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 $\mathcal{P} = \{P_0, \ldots, P_k\}$ 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

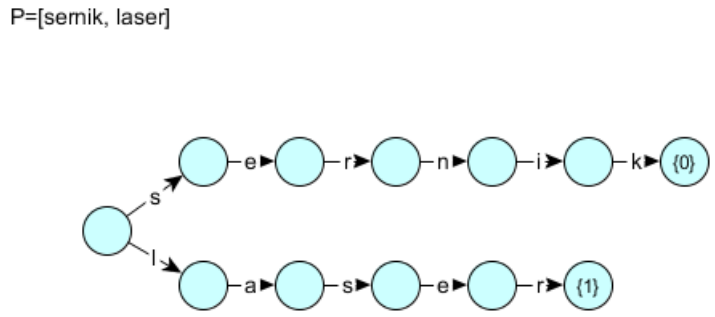
Definicja 1. ***Drzewem trie*** *dla zbioru wzorców $\mathcal{P}$ nazywamy takie ukorzenione drzewo $\mathcal{K}$, że:*

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

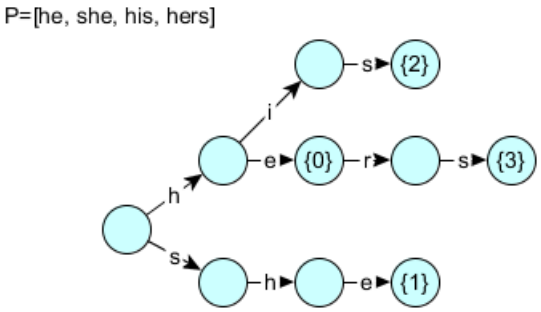
   Definicja 2. ***Etykietą węzła*** *v nazywamy konkatenację etykiet krawędzi znajdujących się na ścieżce z korzenia do v. Oznaczamy ją jako $\mathcal{L}(v)$.*

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.



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



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

1.3 Konstrukcja drzewa trie

Jak budować drzewo trie dla $\mathcal{P} = \{P_0, \ldots, P_k\}$? 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 $O(|P_0| + \cdots + |P_k|) = O(n)$.

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 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 tesktu, n - suma długości wzorców.
By przyspieszyć wyszukiwanie wzorców, rozszerzamy drzewo trie do **automatu**.

1.5 Automat

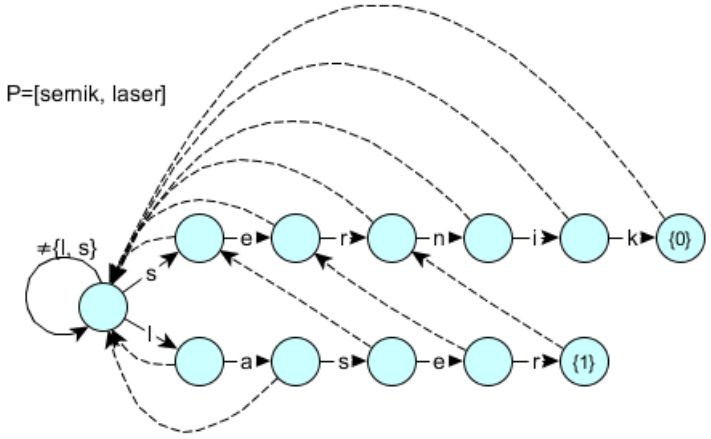
Definicja 3. ***Automatem deterministycznym*** *nazywamy piątkę uporządkowaną* $(\Sigma, Q, q, \delta, F)$*, gdzie*

1. Σ *- skończony alfabet*
2. Q *- skończony zbiór stanów*
3. $q \in Q$*- stan początkowy*
4. $\delta : Q \times \Sigma \to Q$ *- 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 pojeć, nie będzie ściśle deterministyczny. Automat będziemy budować na podstawie drzewa trie, zatem uściślam: zbiór stanów będą stanowiły węzły drzewa, a stanem początkowym będzie korzeń, do którego będę 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 $\mathcal{P} = \{sernik, laser\}$. Przerywaną linią oznaczono krawędzie odpowiadające funkcji $f(q)$

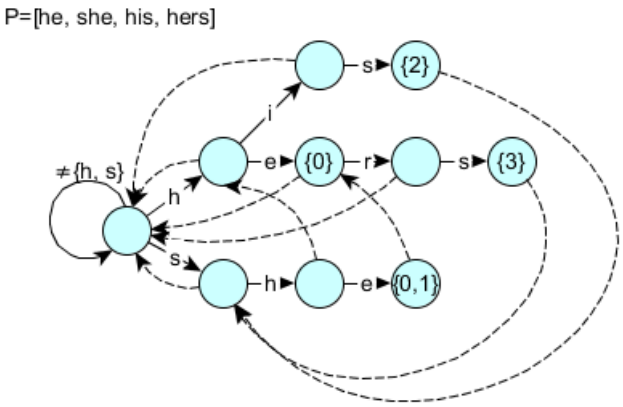
3. Funkcja **wyjścia**, oznaczana jako $out(v)$, zwraca indeksy wzorców, do których znajdujemy dopasowanie w stanie q.

1.6 Przeszukiwanie tekstu

Załóżmy, że mamy do dyspozycji gotowy automat oraz tekst $T[1 \cdots]$, w którym szukamy wzorców. Procedura ta prezentuje się w następujący sposób:
Algorithm 1.1: SEARCH($T[0 \ldots m-1]$)

$q \leftarrow 0$
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):



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 wykonań 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*)

$Q \leftarrow$ QUEUE()
for $a \in \Sigma$
do $\begin{cases} \textbf{if } q \leftarrow g(0,a) \neq 0 \\ \quad \textbf{then } \begin{cases} f(q) \leftarrow 0 \\ \text{Q.ENQUEUE}(q) \end{cases} \end{cases}$
while !Q.ISEMPTY()
do $\begin{cases} r \leftarrow \text{Q.DEQUEUE}() \\ \textbf{for } a \in \Sigma \\ \quad \textbf{do if } u \leftarrow g(r,a) \neq \emptyset \\ \qquad \textbf{then } \begin{cases} \text{Q.ENQUEUE}(u) \\ v \leftarrow f(r) \\ \textbf{while } g(v,a) = \emptyset \\ \quad \textbf{do } v \leftarrow f(v) //(*) \\ f(u) \leftarrow g(v,a) \\ out(u) \leftarrow out(u) \cup out(f(u)) //(**) \end{cases} \end{cases}$

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 sufiks $\mathcal{L}(u)$. Z tego wynika, że powinniśmy spróbować dopasować $f(u) = g(f(r), a)$, bo $\mathcal{L}(f(u))$ może być sufiksem $L(g(f(r), a))$, 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 rozmiary drzewa - tj. $O(n)$. A jak określić, ile razy wykonamy linię (*)?
Rozważmy ścieżkę złożoną z wierzchołków $u_1, \ldots, u_l$, która jest tworzona podczas dodawania wzorca $a_1 \ldots a_l$. Oznaczmy dodatkowo $df(u)$ jako głębokość w drzewie wierzchołka $f(u)$, zatem $df(u_1), \ldots, fd(u_l)$ to ciąg głębokości dla wierzchołków z rozważanej ścieżki, wszystkie są $\geqslant 0$.
Zauważmy ponadto, że głębokość kolejnego wierzchołka może wzrosnąć co najwyżej o 1, czyli $df(u_{i+1}) \leqslant df(u_i) + 1$, zatem wartości df wzrastają sumarycznie co najwyżej o l podczas przechodzenia tej ścieżki.
Kiedy wyliczamy położenie $f(u_{i+1})$, każde wywołanie linii (*) przybliża 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 (*) zostnie wykonana $\leqslant l$ 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 $\mathcal{L}(u)$. Jakiekolwiek 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 wszytskich stanów dla wszytskich 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*)

$Q \leftarrow$ QUEUE()
for $a \in \Sigma$
do { **if** $q \leftarrow g(0,a) \neq 0$
  then Q.ENQUEUE(q)
  $next(0,a) \leftarrow g(0,a)$
while !Q.ISEMPTY()
do { $r \leftarrow$ Q.DEQUEUE()
  for $a \in \Sigma$
    do if $u \leftarrow g(r,a) = \emptyset$
      then { Q.ENQUEUE(u)
        $v \leftarrow f(r)$
        while $g(v,a) = \emptyset$
          do $v \leftarrow f(v)$
        $next(r,a) \leftarrow g(v,a)$
      else { $next(r,a) \leftarrow g(r,a)$
        Q.ENQUEUE($g(r,a)$)

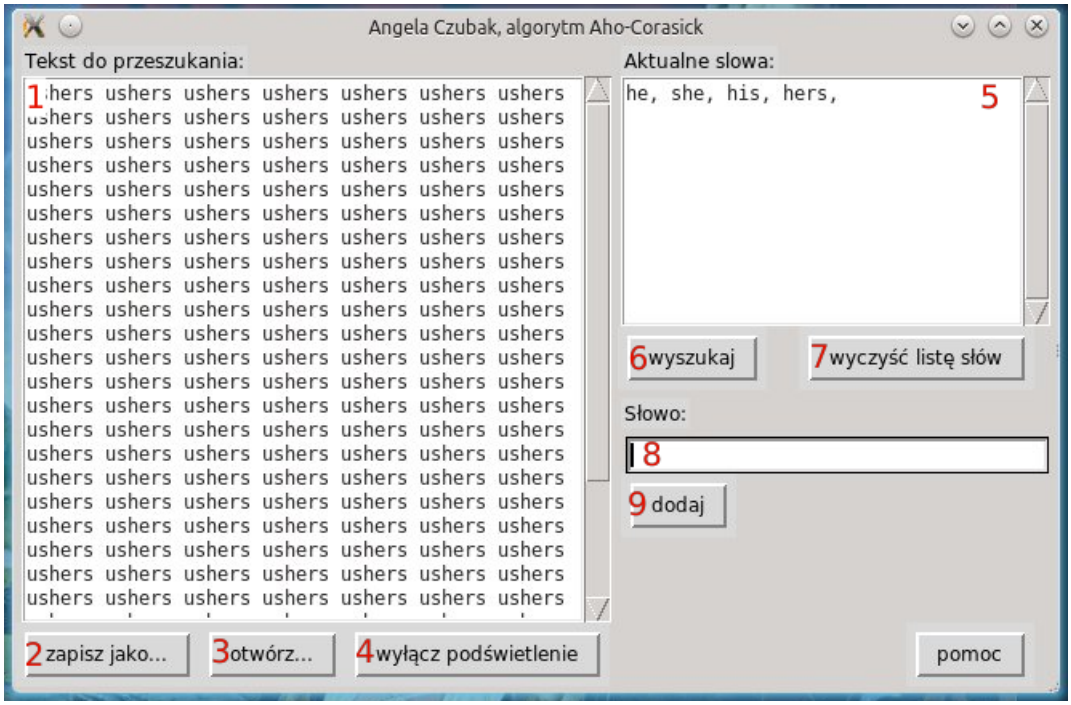
Procedurę tę wykonujemy po wyliczeniu funkcji $f(u)$. Wtedy wyszukiwanie upraszcza się do:
Algorithm 1.4: SEARCHDETERMINISTIC($T[0 \ldots m-1]$)

$q \leftarrow 0$
for $i \leftarrow 0$ **to** $m-1$
do { $q \leftarrow next(q, T[i])$
  comment: przejście jest deterministyczne
  if $out(q) \neq \emptyset$
    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

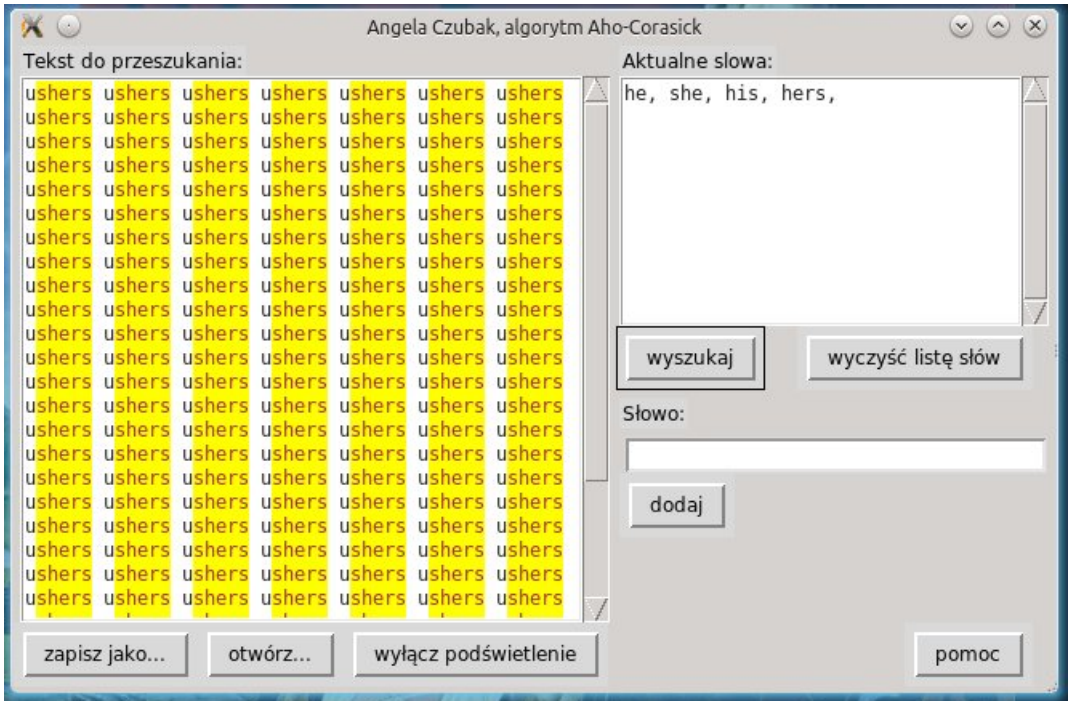
   Jest to pole, w którym umieszczamy tekst, w którym będziemy wyszukiwać wzorce. Możemy tam wprowadzać tekst wprost z klawiatury lub wczytać tekst z pliku. W tym drugim przypadku należy kliknąć przycisk `otwórz...`, a następnie wybrać plik z użyciem okna dialogowego. Tekst, który wprowadzimy do tego pola 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 wyszukania 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 wzorzec, 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 **pola**:

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 **pola**:

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. *__add(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 porownujaca 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 laczyc 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 __add__(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)* - funkjca pomocnicza sprawdzająca, czy etykieta label spełnia warunki etykiety, rzuca wyjątkiem *NodeError* wpp

3. *nodeCorrect(self, node)* - funkjca 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

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from myList import *

class NodeError(Exception):
    '''wyjatek dla klasy Node'''
    def __init__(self, value):
        '''konstruktor, argumentem tresc przy rzucaniu wyjatku'''
        self.napis = value
    def __str__(self):
        '''podaje powod wyjatku'''
        return repr(self.napis)
    __repr__ = __str__

class Node:
    '''klasa opisujaca wezel/stan w automacie/drzewie Trie'''
    def __init__(self):
        '''konstruktor bezargumentowy
           accept = MyList() - pusta lista,
           fail = None
           edges = {}'''
        self.accept = MyList()
        self.edges = {}
        self.fail = None
    def labelCorrect(self, label):
        '''sprawdza, czy label jest poprawna etykieta krawedzi
           jesli nie, rzuca NodeError'''
        if not isinstance(label, (str, unicode)):
            raise NodeError("label must be a character")
        if len(label) != 1:
            raise NodeError("label must be exactly one character long")
    def nodeCorrect(self, node, n=1):
        '''sprawdza, czy node jest obiektem klasy Node
           jesli nie, to rzuca NodeError
           n to numer argumentu, ktorym jest node w jakiejs funkcji
             sluzy uszczegolowieniu, ktory argument jest bledny'''
        if not isinstance(node, Node):
            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             slowa, 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            dopsowac 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 **pola**:

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 slowo 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 strigiem
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* mowi 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 ktorej znalazła słowo, oraz indeks słowa; jeśli tekst nie jest zmienną string, to rzuca *AhoCorasickError*

3.3.3 ahoCorasick.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from node import *
from Queue import *

class AhoCorasickError(Exception):
    '''wyjatek dla klasy Node'''
    def __init__(self, value):
        '''konstruktor, argumentem tresc przy rzucaniu wyjatku'''
        self.napis = value
    def __str__(self):
        '''podaj powod wyjatku'''
        return repr(self.napis)

class AhoCorasick:
    '''klasa opisujaca drzewo Trie / automat, sluzacy wyszukiwaniu wzorcow'''
    def __init__(self):
        '''konstruktor bezargumentowy
           n - korzen drzewa, pusty
           words - pusta liczba slow zakodowanych w drzewie'''
        self.n = Node()
        self.words = []
        self.built = False
    def addWord(self, word):
        '''dodaje slowo word do automatu/drzewa
           rzuca AhoCorasickError, jesli word nie jest stringiem
             lub zbudowano juz automat'''
        if self.built:
            raise AhoCorasickError("automaton has been built already")
        if not isinstance(word, (str, unicode)):
            raise AhoCorasickError("argument is not a string")
        dl = len(word)
        if dl == 0: return #nie dodajemy pustego slowa
        wezel = self.n
        i = 0
        #idziemy dopoki mozemy po istniejacych wezlach
```

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

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

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

3.4 GUI

Działanie interfejsu omówiłam w poprzednim rozdziale. Działanie poszczególnych fragmentów 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              MesBox.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          #wywolanie konstruktora rodzica - rodzicem jest parent
59          Frame.__init__(self, parent)
60          self.parent = parent
61          #ustawiamy tytul okna
62          self.parent.title("Angela Czubak, algorytm Aho-Corasick")
63          #niech wypelnia w obu kierunkach, gdy okno rosnie - FILL=BOTH
64          #expand = True - niech zmienia poszerza ramke, gdy okno rosnie
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, etykiete 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 etykiete, 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,"sdfdfsdfsdfg 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 etykiete, 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 wywolywana po klikniecciu 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 wzorcow'''
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 = FileDial.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 = FileDial.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 otwierajaca 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")#wymiary=650x400, pozycja = (100,100)
249     app = Ramka(root)
250     root.mainloop()
251 
252 if __name__ == '__main__':
253     main()
```

4 Testy

5 Podsumowanie

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