Algorytmy i Struktury Danych
Zadanie offline 5 (25.IV.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`)**,
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 zad5.py`

Zadanie offline 5.

Szablon rozwiązania: `zad5.py`

W roku 2050 spokojny Maksymilian odbywa podróż przez pustynię z miasta A do miasta B. Droga pomiędzy miastami jest linią prostą na której w pewnych miejscach znajdują się plamy ropy. Maksymilian porusza się 24 kołową cysterną, która spala 1 litr ropy na 1 kilometr trasy. Cysterna wyposażona jest w pompę pozwalającą zbierać ropę z plam. Aby dojechać z miasta A do miasta B Maksymilian będzie musiał zebrać ropę z niektórych plam (by nie zabrakło paliwa), co każdorazowo wymaga zatrzymania cysterny. Niestety, droga jest niebezpieczna. Maksymilian musi więc tak zaplanować trasę, by zatrzymać się jak najmniej razy. Na szczęście cysterna Maksymiliana jest ogromna - po zatrzymaniu zawsze może zebrać całą ropę z plamy (w cysternie zmieściłaby się cała ropa na trasie).

Zaproponuj i zaimplementuj algorytm wskazujący, w których miejscach trasy Maksymilian powinien się zatrzymać i zebrać ropę. Algorytm powinien być możliwe jak najszybszy i zużywać jak najmniej pamięci. Uzasadnij jego poprawność i oszacuj złożoność obliczeniową.

Dane wejściowe reprezentowane są jako tablica liczb naturalnych `T`, w której wartość `T[i]` to objętość ropy na polu numer `i` (objętość 0 oznacza brak ropy). Pola mają numery od 0 do $n-1$ a odległość między kolejnymi polami to 1 kilometr. Miasto A znajduje się na polu 0 a miasto B na polu $n-1$. Zakładamy, że początkowo cysterna jest pusta, ale pole 0 jest częścią plamy ropy, którą można zebrać przed wyruszeniem w drogę. Zakładamy również, że zadanie posiada rozwiązanie, t.j. da się dojechać z miasta A do miasta B.

Algorytm należy zaimplementować w funkcji:

```
def plan(T):
   ...
```

która przyjmuje tablicę z opisem pół `T[0]`, ..., `T[n-1]` i zwraca listę pól, na których należy się zatrzymać w celu zebrania ropy. Lista powinna być posortowana w kolejności postojów. Postój na polu 0 również jest częścią rozwiązania.

Przykład. Dla wejścia:

```
#    0 1 2 3 4 5 6 7
T = [3,0,2,1,0,2,5,0]
```

wynikiem jest np. lista `[0,2,5]`.