Algorytmy i Struktury Danych
Kolokwium 2: Zadanie A (12.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. 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 kol2a.py`

Szablon rozwiązania:	`kol2a.py`
Złożoność akceptowalna (3pkt):	$O(n^2)$, gdzie n to łączna liczba punktów.
Złożoność wzorcowa (+1pkt):	$O(n \log n)$, gdzie n to łączna liczba punktów.

Ciężarówka o numerze bocznym 1212 musi przejechać z punktu A do punktu B. Na trasie znajduje się pewna liczba punktów kontrolnych oraz pewna liczba punktów przesiadkowych. Ciężarówką jedzie dwóch kierowców-zmienników, Jacek i Marian. Z punktu A rusza Jacek, ale kierowcy muszą się zmieniać najpóźniej na trzecim punkcie przesiadkowym od ostatniej zmiany. Zadanie polega na takim zaplanowaniu przesiadek, żeby Marian był za kierownicą podczas mijania jak najmniejszej liczby punktów kontrolnych.

Można sobie wyobrazić, że trasa przebiega po jednowymiarowej linii, gdzie punkt A jest na pozycji 0 a punkty przesiadkowe i kontrolne mają współrzędne naturalne (większe od zera, parami różne). Proszę zaimplementować funkcję:

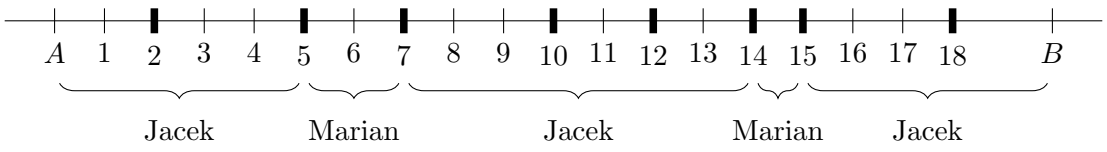
```
def drivers( P, B )
```

która na wejściu otrzymuje tablicę `P` zawierającą współrzędne punktów przesiadkowych i kontrolnych, oraz współrzędną punktu B. Każdy element tablicy `P` to para `(x,t)`, gdzie `x` to współrzędna punktu a `t` ma wartość `True` jeśli jest to punkt przesiadkowy oraz `False` jeśli jest to punkt kontrolny. Tablica `P` nie musi być posortowana. Funkcja powinna zwrócić indeksy punktów przesiadkowych, na których kierowcy zamieniają się miejscami. Funkcja powinna być możliwie jak najszybsza.

Rozważmy następujące dane:

```
p = True
c = False
#     0     1     2     3     4     5     6       7      8      9
P = [(1,c),(3,c),(4,c),(6,c),(8,c),(9,c),(11,c),(13,c),(16,c),(17,c),
     (2,p),(5,p),(7,p),(10,p),(12,p),(14,p),(15,p),(18,p)]
#     10    11    12    13     14     15     16     17
B = 20
```

wywołanie `drivers(P, B)` może zwrócić tablicę `[11,12,15,16]`, odpowiadające poniższej sekwencji zmian (grube kreski odpowiadają punktom przesiadkowym):



Podpowiedź. Proszę się zastanowić, ile punktów kontrolnych musi przejechać Marian w momencie, gdy dojeżdżamy do danego punktu przesiadkowego i dana osoba prowadzi.