Algorytmy i Struktury Danych
Zadanie offline 7 (03.V.2025)

Format rozwiązań

Wysłać należy tylko jeden plik: `zad7.py`

Plik można wysyłać wielokrotnie, liczy się ostatnia wersja zapisana w systemie.

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. zmienianie nazwy funkcji implementującej algorytm, listy jej argumentów, lub nazwy pliku z rozwiązaniem,
2. 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 z wbudowanych funkcji sortujących (można założyć, że mają złożoność $O(n \log n)$).
3. korzystanie z zaawansowanych struktur danych (np. słowników i zbiorów).

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. `.ZIP`, `.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 zad7.py`

Szablon rozwiązania:	`zad7.py`
Złożoność akceptowalna (2.0pkt):	$O(n^3)$
Złożoność wzorcowa (+2.0pkt):	$O(n^2)$
Gdzie n to liczba drzew w sadzie.	

Bajtek jak co roku przygotowuje się do konkursu sadowniczego "*modulo m*". Właśnie skończył liczyć jabłka w swoim sadzie, skrupulatnie notując ile owoców jest na każdym z n drzew. Żeby sad mógł być zgłoszony do konkursu, musi spełniać pewien warunek: suma owoców na drzewach musi być podzielna przez liczbę m, corocznie wybieraną przez bajtocką komisję sadowniczą. Bajtek zastanawia się ile drzew musi wyciąć aby sad spełniał warunek konkursu. Zależy mu na tym by wyciąć możliwe najmniej drzew, gdyż wycinka każdego drzewa to dla niego zawsze trudna decyzja.

Proszę zaimplementować funkcję `orchard(T, m)` która wyznaczy, ile minimalnie drzew musi wyciąć Bajtek, aby sad spełniał warunek konkursu (proszę zwrócić uwagę, że w najgorszym razie Bajtek może wyciąć wszystkie drzewa; spełni to formalne wymaganie konkursu). Tablica T zawiera n-elementów, gdzie wartość $T[i]$ to liczba jabłek na i-tym drzewie, a m to liczba, przez którą suma owoców na drzewach musi być podzielna. Liczba m jest liczbą całkowitą z przedziału $[1, 7n]$. Algorytm powinien być możliwie jak najszybszy. Proszę uzasadnić poprawność zaproponowanego algorytmu oraz oszacować jego złożoność czasową i pamięciową.

Przykład. Dla wejścia:

```
#    0  1  2  3  4   5  6
T = [2, 2, 7, 5, 1, 14, 7]
m = 7
```

wywołanie `orchard(T, m)` powinno zwrócić wartość 2, odpowiadającą wycięciu drzew o indeksach 0 i 4 (z dwoma i jednym jabłkiem), co zostawia sad z sumą $2 + 7 + 5 + 14 + 7 = 35$ jabłek.

Podpowiedź 1. Czy możliwe jest, że we fragmencie sadu składającym się z pierwszych i drzew zostanie j jabłek, jeśli wytniemy k drzew?

Podpowiedź 2. Powyższa podpowiedź daje podstawy do rozwiązania zadania, ale można ją uzupełnić o szereg optymalizacji.