Przygotowanie uczniów do matury z informatyki poprzez opracowanie algorytmów zadania 4.3 na maturze z 2018 r.

Roland Zimek

Zadanie 4.3.

W tym zadaniu rozważmy odległość liter w alfabecie – np. litery A i B są od siebie oddalone o 1, A i E o 4, F i D o 2, a każda litera od siebie samej jest oddalona o 0. Wypisz wszystkie słowa, w których każde dwie litery oddalone są od siebie w alfabecie co najwyżej o 10. Słowa wypisz w kolejności występowania w pliku `sygnaly.txt`, po jednym w wierszu.

Na przykład CGECF jest takim słowem, ale ABEZA nie jest (odległość A – Z wynosi 25).

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

```
AAAAAAAAAI
AAAAAAAAAE
AAAAAAAAAC
AAAAAAAAAH
AAAAAAAAAC
AAAAAAAAAI
AAAAAAAAAA
BB
AAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAB
AAAAAAAAAE
AAAAAAAAAD
AAAAAAAAAI
AAAAAAAAAE
```

W tym zadaniu także można pójść w złą stronę i sprawdzać każde możliwe pary znaków. Na przykład dla tekstu: EABD, należy sprawdzać pary EA, EB, ED, AB, AD oraz BD.

Oczywiście takie podejście też da poprawne wyniki, ale wystarczy zastanowić się jak wiele porównań będzie dla bardzo długiego wyrazu, aby zdać sobie sprawę z ilości obliczeń które będzie musiał wykonać komputer.

Lepszym podejściem do problemu i łatwiejszym do zapisania będzie odczytanie kodów ASCII wszystkich znaków i sprawdzenie różnicy pomiędzy najwyższym kodem ASCII a najniższym. Jeżeli ta różnica będzie mniejsza lub równa 10, to takie słowo będzie rozwiązaniem zadania. Litery w kodzie ASCII są uporządkowane w takiej kolejności jak w alfabecie począwszy od A do Z. Przypomnę, że kody znaków od A do Z, to wartości od 65 do 90. Kod ASCII znaku odczytami dzięki funkcji `ord(`*znak*`)`.

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

for wiersz in tekst:
    maksimum = 64
    minimum = 91
    wiersz = wiersz[:-1]
    for znak in wiersz:
        if (kod := ord(znak)) < minimum:
            minimum = kod
        if kod > maksimum:
            maksimum = kod
    if maksimum - minimum <= 10:
        print(wiersz)

plik.close()
```

W programie dla każdego wiersza tekstu przypisuję wartości początkowe dla zmiennych `maksimum` i `minimum`. Dlaczego ustawiam zmienną `maksimum` na 64 a nie 65, co jest wartością kodu ASCII dla znaku A? Otóż należy przyjąć niższą z możliwych do otrzymania w programie wartości, gdyż przypisując wartość 65, zadeklarowałbym znalezienie w tekście znaku A. Podobna sytuacja odnosi się do zmiennej `minimum`. Ponadto

użyłem zmiennych `maksimum` i `minimum` a nie `max` i `min`, gdyż są to nazwy funkcji, występujące w języku Python.

Wyjaśnienia wymaga jeszcze tylko wiersz z fragmentem kodu `wiersz = wiersz[:-1]`. Odpowiada on za odrzucenie z analizowanego wyrazu ostatniego znaku, którym jest znak przejścia do nowego akapitu (\n), występujący w każdym wierszu sprawdzanego tekstu. Podobnie w zadaniu 4.2 postąpiłem w wierszu wyświetlającym wyniki.

Program działa poprawnie, ale jest długi. Najwięcej poleceń służy wyznaczeniu minimum i maksimum. Warto by pomyśleć jak zoptymalizować program. Operujemy w programie na tekście, a tekst jest pewnego rodzaju sekwencją. Dla sekwencji działają takie funkcje jak `min()` i `max()`, dla tekstu nie dadzą one nam spodziewanego efektu. Tekst jest bowiem traktowany przez te funkcje jako całość, a nie sekwencja znaków. Wystarczy jednak przekształcić tekst w sekwencję znaków, a funkcje zadziałają tak jak tego byśmy oczekiwali. Do przekształcenia tekstu w listę użyję funkcji `list()`.

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

for wiersz in tekst:
    wiersz = wiersz[:-1]
    if ord(max(list(wiersz))) - ord(min(list(wiersz))) <= 10:
        print(wiersz)

plik.close()
```

Ale to nie koniec. W powyższym programie przekształciłem tekst na listę funkcją `list()`. Jednak nie potrzebuję listy. Wystarczy, że otrzymam pojedyncze znaki. Symbol gwiazdki „*” umieszczony przed zmienną z tekstem rozdzieli tekst na pojedyncze znaki (w przypadku zastosowania go dla listy rozdzieli listę na poszczególne elementy). Dzięki temu wiersz z warunkiem `if` mogę zapisać w następujący sposób:

```
if ord(max(*wiersz)) - ord(min(*wiersz)) <= 10:
```

Symbolu gwiazdki należy używać ostrożnie. Złe zrozumienie jego działania może spowodować błędy w programie. Nie można bowiem przypisać pojedynczych znaków (bądź elementów w przypadku listy) przypisać do zmiennej. Dlatego też poniższy program spowoduje błąd:

```
tekst = "Python"
znaki = *tekst
print(znaki)
```

Wystarczy jednak określić jakiego typu oczekujemy, aby program zadziałał poprawnie. Poprawię w tym celu środkowy wiersz:

```
znaki = [*tekst]
```

Otrzymam w ten sposób listę, zawierającą pojedyncze znaki:

```
['P', 'y', 't', 'h', 'o', 'n']
```

Innym, ciekawym sposobem podejście do rozwiązania zadania, będzie program w którym posortuję znaki w każdym analizowanym wierszu. W programie będę używał tekstu pozbawionego ostatniego znaku, będącego znakiem podziału akapitu (\n). Do rozwiązania zadania potrzebujemy jedynie dwóch znaków, tego o najniższym kodzie ASCII i tego o najwyższym kodzie ASCII. W posortowanym zbiorze elementów będzie to pierwszy i ostatni znak.

Do przekształcenia analizowanego teksu użyję opisaną już wcześniej funkcję `set()`. W przypadku podania jako argumentu funkcji `set()` tekstu, nie ustawia ona elementy w kolejności uporządkowanej, konieczne jest więc dodatkowo użycie funkcji sortującej `sorted()`.

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

for wiersz in tekst:
    tekst_wiersza = sorted(set(wiersz[:-1]))
    if ord(tekst_wiersza[-1])-ord(tekst_wiersza[0]) <= 10:
        print(wiersz[:-1])

plik.close()
```

Zapis `ord(tekst_wiersza[-1])-ord(tekst_wiersza[0])` oznacza, że od kodu ASCII ostatniego znaku (`[-1]`) analizowanego wiersza zostanie odjęty kod ASCII pierwszego znaku (pierwszy znak umieszczony jest w tekście ma indeksie `[0]`) . Jeżeli ta różnica będzie co najwyżej równa 10, to taki tekst będzie rozwiązaniem zadania.

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

Przy okazji omawiania zapisywania kodu programu w jak najmniejszej liczbie wierszy, warto wspomnieć o dwóch sposobach które można w tym celu wykorzystać.

Pierwszym z nich jest zapisanie instrukcji bezpośrednio w wierszu pętli lub warunku. Przykładowo, zamiast zapisywać kod w dwóch wierszach:

```
for i in range(5):
    print(i)
```

można taki sam efekt uzyskać poleceniem:

```
for i in range(5): print(i)
```

Co więcej, w takim zapisie można także wykonać kilka instrukcji, rozdzielając je średnikiem:

```
for i in range(5): print(i); print(i*10)
```

Drugim sposobem skrócenia zapisu do jednego wiersza jest użycie operatora trójargumentowego. W przypadku poniższej instrukcji `if…else`:

```
znak = "0"
if znak == "0":
    cyfra = 0
else:
    cyfra = 1
```

można ją zapisać w jednym wierszu:

```
cyfra = 0 if znak == "0" else 1
```

Jest to przyjemny do czytania zapis i wyraźnie skracający program, dlatego zachęcam do ich stosowania.

Jak widać zadania programistyczne na maturze nawet dla poziomu rozszerzonego nie są bardzo trudne, wymagają jednak doświadczenia programistycznego i zastanowienia się nad najlepszym rozwiązaniem. Wybranie złego pomysłu może skutkować znacznie dłuższym czasem potrzebnym na napisanie programu, który dodatkowo będzie dłuższy i mniej czytelny, co w konsekwencji może spowodować trudne do wychwycenia błędy w programie.

Dlatego uważam, że warto przeprowadzać takie analizy zadań które pojawiły się już na wcześniejszych maturach.