Algorytmy i Struktury Danych
Kolokwium 1 (4.IV.2024)

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 (1.0pkt):	$O(n^2)$, gdzie n to liczba elementów w tablicy.
Złożoność wzorcowa (+3.0pkt):	$O(n \log n)$, gdzie n to liczba elementów w tablicy.

Dana jest n-elementowa tablica liczb naturalnych `T`. Dla każdego indeksu $i < n$, rangą elementu na pozycji i określamy liczbę elementów, które w tablicy występują przed elementem i-tym, a ich wartość jest mniejsza od $T[i]$.

Doprecyzowanie: Rozważmy tablicę `[5,3,9,4]`. W tej tablicy dwa pierwsze elementy mają rangę 0 (nie poprzedza ich żaden mniejszy element), 3-ci element ma rangę 2 (przed nim w tablicy znajdują się wartości 5 oraz 3), a ostatni ma rangę 1 (przed nim w tablicy jedynie 3 jest mniejsze).

Proszę zaimplementować funkcję `maxrank(T)`, która dla tablicy `T` o rozmiarze n elementów zwróci maksymalną rangę pośród wszystkich elementów tablicy.

Przykład. Dla wejścia:

```
T = [5,3,9,4]
```

wywołanie `maxrank(T)` powinno zwrócić wartość 2 (odpowiadającą randze elementu na trzeciej pozycji).

Algorytm powinien być możliwie jak najszybszy. Proszę uzasadnić poprawność zaproponowanego algorytmu oraz oszacować jego złożoność czasową i pamięciową.

Algorytmy i Struktury Danych
Kolokwium 2 (23 maja 2024r.)

Format rozwiązań

Wysłać należy tylko jeden plik: `kol2.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 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`),
2. 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. `.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`

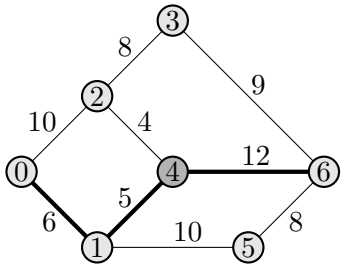
Szablon rozwiązania:	`kol1.py`
Złożoność akceptowalna (1.0pkt):	$O(V^3)$
Złożóność lepsza (3.0pkt):	$O(E \log V)$
Złożoność wzorcowa (4.0pkt):	$O(E+V)$
Gdzie V to liczba wierzchołków w grafie a E to liczba krawędzi.	

Magiczny wojownik chce się przedostać przez góry Bajtocji. Wyrusza z miasteczka s i chce się dostać do miasteczka t. Wojownik dysponuje mapą z zaznaczonymi schroniskami, miasteczkami s i t, oraz łączącymi je szlakami (mapa ma formę grafu gdzie miasteczka i schroniska to wierzchołki a szlaki to krawędzie). Każdy szlak ma przypisaną liczbę godzin potrzebnych, żeby go przebyć (są to liczby naturalne z zakresu od 1 do 16, zapisane jako wagi krawędzi grafu). Kodeks honorowy magicznych wojowników mówi, że wojownik nie może być w drodze bez odpoczynku dłużej niż 16 godzin. Taki odpoczynek musi trwać 8 godzin i musi się odbyć w schronisku. Wojownik chce się dostać z s do t jak najszybciej, ale nie może łamać zasad kodeksu. Gdy wojownik rusza z s jest w pełni wypoczęty, ale nie musi być wypoczęty gdy dotrze do t.

Proszę zaimplementować funkcję `warrior(G, s, t)`, która zwraca ile godzin trwa najszybsza droga wojownika z s do t, przy użyciu mapy opisanej jako graf G. Graf G reprezentowany jest jako lista krawędzi. Każda krawędź to trójka postaci (u, v, w), gdzie u i v to numery wierzchołków a w to liczba godzin potrzebna na przebycie drogi z u do v (oraz z v do u; graf jest nieskierowany). Numery wierzchołków to kolejne liczby naturalne od 0 do $n-1$, gdzie n to liczba wierzchołków. Algorytm powinien być możliwie jak najszybszy. Proszę uzasadnić poprawność zaproponowanego algorytmu oraz oszacować jego złożoność czasową i pamięciową.

Przykład 1. Dla wejścia:

```
G = [ (1,5,10), (4,6,12), (3,2,8),
      (2,4,4) , (2,0,10), (1,4,5),
      (1,0,6) , (5,6,8) , (6,3,9)]
s = 0
t = 6
```



wywołanie `warrior(G,s,t)` powinno zwrócić wartość 31, odpowiadającą trasie $0 \to 1 \to 4$(nocleg) $\to 6$, której przebycie trwa $6 + 5 + 8 + 12 = 31$ godzin.

Algorytmy i Struktury Danych
Kolokwium 3 (20 czerwca 2024r.)

Format rozwiązań

Wysłać należy tylko jeden plik: `kol3.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 kol3.py`

Szablon rozwiązania:	`kol3.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.

Algorytmy i Struktury Danych
Kolokwium uzupełniające (25 czerwca 2024r.)

Format rozwiązań

Wysłać należy tylko jeden plik: `kolu.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 kolu.py`

Szablon rozwiązania:	`kolu.py`
Złożoność akceptowalna (1.0pkt):	$O(n^2)$
Złożoność wzorcowa (+3.0pkt):	$O(n+m)$
Gdzie n to liczba projektów a m to liczba zależności.	

Bajtek musi wykonać w trakcie semestru n projektów: $p_0, p_1, \ldots, p_{n-1}$. Wykonanie każdego projektu trwa jedną jednostkę czasu. Co więcej, Bajtek ma bardzo podzielną uwagę: może wykonywać równocześnie dowolnie wiele projektów (ich równoczesna realizacja wciąż trwa jedną jednostkę czasu). Niestety, realizacja niektórych projektów musi być poprzedzona wykonaniem pewnych innych. Bajtek zastanawia się więc, ile czasu zajmie realizacja wszystkich projektów.

Zaproponuj i zaimplementuj algorytm, który wyliczy minimalną liczbę jednostek czasu w której możliwe jest wykonanie wszystkich n projektów. Algorytm należy zaimplementować jako funkcję `projects(n, L)`, której pierwszym argumentem jest liczba projektów a drugim argumentem jest lista zależności. Każdy element listy `L` to krotka postaci `(p, q)` wskazująca, że projekt numer $p \in \{0, \ldots, n-1\}$ można rozpocząć dopiero po wykonaniu projektu numer $q \in \{0, \ldots, n-1\}$. Można założyć, że istnieje taka kolejność wykonywania projektów, w których spełnione są wszystkie narzucone zależności. Zaproponowany algorytm powinien być możliwe jak najszybszy. Oszacuj jego złożoność obliczeniową.

Przykład. Dla wejścia:

```
n = 4
L = [(3, 1), (1, 2), (1, 0)]
```

wywołanie `projects(n, L)` powinno zwrócić wartość 3, odpowiadającą wykonaniu w pierwszej jednostce czasu projektów nr. 0 i 2, w drugiej jednostce czasu projektu numer 1 a w trzeciej jednostce czasu projektu numer 3.