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.