Algorytmy i Struktury Danych
Zadanie offline 1 (13.III.2025)

Format rozwiązań

Rozwiązanie zadania musi się składać z krótkiego opisu algorytmu (wraz z uzasadnieniem poprawności) oraz jego implementacji. Zarówno opis algorytmu, jak i implementacja powinny się znajdować w tym samym pliku Pythona (rozszerzenie `.py`). Opis powinien być na początku pliku w formie komentarza (w pierwszej linii w komentarzu powinno być imię i nazwisko studenta). Opis nie musi być długi—wystarczy kilka zdań, jasno opisujących ideę algorytmu. Implementacja musi być zgodna z szablonem kodu źródłowego dostarczonym wraz z zadaniem. Niedopuszczalne jest w szczególności:

1. korzystanie z wbudowanych funkcji sortujących,
2. korzystanie z zaawansowanych struktur danych (np. słowników czy zbiorów),
3. zmienianie nazwy funkcji implementującej algorytm, listy jej argumentów, lub nazwy pliku z rozwiązaniem,
4. wypisywanie na ekranie jakichkolwiek napisów innych niż wypisywane przez dostarczony kod (ew. napisy dodane na potrzeby diagnozowania błędów należy usunąć przed wysłaniem zadania).

Dopuszczalne jest natomiast:

1. korzystanie z następujących elementarnych struktur danych: krotka, lista,
2. korzystanie ze struktur danych dostarczonych razem z zadaniem (jeśli takie są).

Wszystkie inne algorytmy lub struktury danych wymagają implementacji przez studenta. Dopuszczalne jest oczywiście implementowanie dodatkowych funkcji pomocniczych w pliku z szablonem rozwiązania.

Zadania niezgodne z powyższymi ograniczeniami otrzymają ocenę 0 punktów. Rozwiązania w innych formatach (np. `.PDF`, `.DOC`, `.PNG`, `.JPG`) z definicji nie będą sprawdzane i otrzymają ocenę 0 punktów, nawet jeśli będą poprawne.

Testowanie rozwiązań

Żeby przetestować rozwiązanie zadania, należy wykonać polecenie: `python zad1.py`

Kilka uwag o Pythonie

```
x = "abcdefg"
y = x[::-1]     # y staje sie nowym napisem "gfedcba"
z = y           # z i y wskazują na ten sam napis (koszt tego przypisania to O(1)
                # niezaleznie od dlugosci napisu y)
ord("a")        # wynikiem jest kod litery 'a' (97)
chr(97)         # wynikiem jest litera o kodzie 97 ("a")
```

Szablon rozwiązania:	`zad1.py`
Pierwszy próg złożoności:	$O(N + n \log n)$, gdzie N to łączna długość napisów w tablicy wejściowej a n to liczba wyrazów.
Drugi próg złożoności:	$O(N \log N)$ lub $O(nk)$ gdzie N to łączna długość napisów w tablicy wejściowej, n to liczba wyrazów a k to długość najdłuższego słowa

Mówimy, że dwa napisy są sobie równoważne, jeśli albo są identyczne, albo byłyby identyczne, gdyby jeden z nich zapisać od tyłu. Na przykład napisy "kot" oraz "tok" są sobie równoważne, podobnie jak napisy "pies" i "pies". Dana jest tablica T zawierająca n napisów o łącznej długości N (każdy napis zawiera co najmniej jeden znak, więc $N \geq n$; każdy napis składa się wyłącznie z małych liter alfabetu łacińskiego). Siłą napisu $T[i]$ jest liczba indeksów j takich, że napisy $T[i]$ oraz $T[j]$ są sobie równoważne. Napis $T[i]$ jest najsilniejszy, jeśli żaden inny napis nie ma większej siły.

Proszę zaimplementować funkcję `g(T)`, która zwraca siłę najsilniejszego napisu z tablicy T. Na przykład dla wejścia:

```
#     0       1       2      3        4      5       6
T = ["pies", "mysz", "kot", "kogut", "tok", "seip", "kot"]
```

wywołanie `g(T)` powinno zwrócić 3. Algorytm powinien być możliwie jak najszybszy. Proszę podać złożoność czasową i pamięciową zaproponowanego algorytmu.