

Algorytmy i Struktury Danych

Kolokwium 2 (25.V.2023)

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`, `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 kol2.py`

Szablon rozwiązania:	kol2.py
Złożoność podstawowa (1.0pkt):	$O(E^2)$
Złożoność akceptowalna (+1.0pkt):	$O(VE \log^* E)$
Złożoność wzorcowa (+2.0pkt):	$O(VE)$
V to liczba wierzchołków a E to liczba krawędzi grafu; \log^* to logarytm iterowany.	

Dany jest ważony, nieskierowany graf $G = (V, E)$, którego wagi krawędzi opisuje funkcja $w: E \rightarrow \mathbb{N}$. Wiadomo, że wagi krawędzi są parami różne. Niech T będzie pewnym drzewem rozpinającym G , m będzie najmniejszą wagą krawędzi z T a M będzie największą wagą krawędzi z T . Mówimy, że T jest *piękne* jeśli każda krawędź spoza T albo ma wagę mniejszą niż m albo większą niż M . Wagą drzewa rozpinającego jest suma wag jego krawędzi. Zadanie polega na implementacji funkcji:

```
beautree( G )
```

która na wejściu otrzymuje graf reprezentowany w postaci listowej i zwraca wagę najlżejszego pięknego drzewa rozpinającego G lub `None` jeśli takie drzewo nie istnieje. Użyty algorytm powinien być możliwie jak najszybszy. Proszę uzasadnić poprawność zaproponowanego algorytmu oraz oszacować jego złożoność czasową i pamięciową.

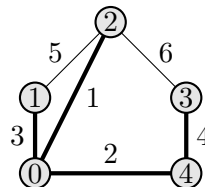
Reprezentacja grafu. Niech G będzie argumentem funkcji `beautree`. Graf G ma wierzchołki o numerach od 0 do $n - 1$, gdzie:

```
n = len(G)
```

Dla danego wierzchołka i , $G[i]$ to lista par postaci (j, w) , gdzie j to numer wierzchołka do którego prowadzi krawędź z i a w to jej waga.

Przykład. Dla wejścia:

```
G = [ [(1,3), (2,1), (4,2)], # 0
      [(0,3), (2,5)],        # 1
      [(1,5), (0,1), (3,6)], # 2
      [(2,6), (4,4)],        # 3
      [(3,4), (0,2)] ]       # 4
```



wynikiem jest 10. Mamy piękne drzewo rozpinające składające się z krawędzi 0 – 1, 0 – 2, 0 – 4 i 4 – 3 o wadze $3 + 1 + 2 + 4 = 10$.