

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` 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: `python kol3b.py`

Szablon rozwiązania:

kol3b.py

Złożoność akceptowalna (1.5pkt):

$O(n^2)$, gdzie n to liczba lotnisk.

Złożoność wzorcowa (+2.5pkt):

$O(m \log n)$, gdzie m to liczba dróg (tj. krawędzi w G) a n to liczba lotnisk.

Mapa lotnisk w Bitlandii ma postać grafu nieskierowanego $G = (V, E)$, gdzie wierzchołki to lotniska a E to drogi między lotniskami. Każda droga ma pewien koszt wyrażony w złotówkach (trzeba zapłacić za paliwo). Dodatkowo między dowolnymi dwoma lotniskami można przelecieć szybowcem. Przy takim przelocie nie trzeba płacić za paliwo, ale każde lotnisko ma swoje opłaty (takie same za start i lądowanie).

W ramach zadania należy zaimplementować funkcję:

```
def airports( G, A, s, t )
```

która oblicza najniższy możliwy koszt dotarcia z lotniska s do lotniska t przy następujących założeniach:

1. G zawiera graf reprezentowany listowo (czyli dla każdego lotniska u , $G[u]$ to lista par postaci (v, c) , oznaczających że mamy krawędź z u do v o koszcie przejazdu c ; jeśli istnieje krawędź z u do v o koszcie c , to graf zawiera też krawędź z v do u o tym samym koszcie).
2. A to lista opłat lotniskowych (czyli $A[u]$ to opłata na lotnisku u , ponoszona tylko wtedy gdy z tego lotniska startujemy szybowcem lub na nim lądujemy szybowcem).
3. Wierzchołek s to lotnisko startowe a wierzchołek t to lotnisko docelowe.

Rozważmy następujące dane:

```
G = [ [(1,3), (3,2)],      # 0
      [(0,3), (2,20)],     # 1
      [(1,20), (5,1), (3,6)], # 2
      [(0,2), (2,6), (4,1)], # 3
      [(3,1), (5,7)],      # 4
      [(4,7), (2,1)] ]       # 5
```

```
#      0   1   2   3   4   5
A = [50, 100, 1, 20, 2, 70 ]
```

Wywołanie `airports(G, A, 0, 5)` powinno zwrócić wynik 7: z 0 jedziemy do 3, stamtąd do 4, wsiadamy do szybowca i lecimy do 2, po czym jedziemy do 5. Koszt tej trasy to $2 + 1 + (2 + 1) + 1$ (w nawiasie zawarty jest koszt przelotu).

Podpowiedź. Ile lotów maksymalnie i minimalnie warto rozważyć?