

## Algorytmy i Struktury Danych

### Zadanie offline 6 (2.V.2022)

#### 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. modyfikowanie testów dostarczonych wraz z szablonem,
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, kolejka `collections.deque`, kolejka priorytetowa (`queue.PriorityQueue` lub `heapq`),
2. korzystanie ze struktur danych dostarczonych razem z zadaniem (jeśli takie są).
3. 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. `.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 graf nieskierowany  $G = (V, E)$  oraz dwa wierzchołki  $s, t \in V$ . Proszę zaproponować i zaimplementować algorytm, który sprawdza, czy istnieje taka krawędź  $\{p, q\} \in E$ , której usunięcie z  $E$  spowoduje wydłużenie najkrótszej ścieżki między  $s$  a  $t$  (usuwamy tylko jedną krawędź). Algorytm powinien być jak najszybszy i używać jak najmniej pamięci. Proszę skrótkowo uzasadnić jego poprawność i oszacować złożoność obliczeniową.

Algorytm należy zaimplementować jako funkcję:

```
def longer(G, s, t):  
    ...
```

która przyjmuje graf  $G$  oraz numery wierzchołków  $s, t$  i zwraca dowolną krawędź spełniającą warunki zadania, lub `None` jeśli takiej krawędzi w  $G$  nie ma. Graf przekazywany jest jako lista list sąsiadów, t.j.  $G[i]$  to lista sąsiadów wierzchołka o numerze  $i$ . Wierzchołki numerowane są od 0. Funkcja powinna zwrócić krotkę zawierającą numery dwóch wierzchołków pomiędzy którymi jest krawędź spełniająca warunki zadania, lub `None` jeśli takiej krawędzi nie ma. Jeśli w grafie oryginalnie nie było ścieżki z  $s$  do  $t$  to funkcja powinna zwrócić `None`.

**Przykład.** Dla argumentów:

```
G = [ [1, 2],  
       [0, 2],  
       [0, 1] ]  
s = 0  
t = 2
```

wynikiem jest np. krotka: (0, 2)