Algorytmy i Struktury Danych
Egzamin 1, zadania A i B (28 czerwca 2024r.)

Format rozwiązań

Wysłać należy tylko jeden plik: `egz1a.py` lub `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`),
2. korzystanie z wbudowanych funkcji sortujących (można założyć, że mają złożoność $O(n \log n)$),
3. korzystanie z zaawansowanych struktur danych (np. słowników czy 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.

Szablon rozwiązania:	`egz1a.py`
Złożoność akceptowalna (1.0pkt):	$O(VE \log V)$ lub $O(V^3)$
Złożoność wzorcowa (+3.0pkt):	$O(E \log V)$

Gdzie V to liczba wierzchołków w grafie a E to liczba krawędzi.
Należy założyć, że liczba rowerów jest co najwyżej rzędu $O(V)$.

Duathlon na orientację polega na tym, że zawodnik najpierw biegnie ze startu s do wybranego przez siebie roweru (dowolnego z wielu dostępnych), a następnie jedzie na tym rowerze do mety t (może też biec prosto do mety, nie biorąc roweru). Gdy zawodnik wybierze jakiś rower, to nie może go już zmienić (ale nie musi w danym punkcie brać roweru, nawet jak jest dostępny). Luiza Silnoręka startuje w takich zawodach i chce zaplanować to najszybsze pokonanie trasy. Trasa reprezentowana jest jako graf ważony, w którym wierzchołki to punkty orientacyjne (wliczając w to start s i metę t) a krawędzie to ścieżki, którymi można się między tymi punktami poruszać. Każda krawędź (ścieżka) ma czas wyrażony w minutach, jaki Luiza potrzebuje, żeby ją przebiec (są to zawsze liczby naturalne). W każdym punkcie orientacyjnym może być jeden, kilka, lub zero rowerów. Każdy rower opisany jest przez dwie liczby naturalne, p i q. Wiadomo, że jeśli Luiza potrzebuje x minut aby przebyć pewną ścieżkę, to na rowerze opisanym przez p i q przejedzie ją w czasie $x \cdot \frac{p}{q}$ (o ile możnaby oczekiwać, że $p < q$, to niektóre rowery są tak niewygodne, że ten warunek nie zachodzi).

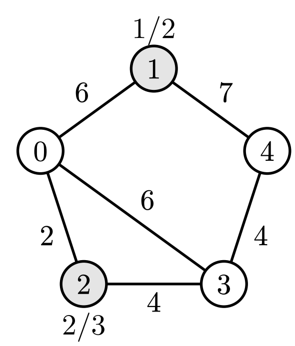
Proszę zaimplementować funkcję `armstrong(B, G, s, t)`, która otrzymuje na wejściu listę B opisującą dostępne rowery, graf G opisujący trasę, oraz numery wierzchołków s i t, a zwraca najmniejszą liczbę minut (zaokrągloną w dół), jaką Luiza potrzebuje na pokonanie trasy duathlonu.

Lista B zawiera trójki postaci (i, p, q), gdzie i to numer wierzchołka, w którym jest rower o parametrach p i q. W danym wierzchołku może być kilka rowerów i wówczas trójka z tą samą wartością i pojawia się w danych kilka razy.

Graf G ma wierzchołki o numerach od 0 do $n-1$, jest nieskierowany i jest reprezentowany przez listę krawędzi. Każda krawędź to trójka w postaci (u, v, w), gdzie u i v to numery wierzchołków, które łączy, a w to liczba minut, przez którą Luiza przebiega tę krawędź.

Przykład 1. Dla wejścia:

```
B = [ (1, 1, 2), (2, 2, 3) ]
G = [ (0,1,6), (1,4,7), (4,3,4),
      (3,2,4), (2,0,3), (0,3,6) ]
s = 0
t = 4
```



wywołanie `armstrong(B,G,s,t)` powinno zwrócić wartość 8: Luiza biegnie przez 3 minuty do punktu 2, wsiada na rower 2/3, jedzie do punktu 3 przez 8/3 minuty, na koniec jedzie do punktu 4 (czyli mety) przez kolejne 8/3 minuty. Jej łączny czas to $3 + 8/3 + 8/3 = 3 + 16/3 = 8 + 1/3$ minuty, czyli zaokrąglony w dół wynik to 8.

Szablon rozwiązania:	`egz1b.py`
Złożoność elementarna (0.5pkt):	$O(n^3 \log n)$
Złożóność lepsza (2.0pkt):	$O(n^2 \log n)$
Złożóność wzorcowa (4.0pkt):	$O(nk)$
Gdzie n to długość ciągu T.	

Dany jest ciąg liczb $T[0], ..., T[n-1]$. Mówimy, że dowolny jego podciąg jest k-spójny jeśli można go stworzyć poprzez wybranie pewnego zakresu elementów $T[i]$, $T[i+1]$, $T[i+2]$, ..., $T[j]$, a następnie usunięciu spośród nich co najwyżej k elementów.

Na przykład jeśli mamy ciąg $3,\ 1,\ -2,\ 7,\ 5,\ 10,\ -8,\ 4$ to $1,\ 5,\ 10$ jest jego 2-spójnym podciągiem: Można go stworzyć wybierając elementy $1,\ -2,\ 7,\ 5,\ 10$ a następnie usuwając -2 i 7.

Proszę zaimplementować funkcję `kstrong( T, k )`, która otrzymuje na wejściu ciąg T i zwraca maksymalną sumę k-spójnego podciągu T. Funkcja powinna być jak najszybsza.

Przykład 1. Dla wejścia:

```
T = [-20, 5, -1, 10, 2, -8, 10]
```

wywołanie `kstrong(T, 1)` powinno zwrócić wartość 26, odpowiadającą 1-spójnemu podciągowi $5,\ -1,\ 10,\ 2,\ 10$ (pomijamy wartość -8).

Algorytmy i Struktury Danych
Egzamin 2/Zaliczenie 3 (5 września 2024r.)

Format rozwiązań

Dla każdeg z zadań należy Wysłać tylko jeden plik.

Plik dla danego zadania 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 czy zbiorów),
4. korzystanie z biblioteki `itertools`.

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.

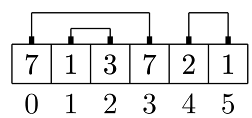
Szablon rozwiązania:	`egz2a.py`
Złożoność akceptowalna (1.0pkt):	$O(2^n \cdot \mathrm{poly}(n))$
Złożóność wzorcowa (+3.0pkt):	$O(n^3)$
Gdzie poly(n) to dowolny wielomian zmiennej n	

Pewien układ elektryczny ma $2n$ wejść ponumerowanych od 0 do $2n-1$. Wejścia należy połączyć przewodami. Do każdego wejścia powinien dochodzić jeden przewód i każdy przewód łączy dokładnie dwa wejścia. Oznacza to, że należy użyć w sumie n przewodów. Zasada działania układu wymaga, żeby przewody nie krzyżowały się, czyli jeśli połączymy przewodami wejście i oraz wejście j (gdzie $i < j$), to żadne z wejść od $i+1$ do $j-1$ nie może być połączone z żadnym z wejść od 0 do $i-1$ ani z żadnym z wejść od $j+1$ do $2n-1$. Dodatkowo dana jest tablica T, gdzie $T[i]$ to parametr mocy i-go wejścia. Kabel, który bezpiecznie łączy wejście i z wejściem j kosztuje $1 + |T[i] - T[j]|$.

Proszę zaimplementować funkcję `wired(T)`, która otrzymuje na wejściu listę T z parametrami mocy wejść, a zwraca minimalny koszt przewodów pozwalających na ich połączenie zgodnie z zasadami. Funkcja powinna być możliwie jak najszybsza.

Przykład 1. Dla wejścia:

```
T = [ 7, 1, 3, 7, 2, 1 ]
```



wywołanie `wired(T)` powinno zwrócić wartość 6: Łączymy wejścia 0 i 3 (koszt 1 + |7 - 7| = 1), 1 i 2 (koszt 1 + |1 - 3| = 3), oraz 4 i 5 (koszt 1 + |2 - 1| = 2)

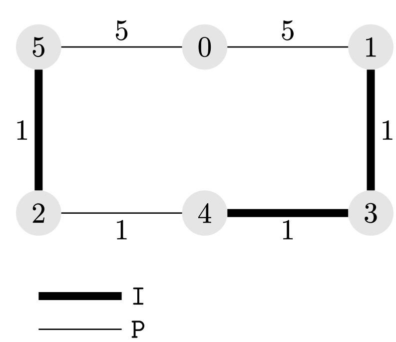
Szablon rozwiązania:	`egz2b.py`
Złożoność podstawowa (1.0pkt):	$O(m^2 \log(m))$
Złożoność lepsza (2.0pkt):	$O(m \log(m))$
Złożoność wzorcowa (4.0pkt):	$O(m)$
gdzie m to liczba linii kolejowych.	

Bajtocja to bardzo bogaty kraj. Pracują w niej inżynierowie z całego świata. Niestety, rodzi to czasem problemy. Okazuje się na przykład, że część linii kolejowych w Bajtocji ma indyjski rozstaw szyn (1676 mm) a reszta ma rozstaw przylądkowy (1067 mm). Główny Rozkładowy, inżynier Armin Mos, ma za zadanie zaplanować najkrótszą trasę ze stacji A do stacji B. Po drodze pociąg będzie potencjalnie musiał przejechać przez pewne inne stacje. Jeśli pociąg wjeżdża na stację i wyjeżdża z niej linią indyjską to przejazd zajmuje 5 jednostek czasu. Wjazd i wyjazd linią przylądkową zajmuje 10 jednostek czasu. Jeśli natomiast na stacji trzeba zmienić rozstaw kół to przejazd zajmuje 20 jednostek czasu. Pomiędzy stacjami pociąg jedzie z prędkością jednego kilometra na jednostkę czasu. Odległości pomiędzy stacjami to dodatnie liczby naturalne nie większe niż 10 kilometrów. Ruszając ze stacji A pociąg ma dopasowany rozkład kół do pierwszej linii, którą się porusza.

Proszę zaimplementować funkcję `tory_amos(E, A, B)`, która otrzymuje na wejściu opis linii kolejowych `E`, numer stacji początkowej `A` oraz numer stacji końcowej `B` i zwraca najkrótszy możliwy czas przejazdu z `A` do `B`. Opis linii kolejowych to lista krotek postaci `(x, y, d, typ)`, gdzie `x` i `y` to numery stacji połączonych linią kolejową, `d` $\in \{1, ..., 10\}$ to długość linii zaś `typ` to jej rozstaw. Jeśli `typ == 'I'` to linia ma rozstaw indyjski. Jeśli `typ == 'P'` to linia ma rozstaw przylądkowy. Każda linia kolejowa jest dwukierunkowa (ma tor w kierunku `x->y` i tor w kierunku `y->x`). Może się zdarzyć, że pewne stacje są połączone zarówno linią indyjską jak i linią przylądkową. Stacje numerowane są od 0.

Przykład 1. Dla wejścia:

```
E = [(0, 1, 5, 'P'), (1, 3, 1, 'I'), (3, 4, 1, 'I'),
     (2, 4, 1, 'P'), (2, 5, 1, 'I'), (0, 5, 5, 'P')]
A = 5
B = 3
```



wywołanie `tory_amos(E, A, B)` powinno zwrócić wartość 41, co odpowiada:

- przejechaniu z 5 do 0 linią przylądkową (5 jednostek czasu),
- przejechaniu stacji 0 linią przylądkową (10 jednostek),
- przejechaniu z 0 do 1 linią przylądkową (5 jednostek),
- zmienia rozstawu kół na stacji 1 (20 jednostek),
- przejechaniu z 1 do 3 linią indyjską (1 jednostka).

Algorytmy i Struktury Danych
Egzamin 3 (12 września 2024r.)

Format rozwiązań

Dla każdego zadania należy wysłać tylko jeden plik.

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 czy 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.

Szablon rozwiązania:	`egz3a.py`
Złożoność podstawowa (1.0pkt):	$O(V^3)$
Złożoność lepsza (2.0pkt):	$O(VE)$
Złożoność wzorcowa (4.0pkt):	$O(V+E)$

gdzie V to liczba drzew a E to liczba połączeń między drzewami.

Planeta Pandora jest porośnięta lasem deszczowym posiadającym rozległy system korzeni. System korzeniowy obejmujący cały las stanowi graf, którego wierzchołki reprezentują drzewa, mogące łączyć się systemami korzeniowymi z innymi drzewami, co jest reprezentowane przez krawędzie grafu. Aby zbadać system korzeni wybrano k drzew i wszczepiono im k różnych gatunków grzyba numerowanych od 0 do $k-1$. W jednostce czasu grzyb rozrasta się z drzewa na wszystkie z którymi bezpośrednio łączy się korzeniami. Jeśli dwa lub więcej gatunków grzyba docierają do drzewa w tej samej jednostce czasu, „wygrywa" ten z najmniejszym indeksem. Proszę zaproponować i zaimplementować algorytm wyznaczający ile drzew docelowo zostanie opanowanych przez grzyb o numerze d.

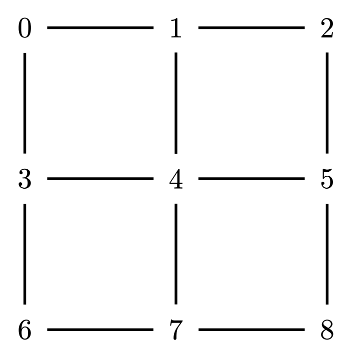
Algorytm należy zaimplementować jako funkcję `mykoryza(G, T, d)`, której argumentami są:

- graf G w postaci list sąsiadów opisujący system korzeni,
- tablica T zawierająca numery drzew, którym wszczepiono grzyby,
- d numer grzyba.

Funkcja powinna zwrócić liczbę drzew opanowanych przez grzyb numer d.

Przykład. Dla wejścia:

```
G = [[1,3],[0,2,4],[1,5],
     [0,4,6],[1,3,5,7],[2,4,8],
     [3,7],[4,6,8],[7,5]]
T = [8,2,6]
d = 1
```



wywołanie `mykoryza(G, T, d)` powinno zwrócić wartość 3.
Opanowane grzybem nr 1 będą drzewa: 0,1,2.

Szablon rozwiązania:	`egz3b.py`
Złożoność podstawowa (1.0pkt):	$O(n^2)$
Złożoność lepsza (2.0pkt):	$O(n \log(n))$
Złożoność wzorcowa (4.0pkt):	$O(n)$
gdzie n to liczba elementów tablicy.	

Liczbą k-pechową (gdzie k jest liczbą naturalną) nazywamy dowolny element ciągu:

$$x_1 = k,$$
$$x_{i+1} = x_i + (x_i \ \% \ i) + 7,$$

gdzie % oznacza resztę z dzielenia. W eksperymencie prof. Bitowego powstała n-elementowa tablica T liczb ze zbioru $\{1, ...n\}$. Próbując zrozumieć wynik eksperymentu, prof. Bitowy poszukuje najdłuższego spójnego fragmentu tablicy T, który zawiera nie więcej niż dwie liczby k-pechowe. Proszę zaproponować i zaimplementować algorytm wyznaczający długość takiego fragmentu tablicy.

Algorytm należy zaimplementować jako funkcję `kunlucky(T, k)`. której argumentami są: tablica T liczb z eksperymentu profesora Bitowego oraz stała k w definicji liczby k-pechowej. Zaproponowany algorytm powinien być możliwie jak najszybszy. Proszę uzasadnić jego poprawność i oszacować złożoność czasową.

Przykład. Dla wejścia:

```
                #2                                                                     #18
    T = [11, 10, 19, 19, 17, 16, 3, 9, 6, 14, 13, 8, 2, 13, 11, 12, 5, 5, 5]
k = 3
```

wywołanie `kunlucky(T, k)` powinno zwrócić wartość 17 odpowiadającą długości przedziału od indeksu 2 do indeksu 18 włącznie. Podkreśleniem zaznaczono liczby k-pechowe (dla $k = 3$). Indeksy nad tablicą wskazują granice szukanego fragmentu tablicy.