Algorytmy i Struktury Danych
Kolokwium 1 (30.III.2023)

Format rozwiązań

Wysłać należy tylko jeden plik: `kol1.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. 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. 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`),

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 kol1.py`

Szablon rozwiązania:	`kol1.py`
Złożoność akceptowalna (2.0pkt):	$O(np)$, gdzie n to liczba elementów w tablicy a p to liczba elementów w przedziałach, w których poszukiwane są elementy sumowane.
Złożoność wzorcowa (+2.0pkt):	$O(n \log n)$, gdzie n to liczba elementów w tablicy.

Dana jest n-elementowa tablica liczb naturalnych `T` oraz dodatnie liczby naturalne k i p, gdzie $k \leq p \leq n$. Niech z_i będzie k-tą największą spośród elementów: `T[i]`, `T[i+1]`, ..., `T[i+p-1]`. Innymi słowy, z_i to k-ty największy element w `T` w przedziale indeksów od i do $i + p - 1$ włącznie.

Doprecyzowanie: Rozważmy tablicę `[17,25,25,30]`. W tej tablicy 1-wszy największy element to 30, 2-gi największy element to 25, 3-ci największy element to także 25 (drugie wystąpienie), a 4-ty największy element to 17.

Proszę zaimplementować funkcję `ksum(T, k, p)`, która dla tablicy `T` (o rozmiarze n elementów) i dodatnich liczb naturalnych k i p ($k \leq p \leq n$) wylicza i zwraca wartość sumy:

$$z_0 + z_1 + z_2 + \ldots + z_{n-p}$$

Przykład. Dla wejścia:

```
T = [7,9,1,5,8,6,2,12]
k = 4
p = 5
```

wywołanie `ksum(T, k, p)` powinno zwrócić wartość 17 (odpowiadającą sumie $5 + 5 + 2 + 5$).
Algorytm powinien być możliwie jak najszybszy. Proszę uzasadnić poprawność zaproponowanego algorytmu oraz oszacować jego złożoność czasową i pamięciową.

Algorytmy i Struktury Danych
Kolokwium 2 (25.V.2023)

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, 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: `python3 kol2.py`

Szablon rozwiązania:	`kol2.py`
Złożoność podstawowa (1.0pkt):	$O(E^2)$
Złożoność akceptowalna (+1.0pkt):	$O(VE \log^* E)$
Złożoność wzorcowa (+2.0pkt):	$O(VE)$
V to liczba wierzchołków a E to liczba krawędzi grafu; $\log^*$ to logarytm iterowany.	

Dany jest ważony, nieskierowany graf $G = (V, E)$, którego wagi krawędzi opisuje funkcja $w: E \to \mathbb{N}$. Wiadomo, że wagi krawędzi są parami różne. Niech T będzie pewnym drzewem rozpinającym G, m będzie najmniejszą wagą krawędzi z T a M będzie największą wagą krawędzi z T. Mówimy, że T jest *piękne* jeśli każda krawędź spoza T albo ma wagę mniejszą niż m albo większą niż M. Wagą drzewa rozpinającego jest suma wag jego krawędzi. Zadanie polega na implementacji funkcji:

```
beautree( G )
```

która na wejściu otrzymuje graf reprezentowany w postaci listowej i zwraca wagę najlżejszego pięknego drzewa rozpinającego G lub `None` jeśli takie drzewo nie istnieje. Użyty algorytm powinien być możliwie jak najszybszy. Proszę uzasadnić poprawność zaproponowanego algorytmu oraz oszacować jego złożoność czasową i pamięciową.

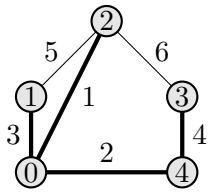
Reprezentacja grafu. Niech `G` będzie argumentem funkcji `beautree`. Graf `G` ma wierzchołki o numerach od 0 do $n-1$, gdzie:

```
n = len(G)
```

Dla danego wierzchołka `i`, `G[i]` to lista par postaci `(j,w)`, gdzi `j` to numer wierzchołka do którego prowadzi krawędź z `i` a `w` to jej waga.

Przykład. Dla wejścia:

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



wynikiem jest 10. Mamy piękne drzewo rozpinające składające się z krawędzi $0-1$, $0-2$, $0-4$ i $4-3$ o wadze $3+1+2+4=10$.

Algorytmy i Struktury Danych
Kolokwium uzupełniajace (4.VII.2023)

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 zaawansowanych struktur danych (np. wbudowanej kolejki priorytetowej, słowników czy zbiorów),
2. korzystanie z wbudowanych algorytmów sortujących,
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, lista, kolejka `collections.deque`.

Wszystkie inne algorytmy (w tym sortowania) lub struktury danych (w tym kolejka priorytetowa) 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 kolu.py`

Szablon rozwiązania:	`kolu.py`
Złożoność akceptowalna (2.0pkt):	$O(n \log n)$
Złożoność wzorcowa (+2.0pkt):	$O(n)$
Gdzie n to liczba kubełków z lodami.	

Danych jest n kubełków z lodami. W każdym kubełku jest pewna objętość lodów (reprezentowana przez dużą liczbę naturalną). Zjedzenie jednego kubełka zajmuje jedną minutę (niezależnie od tego ile zawiera lodów). Z upływem każdej minuty w każdym niepustym kubełku topi się 1 jednostka objętości lodów (i nie można jej później zjeść). Zaproponuj i zaimplementuj algorytm wyliczający maksymalną objętość lodów, którą w sumie można zjeść.

Algorytm należy zaimplementować jako funkcję:

```
ice_cream( T )
```

przyjmującą tablicę liczb naturalnych `T` i zwracającą maksymalną objętość lodów, którą w sumie można zjeść. Element `T[i]` to początkowa objętość lodów w i-tym kubełku. Proszę uzasadnić poprawność zaproponowanego algorytmu i oszacować jego złożoność obliczeniową.

Przykład. Dla wejścia:

```
T = [5, 1, 3, 7, 8]
```

wynikiem jest 17. Lody można jeść zaczynając w pierwszej minucie od czwartego kubełka z 7 jednostkami lodów. W drugiej minucie zjadamy pozostałe 7 jednostki z kubełka nr 5 (początkowo było tam 8 jednostek lodów ale 1 jednostka roztopiła się po pierwszej minucie). W trzeciej minucie zjadamy 3 pozostałe jednostki w pierwszym kubełku. W tym momencie nie możemy zjeść nic więcej ponieważ lody z kubełków 2 i 3 całkowicie się już roztopiły. Tak więc otrzymujemy 7 + 7 + 3 = 17