Algorytmy i Struktury Danych
Egzamin I (30.06.2025)

Format rozwiązań

Wysłać należy tylko jeden plik: `egz1A.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`), słownik,
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 egz1A.py`

Szablon rozwiązania:	`egz1A.py`
Złożoność najmniej satysfakcjonująca (1 pkt):	$O((n+m)^3)$
Złożoność lepsza (2 pkt):	$O((n+m)^2)$
Złożoność akceptowalna (3 pkt):	$O((n+m)\log(n+m))$
Złożoność wzorcowa (4 pkt):	$O(n+m)$

Bitocja i Bajtocja tworzą wspólny spektakl—inscenizację wielkiej BITwy BITowej między tymi krajami. W ramach inscenizacji Bitocja rozstawiła n katapult na pozycjach $k_0, \ldots, k_{n-1}$, a Bajtocja rozstawiła m procesorów na pozycjach $p_0, \ldots, p_{m-1}$. Podczas spektaklu katapulty będą strzelać do procesorów. Reżyser wprowadził jednak pewne ograniczenia:

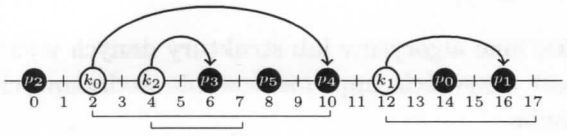
1. Każda katapulta może strzelać wyłącznie do przodu (czyli w kierunku pozycji większych niż jej własna) i ma ograniczony zasięg. Zasięgi katapult dane są jako liczby $r_0, \ldots, r_{n-1}$, po jednej dla każdej katapulty.
2. Strzały z katapult nie mogą się krzyżować. Przykładowo, jeśli katapulta na pozycji 5 strzela do procesora na pozycji 10 to katapulta na pozycji 2 nie może strzelać do procesora na pozycji 7.
3. Do jednego procesora może strzelać tylko jedna katapulta.

Nie jest wymagane, żeby wszystkie katapulty strzelały.

Proszę zaimplementować funkcję `battle(P,K,R)`, która przyjmuje listę pozycji procesorów (`P`), listę pozycji katapult (`K`), oraz listę zasięgów katapult (`R`) i zwraca maksymalną liczbę procesorów, do których mogą strzelać katapulty. Wejściowe listy nie są posortowane, ale wiadomo że pozycje katapult i procesorów są nieujemne i nie przekraczają liczby $4n+4m$. Żadne dwa obiekty (katapulty lub procesory) nie znajdują się na tej samej pozycji. 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
 P = [14, 16, 0, 6, 10, 8]
 K = [ 2, 12, 4]
 R = [ 8,  5, 3]
```



wynikiem jest 3. Na przykład katapulta k_0 może strzelać do procesora p_4, katapulta k_1 do procesora p_1, a katapulta k_2 do procesora p_3.

Algorytmy i Struktury Danych
Egzamin I (30.06.2025)

Format rozwiązań

Wysłać należy tylko jeden plik: `egz1B.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`), słownik,
2. korzystanie z wbudowanych funkcji sortujących (można założyć, że mają złożoność $O(n \log n)$).

Wszystkie inne algorytmy lub struktury danych (w tym słowniki) 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 egz1B.py`

Szablon rozwiązania:	`egz1B.py`
Złożoność podstawowa (2.0pkt):	$O(EV + E^2)$
Złożoność lepsza (3.0pkt):	$O(V^3)$
Złożoność wzorcowa (4.0pkt):	$O(EV + V^2)$
Gdzie V to liczba pracowników a E to liczba powiązań.	

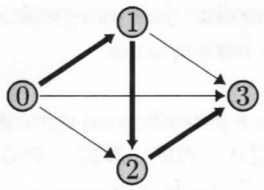
W Uczelni Doskonałości Naukowej funkcjonuje zakulisowy system delegowania zadań pomiędzy pracownikami. Każdy pracownik ma listę współpracowników, którym deleguje weryfikację wymyślonych hipotez badawczych. Pracownicy którzy otrzymają zadanie weryfikacji hipotezy często postępują podobnie, delegując weryfikację kolejnym współpracownikom. Przekazywanie zadań odbywa się w jednym kierunku. Co więcej, listy współpracowników są tak dobrane, że żaden pracownik nie może ponownie otrzymać zadania, które komuś delegował. Hipotezy trafiają ostatecznie do doktorantów, którzy nie mają ich komu delegować i badają je w ramach doktoratu.

Listy służące do delegowania zadań mają niestety pewne krytyczne elementy. Rozważmy parę pracowników (u, v) taką, że u może bezpośrednio delegować zadanie do v (a więc u ma v na swojej liście współpracowników). Parę taką będziemy nazywać **delegacją**. Mówimy, że delegacja (u, v) jest **krytyczna**, jeśli istnieje jakakolwiek para pracowników (a, b) taka, że po usunięciu delegacji (u, v) hipoteza podzlecana przez a nie może dotrzeć do b, podczas gdy pierwotnie było to możliwe (przy czym przejście od a do b mogło potencjalnie prowadzić przez łańcuch kilku współpracowników). Twoim zadaniem jest policzyć, ile takich **krytycznych delegacji** istnieje w systemie.

Proszę zaimplementować funkcję `critical(V, E)`, która zwraca liczbę krytycznych delegacji na uczelni. Argument V to liczba pracowników. Pracownicy numerowani są od 0 do $V - 1$. Argument E to lista delegacji. Każdy element tej listy to para postaci (u, v), oznaczająca, że pracownik u może bezpośrednio delegować zadanie pracownikowi v. Liczba par w liście E będzie zawsze większa niż V, a każda para pojawi się co najwyżej raz. Mamy pewność, że lista delegacji spełnia warunki opisane w treści zadania. Zaproponowany algorytm powinien być możliwie jak najszybszy. Proszę uzasadnić poprawność zaproponowanego algorytmu oraz oszacować jego złożoność czasową i pamięciową.

Przykład 1. Dla wejścia:

```
V = 4
E = [ (0,1), (0,2), (0,3),
      (1,2), (1,3), (2,3) ]
```



wywołanie `critical(V,E)` powinno zwrócić wartość **3** odpowiadającą liczbie istniejących w grafie delegacji krytycznych (zaznaczonych pogrubionymi krawędziami na rysunku powyżej). W podanym przykładzie zadania delegowane przez 0 mogą dotrzeć (potencjalnie pośrednio) do każdego innego pracownia. Pracownik 1 deleguje zadania pracownikom 2 i 3, zaś pracownik 2 tylko pracownikowi 3. Pracownik 3 nie deleguje już zadań nikomu. Delegacja $(0, 1)$ **jest krytyczna** ponieważ jej usunięcie powoduje, że pracownik 0 nie może delegować zadań pracownikowi 1, co przed usunięciem było możliwe. Natomiast delegacja $(0, 2)$ **nie jest krytyczna**, ponieważ zadania pracownika 0 nadal mogą docierać do uprzednio osiągalnych pracowników (po ścieżkach $0 \to 1$, $0 \to 1 \to 2$, oraz $0 \to 1 \to 3$). Jej usunięcie nie wpływa również na możliwość delegowania zadań przez pracowników 1, 2 i 3. Podobne rozumowanie pokazuje, że krytyczne są również delegacje $(1, 2)$ i $(2, 3)$.