Przygotowanie uczniów do matury z informatyki poprzez opracowanie algorytmów zadania 4.2 na maturze z 2018 r.

Roland Zimek

Zadanie 4.2.

Znajdź słowo, w którym występuje największa liczba różnych liter. Wypisz to słowo i liczbę występujących w nim różnych liter. Jeśli słów o największej liczbie różnych liter jest więcej niż jedno, wypisz pierwsze z nich pojawiające się w pliku z danymi.

Dla danych z pliku `przyklad.txt` wynikiem jest:
`AKLMNOPRSTWZA 12`

Przypomnę, że zgodnie z główną treścią zadania każdy wiersz zawiera jedno niepuste słowo złożone z wielkich liter alfabetu angielskiego. Długość jednego słowa nie przekracza 100 znaków.

Jeśli podejdziemy do tego zadania na szybko bez zastanowienia, to możemy wpaść w pułapkę. Przez co namęczymy się trochę nad programem. Na czym polega pułapka? Na tym, że będziemy chcieli zliczyć ile razy występuje każda z litera języka angielskiego, a następnie sprawdzić dla ilu liter otrzymaliśmy wartość większą od zera. Przypomnę, że alfabet angielski składa się z 26 liter: `ABCDEFGHIJKLMNOPQRSTUVWXYZ`.

Oto przykładowy kod programu przy założeniu chyba najgorszego scenariusza (dla uproszczenia zliczmy na razie „tylko” litery `ABCZ`:

```
plik = open("przyklad.txt", "r")
tekst = plik.readlines()
znakA = znakB = znakC = znakZ = 0

for i in range(len(tekst)):
    wiersz = tekst[i]
```

```
    for j in range(len(wiersz)):
        znak = wiersz[j]
        if znak == "A":
            znakA += 1
        elif znak == "B":
            znakB += 1
        elif znak == "C":
            znakC += 1
        ### ... Pozostałe znaki
        elif znak == "Z":
            znakZ == +1
plik.close()
```

Oczywiście program w takiej postaci zlicza łącznie poszczególne znaki w całym tekście, nie sprawdzając który wiersz ma ich najwięcej. Ale już tutaj widać na czym polega błąd w podejściu do tego zadania. Uzupełnienie powyższego programu o zliczanie każdej z liter spowoduje bezsensowne rozrośnięcie się warunku `if…elif`.

Zanim przejdę do dalszej analizy programu, chciałbym zwrócić uwagę na dwa fragmenty. Pierwszy z nich to specyficzna w języku Python deklaracja zmiennych (którą oczywiście należałoby uzupełnić o pozostałe znaki alfabetu):

```
znakA = znakB = znakC = znakZ = 0
```

Ten wiersz tworzy cztery zmiennie, przypisując dla każdej z nich wartość 0. Jak widać jest to bardzo wygodny sposób.

Drugi fragment to cztery wiersze:

```
for i in range(len(tekst)):
    wiersz = tekst[i]
    for j in range(len(wiersz)):
        znak = wiersz[j]
```

W programie Python można ten zapis uprościć, wykorzystując ukierunkowanie tego języka programowania do pracy na listach (tekst też jest odmianą listy, składającej się z wielu liter).

```
for wiersz in tekst:
    for znak in wiersz:
```

Powyższy zapis powoduje pobieranie po kolei każdego elementu listy i wykorzystanie go w pętli. Dlaczego nie zastosowałem tego rodzaju konstrukcji przy rozwiązaniu zadania 4.1? Należało bowiem złączyć odpowiednie litery począwszy od 40 znaku, a takie skrócone rozwiązanie nie daje możliwości sprawdzenia dla którego elementy z kolei jest wykonywane. Można by to obejść stosując zmienną pomocniczą zliczającą ile razy dana pętla się wykonuje, ale jest to bardzo nieeleganckie rozwiązanie i ponownie komplikuje program. Na szczęście w języku Python istnieje odpowiednie rozwiązanie. Chętnym proponuję zapoznanie się z funkcją `enumerate()`.

Wrócę jednak do analizy zadania. Po chwili zastanowienia się można zauważyć, że zamiast sprawdzać każdy znak po kolei, łatwiej będzie odczytać kod ASCII danego znaku i zwiększać wartość odpowiedniego elementu wcześniej utworzonej listy[1]. Kody znaków od A do Z, to wartości od 65 do 90. Nie ma sensu oczywiście tworzyć listy 90. elementowej, wystarczy bowiem 26. elementowa (będę odczytany kod ASCII zmniejszał o 65, dzięki czemu dla litery A użyty zostanie element listy o indeksie `0`. Kod ASCII znaku odczytami dzięki funkcji `ord(`*`znak`*`)`.

Przykład wykorzystujący powyższe rozważania:

```
plik = open("przyklad.txt", "r")
tekst = plik.readlines()
ASCII = [0] * 26

for i in range(len(tekst)):
    wiersz = tekst[i]
```

[1] Zamiast zwiększać wartość elementu listy o 1, można także po prostu przypisać wartość 1. Nie interesuje nas bowiem ile razy dany znak występuje, tylko czy w ogóle występuje.

```
    for j in range(len(wiersz)-1):
        znak = wiersz[j]
        ASCII[ord(znak)-65] += 1
print(ASCII)

plik.close()
```

W wyniku otrzymamy liczby oznaczające ile razy dana litera wystąpiła w analizowanym pliku. I tak litera A pojawiła się łącznie 1203 razy, B 3 razy itd.:

[1203, 3, 2, 1, 3, 0, 0, 1, 3, 0, 1, 1, 2, 2, 2, 2, 0, 3, 2, 3, 1, 0, 1, 0, 0, 982]

Powyższy program nie jest rozwiązaniem analizowanego zadania 4.2, gdyż należałoby dokonać także zliczenia dla każdego wiersza osobno, sprawdzić ile wartości jest niezerowych i na bieżąco zapamiętywać właściwy wiersz.

Warto jeszcze zwrócić uwagę na nieodczytywanie ostatniego znaku w wierszu `for j in range(len(wiersz)-1):`. Ostatnim znakiem jest bowiem polecenie przejścia do nowego akapitu mający kod ASCII 10, co po pomniejszeniu o 65 spowoduje próbę odwołania się do elementu listy o indeksie -55. Oczywiście spowoduje to błąd.

Jak w takim razie podejść do rozwiązania zadania 4.2, skoro zliczanie kolejnych znaków od A do Z nie jest dobrym rozwiązaniem? Można po kolei sprawdzać czy dana litera wystąpiła już wcześniej w wyrazie i jeżeli okaże się, że nie to zapamiętać ją. Do zapamiętania wystąpień znaków najwygodniej wykorzystać listę. Nazwę tę listę `litery` i będę ją czyścił (`litery = []`) za każdym razem kiedy rozpocznę sprawdzanie nowego wiersza w pętli `for wiersz in tekst:`.

Następnie będę brał po kolei każdy znak wiersza (`for znak in wiersz:`) i sprawdzał, czy został już on wcześniej zapisany na liście litery (`if znak not in litery:`). Jeżeli nie będzie występował, to dołączę go na końcu listy przy pomocy funkcji `append()`.

Teraz już wystarczy tylko sprawdzić czy liczba zapamiętanych znaków na liście `litery` jest większa od dotychczas zapamiętanego wiersza z największą liczbą różnych

znaków. Użyję w tym celu zmiennej pomocniczej `roznych_liter`. Jeżeli warunek okaże się poprawny zapamiętam także aktualnie sprawdzany wiersz.

Po przejściu całego pliku, wyświetlę wynik na ekranie i zamknę dostęp do pliku.

```
plik = open("przyklad.txt", "r")
tekst = plik.readlines()
roznych_liter = tekst_wiersza = 0

for wiersz in tekst:
    litery = []
    for znak in wiersz[:-1]:
        if znak not in litery:
            litery.append(znak)
    if roznych_liter < len(litery):
        roznych_liter = len(litery)
        tekst_wiersza = wiersz[:-1]
print(tekst_wiersza, roznych_liter)

plik.close()
```

Wyjaśnienia wymaga zapis `wiersz[:-1]`, pojawiający się w programie dwukrotnie. Fragment w nawiasach kwadratowych `[:-1]` nakazuje użycie wartości zmiennej `wiersz` od pierwszego do przedostatniego znaku. Równoważnym zapisem byłoby `wiersz[0:-1]`, jednak w języku Python, jeżeli bierzemy fragment listy lub tekstu począwszy od pierwszego elementu (czyli mającego indeks `0`), to zero można pominąć. Analogicznie, odczytanie sekwencji od elementu mającego indeks 5 do końca, można zapisać w skrócie `[5:]`.

Dlaczego jednak analizuję zmienną `wiersz` bez ostatniego znaku? Sprawcą tego zamieszania jest znak przejścia do nowego akapitu (\n) który występuje zawsze na końcu wiersza. Siłą rzeczy byłby on także zliczany i zapamiętywany w zmiennej przechowującej niepowtarzające się znaki. Oczywiście nie należy go brać pod uwagę, dlatego usuwam go z programu w miejscach gdzie analizuję zmienną `wiersz`.

Uważam, że takie podejście do zadania 4.2 jest ze względu na czytelność o wiele lepsze, niż zliczanie wszystkich znaków od A do Z.

Jeszcze innym sposobem rozwiązania tego zadania będzie wykorzystanie istniejącej w języku Python funkcji `set()`. Tworzy ona zbiór niepowtarzających się elementów. Zbiory to sekwencje podobne do list i w podobny sposób obsługiwane przez wiele funkcji i metod, ale jedną z różnic jest właśnie fakt, że nie mogą przechowywać kilka razy tego samego elementu.

```
plik = open("przyklad.txt", "r")
tekst = plik.readlines()
roznych_liter = tekst_wiersza = 0

for wiersz in tekst:
    wiersz = wiersz[:-1]
    if roznych_liter < len(set(wiersz)):
        roznych_liter = len(set(wiersz))
        tekst_wiersza = wiersz
print(tekst_wiersza, roznych_liter)

plik.close()
```

W powyższym przykładzie już w pierwszym poleceniu po pętli `for` pozbywam się zbędnego znaku końca wiersza.

Zamiast samemu tworzyć listę niepowtarzających się elementów, wykorzystałem wbudowaną funkcję `set()`.

Zgodnie z główną treścią zadania wynik programu należy zapisać do pliku, ale aby to zrobić proponuję samodzielną modyfikację powyższego programu posiłkując się rozwiązaniem zadania 4.1.

Wrócę na chwilę do polecenia w którym usuwam ostatni znak wiersza:

```
    wiersz = wiersz[:-1]
```

Mógłbym oczywiście pozbyć się tego wiersza i w dalszej części programu wszędzie gdzie występuje zmienna `wiersz` wprowadzać zapis `wiersz[:-1]`. Jest to jednak rozsądne w przypadku kilku wystąpień zmodyfikowanej wartości zmiennej `wiersz`. Gdyby jednak występowała ona w programie znacznie częściej, to nie było by to dobrym rozwiązaniem.

Warto w takim przypadku skorzystać z nowego operatora, który jest dostępny w języku Python od wersji 3.8. Jest to operator morsa, a swoją sympatyczną nazwę zawdzięcza podobieństwu do kłów morsa (`:=`). Pozwala on na między innymi na przypisanie wartości do zmiennej w instrukcji sprawdzającej warunek.

Powyższy program mogę zmodyfikować w ten sposób, że usunę opisywany wiersz a podstawienia dokonam bezpośrednio w warunku instrukcji `if`:

```
    if roznych_liter < len(set(wiersz := wiersz[:-1])):
```