Sveučilište u Rijeci
Fakultet informatike i digitalnih tehnologija

Projektna dokumentacija

Algoritam rastavljanja slogova za Njemačko jezično područje

Antonio Janach

Rijeka, siječanj 2023.

Sadržaj

1. Uvod 1

2. Algoritam 2

2.1. Karakteristike algoritma 2

2.2. Dizajn algoritma 3

3. Lingvistička pravila za njemački jezik 4

3.1. Pretprocesiranje 4

3.2. Abeceda, samoglasnici i suglasnici njemačkog jezika 5

3.3. Razrada algoritma na temelju lingvističkih pravila 5

3.3.1. Rastavljanje složenica 5

3.3.2. Odvojiti prefikse 6

3.3.3. Uzorci za rastavljanje 6

3.3.4. Raspis pravila za rastavljanje st, tz, i pf u sredini riječi 6

3.3.5. Iznimke 7

4. Programski kôd algoritma 8

4.1. Korišteni moduli 8

4.1.1. Modul re – regularni izrazi 8

4.1.2. Modul string 9

4.1.3. Modul compound-split 9

4.2. Opis glavnih dijelova algoritma 9

4.2.1. Globalne liste i globalne varijable 9

4.2.2. Funkcija syllabels_rules_execeptions 10

4.2.3. Funkcija sentence_in_vc 14

4.2.4. Funkcija syllables_rules 15

5. Osnovne upute za korištenje algoritma 19

6. Izrada web aplikacije i korisničkog sučelja 21
6.1. Osnovne upute za korištenje web sučelja 21
7. Testiranje algoritma 25
7.1. Skup testnih riječi 25
7.2. Programska skripta za računanje mjere točnosti 26
8. Rezultati i analiza uspješnosti programske skripte 30
Zaključak 32
Popis slika 33
Popis kôdova 34
Popis naredba 35
Literatura 36
Prilog: programski kod 37

1. Uvod

U ovoj projektnoj dokumentaciji bit će objašnjen i prikazan proces izrade algoritma koji omogućava rastavljanje riječi na slogove, a odnosi se na njemačko jezično područje. Odnosno, bit će dokumentirano specifično tacitno i eksplicitno lingvističko znanje te njegova pretvorba i formalizacija za potrebe kreiranja algoritma silabifikacije.

Konačan cilj projekta je stvaranje algoritma i aplikacije koji bilo koju riječ na njemačkom jeziku zna rastaviti na slogove. Točnije, za bilo koji tekst na ulazu, generira isti taj tekst na izlazu, ali u obliku rastavljenom na slogove dodajući znak razmaka između susjednih slogova.

Primjer za hrvatski tekst na ulazu:

„Analiza slogova kao osnovnih elemenata jezika važna je za različite postupke u domeni računalne analize prirodnoga jezika i govornih tehnologija.“

Rezultat na izlazu:

„A na li za slo go va ka o os nov nih e le me na ta je zi ka va žna je za ra zli či te po stup ke u do me ni ra ču nal ne a na li ze pri ro dno ga je zi ka i go vor nih teh no lo gi ja.“

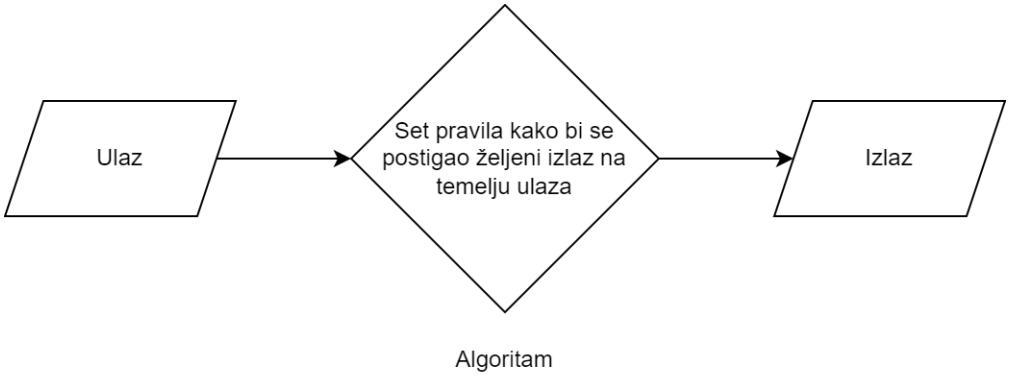
Na temelju prikupljenih eksplicitnih i tacitnih znanja koje su prikupili, definirali i zapisali lingvisti za njemački jezik, cilj je formalno zapisati ta ista pravila koristeći programski jezik Python.

Na osnovu zapisanih pravila potrebno je kreirati algoritam za rastavljanje riječi na slogove. Postupak rastavljanja na slogove nije uvijek trivijalan ni jednoznačan. Zbog toga je potrebno poštovati pravila njemačkog jezika.

2. Algoritam

U ovome poglavlju bit će objašnjeno što je to algoritam. Algoritam je skup konačnih pravila ili uputa koje treba slijediti u izračunima ili drugim operacijama rješavanja problema, ili pak postupak za rješavanje matematičkog problema u konačnom broju koraka koji često uključuje rekurzivne operacije.

Prema tome algoritam se odnosi na niz konačnih koraka za rješavanje određenog problema.



Slika 1: algoritam prikazan na slikovni način

Svaki puta kada koristimo telefon, računalo ili kalkulator primjenjujemo algoritme. Slično tome, algoritmi pomažu obaviti zadatak u programiranju kako bi se dobio očekivani rezultat.

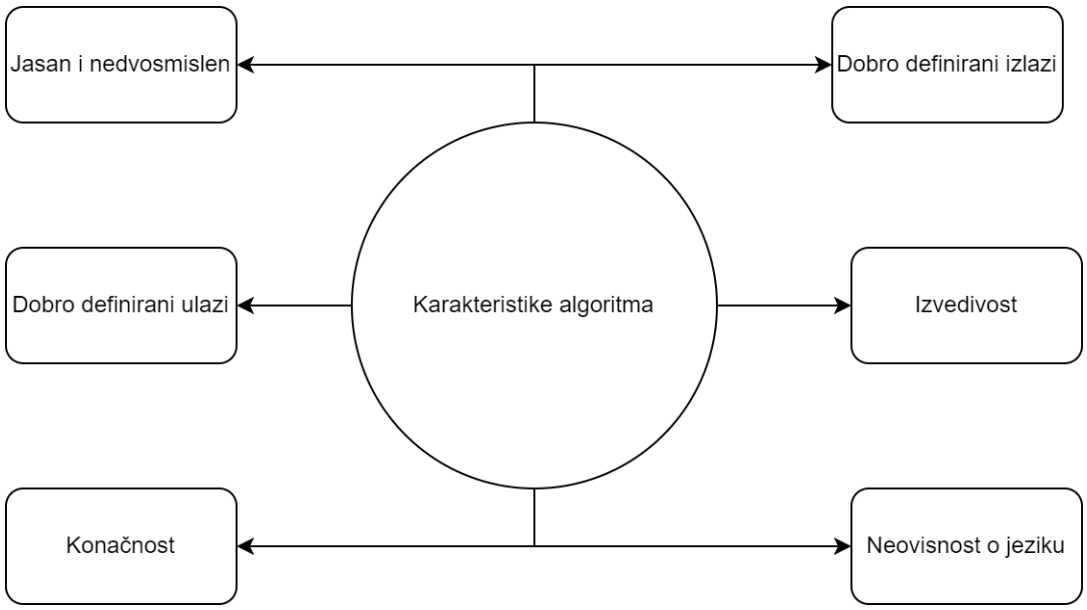
2.1. Karakteristike algoritma

Da bi neke instrukcije bile algoritam, algoritam mora imati sljedeće karakteristike:

a) **Jasan i nedvosmislen:** algoritam bi trebao biti jasan i nedvosmislen. Svaki njegov korak treba biti jasan u svim aspektima i mora voditi samo jednom značenju.
b) **Dobro definirani ulazi:** ako algoritam kaže da treba uzeti ulaze, to bi trebali biti dobro definirani ulazi. Može, ali i ne mora uzeti unos.
c) **Dobro definirani izlazi:** algoritam mora jasno definirati koji će se izlazi dati i on također treba biti dobro definiran. Treba provesti najmanje 1 izlaz.
d) **Konačnost:** algoritam mora biti konačan, tj. trebao bi završiti nakon konačnog vremena.

e) **Izvedivo:** algoritam mora biti jednostavan, generički i praktičan, takav da se može izvršiti s raspoloživim resursima. Ne sadržavati neku tehnologiju budućnosti ili nešto slično.

f) **Neovisan o jeziku:** dizajnirani algoritam mora biti neovisan o jeziku, tj. to moraju biti jednostavne instrukcije koje se mogu implementirati na bilo kojem jeziku, a ipak će izlaz biti isti, kao što se očekuje.



Slika 2: slikovni prikaz karakteristika algoritma

2.2. Dizajn algoritma

Za pisanje algoritma potrebne su sljedeće stvari kao preduvjet:

1. problem koji se algoritmom želi riješiti, tj. jasna definicija problema
2. prilikom rješavanja problema moraju se uzeti u obzir ograničenja problema
3. unos koji treba poduzeti za rješavanje problema
4. rezultat koji se može očekivati kada se problem želi riješiti
5. rješenje problem je unutar zadanih ograničenja

Zatim se pomoću gornjih parametara piše algoritam tako da rješava problem.

3. Lingvistička pravila za njemački jezik

U nastavku bit će prikazana lingvistička pravila koja formalno zapisuju postupak slogovanja.

3.1. Pretprocesiranje

Slova pojedine abecede prirodnog jezika mogu prouzročiti probleme u radu sa znakovnim nizovima zbog načina kodiranja (npr. simboli č, ž, š, đ, ź, ł, ś, ń i slični). Takvi simboli mogu se zamijeniti jednoznačnim simbolima koji su jednostavniji za kodiranje. Ovaj problem je riješen tako da je korišten „Visual Studio Code“ kao radno okruženje.

Zatim je potrebno analizirati abecedu koja se koristi u njemačkom jezičnom području te isprobati učitavanje tekstualne datoteke u kojoj se nalaze takvi specifični znakovi. Također, potrebno je omogućiti da se svaka riječ iz rečenice promatra zasebno, neovisno o razmacima i različitim interpunkcijskim znakovima koji se u rečenici mogu nalaziti. Neki od interpunkcijskih znakova su: !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~.

Također, promatranje svake riječi potrebno je omogućiti formalizmom samoglasnika i suglasnika. Odnosno, svaku riječ na ulazu je potrebno promatrati kao niz samoglasnik (V) i suglasnika (C). Kada riječi promatramo kao nizove samoglasnika i suglasnika (a ne kao originalne riječi s pripadnim morfemima, tj. slovima) tada sukladno definiranim lingvističkim pravilima možemo pronalaziti uzorke koji se u riječi javljaju. Na temelju njih rastaviti riječi na slogove temeljem danih pravila koja se spominju kasnije u ovom poglavlju.

3.2. Abeceda, samoglasnici i suglasnici njemačkog jezika

Abeceda njemačkog jezika:

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, Ä, Ö, Ü, ß

Samoglasnici i suglasnici njemačkog jezika:

SAMOGLASNICI – V – VOWELS:
Vokali (V): a, e, i, o, u, ä, ö, ü, ei, ai, ey, ay, eu, äu, ie, au; aa, ee, oo
SUGLASNICI – C – CONSONANTS:
Konsonanti (C): b c d f g h j k l m n p q r s t v w x y z, ß, sch, ch, ck, qu, ph

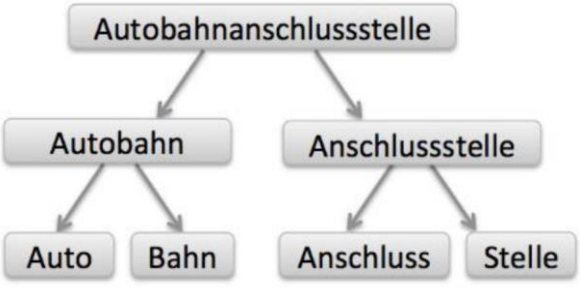
Tablica 1: samoglasnici i suglasnici njemačkog jezika

3.3. Razrada algoritma na temelju lingvističkih pravila

Riječi, rečenice ili tekst koji je dan na ulazu, na izlazu mora biti zapisan u obliku slogova. Znakom razmaka obilježavamo granice između slogova.

3.3.1. Rastavljanje složenica

Najprije je potrebno odvojiti jednokorijenske riječi u složenicama koristeći Python biblioteku **compound-split**.



Slika 3: primjer rastavljanja složenice u jednokorijenske riječi

3.3.2. Odvojiti prefikse

Najprije, prefiks u gramatici nalazi se ispred korijena neke riječi. Dodavanje na početak jedne riječi, nju mijenja u drugu riječ, često suprotnog značenja. Na primjer, kada je prefiks *ne* dodan na riječ sretan, stvara riječ nesretan.

Stoga, za njemačko jezično područje potrebno je odvojiti prefikse koji završavaju sljedećim konsonantima:

an, ab, auf, aus, dis, ein, fehl, her, hin, haupt, in, dar, durch, , los, mit, nach, von, vor, weg, um, un, ur, ent, er, ver, zer, miss, miß, niss, niß, ex, non, super, trans, kon, hoch, stink, stock, tief, tod, erz

Konsonanti koji se nalaze u donjem okviru su dvosložni i oni će se rastaviti u sljedećem koraku:

unter, über, hinter, wider, wieder, weiter, zurück, zurecht, zusammen, hyper, inter

3.3.3. Uzorci za rastavljanje

U sljedećoj tablici su prikazani uzorci za rastavljanje riječi. Za lakšu interpretaciju rastavljanje slogova prikazano je crticom „-“.

VCCV	VC-CV
VCCCV	VCC-CV
VCV	V-CV

Tablica 2: prikaz uzoraka za rastavljanje riječi na slogove

Treće pravilo koje je navedeno u tablici pokriva pravilo da intervokalni h pripada sljedećem slogu, upravo zbog intervokalnog C koji ide u sljedeći slog.

3.3.4. Raspis pravila za rastavljanje st, tz, i pf u sredini riječi

U sljedećoj tablici su prikazana pravila za rastavljanje riječi kad se **st**, **tz** i **pf** nalaze u sredini riječi. Za lakšu interpretaciju rastavljanje slogova prikazano je crticom „-“.

VstV	Vs-tV
VCstV	VCs-tV
VstCV	VS-tCV
VxtCV	Vx-tCV
VtzV	Vt-zV
VCtzV	VCt-zV
VtzCV	Vtz-CV
VpfV	Vp-fV
VCpfV	VCp-fV
VpfCV	Vpf-CV

Tablica 3: prikaz pravila za rastavljanje riječi kad se **st**, **tz** i **pf** nalaze u sredini riječi

3.3.5. Iznimke

U sljedećoj tablici su prikazana pravila za rastavljanje riječi kad se nalazi iznimka.

1. iznimka
Knie se rastavlja kao: Kni-e ako ispred njega stoje: die, der, ili ako prethodna riječ završi na -e ili -er, ali ne rastavlja se ako ispred riječi koja završava na -e stoji *das*, s time da ispred *das* mora biti jedna ili više riječi koje završavaju na -e
2. iznimka
tsch vrijedi kao jedan C kad je na početku riječi, tj. u poziciji tschV = CV i u poziciji na kraju riječi, tj. Vtsch = VC kad se nađe VtschCV, rastavlja se: Vtsch-CV kad se nađe VtschV, rastavlja se: Vt-schV
3. iznimka
Ako se u rečenici nađu dva samoglasnika koji se ne nalaze u popisu vokala, tada oni trebaju biti razdvojeni. Npr. za riječ poetisch i kad se nađu samoglasnici oe, oni trebaju biti rastavljeni u obliku o-e, u konačnici po-e -tisch. Isto vrijedi i s riječi Nationen.

Tablica 4: tablica prikazuje pravila za rastavljanje riječi kad se nalazi iznimka

4. Programski kôd algoritma

U nastavku će biti opisan programski kôd algoritma koji je napisan u programskom jeziku Python. Također, bit će nabrojani i opisani moduli ili biblioteke koje su korištene.

4.1. Korišteni moduli

Moduli koji su korišteni za razvoj algoritma su:

- re
- string
- compound-split

```
import re, string
from compound_split import doc_split
```

Kôd 1: prikaz uključenih modula

Detaljan opis modula nalazi se u nastavku.

4.1.1. Modul re – regularni izrazi

Modul re (engl. regular expression) je ugrađena Python biblioteka koja omogućava podršku za regularne izraze. Regularni izrazi su koristan alat za podudaranje teksta baziranog na definiranom uzorku. Može otkriti prisutnost ili odsutnost teksta tako što ga povezuje s određenim uzorkom, a također može podijeliti uzorak u jedan ili više pod uzoraka. Njegova primarna funkcija je ponuditi pretraživanje, gdje uzima regularni izraz i niz.

Regularni izrazi u Pythonu mogu se koristiti za rudarenje (pretraživanje) ili za validaciju podataka.

Regularni izrazi prilikom kreiranja algoritma su korišteni kako bi se u tekstu kojem su slova zamijenjena u obliku samoglasnika (V) i suglasnika (C) pronašli uzorak koji se odnosi na pravilo rastavljanja. Kad je taj uzorak pronađen tada je primijenjeno pravilo rastavljanja te riječi u zadanom tekstu.

4.1.2. Modul string

Modul string je ugrađena Python biblioteka. Pruža mogućnosti zamjene složenih varijabli i oblikovanja vrijednosti putem format() metode. Omogućuje stvaranje i prilagodbu vlastitog ponašanja oblikovanja niza koristeći istu implementaciju kao ugrađena funkcija format().

Modul string je korišten pri razvoju algoritma najviše zbog funkcije string.punctation. Funkcija string.punctation u sebi sadrži sve interpunkcijske znakove, tako da kad algoritam dobije rečenicu ili tekst kao input može ignorirati sve interpunkcijske znakove i posvetiti se riječima.

4.1.3. Modul compound-split

Modul compound-split je Python modul za razdvajanje njemačkih složenica. Na primjer, njemačka složenica „*autobahnanschlussstelle"* koja treba biti rastavljena na „auto bahn anschluss stelle".

Metoda izračunava vjerojatnost pojavljivanja n-grama na početku, kraju i u sredini riječi i identificira najvjerojatnije mjesto za razdvajanje.

Autor se referencira na to da metoda postiže ~95% točnosti i da je model treniran na milion njemačkih imenica s Wikipedije.

Upute za instalaciju modula compound-split nalazi se u nastavku:

```
python -m venv env --prompt algorithm
pip install compound-split
```

Naredba 1: postavljanje virtualne okoline i instalacija modula compound-split

4.2. Opis glavnih dijelova algoritma

Algoritam se sastoji od globalnih varijabli, globalnih lista i triju funkcija. Detaljan opis uloga funkcije i korištenja globalnih varijabli bit će opisan u ovome dijelu poglavlja.

4.2.1. Globalne liste i globalne varijable

Globalne varijable su one koje nisu definirane unutar funkcije i imaju globalni opseg, dok su lokalne varijable one koje su definirane unutar funkcije i njihov je opseg ograničen samo na tu funkciju. Drugim riječima, možemo reći da su lokalne varijable dostupne samo unutar

funkcije u kojoj su inicijaliziran, dok su globalne varijable dostupne u cijelom programu unutar svake funkcije. Isto vrijedi i za globalne liste.

Algoritam se sastoji od četiri globalne liste:

1. vovels – lista koja sadrži popis samoglasnika
2. consonants – lista koja sadrži popis suglasnika
3. prefixes – lista koja sadrži popis prefiksa
4. splitting – lista koja sadrži popis uzoraka

```
vowels = ["a", "e", "i", "o", "u", "ä", "ö", "ü", "ei", "ai", "ey", "ay",
"eu", "äu", "ie", "au", "aa", "ee", "oo"]

consonants = ["b", "c", "d", "f", "g", "h", "j", "k", "l", "m", "n", "p",
"q", "r", "s", "t", "v", "w", "x", "y", "z", "ß", "sch", "ch", "ck", "qu",
"ph"]

prefixes = ["an", "ab", "auf", "aus", "dis", "ein", "fehl", "her", "hin",
"haupt", "in", "dar", "durch", "los", "mit", "nach", "von", "vor", "weg",
"um", "un", "ur", "ent", "er", "ver", "zer", "miss", "miß", "niss", "niß",
"ex", "non", "super", "trans", "kon", "hoch", "stink", "stock", "tief",
"tod", "erz", "unter", "über", "hinter", "wider", "wieder", "weiter",
"zurück", "zurecht", "zusammen", "hyper", "inter"]

splitting = ["VCCV", "VCCCV", "VCV", "VV"]

sentence = "Mein name ist Antonio"
```

Kôd 2: globalne varijable i globalne liste

4.2.2. Funkcija syllabels_rules_execeptions

Funkcija syllables_rules_exeptions prima jedan parametar, a to je varijabla sentence. Ovom funkcijom rješava se pravilo rastavljanja složenica, 1. iznimke i 2. iznimke. Gdje ujedno funkcija vraća rečenicu u kojoj su pokrivena navedena pravila.

```
def syllables_rules_exceptions(input_sentence):
    global prefixes
    global vowels
    global consonants
```

```
    doc_split.MIDDLE_DOT = " "
    sentence_doc_split = doc_split.doc_split(input_sentence)
    sentence_exceptions_knie = sentence_doc_split.split(" ")
    regex_list_of_list = []
    for word in sentence_exceptions_knie:
        regex = re.findall(r'(?i)knie.*', word)
        if not regex:
            regex_list_of_list.append("X")
        else:
            regex_list_of_list.append(regex)
        regex_list = [val for sublist in regex_list_of_list for val in
sublist]

        sentence_checksum = sentence_exceptions_knie
    if len(regex_list) == len(sentence_checksum):
        index = 0
        while index < len(sentence_checksum):
            if regex_list[index].lower() ==
sentence_exceptions_knie[index].lower():
                if index == 0 and sentence_exceptions_knie[index].lower() ==
"knie":
                    sentence_exceptions_knie[index] =
sentence_exceptions_knie[index]
                elif sentence_exceptions_knie[index - 1][-1:] == "e" and
sentence_exceptions_knie[index - 2] == "das" and
sentence_exceptions_knie[index - 3][-1:] == "e":
                    sentence_exceptions_knie[index] ==
sentence_exceptions_knie[index]
                elif sentence_exceptions_knie[index - 1].lower() == "die" or
sentence_exceptions_knie[index - 1].lower() == "der" or
sentence_exceptions_knie[index - 1][-1:].lower() == "e":
                    letter_index =
sentence_exceptions_knie[index].lower().find("knie")
                    word = sentence_exceptions_knie[index]
                    sentence_exceptions_knie[index] = word[:letter_index +
3] + " " + word[letter_index + 3:]
            index += 1

    sentence_knie = ' '.join(sentence_exceptions_knie)
    sentence_prefixes = sentence_knie.split(" ")

    for prefix in prefixes:
        prefix_length = len(prefix)
        for word in sentence_prefixes:
            if word[:prefix_length] == prefix:
                index = sentence_prefixes.index(word)
                sentence_prefixes[index] = word[:prefix_length] + " " +
word[prefix_length:]
```

```
    sentence_split_prefixes = ' '.join(sentence_prefixes)
    sentence_vc = sentence_split_prefixes.split(" ")

    regex_list_of_list = []
    for word_vc in sentence_vc:
        regex = re.findall(r'.*tsch.*', word_vc)
        if not regex:
            regex_list_of_list.append("X")
        else:
            regex_list_of_list.append(regex)
    regex_list = [val for sublist in regex_list_of_list for val in sublist]

    if len(regex_list) == len(sentence_vc):
        index = 0
        while index < len(sentence_vc):
            if regex_list[index].lower() == sentence_vc[index].lower():
                sentence_length_01 = len(sentence_vc[index]) - 1
                sentence_length_02 = len(sentence_vc[index])
                tsch_length = len("tsch")
                letter_index = sentence_vc[index].find("tsch")

                if letter_index == 0:
                    word = sentence_vc[index]
                    sentence_vc[index] = "C" + word[tsch_length:]

                if letter_index == sentence_length_01 - tsch_length or
letter_index == sentence_length_02 - tsch_length:
                    word = sentence_vc[index]
                    sentence_vc[index] = word[:letter_index] + "C"
            index += 1

    sentence_join = ' '.join(sentence_vc)
    sentence_letters = list(sentence_join)

    counter = 0
    for letter in sentence_letters:
        for sign in consonants:
            if letter == sign.upper() or letter == sign.lower() and letter
!= string.punctuation:
                sentence_letters[counter] = "C"
        for sign in vowels:
            if letter == sign.upper() or letter == sign.lower() and letter
!= string.punctuation:
                sentence_letters[counter] = "V"
        counter += 1

    sentence_in_vc = ''.join(sentence_letters)
```

```
    sentence_vc = sentence_in_vc.split(" ")
    sentence = sentence_split_prefixes.split(" ")

    regex_list_of_list = []
    for word_vc in sentence:
        regex = re.findall(r'.*tsch.*', word_vc)
        if not regex:
            regex_list_of_list.append("X")
        else:
            regex_list_of_list.append(regex)
    regex_list = [val for sublist in regex_list_of_list for val in sublist]

    if len(regex_list) == len(sentence):
        index = 0
        while index < len(sentence):
            if regex_list[index].lower() == sentence[index].lower():
                sentence_length_01 = len(sentence[index]) - 1
                sentence_length_02 = len(sentence[index])
                tsch_length = len("tsch")
                letter_index = sentence[index].find("tsch")

                if letter_index != 0 and letter_index != sentence_length_01
- tsch_length and letter_index != sentence_length_02 - tsch_length:
                    word = sentence[index]
                    if sentence_vc[index][letter_index - 1] == "V" and
sentence_vc[index][letter_index + tsch_length] == "C" and
sentence_vc[index][letter_index + tsch_length + 1] == "V":
                        sentence[index] = word[:letter_index + 4] + " " +
word[letter_index + 4:]
                    if sentence_vc[index][letter_index - 1] == "V" and
sentence_vc[index][letter_index + tsch_length] == "V":
                        sentence[index] = word[:letter_index + 1] + " " +
word[letter_index + 1:]
            index += 1

    sentence_tsch = ' '.join(sentence)
    return sentence_tsch
```

Kôd 3: funkcija slyllabels_rules_exceptions

4.2.3. Funkcija sentence_in_vc

Funkcija sentence_in_vc prima jedan parametar, a to je varijable sentence. Ova funkcija danu rečenicu pretvara u tekst kojem su slova zamijenjena u obliku samoglasnika (V) i suglasnika (C). Gdje funkcija ujedno vraća rečenicu u navedenom obliku. Na primjer, rečenica „Mein name ist Antonio“ je pretvorena u sljedeće „CVVC CVCV VCC VCCVCVV“.

```
def sentence_in_vc(input_sentence):
    global vowels
    global consonants

    sentence_letters = list(input_sentence)
    sentence = input_sentence.split(" ")

    regex_list_of_list = []
    for word_vc in sentence:
        regex = re.findall(r'.*tsch.*', word_vc)
        if not regex:
            regex_list_of_list.append("X")
        else:
            regex_list_of_list.append(regex)
    regex_list = [val for sublist in regex_list_of_list for val in sublist]
    if len(regex_list) == len(sentence):
        index = 0
        while index < len(sentence):
            if regex_list[index].lower() == sentence[index].lower():
                sentence_length_01 = len(sentence[index]) - 1
                sentence_length_02 = len(sentence[index])
                tsch_length = len("tsch")
                letter_index = sentence[index].find("tsch")

                if letter_index == 0:
                    word = sentence[index]
                    sentence[index] = "C" + word[tsch_length:]

                if letter_index == sentence_length_01 - tsch_length or
letter_index == sentence_length_02 - tsch_length:
                    word = sentence[index]
                    sentence[index] = word[:letter_index] + "C"
            index += 1

    sentence_join = ' '.join(sentence)
    sentence_letters = list(sentence_join)

    counter = 0
```

```
    for letter in sentence_letters:
        for sign in consonants:
            if letter == sign.upper() or letter == sign.lower() and letter
!= string.punctuation:
                sentence_letters[counter] = "C"
        for sign in vowels:
            if letter == sign.upper() or letter == sign.lower() and letter
!= string.punctuation:
                sentence_letters[counter] = "V"
        counter += 1

    sentence_vc = ''.join(sentence_letters)
    return sentence_vc
```

Kôd 4: funkcija sentence_in_vc

4.2.4. Funkcija syllables_rules

Funkcija syllabels_rules prima jedan parametar, a to je varijable sentence. ovom funkcijom rješava se 3. pravilo i 3. iznimku. Gdje ujedno funkcija vraća rečenicu u kojoj su pokrivena navedena pravila.

```
def syllables_rules(input_sentence):
    global splitting
    global vowels

    sentence = input_sentence.split(" ")
    sentence_vc = sentence_in_vc(input_sentence).split(" ")

    for split in splitting:
        if split == "VCCV":
            regex_list_of_list = []
            for word_vc in sentence_vc:
                regex = re.findall(r'.*{}.*'.format(split), word_vc)
                if not regex:
                    regex_list_of_list.append("X")
                else:
                    regex_list_of_list.append(regex)
            regex_list = [val for sublist in regex_list_of_list for val in
sublist]
            if len(regex_list) == len(sentence_vc):
                index = 0
                while index < len(sentence_vc):
                    if regex_list[index] == sentence_vc[index]:
                        letter_index = sentence_vc[index].find(split)
                        word = sentence[index]
```

```
                        sentence[index] = word[:letter_index + 2] + " " +
word[letter_index + 2:]

                    index += 1

    sentence_splitting = ' '.join(sentence)
    sentence_splitting_vc = ' '.join(sentence)
    sentence = sentence_splitting.split(" ")
    sentence_vc = sentence_in_vc(sentence_splitting_vc).split(" ")

    for split in splitting:
        if split == "VCCCV":
            regex_list_of_list = []
            for word_vc in sentence_vc:
                regex = re.findall(r'.*{}.*'.format(split), word_vc)
                if not regex:
                    regex_list_of_list.append("X")
                else:
                    regex_list_of_list.append(regex)
            regex_list = [val for sublist in regex_list_of_list for val in
sublist]

            if len(regex_list) == len(sentence_vc):
                index = 0
                while index < len(sentence_vc):
                    if regex_list[index] == sentence_vc[index]:
                        difference_st = sentence[index].find("st") -
sentence_vc[index].find("VCCCV")
                        difference_xt = sentence[index].find("xt") -
sentence_vc[index].find("VCCCV")

                        if difference_st == 1 or difference_xt == 1:
                            letter_index = sentence_vc[index].find(split)
                            word = sentence[index]
                            sentence[index] = word[:letter_index + 2] + " "
+ word[letter_index + 2:]

                        else:
                            letter_index = sentence_vc[index].find(split)
                            word = sentence[index]
                            sentence[index] = word[:letter_index + 3] + " "
+ word[letter_index + 3:]
                    index += 1


    sentence_splitting = ' '.join(sentence)
    sentence_splitting_vc = ' '.join(sentence)
    sentence = sentence_splitting.split(" ")
```

```
    sentence_vc = sentence_in_vc(sentence_splitting_vc).split(" ")

    for split in splitting:
        if split == "VCV":
            regex_list_of_list = []
            for word_vc in sentence_vc:
                regex = re.findall(r'.*{}.*'.format(split), word_vc)
                if not regex:
                    regex_list_of_list.append("X")
                else:
                    regex_list_of_list.append(regex)
            regex_list = [val for sublist in regex_list_of_list for val in
sublist]

            if len(regex_list) == len(sentence_vc):
                index = 0
                while index < len(sentence_vc):
                    if regex_list[index] == sentence_vc[index]:
                        letter_index = sentence_vc[index].find(split)
                        word = sentence[index]
                        sentence[index] = word[:letter_index + 1] + " " +
word[letter_index + 1:]

                    index += 1

    sentence_splitting = ' '.join(sentence)
    sentence_splitting_vc = ' '.join(sentence)
    sentence = sentence_splitting.split(" ")
    sentence_vc = sentence_in_vc(sentence_splitting_vc).split(" ")

    for split in splitting:
        if split == "VV":
            regex_list_of_list = []
            for word_vc in sentence_vc:
                regex = re.findall(r'.*{}.*'.format(split), word_vc)
                if not regex:
                    regex_list_of_list.append("X")
                else:
                    regex_list_of_list.append(regex)
            regex_list = [val for sublist in regex_list_of_list for val in
sublist]

            index = 0
            if len(regex_list) == len(sentence_vc):
                index = 0
                while index < len(sentence_vc):
                    if regex_list[index] == sentence_vc[index]:
                        letter_index = sentence_vc[index].find(split)
```

```
                        word = sentence[index]
                        vowels_check = word[letter_index] +
word[letter_index + 1]
                        if vowels_check.lower() not in vowels:
                            sentence[index] = word[:letter_index + 1] + " "
+ word[letter_index + 1:]
                    index += 1

    sentence_splitting = ' '.join(sentence
    return re.sub(' +', ' ', sentence_splitting)
```

Kôd 5: funkcija syllabels_rules

5. Osnovne upute za korištenje algoritma

U ovome poglavlju bit će prikazano kako koristiti algoritam. Algoritam je jednostavan za korištenje, bitno je imati ulaznu rečenicu koja se želi rastaviti na slogove.

Na primjer, ako u algoritmu zadamo rečenicu „Mein name ist Antonio.“ tada tu rečenicu pomoću varijable prosljeđujemo u funkciju.

```
sentence = "Mein name ist Antonio."

syllables_rules(syllables_rules_exceptions(sentence))
```

Kôd 6: primjer prosljeđivanja varijable koja je tipa string u funkciju

Kao što vidimo iz primjera koristi se ugnježđivanje funkcije, gdje u funkciju syllables_rules prosljeđujemo izlaz funkcije syllables_rules_exceptions, u koju opet prosljeđujemo danu rečenicu.

Algoritam je implementiran na ovaj način jer temeljem testiranja ovako dobiva najbolje rezultate, odnosno ne događaju se anomalije i sudaranja pravila jedno s drugim prilikom pokretanja programa.

Međutim, jednim prolazom algoritma kroz riječ ili rečenicu ne razdvaja sve slogove. Stoga, kako bi algoritam u potpunosti rastavio sve slogove u riječi ili rečenici potrebno je pokrenuti funkcije u tri kruga, odnosno tri puta. Pri kreiranju algoritma zaključak je da pokretanje funkcije u tri kruga rastavlja sve riječi koje u prethodna dva kruga nisu obuhvaćene.

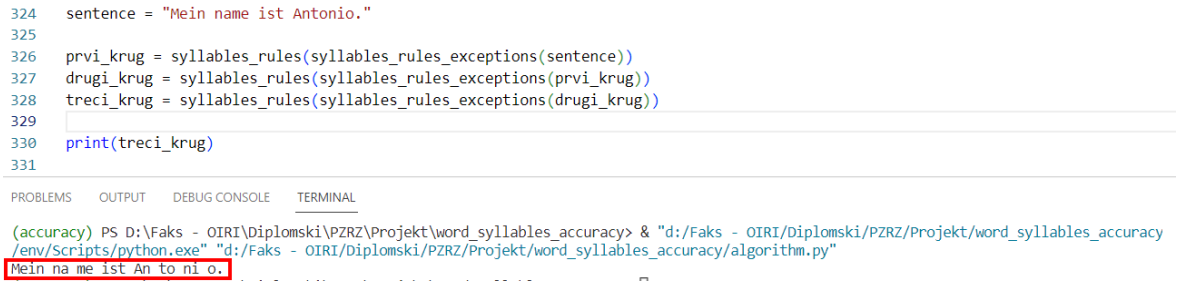
```
sentence = "Mein name ist Antonio."

prvi_krug = syllables_rules(syllables_rules_exceptions(sentence))
drugi_krug = syllables_rules(syllables_rules_exceptions(prvi_krug))
treci_krug = syllables_rules(syllables_rules_exceptions(drugi_krug))

print(treci_krug)
```

Kôd 7: primjer pokretanja funkcija u tri kruga

Kao što možemo vidjeti u prethodnom primjeru nad danom rečenicom pokrenuli smo funkciju syllables_rules tri puta i spremili ju u varijablu, te smo tu varijablu kao rezultat pokrenute funkcije proslijedili ponovno toj istoj funkciji syllables_rules. Na kraju smo varijablu treci_krug ispisali na konzolu kao konačan rezultat rastavljene riječi ili rečenice.



Slika 4: prikaz rastavljene rečenice

6. Izrada web aplikacije i korisničkog sučelja

Nakon što je algoritam kreiran, moguće je izraditi funkcionalnu web aplikaciju. Izrađeno je programsko sučelje u kojem korisnik može odabrati jednu od dviju opcija.

1. Korisnik samostalno upisuje riječ ili rečenicu na ulazu preko standardnog ulaza (tipkovnica), a program će mu odmah za uneseni tekst ispisati na zaslonu rastavljeni tekst na slogove.
2. Korisnik odabire putanju na svojem računalu do tekstualne datoteke (.txt) u kojoj je zapisan tekst kojeg želi očitati na ulazu, a rastavljeni tekst na slogove ispisuje na ekran.

Web aplikacija izrađena je u Flask-u. Flask je Python web *framework,* namijenjen razvoju web aplikacija. Po svojoj veličini smatra se *micro-frameworkom,* zbog čega je vrlo popularan kao alat za učenje web programiranja u Pythonu. Naziva ga se i *non-full-stack frameworkom,* za razliku od *full-stack frameworka* poput Djanga ili TurboGearsa. No, to ne znači da Flak nije moćan poput navedenih platformi, jer je dizajniran sa svrhom da bude lako proširivi uz pomoć tzv. *Flask ekstenzija*. Dakle, sam Flask nudi temelje za razvoj, a brojne ekstenzije nude mogućnost da web aplikacija bude lako proširiva s onime što korisnik zahtjeva (poput rada s web formama, autentifikacije, pristupa bazama podataka, prijavama i sl.).

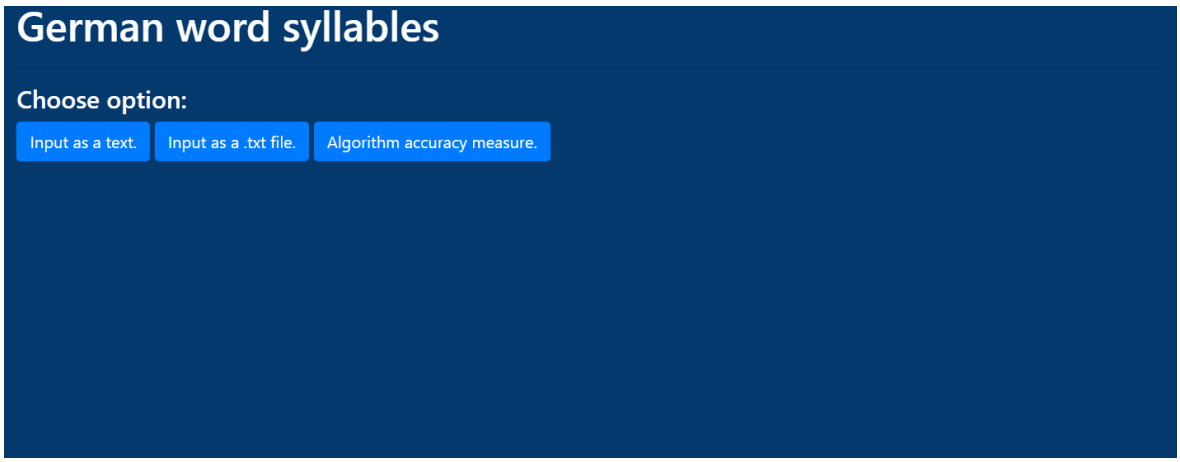
Za razvoj web aplikacije odabran je Flask jer izrada same web aplikacije nije zahtjevna, odnosno zahtjeva samo jedan ulaz i jedan izlaz.

U ovome poglavlju Flask se neće spominjati u detalje već će biti prikazane snimke zaslona kako je aplikacija implementirana i kako ju koristiti kao korisnik. A kôd web aplikacije može se pronaći u zadnjem poglavlju „Prilog: programski kôd“ gdje se nalazi URL do GitHub repozitorija.

6.1. Osnovne upute za korištenje web sučelja

Početna stranica web sučelja sastoji se od naslova, podnaslova i triju gumba. Korisnik na odabir gumba odabire željenu opciju koju želi izvršiti nad tekstom. Tako na primjer klikom

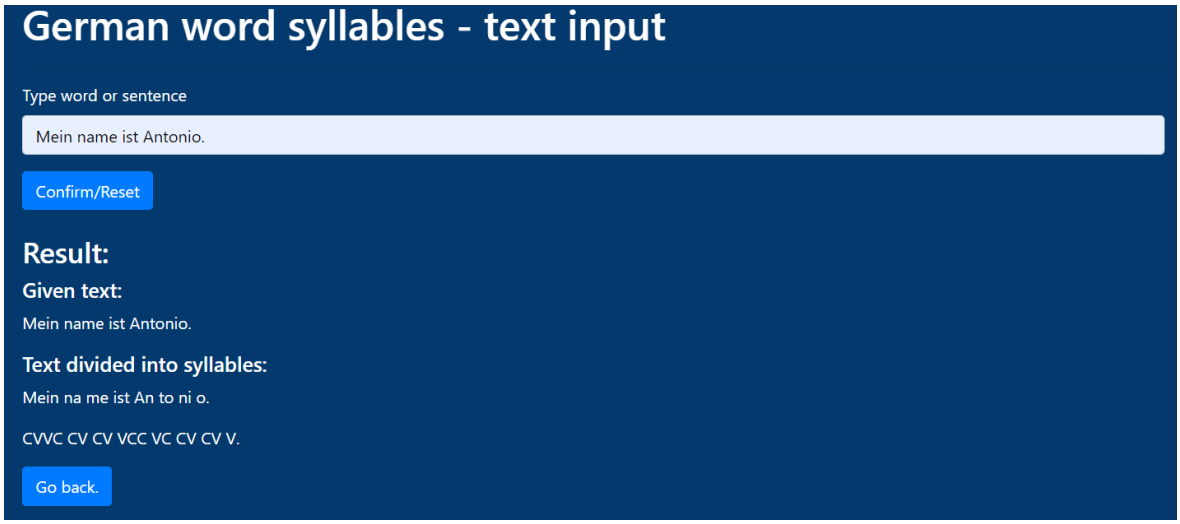
na prvi gumb korisnik odabire opciju da će ručno (pomoću tipkovnice) unijeti tekst za obradu, odnosno na rastavljanje na slogove.



Slika 5: prikaz početne stranice web sučelja

Kad korisnik pritisne na gumb „Input as a text.“ otvara mu se web stranica koja se sastoji od:

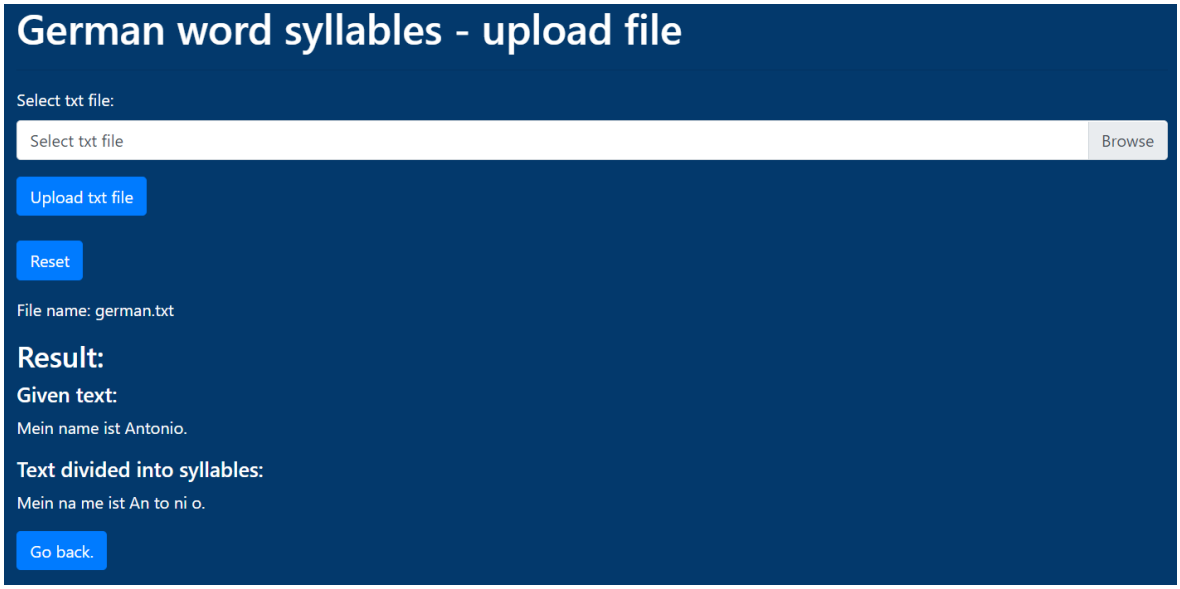
a) naslova
b) forme za unos teksta
c) gumba za potvrdu i resetiranje rezultata
d) polja za prikaz rezultata
e) Gumb za vraćanje na početnu stranicu



Slika 6: prikaz korisničkog sučelja kad korisnik pritisne gumb „Input as a text.“

Kad korisnik pritisne na gumb „Input as a .txt file.“ otvara mu se web stranica koja se sastoji od:

a) naslova
b) forme za unos dokumenta u .txt formatu
c) gumba za učitavanje dokumenta za obradu
d) gumba za resetiranje
e) polja za prikaz rezultata
f) gumb za vraćanje na početnu stranicu



Slika 7: prikaz korisničkog sučelja kad korisnik pritisne gumb „Input as a .txt file.“

Kad korisnik pritisne na gumb „Algorithm accuracy measure.“ otvara mu se web stranica koja se sastoji od:

a) naslova
b) testnih riječi
c) verificiranih razdvojenih riječi
d) rezultata koji algoritam postiže pri razdvajanju testnih riječi
e) gumba za vraćanje na početnu stranicu

Slika 8: web stranica koja prikazuje mjeru točnosti algoritma

7. Testiranje algoritma

Načinjeni algoritam potrebno je testirati na pripremljenom skupu testnih riječi za njemački jezik. Uspješnost algoritma izražena je numerički koristeći mjeru kojom se mjeri točnost (engl accuracy). Točnost algoritma mjeri se na temelju uspješno rastavljenih riječi u skupu svih riječi. Izračun je napravljen programskom skriptom koja to sama računa.

Također, kako bi preciznost uspješno rastavljenih riječi bila što točnija korištena je mjera za djelomičnu točnost (npr. kada je dio riječi rastavljen dobro, a drugi dio nije i sl.)

7.1. Skup testnih riječi

U nastavku se nalazi popis testnih riječi koje su dane na testiranje i evaluaciju. Također, za dane testne riječi, dani su primjeri i za točno rastavljene riječi kako bi se mogla napraviti mjera točnosti.

Skup testnih riječi i verificiranih rastavljenih riječi u Pythonu, napravljene su dvije liste kako bi se nad njima mogla izvršiti mjera točnosti.

Skup testnih riječi:

```
test_words = ["Diät", "Knie", "die knie", "der knie", "reise knie", "Auto",
"Seeufer", "Katze", "Tatze", "Pfütze", "putzen", "platzen", "Bürste",
"Kiste", "Hamster", "Fenster", "hinstellen", "darstellen", "erstarren",
"plötzlich", "Postauto", "Kratzbaum", "boxen", "heben", "rodeln",
"Schifffahrt", "Mussspiel", "wichtigsten", "besuchen", "gewinnen",
"vergessen", "abangeln", "Kreuzotter", "poetisch", "Nationen", "aber",
"über", "Kreuzklemme", "Foxtrott", "witzlos", "witzig", "wegschmeißen",
"Bettüberzug", "wirtschaft", "Beziehungsknatsch", "Gletscher",
"Wurstscheibe", "Borretschgewächs", "Bodden", "Handball", "Neubau",
"Stalltür", "Autobahnanschlussstelle", "Laufschuhe", "Baustelle",
"Lebkuchen", "Himbeere", "Klassenzimmer", "Hubbleteleskop", "Botschaft",
"Schokoladenfabrik", "Hühnersuppe", "Schweinebraten", "Halsschmerzen",
"Weltanschauung", "Weltschmerz", "Weihnachtsbaum", "Kugelschreiber",
"Bohnensalat", "Freundschaftsbezeigung", "Weihnachtsmannfigur",
"Glasflächenreinigung"]
```

Kôd 8: prikaz skupa testnih riječi

Skup testnih riječi koje su ispravno rastavljene:

```
verify_words = ["Di ät", "Knie", "die kni e", "der kni e", "rei se kni e",
"Au to", "See ufer", "Kat ze", "Tat ze", "Pfüt ze", "put zen", "plat zen",
"Bürs te", "Kis te", "Hams ter", "Fens ter", "hin stel len", "dar stel len",
"er star ren", "plötz lich", "Post au to", "Kratz baum", "bo xen", "he ben",
"ro deln", "Schiff fahrt", "Muss spiel", "wich tigs ten", "be su chen", "ge
win nen", "ver ges sen", "ab an geln", "Kreuz ot ter", "po e tisch", "Na ti
o nen", "a ber", "ü ber", "Kreuz klem me", "Fox trott", "witz los ", "wit
zig", "weg schmei ßen", "Bett ü ber zug", "wirt schaft", "Be zieh ungs
knatsch", "Glet scher", "Wurst schei be", "Bor retsch ge wächs", "Bod den",
"Hand ball", "Neu bau", "Stall tür", "Au to bahn an schluss stel le", "Lauf
schu he", "Bau stel le", "Leb ku chen", "Him bee re", "Klas sen zim mer",
"Hub ble te le skop", "Bot schaft", "Scho ko la den fa brik", "Hüh ner sup
pe", "Schwei ne bra ten", "Hals schmer zen", "Welt an schau ung", "Welt
schmerz", "Weih nachts baum", "Ku gel schrei ber", "Boh nen sa lat", "Freund
schafts be zei gung", "Weih nachts mann fi gur", "Glas fläch en rei ni
gung"]
```

Kôd 9: prikaz testnih riječi onako kako trebaju biti rastavljene

7.2. Programska skripta za računanje mjere točnosti

Za izradu programske skripte koja računa mjeru točnosti rastavljanja riječi na slogove kreirana je nova Python datoteka imena test_accuracy.py unutar direktorija u kojem se nalazi Python datoteka algoritma imena algorithm.py. Cilj koji se želi postići je da se u datoteku imena test_accuracy.py uveze datoteka algorithm.py radi čitkosti kôda.

```
import algorithm
```

Kôd 10: prikaz kôda za uvoz datoteke imena algorithm.py u datoteku test_accuracy.py

Programska skripta računa mjeru točnosti točno rastavljenih riječi od ukupno riječi i mjeru točnosti djelomično točno rastavljenih riječi. Mjera djelomično točno rastavljenih riječi računa se prema točno rastavljenim dijelovima riječi od ukupno točno rastavljenih dijelova riječi. Na primjer, ako dana riječ „hinstellen“ i treba biti rastavljena na način „hin stel len“, a algoritam je riječ rastavio na „hin stell en“ tada se točnost mjeri prema točno rastavljenim dijelovima riječi od ukupno točno rastavljenih dijelova riječi. Što je u ovom slučaju 1 točan slog, a ostala 2 su netočna i to daje rezultat 1/3.

Moduli koji su korišteni za izradu programske skripte su prettytable i termcolor. Modul prettytable je korišten za generiranje ASCII tablice, dok je modul termcolor korišten za isticanje rezultata na terminal u boji.

Module je potrebno zasebno instalirati jer nisu dio standardne Python biblioteke. Instalacija modula se izvršava na sljedeći način.

```
pip install prettytable
pip install termcolor
```

Naredba 2: instalacija zasebno korištenih modula

```
from prettytable import PrettyTable
from termcolor import colored
```

Kôd 11: moduli koji su korišteni za izradu programske skripte

Osim prethodno spomenutih skupa testnih riječi, riječi koje su ispravno rastavljene i opisa korištenih modula, u nastavku se nalazi programska skripta koja obuhvaća navedeno.

```
import algorithm
from prettytable import PrettyTable
from termcolor import colored

test_words = ["Diät", "Knie", "die knie", "der knie", "reise knie", "Auto",
"Seeufer", "Katze", "Tatze", "Pfütze", "putzen", "platzen", "Bürste",
"Kiste", "Hamster", "Fenster", "hinstellen", "darstellen", "erstarren",
"plötzlich", "Postauto", "Kratzbaum", "boxen", "heben", "rodeln",
"Schifffahrt", "Mussspiel", "wichtigsten", "besuchen", "gewinnen",
"vergessen", "abangeln", "Kreuzotter", "poetisch", "Nationen", "aber",
"über", "Kreuzklemme", "Foxtrott", "witzlos", "witzig", "wegschmeißen",
"Bettüberzug", "wirtschaft", "Beziehungsknatsch", "Gletscher",
"Wurstscheibe", "Borretschgewächs", "Bodden", "Handball", "Neubau",
"Stalltür", "Autobahnanschlussstelle", "Laufschuhe", "Baustelle",
"Lebkuchen", "Himbeere", "Klassenzimmer", "Hubbleteleskop", "Botschaft",
"Schokoladenfabrik", "Hühnersuppe", "Schweinebraten", "Halsschmerzen",
"Weltanschauung", "Weltschmerz", "Weihnachtsbaum", "Kugelschreiber",
"Bohnensalat", "Freundschaftsbezeigung", "Weihnachtsmannfigur",
"Glasflächenreinigung"]

verify_words = ["Di ät", "Knie", "die kni e", "der kni e", "rei se kni e",
"Au to", "See ufer", "Kat ze", "Tat ze", "Pfüt ze", "put zen", "plat zen",
"Bürs te", "Kis te", "Hams ter", "Fens ter", "hin stel len", "dar stel len",
"er star ren", "plötz lich", "Post au to", "Kratz baum", "bo xen", "he ben",
"ro deln", "Schiff fahrt", "Muss spiel", "wich tigs ten", "be su chen", "ge
win nen", "ver ges sen", "ab an geln", "Kreuz ot ter", "po e tisch", "Na ti
o nen", "a ber", "ü ber", "Kreuz klem me", "Fox trott", "witz los ", "wit
zig", "weg schmei ßen", "Bett ü ber zug", "wirt schaft", "Be zieh ungs
knatsch", "Glet scher", "Wurst schei be", "Bor retsch ge wächs", "Bod den",
"Hand ball", "Neu bau", "Stall tür", "Au to bahn an schluss stel le", "Lauf
schu he", "Bau stel le", "Leb ku chen", "Him bee re", "Klas sen zim mer",
"Hub ble te le skop", Bot schaft", "Scho ko la den fa brik", "Hüh ner sup
pe", "Schwei ne bra ten", "Hals schmer zen", "Welt an schau ung", "Welt
```

```
schmerz", "Weih nachts baum", "Ku gel schrei ber", "Boh nen sa lat", "Freund
schafts be zei gung", "Weih nachts mann fi gur", "Glas fläch en rei ni
gung"]

correct_words = []

partial_correct_words = []

table = PrettyTable()
table.field_names = ['Test word', 'Verify word', 'Algorithm output',
'Partial correct word', 'Percent of partial word accuracy']

counter = 0
for word in test_words:
    prvi_krug =
algorithm.syllables_rules(algorithm.syllables_rules_exceptions(word))
    drugi_krug =
algorithm.syllables_rules(algorithm.syllables_rules_exceptions(prvi_krug))
    treci_krug =
algorithm.syllables_rules(algorithm.syllables_rules_exceptions(drugi_krug))

    algorithm_output = treci_krug

    if algorithm_output == verify_words[counter]:
        correct_words.append(1)
        partial_correct_words.append(1)
        table.add_row([word, verify_words[counter], algorithm_output, "yes -
all", format(1, ".2f")])
    else:
        correct_words.append(0)
        word_split = algorithm_output.split()
        verify_words_split = verify_words[counter].split()

        part_number = len(verify_words_split)
        part_correct = 0
        for part_vws in verify_words_split:
            for part_ws in word_split:
                if part_ws == part_vws:
                    part_correct += 1

        partial_accuracy = part_correct / part_number
        partial_correct_words.append(partial_accuracy)
        table.add_row([word, verify_words[counter], algorithm_output, f"no -
{part_correct} of {part_number}", format(partial_accuracy, ".2f")])

    counter += 1
```

```
table.add_row([colored("---", "yellow"), colored("---", "yellow"),
colored("---", "yellow"), colored(f" correct:
{sum(partial_correct_words):.2f} of {len(partial_correct_words) -
sum(partial_correct_words):.2f}", "yellow"),
colored(format(sum(partial_correct_words) / len(partial_correct_words),
".2f"), "yellow")])

print(table)

print(f"Full measure of accuracy: {sum(correct_words) /
len(correct_words):.2f}\n"
      f"Correct words: {sum(correct_words)}\n"
      f"Uncorrect words: {len(correct_words) - sum(correct_words)}\n"
      f"Number of words: {len(correct_words)}\n")

print(f"Partial measure of accuracy: {sum(partial_correct_words) /
len(partial_correct_words):.2f}\n"
      f"Partial correct words: {sum(partial_correct_words):.2f}\n"
      f"Partial uncorrect words: {len(partial_correct_words) -
sum(partial_correct_words):.2f}\n"
      f"Number of words: {len(partial_correct_words)}\n")
```

Kôd 12: programska skripta za računanje mjere točnosti

8. Rezultati i analiza uspješnosti programske skripte

Ovaj dio dokumentacije uključuje opise i interpretaciju dobivenih rezultata programske skripte. Odnosi se na riječi koje je algoritam rastavio iz skupa riječi za testiranje.

Rezultat dobiven izračunom mjere točnosti potpuno rastavljenih riječi iznosi: 0.78. Odnosno 78% uspješnosti.

```
Measure of accuracy: 0.78
Correct words: 56
Uncorrect words: 16
Number of words: 72
```

Kôd 13: mjera točnosti potpuno rastavljenih riječi – izlaz iz terminala

Dok rezultat dobiven izračunom mjere točnosti djelomično rastavljenih riječi iznosi: 0.87. Odnosno 87% uspješnosti što je za 9% više u odnosu na prethodni rezultat.

```
Partial measure of accuracy: 0.87
Partial correct words: 62.53
Partial uncorrect words: 9.47
Number of words: 72
```

Kôd 14: mjera točnosti djelomično rastavljenih riječi – izlaz iz terminala

Algoritam ne daje mjeru točnosti od 100% i teško da će se postići takva točnost upravo zbog korištenog modula (compound-split) za rastavljanje složenica u njemačkom jezičnom području. Postoji još jedan modul (german_compound_splitter) koji rastavlja složenice s puno većom točnošću jer koristi njemački rječnik koji obuhvaća više riječi. Navedeni modul nije implementiran jer zadatak nalaže korištenje compound-split modula.

Također, unutar algoritma implementirana su sva pravila koja su u 3. poglavlju navedena („Lingvistička pravila za njemački jezik“).

```
(accuracy) PS D:\Faks - OIRI\Diplomski\PZRZ\Projekt\word_syllables_accuracy> & "d:/Faks - OIRI/Diplomski/PZRZ/Projekt/word_syllables_accuracy/env/Scripts/python.exe" "d:/Faks - OIRI/Diplomski/PZRZ/Projekt/word_syllables_accuracy/test_accuracy.py"
```

Test word	Verify word	Algorithm output	Correct word	Percent of partial word accuracy
Diät	Di ät	Di ät	yes - all	1.00
Knie	Knie	Knie	yes - all	1.00
die knie	die kni e	die kni e	yes - all	1.00
der knie	der kni e	der kni e	yes - all	1.00
reise knie	rei se kni e	rei se kni e	yes - all	1.00
Auto	Au to	Au to	yes - all	1.00
Seeufer	See ufer	Seeu fer	no - 0 of 2	0.00
Katze	Kat ze	Kat ze	yes - all	1.00
Tatze	Tat ze	Tat ze	yes - all	1.00
Pfütze	Pfüt ze	Pfüt ze	yes - all	1.00
putzen	put zen	put zen	yes - all	1.00
platzen	plat zen	plat zen	yes - all	1.00
Bürste	Bürs te	Bürs te	yes - all	1.00
Kiste	Kis te	Kis te	yes - all	1.00
Hamster	Hams ter	Hams ter	yes - all	1.00
Fenster	Fens ter	Fens ter	yes - all	1.00
hinstellen	hin stel len	hin stel len	yes - all	1.00
darstellen	dar stel len	dar stel len	yes - all	1.00
erstarren	er star ren	er star ren	yes - all	1.00
plötzlich	plötz lich	plötz lich	yes - all	1.00
Postauto	Post au to	Post au to	yes - all	1.00
Kratzbaum	Kratz baum	Kratz baum	yes - all	1.00
boxen	bo xen	bo xen	yes - all	1.00
heben	he ben	he ben	yes - all	1.00
rodeln	ro deln	ro deln	yes - all	1.00
Schifffahrt	Schiff fahrt	Schiff fahrt	yes - all	1.00
Mussspiel	Muss spiel	Muss spiel	yes - all	1.00
wichtigsten	wich tigs ten	wich tigs ten	yes - all	1.00
besuchen	be su chen	be suc hen	no - 1 of 3	0.33
gewinnen	ge win nen	ge win nen	yes - all	1.00
vergessen	ver ges sen	ver ges sen	yes - all	1.00
abangeln	ab an geln	ab an geln	yes - all	1.00
Kreuzotter	Kreuz ot ter	Kreuz ot ter	yes - all	1.00
poetisch	po e tisch	po e tisch	yes - all	1.00
Nationen	Na ti o nen	Na ti o nen	yes - all	1.00
aber	a ber	ab er	no - 0 of 2	0.00
über	ü ber	ü ber	no - 2 of 2	1.00
Kreuzklemme	Kreuz klem me	Kreuz klem me	yes - all	1.00
Foxtrott	Fox trott	Fox trott	yes - all	1.00
witzlos	witz los	witz los	yes - all	1.00
witzig	wit zig	wit zig	yes - all	1.00
wegschmeißen	weg schmei ßen	weg schmei ßen	yes - all	1.00
Bettüberzug	Bett ü ber zug	Bett ü ber zug	yes - all	1.00
wirtschaft	wirt schaft	wirtschaft	no - 0 of 2	0.00
Beziehungsknatsch	Be zieh ungs knatsch	Be zie hungsknatsch	no - 1 of 4	0.25
Gletscher	Glet scher	Glet scher	yes - all	1.00
Wurstscheibe	Wurst schei be	Wurst schei be	yes - all	1.00
Borretschgewächs	Bor retsch ge wächs	Bor retsch ge wächs	yes - all	1.00
Bodden	Bod den	Bod den	yes - all	1.00
Handball	Hand ball	Hand ball	yes - all	1.00
Neubau	Neu bau	Neu bau	yes - all	1.00
Stalltür	Stall tür	Stall tür	yes - all	1.00
Autobahnanschlussstelle	Au to bahn an schluss stel le	Au to bahn an schlussstel le	no - 5 of 7	0.71
Laufschuhe	Lauf schu he	Lauf schu he	yes - all	1.00
Baustelle	Bau stel le	Baus tel le	no - 1 of 3	0.33
Lebkuchen	Leb ku chen	Leb kuc hen	no - 1 of 3	0.33
Himbeere	Him bee re	Him bee re	yes - all	1.00
Klassenzimmer	Klas sen zim mer	Klas sen zim mer	yes - all	1.00
Hubbleteleskop	Hub ble te le skop	Hubb le te les kop	no - 2 of 5	0.40
Botschaft	Bot schaft	Bot schaft	yes - all	1.00
Schokoladenfabrik	Scho ko la den fa brik	Scho ko la den fab rik	no - 4 of 6	0.67
Hühnersuppe	Hüh ner sup pe	Hüh ner sup pe	yes - all	1.00
Schweinebraten	Schwei ne bra ten	Schwei ne bra ten	yes - all	1.00
Halsschmerzen	Hals schmer zen	Hals schmer zen	yes - all	1.00
Weltanschauung	Welt an schau ung	Welt an schauung	no - 2 of 4	0.50
Weltschmerz	Welt schmerz	Welt schmerz	yes - all	1.00
Weihnachtsbaum	Weih nachts baum	Weih nach tsbaum	no - 1 of 3	0.33
Kugelschreiber	Ku gel schrei ber	Ku gel schrei ber	yes - all	1.00
Bohnensalat	Boh nen sa lat	Boh nen sa lat	yes - all	1.00
Freundschaftsbezeigung	Freund schafts be zei gung	Freundschaftsbe zei gung	no - 2 of 5	0.40
Weihnachtsmannfigur	Weih nachts mann fi gur	Weih nach tsmann fi gur	no - 3 of 5	0.60
Glasflächenreinigung	Glas fläch en rei ni gung	Glas fläc hen rei ni gung	no - 4 of 6	0.67
---	---	---	correct: 62.53 of 9.47	0.87

Slika 9: Tablični prikaz kako je algoritam rastavio riječi

Zaključak

U ovoj projektnoj dokumentaciji objašnjeno je što je to algoritam, koje su karakteristike i kako dizajnirati dobar algoritam. Navedena su lingvistička pravila koja je stručna osoba, odnosno lingvist za njemačko jezično područje dostavio. Opisan je programski kôd algoritma i moduli koji su korišteni zajedno sa osnovnim uputama za korištenje algoritma.

Za kreiran algoritam, izrađena je web aplikacija i korisničko sučelje gdje korisnik može odabrati željeni opciju za rastavljanje teksta. Unos teksta ručno pomoću tipkovnice ili unos teksta korištenjem .txt datoteke. Web aplikacija sadrži i opciju koju korisnik može odabrati kako bi dobio uvid u točnost samog algoritma na danom skupu za testiranje.

Za kraj je kreirana skripta za testiranje točnosti algoritma. Algoritam je testiran na skupu riječi koje je lingvist dostavio, a u svrhu mjerenja točnosti algoritma. Skripta za testiranje točnosti algoritma mjeri potpunu točnost i djelomičnu točnost. Bolji rezultat mjerenja točnosti postiže se testiranjem djelomične točnosti, a rezultat je bolji za 9% u odnosu na mjerenje potpune točnosti.

Prilikom izrade ovog algoritma može se zaključiti da se točnost od 100% teško postiže iako su sva pravila implementirana u algoritam. Razlog tome je što modul compound-split ne rastavlja dovoljno dobro njemačke složenice. Kako bi potencijalno riješili taj problem potrebno je trenirati model na složenicama koje se nalaze u testnome skupu, koristiti german_compound_splitter modul ili ručno napraviti iznimke.

Popis slika

Slika 1: algoritam prikazan na slikovni način 2

Slika 2: slikovni prikaz karakteristika algoritma 3

Slika 3: primjer rastavljanja složenice u jednokorijenske riječi 5

Slika 4: prikaz rastavljene rečenice 20

Slika 5: prikaz početne stranice web sučelja 22

Slika 6: prikaz korisničkog sučelja kad korisnik pritisne gumb „Input as a text.“ 22

Slika 7: prikaz korisničkog sučelja kad korisnik pritisne gumb „Input as a .txt file.“ 23

Slika 8: web stranica koja prikazuje mjeru točnosti algoritma 24

Slika 9: Tablični prikaz kako je algoritam rastavio riječi 31

Popis kôdova

Kôd 1: prikaz uključenih modula 8

Kôd 2: globalne varijable i globalne liste 10

Kôd 3: funkcija slyllabels_rules_exceptions 13

Kôd 4: funkcija sentence_in_vc 15

Kôd 5: funkcija syllabels_rules 18

Kôd 6: primjer prosljeđivanja varijable koja je tipa string u funkciju 19

Kôd 7: primjer pokretanja funkcija u tri kruga 19

Kôd 8: prikaz skupa testnih riječi 25

Kôd 9: prikaz testnih riječi onako kako trebaju biti rastavljene 26

Kôd 10: prikaz kôda za uvoz datoteke imena algorithm.py u datoteku test_accuracy.py 26

Kôd 12: moduli koji su korišteni za izradu programske skripte 27

Kôd 13: programska skripta za računanje mjere točnosti 29

Kôd 14: mjera točnosti potpuno rastavljenih riječi – izlaz iz terminala 30

Kôd 15: mjera točnosti djelomično rastavljenih riječi – izlaz iz terminala 30

Popis naredba

Naredba 1: postavljanje virtualne okoline i instalacija modula compound-split..... 9

Naredba 2: instalacija zasebno korištenih modula 27

Literatura

[1] W3SCHOOLS, Python Lists, https://www.w3schools.com/python/python_lists.asp, siječanj 2023.

[2] W3SCHOOLS, Python – Global Variables, https://www.w3schools.com/python/python_variables_global.asp, siječanj 2023.

[3] W3SCHOOLS, Python Functions, https://www.w3schools.com/python/python_functions.asp, siječanj 2023.

[4] Python, re – Regial expression operations, https://docs.python.org/3/library/re.html, siječanj 2023.

[5] Regex Pal, Regex tester, https://www.regexpal.com/, siječanj 2023.

[6] GeeksForGeeks, Creating Tables with PrettyTable Library – Python, https://www.geeksforgeeks.org/creating-tables-with-prettytable-library-python/, siječanj, 2023.

[7] Replit, How to Use Termcolor In Python, https://replit.com/talk/learn/How-to-Use-Termcolor-In-Python/24684, sijačanj 2023.

[8] Csatlas, Python 3: Import Another Python File as a Module, https://csatlas.com/python-import-file-module/, siječanj 2023.

[9] Wikipedia, Njemački standardni jezik, https://hr.wikipedia.org/wiki/Njema%C4%8Dki_standardni_jezik, siječanj 2023.

[10] Wikipedia, German verbs, https://en.wikipedia.org/wiki/German_verbs, siječanj 2023.

[11] Intellecta, Njemačke složenice – nisu tako strašne!, https://www.intellecta.hr/jezicne-zanimljivosti/njemacke-slozenice/, siječanj 2023.

[12] Silbentrennung, slična web aplikacija za rastavljanje na slogove, https://www.silbentrennung24.de/, siječanj 2023.

[13] GitHub, JoelNiklaus CompoundSplit, https://github.com/JoelNiklaus/CompoundSplit, siječanj 2023.

[14] GitHub, repodiac german_compounder_splitter, https://github.com/repodiac/german_compound_splitter, siječanj 2023.

Prilog: programski kod

URL na GitHub repozitorij:

https://github.com/ajanach/word_syllables