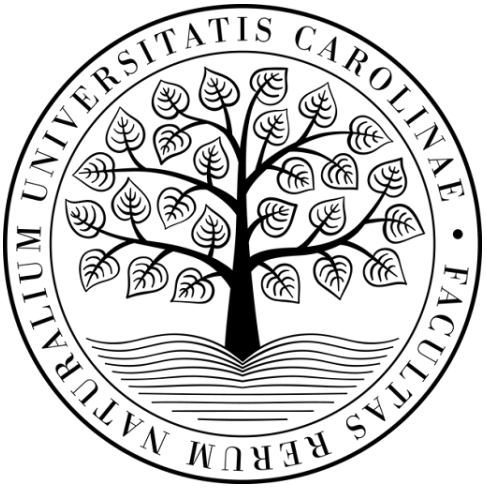
Univerzita Karlova

Prírodovedecká fakulta



Daniel Kalakay

2. ročník študijného programu Sociálna geografia a geoinformatika

Výpočet četností znaků v textu.

Dokumentácia k skúškovému programu na predmet:

Úvod do programování

Akademický rok 2025/2026

Praha 20. 1. 2026

Obsah

Úvod a stručné zadanie

Táto práca sa zaoberá návrhom, vypracovaním a následným zdokumentovaním programu pre výpočet absolútnej a relatívnej početnosti znakov vo vstupnom texte zadaného užívateľom. Celý program bol zostrojený v jazyku Python a v prostredí PyCharm, využíva princípy objektovo orientovaného programovania (OOP).

Cieľom práce je vytvoriť program, ktorý je prehľadný, rozšíriteľný a užívateľsky jednoduchý na používanie. Program by mal podľa zadania analyzovať text na základe predom určených písmen a znakov Slovenskej abecedy. Úspešne by mal filtrovať nepodporované znaky bez zbytočného výpisu a výsledky by mal zobrazovať od najpočetnejšieho znaku.

Presné zadanie úlohy

Pre vstupný text zahrňujúci písmena "A-Ž", "a-ž", číslovky "0-9" a špeciálne znaky ",.!?;" oddelené medzerami vypočítajte absolútnu a relatívnu početnosť jednotlivých znakov. Výsledok vo formáte znak / absolútna početnosť / relatívna početnosť vypíšte zostupne podľa hodnôt absolútnych početností. Ak sa v texte budú nachádzať nepodporované znaky, ignorujte ich.

Analýza úlohy

Úlohu môžeme jednoduchšie preformátovať do formy analýzy textového reťazca v ktorom je nutné zapracovať nasledovné parametre:

- Rozlíšiť podporované znaky od nepodporovaných
- Zosumarizovať finálnu početnosť znaku
- Vypočítať relatívnu početnosť znaku vzhľadom k celému textu
- Zoradiť následné výsledky zostupne
- Vypísať výsledky

Z obecného zadania pre riešenie každého príkladu, používať OOP, je určite najvýhodnejšie zabaliť celý proces do jednej triedy, ktorá následne ponesie zodpovednosť za spravovanie všetkých dát, či už to je výpočet početností, uchovávanie týchto dát a následné vypísanie výsledkov. Možno jedinou výhodou tohto prístupu je menšie zaťaženie celkového procesu, jednoduchšia manipulácia s dátami pri prípadnom rozširovaní programu a globalizácia použitých premenných.

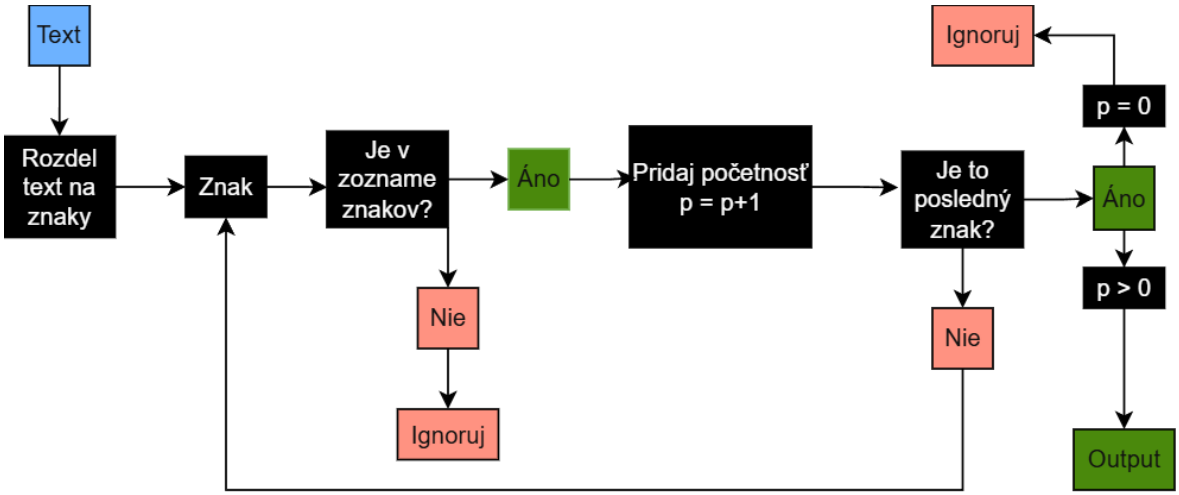
Voľba algoritmu

Návrh objektu

Celý náš program bude postavený okolo našej triedy „Analyzátor", ktorá reprezentuje celý proces rozdelený do sub-častí ako napríklad „Hlavná funkcia" a „Výpis". Trieda obsahuje prázdnu premennú text do ktorej sa dodá vstup od užívateľa, množinu všetkých podporovaných znakov podľa zadania, slovník pre ukladanie absolútnej početnosti jednotlivých znakov a premennú uchovávajúcu celkový počet spracovaných znakov. Jednotlivé sub-časti odpovedajú logickým krokom navrhnutých v časti analýza úlohy.

Použité postupy a algoritmy

Text náš program spracováva znak po znaku pomocou sekvenčného priechodu celým reťazcom. V jednoduchosti by sa dal celý program graficky ukázať nasledovne:



Táto schéma zjednodušene ukazuje ako zhruba náš program pracuje. Každý znak je porovnávaný s množinou podporovaných znakov. Ak je znak súhlasný so znakom, ktorý náš program podporuje zvýši sa jeho početnosť v našom slovníku. Relatívnu početnosť program rieši až na konci celého programu a to ako podiel absolútnej početnosti znaku a celkového počtu znakov v našom slovníku, prepočítavaný je na percenta.

$$\text{RelativnaPoc} = \frac{\text{AbsolutnaPoc}}{\text{ZnakSpolu}} * 100$$

Najjednoduchšie je si celú problematiku ukázať na jednoduchom príklade, napríklad máme podporované znaky : ***M, m, a***

1. Používateľ zadá text: „***Mama!?***“
2. Program prevedie text na znaky nasledovne: ***'M'***, ***'a'***, ***'m'***, ***'a'***, ***'!'***, ***'?'***
3. Následne začne postup sekvenčného triedenia
4. Zoberie si znak s indexom 0, v našom prípade písmeno ***M***
5. Skontroluje či je v zozname podporovaných znakov
6. Znak sa v zozname nachádza, v našom slovníku sa mu zvýši hodnota o 1
7. {'M':1}
8. Pokračuje ďalším znakom ***a***
9. {'M':1, 'a':1}
10. Takýmto postupom prejde celý reťazec až sa dostaneme k výsledku
11. {'M':1, 'a':2, 'm':1}
12. Znaky ***!*** a ***?*** odignoruje keďže sa nenachádzajú medzi podporovanými
13. Vypočíta relatívnu početnosť
14. Vypíše výsledok

Diskusia k výberu algoritmu

Keď som sa na príklad prvotne pozrel povedal som si že celú túto problematiku dokáže vyriešiť štandardná knižnica v Pythone, konkrétne Counter. Toto riešenie by bolo oveľa rýchlejšie, menej transparentné a didakticky neprínosné Hlavne by nesplňovalo by generálne zadanie spracovať program pomocou OOP. Vo zvolenom algoritme je jednoduchšie od sledovať celý proces spracovania dát a poskytuje výrazne väčšiu kontrolu nad výberom znakov.

Program

Výsledný kód používa triedu Analyzator, ktorá zastrešuje všetku funkcionalitu potrebnú pre analýzu textu. Hlavnou dátovou štruktúrou je slovník, v ktorom sú uložené absolútne početnosti jednotlivých znakov. Množina podporovaných znakov slúži k efektívnej kontrole vstupného textu. Štruktúra celého programu je veľmi jednoduchá, celý program zaberá do 40 riadkov. Hlavný program je o to jednoduchší, zoberie iba vstup od užívateľa, zavolá funkciu na výpočet početnosti a následne metódu pre výpis výsledkov.

```
class Analyzator:
    def __init__(self, text):
        self.text = text
        self.pocetnost = {}
        self.support = set(
            "AÁÄBCČDĎEÉFGHIÍJKLĽĹMNŇOÓÔPQRŔSŠTŤUÚVWXYÝZŽ"
            "aáäbcčdďeéfghiíjklľĺmnňoóôpqrŕsštťuúvwxyýzž"
            "0123456789"
            ",.?!;"
        )
        self.celk_poc = 0

    def hlavnafunkcia(self):
        self.pocetnost.clear()
        self.celk_poc = 0

        for znak in self.text:
            if znak in self.support:
                self.pocetnost[znak] = self.pocetnost.get(znak, 0) + 1
                self.celk_poc += 1

    def vypis(self):
        zoradene = sorted(
            self.pocetnost.items(),
            key=lambda x: x[1],
            reverse=True
        )

        print("Znak / Absolútna / Relatívna")


        for znak, abs_cetnost in zoradene:
            relativna = abs_cetnost / self.celk_poc
            relativna=relativna*100
            print(f"{znak} / {abs_cetnost} / {relativna:.1f}% textu")

txt = input("Zadaj svoj text: ")
analyzator = Analyzator(txt)
analyzator.hlavnafunkcia()
analyzator.vypis()
```

Alternatívne prístupy

Možnou, autorom uprednostňovanou, metódou by bolo klasické procedurálne riešenie bez použitia tried, program by používal iba funkcie a globálne premenné. Také riešenie by bolo určite menej prehľadné a ťažko rozšíriteľné ale za to rýchlejšie na naprogramovanie a možno časovo úspornejšie. Ďalšou metódou by mohlo byť vytvorenie separátnych tried, napríklad vstup, výpočet a výstup. Vzhľadom k jednoduchosti zadania by to bolo zbytočne komplikované a zdĺhavé.

Vstupné dáta

Vstupné dáta užívateľ vpisuje do vo forme riadkov ako súvislý textový reťazec. Na dĺžke ani na medzerách v texte nezáleží, autor sa snažil vytvoriť tzv. „blbovzdorný“ program. Užívateľ nemusí text špeciálne upravovať, program ignoruje všetky znaky, ktoré nie sú podporované automaticky.

Výstupné dáta

Výstup programu je vypisovaný do konzoly vo forme riadkov obsahujúcich znak, absolútnu početnosť a relatívnu početnosť v percentách. Relatívna početnosť ukazuje, aký podiel z celkového počtu podporovaných znakov v texte daný znak tvorí. Výstup je zoradený zostupne, najčastejšie znaky sú uvedené ako prvé.

Priebeh práce

Práca na programe prekvapivo začala na papieri, autor nakreslil schému a podľa nej sa riadil po celú polhodinu programovania celého programu. Počas procesu autor vybral vhodnú štruktúru a následne implementoval hlavnú funkciu pre výpočtu početností a výstupu.

Po dokončení programu bol program testovaný na rôznych sadách vstupných textov a boli dokončené úpravy formátu výstupu.

Čo by bolo vhodné pridať

Autor sa tuho zamyslel nad touto otázkou a prišiel k výsledku že možný výber užívateľa vlastného textového súboru by sa do pokročilejšieho programu zišla, rovnako možno aj užívateľský výber podporovaných znakov. Nezabudnúť netreba ale aj na ukladanie výstupných dát, napríklad, do externého súboru.

Záverečný povzdych

Práca na tomto dokumente autora maximálne vyčerpala, pár krát si počas vytvárania programu pomocou OOP kládol otázku či sa nejedná o snahu používať naberačku na polievku ku konzumácii malého jogurtu. Na druhú stranu ale môžeme konštatovať že výsledný kód vyzerá aspoň kultivovane. Tvorba tohto dokumentu je zabrala asi až priveľa času na to aký je program triviálny. Celkovo by som tento zážitok označil za viac frustrujúci ako uspokojivý, poučný určite ale je. Autor sa naučil, alebo v to aspoň dúfa, písať dokumentáciu k programom a vie že už by to znovu nerobil. Musí uznať že forma vyjadrovania doktora Kryla na neho urobila dobrý dojem.