Algorytmy i Struktury Danych
Zadanie offline 6 (6.V.2024)

## 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 zaawansowanych struktur danych (np. słowników czy zbiorów),
2. zmienianie nazwy funkcji implementującej algorytm, listy jej argumentów, lub nazwy pliku z rozwiązaniem,
3. 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, kolejka `collections.deque`, kolejka priorytetowa (`queue.PriorityQueue`),
2. korzystanie z wbudowanych funkcji sortujących (można założyć, że mają złożoność $O(n \log n)$).

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. `.ZIP`, `.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: `python3 zad6.py`

## Zadanie offline 6.

**Szablon rozwiązania:** `zad6.py`

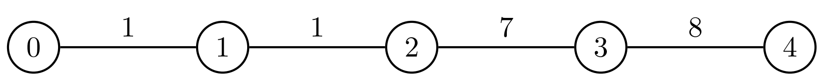
Dany jest ważony, nieskierowany graf $G$ oraz *dwumilowe buty* - specjalny sposób poruszania się po grafie. Dwumilowe buty umożliwiają pokonywanie ścieżki złożonej z dwóch krawędzi grafu tak, jakby była ona pojedynczą krawędzią o wadze równej maksimum wag obu krawędzi ze ścieżki. Istnieje jednak ograniczenie - pomiędzy każdymi dwoma użyciami *dwumilowych butów* należy przejść w grafie co najmniej jedną krawędź w sposób zwyczajny. Macierz $G$ zawiera wagi krawędzi w grafie, będące liczbami naturalnymi, wartość 0 oznacza brak krawędzi.
Proszę opisać, zaimplementować i oszacować złożoność algorytmu znajdowania najkrótszej ścieżki w grafie z wykorzystaniem mechanizmu *dwumilowych butów.*
Rozwiązanie należy zaimplementować w postaci funkcji:

```
def jumper(G, s, w):
    ...
```

która zwraca długość najkrótszej ścieżki w grafie $G$ pomiędzy wierzchołkami $s$ i $w$, zgodnie z zasadami używania *dwumilowych butów.*

Zaimplementowana funkcja powinna być możliwie jak najszybsza. Proszę przedstawić złożoność czasową oraz pamięciową użytego algorytmu.

**Przykład:** Rozważmy następujący graf:




Najkrótszą ścieżką między wierzchołkami 0 i 4 wykorzystującą *dwumilowe buty* będzie ścieżka $[0, 1, 2, 4]$ o długości 10 (z krawędzią $(2, 4)$ będącą dwumilowym skokiem). Ścieżka złożona z dwóch dwumilowych skoków, $[0, 2, 4]$, byłaby krótsza, ale nie spełnia warunków zadania.