0, 103, 56),
 (0, 169, 56),
 (0, 653, 56),
 (0, 851, 56),
 (0, 873, 56),
 (0, 1687, 60),
 (0, 1533, 61),
 (0, 279, 63),
 (0, 1357, 64),
 (0, 521, 65),
 (0, 1445, 65),
 (0, 1797, 69),
 (0, 257, 72),
 (0, 1863, 72),
 (0, 785, 75),
 (0, 1775, 77),
 (0, 499, 78),
 (0, 1269, 78),
 (0, 1137, 80),
 (0, 1665, 80),
 (0, 477, 82),
 (0, 939, 83),
 (0, 191, 84),
 (0, 455, 84),
 (0, 323, 88),
 (0, 169, 90),
 (0, 675, 90),
 (0, 697, 93),
```

```
(0, 1555, 93),
(0, 257, 94),
(0, 631, 97),
(0, 345, 98),
(0, 1093, 100),
(0, 1511, 100),
(0, 103, 105),
(0, 125, 105),
(0, 895, 106),
(0, 1687, 107),
(0, 1137, 110),
(0, 807, 111),
(0, 1643, 115),
(0, 235, 120),
(0, 389, 120),
(0, 1291, 120),
(0, 1379, 122),
(0, 147, 126),
(0, 279, 126),
(0, 543, 127),
(0, 455, 128),
(0, 829, 132),
(0, 1203, 132),
(0, 1401, 132),
(0, 1269, 136),
(0, 1181, 138),
(0, 1555, 141),
(0, 1709, 145),
(0, 1445, 146),
(0, 521, 148),
(0, 1225, 148),
(0, 1357, 150),
(0, 1247, 153),
(0, 1467, 153),
(0, 1621, 155),
(0, 147, 156),
(0, 587, 159),
(0, 499, 160),
(0, 1181, 160),
(0, 411, 164),
(0, 565, 164),
(0, 1159, 165),
(0, 477, 167),
(0, 1027, 167),
(0, 1709, 167),
(0, 961, 169),
(0, 1335, 170),
(0, 1049, 171),
(0, 323, 176),
(0, 1401, 176),
(0, 81, 177),
(0, 1291, 177),
(0, 1599, 179),
(0, 1181, 182),
(0, 1489, 182),
(0, 1797, 189),
(0, 1731, 191),
(0, 1137, 193),
(0, 169, 194),
(0, 741, 195),
(0, 521, 197),
(0, 1599, 197),
(0, 477, 201),
(0, 1049, 201),
(0, 81, 203),
(0, 1313, 203),
(0, 37, 207),
(0, 169, 212),
(0, 1269, 213),
(0, 807, 217),
(0, 653, 218),
(0, 1137, 218),
(0, 1709, 218),
(0, 1841, 218),
(0, 961, 220),
(0, 1467, 220),
(0, 345, 222),
(0, 1511, 224),
(0, 521, 225),
(0, 895, 226),
(0, 1181, 226),
(0, 1731, 226),
(0, 389, 228),
```

```
(0, 917, 229),
(0, 1445, 230),
(0, 1357, 231),
(0, 1335, 232),
(0, 1819, 239),
(0, 851, 243),
(0, 455, 245),
(0, 147, 247),
(0, 367, 248),
(0, 433, 248),
(0, 191, 258),
(0, 543, 259),
(0, 587, 259),
(0, 1313, 259),
(0, 719, 261),
(0, 1247, 262),
(0, 37, 263),
(0, 631, 265),
(0, 1225, 266),
(0, 1753, 266),
(0, 1379, 268),
(0, 279, 269),
(0, 1467, 269),
(0, 103, 272),
(0, 345, 273),
(0, 125, 274),
(0, 367, 274),
(0, 565, 276),
(0, 1357, 277),
(0, 829, 281),
(0, 433, 283),
(0, 1731, 284),
(0, 1555, 291),
(0, 1203, 292),
(0, 1643, 292),
(0, 1071, 295),
(0, 1357, 295),
(0, 653, 297),
(0, 1731, 302),
(0, 59, 303),
(0, 1247, 303),
(0, 1797, 308),
(0, 477, 309),
(0, 1753, 309),
(0, 1863, 317),
(0, 169, 322),
(0, 1599, 322),
(0, 37, 323),
(0, 829, 323),
(0, 1071, 325),
(0, 125, 328),
(0, 1313, 329),
(0, 1335, 331),
(0, 59, 333),
(0, 763, 333),
(0, 1467, 335),
(0, 1489, 338),
(0, 1621, 338),
(0, 1731, 343),
(0, 1423, 344),
(0, 1577, 345),
(0, 1555, 346),
(0, 917, 347),
(0, 499, 348),
(0, 1379, 349),
(0, 1181, 351),
(0, 1225, 351),
(0, 1313, 351),
(0, 103, 353),
(0, 147, 354),
(0, 1291, 355),
(0, 1027, 356),
(0, 543, 359),
(0, 1775, 361),
(0, 1533, 363),
(0, 1797, 366),
(0, 455, 367),
(0, 873, 369),
(0, 389, 370),
(0, 1313, 372),
(0, 785, 375),
(0, 1247, 375),
(0, 1203, 376),
```

```
(0, 1335, 379),
(0, 675, 381),
(0, 1423, 384),
(0, 1797, 384),
(0, 1863, 386),
(0, 1731, 387),
(0, 389, 388),
(0, 961, 390),
(0, 1753, 390),
(0, 59, 391),
(0, 1247, 393),
(0, 1203, 398),
(0, 81, 399),
(0, 279, 400),
(0, 1709, 400),
(0, 191, 406),
(0, 1775, 406),
(0, 1819, 406),
(0, 829, 407),
(0, 631, 408),
(0, 1841, 408),
(0, 59, 409),
(0, 345, 409),
(0, 1225, 411),
(0, 1291, 414),
(0, 587, 416),
(0, 81, 417),
(0, 653, 419),
(0, 1005, 421),
(0, 1423, 430),
(0, 1027, 434),
(0, 213, 435),
(0, 323, 435),
(0, 1115, 436),
(0, 1731, 438),
(0, 1049, 440),
(0, 1313, 440),
(0, 1291, 442),
(0, 1225, 444),
(0, 499, 445),
(0, 257, 446),
(0, 565, 447),
(0, 455, 449),
(0, 1203, 451),
(0, 521, 452),
(0, 1511, 456),
(0, 279, 460),
(0, 785, 460),
(0, 719, 463),
(0, 741, 463),
(0, 697, 470),
(0, 1093, 470),
(0, 235, 472),
(0, 939, 474),
(0, 367, 476),
(0, 1643, 484),
(0, 37, 487),
(0, 1225, 488),
(0, 1467, 488),
(0, 1489, 488),
(0, 1577, 489),
(0, 191, 492),
(0, 323, 495),
(0, 565, 496),
(0, 1753, 497),
(0, 1841, 498),
(0, 1511, 500),
(0, 1643, 502),
(0, 785, 504),
(0, 939, 504),
(0, 1731, 508),
(0, 1775, 509),
(0, 433, 512),
(0, 587, 513),
(0, 169, 518),
(0, 1445, 523),
(0, 1599, 528),
(0, 543, 529),
(0, 213, 536),
(0, 1049, 536),
(0, 741, 537),
(0, 1203, 541),
(0, 1511, 545),
```

```
(0, 653, 548),
(0, 147, 549),
(0, 785, 552),
(0, 587, 553),
(0, 1071, 555),
(0, 1753, 555),
(0, 169, 557),
(0, 1731, 559),
(0, 1775, 561),
(0, 279, 563),
(0, 81, 565),
(0, 477, 565),
(0, 1269, 566),
(0, 1379, 568),
(0, 1445, 568),
(0, 631, 569),
(0, 983, 569),
(0, 1467, 572),
(0, 323, 573),
(0, 1753, 573),
(0, 1357, 575),
(0, 191, 577),
(0, 345, 577),
(0, 59, 578),
(0, 1335, 578),
(0, 763, 580),
(0, 1049, 585),
(0, 169, 586),
(0, 1071, 588),
(0, 81, 589),
(0, 1225, 589),
(0, 1775, 589),
(0, 235, 593),
(0, 433, 594),
(0, 1005, 599),
(0, 499, 601),
(0, 829, 601),
(0, 367, 604),
(0, 1291, 604),
(0, 983, 611),
(0, 785, 614),
(0, 1731, 617),
(0, 125, 619),
(0, 191, 621),
(0, 37, 623),
(0, 1313, 626),
(0, 1797, 626),
(0, 565, 630),
(0, 807, 630),
(0, 147, 632),
(0, 1225, 632),
(0, 1643, 635),
(0, 455, 637),
(0, 1115, 638),
(0, 1379, 638),
(0, 697, 641),
(0, 873, 641),
(0, 1357, 641),
(0, 59, 642),
(0, 675, 642),
(0, 719, 644),
(0, 1181, 644),
(0, 785, 647),
(0, 961, 652),
(0, 433, 655),
(0, 653, 655),
(0, 367, 657),
(0, 1225, 662),
(0, 1665, 662),
(0, 1291, 664),
(0, 741, 668),
(0, 1005, 668),
(0, 1071, 672),
(0, 1511, 672),
(0, 763, 673),
(0, 697, 674),
(0, 1621, 678),
(0, 191, 680),
(0, 1797, 682),
(0, 1731, 685),
(0, 1709, 688),
(0, 499, 695),
(0, 961, 695),
```

```
(0, 543, 697),
(0, 1313, 699),
(0, 59, 700),
(0, 389, 705),
(0, 169, 708),
(0, 1445, 712),
(0, 477, 713),
(0, 1115, 728),
(0, 763, 733),
(0, 1225, 738),
(0, 499, 744),
(0, 587, 750),
(0, 37, 751),
(0, 1115, 804),
(1, 59, 26),
(1, 147, 26),
(1, 235, 26),
(1, 697, 26),
(1, 1335, 26),
(1, 1621, 26),
(1, 1687, 26),
(1, 1731, 26),
(1, 1159, 33),
(1, 1203, 33),
(1, 829, 34),
(1, 1225, 35),
(1, 939, 36),
(1, 961, 36),
(1, 1599, 36),
(1, 1071, 37),
(1, 433, 39),
(1, 1137, 39),
(1, 125, 44),
(1, 895, 44),
(1, 1159, 44),
(1, 191, 48),
(1, 1071, 48),
(1, 455, 52),
(1, 675, 61),
(1, 1203, 65),
(1, 1775, 66),
(1, 235, 67),
(1, 37, 68),
(1, 1291, 69),
(1, 1555, 69),
(1, 367, 70),
(1, 1027, 73),
(1, 301, 75),
(1, 565, 76),
(1, 345, 81),
(1, 719, 81),
(1, 1181, 91),
(1, 1401, 93),
(1, 1577, 93),
(1, 1709, 94),
(1, 829, 96),
(1, 1291, 97),
(1, 1203, 99),
(1, 1599, 100),
(1, 1731, 100),
(1, 1181, 102),
(1, 983, 104),
(1, 191, 107),
(1, 213, 112),
(1, 1159, 115),
(1, 763, 117),
(1, 1863, 118),
(1, 961, 119),
(1, 1577, 121),
(1, 1775, 122),
(1, 411, 124),
(1, 719, 125),
(1, 1797, 128),
(1, 785, 132),
(1, 81, 134),
(1, 807, 134),
(1, 1687, 135),
(1, 631, 142),
(1, 1489, 142),
(1, 477, 144),
(1, 169, 145),
(1, 917, 146),
(1, 1533, 146),
```

```
(1, 1093, 149),
(1, 257, 151),
(1, 191, 153),
(1, 213, 154),
(1, 1423, 155),
(1, 1071, 159),
(1, 433, 163),
(1, 653, 170),
(1, 895, 170),
(1, 1753, 172),
(1, 675, 176),
(1, 631, 180),
(1, 873, 183),
(1, 125, 185),
(1, 345, 186),
(1, 367, 186),
(1, 1445, 186),
(1, 411, 187),
(1, 851, 192),
(1, 697, 193),
(1, 807, 193),
(1, 1533, 194),
(1, 1775, 199),
(1, 917, 207),
(1, 1709, 207),
(1, 961, 209),
(1, 1335, 209),
(1, 939, 210),
(1, 1621, 212),
(1, 565, 214),
(1, 1225, 214),
(1, 1071, 215),
(1, 323, 216),
(1, 1643, 216),
(1, 1863, 221),
(1, 741, 222),
(1, 1247, 223),
(1, 631, 224),
(1, 81, 225),
(1, 433, 225),
(1, 1533, 227),
(1, 1555, 229),
(1, 675, 236),
(1, 279, 240),
(1, 1027, 240),
(1, 1841, 241),
(1, 345, 245),
(1, 1005, 245),
(1, 983, 247),
(1, 763, 250),
(1, 829, 252),
(1, 1577, 254),
(1, 1731, 255),
(1, 1423, 259),
(1, 1071, 261),
(1, 59, 263),
(1, 1709, 263),
(1, 785, 268),
(1, 961, 269),
(1, 1401, 270),
(1, 257, 273),
(1, 1269, 273),
(1, 499, 281),
(1, 1445, 281),
(1, 521, 285),
(1, 1797, 285),
(1, 1093, 287),
(1, 1313, 287),
(1, 1511, 287),
(1, 1863, 287),
(1, 851, 288),
(1, 587, 297),
(1, 455, 298),
(1, 1027, 299),
(1, 939, 300),
(1, 1577, 302),
(1, 499, 303),
(1, 785, 305),
(1, 147, 308),
(1, 235, 311),
(1, 763, 311),
(1, 1841, 311),
(1, 1335, 314),
```

```
(1, 1533, 316),
(1, 961, 317),
(1, 1269, 318),
(1, 1401, 319),
(1, 1555, 320),
(1, 741, 323),
(1, 807, 324),
(1, 345, 329),
(1, 433, 329),
(1, 389, 330),
(1, 213, 332),
(1, 1269, 336),
(1, 1797, 337),
(1, 1819, 339),
(1, 785, 346),
(1, 213, 348),
(1, 169, 349),
(1, 323, 350),
(1, 1863, 350),
(1, 125, 351),
(1, 587, 353),
(1, 81, 355),
(1, 719, 355),
(1, 1643, 356),
(1, 345, 357),
(1, 1269, 357),
(1, 829, 363),
(1, 1467, 363),
(1, 1753, 365),
(1, 323, 366),
(1, 565, 366),
(1, 1049, 369),
(1, 191, 373),
(1, 1181, 374),
(1, 1467, 374),
(1, 1665, 375),
(1, 1093, 376),
(1, 1819, 378),
(1, 125, 379),
(1, 983, 379),
(1, 1357, 380),
(1, 763, 382),
(1, 1511, 382),
(1, 1401, 383),
(1, 1665, 386),
(1, 1621, 391),
(1, 873, 392),
(1, 257, 395),
(1, 477, 399),
(1, 587, 399),
(1, 1181, 402),
(1, 37, 406),
(1, 235, 407),
(1, 1797, 408),
(1, 1335, 411),
(1, 1599, 415),
(1, 1577, 417),
(1, 1467, 419),
(1, 103, 420),
(1, 873, 420),
(1, 169, 423),
(1, 1313, 423),
(1, 1709, 423),
(1, 1181, 425),
(1, 1401, 428),
(1, 719, 429),
(1, 1357, 430),
(1, 367, 436),
(1, 1753, 439),
(1, 763, 448),
(1, 37, 451),
(1, 1709, 451),
(1, 675, 454),
(1, 1621, 461),
(1, 213, 462),
(1, 191, 463),
(1, 1181, 464),
(1, 1555, 464),
(1, 1423, 465),
(1, 477, 467),
(1, 147, 471),
(1, 103, 473),
(1, 1247, 473),
```

```
(1, 1731, 473),
(1, 1203, 474),
(1, 1401, 475),
(1, 257, 478),
(1, 1027, 479),
(1, 1665, 480),
(1, 1291, 481),
(1, 1841, 481),
(1, 147, 482),
(1, 873, 485),
(1, 741, 486),
(1, 345, 489),
(1, 433, 490),
(1, 235, 499),
(1, 367, 499),
(1, 1863, 499),
(1, 697, 503),
(1, 1005, 503),
(1, 763, 505),
(1, 1269, 505),
(1, 37, 510),
(1, 521, 511),
(1, 1423, 513),
(1, 697, 514),
(1, 741, 514),
(1, 1709, 516),
(1, 1819, 516),
(1, 719, 521),
(1, 873, 522),
(1, 1093, 523),
(1, 1247, 523),
(1, 1621, 523),
(1, 653, 526),
(1, 1753, 526),
(1, 147, 527),
(1, 279, 528),
(1, 1467, 528),
(1, 1027, 529),
(1, 807, 532),
(1, 1313, 533),
(1, 1489, 533),
(1, 983, 540),
(1, 1379, 540),
(1, 1665, 544),
(1, 1731, 548),
(1, 697, 550),
(1, 939, 550),
(1, 829, 551),
(1, 1841, 552),
(1, 389, 556),
(1, 367, 558),
(1, 1797, 558),
(1, 1115, 561),
(1, 1027, 563),
(1, 543, 564),
(1, 675, 567),
(1, 1621, 568),
(1, 851, 569),
(1, 1005, 569),
(1, 455, 570),
(1, 1533, 572),
(1, 389, 582),
(1, 1093, 585),
(1, 1841, 587),
(1, 939, 590),
(1, 455, 592),
(1, 1819, 593),
(1, 1599, 594),
(1, 1335, 601),
(1, 147, 603),
(1, 1027, 605),
(1, 807, 607),
(1, 543, 609),
(1, 653, 609),
(1, 1643, 611),
(1, 169, 612),
(1, 697, 612),
(1, 1577, 614),
(1, 1071, 616),
(1, 235, 618),
(1, 1753, 619),
(1, 1819, 621),
(1, 1423, 622),
```

```
(1, 1467, 623),
(1, 741, 626),
(1, 1379, 627),
(1, 1335, 629),
(1, 1005, 632),
(1, 235, 635),
(1, 345, 638),
(1, 851, 638),
(1, 763, 639),
(1, 1665, 640),
(1, 37, 646),
(1, 389, 646),
(1, 1071, 649),
(1, 983, 651),
(1, 1621, 656),
(1, 807, 657),
(1, 279, 659),
(1, 587, 660),
(1, 1445, 660),
(1, 1093, 663),
(1, 1313, 663),
(1, 1181, 665),
(1, 257, 667),
(1, 1423, 672),
(1, 433, 673),
(1, 235, 674),
(1, 125, 675),
(1, 389, 677),
(1, 367, 680),
(1, 1467, 687),
(1, 1841, 687),
(1, 1093, 691),
(1, 147, 692),
(1, 1027, 692),
(1, 1291, 692),
(1, 1555, 692),
(1, 1049, 696),
(1, 719, 697),
(1, 1423, 700),
(1, 279, 703),
(1, 1027, 703),
(1, 1401, 709),
(1, 851, 716),
(1, 631, 721),
(1, 1313, 722),
(1, 719, 729),
(1, 543, 732),
(1, 1115, 774),
(1, 543, 782),
(1, 499, 786),
(1, 1291, 786),
(2, 191, 31),
(2, 719, 31),
(2, 741, 31),
(2, 499, 33),
(2, 1445, 36),
(2, 1489, 36),
(2, 521, 37),
(2, 1379, 37),
(2, 1709, 37),
(2, 1863, 37),
(2, 1269, 38),
(2, 169, 39),
(2, 653, 39),
(2, 851, 39),
(2, 873, 39),
(2, 37, 40),
(2, 455, 41),
(2, 785, 42),
(2, 323, 43),
(2, 477, 43),
(2, 631, 43),
(2, 1423, 43),
(2, 983, 44),
(2, 257, 47),
(2, 961, 47),
(2, 1555, 47),
(2, 1599, 47),
(2, 587, 48),
(2, 433, 50),
(2, 741, 55),
(2, 411, 56),
(2, 697, 56),
```

```
(2, 1115, 56),
(2, 1313, 56),
(2, 1621, 56),
(2, 301, 58),
(2, 829, 58),
(2, 1555, 58),
(2, 1797, 58),
(2, 499, 61),
(2, 1313, 67),
(2, 1423, 70),
(2, 1709, 72),
(2, 169, 73),
(2, 653, 73),
(2, 873, 73),
(2, 1533, 77),
(2, 1071, 80),
(2, 1291, 80),
(2, 1643, 80),
(2, 1753, 80),
(2, 1819, 80),
(2, 961, 81),
(2, 1489, 81),
(2, 741, 88),
(2, 1225, 92),
(2, 301, 93),
(2, 807, 93),
(2, 1467, 93),
(2, 939, 94),
(2, 455, 95),
(2, 653, 97),
(2, 37, 98),
(2, 1775, 98),
(2, 59, 101),
(2, 675, 101),
(2, 367, 102),
(2, 1577, 104),
(2, 257, 105),
(2, 1841, 105),
(2, 81, 106),
(2, 829, 107),
(2, 851, 108),
(2, 147, 109),
(2, 279, 109),
(2, 1533, 109),
(2, 521, 110),
(2, 543, 110),
(2, 1203, 110),
(2, 587, 114),
(2, 719, 114),
(2, 433, 117),
(2, 1247, 117),
(2, 1797, 117),
(2, 191, 118),
(2, 1049, 120),
(2, 1753, 121),
(2, 631, 125),
(2, 565, 127),
(2, 1863, 129),
(2, 1599, 130),
(2, 983, 132),
(2, 1379, 133),
(2, 1731, 135),
(2, 367, 137),
(2, 1291, 137),
(2, 1643, 143),
(2, 543, 144),
(2, 807, 145),
(2, 1269, 147),
(2, 1181, 149),
(2, 1665, 149),
(2, 1577, 151),
(2, 961, 152),
(2, 1797, 154),
(2, 169, 156),
(2, 719, 156),
(2, 1269, 158),
(2, 675, 159),
(2, 1225, 159),
(2, 1643, 159),
(2, 81, 160),
(2, 1093, 160),
(2, 741, 161),
(2, 1753, 161),
```

```
(2, 1467, 164),
(2, 455, 166),
(2, 1357, 167),
(2, 1599, 168),
(2, 345, 169),
(2, 1005, 173),
(2, 565, 175),
(2, 1665, 175),
(2, 1797, 178),
(2, 763, 179),
(2, 1577, 179),
(2, 807, 182),
(2, 169, 183),
(2, 1555, 183),
(2, 829, 184),
(2, 59, 186),
(2, 1225, 186),
(2, 1313, 186),
(2, 103, 187),
(2, 785, 187),
(2, 1511, 187),
(2, 389, 193),
(2, 983, 194),
(2, 1819, 195),
(2, 1159, 196),
(2, 235, 197),
(2, 411, 198),
(2, 1511, 198),
(2, 1841, 200),
(2, 279, 201),
(2, 565, 203),
(2, 1467, 203),
(2, 697, 204),
(2, 1643, 205),
(2, 191, 208),
(2, 543, 208),
(2, 213, 209),
(2, 895, 215),
(2, 37, 218),
(2, 917, 218),
(2, 1753, 220),
(2, 1797, 220),
(2, 125, 222),
(2, 829, 225),
(2, 1071, 226),
(2, 1819, 228),
(2, 587, 232),
(2, 1643, 234),
(2, 81, 236),
(2, 1181, 237),
(2, 1709, 239),
(2, 961, 241),
(2, 719, 244),
(2, 1269, 245),
(2, 37, 246),
(2, 1203, 246),
(2, 1291, 246),
(2, 1225, 248),
(2, 873, 250),
(2, 741, 252),
(2, 1841, 252),
(2, 1489, 255),
(2, 345, 256),
(2, 389, 256),
(2, 939, 257),
(2, 895, 258),
(2, 653, 261),
(2, 829, 263),
(2, 1357, 266),
(2, 235, 267),
(2, 1621, 267),
(2, 521, 268),
(2, 191, 269),
(2, 587, 270),
(2, 1423, 270),
(2, 389, 272),
(2, 477, 274),
(2, 1137, 278),
(2, 697, 279),
(2, 785, 279),
(2, 1489, 279),
(2, 1599, 279),
(2, 1291, 280),
```

```
(2, 1467, 280),
(2, 323, 283),
(2, 345, 284),
(2, 125, 285),
(2, 279, 285),
(2, 895, 285),
(2, 1709, 287),
(2, 1533, 288),
(2, 1467, 291),
(2, 1445, 292),
(2, 1335, 296),
(2, 829, 297),
(2, 1093, 298),
(2, 213, 299),
(2, 851, 299),
(2, 741, 301),
(2, 983, 303),
(2, 1291, 304),
(2, 1599, 305),
(2, 961, 306),
(2, 279, 307),
(2, 565, 307),
(2, 1269, 307),
(2, 1467, 307),
(2, 1665, 309),
(2, 1027, 310),
(2, 1423, 310),
(2, 1819, 311),
(2, 81, 317),
(2, 1225, 317),
(2, 345, 318),
(2, 1005, 318),
(2, 1247, 320),
(2, 455, 322),
(2, 763, 322),
(2, 521, 324),
...]
```

Zad5

'p a t t e r n' occurences

In [14]:
```
patterns = [img_pattern]
find_2d_matches(img, patterns, img=True)
```

Out[14]:
```
[(0, 584, 142), (0, 474, 184), (0, 540, 242), (0, 628, 248), (0, 496, 428)]
```

Zad6

Patterns

In [93]:
```
# img_a 10x8
# img_pattern 17x99

size100 = cv2.imread("size100.png")
size100 = cv2.cvtColor(size100, cv2.COLOR_BGR2GRAY)

size200 = cv2.imread("size200.png")
size200 = cv2.cvtColor(size200, cv2.COLOR_BGR2GRAY)

size400 = cv2.imread("size400.png")
size400 = cv2.cvtColor(size400, cv2.COLOR_BGR2GRAY)

size800 = cv2.imread("size800.png")
size800 = cv2.cvtColor(size800, cv2.COLOR_BGR2GRAY)
```

In [94]:
```
def time_test(imgs, patterns):
    n_imgs = len(imgs)
    n_patterns = len(patterns)

    building_times = [0]*n_patterns
    matching_times = [0]*n_patterns
```

```
    for i in range(n_imgs):
        for j in range(n_patterns):
            res = find_2d_matches(imgs[i], patterns[j], img=True, timer=True)
            building_times[j] += res[0]
            matching_times[j] += res[1]
    return building_times, matching_times
```

Times

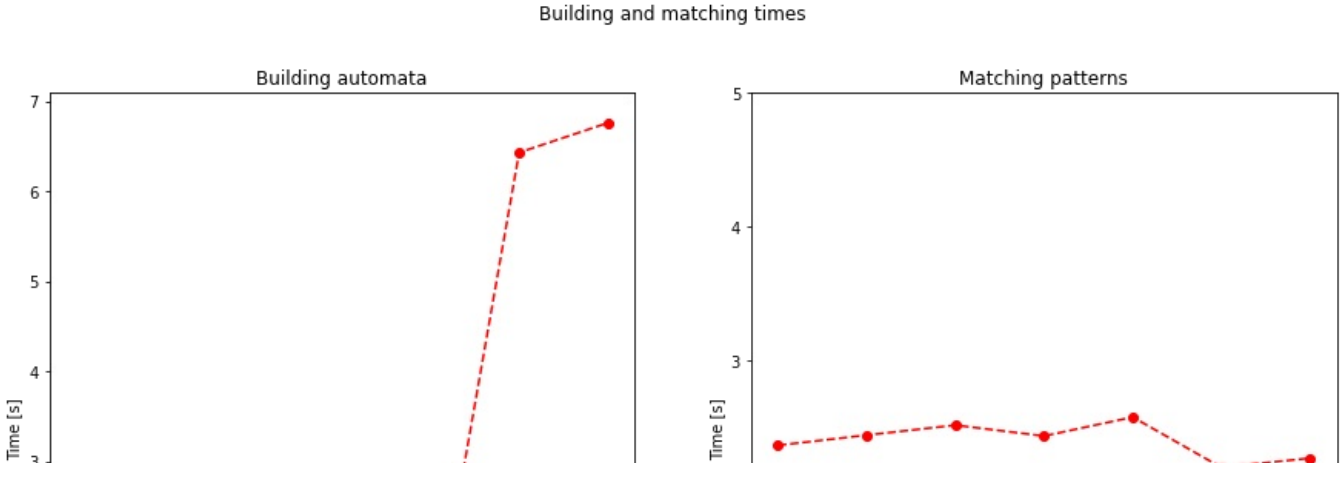
In [95]:
```
all_patterns = [img_a, img_pattern, size100, size200, size400, size800]
patterns = [[img_a], [img_pattern], [size100], [size200], [size400], [size800], all_patterns]
res = time_test([img], patterns)
building_times = res[0]
matching_times = res[1]
```

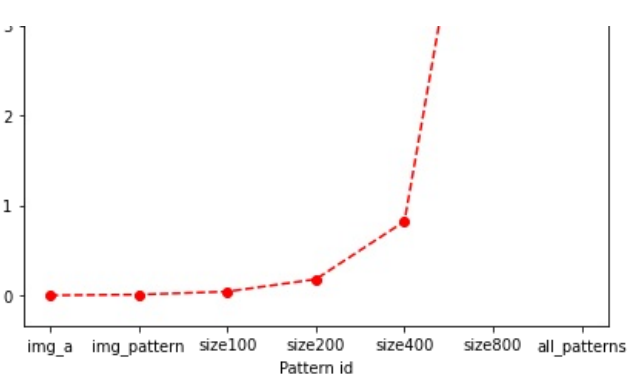
```
Pattern size : [(10, 9)]
Automata building time : 0.0010018348693847656
Pattern matching time : 2.3642303943634033

Pattern size : [(17, 99)]
Automata building time : 0.009999275207519531
Pattern matching time : 2.440490484237671

Pattern size : [(100, 100)]
Automata building time : 0.0429990291595459
Pattern matching time : 2.514774799346924

Pattern size : [(200, 200)]
Automata building time : 0.18000125885009766
Pattern matching time : 2.4339022636413574

Pattern size : [(400, 400)]
Automata building time : 0.8229978084564209
Pattern matching time : 2.57515549659729

Pattern size : [(800, 800)]
Automata building time : 6.4292707443237305
Pattern matching time : 2.1990089416503906

Pattern size : [(10, 9), (17, 99), (100, 100), (200, 200), (400, 400), (800, 800)]
Automata building time : 6.758389711380005
Pattern matching time : 2.2682864665985107
```

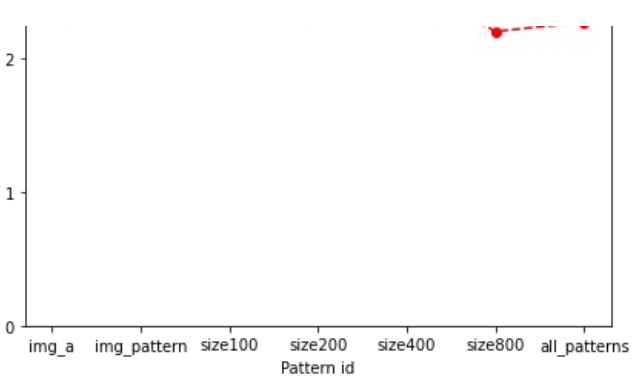
Plot times

In [96]:
```
patterns_id = ["img_a", "img_pattern", "size100", "size200", "size400", "size800", "all_patterns"]

fig, ax = plt.subplots(1, 2, figsize=(15,8))
fig.suptitle("Building and matching times")
ax[0].plot(patterns_id, building_times, "r--o")
ax[0].set_title("Building automata")
ax[0].set_xlabel("Pattern id")
ax[0].set_ylabel("Time [s]")
ax[1].plot(patterns_id, matching_times, "r--o")
ax[1].set_ylim([0, 5])
ax[1].set_title("Matching patterns")
ax[1].set_xlabel("Pattern id")
ax[1].set_ylabel("Time [s]")
```

Out[96]:
```
Text(0, 0.5, 'Time [s]')
```







Zad7

2, 4, 8 fragments

In [ ]:
```
Dla każdego fragmentu automat jest budowany od nowa ale nie ma to znaczenia dla czasów przeszukiwania.
```

In [97]:
```
img.shape
```

Out[97]:
```
(1900, 860)
```

In [98]:
```
img_2 = [img[i*img.shape[0]//2:(i+1)*img.shape[0]//2, :] for i in range(2)]
print(img_2[0].shape, img_2[1].shape)
img_4 = [img[i*img.shape[0]//4:(i+1)*img.shape[0]//4, :] for i in range(4)]
print(img_4[0].shape, img_4[1].shape, img_4[2].shape, img_4[3].shape)
img_8 = [img[i*img.shape[0]//8:(i+1)*img.shape[0]//8, :] for i in range(8)]
print(img_8[0].shape, img_8[1].shape, img_8[2].shape, img_8[3].shape,
      img_8[4].shape, img_8[5].shape, img_8[6].shape, img_8[7].shape)
```

```
(950, 860) (950, 860)
(475, 860) (475, 860) (475, 860) (475, 860)
(237, 860) (238, 860) (237, 860) (238, 860) (237, 860) (238, 860) (237, 860) (238, 860)
```

Img 2 fragments

In [99]:
```
all_patterns = [img_a, img_pattern, size100, size200, size400, size800]
patterns = [[img_a], [img_pattern], [size100], [size200], [size400], [size800], all_patterns]
res = time_test(img_2, patterns)
building_times = res[0]
matching_times = res[1]
```

```
Pattern size : [(10, 9)]
Automata building time : 0.0012252330780029297
Pattern matching time : 1.3282525539398193

Pattern size : [(17, 99)]
Automata building time : 0.00896763801574707
Pattern matching time : 1.1780564785003662

Pattern size : [(100, 100)]
Automata building time : 0.04705333709716797
Pattern matching time : 1.2867436408996582

Pattern size : [(200, 200)]
Automata building time : 0.2090623378753662
Pattern matching time : 1.2780427932739258

Pattern size : [(400, 400)]
Automata building time : 1.6142349243164062
Pattern matching time : 1.3548712730407715

Pattern size : [(800, 800)]
Automata building time : 4.921627521514893
Pattern matching time : 1.183304786682129

Pattern size : [(10, 9), (17, 99), (100, 100), (200, 200), (400, 400), (800, 800)]
Automata building time : 6.599594831466675
Pattern matching time : 1.2499055862426758
```

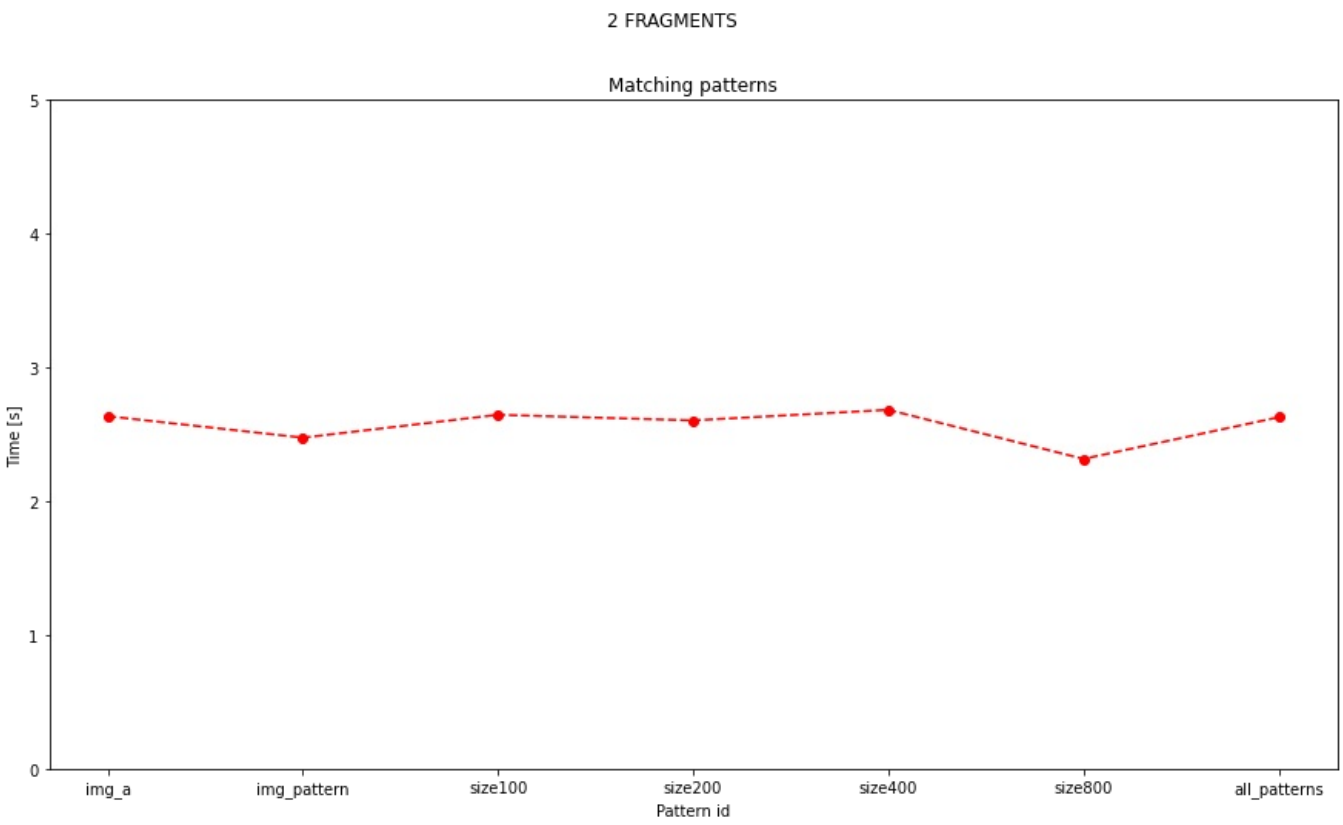
```
Pattern size : [(10, 9)]
Automata building time : 0.0010001659393310547
Pattern matching time : 1.3058969974517822

Pattern size : [(17, 99)]
Automata building time : 0.4969966411590576
Pattern matching time : 1.2957923412322998

Pattern size : [(100, 100)]
Automata building time : 0.046999454498291016
Pattern matching time : 1.3579652309417725

Pattern size : [(200, 200)]
Automata building time : 0.19803190231323242
Pattern matching time : 1.3247194290161133

Pattern size : [(400, 400)]
Automata building time : 1.0880215167999268
Pattern matching time : 1.326977252960205

Pattern size : [(800, 800)]
Automata building time : 4.801287889480591
Pattern matching time : 1.1318275928497314

Pattern size : [(10, 9), (17, 99), (100, 100), (200, 200), (400, 400), (800, 800)]
Automata building time : 6.6137495040893555
Pattern matching time : 1.3763060569763184
```

In [101…

```
fig, ax = plt.subplots(1, 1, figsize=(15,8))
fig.suptitle("2 FRAGMENTS")
ax.plot(patterns_id, matching_times, "r--o")
ax.set_ylim([0, 5])
ax.set_title("Matching patterns")
ax.set_xlabel("Pattern id")
ax.set_ylabel("Time [s]")
```

Out[101…

```
Text(0, 0.5, 'Time [s]')
```



Img 4 fragments

In [102…

```
all_patterns = [img_a, img_pattern, size100, size200, size400, size800]
patterns = [[img_a], [img_pattern], [size100], [size200], [size400], [size800], all_patterns]
res = time_test(img_4, patterns)
building_times = res[0]
```

```
matching_times = res[1]
```

```
Pattern size : [(10, 9)]
Automata building time : 0.0017347335815429688
Pattern matching time : 0.5956242084503174

Pattern size : [(17, 99)]
Automata building time : 0.008032798767089844
Pattern matching time : 0.637005090713501

Pattern size : [(100, 100)]
Automata building time : 0.03999781608581543
Pattern matching time : 0.6484918594360352

Pattern size : [(200, 200)]
Automata building time : 0.18562102317810059
Pattern matching time : 0.6090335845947266

Pattern size : [(400, 400)]
Automata building time : 1.4746403694152832
Pattern matching time : 0.6730272769927979

Pattern size : [(800, 800)]
Automata building time : 6.961668014526367
Pattern matching time : 0.6550328731536865

Pattern size : [(10, 9), (17, 99), (100, 100), (200, 200), (400, 400), (800, 800)]
Automata building time : 7.163971185684204
Pattern matching time : 0.6050000190734863

Pattern size : [(10, 9)]
Automata building time : 0.0010018348693847656
Pattern matching time : 0.6634814739227295

Pattern size : [(17, 99)]
Automata building time : 0.5072140693664551
Pattern matching time : 0.6025941371917725

Pattern size : [(100, 100)]
Automata building time : 0.04399442672729492
Pattern matching time : 0.6644248962402344

Pattern size : [(200, 200)]
Automata building time : 0.19734692573547363
Pattern matching time : 0.631885290145874

Pattern size : [(400, 400)]
Automata building time : 1.1679224967956543
Pattern matching time : 0.6780388355255127

Pattern size : [(800, 800)]
Automata building time : 5.822889566421509
Pattern matching time : 0.7770240306854248

Pattern size : [(10, 9), (17, 99), (100, 100), (200, 200), (400, 400), (800, 800)]
Automata building time : 8.126817464828491
Pattern matching time : 0.6988804340362549

Pattern size : [(10, 9)]
Automata building time : 0.0009980201721191406
Pattern matching time : 0.728297233581543

Pattern size : [(17, 99)]
Automata building time : 0.011967897415161133
Pattern matching time : 0.7232234477996826

Pattern size : [(100, 100)]
Automata building time : 0.04603290557861328
Pattern matching time : 0.6639814376831055

Pattern size : [(200, 200)]
Automata building time : 0.2039661407470703
Pattern matching time : 0.6477727890014648

Pattern size : [(400, 400)]
Automata building time : 2.069396734237671
Pattern matching time : 0.9665102958679199

Pattern size : [(800, 800)]
Automata building time : 5.8214569091796875
Pattern matching time : 0.6178364753723145

Pattern size : [(10, 9), (17, 99), (100, 100), (200, 200), (400, 400), (800, 800)]
Automata building time : 7.185697555541992
```

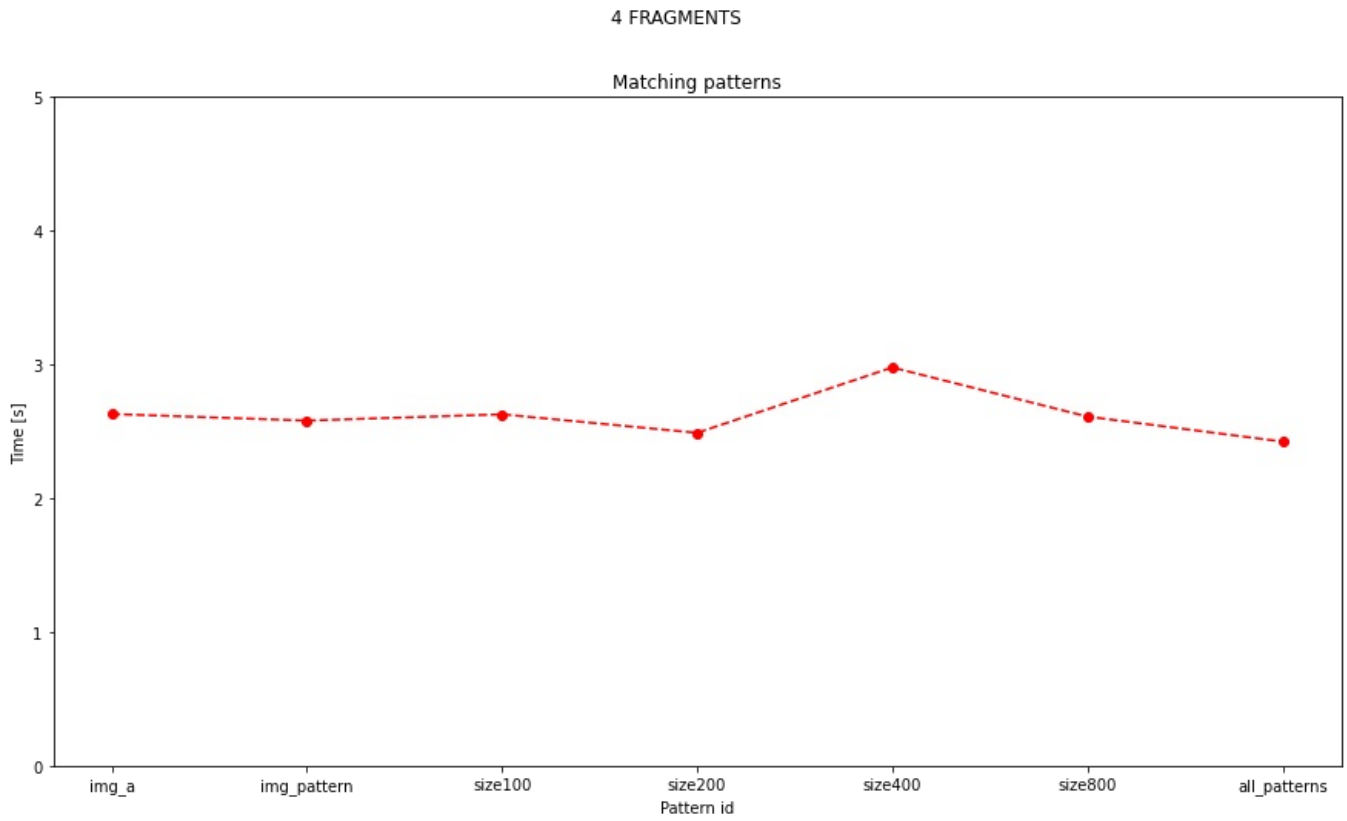
```
Pattern matching time : 0.5832703113555908

Pattern size : [(10, 9)]
Automata building time : 0.0009932518005371094
Pattern matching time : 0.6407020092010498

Pattern size : [(17, 99)]
Automata building time : 0.013998270034790039
Pattern matching time : 0.6161849498748779

Pattern size : [(100, 100)]
Automata building time : 0.0429990291595459
Pattern matching time : 0.6489653587341309

Pattern size : [(200, 200)]
Automata building time : 0.1770024299621582
Pattern matching time : 0.5987985134124756

Pattern size : [(400, 400)]
Automata building time : 1.4789419174194336
Pattern matching time : 0.6585071086883545

Pattern size : [(800, 800)]
Automata building time : 4.685393333435059
Pattern matching time : 0.5591716766357422

Pattern size : [(10, 9), (17, 99), (100, 100), (200, 200), (400, 400), (800, 800)]
Automata building time : 6.125709772109985
Pattern matching time : 0.5349631309509277
```

In [103…

```
fig, ax = plt.subplots(1, 1, figsize=(15,8))
fig.suptitle("4 FRAGMENTS")
ax.plot(patterns_id, matching_times, "r--o")
ax.set_ylim([0, 5])
ax.set_title("Matching patterns")
ax.set_xlabel("Pattern id")
ax.set_ylabel("Time [s]")
```

Out[103…

```
Text(0, 0.5, 'Time [s]')
```



8 fragments

In [104…

```
all_patterns = [img_a, img_pattern, size100, size200, size400, size800]
patterns = [[img_a], [img_pattern], [size100], [size200], [size400], [size800], all_patterns]
```

```
res = time_test(img_8, patterns)
building_times = res[0]
matching_times = res[1]
```

```
Pattern size : [(10, 9)]
Automata building time : 0.001001596450805664
Pattern matching time : 0.2961294651031494

Pattern size : [(17, 99)]
Automata building time : 0.009968996047973633
Pattern matching time : 0.2950422763824463

Pattern size : [(100, 100)]
Automata building time : 0.042033433391418457
Pattern matching time : 0.3100755214691162

Pattern size : [(200, 200)]
Automata building time : 0.1770038604736328
Pattern matching time : 0.30299806594848633

Pattern size : [(400, 400)]
Automata building time : 1.503422737121582
Pattern matching time : 0.31799983978271484

Pattern size : [(800, 800)]
Automata building time : 4.399275064468384
Pattern matching time : 0.2610504627227783

Pattern size : [(10, 9), (17, 99), (100, 100), (200, 200), (400, 400), (800, 800)]
Automata building time : 6.125144958496094
Pattern matching time : 0.26799964904785156

Pattern size : [(10, 9)]
Automata building time : 0.001013040542602539
Pattern matching time : 0.29218268394470215

Pattern size : [(17, 99)]
Automata building time : 0.009032726287841797
Pattern matching time : 0.2850005626678467

Pattern size : [(100, 100)]
Automata building time : 0.04200148582458496
Pattern matching time : 0.3039994239807129

Pattern size : [(200, 200)]
Automata building time : 0.17499780654907227
Pattern matching time : 0.2909998893737793

Pattern size : [(400, 400)]
Automata building time : 1.5030651092529297
Pattern matching time : 0.32200026512145996

Pattern size : [(800, 800)]
Automata building time : 4.437562704086304
Pattern matching time : 0.27700066566467285

Pattern size : [(10, 9), (17, 99), (100, 100), (200, 200), (400, 400), (800, 800)]
Automata building time : 6.0928685665130615
Pattern matching time : 0.27300000190734863

Pattern size : [(10, 9)]
Automata building time : 0.0
Pattern matching time : 0.28400230407714844

Pattern size : [(17, 99)]
Automata building time : 0.008002281188964844
Pattern matching time : 0.27816152572631836

Pattern size : [(100, 100)]
Automata building time : 0.04603147506713867
Pattern matching time : 0.30103564262390137

Pattern size : [(200, 200)]
Automata building time : 0.1790003776550293
Pattern matching time : 0.2960326671600342

Pattern size : [(400, 400)]
Automata building time : 1.4171183109283447
Pattern matching time : 0.30404067039489746

Pattern size : [(800, 800)]
Automata building time : 4.434962272644043
Pattern matching time : 0.2700376510620117
```

```
Pattern size : [(10, 9), (17, 99), (100, 100), (200, 200), (400, 400), (800, 800)]
Automata building time : 6.321556806564331
Pattern matching time : 0.28600168228149414

Pattern size : [(10, 9)]
Automata building time : 0.0
Pattern matching time : 0.281033992767334

Pattern size : [(17, 99)]
Automata building time : 0.49550700187683105
Pattern matching time : 0.3286290168762207

Pattern size : [(100, 100)]
Automata building time : 0.05300283432006836
Pattern matching time : 0.34304261207580566

Pattern size : [(200, 200)]
Automata building time : 0.24295878410339355
Pattern matching time : 0.3231632709503174

Pattern size : [(400, 400)]
Automata building time : 1.2380640506744385
Pattern matching time : 0.32812929153442383

Pattern size : [(800, 800)]
Automata building time : 4.782065391540527
Pattern matching time : 0.2669992446899414

Pattern size : [(10, 9), (17, 99), (100, 100), (200, 200), (400, 400), (800, 800)]
Automata building time : 6.159694671630859
Pattern matching time : 0.27303552627563477

Pattern size : [(10, 9)]
Automata building time : 0.0
Pattern matching time : 0.28603672981262207

Pattern size : [(17, 99)]
Automata building time : 0.006998300552368164
Pattern matching time : 0.28603506088256836

Pattern size : [(100, 100)]
Automata building time : 0.04301166534423828
Pattern matching time : 0.30199766159057617

Pattern size : [(200, 200)]
Automata building time : 0.18600130081176758
Pattern matching time : 0.28896474838256836

Pattern size : [(400, 400)]
Automata building time : 1.3970046043395996
Pattern matching time : 0.30699944496154785

Pattern size : [(800, 800)]
Automata building time : 4.451683521270752
Pattern matching time : 0.26904964447021484

Pattern size : [(10, 9), (17, 99), (100, 100), (200, 200), (400, 400), (800, 800)]
Automata building time : 6.2577526569366455
Pattern matching time : 0.2569615840911865

Pattern size : [(10, 9)]
Automata building time : 0.0
Pattern matching time : 0.29903435707092285

Pattern size : [(17, 99)]
Automata building time : 0.011123895645141602
Pattern matching time : 0.31090784072875977

Pattern size : [(100, 100)]
Automata building time : 0.0429990291595459
Pattern matching time : 0.31096792221069336

Pattern size : [(200, 200)]
Automata building time : 0.17299985885620117
Pattern matching time : 0.2919905185699463

Pattern size : [(400, 400)]
Automata building time : 1.493001937866211
Pattern matching time : 0.3000345230102539

Pattern size : [(800, 800)]
Automata building time : 4.608298301696777
Pattern matching time : 0.26865673065185547
```

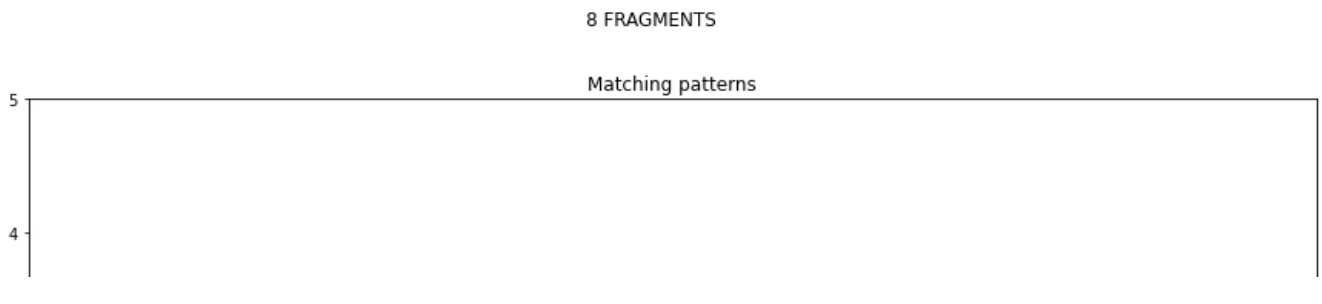
```
Pattern size : [(10, 9), (17, 99), (100, 100), (200, 200), (400, 400), (800, 800)]
Automata building time : 6.227440118789673
Pattern matching time : 0.25858521461486816

Pattern size : [(10, 9)]
Automata building time : 0.0009660720825195312
Pattern matching time : 0.29506993293762207

Pattern size : [(17, 99)]
Automata building time : 0.00999903678894043
Pattern matching time : 0.2971687316894531

Pattern size : [(100, 100)]
Automata building time : 0.04029273986816406
Pattern matching time : 0.30603623390197754

Pattern size : [(200, 200)]
Automata building time : 0.213029146194458
Pattern matching time : 0.2864556312561035

Pattern size : [(400, 400)]
Automata building time : 1.4609975814819336
Pattern matching time : 0.29503393173217773

Pattern size : [(800, 800)]
Automata building time : 4.51861310005188
Pattern matching time : 0.26000142097473145

Pattern size : [(10, 9), (17, 99), (100, 100), (200, 200), (400, 400), (800, 800)]
Automata building time : 6.114511251449585
Pattern matching time : 0.26696181297302246

Pattern size : [(10, 9)]
Automata building time : 0.0010006427764892578
Pattern matching time : 0.28799915313720703

Pattern size : [(17, 99)]
Automata building time : 0.006998777389526367
Pattern matching time : 0.29900288581848145

Pattern size : [(100, 100)]
Automata building time : 0.04519248008728027
Pattern matching time : 0.3120002746582031

Pattern size : [(200, 200)]
Automata building time : 0.17399859428405762
Pattern matching time : 0.3079671859741211

Pattern size : [(400, 400)]
Automata building time : 1.4152414798736572
Pattern matching time : 0.3459649085998535

Pattern size : [(800, 800)]
Automata building time : 4.684085369110107
Pattern matching time : 0.261000394821167

Pattern size : [(10, 9), (17, 99), (100, 100), (200, 200), (400, 400), (800, 800)]
Automata building time : 6.16321873664856
Pattern matching time : 0.28499817848205566
```

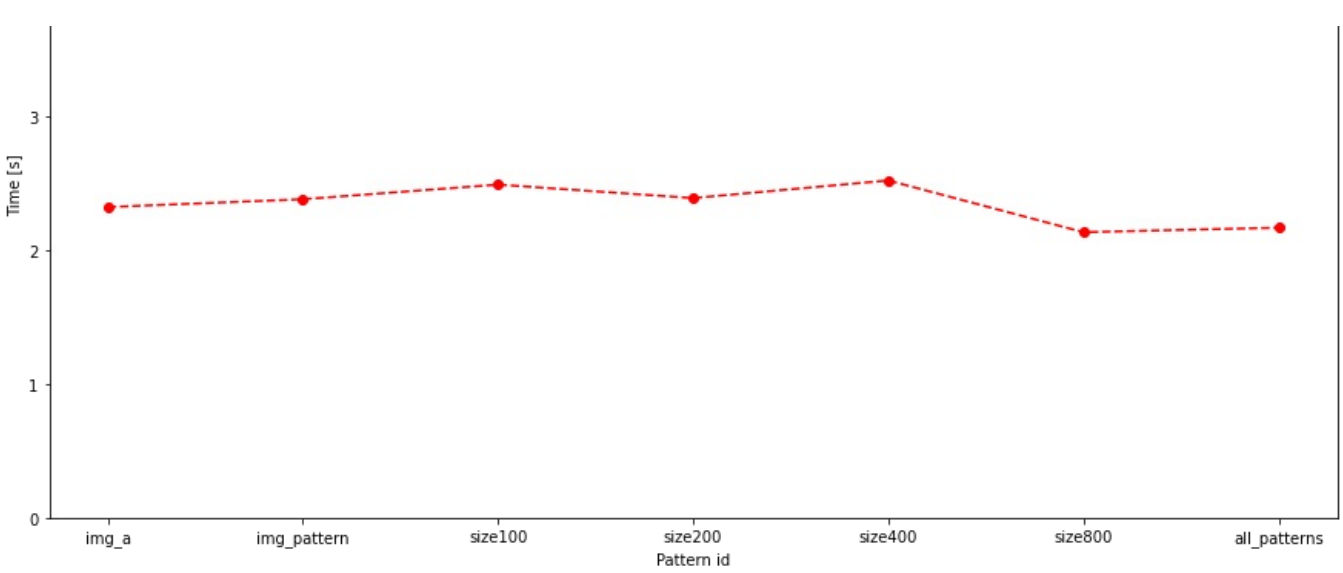
In [105…

```
fig, ax = plt.subplots(1, 1, figsize=(15,8))
fig.suptitle("8 FRAGMENTS")
ax.plot(patterns_id, matching_times, "r--o")
ax.set_ylim([0, 5])
ax.set_title("Matching patterns")
ax.set_xlabel("Pattern id")
ax.set_ylabel("Time [s]")
```

Out[105…

```
Text(0, 0.5, 'Time [s]')
```





In [ ]:

```
Czasy przeszukiwania wyszły sumarycznie takie same, brakuje wyjścia z funkcji jeśli rozmiar wzorca przekracza roz
przeszukiwanego pliku więc dlatego czasy się zgadzają dla 4 i 8 fragmentów mimo to że wzorzec "size800" przekracz
Ale też istnieje duża szansa że wzorzec znajdować się będzie we fragmencie pliku gdzie zachodzi rozfragmentowanie
lepiej nie dzielić na fragmenty albo na przynajmniej takie aby wzorzec mógł się w nich znaleść 2mx2n.
```