Lab2_Piotr_Kica

March 26, 2021

Function to check if last char is unique

[1]:
```
# I assume that # cannot occur in any text except once on the very end.
# Test strings are in last cell

def check_last_char(text):
    last_char = text[-1]
    for char in text[0:-1]:
        if char == last_char:
            return text + "#"
    return text
```

[4]:
```
print(check_last_char(s4))
print(check_last_char("abcdccda"))
```

```
abcbccd
abcdccda#
```

0.0.1 Trie

[5]:
```
class Node:
    def __init__(self, char = None):
        self.char = char
        self.children = dict()

class Trie:
    def __init__(self, text):
        self.root = Node()
        for char_idx in range(len(text)):
            current = self.root
            prefix = text[char_idx:]
            for prefix_char in prefix:
                if prefix_char not in current.children:
                    current.children[prefix_char] = Node(prefix_char)
                current = current.children[prefix_char]

    def find(self, text):
        current = self.root
        for char in text:
```

```
            if char not in current.children:
                return False
            current = current.children[char]
        return True
```

[6]:
```
article_trie = Trie(article)
article_trie.find("USTAWA")
```

[6]:
```
True
```

[7]:
```
class SuffixNode:
    def __init__(self, tree, start, end):
        self.tree = tree
        self.start = start
        self.end = end
        self.parent = None
        self.depth = 0
        self.children = dict()

    def get_char(self, index):
        return self.tree.text[self.start + index]

    def get_child(self, char):
        if char in self.children:
            return self.children[char]
        else:
            return None

    def add_node_between(self, depth):
        new_node = SuffixNode(self.tree, self.start, self.start + depth - 1)
        self.start += depth
        self.parent.children[self.tree.text[new_node.start]] = new_node
        new_node.parent = self.parent
        new_node.children[self.get_char(0)] = self
        self.parent = new_node
        new_node.depth = new_node.parent.depth + depth
        return new_node

    def graft(self, start):
        new_node = SuffixNode(self.tree, start, self.tree.n-1)
        self.children[self.tree.text[start]] = new_node
        new_node.parent = self
        return new_node


class SuffixTree:
    def __init__(self, text):
```

```
        self.text = text
        self.n = len(text)
        self.root = SuffixNode(self, 0, -1)
        child = SuffixNode(self, 0, self.n-1)
        child.parent = self.root
        child.depth = self.n
        self.root.children[text[0]] = child

        for i in range(1, self.n - 1):
            node = self.root
            depth = 0

            while node.get_child(self.text[depth + i]):
                node = node.get_child(self.text[depth + i])
                depth += 1
                node_depth = 1

                while node.start + node_depth <= node.end and node.
↪get_char(node_depth) == self.text[depth + i]:
                    depth += 1
                    node_depth += 1

                if node.start + node_depth <= node.end and node.
↪get_char(node_depth) != self.text[depth + i]:
                    node = node.add_node_between(node_depth)
                    break

            node.graft(depth + i)

    def find(self, text):
        current = self.root
        depth = 0

        for char in text:
            if current.start + depth <= current.end:
                if current.get_char(depth) != char:
                    return False
                depth += 1
            else:
                current = current.get_child(char)
                if current is None:
                    return False
                depth = 1

        return True
```

[8]:
```
article_SuffixTree = SuffixTree(article)
article_SuffixTree.find("USTAWA")
```

[8]:
```
True
```

Validation tests

[9]:
```
# Naive pattern matching algorithm
def naive(text, pattern):
    matches = []
    for i in range(len(text)-len(pattern)+1):
        if text[i:i+len(pattern)] == pattern:
            matches.append(i)
    return matches
```

[10]:
```
# All should be true

# String is found in text:
print((len(naive(s1, "bb")) > 0) == Trie(s1).find("bb"))
print((len(naive(s2, "babd")) > 0) == Trie(s2).find("babd"))
print((len(naive(s3, "ababcd")) > 0) == Trie(s3).find("ababcd"))
print((len(naive(article, "Art")) > 0) == Trie(article).find("Art"))
# String is not found in text:
print((len(naive(s3, "ababcd")) > 0) == Trie(s3).find("ababcd"))
print((len(naive(s4, "ab")) > 0) == Trie(s4).find("ab"))
print((len(naive(s5, "cc")) > 0) == Trie(s5).find("cc"))
print((len(naive(article, "segfault")) > 0) == Trie(article).find("segfault"))

# Trie is correct
```

```
True
True
True
True
True
True
True
True
```

[11]:
```
# All should be true

# String is found in text:
print((len(naive(s1, "bb")) > 0) == SuffixTree(s1).find("bb"))
print((len(naive(s2, "babd")) > 0) == SuffixTree(s2).find("babd"))
print((len(naive(s3, "ababcd")) > 0) == SuffixTree(s3).find("ababcd"))
print((len(naive(article, "Art")) > 0) == SuffixTree(article).find("Art"))
# String is not found in text:
print((len(naive(s3, "eeeeeee")) > 0) == SuffixTree(s3).find("eeeeeee"))
```

```
print((len(naive(s4, "ee")) > 0) == SuffixTree(s4).find("ee"))
print((len(naive(s5, "dźwig")) > 0) == SuffixTree(s5).find("dźwig"))
print((len(naive(article, "segfault")) > 0) == SuffixTree(article).
 ↪find("segfault"))

# SuffixTree is correct
```

```
True
True
True
True
True
True
True
True
```

0.0.2 Performance tests

[12]:
```
from time import time

def timer(f, text):
    start = time()
    if f in ("trie", "Trie"):
        Trie(text)
    else:
        SuffixTree(text)
    end = time()
    print(f"Czas wykonania {f} :", end-start, "s")
```

[13]:
```
timer("Trie", s1)
timer("Trie", "Jeden może to wszyscy mogą"*30+"#")
timer("Trie", "Jak to jest byc skryba, dobrze?" * 100 +"#")
timer("Trie", miniarticle)
timer("Trie", article)

timer("SuffixTree", s1)
timer("SuffixTree", "Jeden moze to wszyscy moga"*30+"#")
timer("SuffixTree", "Jak to jest byc skryba, dobrze?" * 100 +"#")
timer("SuffixTree", miniarticle)
timer("SuffixTree", article)
```

```
Czas wykonania Trie : 0.0 s
Czas wykonania Trie : 0.06525015830993652 s
Czas wykonania Trie : 0.586336612701416 s
Czas wykonania Trie : 1.4023330211639404 s
Czas wykonania Trie : 7.581031560897827 s
Czas wykonania SuffixTree : 0.0 s
Czas wykonania SuffixTree : 0.10004043579101562 s
```

```
Czas wykonania SuffixTree : 1.9399893283843994 s
Czas wykonania SuffixTree : 0.002053499221801758 s
Czas wykonania SuffixTree : 0.02546977996826172 s
```

[3]:
```
s1 = "bbbd"
s2 = "aabbabd"
s3 = "ababcd"
s4 = "abcbccd"
s5 = "abcbccda#"

miniarticle = """Przychodów (dochodów) opodatkowanych w formach zryczałtowanych␣
 ↪nie łączy się z
przychodami (dochodami) z innych źródeł podlegającymi opodatkowaniu na
podstawie ustawy z dnia 26 lipca 1991 r. o podatku dochodowym od osób
fizycznych (Dz. U. z 1993 r. Nr 90, poz. 416 i Nr 134, poz. 646, z 1994 r. Nr
43, poz. 163, Nr 90, poz. 419, Nr 113, poz. 547, Nr 123, poz. 602 i Nr 126,
poz. 626, z 1995 r. Nr 5, poz. 25 i Nr 133, poz. 654, z 1996 r. Nr 25, poz.
113, Nr 87, poz. 395, Nr 137, poz. 638, Nr 147, poz. 686 i Nr 156, poz. 776, z
1997 r. Nr 28, poz. 153, Nr 30, poz. 164, Nr 71, poz. 449, Nr 85, poz. 538, Nr
96, poz. 592, Nr 121, poz. 770, Nr 123, poz. 776, Nr 137, poz. 926, Nr 139,
poz. 932-934 i Nr 141, poz. 943 i 945 oraz z 1998 r. Nr 66, poz. 430, Nr 74,
poz. 471, Nr 108, poz. 685 i Nr 117, poz. 756), zwanej dalej "ustawą o podatku
dochodowym#"""


article = """



Dz.U. z 1998 r. Nr 144, poz. 930




                                    USTAWA
                                z dnia 20 listopada 1998 r.

          o zryczałtowanym podatku dochodowym od niektórych przychodów
                          osiąganych przez osoby fizyczne

                                    Rozdział 1
                                Przepisy ogólne

                                    Art. 1.
Ustawa reguluje opodatkowanie zryczałtowanym podatkiem dochodowym niektórych
przychodów (dochodów) osiąganych przez osoby fizyczne prowadzące pozarolniczą
działalność gospodarczą oraz przez osoby duchowne.
```

```
                                    Art. 2.
1. Osoby fizyczne osiągające przychody z pozarolniczej działalności
  gospodarczej opłacają zryczałtowany podatek dochodowy w formie:
   1) ryczałtu od przychodów ewidencjonowanych,
   2) karty podatkowej.
2. Osoby duchowne, prawnie uznanych wyznań, opłacają zryczałtowany podatek
  dochodowy od przychodów osób duchownych.
3. Wpływy z podatku dochodowego opłacanego w formie ryczałtu od przychodów
  ewidencjonowanych oraz zryczałtowanego podatku dochodowego od przychodów
  osób duchownych stanowią dochód budżetu państwa.
4. Wpływy z karty podatkowej stanowią dochody gmin.

                                    Art. 3.
Przychodów (dochodów) opodatkowanych w formach zryczałtowanych nie łączy się z
przychodami (dochodami) z innych źródeł podlegającymi opodatkowaniu na
podstawie ustawy z dnia 26 lipca 1991 r. o podatku dochodowym od osób
fizycznych (Dz. U. z 1993 r. Nr 90, poz. 416 i Nr 134, poz. 646, z 1994 r. Nr
43, poz. 163, Nr 90, poz. 419, Nr 113, poz. 547, Nr 123, poz. 602 i Nr 126,
poz. 626, z 1995 r. Nr 5, poz. 25 i Nr 133, poz. 654, z 1996 r. Nr 25, poz.
113, Nr 87, poz. 395, Nr 137, poz. 638, Nr 147, poz. 686 i Nr 156, poz. 776, z
1997 r. Nr 28, poz. 153, Nr 30, poz. 164, Nr 71, poz. 449, Nr 85, poz. 538, Nr
96, poz. 592, Nr 121, poz. 770, Nr 123, poz. 776, Nr 137, poz. 926, Nr 139,
poz. 932-934 i Nr 141, poz. 943 i 945 oraz z 1998 r. Nr 66, poz. 430, Nr 74,
poz. 471, Nr 108, poz. 685 i Nr 117, poz. 756), zwanej dalej "ustawą o podatku
dochodowym".#
"""
```

[ ]: