Lab1_Piotr_Kica

March 12, 2021

0.0.1 Naive pattern matching algorithm

[14]:
```
def naive(text, pattern, _ = None):
    matches = []
    for i in range(len(text)-len(pattern)+1):
        if text[i:i+len(pattern)] == pattern:
            matches.append(i)
    return matches
```

[15]:
```
naive("abccd","cc")
```

[15]:
```
[2]
```

0.0.2 Finite automata

[4]:
```
import re

def transition_table(pattern):
    t_table = []
    alphabet = set(pattern)

    for q in range(0, len(pattern) + 1):

        t_table.append({})
        for symbol in alphabet:

            k = min(len(pattern) + 1, q + 1)
            suffix = pattern[:q] + symbol

            while k >= 0 and pattern[:k] != suffix[q - k + 1:]:
                k = k - 1
                #print(k)
            t_table[q][symbol] = k
    return t_table

def finite_automata(text, pattern, preprocessing = None):
    matches = []
    q = 0
```

```
    if preprocessing is None:
        t_table = transition_table(pattern)
    else:
        t_table = preprocessing

    for s in range(len(text)):
        if text[s] in t_table[q]:
            q = t_table[q][text[s]]
            if q == len(t_table) - 1:
                matches.append(s+1-q)
                # s + 1 - ponieważ przeczytaliśmy znak o indeksie s, więc␣
↪przesunięcie jest po tym znaku
        else:
            q=0

    return matches
```

[5]:
```
transition_table("ababac")
```

[5]:
```
[{'a': 1, 'c': 0, 'b': 0},
 {'a': 1, 'c': 0, 'b': 2},
 {'a': 3, 'c': 0, 'b': 0},
 {'a': 1, 'c': 0, 'b': 4},
 {'a': 5, 'c': 0, 'b': 0},
 {'a': 1, 'c': 6, 'b': 4},
 {'a': 1, 'c': 0, 'b': 0}]
```

[6]:
```
len(finite_automata("ababbababbvcdaba"*10, "aba"))
```

[6]:
```
30
```

0.0.3 KMP algorithm

[8]:
```
def prefix_function(pattern):
    pi = [0]
    k = 0
    for q in range(1, len(pattern)):
        while(k > 0 and pattern[k] != pattern[q]):
            k = pi[k-1]
        if(pattern[k] == pattern[q]):
            k = k + 1
        pi.append(k)
    return pi

def kmp_string_matching(text, pattern, preprocessing = None):
    if preprocessing is None:
```

```
        pi = prefix_function(pattern)
    else:
        pi = preprocessing
    q = 0
    matches = []
    for i in range(0, len(text)):
        while(q > 0 and pattern[q] != text[i]):
            q = pi[q-1]
        if(pattern[q] == text[i]):
            q = q + 1
        if(q == len(pattern)):
            matches.append(i+1-q)
            q = pi[q-1]

    return matches
```

[9]:
```
prefix_function("ababaca")
```

[9]:
```
[0, 0, 1, 2, 3, 0, 1]
```

[10]:
```
print(kmp_string_matching("abaabaaaaba", "aba"))
```

```
[0, 3, 8]
```

Timer function

[11]:
```
from time import time
def timer(algorithm, text, pattern, preprocessing = None):
    t1 = time()
    algorithm(text, pattern, preprocessing)
    t2 = time()
    print("Czas wykonania algorytmu:" + str(t2-t1) + "s")
```

0.0.4 Time comparition

[19]:
```
# RUN CELL WITH ARTICLE BEFORE
timer(naive, article, "art")
```

```
Czas wykonania algorytmu:0.05187034606933594s
```

[20]:
```
timer(finite_automata, article, "art")
```

```
Czas wykonania algorytmu:0.03453850746154785s
```

[22]:
```
timer(kmp_string_matching, article, "art")
```

```
Czas wykonania algorytmu:0.04773211479187012s
```

[23]:
```
print(len(naive(article, "art")))
print(len(finite_automata(article, "art")))
print(len(kmp_string_matching(article, "art")))
print(naive(article, "art") == finite_automata(article, "art") ==␣
 ↪kmp_string_matching(article, "art"))
print(naive(article, "art"))
```

```
273
273
273
True
[1153, 1502, 4689, 4731, 4876, 5079, 5145, 5946, 6036, 7263, 7508, 7778, 8041,
8296, 9101, 9956, 10019, 10221, 11119, 11204, 11615, 13191, 15281, 15355, 16089,
16258, 16403, 16544, 16613, 16837, 16853, 23634, 24058, 24149, 24583, 24680,
24777, 24928, 25527, 25686, 26998, 27285, 27476, 27539, 27589, 27854, 28370,
28555, 28763, 30961, 31018, 31093, 31359, 31808, 32606, 32965, 33050, 33265,
33592, 34648, 34734, 35508, 36152, 37140, 37540, 38448, 38592, 39053, 39207,
39433, 39565, 39977, 41149, 41826, 42025, 42195, 42368, 42501, 42715, 42893,
42938, 43444, 43552, 43784, 44587, 44650, 44950, 45007, 45290, 45398, 47316,
47419, 48782, 48817, 48903, 49049, 49256, 49313, 49485, 49556, 49912, 49976,
50099, 50157, 50699, 51047, 51176, 51963, 52068, 52269, 52549, 53005, 53029,
53208, 53785, 53928, 54075, 54134, 54767, 55072, 55276, 55462, 55804, 55988,
56824, 56908, 57161, 57546, 57797, 57929, 57986, 58277, 58375, 58871, 58963,
59392, 59520, 59946, 60293, 60546, 60791, 61259, 61767, 62460, 62607, 62660,
63501, 63695, 63786, 63866, 65072, 65111, 65173, 66021, 66327, 66979, 67053,
67575, 67692, 67717, 67849, 67923, 68223, 68851, 69043, 69169, 69246, 69601,
69694, 69791, 70106, 70509, 70661, 70695, 71558, 71702, 72103, 72311, 73112,
74250, 75343, 75467, 75483, 75490, 75714, 75800, 75868, 76244, 77027, 78044,
78555, 78869, 78933, 78967, 79088, 79267, 79338, 79407, 79445, 79579, 79819,
79881, 80845, 83231, 84575, 84825, 85831, 86790, 86816, 87090, 87171, 87356,
87396, 87637, 87704, 87944, 88030, 88070, 88133, 88388, 88662, 88937, 89044,
89313, 89322, 89338, 89358, 90164, 90215, 90569, 91679, 91792, 92913, 93363,
93423, 93558, 94600, 94805, 95978, 96117, 97282, 98767, 99822, 102950, 104138,
104720, 105764, 105970, 110183, 115007, 115160, 116106, 144049, 158604, 159475,
161578, 162785, 163966, 168894, 169071, 178450, 185628, 200527, 200625, 202693,
206791, 209178, 211816, 212316, 217441, 217888, 223165, 223249]
```

0.0.5 Cases where performance differ a lot

>5x improvement over naive

[18]:
```
pattern = 'X' * 70000
text = (pattern +'D') * 25
print("Naiwny algorytm:")
timer(naive,text, pattern)
print()

print("Automat skończony:")
```

```
timer(finite_automata,text, pattern, transition_table(pattern))
print()

print("Algorytm KMP:")
timer(kmp_string_matching, text, pattern, prefix_function(pattern))
```

```
Naiwny algorytm:
Czas wykonania algorytmu:6.2361907958984375s

Automat skończony:
Czas wykonania algorytmu:0.3621647357940674s

Algorytm KMP:
Czas wykonania algorytmu:0.5906519889831543s
```

Transition table creation time is >5x longer than prefix function in KMP

[24]:
```
pattern2 = "X"*3000+"D"
t1 = time()
transition_table(pattern2)
t2 = time()
print("Czas wykonania algorytmu:" + str(t2-t1) + "s")

t1 = time()
prefix_function(pattern2)
t2 = time()
print("Czas wykonania algorytmu:" + str(t2-t1) + "s")
```

```
Czas wykonania algorytmu:2.5707316398620605s
Czas wykonania algorytmu:0.0030448436737060547s
```

0.0.6 Test text

[18]:
```
# Tekst usunięty dla poprawnego skonwertowania do pdf
```

[ ]:
```

```