Algorytmy i Struktury Danych
Zadanie offline 1 (15.IV.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 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. modyfikowanie testów dostarczonych wraz z szablonem,
5. 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,
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: `python3 zad4.py`

Zadanie offline 4.

Szablon rozwiązania: `zad4.py`

Dany jest nieskierowany graf $G = (V, E)$, którego wierzchołki reprezentują punkty nawigacyjne nad Bajtocją, a krawędzie reprezentują korytarze powietrzne między tymi punktami. Każdy korytarz powietrzny $e_i \in E$ powiązany jest z optymalnym pułapem przelotu $p_i \in \mathbb{N}$ (wyrażonym w metrach). Przepisy dopuszczają przelot danym korytarzem, jeśli pułap samolotu różni się od optymalnego najwyżej o t metrów. Graf jest dany w postaci listy krawędzi $L = [(u_1, v_1, p_1), (u_2, v_2, p_2), ...(u_n, v_n, p_n)]$, gdzie: $u_i, v_i \in \mathbb{N}$ to numery punktów nawigacyjnych, a p_i to optymalny pułap przelotu. Ponieważ graf jest nieskierowany na liście L umieszczono wyłącznie krotki, w których $u_i < v_i$

Proszę zaimplementować funkcję `Flight(L,x,y,t)`, która sprawdza czy istnieje możliwość przelotu z zadanego punktu $x \in V$ do zadanego punktu $y \in V$ w taki sposób, żeby samolot nigdy nie zmieniał pułapu. Algorytm powinien być poprawny i możliwie jak najszybszy. Proszę oszacować jego złożoność czasową.

Przykład

Dla danych wejściowych:

```
L = [(0,1,2000),(0,2,2100),(1,3,2050),(2,3,2300),
     (2,5,2300),(3,4,2400),(3,5,1990),(4,6,2500),(5,6,2100)]

x = 0
y = 6
t = 60
```

Poprawną odpowiedzią jest: $True$, lot na pułapie 2045, wiedzie przez punkty 0 1 3 5 6.
W przypadku gdyby $t = 50$, odpowiedzią powinno być $False$.