Algorytmy tekstowe – Drzewa sufiksów

Tomasz Gargula

1 Implementacja algorytmów

W celu wizualizacji drzew użyłem pakietu pydot, stąd w kodzie pojawiają się odniesienia do klas z tego pakietu.

1.1 Drzewo trie

Zdecydowałem się na implementację struktur, korzystając głównie z paradygmatu obiektowego. Poniżej kod odpowiedzialny za stworzenie węzła w drzewie Trie.

```
class TrieNode:
    def __init__(self, letter=None, parent=None):
        self.letter = letter
        self.parent = parent
        self.children = {}

    def _save(self, graph, parent=None, prefix=""):
        word = prefix + self.letter
        node = pydot.Node(word, label=self.letter, shape="point")
        graph.add_node(node)

        if parent is not None:
            graph.add_edge(pydot.Edge(parent, node, label=self.letter))

        for child in self.children.values():
            child._save(graph, node, word)
```

Oraz kod odpowiedzialny za stworzenie struktury:

```
class Trie:
    def __init__(self, text):
        self.root = TrieNode()
        self.text = text
        self._add(text)

    def _add(self, text):
        n = len(text)
        for i in range(n):
            pointer = self.root
            for j in range(i, n):
                letter = text[j]
                if letter not in pointer.children:
                    pointer.children[letter] = TrieNode(letter, pointer)
                pointer = pointer.children[letter]

    def save(self, filename):
        graph = pydot.Dot(graph_type="digraph", label=f"Trie: {self.text}", labelloc='top')

        root = pydot.Node("root", shape="point")
        graph.add_node(root)
```

```
        for child in self.root.children.values():
            child._save(graph, root)

        graph.write_png(filename)
```

1.2 Drzewo sufiksów

Analogicznie do drzew trie rozdzieliłem klasy opisujące całą strukturę drzewa sufiksów oraz jej węzły. W implementacji struktury korzystam z funkcji wyznaczającej długość wspólnego prefiksa dwóch podanych słów równej długości. Kod zamieszczam poniżej.

```
def common_suffix_size(A, B):
    ctr = 0
    for i, letter in enumerate(A):
        if letter == B[i]:
            ctr += 1
        else:
            return ctr
    return ctr
```

Poniżej kod odpowiedzielany za stworzenie węzła w drzewie sufiksów.

```
class SuffixTreeNode:
    def __init__(self, start=None, end=None, parent=None):
        self.start = start
        self.end = end
        self.parent = parent
        self.children = []

    def length(self):
        return self.end - self.start + 1

    def slice(self):
        return (self.start, self.end)

    def subtext(self, text):
        return text[self.start:self.end+1]

    def _save(self, G, parent, text):
        node = pydot.Node(str(self), shape="point")
        G.add_node(node)
        label = str(self.subtext(text) if self.children or
                    self.start == self.end else self.slice())
        edge = pydot.Edge(parent, node, label=label)
        G.add_edge(edge)

        for child in self.children:
            child._save(G, node, text)

    def _add(self, text, i):
        for child in self.children:
            if text[child.start] == text[i]:
                A = text[i:i + child.length()]
                B = text[child.start:child.start + len(A)]
                if A == B:
                    child._add(text, i + child.length())
                    return
                size = common_suffix_size(A, B)
```

```
                subnode = SuffixTreeNode(child.start + size, child.end, parent=child)
                child.end = child.start + size - 1
                subnode.children = child.children
                leaf = SuffixTreeNode(i + size, len(text) - 1, parent=child)
                child.children = [subnode, leaf]
                return
        leaf = SuffixTreeNode(start=i, end=len(text) - 1, parent=self)
        self.children.append(leaf)
```

Oraz kod odpowiedzialny za stworzenie struktury:

```
class SuffixTree:
    def __init__(self, text):
        self.root = SuffixTreeNode()
        self.text = text
        for i in range(len(text)):
            self.root._add(text, i)

    def save(self, filename):
        G = pydot.Dot(graph_type="graph", label=f"SuffixTree: {self.text}", labelloc='top')

        root = pydot.Node(str(None), shape="point")
        G.add_node(root)

        for child in self.root.children:
            child._save(G, root, self.text)

        G.write_png(filename)
```

2 Sterowniki

2.1 Benchmarki

Poniżej znajduje się kod, który jest odpowiedzialny za przesetowanie algorytmów pod względem czasu i porównanie ich ze sobą dla zaproponowanych danych do ćwiczenia.

```
from pathlib import Path
from time import time

from trees import Trie, SuffixTree


def benchmark(Class, *args):
    start = time()
    Class(*args)
    return time() - start


if __name__ == '__main__':
    with open(Path('data', 'act.txt'), 'r') as f:
        x = f.read()
    x = ''.join(x) + '$'

    data = ('bbbd', 'aabbabd', 'ababcd', 'abcbccd')

    for text in data:
        print(f'Trie ({text}):\t\t{benchmark(Trie, text)}')
        print(f'SuffixTree ({text}):\t{benchmark(SuffixTree, text)}')
```

```
    print(f'Trie (act.txt):\t\t{benchmark(Trie, x)}')
    print(f'SuffixTree (act.txt):\t{benchmark(SuffixTree, x)}')
```

Poniżej zamieszczam wyniki:

```
Trie (bbbd):            1.9073486328125e-05
SuffixTree (bbbd):      0.0003688335418701172
Trie (aabbabd):         1.9550323486328125e-05
SuffixTree (aabbabd):   7.319450378417969e-05
Trie (ababcd):          6.723403930664062e-05
SuffixTree (ababcd):    1.6450881958007812e-05
Trie (abcbccd):         1.7404556274414062e-05
SuffixTree (abcbccd):   2.09808349609375e-05
Trie (act.txt):         8.21181058883667
SuffixTree (act.txt):   0.01808333396911621
```

Można zauważyć, że mimo faktu, że oba algorytmy mają złożoność $O(n^2)$, to stworzenie drzewa sufiksów trwa krócej.

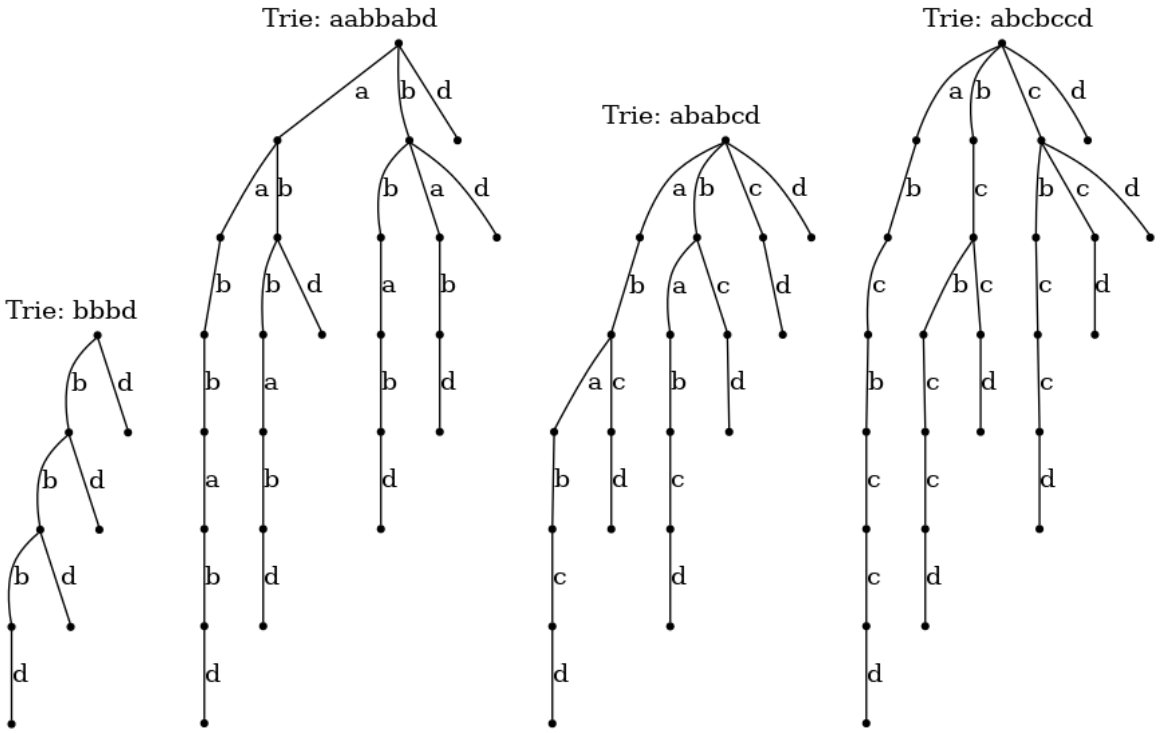
2.2 Wizualizacja

Napisałem również kod, który tworzy pliki png z zaimplementowanych struktur. Poniżej znajduje się kod.

```
from pathlib import Path

from trees import Trie, SuffixTree

if __name__ == '__main__':

    dir = {
        Trie: 'trie',
        SuffixTree: 'suffix_tree'
    }

    data = ('bbbd', 'aabbabd', 'ababcd', 'abcbccd')

    for text in data:
        filename = f'{text}.png'
        T = Trie(text)
        T.save(Path('out', dir[Trie], filename))
        ST = SuffixTree(text)
        ST.save(Path('out', dir[SuffixTree], filename))
```

3 Wizualizacja

3.1 Drzewa trie



3.2 Drzewa sufiksów