Otázka 4 – Výpočet četností znaků v textu

1. Rozbor problému

Program je určen pro analýzu frekvence jednotlivých znaků v zadaném řetězci (zadán uživatelem). Cílem programu je vytisknout frekvenci každého znaku ve formátu „znak: frekvence“ v sestupném pořadí. Program ignoruje určité znaky, které se sice běžně vyskytují ve větách, nicméně jejich počítání je většinou spíše nechtěné – čárka, tečka, mezera atd.

2. Existující algoritmy

Existuje několik existujících algoritmů, které řeší tento problém. Nejjednodušším je ruční výpočet frekvencí znaků v řetězci, kdy se daný řetězec prochází znak po znaku a počítá se kolikrát se každý znak vyskytuje. Dalším způsobem je použití funkcí v rámci Python knihoven, například Counter, která automaticky počítá frekvenci jednotlivých prvků.

3. Popis zvoleného algoritmu

Pro můj program jsem si zvolil jednodušší algoritmus, který řetězec prochází znak po znaku a počítá frekvenci každého znaku. Pokud se daný znak do doby jeho kontroly v programu ještě neobjevil, tak mu program přiřadí hodnotu 1, pokud už se tam objevil nkrát, tak mu přiřadí hodnotu n+1. Zároveň jsou zde i znaky, u kterých program frekvenci nezjišťuje, respektive je ignoruje.

4. Struktura programu

Program se skládá z jedné hlavní třídy ‘**CharFrequencyAnalyzer’**, která v sobě obsahuje metody ‚**__init__**‘, ‚**char_freq**‘, ‚**print_char_freq**‘, ‚**get_valid_data**’ a ‚**main**‘:

- ‚**__init__(self, string)**‘ – inicializátor, který přijímá vstupní řetězec a ukládá ho do atributu ‚**self.string**’. Tato metoda také definuje seznam znaků, které budou vyřazeny – ‚**self.unwanted_chars**’ a take kapitalizuje vstupní string (k → K);
- ‚**char_freq(self)**‘, která počítá frekvenci všech znaků vstupního řetězce a ukládá ji do slovníku ‚**char_freq**’. Poté tato metoda znaky ještě seřadí podle frekvence a uloží je do atributu ‚**self.char_freq**’;
- ‚**print_char_freq(self, char_freq)**‘ si bere jako vstup ‚**char_freq**’ a vytiskne frekvenci znaků ve formátu „znak: frekvence“, a to v sestupném pořadí. Program nerozlišuje velikost znaků, tedy například při zadání stringu ‘Kk’ bude výsledkem ‘K: 2’;
- ‚**get_valid_data**’ žádá uživatele o vstup a kontroluje správnost tohoto vstupu. Specificky nejdřív zkontroluje, že vůbec uživatel zadal nějaký vstup. Pokud nezadal, tak program uživateli do terminálu vypíše chybu o prázdném vstupu a zeptá se uživatele o nový vstup. Pokud uživatel něco zadal, ale jednalo se pouze o znaky, které jsou v proměnné ‘**unwanted_chars**’, tak program uživateli vypíše chybovou hlášku o zadání nevhodných znaků a opět ho vyzve k zadání nového vstupu;
- ‚**main(self)**‘ volá ‚**get_valid_data()**’, **char_freq()**‘ a ‚**print_char_freq(char_freq)**‘.

Na úplném konci programu jsou dva řádky, ve kterých je nejdříve vytvořená instance třídy ‚**CharFrequencyAnalyzer**' s prázdným řetězcem jako vstupní hodnotu. V druhém řádku je poté volána metoda main této instance, která zajišťuje volání metod ‚**get_valid_data()**', **char_freq()**' a ‚**print_char_freq(char_freq)**'. Tedy v těchto dvou řádcích je zajišťován průchod celého programu.

5. Vstupní a výstupní data

Vstupními daty programu je řetězec zadaný uživatelem. Musí se jednat o korektní vstup, viz 4. Struktura programu – metoda ‚**get_valid_data**'.

Výstupem programu je seřazený seznam znaků a jejich frekvence v zadaném řetězci. Tento seznam je vytištěn do terminálu. V současné verzi se pouze uživateli ukáže a tam skončí, neukládá se do žádného souboru nebo do nějaké proměnné.

6. Problematická místa

Největším problémem jsou '**unwanted_chars**', respektive jejich statická podoba. V momentální podobě se jedná o ručně zapsané výjimky, nicméně by bylo bláhové se domnívat, že to podchytilo všechny znaky, které by bylo vhodné ignorovat. Odlišnou a v něčem lepší, ač zároveň také v něčem limitující variantou by byla aplikace a případná modifikace vybrané lokální abecedy přes knihovnu re. Díky tomu by byly žádané znaky snadněji modifikovatelné, ale byla by méně praktická pro znaky, které se využívají jen v některých jazycích. Mým cílem bylo především vyřadit běžně se vyskytující znaky, které jsou využívány napříč jazyky užívající latinku, spíše než začlenit velké množství znaků, které jednotlivé jazyky užívají a zároveň není vhodné je vyřadit.

Ospravedlnil jsem si to především tím, že často se vyskytujících nechtěných znaků bude méně než znaků jako š, ř a podobně v různých jazycích, jelikož tečka, čárka a podobné znaky jsou relativně univerzální. Samozřejmě ale záleží na tom pro jakého uživatele by byl program určen. Pokud by to bylo pouze pro uživatele užívající ten samý jazyk, tak by aplikace lokální knihovny byla praktičtější pro případné úpravy. Pokud má být ale program univerzální, tak by aplikace knihovny a přidávání všech možných znaků specifických pro různé jazyky bylo velmi nepřehledné a v případě hodně znaků by to mohlo být i neefektivní, protože to prochází celý tento seznam a hledá shodu.

7. Možná vylepšení

Vylepšení tohoto programu je opravdu nespočet, takže vypíšu především ty, které považuji za nejvhodnější a zároveň za relativně jednoduše implementovatelné:

1) implementace abecedy pomocí knihovny re, ale problémy viz 6. Problematická místa,
2) implementace metody pro ukládání výsledků analýzy do souboru nebo databáze pro další užití,
3) přidání možnosti zadání více vstupních řetězců najednou a jejich srovnání,
4) automatizace seznamu nežádoucích znaků,
5) možnost vyhledávání konkrétních znaků v analýze,
6) grafické rozhraní, pro pohodlí uživatele (jak pro vstup, tak pro vizualizaci výsledků).

Otázka 94 – Výpočet modusu pro nesetříděnou posloupnost tvořenou n prvky

1. Rozbor problému

Problém, který tento program řeší, je nalezení nejčastěji se vyskytujícího prvku (modus) v CSV souboru. Program správně reaguje na situaci, kdy je vstupní soubor prázdný nebo neexistuje na příslušném místě na disku.

2. Existující algoritmy

Pro nalezení modusu v souboru lze využít více přístupů, včetně jejich kombinace:

- jedním z nejčastějším je použití knihovny Counter z modulu collections, která počítá výskyt jednotlivých prvků v seznamu a vrací seznam dvojic (prvek: počet výskytů),
- využití slovníku k uchování počtu výskytů jednotlivých prvků a následné vyhledání prvku s nejvyšším počtem výskytů
- použití nějakého algoritmu pro řazení (Bubble sort, Merge sort, Timsort a další) pro seřazení prvků a následné procházení seznamu pro nalezení nejčastěji se opakujícího prvku. Algoritmů na řazení je opět více, zde jen rychle popíšu výhody a nevýhody několika z nich:
    - **Bubble sort**: prochází seznam několikrát a porovnává každý prvek se svým sousedem – pokud jsou ve špatném pořadí, tak prvky prohodí. To se opakuje, dokud není seznam seřazen. Algoritmus je jednoduchý na implementaci a snadno pochopitelný, ale kvůli tomu jak funguje je nevhodný pro velké datasety;
    - **Selection sort**: prochází seznam a vybírá nejmenší prvek, který přesune na první pozici. Tento proces se opakuje, dokud není seznam seřazen. Algoritmus je sice rychlejší než Bubble sort, ale stále je kvůli svému fungování nevhodný na velké datasety.
    - **Insertion sort**: prochází seznam a vkládá každý prvek na správné místo v již seřazené části seznamu. Je rychlejší než Bubble sort a Selection sort pro menší datasety, ale pro velká data je stále pomalý;
    - **Merge sort**: využívá principu dělení a spojování. Rozdělí seznam na dva podseznamy, které se řadí zvlášť. Ty se poté spojí do jednoho seřazeného seznamu a tento proces se opakuje, dokud seznam není seřazen. Jedná se o stabilní algoritmus, tedy zachovává pořadí dvou totožných hodnot podle toho v jakém pořadí byly v původním seznamu. Pro velké datové sady je výrazně efektivnější než všechny výše zmíněné algoritmy, ale existují i rychlejší algoritmy;
    - **Quick sort**: využívá principu „rozděl a panuj“. Vybere se hlavní prvek a všechny ostatní se rozdělí na dvě skupiny – prvky menší než hlavní prvek a prvky větší než hlavní prvek. Proces se opakuje pro obě skupiny do té doby, než je seznam seřazen. Jedná se o velmi efektivní algoritmus pro řazení velkých dat, ale má vyšší složitost a není stabilní (nezachovává pořadí);

- **Timsort**: jedná se o hybridní algoritmus, který kombinuje přístup Merge sort a Insertion sort. Nejdříve využívání třídění přímým výběrem (Insertion) pro menší data, a poté třídění sloučením pro velké části dat. Nakonec používá třídění sloučením pro velké části dat, které jsou seřazeny pomocí přímého výběru. Ve finále se jedná o algoritmus, který je relativně efektivní pro malé i velké datové sady, a zároveň je jednoduchý na implementaci (v Python je to defaultní algoritmus pro třídění).

3. Popis zvoleného algoritmu

V programu bylo využito více algoritmů, pro různé části programu. Pro účely čtení a zpracování CSV souboru se používá knihovna CSV. Pro účely hledání nejčastějšího prvku (modusu) se používá algoritmus Bubble sort (výhody a nevýhody v 2. Existující algoritmy) pro seřazení dat a následně Counter z knihovny collections pro počítání frekvence jednotlivých prvků. Kontrola, zda vůbec soubor existuje a zda není prázdný se provádí v metodě ‚**open_file()**' a algoritmy jsou implementovány v rámci třídy ‚**MyFileProcesso**r'.

4. Struktura programu

Struktura programu se skládá z několika hlavních částí. První z nich je třída ‚**MyFileProcessor**', která zajišťuje zpracování vstupního souboru. Tato třída obsahuje funkce ‚**open_file**', ‚**prepare_data**', ‚**bubble_sort**', ‚**modus_finder**' a ‚**print_modus**':

- ‚**open_file**' slouží k otevření a kontrole vstupního souboru, přičemž se kontroluje, zda soubor existuje a zda není prázdný. Pokud soubor neexistuje nebo je prázdný, program se ukončí s příslušnou chybovou hláškou, odlišnou zdali se jedná o prázdný soubor nebo soubor nebyl nalezen. Poté tento seznam seznamů vrací jako proměnnou ‚**data**' pro další užití;
- ‚**prepare_data(data)**' bere proměnnou data jako vstupní proměnnou a slouží k přeměně seznamu seznamů na jeden velký seznam. Tentokrát vrací proměnnou ‚**data_sorted**';
- ‚**bubble_sort(data_sorted)**' slouží k seřazení dat pomocí Bubble sort algoritmu. Jako vstupní proměnnou bere proměnnou ‚**data_sorted**';
- ‚**modus_finder(data_sorted)**' slouží k výpočtu modusu (nejčastěji se vyskytující hodnoty) pomocí knihovny Counter. Výsledkem je dvojice modus a hodnota modusu, to poté vrací jako dvě proměnné - ‚**most_common**' a ‚**frequency**';
- ‚**print_modus(most_common, frequency)**' slouží k výpisu výsledků, vstupy jsou ‚**most_common**' a ‚**frequency**'
- ‚**main**' poté spouští celý program. Nejprve vyzve uživatele k zadání jména vstupního souboru, a pokud se jedná o validní vstup, tak vytvoří instanci ‚**MyFileProcessor**'

5. Vstupní a výstupní data

Vstupními daty tohoto programu je CSV soubor, který uživatel zadá při spuštění programu. Program bude pokračovat jen v případě, že splní podmínky, tedy že existuje a že není prázdný.

Výstupními daty je nejčastější hodnota v souboru a počet výskytu této hodnoty. Tyto hodnoty jsou vypsány do terminálu, nejsou ukládány do jiného CSV souboru. Nicméně v průběhu programu dochází k ukládání nejčastější hodnoty (‚**most_common**') a počet výskytu této hodnoty (‚**frequency**'), takže by šlo program snadno rozšířit a tyto proměnné dále využít.

6. Problematická místa

Prvotní úpravou, kterou zde vidím je především třídící algoritmus. Ač je Bubble sort jednoduchý a často užívaný algoritmus, tak je velmi neefektivní pro velké datové sady a může dojít k dlouhému trvání programu. Pro mé užití byl více než dostatečně efektivní, ale pro velké datové sady by to byl problém. Pokud nevíme jak velký datový soubor může být využit, tak lze vcelku bezpečně použít defaultní Timsort (ten nebyl užit kvůli tomu, aby byla ukázána implementace jiného než defaultního algoritmu), případně Merge sort nebo Quick sort.

Dalším zefektivněním by mohlo být použití konkrétního typu dat, tedy jaký typ dat může být předán do programu a jaký typ dat bude vrácen z programu. Ač program v momentální verzi očekává CSV soubor jako vstupní data, ale nekontroluje konkrétní typ dat uvnitř tohoto souboru. Momentálně je používán seznam seznamů, což ale může být neefektivní a komplikované pro další zpracování dat.

7. Možná vylepšení

Možných vylepšení je opět hodně, takže zmiňuji ty nejdůležitější:

1) použití rychlejšího třídícího algoritmu, buď timsort jakožto univerzálního algoritmu pokud nevíme s jakými daty bude uživatel pracovat nebo specifického algoritmu vhodného pro daný dataset;
2) implementace kontroly typu dat vstupního souboru
3) použití knihovny Pandas pro zpracování CSV souborů – práce s daty může díky Pandas být snažší a efektivnější, programátoři navíc tuto knihovnu používají velmi často a je tedy pravděpodobnější, že jí budou rozumět. Na druhou stranu ale využití externí knihovny limituje využití uživateli, kteří Pythonu příliš nerozumí, navíc když se tato externí knihovna musí doinstalovat;
4) implementace funkce pro výpis více než jednoho modusu pro případ, že existují dva a více prvky, které mají stejný počet výskytu v souboru;
5) implementace grafického uživatelského rozhraní pro snadnější ovládání programu
6) implementace možnosti zpracovávat vstupní data z různých typů souborů