

Języki Skryptowe

dokumentacja projektu Tataraki i balonik

Kamil Ptak, grupa 4/8

15 grudnia 2022

Część I

Opis programu

W pliku słownik.txt znajduje się słownik, w którym słowa (każde w nowej linii) posortowane są alfabetycznie.

Napisz program, który dla zadanej początkowej litery słowa, poszukiwał będzie w tym słowniku wyrazów, które da się podzielić na pewnym miejscu w ten sposób, że zarówno do miejsca podziału jak i od miejsca podziału (tę część czytamy wspak), tak powstałe słowa również znajdują się w tym słowniku.

Przykładowo, jeśli podalibyśmy jako argument literę t, to program mógłby znaleźć słowo tataraki, bo dzieląc je po czwartej literze, otrzymamy słowa tata i ikar, a dla litery b, program mógłby zwrócić słowo balonik (podział po trzeciej literze na słowa bal i kino).

Zakładamy dodatkowo, że zarówno poszukiwane słowo, jak i jego składowe, są co najmniej dwuliterowe.

Program ma zwracać wszystkie wyrazy spełniające warunki zadania (na zadaną literę początkową).

Instrukcja obsługi

Aby uruchomić program należy włączyć skrypt menu.sh otwierający menu obsługi naszego programu. Po uruchomieniu wyświetli nam się tekst z instrukcją obsługi programu, wymagający podania przez użytkownika liczby w celu wykonania odpowiadającej mu funkcji. Możliwe wybory są następujące:

1. Uruchom program - Uruchamia program pobierając przy tym wszystkie dane z katalogu input i tworzy raport.html, który jest następnie wyświetlany w domyślnej przeglądarce systemowej
2. Wyświetl informacje - Wypisuje na ekranie konsoli opis założeń programu
3. Backup - Tworzy kopię zapasową danych w katalogu backups zawierającą raport.html oraz zawartość folderów input i output
4. Zakoncz - Zamyka menu, kończąc tym samym program.

Struktura danych programu

Program składa się z następującej struktury danych, wymaganych do prawidłowego uruchomienia aplikacji:

- menu.sh - Skrypt bash będący menu, którym uruchamia się program, wyświetla informacje o programie jak i tworzy kopie zapasową danych otrzymanych w wyniku wykonania tegoż programu

- tataraki.py - Skrypt python zawierający główny program, pobierający pliki wejściowe zawierające literę początkową słowa i tworzący plik wyjścia zawierające słowa z pliku słownik.txt, które są zgodne z założeniami projektu

- raport.py - Skrypt python pobierający dane z plików wejścia oraz wyjścia i generujący plik raport.html zawierający raport wszystkich danych w postaci tabeli

- Katalog wordbases zawierający bazowo tylko jeden plik:

- słownik.txt - Plik tekstowy zawierający szeroki zbiór słów ze słownika języka Polskiego

- Katalog resources zawierający:

- info.txt - Plik tekstowy zawierający opis programu, wyświetlany z poziomu menu.sh po wybraniu odpowiedniej opcji

- menu.txt - Plik tekstowy zawierający opis korzystania z menu.sh, wyświetlany na każdym etapie korzystania z programu

- style.css - Plik kaskadowego arkusza styli wykorzystywany podczas generowania raportu w celu stylizacji tegoż raportu, jak i ukazania danych w sposób właściwy

- Katalog input zawierający pliki wejściowe, nazwane według klucza:

$$input0.txt, input1.txt \dots inputX.txt \quad (1)$$

gdzie X to kolejna liczba rzeczywista.

Część II

Opis działania

Skrypt `menu.sh` pobiera kolejno wszystkie nazwy plików wejściowych z katalogu `input` i przekazuje je pojedynczo jako argument do `tataraki.py`. Następnie dla każdej pobranej nazwy program otwiera podany plik oraz czytuje z niego pojedynczą literę, która jest przypisywana do zmiennej `keyletter`. Aplikacja pobiera słowa z pliku `słownik.txt` do listy `wordbank`, oraz tworzy drugą listę zawierającą odwrócone wersje tych słów nazwaną `reversedWordbank`. Następnie tworzona jest kolejna lista zawierająca jedynie te słowa ze `słownik.txt`, które rozpoczynają się na tą samą literę co ta podana w pliku wejściowym, nazwana `keywords`, oraz pobiera z niej długość najdłuższego słowa `longest_word`.

Program następnie łączy ze sobą wszystkie słowa z list `keywords` i `reversedWordbank`, których łączna długość nie przekracza długości `longest_word`, i sprawdza czy znajdują się one w zbiorze `keywords`. Jeśli tak, to wpisywane są one do pliku `temp_wbuffer.txt` wraz ze słowami składowymi, przy czym drugie słowo jest ponownie odwracane.

Finalnie program kolejno wypisuje zawartość pliku `temp_wbuffer.txt` do pliku `output` o tym samym numerze co ten znajdujący się w nazwie pliku `input`.

Otrzymane w ten sposób wyniki są następnie przetwarzane przez `raport.py`, który tworzy plik `raport.html`, który w tabeli umieszcza zarówno zawartość plików `input` oraz `output`, tylko i wyłącznie jeśli numer w ich nazwie jest taki sam, co sprawia, że raport zawiera jedynie pomyślnie wykonane iteracje programu.

Ostatecznie uruchamiana jest domyślna przeglądarka użytkownika, w której wyświetla się utworzony `raport.html`

Algorytm

Data: Dane wejściowe plik *input*

Result: Dane wyjściowe plik *output*

Do *keyletter* przypisz wartość *input*

Do *wordbank* wczytaj zawartość pliku *słownik.txt*

Do *reversedWordbank* przypisz odwrócone słowa z *wordbank*

Do *keywords* przypisz słowa z *wordbank* zaczynające się na *keyletter*

Do *longest_word* przypisz długość najdłuższego słowa z *keywords*

Utwórz *buforx*, *bufory*

```
for x in keywords do
  for y in reversedWordbank do
    if długość(x+y) <= longest_word and x+y in keywords then
      if Jesli x znajduje sie w buforx i y znajduje sie w bufor then
        | continue
      else
        | Dopisz do pliku temp_wbuffer: x+y x odwrócone(y)
        | Dopisz x do buforx
        | Dopisz y do bufory
      end
    end
  end
end

if temp_wbuffer nie jest pusty then
  | Przypisz do output zawartość temp_wbuffer
end
```

Algorithm 1: Algorytm wyszukiwania słów zaczynających się na daną literę, składających się ze zwykłego słowa i słowa pisanego wspak.

Implementacja systemu

Skrypt menu.sh przy uruchomieniu, jak i po każdym odświeżeniu pobiera i wypisuje z pliku "menu.txt" znajdującego się w katalogu "resources" instrukcje obsługi menu aplikacji.

Uruchomienie programu z poziomu skryptu menu.sh powoduje sprawdzenie czy istnieje katalog na pliki wynikowe "output", jeśli tak to usuwa go wraz z zawartością oraz tworzy go na nowo, następnie uruchamia tataraki.py wraz z argumentem, którym jest nazwa pliku wejścia z katalogu "input". Jeśli program znalazł wynik i zakończył się powodzeniem, tworzy on plik wynikowy do katalogu "output". Następnie usuwany jest plik "temp_wbuffer.txt" o ile takowy istnieje oraz sprawdzane jest czy w katalogu "output" znajdują się jakiekolwiek pliki. Jeśli tak to uruchamiany jest skrypt raport.py, który pobiera dane odnośnie stylizowania strony z pliku "style.css" znajdującego się w katalogu "resources", tworzy plik raport.html w głównym katalogu projektu, a jeśli nie to program kończy się z odpowiednim komunikatem.

Opcja wyświetlająca informacje wypisuje dane z pliku "info.txt" znajdującego się w katalogu "resources".

Wybranie funkcji backup sprawdza czy istnieje w głównym katalogu plik raport.html, jeśli tak to tworzy katalog "backups" o ile takowy już nie istnieje, następnie tworzy katalog nazwany aktualną datą i godziną wewnątrz folderu "backups", który natomiast zawiera kopię całego katalogu "input", katalogu "output" oraz pliku raport.html.

Funkcje zawarte w tataraki.py

```
1 //Funkcja zwracająca podane słowo "word" napisane wstecz
2 def reverse(word):
3     return word[::-1]
4
5 //Funkcja pobierająca nazwę błędu "errorname", kończąca program
   przedwcześnie i wypisująca w konsoli odpowiedni komunikat
6 def errorcodes(errorname):
7     if errorname == "keyletter":
8         sys.exit(f"{input} - Podana wartość nie jest litera")
9     if errorname == "input":
10        sys.exit("Nie znaleziono pliku input")
11    if errorname == "dictionary":
12        sys.exit("Nie znaleziono pliku słownikowego")
13    if errorname == "empty":
14        sys.exit(f"{input} - Nie znaleziono słowa zaczynającego się
           na ta literę")
15    if errorname == "nooutput":
16        sys.exit(f"{input} - Nie znaleziono wyników spełniających
           warunki")
```

Funkcje zawarte w raport.py

Raport.py zawiera klasę HTMLCreator, posiadającą następujące metody w celu sprawniejszego wygenerowania raportu.html

```
1     def __init__(self): //Przy inicjacji klasy usuwa stary raport.html
2         if exists("raport.html"):
3             os.remove("raport.html")
4
5         //Tworzy listy plikow input i output
6         self.outputfiles = [file for file in listdir("output") if isfile
7                             (join("output", file))]
8         self.inputfiles = [file for file in listdir("input") if isfile(
9                             join("input", file))]
10
11        //Tworzy podstawowe pliki html
12        now = datetime.now()
13        self.fulldate = now.strftime("%d-%m-%Y %H:%M:%S")
14        self.css = open("resources/style.css", "r")
15        self.html = open("raport.html", "w")
16        self.html.write(f"""<!DOCTYPE html>
17            <html>
18            <head>
19                <title>Raport z dnia {self.fulldate}</title>
20                <style>
21                    {self.css.read()}
22                </style>
23            </head>
24            <body>
25                <div class="container">
26                    <h1>Raport z dnia {self.fulldate}</h1>
27                    <table>
28                        <tr>
29                            <th>Input</th>
30                            <th>Output</th>
31                        </tr>\n""")
32
33        //Tworzy znacznik html, ktory jest automatycznie domykany
34        def doubletag(self, tag, content=""):
35            self.html.write(f"<{tag}>{content}</{tag}>\n")
36
37        //Tworzy znacznik html bez domknięcia
38        def singletag(self, tag):
39            self.html.write(f"<{tag}>\n")
```

Testy

Dane wejściowe:

input0.txt = b

input1.txt = t

input2.txt = !

input3.txt = k

input4.txt = X

input5.txt = B Dane wyjściowe:

output0.txt = balonik - bal kino

output1.txt = tataraki - tata ikar

output3.txt = kartonik - kart kino

```
-----
                        Menu
-----
1. Uruchom program
2. Wyświetl informacje
3. Backup
4. Zakończ
-----

input2.txt - Podana wartosc nie jest litera
input4.txt - Nie znaleziono slowa zaczynajacego sie na ta litere
input5.txt - Nie znaleziono slowa zaczynajacego sie na ta litere

Program uruchomiony pomyslnie, otwieram wygenerowany raport
```

Rysunek 1: Wynik programu w konsoli

Raport z dnia 15-12-2022 11:12:25	
Input	Output
t	tataraki - tata ikar
b	balonik - bal kino
k	kartonik - kart kino

@Kamil Ptak 2022r.

Rysunek 2: Utworzony raport

Eksperymenty

Podczas realizacji z uwagi na duży początkowy rozmiar pliku słownik.txt, próbowałem zastosować kilka różnych algorytmów i sposobów przechowywania danych tymczasowych wymaganych do wykonania programu, w celu zoptymalizowania czasu jaki aplikacja wymagała, aby się wykonać.

Pierwotnie program przechowywał na osobnych listach pierwsze słowo (firstword), drugie słowo (secondword), oraz całe słowo(fullword), a algorytm wyglądał następująco:

```
1     for x in fullword:
2         for y in firstword:
3             if len(y) > 1 and len(y) <= len(x)-2:
4                 temp_word = fullword.replace(firstword, "")
5                 for z in secondword:
6                     if len(z) > 1 and len(z) == len(temp_word):
7                         print(f"{x} - {y} {z}")
```

Skutkowało to długim czasem wykonywania z powodu zagnieżdżonych pętli i dużej złożoności czasowej.

Następną próbą realizacji było przechowywanie wszystkich możliwych kombinacji pierwszego słowa i drugiego słowa w pliku, a następnie sprawdzanie czy znajduje się on w zbiorze wordlist, co jednak skutkowało dużym zużyciem przestrzeni dyskowej powodując zatrzymanie programu z powodu braku miejsca na dysku. Przechowanie tych samych danych w postaci list również nie odniosło skutku, gdyż zużywało to za dużo pamięci RAM, również skutkując niepowodzeniem programu.

Szukając alternatywnych sposobów na przechowanie danych, próbowałem zastosować set oraz dictionary zamiast list, jednakże mimo szybszych operacji nie pozwalają one na przechowywanie duplikatów, co sprawia, że program nie wyświetlał wszystkich możliwych rozwiązań.

Pełen kod aplikacji

tataraki.py

```
1  # -*- coding: utf-8 -*-
2  import os
3  import sys
4  import re
5
6  input = str(sys.argv[1])
7
8
9  def reverse(word):
10     return word[::-1]
11
12
13  def errorcodes(errorname):
14     if errorname == "keyletter":
15         sys.exit(f"{input} - Podana wartosc nie jest litera")
16     if errorname == "input":
17         sys.exit("Nie znaleziono pliku input")
18     if errorname == "dictionary":
19         sys.exit("Nie znaleziono pliku slownikowego")
20     if errorname == "empty":
21         sys.exit(f"{input} - Nie znaleziono slowa zaczynajacego sie na
22             ta litere")
23     if errorname == "nooutput":
24         sys.exit(f"{input} - Nie znaleziono wynikow spelniajacych
25             warunki")
26
27
28  database = "wordbases/slownik.txt"
29  wordbank = [] # zawiera wszystkie slowa
30  reversedWordbank = [] # zawiera wszystkie slowa odwrocone
31
32  file_number = ""
33  for letters in input:
34     if letters.isdigit():
35         file_number += letters
36
37  try:
38     keyletter = open(f"input/{input}", "r")
39     keyletter = keyletter.read(1)
40  except:
41     errorcodes("input")
42
43  if not bool(re.match('^[a-zA-Z]*$', keyletter)):
44     errorcodes("keyletter")
45
46  try:
47     read = open(database, "r")
48  except:
49     errorcodes("dictionary")
```

```

49 for x in read:
50     x = x.strip()
51     if len(x) > 1:
52         wordbank.append(x)
53
54 read.close()
55
56 for x in wordbank:
57     reversedWordbank.append(reverse(x))
58
59 keywords = [] # zawiera same slowa zaczynajace sie na keyletter
60 for x in wordbank:
61     if x.startswith(keyletter):
62         keywords.append(x)
63
64 try:
65     longest_word = len(max(keywords, key=len))
66 except:
67     errorcodes("empty")
68
69 wordsbuffer = open(f'temp_wbuffer.txt', 'w')
70 buforx=[]
71 bufory=[]
72
73 for x in keywords:
74     for y in reversedWordbank:
75         if len(x + y) <= longest_word and x + y in keywords:
76             if x in buforx and y in bufory:
77                 pass
78             else:
79                 wordsbuffer.write(x + y + " " + x + " " + reverse(y) + "
80                                     \n")
81                 buforx.append(x)
82                 bufory.append(y)
83
84 wordsbuffer.close()
85
86 if os.stat("temp_wbuffer.txt").st_size == 0:
87     errorcodes("nooutput")
88
89 count = 0
90 with open(r"temp_wbuffer.txt", 'r') as fp:
91     for count, line in enumerate(fp):
92         pass
93
94 wordsbuffer_len = count + 1
95 wordsbuffer = open(f'temp_wbuffer.txt', 'r')
96 outputfile = open(f"output/output{file_number}.txt", "w")
97
98 for x in range(0, wordsbuffer_len):
99     line = wordsbuffer.readline().strip()
100     word = line.split(" ")[0]
101     firstw = line.split(" ")[1]
102     secondw = line.split(" ")[2]
103     outputfile.write(f"{word} - {firstw} {secondw}\n")

```

```
103
104 wordsbuffer.close()
105 os.remove("temp_wbuffer.txt")
106 outputfile.close()
```

raport.py

```
1 import os
2 from datetime import date, datetime
3 from os import listdir
4 from os.path import isfile, join, exists
5
6
7 class HTMLCreator:
8     def __init__(self):
9         if exists("raport.html"):
10             os.remove("raport.html")
11
12         self.outputfiles = [file for file in listdir("output") if isfile(
13             join("output", file))]
14         self.inputfiles = [file for file in listdir("input") if isfile(
15             join("input", file))]
16
17         now = datetime.now()
18         self.fulldate = now.strftime("%d-%m-%Y %H:%M:%S")
19         self.css = open("resources/style.css", "r")
20         self.html = open("raport.html", "w")
21         self.html.write(f"<!DOCTYPE html>
22             <html>
23             <head>
24                 <title>Raport z dnia {self.fulldate}</title>
25                 <style>
26                     {self.css.read()}
27                 </style>
28             </head>
29             <body>
30                 <div class="container">
31                     <h1>Raport z dnia {self.fulldate}</h1>
32                     <table>
33                         <tr>
34                             <th>Input</th>
35                             <th>Output</th>
36                         </tr>\n"")
37
38     def doubletag(self, tag, content=""):
39         self.html.write(f"<{tag}>{content}</{tag}>\n")
40
41     def singletag(self, tag):
42         self.html.write(f"<{tag}>\n")
43
44 raport = HTMLCreator()
45
46 for x in range(len(raport.outputfiles)):
47     output_number = ""
48     for letters in raport.outputfiles[x]:
49         if letters.isdigit():
50             output_number += letters
51
52     for y in range(len(raport.inputfiles)):
```

```

52     input_number = ""
53     for letters in raport.inputfiles[y]:
54         if letters.isdigit():
55             input_number += letters
56
57     if input_number == output_number:
58         raport.singletag("tr")
59         inputfile = open(f"input/{raport.inputfiles[y]}", "r")
60         raport.doubletag("td", f"{inputfile.read()}")
61
62         raport.singletag("td")
63         outputfile = open(f"output/{raport.outputfiles[x]}", "r")
64         lines = outputfile.readlines()
65         for index, singleline in enumerate(lines):
66             raport.doubletag("div", f"{singleline.strip()}")
67             raport.singletag("/td")
68
69         raport.singletag("/tr")
70
71 raport.singletag("/table")
72 raport.doubletag("footer", "@Kamil Ptak 2022r.")
73 raport.singletag("/div")
74 raport.singletag("/body")
75 raport.singletag("/html")
76 raport.html.close()

```

menu.sh

```
1 #!/bin/bash
2 stop(){
3     echo
4     read -p "Wcisnij enter by kontynuowac" enter
5     menu
6 }
7
8 menu(){
9     clear
10    while read -r line
11    do
12        echo "$line"
13    done < resources/menu.txt
14
15    read -n 1 -s wybor;
16
17    case $wybor in
18    "1")#Uruchom program
19        dir="output"
20        if [ -d $dir ]
21        then
22            rm -r $dir
23        fi
24        mkdir $dir
25        echo
26        for file in "input"/ *
27        do
28            file="${file:6}"
29            python tataraki.py "$file"
30        done
31        if [ -f "temp_wbuffer.txt" ];
32        then
33            rm temp_wbuffer.txt
34        fi
35        if [ -d $dir ]
36        then
37            if [ "$(ls -A $dir)" ]; then
38                python raport.py
39                echo
40                echo "Program uruchomiony pomyslnie, otwieram wygenerowany
41                    raport"
42                xdg-open raport.html </dev/null >/dev/null 2>&1 & disown
43                #not empty
44            else
45                echo "Brak wynikow dzialania programu"
46                #empty
47            fi
48        fi
49        stop
50        ;;
51    "2")#Wyswietl informacje
52        echo
53        while read -r line
```

```

53     do
54     echo "$line"
55     done < resources/info.txt
56
57     stop
58     ;;
59 "3")#Backup
60     if [ -f "raport.html" ]; then
61         if [ ! -d "backups" ];
62         then
63             mkdir backups
64         fi
65
66         printf -v date '%(%d-%m-%Y-%H:%M:%S)T\n' -1
67         mkdir backups/$date
68
69         cp -R input backups/$date
70         cp -R output backups/$date
71         cp raport.html backups/$date
72
73         echo "Backup zostal utworzony pomyslnie"
74         stop
75     else
76         echo "Nie znaleziono pliku raport.html"
77     fi
78     stop
79     ;;
80 "4")#Wyjdz
81     clear
82     exit
83     ;;
84 *)#Niepoprawne polecenie
85     echo "Wprowadzono niepoprawne polecenie"
86     stop
87     ;;
88 esac
89 }
90
91 menu

```
