Raport - Algorytmy Tekstowe - Laboratorium 6

Łukasz Sochacki

Podpunkt 1.

W celu realizacji podpunktu 1 na początku została zaimplementowana klasa Node w celu łatwiejszego wykonywania operacji będących częścią algorytmu :

```
class Node:
    def __init__(self):
        self.state = 0
        self.connected = None
        self.connections = {}
```

Następnie została zaimplementowana klasa Automata, która realizuje algorytm wyszukiwania wzorca 2D w tekście. Klasa ta składa się z następujących elementów :

Inicjalizator :

```
class Automata:
    def __init__(self, pattern):
        self.states = []
        self.automata = {}
        self.parent = Node()
        self.current = None
        self.pattern = pattern
```

Funkcja initializer, która na bazie otrzymanego wzorca do znalezienia w tekście odpowiednio przekształca zmienne w klasie :

```
def initializer(self):
    self.current = self.parent
    counter = 1
    n = len(self.pattern)
    m = len(self.pattern[0])
    unique = set()
    queue = Queue()
    i = 0
    while i < m:
        tmp = self.parent
        j = 0
        while j < n:
            sign = self.pattern[j][i]
            if sign in tmp.connections:
                pass
            else:
                unique.add(sign)
                tmp.connections[sign] = Node()
```

```
                tmp.connections[sign].state = counter
                counter += 1
            tmp = tmp.connections[sign]
            j += 1
        i += 1

    for el in unique:
        passes = self.parent.connections
        if el not in passes:
            passes[el] = self.parent
        else:
            passes[el].connected = self.parent
            queue.put(passes[el])

    while not queue.empty():
        taken = queue.get()
        passes = taken.connections
        for el in unique:
            if el not in passes:
                pass
            else:
                next = passes[el]
                queue.put(next)
                parent = taken.connected
                while el not in parent.connections:
                    parent = parent.connected
                next.connected = parent.connections[el]

    i = 0
    while i < m:
        self.states.append(0)
        j = 0
        while j < n:
            self.states[len(self.states) - 1] =
self.getChar(self.pattern[j][i])
            j += 1
        self.current = self.parent
        i += 1

    for el in self.states:
        if el in self.automata.keys():
            pass
        else:
            self.automata[el] = [0] * (len(self.states) + 1)

    counter = 0
    self.automata[self.states[0]][0] = 1
    for i in range(len(self.automata)):
        for el in self.automata.values():
            el[i] = el[counter]
        if i >= len(self.states):
            pass
        else:
            self.automata[self.states[i]][i] = i + 1
            counter = self.automata[self.states[i]][counter]
```

Funkcja getChar znajdująca informacje o danym znaku:

```
def getChar(self, sign):
    if self.current is None:
```

```
            return None

        while sign not in self.current.connections.keys():
            self.current = self.current.connected
            if self.current is not None:
                pass
            else:
                self.current = self.parent
                return self.current.state

        self.current = self.current.connections[sign]
        return self.current.state
```

Funkcja parser przekształcająca otrzymany tekst :

```
def parser(self, line):
    res = []
    n = len(line)
    which = 0
    i = 0
    while i < n:
        if line[i] in self.automata:
            which = self.automata[line[i]][which]
            if which == len(self.states):
                res.append(i)
        else:
            which = 0
            i += 1
            continue
        i += 1
    return res
```

Funkcja find, która znajduje wszystkie wystąpienia wzorca w otrzymanym tekście :

```
def find(self, word):
    res = []
    output = []
    length = 0
    found = []

    for el in word:
        if len(el) > length:
            length = len(el)
        found.append([])

    i = 0
    while i < length:
        j = 0
        while j < len(word):
            if i < len(word[j]):
                found[j].append(self.getChar(word[j][i]))
            j += 1
        self.current = self.parent
        i += 1

    i = 0
```

```
    while i < len(found):
        parsed = self.parser(found[i])
        if len(parsed) > 0:
            res.append((i, parsed))
        i += 1

    for el in res:
        for sign in el[1]:
            output.append((el[0] - len(self.pattern) + 1, sign -
len(self.pattern[0]) + 1))

    return output
```

Podpunkt 2.

W celu realizacji podpunktu została utworzona poniższa funkcja, która otrzymując dany tekst oraz wzorzec wykorzystuje algorytm z podpunktu 1 w celu znalezienia wszystkich wystąpień wzorca w danym tekście.

```
def checkOccurances(text, pattern):
    res = Automata(pattern)
    res.initializer()
    print(f'Coordinates of occurances : {res.find(text)}')
    print(f'All occurances : {len(res.find(text))}')
```

W celu realizacji podpunktu jako wzorzec został podany wzorzec 2D składający się z dwóch takich samych liter w pionie. Wynik jest wyświetlany jako lista par. Para zawiera informację o tym w jakim miejscu dwa znaki są takie same w postaci wartości liczbowej mówiącej gdzie znak jest położony w linii tekstu.

```
with open('haystack.txt', 'r') as f:
    lines = f.readlines()

unique = []
for el in lines:
    for sign in el:
        if sign not in unique:
            unique.append(sign)

res = []
for sign in unique:
    pattern = [[sign], [sign]]
    automat = Automata(pattern)
    automat.initializer()
    res.extend(automat.find(lines))

print(res)
print(len(res))
```

Podpunkt został zrealizowany z następującymi rezultatami.

Łączna liczba wystąpień sytuacji opisanej w podpunkcie : 435

Fragment przykładowego wyniku : [(0, 83), (1, 9), (14, 54), (19, 37), (20, 56), (21, 62), (31, 1), (35, 18), (39, 53), (51, 32) ,...]

Podpunkt 3.

W celu realizacji podpunktu została wykorzystana funkcja oraz realizacja wzorca 2D wspomniana w podpunkcie numer 2.

Poniższy fragment kodu zwraca odpowiedź do podpunktu numer 3.

```
checkOccurances(lines, [['t', 'h'], ['t', 'h']])
checkOccurances(lines, [['t', ' ', 'h'], ['t', ' ', 'h']])
```

Podpunkt został zrealizowany z następującymi rezultatami.

Wzorzec ‘th’

Łączna liczba wystąpień sytuacji opisanej w podpunkcie : 0

Wzorzec ‘t h’

Łączna liczba wystąpień sytuacji opisanej w podpunkcie : 1

Otrzymany wynik : [(37, 0)]

Podpunkt 4

W celu realizacji podpunktu została zaimplementowana funkcja, która otrzymując obraz przetwarza go w celu możliwości wyszukiwania wzorców w tekście.

```
def imageProcess(image):
    laodedimage = image.load()
    res = []
    n = image.height
    m = image.width
    i = 0
    while i < n:
        pixels = []
        j = 0
        while j < m:
            pixels.append(laodedimage[j, i][0])
            j += 1
        res.append(pixels)
        i += 1
    return res
```

Dodatkowo została zaimplementowana funkcja z podpunktu 2. działająca na przetworzonych obrazach zamiast na czystych tekstach.

```
def checkOccurancesImage(text, patternname):
    with Image.open(patternname, 'r') as f:
```

```
        pattern = imageProcess(f)

    res = Automata(pattern)
    res.initializer()
    print(f'Coordinates of occurances : {res.find(text)}')
    print(f'All occurances : {len(res.find(text))}')
```

Podczas realizacji podpunktu zostały wybrane litery ‘m’,’s’ oraz ‘i’. Po odpaleniu poniższego kodu zostały osiągnięte następujące rezultaty.

```
with Image.open('haystack.png', 'r') as f:
    lines = imageProcess(f)

checkOccurancesImage(lines, 'm.png')
checkOccurancesImage(lines, 's.png')
checkOccurancesImage(lines, 'i.png')
```

Osiągnięte rezultaty dla litery ‘m’ :

- **Łączna ilość wystąpień** : 131
- **Przykładowe znalezione pozycje** : [(37, 140), (37, 469), (37, 722), …]

Osiągnięte rezultaty dla litery ‘s’ :

- **Łączna ilość wystąpień** : 334
- **Przykładowe znalezione pozycje** : [(37, 126), (37, 185), (37, 391), …]

Osiągnięte rezultaty dla litery ‘i’ :

- **Łączna ilość wystąpień** : 343
- **Przykładowe znalezione pozycje** : [(33, 135), (33, 429), (33, 505), …]

Podpunkt 5

Podpunkt został zrealizowany analogicznie do podpunktu poprzedniego. Poniższy kod dał następujące rezultaty :

```
checkOccurancesImage(lines, 'pattern.png')
```

Osiągnięte rezultaty dla ‘p a t t e r n’ :

- **Łączna ilość wystąpień** : 6

- **Przykładowe znalezione pozycje** : [(147, 365), (147, 674), (169, 107), (169, 390), (169, 500), (191, 66)]

Podpunkt 6

Na początku została utworzona funkcja, która otrzymując dany tekst oraz wzorzec tworzy automat na ich podstawie oraz znajduje wystąpienia wzorca w podanym tekście. Funkcja osobno liczy czas tworzenia się automatu oraz jego inicjalizacji oraz i osobno czas znajdywania wystąpień wzorca. Funkcja na końcu wraca wyliczone czasy.

```
def getTime(word, pattern):
    start = time.time()
    res = Automata(pattern)
    res.initializer()
    end = time.time()
    builder = end - start
    start = time.time()
    res.find(word)
    end = time.time()
    finder = end - start
    return builder, finder
```

Dodatkowo została wykorzystana poniższa w celu wyświetlenia otrzymanych czasów dla różnych wzorców.

```
def printTime(word, pattern):
    builded, found = getTime(word, pattern)
    print(f'Building time : {builded} s')
    print(f'Finding time : {found} s')
```

Poniżej znajdują się rezultaty wraz z kodem, który został uruchomiony w celu uzyskania ich.

```
with Image.open('first.png','r') as f:
    first = imageProcess(f)

with Image.open('second.png', 'r') as f:
    second = imageProcess(f)

with Image.open('third.png', 'r') as f:
    third = imageProcess(f)

printTime(lines, first)
printTime(lines, second)
printTime(lines, third)
```

Osiągnięte rezultaty dla pierwszego wzorca o długości około 10 :

- **Czas budowania automatu** : 0.004999876022338867 s
- **Czas znajdywania wzorca** : 2.0249693393707275 s

Osiągnięte rezultaty dla pierwszego wzorca o długości około 35 :

- **Czas budowania automatu** : 0.034035682678222656 s
- **Czas znajdywania wzorca** : 1.9769632816314697 s

Osiągnięte rezultaty dla pierwszego wzorca o długości około 90 :

- **Czas budowania automatu** : 0.17403602600097656 s
- **Czas znajdywania wzorca** : 1.8840034008026123 s

Podpunkt 7

Na początku została utworzona funkcja otrzymująca tekst, który jest następnie dzielony na tyle części ile zostało podane.

```
def divider(text, amount):
    res = []
    for i in range(amount):
        res.append(text[i::amount])
    return res
```

Następnie została utworzona funkcja, która wyświetla łączny czas znajdywania wzorca w podzielonym tekście.

```
def timeWithDivide(text,amount):
    timer = 0
    for el in text:
        builded, founded = getTime(el, second)
        timer += founded
    print(f'Founding time : {timer} s')
```

W celu realizacji podpunktu został wykorzystany wzorzec o średniej długości z podpunktu 6. Poniżej znajdują się osiągnięte rezultaty :

```
divided = divider(lines, 2)
timeWithDivide(divided,second)
divided = divider(lines, 4)
timeWithDivide(divided,second)
divided = divider(lines, 8)
timeWithDivide(divided,second)
print('--------------------------------')
```

Osiągnięte rezultaty dla pliku podzielonego na 2 części :

- **Czas znajdywania wzorca** : 1.9240176677703857 s

Osiągnięte rezultaty dla pliku podzielonego na 4 części :

- **Czas znajdywania wzorca** : 1.8629755973815918 s

Osiągnięte rezultaty dla pliku podzielonego na 8 części :

- **Czas znajdywania wzorca** : 1.7660083770751953 s