Dokumentace k úlohám z předmětu Úvod do programování

Autor:
Vít Suchomel, 3. ročník, SGGI

Úloha 1

1. Zadání

Cílem úlohy je vytvořit program, který ve vstupním řetězci nalezne nejkratší a nejdelší slovo a určí počet jeho znaků. Vstupní text může obsahovat písmena „A-Ž“, „a-ž“, číslice „0-9“ a speciální znaky „,.?!;“ oddělené mezerami. Pokud existuje více slov se stejnou délkou, program vypíše všechna v pořadí výskytu. Nepodporované znaky (např. interpunkce nalepená na slova) mají být ignorovány (odstraněny), aby nezkreslovaly délku slov.

2. Popis a rozbor problému

Problém spadá do kategorie zpracování textových řetězců. Hlavní výzvou je správné očištění vstupu – tedy oddělení samotných slov od interpunkce a číslic, které nejsou považovány za součást slova.

Vstupní řetězec je nutné nejprve rozdělit na jednotlivé slova podle oddělovače (mezera). Následně je třeba každé slovo očistit od nežádoucích znaků. V tomto řešení jsou za nežádoucí znaky považovány interpunkční znaménka a číslice, jak je implementováno v seznamu *znaky_k_orezani*.

Po získání seznamu validních slov se jedná o problém nalezení extrémů (minima a maxima) v poli, s tím specifikem, že nehledáme pouze jednu hodnotu, ale všechna slova odpovídající této hodnotě.

3. Existující algoritmy

Pro řešení tohoto problému existují dva hlavní přístupy

1. Průchodový: text je čten po znacích a pamatuje si jen aktuální maximum a minimum. Tento přístup šetří paměť počítače, ale je složitější.

2. Načtení do paměti: text je rozdělen na slova (tokeny), které jsou uloženy do seznamu, ze kterého jsou následně vybrány extrémy (nejdelší a nejkratší slova). Tento přístup je jednodušší, ale paměťově náročnější. Pro velké textové soubory by nebyl vhodný.

4. Popis zvoleného algoritmu

Zvolený algoritmus pro řešení úlohy je založen na sekvenčním zpracování a filtraci textových dat v paměti počítače. Proces lze rozdělit do tří hlavních fází: tokenizace, normalizace a vyhledávání extrémů.

1. Tokenizace a hrubé očištění: Vstupní řetězec je nejprve rozdělen na menší celky (tokeny) podle oddělovače, kterým je v tomto případě mezera (nebo posloupnost bílých znaků). Tím získáme prvotní seznam potenciálních slov.
2. Normalizace a validace: Algoritmus následně prochází každý token a provádí jeho očištění. Z okrajů tokenu jsou odstraněny interpunkční znaménka a číslice, které nejsou součástí lexikálního významu slova. Takto upravený token je podroben validaci – je přijat pouze tehdy, pokud obsahuje výhradně písmena abecedy. Tímto krokem jsou vyloučena samostatná čísla nebo slova obsahující speciální znaky uvnitř.
3. Vyhledávání extrémů: Nad seznamem validních slov je provedeno vyhledání globálního extrému (maximální a minimální délka). Vzhledem k požadavku na vypsání *všech* slov dané délky v původním pořadí, nehledá algoritmus pouze první výskyt. Místo toho nejprve zjistí hodnotu délky (např. 15 znaků) a následně vyfiltruje všechna slova, která této délce odpovídají.

5. Struktura programu

Program je implementován v jazyce Python a je navržen pomocí objektově orientovaného programování (OOP). To umožňuje jasné oddělení aplikační logiky, která se stará o zpracování dat, od uživatelského rozhraní zajišťujícího vstup a výstup.

Jádrem aplikace je třída *AnalyzaTextu*, která zapouzdřuje veškeré operace nad vstupním řetězcem. Instance této třídy uchovává v atributu *self.text* původní znění textu a v atributu *self.slova* seznam již validních, očištěných lexikálních jednotek připravených k analýze.

Při inicializaci instance (metoda *__init__*) dochází k transformaci vstupních dat. Řetězec je nejprve rozdělen na jednotlivé tokeny podle bílých znaků. Následně program iteruje přes tyto tokeny a provádí jejich normalizaci. Z každého tokenu jsou odstraněny nežádoucí znaky definované v proměnné *znaky_k_orezani*, která kombinuje interpunkční znaménka z modulu *string* a číslice 0–9. Pokud výsledný řetězec obsahuje pouze písmena abecedy, je přidán do interního seznamu slov.

Samotné vyhledávání extrémů zajišťují metody *najdi_nejdelsi()* a *najdi_nejkratsi()*. Obě metody pracují na identickém principu: nejprve detekují hraniční délku (maximum či minimum) v celém seznamu slov a následně pomocí generátorové notace vyfiltrují a vrátí všechna slova, která této délce odpovídají. Tento postup zaručuje, že v případě shody délek nedojde ke ztrátě dat a uživateli jsou prezentována všechna relevantní slova. Metody vracejí n-tici obsahující seznam nalezených slov a jejich číselnou délku.

Řídící část programu (main) se nachází mimo definici třídy. Zajišťuje interakci s uživatelem prostřednictvím standardního vstupu a instancuje třídu *AnalyzaTextu*. Důležitou součástí této vrstvy je ošetření stavu, kdy uživatel zadá text bez platných slov. V takovém případě program, namísto vyvolání výjimky při hledání extrémů nad prázdným seznamem, vypíše informativní hlášení a ukončí se. Výsledky analýzy jsou formátovány pomocí f-stringů pro přehledný výpis na standardní výstup.

6. Problematické situace a ošetření

Primárním problémem je validita vstupu. Pokud by uživatel zadal řetězec, který neobsahuje žádná slova (např. prázdný řetězec, pouze mezery nebo řadu čísel), standardní metody pro hledání min/max hodnot by selhaly. Tento stav je v programu ošetřen podmínkou kontrolující délku seznamu *analyza.slova*. V případě prázdného seznamu program nevypisuje chybové hlášky interpretu, ale informuje uživatele o neplatném vstupu a korektně se ukončí.

Druhou, komplexnější problematikou, je definice toho, co tvoří "slovo". Algoritmus využívá metodu *strip()* pro odstranění interpunkce z okrajů, což funguje spolehlivě pro většinu běžných případů (např. slovo na konci věty s tečkou). Nicméně, slova obsahující speciální znaky uvnitř (např. složená slova typu "ne-moc" nebo e-mailové adresy) narážejí na striktní kontrolu metodou *isalpha()*. V aktuální implementaci bylo rozhodnuto tato slova považovat za neplatná a do analýzy je nezahrnovat, aby nedocházelo ke zkreslení délky slova započítáním nepísmenných znaků.

7. Vstupní a výstupní data

Interakce s uživatelem probíhá prostřednictvím standardního konzolového vstupu a výstupu. Vstupem je libovolný textový řetězec zadaný na jedné řádce, který může obsahovat diakritiku, číslice i interpunkci. Program tento vstup načítá pomocí funkce *input()*.

Výstupem aplikace je strukturovaný textový výpis. V případě úspěšné analýzy program nejprve zopakuje analyzovaný text pro kontrolu. Následně jsou na samostatných řádcích vypsány výsledky ve formátu: seznam nalezených slov (oddělený čárkami, jde-li o seznam) a jejich číselná délka. Díky využití f-stringů je výstup přehledný a dynamicky formátovaný podle obsahu proměnných *vysledek_nejdelsi* a *vysledek_nejkratsi*.

Příklad vstupních a výstupních dat je vidět na obrázku 1.

```
PS C:\Škola\_Úvod do programování\Zkouška> & "C:/Users/Vít Suchomel/AppData/Local/Mic
hon3.11.exe" "c:/Škola/_Úvod do programování/Zkouška/ukol_1_slova.py"
Vložte text, ve kterém chcete vyhledat nejdelš a nejkratší slovo: Ahoj světe! 123.
Analyzovaný text: Ahoj světe! 123.
Nejdelší slovo je/jsou: ['světe']. Počet znaků: 5
Nejkratší slovo je/jsou: ['Ahoj']. Počet znaků: 4
PS C:\Škola\_Úvod do programování\Zkouška>
```

Obrázek 1 – Příklad vstupních a výstupních dat

8. Závěr a možná vylepšení

Cílem práce bylo vytvořit, pokud možno, robustní nástroj pro základní textovou analýzu, konkrétně pro extrakci nejdelších a nejkratších slov. Výsledná aplikace splňuje všechny body zadání. Zvolený algoritmus, který lineárně prochází text a separuje slova od interpunkce, se ukázal jako efektivní pro běžné délky textů.

Během testování se potvrdilo, že největší výzvou textové analýzy je nejednoznačnost definice "slova" v přirozeném jazyce, zejména v kontextu speciálních znaků uvnitř výrazů (např. složené výrazy se spojovníkem). Současná implementace volí bezpečný přístup a taková slova ignoruje, pokud nesplňují podmínku *isalpha()*.

Možným rozšířením by mohla být implementace opatření, která by umožnily analyzovat i slovo jako je například „R2D2“, „je-li“ apod. Současný stav programu tyto slova ignoruje a považuje za chybná. Dále by bylo vhodné rozšířit program o možnost načítání textu ze souboru (*.txt*), což by umožnilo analýzu rozsáhlejších textů, a případně doplnit statistiku četnosti jednotlivých slov.

Literatura

BAYER, T.: Programování 9 – Třídy a objekty. Přednáškové materiály. Přírodovědecká fakulta Univerzity Karlovy, Praha, https://web.natur.cuni.cz/~bayertom/images/courses/Prog1/programovani9.pdf (28. 1. 2026).

GEEKSFORGEEKS (2025): Python String Methods. GeeksforGeeks.org, https://www.geeksforgeeks.org/python-string-methods/ (cit. 28. 1. 2026).

PILGRIM, M. (2012): Ponořme se do Pythonu 3. CZ.NIC, https://diveintopython3.py.cz/index.html (28. 1. 2026).

PYLADIES CZ (2017): Kurz Pythonu pro začátečníky – Brno jaro 2017. Naucse.python.cz, https://naucse.python.cz/2017/pyladies-brno-jaro-po/ (28. 1. 2026).

REAL PYTHON (2025): Python's .split(): The Efficient Way to Split Strings. Realpython.com, https://realpython.com/python-split-string/ (28. 1. 2026).

Zdrojový kód programu

```
#Hledání nejkratšího a nejdelšího slova
#Vít Suchomel, 3. ročník, SGGI
#zimní semestr 2025/26
#Úvod do programování

import string

class AnalyzaTextu:

  #Třída pro zpracování a analýzu textového řetězce.
  #Očišťuje vstupní text od interpunkce a číslic.
  #Vyhledává nejdelší a nejkratší slova.

  def __init__(self, zadany_text):

    #Funkce rozděluje text na slova (tokeny).
    #Ukládá očištěná slova do seznamu self.slova

    self.text = zadany_text
    neocistena_slova = zadany_text.split()

    self.slova = []

    znaky_k_orezani = string.punctuation + "0123456789"

    for slovo in neocistena_slova:
      ciste_slovo = slovo.strip(znaky_k_orezani)
```

```
        if len(ciste_slovo) > 0 and ciste_slovo.isalpha():
            self.slova.append(ciste_slovo)

    def najdi_nejdelsi(self):

        #Funkce hledá nejdelší slova v seznamu.
        ##Pokud je seznam prázdný, vrací [], 0.

        if not self.slova:
            return [], 0
        max_delka = max(len(slovo) for slovo in self.slova)

        nejdelsi = [slovo for slovo in self.slova if len(slovo) == max_delka]
        return nejdelsi, max_delka

    def najdi_nejkratsi(self):

        #Funkce hledá nejkratší slova v seznamu.
        #Pokud je seznam prázdný, vrací [], 0.

        if not self.slova:
            return [], 0

        min_delka = min(len(slovo) for slovo in self.slova)
        nejkratsi = [slovo for slovo in self.slova if len(slovo) == min_delka]

        return nejkratsi, min_delka
```

```
#Hlavní část programu, zajišťující jeho běh

vlozte_text = input("Vložte text, ve kterém chcete vyhledat nejdelš a nejkratší slovo: ")
analyza = AnalyzaTextu(vlozte_text)

if len(analyza.slova) == 0:
   print("Chyba: Zadal jste neplatný text (žádná platná slova).")

else:

   vysledek_nejdelsi, max_delka = analyza.najdi_nejdelsi()
   vysledek_nejkratsi, min_delka = analyza.najdi_nejkratsi()

   print(f"Analyzovaný text: {vlozte_text}")
   print(f"Nejdelší slovo je/jsou: {vysledek_nejdelsi}. Počet znaků: {max_delka}")
   print(f"Nejkratší slovo je/jsou: {vysledek_nejkratsi}. Počet znaků: {min_delka}")
```