

Algorytmy i struktury danych z językiem Python

Algorytm Aho-Corasick

Angela Czubak

1 Teoria

1.1 Wprowadzenie

Algorytm Aho-Corasick został opracowany przez Alfreda V. Aho oraz Margaret J. Corasick. Jego celem jest znalezienie wzorców $\mathcal{P} = \{P_0, \dots, P_k\}$ pochodzących z pewnego słownika w tekście.

Cechą charakterystyczną tego algorytmu jest to, że szukanie wystąpień zadanych słów następuje "na raz", dzięki czemu złożoność obliczeniowa tego algorytmu wynosi $O(m + z + n)$, gdzie m - długość tekstu, w którym wyszukujemy, z - liczba wystąpień wzorców w zadanym tekście oraz $n = \sum_{i=0}^k |P_i|$ - sumy długości tychże.

Algorytm ten jest stosowany np. w komendzie UNIX-a - **fgrep**.

Idea algorytmu opiera się na drzewach trie i automatach, których tworzenie omówię dalej.

1.2 Drzewo trie

Definicja 1. *Drzewem trie dla zbioru wzorców \mathcal{P} nazywamy takie ukorzenione drzewo \mathcal{K} , że:*

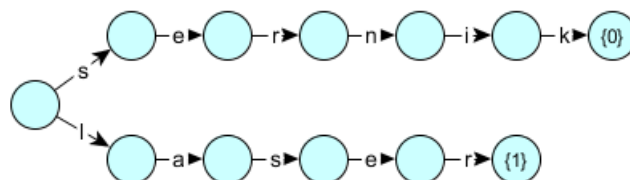
1. *Każda krawędź drzewa \mathcal{K} jest etykietowana jakimś znakiem*
2. *Każde dwie krawędzie wychodzące z jednego wierzchołka mają różne etykiety*

Definicja 2. *Etykietą węzła v nazywamy konkatencję etykiet krawędzi znajdujących się na ścieżce z korzenia do v . Oznaczamy ją jako $\mathcal{L}(v)$.*

3. *Dla każdego $P \in \mathcal{P}$ istnieje wierzchołek v taki, że $\mathcal{L}(v) = P$, oraz*
4. *Etykieta $\mathcal{L}(v)$ jakiegokolwiek liścia v jest równa jakiemuś $P \in \mathcal{P}$*

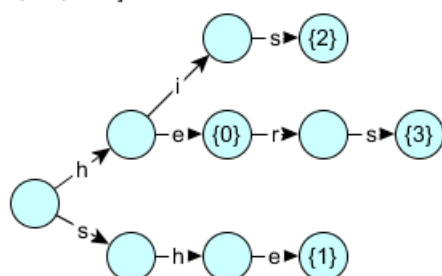
Przykładowe rysunki drzew trie znajdują się na następnej stronie. Numer w wierzchołku oznacza indeks słowa należącego do słownika, które jest etykietą tego wierzchołka.

$P=[\text{semik, laser}]$



Rysunek 1: Drzewo trie dla słów $\mathcal{P} = \{\text{semik, laser}\}$

$P=[\text{he, she, his, hers}]$



Rysunek 2: Drzewo trie dla słów $\mathcal{P} = \{\text{he, she, his, hers}\}$

1.3 Konstrukcja drzewa trie

Jak budować drzewo trie dla $\mathcal{P} = \{P_0, \dots, P_k\}$? Procedura jest następująca:

1. Rozpocznij od stworzenia korzenia
2. Umieszczaj kolejne wzorce jeden po drugim według poniższych kroków:
 - (a) Poczynając od korzenia, podążaj ścieżką etykietowaną kolejnymi znakami wzorca P_i
 - (b) Jeśli ścieżka kończy się przed P_i , to dodawaj nowe krawędzie i węzły dla pozostałych znaków P_i
 - (c) Umieść identyfikator i wzorca P_i w ostatnim wierzchołku ścieżki

Jak łatwo zauważyć, konstrukcja drzewa zajmuje $O(|P_0| + \dots + |P_k|) = O(n)$.

1.4 Wyszukiwanie wzorca w drzewie

Wyszukiwanie wzorca P odbywa się następująco:

Tak długo, jak to możliwe, podążaj ścieżką etykietowaną kolejnymi znakami P

1. Jeśli ścieżka prowadzi to wierzchołka z pewnym identyfikatorem, P jest słowem w naszym słowniku \mathcal{P}
2. Jeśli ścieżka kończy się przed P , to słowa nie ma w słowniku
3. Jeśli ścieżka kończy się w wierzchołku bez identyfikatora, to słowa nie ma w słowniku

Wyszukiwanie zajmuje więc $O(|P|)$.

Naiwnie postępując, moglibyśmy chcieć wyszukiwać wzorce w tekście tak, by dla każdego znaku tekstu próbować iść wzdłuż krawędzi odpowiadającym kolejnym znakom - jeśli po drodze przejdziemy przez wierzchołki z identyfikatorami, to znaleźliśmy słowa im odpowiadające. Gdy już nie ma krawędzi, którą moglibyśmy przejść, zaczynamy wyszukiwanie dla kolejnego znaku tekstu. Jednak takie wyszukiwanie zajęłoby $O(nm)$ czasu, gdzie m - długość tekstu, n - suma długości wzorców.

By przyspieszyć wyszukiwanie wzorców, rozszerzamy drzewo trie do **automatu**.

1.5 Automat

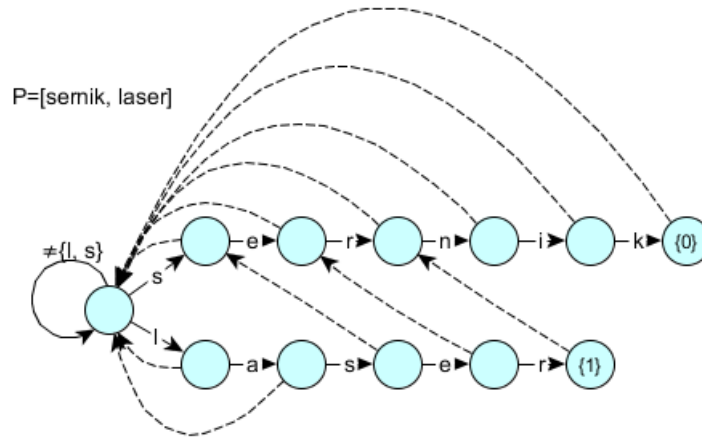
Definicja 3. *Automatem deterministycznym nazywamy piątkę uporządkowaną $(\Sigma, Q, q, \delta, F)$, gdzie*

1. Σ - skończony alfabet
2. Q - skończony zbiór stanów
3. $q \in Q$ - stan początkowy
4. $\delta : Q \times \Sigma \rightarrow Q$ - funkcja przejścia, przypisującą parze (q, a) nowy stan p , w którym znajdzie się automat po przeczytaniu symbolu a w stanie q
5. $F \subset Q$ - zbiór stanów końcowych

W naszym przypadku automat, w związku z wprowadzeniem dodatkowych pojęć, nie będzie ściśle deterministyczny. Automat będziemy budować na podstawie drzewa trie, zatem uściślam: zbiór stanów będą stanowiący węzły drzewa, a stanem początkowym będzie korzeń, do którego będą się czasami odnosiła jako 0. Wprowadzamy następujące funkcje:

1. Funkcja **goto**, oznaczana jako $g(q, a)$ - jest to odpowiednik funkcji przejścia δ , odpowiada ona krawędziom w drzewie, ponadto zachodzą jeszcze pewne własności; w skrócie:
 - jeśli krawędź (q, v) jest etykietowana przez a , to $g(q, a) = v$
 - $g(0, a) = 0$ dla każdego a nie będącego etykietą krawędzi wychodzącej z korzenia - automat ma pozostać w stanie początkowym, jeśli nie można znaleźć dopasowania
 - w przeciwnym przypadku $g(q, a) = \emptyset$ - brak przejścia w automacie
2. Funkcja **failure**, oznaczana jako $f(q)$, dla każdego stanu różnego od początkowego ($q \neq 0$) zwraca stan, do którego powinniśmy się udać w przypadku niemożności zastosowania funkcji $g(q, a)$ - nie istnieje krawędź wychodząca z q , etykietowana przez a . Stanem tym jest węzeł odpowiadający najdłuższemu właściwemu sufiksowi $L(q)$ (najdłuższemu podsłowu

y , krótszemu niż samo słowo s , takiemu, że istnieje słowo t o niezerowej długości, że $s = ty$). Chodzi o to, by nie przegapić żadnego potencjalnego dopasowania wzorca - np. biorąc słowa *laser*, *sernik* i szukając w tekście *lasernik*, zaczniemy od dopasowania do słowa *laser*, a powinniśmy mieć jeszcze możliwość przejścia do gałęzi odpowiadającej słowu *sernik*, by także je odnaleźć. Funkcje przejścia w tym przypadku przedstawiono na rysunku (3). Funkcja $f(q)$ jest zawsze dobrze zdefiniowana, gdyż $\mathcal{L}(0) = \epsilon$ jest sufiksem każdego słowa.



Rysunek 3: Automat dla słów $\mathcal{P} = \{\textit{sernik}, \textit{laser}\}$. Przerywaną linią oznaczono krawędzie odpowiadające funkcji $f(q)$

3. Funkcja **wyjścia**, oznaczana jako $out(v)$, zwraca indeksy wzorców, do których znajdujemy dopasowanie w stanie q .

1.6 Przeszukiwanie tekstu

Załóżmy, że mamy do dyspozycji gotowy automat oraz tekst $T[1 \dots]$, w którym szukamy wzorców. Procedura ta prezentuje się w następujący sposób:

Algorithm 1.1: SEARCH($T[0 \dots m - 1]$)

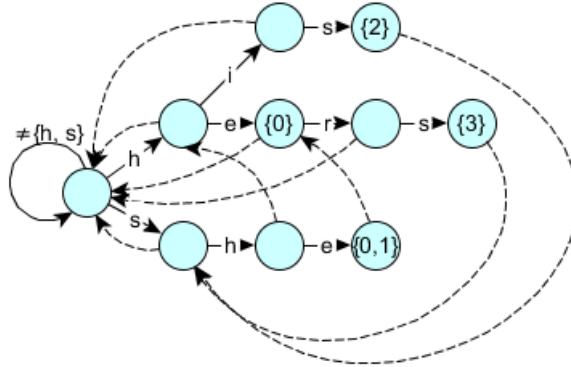
```

 $q \leftarrow 0$ 
for  $i \leftarrow 0$  to  $m - 1$ 
do
  while  $g(q, T[i]) = \emptyset$ 
  do
     $q \leftarrow f(q)$ 
    comment: podążaj za funkcją failure, aż znajdziesz dopasowanie
   $q \leftarrow g(q, T[i])$ 
  comment: przejdź do dopasowanego stanu
  if  $out(q) \neq \emptyset$ 
  then output  $(i, out(q))$ 

```

Weźmy automat jak na rysunku (4):

$P=[he, she, his, hers]$



Rysunek 4: Automat dla słów $\mathcal{P} = \{he, she, his, hers\}$. Przerywaną linią oznaczono krawędzie odpowiadające funkcji $f(q)$

Przeszukamy przy jego pomocy tekst *ushers*:

1. Czytamy znak u - zostajemy w korzeniu
2. Czytamy znak s - przechodzimy po odpowiedniej krawędzi, czyli idziemy do węzła etykietowanego s
3. Czytamy znak h - przechodzimy po odpowiedniej krawędzi, czyli idziemy do węzła etykietowanego sh
4. Czytamy znak e - przechodzimy po odpowiedniej krawędzi, czyli idziemy do węzła etykietowanego she ; wypisujemy, że znaleźliśmy słowa o indeksach 0 i 1 na pozycji 3
5. Czytamy znak r - korzystamy z krawędzi **failure**, następnie przechodzimy po odpowiedniej krawędzi, czyli idziemy do węzła etykietowanego her
6. Czytamy znak s - przechodzimy po odpowiedniej krawędzi, czyli idziemy do węzła etykietowanego $hers$; wypisujemy, że znaleźliśmy słowo o indeksie 3 na pozycji 5

Złożoność czasowa takiej procedury jest $O(m + z)$, gdzie m - długość tekstu, w którym wyszukujemy, a z - liczba wystąpień wzorca w tekście.

Wynika to z faktu, że liczba wywołań funkcji $f(q)$ jest ograniczona z góry przez m - w danym momencie możemy wywołać funkcję $f(q)$ co najwyżej tyle razy, ile znaków zdążyliśmy przeczytać z T , a w sumie możemy ich przeczytać m (po wywołaniu tej funkcji "przesuwamy" początek sufiksu na pewną literę z ciągu T - przesunąć ten początek możemy maksymalnie m razy).

Podobnie, funkcja $g(q, a)$ jest wywoływana dokładnie raz dla każdego $a \in T$ - zostanie ona wywołana m razy. Wystąpienie wzorca możemy zgłaszać w czasie stałym, stąd zgłoszenie wszystkich zajmie $O(z)$.

1.7 Budowa automatu

W konstrukcji automatu możemy wyróżnić dwie fazy.

1.7.1 Faza I

1. Tworzymy drzewo trie dla słownika \mathcal{P}
 item Dla każdego $P_i \in \mathcal{P}$ ustawiamy $out(v) = i$ dla wierzchołka v etykietowanego przez P_i
2. Uzupełnij funkcje przejść dla korzenia

$$g(0, a) = 0$$

dla każdego znaku a (należącego do alfabetu Σ) takiego, że nie etykietuje on żadnej krawędzi wychodzącej z korzenia

Złożoność czasowa takiej procedury, dla pewnego niezmiennego alfabetu, wynosi $O(n)$, gdzie $n = \sum_{i=0}^k |P_i|$.

1.7.2 Faza II

Przedstawię ją w postaci pseudokodu:

Algorithm 1.2: PHASE2(void)

```

Q ← QUEUE()
for a ∈ Σ
  do { if q ← g(0, a) ≠ 0
      then { f(q) ← 0
           Q.ENQUEUE(q)
    while !Q.ISEMPTY()
      do { r ← Q.DEQUEUE()
          for a ∈ Σ
            do if u ← g(r, a) ≠ ∅
                then { Q.ENQUEUE(u)
                     v ← f(r)
                     while g(v, a) = ∅
                       do v ← f(v)/(*)
                     f(u) ← g(v, a)
                     out(u) ← out(u) ∪ out(f(u))/(**)

```

Jak widać funkcje f i out są wyliczane dla wierzchołków w kolejności BFS. Dzięki temu wierzchołki znajdujące się bliżej korzenia zostały już obsłużone, gdy zachodzi potrzeba skorzystania z odpowiednich funkcji na nich wykonywanych.

Rozważmy wierzchołki r i $u = g(r, a)$, w takim przypadku r jest rodzicem u . Co więcej, $\mathcal{L}(u) = \mathcal{L}(r)a$. Jakie więc powinno być $f(u)$? Przypomnijmy, że $f(u)$ powinno wskazywać na najdłuższy właściwy sufix $\mathcal{L}(u)$. Z tego wynika, że powinniśmy spróbować dopasować $f(u) = g(f(r), a)$, bo $\mathcal{L}(f(u))$ może być sufiksem $\mathcal{L}(g(f(r), a))$, o ile taka krawędź istnieje. Jeśli jej nie ma, to próbujemy

$f(u) = g(f(f(r)), a)$, itd., aż znajdziemy odpowiedni wierzchołek (pesymistycznie może to być korzeń). W linii oznaczonej (*) wykonujemy właśnie te czynności.

Czynności oznaczone (**) wykonujemy, gdyż wzorce rozpoznawane w stanie $f(u)$ (i jedynie te) są właściwymi sufiksami $\mathcal{L}(u)$ i dlatego powinny być rozpoznawane także w stanie u .

Jaka jest złożoność powyższej procedury? Zauważmy, że jest ona podobna do BFS-a - stąd przechodzenie po drzewie, pomijając linię oznaczoną (*), zajmie czas proporcjonalny do rozmiaru drzewa - tj. $O(n)$. A jak określić, ile razy wykonamy linię (*)?

Rozważmy ścieżkę złożoną z wierzchołków u_1, \dots, u_l , która jest tworzona podczas dodawania wzorca $a_1 \dots a_l$. Oznaczmy dodatkowo $df(u)$ jako głębokość w drzewie wierzchołka $f(u)$, zatem $df(u_1), \dots, df(u_l)$ to ciąg głębokości dla wierzchołków z rozważanej ścieżki, wszystkie są ≥ 0 .

Zauważmy ponadto, że głębokość kolejnego wierzchołka może wzrosnąć co najwyżej o 1, czyli $df(u_{i+1}) \leq df(u_i) + 1$, zatem wartości df wzrastają sumarycznie co najwyżej o l podczas przechodzenia tej ścieżki.

Kiedy wyliczamy położenie $f(u_{i+1})$, każde wywołanie linii (*) przybliży v do korzenia, a stąd wartość $df(u_{i+1})$ będzie mniejsza od $df(u_i)$ co najmniej o jeden. W związku z ograniczeniem od dołu, możemy zmniejszać kolejne wartości $df(u_i)$ co najwyżej tyle, ile razy zostały one zwiększone, czyli linia (*) zostanie wykonana $\leq l$ razy dla pewnego wzorca o długości l .

Sumaryczna długość wzorców wynosi n , dlatego podczas budowy automatu linia (*) zostanie wykonana co najwyżej n razy.

Zastanówmy się jeszcze, ile czasu zajmuje wykonanie linii (**). Zauważmy, że przed wykonaniem przypisania, $out(u) = \emptyset$ albo $out(u)$ jest równy indeksowi $\mathcal{L}(u)$. Jakielwiek wzorce znajdują się w $out(f(u))$, są one na pewno krótsze niż $\mathcal{L}(u)$, bo $f(u)$ jest bliżej korzenia - zatem zbiory te są rozłączne. Możemy więc przyjąć, że reprezentujemy je przez listy, do których da się dołączać drugą w stałym czasie.

Zatem złożoność czasowa drugiej fazy wynosi $O(n)$.

1.8 Determinizacja automatu

Ze względu na występowanie w automacie funkcji $f(u)$ jest on nie deterministyczny - nie znamy przejść ze wszystkich stanów dla wszystkich możliwych znaków (np. a), będziemy musieli więc czasem wykonać wiele przejść z użyciem funkcji $f(u)$, aż dojdziemy do stanu v , w którym istnieje dobrze określone przejście $g(v, a)$. Można jednak podejść inaczej o budowy automatu - mianowicie wprowadzić funkcję $next(u, a)$, którą wyliczamy w następujący sposób:

Algorithm 1.3: PHASE3(*void*)

```

Q ← QUEUE()
for a ∈ Σ
  do { if q ← g(0, a) ≠ 0
        then Q.ENQUEUE(q)
        next(0, a) ← g(0, a)
  while !Q.ISEMPTY()
    do { r ← Q.DEQUEUE()
          for a ∈ Σ
            do if u ← g(r, a) = ∅
                  then { Q.ENQUEUE(u)
                         v ← f(r)
                         while g(v, a) = ∅
                           do v ← f(v)
                         next(r, a) ← g(v, a)
                  else { next(r, a) ← g(r, a)
                        Q.ENQUEUE(g(r, a))

```

Procedurę tę wykonujemy po wyliczeniu funkcji $f(u)$. Wtedy wyszukiwanie upraszcza się do:

Algorithm 1.4: SEARCHDETERMINISTIC($T[0 \dots m-1]$)

```

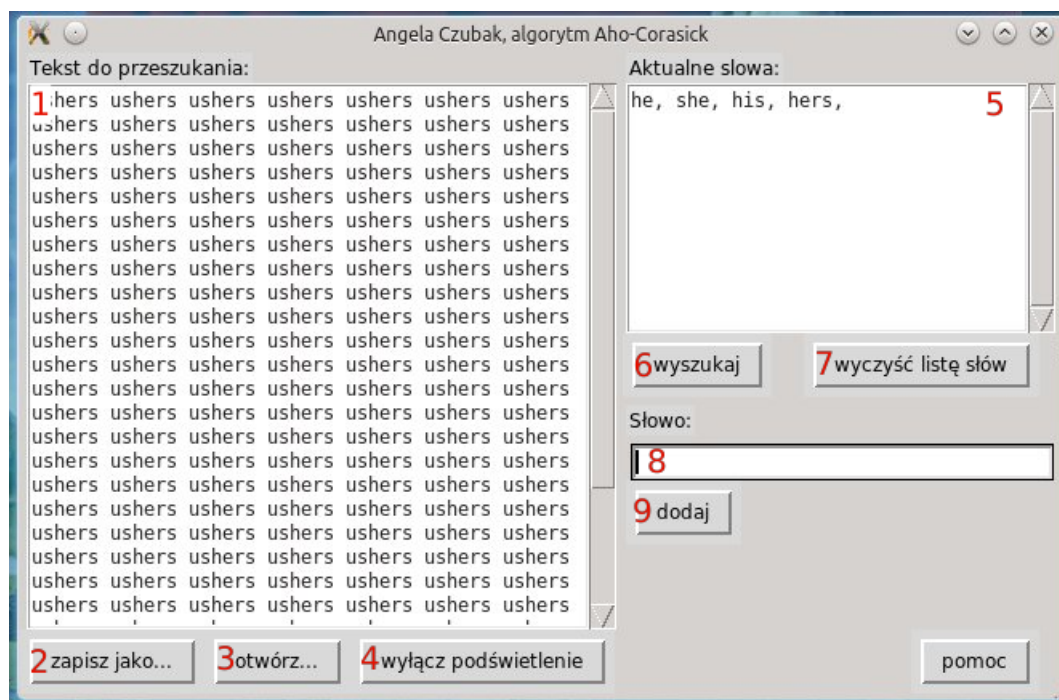
q ← 0
for i ← 0 to m-1
  do { q ← next(q, T[i])
        comment: przejście jest deterministyczne
        if out(q) ≠ ∅
          then output (i), out(q)

```

Jednak wprowadzenie takich przejść wiąże się ze znacznym obciążeniem pamięciowym, dlatego ja w swojej implementacji pominę te kroki.

2 Opis interfejsu

Zaimplementowano interfejs graficzny ułatwiający korzystanie z napisanego kodu. Został on przedstawiony na rysunku (5).



Rysunek 5: Interfejs graficzny programu zaliczeniowego

Poniżej znajduje się opis poszczególnych elementów interfejsu:

1. TEKST DO PRZESZUKANIA

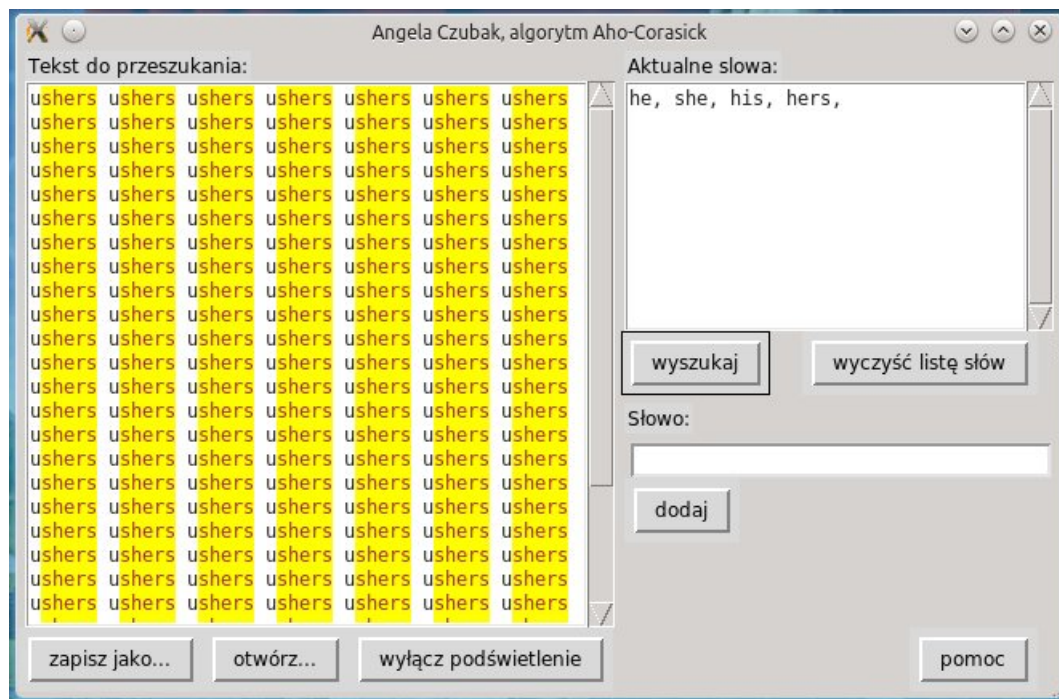
Jest to pole, w którym umieszczamy tekst, w którym będziemy wyszukiwać wzorce. Możemy tam wprowadzać tekst wprost z klawiatury lub wczytać tekst z pliku. W tym drugim przypadku należy kliknąć przycisk **otwórz...**, a następnie wybrać plik z użyciem okna dialogowego. Tekst, który wprowadzimy do tego pola możemy zapisać. By to zrobić, należy kliknąć przycisk **zapisz jako...**, a następnie wybrać odpowiednią nazwę pliku. Po wykonaniu wyszukania znalezione wzorce zostaną podświetlone na żółto (patrz rysunek (6)). By wyłączyć to podświetlenie, należy kliknąć na przycisk **wyłącz podświetlenie**.

2. ZAPISZ JAKO...

Tekst, który wprowadzimy do tego pola możemy zapisać. By to zrobić, należy kliknąć przycisk **zapisz jako...**, a następnie wybrać odpowiednią nazwę pliku.

3. OTWÓRZ...

Zamiast wpisywać tekst, możemy otworzyć gotowy plik tekstowy. W tym celu należy kliknąć przycisk **otwórz...**, a następnie wybrać plik z użyciem okna dialogowego.



Rysunek 6: Interfejs graficzny programu zaliczeniowego, działanie podświetlenia

4. WYŁĄCZ PODŚWIETLENIE

Po wykonaniu wyszukiwania znalezione wzorce zostaną podświetlone na żółto (patrz rysunek (6)). By wyłączyć to podświetlenie, należy kliknąć na przycisk **wyłącz podświetlenie**.

5. AKTUALNE SŁOWA

W tym polu znajdują się słowa (wzorce), które będą wyszukiwane w tekście przy użyciu algorytmu Aho-Corasick. By wyszukać wzorce, należy kliknąć przycisk **wyszukaj**, wtedy znalezione wystąpienia z pola **Tekst do przeszukania** zostaną podświetlone na żółto (patrz rysunek (6)). By wyczyścić listę słów, należy kliknąć przycisk **wyczyść listę słów**. By dodać słowo należy umieścić wymyślony przez nas wzorzec w polu **Słowo**, a następnie wcisnąć ENTER na klawiaturze lub przycisk **dodaj**.

6. WYSZUKAJ

By wyszukać wzorce, należy kliknąć przycisk **wyszukaj**, wtedy znalezione wystąpienia z pola **Tekst do przeszukania** zostaną podświetlone na żółto (patrz rysunek (6)).

7. WYCZYŚĆ LISTĘ SŁÓW

By wyczyścić listę słów, należy kliknąć przycisk **wyczyść listę słów**.

8. SŁOWO

W tym polu wpisujemy wzorec, który chcemy wyszukiwać w tekście znajdującym się w polu **Tekst do wyszukania**. Następnie należy nacisnąć ENTER na klawiaturze lub przycisk **dodaj**.

9. DODAJ

Po wpisaniu słowa w polu **Słowo**, które chcemy wyszukać w tekście znajdującym się w polu **Tekst do wyszukania**, można nacisnąć przycisk **dodaj**, by dodać słowo do listy wzorców.

3 Kod źródłowy

Cały kod oraz opis zmian można podejrzeć na <https://github.com/cosmia/pythonProject>.

3.1 Klasa MyList

Ponieważ złożoność dołączania jednej listy do drugiej w *Pythonie* jest zależna od długości tej drugiej, postanowiono napisać własną implementację listy tak, aby łącznie następowało w czasie stałym.

3.1.1 MyListError

Wyjątek związany z operacjami na obiektach klasy *MyList*

3.1.2 Element

Klasa opisująca element listy.

Posiada ona następujące **poła**:

1. *arg* - zawartość tego elementu listy
2. *follow* - następny element na liście

Metody:

1. konstruktor `__init__(self, arg=None, follow=None)`- *arg* - element znajdujący się w liście, *follow* - następny element na liście
2. `setData(self, arg)` - ustawia zawartość elementu listy na *arg*
3. `setNext(self, follow)` -ustawia następny element na liście na *follow*, rzuca wyjątek *MyListError*, jeśli *follow* nie jest klasy *Element*
4. `getData(self)` - zwraca zawartość elementu listy
5. `getNext(self)` - zwraca następny element na liście

3.1.3 MyList

Klasa opisująca moją wersję listy.

Posiada ona cztery **poła**:

1. *first* - pierwszy element listy, jeśli lista jest pusta, jest to *None*
2. *last* - ostatni element listy, jeśli jest to lista pusta, jest to *None*
3. *current* - jest to zmienna pomocnicza, używana przy iterowaniu listy, początkowo ustawiona na *None*
4. *length* - długość listy

Metody:

1. konstruktor bezargumentowy
2. *add(self, argument)* - dodaje element argument do listy na ostatnią pozycję
3. *__iadd(self, other)__* - dodaje do siebie dwa obiekty *MyList*, zmienia pierwszy obiekt, zwraca wskaźnik na pierwszy obiekt; rzuca *MyListError*, jeśli drugi argument nie jest obiektem *MyList*
4. *__iter(self)__* - zwraca iterator dla listy *MyList*
5. *next(self)* - zwraca następny element na liście, metoda dla iteratora
6. *__len(self)__* - zwraca długość listy
7. *__eq(self, other)__* - metoda porównująca listy, zwraca *True*, jeśli listy równe, *False* wpp
8. *__ne(self, other)__* - metoda sprawdzająca, czy listy są różne, zwraca *True*, jeśli tak; *False* wpp
9. *__contains(self, other)__* - metoda sprawdzająca, czy lista zawiera *other*, zwraca *True*, jeśli tak; *False* wpp

3.1.4 myListy.py

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  class MyListError(Exception):
5      '''wyjatek dla klasy MyList'''
6      def __init__(self, mes):
7          '''konstruktor, argumentem tresc przy rzucaniu wyjatku'''
8          self.value = mes
9      def __str__(self):
10         '''podaje tresc wyjatku'''
11         return self.value
12
13  class Element:
14      '''klasa opisujaca element MyList'''
```

```

15     def __init__(self, arg=None, follow=None):
16         '''konstruktor; arg - element znajdujacy sie w liscie,
17            follow - nastepny element na liscie'''
18         if follow is not None and not isinstance(follow, Element):
19             raise MyListError("argument is not an Element")
20         self.arg = arg
21         self.follow = follow
22     def setData(self, arg):
23         '''ustawienie zawartosci elementu listy na arg'''
24         self.arg = arg
25     def setNext(self, follow):
26         '''ustawienie nastepnego elementu na liscie na follow'''
27         if not isinstance(follow, Element):
28             raise MyListError("argument is not an Element")
29         self.follow = follow
30     def getData(self):
31         '''zwraca zawartosc elementu listy'''
32         return self.arg
33     def getNext(self):
34         '''zwraca nastepny element na liscie'''
35         return self.follow
36
37     class MyList:
38         '''lista, ktora bedzie mozna laczyć z druga w czasie stalym
39            jest to uproszczona lista, nie zawiera np. usuwania elementow,
40            gdyz nie wydaje sie to potrzebne'''
41     def __init__(self):
42         '''konstruktor, tworzy pusta liste'''
43         self.first = None
44         self.last = None
45         self.current = None
46         self.length = 0
47     def add(self, argument):
48         '''dodaje argument do listy na ostatniej pozycji'''
49         if self.first is None:
50             self.first = Element(argument, None)
51             self.last = self.first
52         else:
53             tmp = Element(argument, None)
54             self.last.setNext(tmp)
55             self.last = tmp
56         self.length += 1
57     def __iadd__(self, other):
58         '''dodaje do siebie dwa obiekty MyList
59            zmienia pierwszy obiekt, zwraca wskaznik na pierwszy obiekt'''
60         if other is None or other.first is None:
61             return self
62         if not isinstance(other, MyList):
63             raise MyListError("the other argument is not a MyList")
64         if self.first is None:

```

```

65         self.first = other.first
66         self.length = other.length
67         self.last = other.last
68         return self
69         #print other.first
70         self.last.setNext(other.first)
71         self.length += other.length
72         return self
73     def __iter__(self):
74         '''metoda zwracajaca iterator'''
75         self.current = self.first
76         return self
77     def next(self):
78         '''zwraca nastepny element na liscie'''
79         if self.current is None:
80             raise StopIteration
81         else:
82             tmp = self.current.getData()
83             self.current = self.current.getNext()
84             return tmp
85     def __len__(self):
86         '''metoda zwracajaca dlugosc listy'''
87         return self.length
88     def __eq__(self, other):
89         '''metoda porownujaca listy
90         zwraca True, jesli listy rowne, False wpp'''
91         if not isinstance(other, MyList):
92             return False
93         dl = len(other)
94         if dl != len(self):
95             return False
96         iter1 = iter(self)
97         iter2 = iter(other)
98         for i in range(dl):
99             e1 = iter1.next()
100             e2 = iter2.next()
101             if e2 != e1:
102                 return False
103         return True
104     def __ne__(self, other):
105         '''metoda sprawdzajaca, czy listy sa rozne
106         zwraca True, jesli tak; False wpp'''
107         return not self == other
108     def __contains__(self, other):
109         '''metoda sprawdzajaca, czy lista zawiera other
110         zwraca True, jesli tak; False wpp'''
111         for i in self:
112             if other == i:
113                 return True
114         return False

```

3.2 Klasa Node

Przyjęłam następującą konwencję: string to dla mnie zmienna *str* lub *unicode*

3.2.1 NodeError

Wyjątek związany z operacjami na obiektach klasy Node

3.2.2 Node

Klasa opisująca węzeł w drzewie/automacie.

Posiada ona trzy **pola**:

1. *accept* - obiekt *MyList*. Jeśli jest to lista pusta, to taki węzeł nie jest akceptujący. W przeciwnym przypadku lista ta zawiera wartości numeryczne, a te numery są indeksami słów, które zostały zaakceptowane
2. *edges* - słownik, początkowo pusty. Jego kluczami są znaki (stringi od długości 1), natomiast wartościami są inne węzły, tj. *edges['a']* wskazuje na węzeł, do którego powinniśmy przejść z tego stanu, jeśli przeczytamy literę 'a'
3. *fail* - krawędź porażki (ang. *fail edge*) - opisuje ona do jakiego stanu przejść, jeśli nie ma żadnej innej pasującej krawędzi do przeczytanego znaku. Znak ten będziemy próbowali przetworzyć w wskazanym przez nią stanie, który odpowiada najdłuższemu właściwemu sufiksowi słowa, do którego próbowaliśmy dopasować do tej pory

Metody:

1. konstruktor bezargumentowy *__init__(self)*
2. *labelCorrect(self, label)* - funkcja pomocnicza sprawdzająca, czy etykieta *label* spełnia warunki etykiety, rzuca wyjątkiem *NodeError* wpp
3. *nodeCorrect(self, node)* - funkcja pomocnicza sprawdzająca, czy *node* jest węzłem (obiektem klasy *Node*), rzuca wyjątkiem *NodeError* wpp
4. *getAccept(self)* - zwraca listę (obiekt *MyList*) indeksów akceptowanych słów lub listę pustą, jeśli ten stan nie akceptuje żadnego słowa
5. *getLabels(self)* - zwraca listę etykiet krawędzi wychodzących z tego węzła (nie dotyczy *fail*)
6. *getAim(self, label)* - zwraca węzeł (obiekt klasy *Node*), na który wskazuje krawędź etykietowana przez *label* lub *None*, jeśli nie ma takiej krawędzi; rzuca *NodeError*, jeśli etykieta *label* niepoprawna
7. *getFail(self)* - zwraca węzeł, który opisuje stan odpowiadający najdłuższemu właściwemu sufiksowi bieżącego węzła
8. *setAccept(self, numer)* - ustalamy, że ten węzeł akceptuje słowo indeksowane numerem *numer* lub listę (obiekt *MyList*) indeksów znajdujących się w *numer*; rzuca *NodeError*, jeśli *numer* nie jest listą (obiektem *MyList*) całkowitych liczb nieujemnych lub całkowitą liczbą nieujemną

9. *setAim(self, label, node)* - ustawia następującą krawędź wychodzącą ze stanu (obiektu, na którym wywoływana metoda): przy przeczytaniu w tym stanie znaku *label* powinniśmy przejść do węzła *node*, rzuca *NodeError*, jeśli etykieta *label* niepoprawna lub *node* nie jest obiektem klasy *Node*
10. *setFail(self, node)* - ustalamy, że najdłuższy właściwy sufiks słowa rozważonego do tej pory odpowiada węzłowi *node*, rzuca *NodeError*, jeśli *node* nie jest obiektem klasy *Node*

3.2.3 node.py

```

1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  from myList import *
5
6  class NodeError(Exception):
7      '''wyjatek dla klasy Node'''
8      def __init__(self, value):
9          '''konstruktor, argumentem tresc przy rzucaniu wyjatku'''
10         self.napis = value
11     def __str__(self):
12         '''podaje powod wyjatku'''
13         return repr(self.napis)
14     __repr__ = __str__
15
16 class Node:
17     '''klasa opisujaca wezel/stan w automacie/drzewie Trie'''
18     def __init__(self):
19         '''konstruktor bezargumentowy
20         accept = MyList() - pusta lista,
21         fail = None
22         edges = {}'''
23         self.accept = MyList()
24         self.edges = {}
25         self.fail = None
26     def labelCorrect(self, label):
27         '''sprawdza, czy label jest poprawna etykieta krawedzi
28         jesli nie, rzuca NodeError'''
29         if not isinstance(label, (str, unicode)):
30             raise NodeError("label must be a character")
31         if len(label) != 1:
32             raise NodeError("label must be exactly one character long")
33     def nodeCorrect(self, node, n=1):
34         '''sprawdza, czy node jest obiektem klasy Node
35         jesli nie, to rzuca NodeError
36         n to numer argumentu, ktorym jest node w jakiej funkcji
37         sluzy uszczegolowieniu, ktory argument jest bledny'''
38         if not isinstance(node, Node):
39             lan = "the "
```



```

40         if n != 1: lan += "second "
41         lan += "argument must be a node"
42         raise NodeError(lan)
43     def getAccept(self):
44         '''zwraca liste indeksow slow, ktore akceptuje ten wezel,
45         lista jest pusta, jesli ten wezel niczego nie akceptuje'''
46         return self.accept
47     def getLabels(self):
48         '''zwraca liste etykiet dla krawedzi wychodzacych z tego wezla'''
49         return self.edges.keys()
50     def getAim(self, label):
51         '''zwraca wezel, do ktorego prowadzi krawedz z etykieta label
52         jesli brak takiej krawedzi, zwraca None'''
53         self.labelCorrect(label)
54         if label not in self.edges:
55             if self.fail is None:
56                 return self
57             else:
58                 return None
59         else:
60             return self.edges[label]
61     def getFail(self):
62         '''zwraca wezel odpowiadajacy najdluzszemu wlasciwemu sufiksowi
63         slova, ktore do ktorego probowalismy znalezc dopasowanie'''
64         return self.fail
65     def setAccept(self, number):
66         '''ustalamy, ze ten wezel akceptuje slowo o indeksie number lub slowa o
67         indeksach z MyList number
68         rzuca NodeError, jesli number nie jest calkowita liczba
69         nieujemna albo obiektem MyList calkowitych liczb nieujemnych'''
70         if isinstance(number, MyList):
71             for i in number:
72                 if not isinstance(i, (long, int)) or i < 0:
73                     mes = ("argument should be a non-negative integer" +
74                           " or long or a set of those")
75                     raise NodeError(mes)
76                 #print number
77             self.accept += number
78             return
79         if not isinstance(number, (long, int)):
80             raise NodeError(("argument should be an integer or long"+
81                             " or a set of those"))
82         if number < 0:
83             raise NodeError("argument must be non-negative")
84         self.accept.add(number)
85     def setAim(self, label, node):
86         '''ustalamy, ze z tego wezla bedzie wychodzic krawedz
87         etykietowana label i bedzie ona prowadzic do node
88         rzuca NodeError, jesli label niepoprawna lub node nie jest wezlem
89         '''

```

```

90         self.labelCorrect(label)
91         self.nodeCorrect(node,2)
92         self.edges[label] = node
93     def setFail(self, node):
94         '''ustalamy, ze najdluzszy sufiks slowa, do ktorego probowalismy
95            dopisowac w tym wezle odpowiada wezlowi node
96            rzuca wyjatkiem, jesli node nie jest wezlem'''
97         self.nodeCorrect(node)
98         self.fail = node

```

3.3 Klasa AhoCorasick

Przyjęłam następującą konwencję: string to dla mnie zmienna *str* lub *unicode*

3.3.1 AhoCorasickError

Wyjątek związany z operacjami na obiektach klasy AhoCorasick

3.3.2 AhoCorasick

Klasa opisująca drzewo/automat.

Posiada ona trzy **poła**:

1. *n* - korzeń drzewa. Na początku nie ma on żadnych krawędzi, jest to domyślny obiekt tworzony jako *Node*.
2. *words* - lista słów. Na początku jest ona pusta, powiększa się przy dodawaniu słów.
3. *built* - zmienna mówiąca, czy został już zbudowany automat, początkowo wynosi *False*

Metody:

1. konstruktor bezargumentowy *__init__(self)*
2. *addWord(self, word)* - dodaje słowo *word* do automatu/drzewa. Rzuca *AhoCorasickError*, jeśli *word* nie jest stringiem lub zbudowano już automat
3. *lookUp(self, word)* - sprawdza, czy słowo *word* występuje w drzewie, zwraca *True*, jeśli tak; *False* w przeciwnym przypadku. Rzuca *AhoCorasickError*, jeśli *word* nie jest stringiem
4. *build(self)* - buduje automat skończony na podstawie drzewa powstałego w skutek dodawania słów metodą *AhoCorasick.addWord*
5. *makeTree(self, wordList)* - konstruuje drzewo i automat na podstawie listy słów *wordList*, także dodaje do istniejącego automatu wzorce z *wordList*, tj. jeśli w automacie są już jakieś słowa, to nie usuwa ich. Rzuca *AhoCorasickError*, jeśli *wordList* nie jest listą stringów lub zbudowano już automat

6. `clear(self)` - czyści automat i drzewo; po wykonaniu tej metody obiekt klasy jest w stanie jak zaraz po wywołaniu konstruktora
7. `search(self, tekst, returnSet=False)` - wyszukuje wzorce w zmiennej tekst, zwraca string z wiadomością o wynikach; domyślny argument `returnSet` mówi o formacie zwracanej wartości: jeśli `returnSet` jest `False`, to zwracamy string z informacjami; jeśli `returnSet` jest `True`, to zwracamy zbiór krotek o długości dwa, krotka zawiera, w podanej kolejności, pozycję, na której znalazła słowo, oraz indeks słowa; jeśli tekst nie jest zmienną string, to rzuca `AhoCorasickError`

3.3.3 ahoCorasick.py

```

1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  from node import *
5  from Queue import *
6
7  class AhoCorasickError(Exception):
8      '''wyjatek dla klasy Node'''
9      def __init__(self, value):
10         '''konstruktor, argumentem tresc przy rzucaniu wyjatku'''
11         self.napis = value
12     def __str__(self):
13         '''podaaj powod wyjatku'''
14         return repr(self.napis)
15
16 class AhoCorasick:
17     '''klasa opisujaca drzewo Trie / automat, sluzacy wyszukiwaniu wzorcow'''
18     def __init__(self):
19         '''konstruktor bezargumentowy
20         n - korzen drzewa, pusty
21         words - pusta liczba slow zakodowanych w drzewie'''
22         self.n = Node()
23         self.words = []
24         self.built = False
25     def addWord(self, word):
26         '''dodaje slowo word do automatu/drzewa
27         rzuca AhoCorasickError, jesli word nie jest stringiem
28         lub zbudowano juz automat'''
29         if self.built:
30             raise AhoCorasickError("automaton has been built already")
31         if not isinstance(word, (str, unicode)):
32             raise AhoCorasickError("argument is not a string")
33         #zamiana na unicode!!!
34         if isinstance(word, str): word = word.decode("utf-8")
35         dl = len(word)
36         if dl == 0: return #nie dodajemy pustego slowa
37         wezel = self.n

```

```

38     i = 0
39     #idziemy dopoki mozemy po istniejacych wezlach
40     while i < dl:
41         litera = word[i]
42         labels = wezel.getLabels()
43         if litera in labels:
44             wezel = wezel.getAim(litera)
45         else:
46             break
47         i += 1
48     #a teraz tworzymy nowe, jesli taka potrzeba
49     while i < dl:
50         litera = word[i]
51         wezel.setAim(litera, Node())
52         wezel = wezel.getAim(litera)
53         #wezel.setFail(self.n)
54         #na poczatku najdluzszy wlasciwy sufiks to slowo puste
55         #mozna to w sumie robic przy budowaniu automatu...
56         i += 1
57     #jesli jeszcze nie dodalismy tego slowa
58     if wezel.getAccept() == MyList():
59         ktore = len(self.words)
60         wezel.setAccept(ktore)
61         self.words.append(word)
62     def lookUp(self, word):
63         '''sprawdza, czy dane slowo wystepuje w drzewie
64         zwraca True, jesli tak; False wpp
65         rzuca AhoCorasickError, jesli word nie jest strigiem'''
66         if not isinstance(word, (str,unicode)):
67             raise AhoCorasickError("argument is not a string")
68         if word == "": return False
69         if isinstance(word, str): word = word.decode("utf-8")
70         i = 0
71         dl = len(word)
72         wezel = self.n
73         while i < dl:
74             litera = word[i]
75             labels = wezel.getLabels()
76             if litera not in labels:
77                 return False
78             wezel = wezel.getAim(litera)
79             i += 1
80         if wezel.getAccept() != MyList():
81             return True
82         return False
83     def build(self):
84         '''konstruuje automat skonczony na podstawie drzewa, ktore
85         powstaje podczas dodawania slow metoda addWord'''
86         q = Queue()
87         for i in self.n.getLabels():

```

```

88         s = self.n.getAim(i)
89         s.setFail(self.n)
90         q.put(s)
91     while not q.empty():
92         r = q.get()
93         for a in r.getLabels():
94             u = r.getAim(a)
95             q.put(u)
96             v = r.getFail()
97             while v.getAim(a) is None: #jesli stad nie ma tego przejścia
98                 v = v.getFail() #to szukaj krotszego dopasowania
99             u.setFail(v.getAim(a))
100             #dodaj nowe akceptowane slow
101             u.setAccept(u.getFail().getAccept())
102     self.built = True
103 def makeTree(self, wordList):
104     '''konstruuje drzewo i automat na podstawie listy slow wordList
105     takze dodaje do istniejacego automatu wzorce z wordList
106     rzuca AhoCorasickError, jesli wordList nie jest lista stringow
107     lub zbudowano juz automat'''
108     if self.built:
109         raise AhoCorasickError("automaton has been built already")
110     if not isinstance(wordList, list):
111         raise AhoCorasickError("argument is not a list")
112     for i in wordList:
113         if not isinstance(i, (str, unicode)):
114             raise AhoCorasickError("element of the list is not a string")
115     for i in wordList:
116         self.addWord(i)
117     self.build()
118 def clear(self):
119     '''czysci automat i drzewo'''
120     self.words = []
121     self.n = Node()
122     self.built = False
123 def search(self, tekst, returnSet=False):
124     '''wyszukuje wzorce w zmiennej tekst, zwraca string z wiadomoscia o
125     wynikach
126     domyslne argument returnSet mowi o formacie zwracanej wartosci
127     jesli returnSet jest False, to zwracamy string z informacjami
128     jesli returnSet jest True, to zwracamy zbior krotek o dlugosci dwa,
129     krotka zawiera pozycje, na ktorej znalazla slowo, oraz indeks slowa
130     jesli tekst nie jest zmienna string, to rzuca AhoCorasickError'''
131     if not isinstance(tekst, (str, unicode)):
132         raise AhoCorasickError("argument is not a string")
133     node = self.n
134     if not returnSet: message = ""
135     else: message = set()
136     dl = len(tekst)
137     for i in range(dl):

```

```

138         while not node.getAim(tekst[i]):
139             node = node.getFail()
140         node = node.getAim(tekst[i])
141         if node.getAccept() != set():
142             zbior = node.getAccept()
143             for j in zbior:
144                 if returnSet:
145                     message.add((i,j))
146                 else:
147                     message += ("Found \""+self.words[j]+"\" in position "+
148                                str(i)+"\n")
149         if not returnSet and message == "":
150             message = "Nothing found\n"
151         if not returnSet: message = message[:len(message)-1]
152         return message

```

3.4 GUI

Działanie interfejsu omówiłam w poprzednim rozdziale. Co robią poszczególne fragmenty kodu, można wywnioskować z komentarzy dokumentujących.

3.4.1 program.py

```

1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  from Tkinter import Tk, Frame, Text, BOTH, W, N, E, S, DISABLED, NORMAL, WORD
5  from ttk import Style, Button, Label, Entry, Scrollbar
6  from ScrolledText import ScrolledText
7  from Tkconstants import END, FIRST
8  import tkMessageBox as MesBox
9  import tkFileDialog as FileDial
10 from ahoCorasick import *
11
12 class Pomoc(Frame):
13     '''okno z pomoca'''
14     def __init__(self, parent):
15         '''tworzy okno pomocy'''
16         Frame.__init__(self, parent)
17         self.parent = parent
18         self.parent.title("Pomoc")
19         self.pack(fill=BOTH, expand=True)
20         self.style = Style()
21         self.style.theme_use("classic")
22         self.columnconfigure(0, weight=1)
23         self.rowconfigure(1, weight=1)
24         self.label = Label(self, text="Treść pomocy")
25         self.label.grid(row=0, column=0, sticky=W)
26         self.tekst = ScrolledText(self, bg="white", wrap=WORD)
27         self.tekst.grid(row=1, column=0, sticky=E+W+N+S)

```

```

28         self.wczytajPomoc()
29     def wczytajPomoc(self):
30         '''wczytuje tresc pomocy z pliku'''
31         read = False
32         opened = False
33         try:
34             plik = open("pomoc.rd", "r")
35             opened = True
36             wiadomosc = plik.read()
37             read = True
38         except Exception as e:
39             mes = str(e)
40             MessageBox.showerror("Błąd", mes)
41         finally:
42             if read:
43                 self.tekst.delete("1.0", "end")
44                 self.tekst.insert("1.0", wiadomosc)
45                 self.tekst.config(state=DISABLED)
46             if opened:
47                 plik.close()
48
49
50     class Ramka(Frame):
51         def __init__(self, parent):
52             '''tworzy obiekt Example dziedziczacy po Frame, rodzicem ma byc parent'''
53             self.listaSlow = [] #lista slow do wyszukania
54             self.buildAho = False #na poczatku nie musimy budowac automatu
55             self.Aho = AhoCorasick()
56             self.highlighted = False
57             #ramka znajduje sie w oknie...
58             #wywołanie konstruktora rodzica - rodzicem jest parent
59             Frame.__init__(self, parent)
60             self.parent = parent
61             #ustawiamy tytuł okna
62             self.parent.title("Angela Czubak, algorytm Aho-Corasick")
63             #niech wypelnia w obu kierunkach, gdy okno rośnie - FILL=BOTH
64             #expand = True - niech zmienia poszerza ramke, gdy okno rośnie
65             self.pack(fill=BOTH, expand=True)
66             #ustawiamy styl
67             self.style = Style()
68             self.style.theme_use("classic")
69             #co robic przy zamykaniu okna
70             self.parent.protocol("WM_DELETE_WINDOW", self.question)
71             self.initUI()
72         def initUI(self):
73             '''zajmuje sie rozkladem poszczegolnych elementow'''
74             self.columnconfigure(3, weight=1)
75             self.rowconfigure(6, weight=1)
76             self.drawMain()
77             self.drawList()

```

```

78         self.drawInput()
79     def drawMain(self):
80         '''rysuje glowne pole tekstowe, etykiety tego pola, klawisze zapisz i
81            otworz'''
82         #pierwsza etykieta
83         self.label0 = Label(self, text="Tekst do przeszukania:")
84         self.label0.grid(row=0, column=0, padx=4, columnspan=2, sticky=W)
85         #glowne pole tekstowe
86         self.tekst = ScrolledText(self, bg="white", wrap=WORD)
87         self.tekst.grid(row=1, column=0, columnspan=4, rowspan=6, padx=4,
88                        sticky=E+W+N+S)
89         self.tekst.tag_configure("highlight", background="yellow",
90                                foreground="brown")
91         self.saveAsButton = Button(self, text="zapisz jako...",
92                                   command=self.fileSave) #klawisz zapisywania
93         self.saveAsButton.grid(row=7, column=0, sticky=W)
94         #klawisz otwierania
95         self.openButton = Button(self, text="otwórz...", command=self.fileOpen)
96         self.openButton.grid(row=7, column=1, sticky=W)
97         self.clearText = Button(self, text="wyłącz podświetlenie",
98                                command=self.unhighlight)
99         self.clearText.grid(row=7, column=2, sticky=W)
100        self.helpButton = Button(self, text="pomoc", command=self.showHelp)
101        self.helpButton.grid(row=7, column=5, sticky=E, padx=12)
102    def drawList(self):
103        '''rysuje etykiety, liste slow do wyszukania oraz klawisz wyszukania
104           i czyszczenia listy'''
105        self.label1 = Label(self, text="Aktualne slowa:") #etykieta z boku
106        self.label1.grid(column=4, row=0, padx=2, sticky=N+W)
107        self.pole = ScrolledText(self, bg="white", height=10, width=35)
108        #pole ze slowami, wysokosc w liczbie znakow
109        self.pole.grid(column=4, row=1, padx=2, columnspan=2)
110        self.pole.config(state=DISABLED)
111        #self.pole.insert(END, "sdfdsdfdsdfg dfgfdgfd fgsdg")
112        self.clear = Button(self, text="wyczyść listę słów",
113                           command=self.clearList)
114        self.clear.grid(column=5, row=2, sticky=W)
115        self.clear.bind('<Return>', self.clearList)
116        self.search = Button(self, text="wyszukaj", command=self.search)
117        self.search.grid(column=4, row=2, sticky=W)
118    def drawInput(self):
119        '''rysuje etykiety, pole wprowadzania i klawisz dodawania slowa'''
120        self.label2 = Label(self, text="Słowo:")
121        self.label2.grid(column=4, row=3, pady=5, padx=2, sticky=W)
122        self.wejscie = Entry(self, width=32)
123        self.wejscie.bind('<Return>', self.addWord)
124        self.wejscie.grid(column=4, row=4, padx=5, columnspan=3, sticky=W)
125        self.add = Button(self, text="dodaj", command=self.addWord)
126        self.add.grid(column=4, row=5, padx=2, sticky=W)
127        self.add.bind('<Return>', self.addWord)

```



```

128     def addWord(self, event=None):
129         '''metoda wywoływana po kliknięciu klawisza <dodaj>'''
130         wartosc = self.wejscie.get()
131         #print type(wartosc)
132         if wartosc != "" and wartosc not in self.listaSlow:
133             if not self.buildAho:
134                 self.buildAho = True
135                 maxlen = self.pole["width"]
136                 improved = wartosc + ", "
137                 self.pole.config(state=NORMAL)
138                 lenNow = len(self.pole.get("1.0", "end"))-1
139                 linesBefore = lenNow/maxlen
140                 signsAfter = lenNow + len(improved)
141                 linesAfter = signsAfter/maxlen
142                 if (signsAfter-1)%maxlen == 0:
143                     self.pole.insert("end", wartosc+",")
144                 else:
145                     if (linesAfter > linesBefore and len(improved) <= maxlen and
146                         signsAfter%maxlen > 0):
147                         for i in range(maxlen - lenNow%maxlen):
148                             self.pole.insert("end", " ")
149                         self.pole.insert("end", improved)
150                         #self.pole.insert("end", improved)
151                         self.pole.config(state=DISABLED)
152                         self.listaSlow.append(wartosc)
153             self.wejscie.delete(0, END)
154     def clearList(self, event=None):
155         '''metoda czyszczaca liste slow'''
156         if self.listaSlow != []:
157             self.pole.config(state=NORMAL)
158             self.pole.delete("1.0", "end")
159             self.pole.config(state=DISABLED)
160             self.listaSlow = []
161             self.Aho = AhoCorasick()
162             self.buildAho = True
163     def question(self):
164         '''metoda rysujaca okienko "czy na pewno chcesz zakonczyc"'''
165         wyn = MesBox.askokcancel("Koniec",
166                                 "Czy na pewno chcesz wyjść z aplikacji?")
167         if wyn: quit()
168         print wyn
169     def unhighlight(self):
170         '''metoda wylaczajaca aktualne podswietlenie wyszukanych wzorców'''
171         if self.highlighted:
172             self.tekst.tag_remove("highlight", "1.0", "end")
173             self.highlight = False
174     def search(self):
175         '''metoda wyszukujaca wzorce'''
176         self.unhighlight()
177         if self.buildAho:

```

```

178         self.Aho.clear()
179         self.Aho.makeTree(self.listaSlow)
180         self.Aho.build()
181         self.buildAho = False
182     tekst = self.tekst.get("1.0", "end")
183     #print type(tekst)
184     res = self.Aho.search(tekst, True)
185     if set(res) != set():
186         #self.tekst.tag_add("highlight", "5.0", "6.0")
187         self.highlighted = True
188         for i in res:
189             end = i[0]+1
190             nr = i[1] #nr slowa
191             start = end - len(self.Aho.words[nr])
192             first = "1.0"+str(start)+"c"
193             last = "1.0"+str(end)+"c"
194             self.tekst.tag_add("highlight", first, last)
195 def fileOpen(self):
196     '''metoda otwierajaca plik'''
197     opened = False
198     read = False
199     rozszerzenia = [('tekstowe', '*.txt'), ('tekstowe', '*.dat'),
200                     ('wszystkie', '*')]
201     try:
202         okno = FileDialog.Open(self, filetypes=rozszerzenia)
203         nazwaPliku = okno.show()
204         #print nazwaPliku
205         if nazwaPliku != '' and nazwaPliku != ():
206             plik = open(nazwaPliku, "r") #do odczytu
207             opened = True
208             tekst = plik.read()
209             read = True
210     except Exception as e:
211         mes = str(e)
212         MesBox.showerror("Błąd", mes)
213     finally:
214         if opened:
215             plik.close()
216         if read:
217             self.highlighted = False
218             self.tekst.tag_remove("highlight", "1.0", "end")
219             self.tekst.delete("1.0", "end")
220             self.tekst.insert("end", tekst)
221 def fileSave(self):
222     '''metoda zapisujaca plik'''
223     rozszerzenia = [('tekstowe', '*.txt'), ('tekstowe', '*.dat'),
224                     ('wszystkie', '*')]
225     opened = False
226     try:
227         okno = FileDialog.SaveAs(filetypes=rozszerzenia, title="Zapisz plik")

```

```

228         nazwaPliku = okno.show()
229         if nazwaPliku != '' and nazwaPliku != ():
230             tresc = self.tekst.get("1.0", "end")
231             plik = open(nazwaPliku, "w")
232             opened = True
233             plik.write(tresc.encode("utf-8"))
234         except Exception as e:
235             mes = str(e)
236             MesBox.showerror("Błąd", mes)
237         finally:
238             if opened: plik.close()
239     def showHelp(self):
240         '''metoda otwierająca okno pomocy'''
241         pomoc = Tk()
242         pomoc.geometry("400x500+150+150")
243         okno = Pomoc(pomoc)
244         pomoc.mainloop()
245
246     def main():
247         root = Tk() #glowne okno aplikacji
248         root.geometry("650x400+100+100")#wymiarzy=650x400, pozycja = (100,100)
249         app = Ramka(root)
250         root.mainloop()
251
252     if __name__ == '__main__':
253         main()

```

4 Testy

Dosyć dokładnie przetestowano moduły *myList*, *node* oraz *ahoCorasick*. Omówienie i sam kod zamieszczono niżej. Wszystkie testy kończą się pozytywnie.

4.1 Test modułu myList

4.1.1 Opis

Testy:

- *test_init* - czy konstruktor tworzy węzeł ze spodziewanymi parametrami początkowymi
- *test_Element* - test sprawdzający tworzenie obiektów klasy *Element* oraz ustawianie zawartości elementu i następnego elementu, a także rzucanie wyjątków przy próbie ustawienia następnego elementu na coś nie będącego klasy *Element*
- *test_add* - sprawdza działanie dodawania pojedynczego elementu do listy *MyList* - czy ostatnie element jest tym, który przed chwilą dodaliśmy

- *test_contains* - sprawdza, czy elementy, które do tej pory dodawaliśmy do listy, są na niej znajdowane, a także, czy nie znajdujemy elementów, których tam nie dodaliśmy.
- *test_equal* - sprawdza, czy obiekty *MyList*, którym dodawaliśmy te same elementy w tej samej kolejności są uważane za równe; bada także różnicę tych obiektów, gdy mają inne elementy
- *test_join* - sprawdza, czy rzeczywiście dokonuje się dodawanie dwóch obiektów *MyList* oraz, że można dodać obiekt *MyList* do pustego obiektu *MyList*

4.1.2 myListTest.py

```

1  #!/usr/bin/python
2   -*- coding: utf-8 -*-
3
4  from myList import *
5
6  import unittest
7
8  class TestMyList(unittest.TestCase):
9      '''kod testujący klasę MyList'''
10     def setUp(self):
11         pass
12     def test_init(self):
13         '''sprawdzam działanie konstruktora'''
14         l = MyList()
15         self.assertEqual(l.first, None)
16         self.assertEqual(l.last, None)
17         self.assertEqual(l.current, None)
18         self.assertEqual(len(l), 0)
19     def test_Element(self):
20         '''kod testujący tworzenie Elementu'''
21         n = Element()
22         n2 = Element(3)
23         n3 = Element(4, n2)
24         self.assertEqual(n2.getData(), 3)
25         self.assertEqual(n3.getNext(), n2)
26         self.assertRaises(MyListError, Element, 3, 4)
27         self.assertRaises(MyListError, n2.setNext, "3")
28         n.setData(5)
29         self.assertEqual(n.getData(), 5)
30         n.setNext(n3)
31         self.assertEqual(n.getNext(), n3)
32     def test_add(self):
33         '''kod testujący dodawanie elementu do listy'''
34         l = MyList()
35         l.add(4)
36         self.assertEqual(l.first.getData(), 4)
37         self.assertEqual(l.last.getData(), 4)

```

```

38         l.add(8)
39         self.assertEqual(l.last.getData(), 8)
40         l.add(7)
41         self.assertEqual(l.last.getData(), 7)
42     def test_contains(self):
43         '''kod testujacy sprawdzanie, czy element nalezy do listy'''
44         l = MyList()
45         self.assertFalse(4 in l)
46         self.assertFalse(8 in l)
47         self.assertFalse(7 in l)
48         self.assertFalse(13 in l)
49         l.add(4)
50         l.add(8)
51         l.add(7)
52         self.assertTrue(4 in l)
53         self.assertTrue(8 in l)
54         self.assertTrue(7 in l)
55         self.assertFalse(13 in l)
56     def test_equal(self):
57         '''kod testujacy porownywanie list'''
58         l1 = MyList()
59         l2 = MyList()
60         self.assertEqual(l1, l2)
61         l1.add(3)
62         self.assertNotEqual(l1, l2)
63         l1.add(4)
64         l1.add(9)
65         l2.add(3)
66         l2.add(4)
67         l2.add(9)
68         self.assertEqual(l1, l2)
69         l2.add(7)
70         self.assertNotEqual(l1, l2)
71     def test_join(self):
72         '''kod testujacy laczenie list'''
73         l1 = MyList()
74         l2 = MyList()
75         l3 = MyList()
76         l1.add(2)
77         l2.add(5)
78         l2.add(4)
79         l3.add(2)
80         l3.add(5)
81         l3.add(4)
82         l1 += l2
83         self.assertEqual(l1, l3)
84         l4 = MyList()
85         l4 += l3
86         self.assertEqual(l4, l3)
87     def tearDown(self):

```

```

88         pass
89
90     if __name__ == "__main__":
91         unittest.main()

```

4.2 Test modułu node

4.2.1 Opis

Testy:

- *test_init* - czy konstruktor tworzy węzeł ze spodziewanymi parametrami początkowymi
- *test_accept* - test sprawdzający działanie metod *Node.setAccept*, *Node.getAccept* - czy rzeczywiście dodawane są odpowiednie indeksy, czy jest rzucony *NodeError*, gdy indeks niepoprawny. Możliwe jest także dodawanie zbiorów indeksów, badamy więc, czy funkcja rzuca wyjątek, gdy ten zbiór niepoprawny
- *test_aim* - sprawdza, czy metody *Node.getAim* i *Node.setAim* rzucają wyjątek *NodeError*, jeśli argumenty funkcji są niepoprawne, sprawdza, czy *Node.setAim(self, label, node)* rzeczywiście ustawia krawędź etykietowaną *label* na stan/węzeł *node*. Ponadto, domyślnie węzeł jest korzeniem - dla dowolnej poprawnej etykiety *Node.getAim* powinno zwrócić ten sam korzeń. W późniejszym kodzie węzeł nie będący korzeniem będzie miał atrybut *fail* ustawiony (domyślnie jest *None*) - wtedy wywołanie *getAim(b)*, kiedy nie ma krawędzi etykietowanej *b* powinno zwrócić *None*
- *test_fail* - sprawdza, czy metoda *Node.setFail* rzuca wyjątek *NodeError*, jeśli argument nie jest obiektem klasy *Node*, sprawdza, czy *Node.setFail(self, node)* rzeczywiście ustawia krawędź porażki na stan/węzeł *node*. Ponadto, domyślnie węzeł ma tę krawędź ustawioną na *None* - sprawdzamy, czy tak rzeczywiście jest.

4.2.2 nodeTest.py

```

1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  from node import *
5
6  import unittest
7
8  class TestNode(unittest.TestCase):
9      '''kod testujący klasę Node'''
10     def setUp(self):
11         pass
12     def test_init(self):
13         '''sprawdzam działanie konstruktora'''
14         n = Node()
15         self.assertEqual(n.accept, MyList())

```

```

16         self.assertEqual(n.fail, None)
17         self.assertEqual(n.edges, {})
18     def test_accept(self):
19         '''kod testujacy metody setAccept i getAccept'''
20         n = Node()
21         n.setAccept(4)
22         n.setAccept(8)
23         s = n.getAccept()
24         self.assertTrue(4 in s)
25         self.assertTrue(8 in s)
26         self.assertFalse(7 in s)
27         self.assertRaises(NodeError, n.setAccept, 6.0)
28         self.assertRaises(NodeError, n.setAccept, -7)
29         mySet = MyList()
30         mySet.add(5)
31         n.setAccept(mySet)
32         secondSet = MyList()
33         secondSet.add(4)
34         secondSet.add(8)
35         secondSet.add(5)
36         result = n.getAccept()
37         self.assertEqual(result, secondSet)
38         mySet = MyList()
39         mySet.add(5)
40         mySet.add(8.0)
41         self.assertRaises(NodeError, n.setAccept, mySet)
42         mySet = MyList()
43         mySet.add(5)
44         mySet.add(-8)
45         self.assertRaises(NodeError, n.setAccept, mySet)
46     def test_aim(self):
47         '''kod testujacy dzialanie metod getAim i setAim'''
48         n = Node()
49         self.assertRaises(NodeError, n.getAim, 7)
50         self.assertRaises(NodeError, n.getAim, "")
51         self.assertRaises(NodeError, n.getAim, "df")
52         #domyslnie kazdy wezel jest korzeniem jakiegos drzewa
53         #powinien dla kazdej 'litera' zwracac lacze na siebie,
54         #o ile nie ustanowiono inaczej
55         self.assertEqual(n.getAim("a"), n)
56         self.assertEqual(n.getAim("e"), n)
57         n2 = Node()
58         n.setAim("e", n2)
59         self.assertEqual(n.getAim("a"), n)
60         self.assertEqual(n.getAim("e"), n2)
61         #n2 nie jest korzeniem - dlatego ustawiam jego fail na
62         #inny wezel, np. n
63         n2.fail = n
64         self.assertEqual(n2.getAim("a"), None)
65         self.assertRaises(NodeError, n.setAim, 7, n2)

```

```

66         self.assertRaises(NodeError, n.setAim, "", n2)
67         self.assertRaises(NodeError, n.setAim, "df", n2)
68         self.assertRaises(NodeError, n.setAim, "a", "gfg")
69     def test_fail(self):
70         '''kod testujacy dzialanie metod getFail i setFail'''
71         n1 = Node()
72         n2 = Node()
73         self.assertEqual(n1.getFail(), None)
74         self.assertEqual(n2.getFail(), None)
75         n2.setFail(n1)
76         self.assertEqual(n2.getFail(), n1)
77         self.assertRaises(NodeError, n1.setFail, "gfgf")
78     def tearDown(self):
79         pass
80
81 if __name__ == "__main__":
82     unittest.main()

```

4.3 Test modułu ahoCorasick

4.3.1 Opis

Testy:

- *test_init* - czy konstruktor tworzy korzeń ze spodziewanymi parametrami początkowymi, czy początkowa lista słów jest pusta
- *test_tree* - test sprawdzający działanie metod *AhoCorasick.addWord(self, word)* i *AhoCorasick.lookUp(self, word)* - czy rzeczywiście dodawane są odpowiednie słowa, czy nie znajdujemy słów, których nie ma w drzewie, czy następuje rzucanie wyjątku *AhoCorasickError*, gdy *word* nie jest stringiem
- *test_build* - test sprawdzający działanie metody *AhoCorasick.build(self)* - czy zbudowany automat jest poprawny, jak na rysunku (3) automat 1
- *test_build2* - test sprawdzający działanie metody *AhoCorasick.build(self)* - czy zbudowany automat jest poprawny dla większej liczby słów, większej różnorodności przejść, jak na rysunku (4) automat 2
- *test_unicode* - kod pokazujący, że dodawanie słów raz jako *str* a raz jako *unicode* działa poprawnie - "q" != u"q" etc., długości tych zmiennych przy sprawdzeniu funkcją *len* są różne, a jednak dzięki dekodowaniu *str* do *unicode* wszystko działa
- *test_clear* - kod testujący czyszczenie automatu
- *test_search* - kod testujący wyszukiwanie wzorców w tekście - wzorców "he", "she", "his", "hers" w tekście "ushers", powyższych wzorców w tekście "w tym tekście nic nie znajdzie" oraz w tekście ""

4.3.2 ahoTest.py

```
1  #!/usr/bin/python
2   -*- coding: utf-8 -*-
3
4  from ahoCorasick import *
5  import unittest
6
7  class TestAho(unittest.TestCase):
8      '''kod testujacy klase AhoCorasick'''
9      def setUp(self):
10         pass
11     def test_init(self):
12         '''sprawdzam dzialanie konstruktora'''
13         a = AhoCorasick()
14         #porownianie korzenia z domyslnym wezlem + czy lista slow pusta
15         self.assertEqual(a.n.getAccept(), MyList())
16         self.assertEqual(a.n.getFail(), None)
17         self.assertEqual(a.n.getLabels(), [])
18         self.assertEqual(a.words, [])
19     def test_tree(self):
20         '''sprawdzam dodawanie i wyszukiwanie slow w drzewie'''
21         a = AhoCorasick()
22         self.assertFalse(a.lookup("nie ma"))
23         self.assertFalse(a.lookup("tak"))
24         self.assertFalse(a.lookup("ta"))
25         a.addWord("tak")
26         self.assertFalse(a.lookup("nie ma"))
27         self.assertTrue(a.lookup("tak"))
28         self.assertFalse(a.lookup("ta"))
29         a.addWord("ta")
30         self.assertFalse(a.lookup("nie ma"))
31         self.assertTrue(a.lookup("tak"))
32         self.assertTrue(a.lookup("ta"))
33         #testowanie wyjatkow
34         self.assertRaises(AhoCorasickError, a.addWord, 7)
35         self.assertRaises(AhoCorasickError, a.lookup, 7)
36     def test_build(self):
37         '''kod testujacy metode AhoCorasick.build'''
38         a = AhoCorasick()
39         a.addWord("sernik")
40         a.addWord("laser")
41         a.build()
42         r = a.n
43         l = r.getAim("l")
44         la = l.getAim("a")
45         las = la.getAim("s")
46         lase = las.getAim("e")
47         laser = lase.getAim("r")
48         s = r.getAim("s")
```

```

49     se = s.getAim("e")
50     ser = se.getAim("r")
51     sern = ser.getAim("n")
52     serni = sern.getAim("i")
53     sernik = serni.getAim("k")
54     self.assertEqual(l.getFail(), r)
55     self.assertEqual(la.getFail(), r)
56     self.assertEqual(las.getFail(), s)
57     self.assertEqual(lase.getFail(), se)
58     self.assertEqual(laser.getFail(), ser)
59     self.assertEqual(l.getAccept(), MyList())
60     self.assertEqual(la.getAccept(), MyList())
61     self.assertEqual(las.getAccept(), MyList())
62     self.assertEqual(lase.getAccept(), MyList())
63     zbior = MyList()
64     zbior.add(1)
65     self.assertEqual(laser.getAccept(), zbior)
66     self.assertEqual(a.words[1], "laser")
67     self.assertEqual(s.getFail(), r)
68     self.assertEqual(se.getFail(), r)
69     self.assertEqual(ser.getFail(), r)
70     self.assertEqual(sern.getFail(), r)
71     self.assertEqual(serni.getFail(), r)
72     self.assertEqual(sernik.getFail(), r)
73     self.assertEqual(s.getAccept(), MyList())
74     self.assertEqual(se.getAccept(), MyList())
75     self.assertEqual(ser.getAccept(), MyList())
76     self.assertEqual(sern.getAccept(), MyList())
77     self.assertEqual(serni.getAccept(), MyList())
78     zbior = MyList()
79     zbior.add(0)
80     self.assertEqual(sernik.getAccept(), zbior)
81     self.assertEqual(a.words[0], "sernik")
82 def test_build2(self):
83     '''kod testujacy metode AhoCorasick.build dla bardziej skomplikowanego
84        automatu'''
85     a = AhoCorasick()
86     a.addWord("he")
87     a.addWord("she")
88     a.addWord("his")
89     a.addWord("hers")
90     a.build()
91     r = a.n
92     h = r.getAim("h")
93     hi = h.getAim("i")
94     his = hi.getAim("s")
95     he = h.getAim("e")
96     her = he.getAim("r")
97     hers = her.getAim("s")
98     s = r.getAim("s")

```

```

99     sh = s.getAim("h")
100     she = sh.getAim("e")
101     self.assertEqual(s.getFail(), r)
102     self.assertEqual(s.getAccept(), MyList())
103     self.assertEqual(sh.getFail(), h)
104     self.assertEqual(sh.getAccept(), MyList())
105     zbior = MyList(); zbior.add(0); zbior.add(1)
106     self.assertEqual(she.getFail(), he)
107     self.assertEqual(set(she.getAccept()), set(zbior))
108     self.assertEqual(a.words[1], "she")
109     self.assertEqual(a.words[0], "he")
110     self.assertEqual(h.getFail(), r)
111     self.assertEqual(set(h.getAccept()), set())
112     self.assertEqual(he.getFail(), r)
113     self.assertEqual(set(he.getAccept()), set([0]))
114     self.assertEqual(her.getFail(), r)
115     self.assertEqual(set(her.getAccept()), set())
116     self.assertEqual(hers.getFail(), s)
117     self.assertEqual(set(hers.getAccept()), set([3]))
118     self.assertEqual(a.words[3], "hers")
119     self.assertEqual(hi.getFail(), r)
120     self.assertEqual(set(hi.getAccept()), set())
121     self.assertEqual(his.getFail(), s)
122     self.assertEqual(set(his.getAccept()), set([2]))
123     self.assertEqual(a.words[2], "his")
124     def test_unicode(self):
125         '''kod pokazujący, że dodawanie słów w postaci str jak i unicode
126         działa poprawnie'''
127         a = AhoCorasick()
128         a.addWord("ąćęłńóśźż")
129         a.addWord(u"ąćęłńóśźż")
130         self.assertTrue(a.lookup("ąćęłńóśźż"))
131         self.assertTrue(a.lookup(u"ąćęłńóśźż"))
132         self.assertEqual(len(a.words), 1)
133     def test_clear(self):
134         '''kod testujący czyszczenie automatu'''
135         a = AhoCorasick()
136         a.makeTree(["laser", "sernik"])
137         a.clear()
138         self.assertEqual(a.words, [])
139         self.assertEqual(set(a.n.getAccept()), set())
140         self.assertEqual(a.n.getFail(), None)
141         self.assertEqual(a.n.getLabels(), [])
142     def test_search(self):
143         '''test sprawdzający wyszukiwanie wzorców w tekście'''
144         a = AhoCorasick()
145         a.makeTree(["he", "she", "his", "hers"])
146         res = a.search("ushers")
147         e = ("Found \"she\" in position 3\nFound \"he\" in position 3\n"
148             "Found \"hers\" in position 5")

```

```

149         self.assertEqual(res,e)
150         res = a.search("")
151         self.assertEqual(res, "Nothing found")
152         res = a.search("w tym tekście nic nie znajdzie")
153         self.assertEqual(res, "Nothing found")
154         res = a.search("ushers", True)
155         #pierwsza czesc krotki - pozycja, druga - indeks slowa
156         self.assertEqual(res,set([(3,0),(3,1),(5,3)]))
157         res = a.search("w tym tekście nic nie znajdzie",True)
158         self.assertEqual(res,set())
159         res = a.search("",True)
160         self.assertEqual(res,set())
161         self.assertRaises(AhoCorasickError, a.search, 7)
162     def tearDown(self):
163         pass
164
165 if __name__ == "__main__":
166     unittest.main()

```

5 Podsumowanie

Wyniki testów wskazują, że program działa w porządku. Interfejs graficzny wydaje się być dosyć czytelny i dobrze oddaje wyniki działania algorytmu Aho-Corasick.

Jednak należy uważać przy stosowaniu różnych typów znakowych - Python przy porównaniu znaków ze zmiennych typu *str* i *unicode* ostrzega:

UnicodeWarning: Unicode equal comparison failed to convert both arguments to Unicode - interpreting them as being unequal

Dlatego, przy dodawaniu słów dopisałam kawałek kodu, który dekoduje zmienne *str* do *unicode*.

Ponadto, początkowo korzystano z *set* jako pojemnika na indeksy słów akceptowanych przez automat. Ponieważ jednak, jak napisałam we wprowadzeniu, w rozważanym przez nas przypadku zbiory takich indeksów będą rozłączne, dopisałam własną implementację szybko złączalnych list. Standardowo Python dokłada drugą listę do pierwszej w $O(k)$, gdzie k - długość drugiej listy (<https://wiki.python.org/moin/TimeComplexity>), więc było to wskazane.

6 Bibliografia

- http://pl.wikipedia.org/wiki/Algorytm_Aho-Corasick
- <http://www.cs.uku.fi/~kilpelai/BSA05/lectures/slides04.pdf>
- http://pl.wikipedia.org/wiki/Deterministyczny_automat_skończony
- <http://pl.wikipedia.org/wiki/Pods\OT4\lowo>
- <https://wiki.python.org/moin/TimeComplexity>