Przygotowanie uczniów do matury z informatyki poprzez opracowanie algorytmów zadania 4.1 na maturze z 2015 r.

Roland Zimek

Poziom rozszerzony na maturze z 2015 roku, zawiera zadanie 4, które należy rozwiązać tworząc odpowiedni program. Treść tego zadania jest następująca[1]:

Zadanie 4. LICZBY BINARNE

W pliku liczby.txt znajduje się 1000 liczb naturalnych zapisanych binarnie. Każda liczba zapisana jest w osobnym wierszu. Pierwsze pięć wierszy zawiera następujące liczby:

```
11010100111
11110111111011101
1010100111010100
1101111111111111111111010100101010101001
1010110011001101010011111010101010101111
```

Każda liczba binarna zawiera co najwyżej 250 cyfr binarnych, co oznacza, że w wielu językach programowania wartości niektórych z tych liczb nie da się zapamiętać w pojedynczej zmiennej typu całkowitoliczbowego, np. w języku C++ w zmiennej typu int.

Zadanie 4.1.

Podaj, ile liczb z pliku `liczby.txt` ma w swoim zapisie binarnym więcej zer niż jedynek.

Przykład: Dla zestawu liczb:

```
101011010011001100111
10001001
1000000
101010011100
100010
```

wynikiem jest liczba 3 (3 podkreślone liczby mają w swoim zapisie więcej zer niż jedynek).

Jak można to zadanie rozwiązać przy pomocy języka Python? Po pierwsze niezbędna jest znajomość różnych systemów liczbowych, a w szczególności systemu binarnego

[1] https://cke.gov.pl/egzamin-maturalny/egzamin-w-nowej-formule/arkusze/

(zwanego także systemem dwójkowym, lub zero–jedynkowym) i konwersji pomiędzy różnymi systemami. Nie będę tutaj szczegółowo omawiał tego zagadnienia, gdyż uczeń przystępujący do matury rozszerzonej powinien posiadać taką wiedzę.

W głównej treści zadania znajduje się informacja, że każda liczba binarna zawiera co najwyżej 250 cyfr binarnych, co ma być ułatwieniem dla osób piszących program w językach w których należy deklarować typ zmiennych i ponadto mających ograniczenia co do długości, tak jak na przykład jest to w języku C++. Na szczęście w języku Python nie ma potrzeby deklaracji typu zmiennych – Python sam dynamicznie określa jaki to jest typ zmiennej na podstawie przypisanej mu wartości. Ponadto nie posiada ograniczenia długości, więc w programie można analizować wiersze mające większą długość niż 250 cyfr binarnych.

Sposoby otwierania i zamykania plików w języku Python zostały omówione w poprzednim artykule omawiającym zadanie maturalne z 2018 roku – WEGA. Sposób ten nie zawsze jednak jest optymalny.

W szczególności zamiast zamykać dostęp do pliku poleceniem `plik.close()` na końcu pliku, warto byłoby je umieścić na początku, bezpośrednio po odczycie. Dzięki temu plik nie będzie cały czas otwarty, dzięki czemu szybciej zwolnimy trochę pamięci komputera zawierającą odwołanie do pliku. Ponadto w przypadku wystąpienia błędu krytycznego, przerywającego program, plik będzie już zamknięty i nie będzie odwołanie do niego zajmowało pamięci komputera.

Powyższe zmiany są słuszne jedynie w przypadku, gdy nie będziemy zamierzali wielokrotnie sięgać do pliku w trakcie wykonywania programu. Dlatego warto ją stosować gdy zamierzamy tylko raz odczytać zawartość pliku.

W takim przypadku program mógłby wyglądać następująco:

```
plik = open("liczby.txt", "r")
binarne = plik.readlines()
plik.close()

print(binarne)
#Pozostała część programu
```

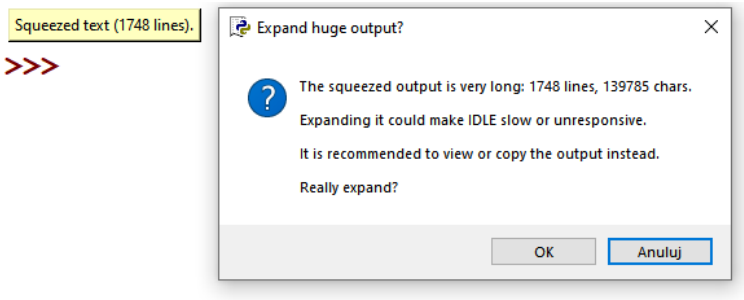
Istnieje jednak jeszcze inny, zgrabniejszy i lepszy sposób jednokrotnego odczytania zawartości pliku.

```
with open("liczby.txt") as plik:
    binarne = [i for i in plik]

print(binarne)
#Pozostała część programu
```

Dzięki zastosowaniu instrukcji `with`, zostanie utworzone odwołanie do pliku i przyporządkowane go do podanej nazwy zmiennej. Następnie należy przepisać zawartość pliku do zmiennej. Po wykonaniu tej czynności dostęp do pliku zostanie zamknięty i zwolniona zostanie pamięć komputera w której zapamiętano to odwołanie. Wadą takiego rozwiązania jest to, że nie będzie można w programie ponownie odwołać się do zawartości pliku.

Powyższy program otworzy do czytania plik `liczby.txt`, a następnie przepisze jego zawartość do zmiennej `binarne`, rozdzielając każdy wiersz jako osobny argument. Polecenie `print()` wyświetla na ekranie zawartość zmiennej. Po uruchomieniu tego programu na ekranie zostanie wyświetlony komunikat: `Squeezed text (1748)` lub podobny. Spowodowane jest to tym, że wczytany tekst jest bardzo długi (zawiera 1000 liczb binarnych) i Python domyślnie nie wyświetla tak długich zmiennych. Jeżeli jednak chcesz zobaczyć wynik działania polecenia `print()`, kliknij dwukrotnie żółty przycisk, a następnie potwierdź swoją decyzję klikając przycisk OK.



Z treści zadania 4.1. wynika, że nie musimy przeliczać liczb binarnych do systemu dziesiętnego, a jedynie traktować je jako zwykły tekst. Tak też są one zapisane zarówno w pliku, jak i w zmiennej `binarne`.

Wystarczy więc po kolei brać wiersze ze zmiennej `binarne` i sprawdzać kolejne znaki. Jeżeli znakiem tym będzie `0` to zwiększamy zmienną `cyfra0` o 1. Analogicznie postępujemy dla znaku `1`.

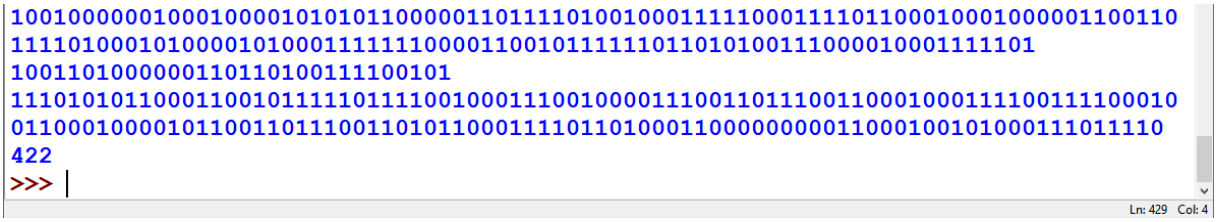
Jeżeli w przypadku analizowanego wiersza wartość zmiennej `cyfra0` będzie większa od wartości zmiennej `cyfra1`, to wypisujemy ten wiersz na ekranie i zwiększamy wartość zmiennej licznik o 1.

```
with open("liczby.txt") as plik:
    binarne = [i for i in plik]

licznik = 0
for wiersz in binarne:
    cyfra0 = cyfra1 = 0
    for znak in wiersz.strip():
        if znak == "0":
            cyfra0 += 1
        else:
            cyfra1 += 1
    if cyfra0 > cyfra1:
        print(wiersz.strip())
        licznik += 1
print(licznik)
```

W powyższym przykładzie dla każdego analizowanego wiersz przypisuję na nowo do zmiennych `cyfra0` i `cyfra1` wartość 0. Zapisałem to w jednym wierszu: `cyfra0 = cyfra1 = 0`. Ponadto w dwóch miejscach zastosowałem metodę `.strip()`, która usuwa tak zwane białe znaki na początku i końcu tekstu. Białym znakiem jest znak końca akapitu (\n), występujący na końcu prawie wszystkich wierszy. Dlaczego na końcu prawie wszystkich wierszy? Gdyż ostatni wiersz w pliku nie posiada na końcu znaku (\n). Dlatego właśnie użyłem zapisu z metodą `.strip()`, a nie z dodatkiem `[:-1]`, gdyż usunąłbym znaczący znak.

Po uruchomieniu programu otrzymamy na ekranie długą listę wierszy spełniających warunki zadania i wartość 422, jako liczbę tych wierszy.



Podany powyżej program jest całkiem przejrzysty i w miarę krótki. Daje też poprawne wyniki. Jeżeli jednak skorzystamy w programie z metody `.count()`, możemy program znacząco uprościć.

Metoda ta zlicza wystąpienia podanego podciągu w analizowanej zmiennej.

```
with open("liczby.txt") as plik:
    binarne = [i for i in plik]


licznik = 0
for wiersz in binarne:
    if wiersz.count("0") > wiersz.count("1"):
        print(wiersz.strip())
        licznik += 1
print(licznik)
```

Jak widać taki zapis jest krótszy i bardziej przejrzysty.

Mogę jednak wykorzystać listy składane aby jeszcze bardziej skrócić program i zapisać go w jednym wierszu. Będę jednak musiał zrezygnować z wyświetlania wierszy spełniających warunki zadania. Na szczęście nie jest to wymóg zadania 4.1.

Sposób ten polega na utworzeniu listy składanej, w której pozostawię tylko wiersze spełniające kryterium. Wystarczy potem sprawdzić długość otrzymanej listy aby rozwiązać zadanie.

Ostatecznie program będzie wyglądał następująco:

```
with open("liczby.txt") as plik:
    binarne = [i for i in plik]


print(len([i for i in binarne if i.count("0") > i.count("1")]))
```

Dzięki temu, że Python nie ma ograniczeń co do długości przechowywania zmiennych, nie wymaga deklaracji zmiennych i posiada kapitalne rozwiązanie w postaci list składanych programy mogą być bardzo krótkie i przejrzyste. Rozwiązania w innych językach programowania są o wiele bardziej długie i wymagają zmierzenia się z dodatkowymi problemami których użytkownik Pythona nie doświadcza.