Algorytmy i Struktury Danych
Zadanie offline 1 (7.III.2022)

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 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. 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,
2. korzystanie ze struktur danych dostarczonych razem z zadaniem (jeśli takie są).

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 zad1.py`

Zadanie offline 1.

Szablon rozwiązania: `zad1.py`

Węzły jednokierunkowej listy odsyłaczowej reprezentowane są w postaci:

```
class Node:
    def __init__(self):
        self.val  = None  # przechowywana liczba rzeczywista
        self.next = None  # odsyłacz do nastepnego elementu
```

Niech p będzie wskaźnikiem na niepustą listę odsyłaczową zawierającą parami różne liczby rzeczywiste $a_1, a_2, \ldots, a_n$ (lista nie ma wartownika). Mówimy, że lista jest k-chaotyczna jeśli dla każdego elementu zachodzi, że po posortowaniu listy znalazłby się na pozycji różniącej się od bieżącej o najwyżej k. Tak więc 0-chaotyczna lista jest posortowana, przykładem 1-chaotycznej listy jest $1, 0, 3, 2, 4, 6, 5$, a $(n-1)$-chaotyczna lista długości n może zawierać liczby w dowolnej kolejności. Proszę zaimplementować funkcję `SortH(p,k)`, która sortuje k-chaotyczną listę wskazywaną przez p. Funkcja powinna zwrócić wskazanie na posortowaną listę. Algorytm powinien być jak najszybszy oraz używać jak najmniej pamięci (w sensie asymptotycznym, mierzonym względem długości n listy oraz parametru k). Proszę skomentować jego złożoność czasową dla $k = \Theta(1)$, $k = \Theta(\log n)$ oraz $k = \Theta(n)$.

Algorytmy i Struktury Danych
Zadanie offline 2 (14.III.2022)

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 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. 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,
2. korzystanie ze struktur danych dostarczonych razem z zadaniem (jeśli takie są).

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 zad2.py`

Zadanie offline 2.

Szablon rozwiązania: `zad2.py`

Dany jest ciąg przedziałów domkniętych $L = [[a_1, b_1], \ldots, [a_n, b_n]]$. Początki i końce przedziałów są liczbami naturalnymi. Poziomem przedziału $c \in L$ nazywamy liczbę przedziałów w L, które w całości zawierają się w c (nie licząc samego c). Proszę zaproponować i zaimplementować algorytm, który zwraca maksimum z poziomów przedziałów znajdujących się w L. Proszę uzasadnić poprawność algorytmu i oszacować jego złożoność obliczeniową.

Algorytm należy zaimplementować jako funkcję postaci:

```
def depth( L ):
  ...
```

która przyjmuje listę przedziałów `L` i zwraca maksimum z poziomów przedziałów w `L`.

Przykład. Dla listy przedziałów:

```
L = [ [1, 6],
      [5, 6],
      [2, 5],
      [8, 9],
      [1, 6]]
```

wynikiem jest liczba 3.

Algorytmy i Struktury Danych
Zadanie offline 3 (21.III.2022)

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 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. 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,
2. korzystanie ze struktur danych dostarczonych razem z zadaniem (jeśli takie są).

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 zad3.py`

Zadanie offline 3.

Szablon rozwiązania: `zad3.py`

Mamy daną N elementową tablicę T liczb rzeczywistych, w której liczby zostały wygenerowane z pewnego rozkładu losowego. Rozkład ten mamy zadany jako k przedziałów $[a_1, b_1], [a_2, b_2], \ldots, [a_k, b_k]$ takich, że i-ty przedział jest wybierany z prawdopodobieństwem c_i, a liczba z przedziału ($x \in [a_i, b_i]$) jest losowana zgodnie z rozkładem jednostajnym. Przedziały mogą na siebie nachodzić. Liczby a_i, b_i są liczbami naturalnymi ze zbioru $\{1, \ldots, N\}$.

Proszę zaimplementować funkcję `SortTab(T, P)` sortująca podaną tablicę i zwracająca posortowaną tablicę jako wynik. Pierwszy argument to tablica do posortowania a drugi to opis przedziałów w postaci:

```
P = [(a_1,b_1,c_1), (a_2,b_2,c_2), ..., (a_k,b_k,c_k)]}.
```

Na przykład dla wejścia:

```
T = [6.1, 1.2, 1.5, 3.5, 4.5, 2.5, 3.9, 7.8]
P = [(1, 5, 0.75) , (4, 8, 0.25)]
```

po wywołaniu `SortTab(T,P)` tablica zwrócona w wyniku powinna mieć postaci:

```
T = [1.2, 1.5, 2.5, 3.5, 3.9, 4.5, 6.1, 7.8]
```

Algorytm powinien być możliwie jak najszybszy. Proszę podać złożoność czasową i pamięciową zaproponowanego algorytmu.

Algorytmy i Struktury Danych
Zadanie offline 4 (19.IV.2022)

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,
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 zad4.py`

Zadanie offline 4.

Szablon rozwiązania: `zad4.py`

Inwestor planuje wybudować nowe osiedle akademików. Architekci przedstawili projekty budynków, z których inwestor musi wybrać podzbiór spełniając jego oczekiwania. Każdy budynek reprezentowany jest jako prostokąt o pewnej wysokości h, podstawie od punktu a do punktu b, oraz cenie budowy w (gdzie h, a, b i w to liczby naturalne, przy czym $a < b$). W takim budynku może mieszkać $h \cdot (b - a)$ studentów.

Proszę zaimplementować funkcję:

```
def select_buildings(T, p):
    ...
```

która przyjmuje:

- Tablicę `T` zawierająca opisy `n` budynków. Każdy opis to krotka postaci `(h, a, b, w)`, zgodnie z oznaczeniami wprowadzonymi powyżej.
- Liczbę naturalną `p` określającą limit łącznej ceny wybudowania budynków.

Funkcja powinna zwrócić tablicę z numerami budynków (zgodnie z kolejnością w `T`, numerowanych od 0), które nie zachodzą na siebie, kosztują łącznie nie więcej niż `p` i mieszczą maksymalną liczbę studentów. Jeśli więcej niż jeden zbiór budynków spełnia warunki zadania, funkcja może zwrócić dowolny z nich. Dwa budynki nie zachodzą na siebie, jeśli nie mają punktu wspólnego.

Można założyć, że zawsze istnieje rozwiązanie zawierające co najmniej jeden budynek. Funkcja powinna być możliwie jak najszybsza i zużywać jak najmniej pomięci. Należy bardzo skrótowo uzasadnić jej poprawność i oszacować złożoność obliczeniową.

Przykład. Dla argumentów:

```
T = [ (2, 1, 5, 3),
      (3, 7, 9, 2),
      (2, 8, 11, 1) ]
p = 5
```

wynikiem może być tablica: `[ 0, 2 ]`

Algorytmy i Struktury Danych
Zadanie offline 5 (25.IV.2022)

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`)**,
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 zad5.py`

Zadanie offline 5.

Szablon rozwiązania: `zad5.py`

W roku 2050 spokojny Maksymilian odbywa podróż przez pustynię z miasta A do miasta B. Droga pomiędzy miastami jest linią prostą na której w pewnych miejscach znajdują się plamy ropy. Maksymilian porusza się 24 kołową cysterną, która spala 1 litr ropy na 1 kilometr trasy. Cysterna wyposażona jest w pompę pozwalającą zbierać ropę z plam. Aby dojechać z miasta A do miasta B Maksymilian będzie musiał zebrać ropę z niektórych plam (by nie zabrakło paliwa), co każdorazowo wymaga zatrzymania cysterny. Niestety, droga jest niebezpieczna. Maksymilian musi więc tak zaplanować trasę, by zatrzymać się jak najmniej razy. Na szczęście cysterna Maksymiliana jest ogromna - po zatrzymaniu zawsze może zebrać całą ropę z plamy (w cysternie zmieściłaby się cała ropa na trasie).

Zaproponuj i zaimplementuj algorytm wskazujący, w których miejscach trasy Maksymilian powinien się zatrzymać i zebrać ropę. Algorytm powinien być możliwe jak najszybszy i zużywać jak najmniej pamięci. Uzasadnij jego poprawność i oszacuj złożoność obliczeniową.

Dane wejściowe reprezentowane są jako tablica liczb naturalnych `T`, w której wartość `T[i]` to objętość ropy na polu numer `i` (objętość 0 oznacza brak ropy). Pola mają numery od 0 do $n-1$ a odległość między kolejnymi polami to 1 kilometr. Miasto A znajduje się na polu 0 a miasto B na polu $n-1$. Zakładamy, że początkowo cysterna jest pusta, ale pole 0 jest częścią plamy ropy, którą można zebrać przed wyruszeniem w drogę. Zakładamy również, że zadanie posiada rozwiązanie, t.j. da się dojechać z miasta A do miasta B.

Algorytm należy zaimplementować w funkcji:

```
def plan(T):
   ...
```

która przyjmuje tablicę z opisem pół `T[0]`, ..., `T[n-1]` i zwraca listę pól, na których należy się zatrzymać w celu zebrania ropy. Lista powinna być posortowana w kolejności postojów. Postój na polu 0 również jest częścią rozwiązania.

Przykład. Dla wejścia:

```
#    0 1 2 3 4 5 6 7
T = [3,0,2,1,0,2,5,0]
```

wynikiem jest np. lista `[0,2,5]`.

Algorytmy i Struktury Danych
Zadanie offline 6 (2.V.2022)

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` lub `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 zad6.py`

Zadanie offline 6.

Szablon rozwiązania: `zad6.py`

Dany jest graf nieskierowany $G = (V, E)$ oraz dwa wierzchołki $s, t \in V$. Proszę zaproponować i zaimplementować algorytm, który sprawdza, czy istnieje taka krawędź $\{p, q\} \in E$, której usunięcie z E spowoduje wydłużenie najkrótszej ścieżki między s a t (usuwamy tylko jedną krawędź). Algorytm powinien być jak najszybszy i używać jak najmniej pamięci. Proszę skrótowo uzasadnić jego poprawność i oszacować złożoność obliczeniową.

Algorytm należy zaimplementować jako funkcję:

```
def longer(G, s, t):
   ...
```

która przyjmuje graf `G` oraz numery wierzchołków `s`, `t` i zwraca dowolną krawędź spełniającą warunki zadania, lub `None` jeśli takiej krawędzi w `G` nie ma. Graf przekazywany jest jako lista list sąsiadów, t.j. `G[i]` to lista sąsiadów wierzchołka o numerze `i`. Wierzchołki numerowane są od 0. Funkcja powinna zwrócić krotkę zawierającą numery dwóch wierzchołków pomiędzy którymi jest krawędź spełniająca warunki zadania, lub `None` jeśli takiej krawędzi nie ma. Jeśli w grafie oryginalnie nie było ścieżki z `s` do `t` to funkcja powinna zwrócić `None`.

Przykład. Dla argumentów:

```
G = [ [1, 2],
      [0, 2],
      [0, 1] ]
s = 0
t = 2
```

wynikiem jest np. krotka: `(0, 2)`

Algorytmy i Struktury Danych
Zadanie offline 7 (16.V.2022)

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` lub `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 zad7.py`

Zadanie offline 7.

Szablon rozwiązania: `zad7.py`

Dane jest N miast, każde miasto jest otoczone murem, w którym znajdują się dwie bramy: północna i południowa. Jeżeli przyjedziemy do miasta jedną z bram musimy wyjechać z niego tą drugą. Wyjeżdżając każdą z bram można dojechać bezpośrednio do jednego lub więcej innych miast. Proszę zaproponować i zaimplementować algorytm który sprawdza czy wyruszając z jednego z miast można odwiedzić wszystkie miasta dokładnie jeden raz i powrócić do miasta, z którego się wyruszyło. Algorytm powinien być jak najszybszy i używać jak najmniej pamięci. Proszę skrótowo uzasadnić jego poprawność i oszacować złożoność obliczeniową.

Algorytm należy zaimplementować jako funkcję:

```
def droga(G):
   ...
```

która przyjmuje sieć połączeń `G` pomiędzy miastami i zwraca listę numerów odwiedzanych miast albo `None` jeśli takiej drogi nie ma. Sieć połączeń jest dana w postaci listy, która dla każdego miasta zawiera dwie listy: miast dostępnych z bramy północnej oraz miast dostępnych z bramy południowej. Miasta numerowane są od 0.

Przykład. Dla argumentów:

```
G = [ ([1],[2,3,4]),
      ([0],[2,5,6]),
      ([1,5,6],[0,3,4]),
      ([0,2,4],[5,7,8]),
      ([0,2,3],[6,7,9]),
      ([1,2,6],[3,7,8]),
      ([1,2,5],[4,7,9]),
      ([4,6,9],[3,5,8]),
      ([3,5,7],[9]),
      ([4,6,7],[8]) ]
```

wynikiem jest np. lista: ([0, 1, 5, 7, 9, 8, 3, 2, 6, 4])

Algorytmy i Struktury Danych
Zadanie offline 8 (23.V.2022)

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` lub `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 zad8.py`

Zadanie offline 8.

Szablon rozwiązania: `zad8.py`

W pewnym państwie, w którym znajduje się N miast, postanowiono połączyć wszystkie miasta siecią autostrad, tak aby możliwe było dotarcie autostradą do każdego miasta. Ponieważ kontynent, na którym leży państwo jest płaski położenie każdego z miast opisują dwie liczby x, y, a odległość w linii prostej pomiędzy miastami liczona w kilometrach wyraża się wzorem $len = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. Z uwagi na oszczędności materiałów autostrada łączy dwa miasta w linii prostej.

Ponieważ zbliżają się wybory prezydenta, wszystkie autostrady zaczęto budować równocześnie i jako cel postanowiono zminimalizować czas pomiędzy otwarciem pierwszej i ostatniej autostrady. Czas budowy autostrady wyrażony w dniach wynosi $\lceil len \rceil$ (sufit z długości autostrady wyrażonej w km).

Proszę zaimplementować funkcję `highway(A)`, która dla danych położeń miast wyznacza minimalną liczbę dni dzielącą otwarcie pierwszej i ostatniej autostrady.

Przykład Dla tablicy `A =[(10,10),(15,25),(20,20),(30,40)]` wynikiem jest 7 (Autostrady pomiędzy miastami 0-1, 0-2, 1-3).

Algorytmy i Struktury Danych
Zadanie offline 9 (30.V.2022)

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` lub `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 zad9.py`

Zadanie offline 9.

Szablon rozwiązania: `zad9.py`

W pewnym państwie znajdują się miasta, połączone siecią jednokierunkowych rurociągów, każdy o określonej przepustowości. Złoża ropy zostały wyczerpane, jednak w jednym z miast odkryto niewyczerpane źródło nowego rodzaju paliwa. Postanowiono zbudować dwie fabryki w różnych miastach oczyszczające nowe paliwo. Z pewnych względów fabryki te nie mogą znajdować się w mieście, w którym odkryto nowe złoża i nowe paliwo będzie transportowane istniejącą siecią rurociągów. Należy wskazać dwa miasta w których należy zbudować fabryki aby zmaksymalizować produkcję oczyszczonego paliwa.

Proszę zaimplementować funkcję `maxflow(G,s)`, która dla istniejącej sieci rurociągów `G` i miasta, w którym odkryto złoże `s`, zwróci maksymalną łączną przepustowość do dwóch miast w których należy zbudować fabryki. Miasta są ponumerowane kolejnymi liczbami $0, 1, 2, ...$ Sieć rurociągów opisuje lista trójek: (miasto w którym rozpoczyna się rurociąg, miasto w którym się kończy rurociąg, przepustowość rurociągu)

Przykład Dla sieci `G = [(0,1,7),(0,3,3),(1,3,4),(1,4,6),(2,0,9),(2,3,7),(2,5,9),(3,4,9),(3,6,2),(5,3,3),(5,6,4),(6,4,8)]` oraz miasta `s=2` wynikiem jest 25 (miasta 4 i 5).