Algorytmy i Struktury Danych
Zadanie offline 1 (4.III.2024)

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 2 (11.III.2024)

Format rozwiązań

Wysłać należy tylko jeden plik: `zad2.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, 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

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 zad2.py`

Zadanie offline 2.

Szablon rozwiązania: `zad2.py`

Dana jest n-elementowa tablica liczb naturalnych `T` oraz dodatnie liczby naturalne k i p, gdzie $k \leq p \leq n$. Niech z_i będzie k-tym największym spośród elementów: `T[i]`, `T[i+1]`, ..., `T[i+p-1]`. Innymi słowy, z_i to k-ty największy element w `T` w przedziale indeksów od i do $i+p-1$ włącznie.

Doprecyzowanie: Rozważmy tablicę `[17,25,25,30]`. W tej tablicy 1-wszy największy element to 30, 2-gi największy element to 25, 3-ci największy element to także 25 (drugie wystąpienie), a 4-ty największy element to 17.

Proszę zaimplementować funkcję `ksum(T, k, p)`, która dla tablicy `T` (o rozmiarze n elementów) i dodatnich liczb naturalnych k i p ($k \leq p \leq n$) wylicza i zwraca wartość sumy:

$$z_0 + z_1 + z_2 + \ldots + z_{n-p}$$

Przykład. Dla wejścia:

```
T = [7,9,1,5,8,6,2,12]
k = 4
p = 5
```

wywołanie `ksum(T, k, p)` powinno zwrócić wartość 17 (odpowiadającą sumie $5+5+2+5$).
Algorytm powinien być możliwie jak najszybszy. Proszę uzasadnić poprawność zaproponowanego algorytmu oraz oszacować jego złożoność czasową i pamięciową.

Algorytmy i Struktury Danych
Zadanie 3 (18.III.2024)

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, 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.

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: `python zad3.py`

Zadanie offline 3.

Szablon rozwiązania: `zad3.py`

Dany jest zbiór $P = \{p_1, \ldots, p_n\}$ punktów na płaszczyźnie. Współrzędne punktów to liczby naturalne ze zbioru $\{1, \ldots, n\}$. Mówimy, że punkt $p_i = (x_i, y_i)$ dominuje punkt $p_j = (x_j, y_j)$ jeśli zachodzi:

$$x_i > x_j \text{ oraz } y_i > y_j.$$

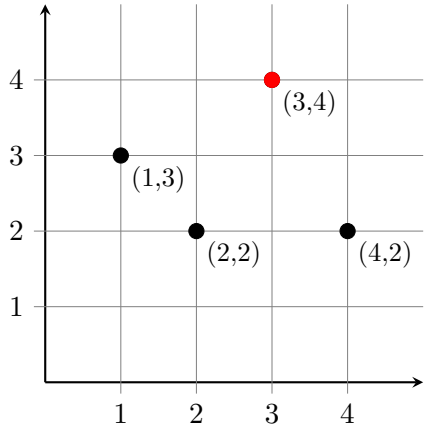
Siłą danego punktu jest to ile punktów dominuje. Zadanie polega na implementacji funkcji:

```
dominance( P )
```

która na wejściu otrzymuje listę `P` zawierającą n punktów (każdy reprezentowany jako para liczb ze zbioru $\{1, \ldots, n\}$) i zwraca siłę najsilniejszego z nich. Funkcja powinna być możliwie jak najszybsza.

Przykład. Dla wejścia:

```
P = [(1,3),
     (3,4),
     (4,2),
     (2,2)]
```



wynikiem jest 2. Punkt o współrzędnych $(3, 4)$ dominuje punkty o współrzędnych $(1, 3)$ oraz $(2, 2)$.

Algorytmy i Struktury Danych
Zadanie offline 1 (15.IV.2024)

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 zad4.py`

Zadanie offline 4.

Szablon rozwiązania: `zad4.py`

Dany jest nieskierowany graf $G = (V, E)$, którego wierzchołki reprezentują punkty nawigacyjne nad Bajtocją, a krawędzie reprezentują korytarze powietrzne między tymi punktami. Każdy korytarz powietrzny $e_i \in E$ powiązany jest z optymalnym pułapem przelotu $p_i \in \mathbb{N}$ (wyrażonym w metrach). Przepisy dopuszczają przelot danym korytarzem, jeśli pułap samolotu różni się od optymalnego najwyżej o t metrów. Graf jest dany w postaci listy krawędzi $L = [(u_1, v_1, p_1), (u_2, v_2, p_2), ...(u_n, v_n, p_n)]$, gdzie: $u_i, v_i \in \mathbb{N}$ to numery punktów nawigacyjnych, a p_i to optymalny pułap przelotu. Ponieważ graf jest nieskierowany na liście L umieszczono wyłącznie krotki, w których $u_i < v_i$

Proszę zaimplementować funkcję `Flight(L,x,y,t)`, która sprawdza czy istnieje możliwość przelotu z zadanego punktu $x \in V$ do zadanego punktu $y \in V$ w taki sposób, żeby samolot nigdy nie zmieniał pułapu. Algorytm powinien być poprawny i możliwie jak najszybszy. Proszę oszacować jego złożoność czasową.

Przykład

Dla danych wejściowych:

```
L = [(0,1,2000),(0,2,2100),(1,3,2050),(2,3,2300),
     (2,5,2300),(3,4,2400),(3,5,1990),(4,6,2500),(5,6,2100)]

x = 0
y = 6
t = 60
```

Poprawną odpowiedzią jest: $True$, lot na pułapie 2045, wiedzie przez punkty 0 1 3 5 6.
W przypadku gdyby $t = 50$, odpowiedzią powinno być $False$.

Algorytmy i Struktury Danych
Zadanie offline 5 (22.IV.2024)

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. 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: `python3 zad5.py`

Zadanie offline 5.

Szablon rozwiązania: `zad5.py`

Układ planetarny Algon składa się z n planet o numerach od 0 do $n-1$. Niestety własności fizyczne układu powodują, że nie da się łatwo przelecieć między dowolnymi dwiema planetami. Na szczęście mozolna eksploracja kosmosu doprowadziła do stworzenia listy E dopuszczalnych bezpośrednich przelotów. Każdy element listy E to trójka postaci (u, v, t), gdzie u i v to numery planet (można założyć, że $u < v$) a t to czas podróży między nimi (przelot z u do v trwa tyle samo co z v do u). Dodatkową nietypową własnością układu Algon jest to, że niektóre planety znajdują się w okolicy osobliwości. Znajdując się przy takiej planecie możliwe jest zagięcie czasoprzestrzeni umożliwiające przedostanie się do dowolnej innej planety leżącej przy osobliwości w czasie zerowym.

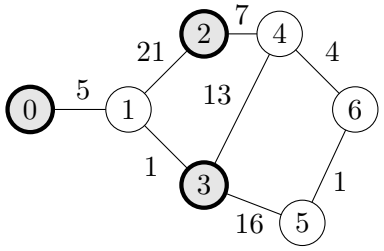
Zadanie polega na zaimplementowaniu funkcji:

```
def spacetravel( n, E, S, a, b )
```

która zwraca najkrótszy czas podróży z planety a do planety b, mając do dyspozycji listę możliwych bezpośrednich przelotów E oraz listę S planet znajdujących się koło osobliwości. Jeśli trasa nie istnieje, to funkcja powinna zwrócić `None`.

Rozważmy następujące dane:

```
E = [(0,1, 5),
     (1,2,21),
     (1,3, 1),
     (2,4, 7),
     (3,4,13),
     (3,5,16),
     (4,6, 4),
     (5,6, 1)]
S = [ 0, 2, 3 ]
a = 1
b = 5
n = 7
```



wywołanie `startravel(n, E, S, a, b)` powinno zwrócić liczbę 13. Odwiedzamy po kolei planety 1, 3, 2, 4, 6 i kończymy na planecie 5 (z planety 2 do 3 dostajemy się przez zagięcie czasoprzestrzeni). Gdyby $a = 1$ a $b = 2$ to wynikiem byłby czas przelotu 1.

Algorytmy i Struktury Danych
Zadanie offline 6 (6.V.2024)

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. 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: `python3 zad6.py`

Zadanie offline 6.

Szablon rozwiązania: `zad6.py`

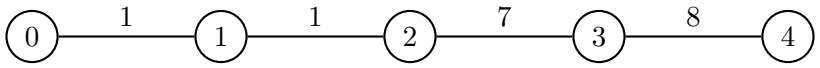
Dany jest ważony, nieskierowany graf G oraz *dwumilowe buty* - specjalny sposób poruszania się po grafie. Dwumilowe buty umożliwiają pokonywanie ścieżki złożonej z dwóch krawędzi grafu tak, jakby była ona pojedynczą krawędzią o wadze równej maksimum wag obu krawędzi ze ścieżki. Istnieje jednak ograniczenie - pomiędzy każdymi dwoma użyciami *dwumilowych butów* należy przejść w grafie co najmniej jedną krawędź w sposób zwyczajny. Macierz G zawiera wagi krawędzi w grafie, będące liczbami naturalnymi, wartość 0 oznacza brak krawędzi.
Proszę opisać, zaimplementować i oszacować złożoność algorytmu znajdowania najkrótszej ścieżki w grafie z wykorzystaniem mechanizmu *dwumilowych butów*.
Rozwiązanie należy zaimplementować w postaci funkcji:

```
def jumper(G, s, w):
    ...
```

która zwraca długość najkrótszej ścieżki w grafie G pomiędzy wierzchołkami s i w, zgodnie z zasadami używania *dwumilowych butów*.

Zaimplementowana funkcja powinna być możliwie jak najszybsza. Proszę przedstawić złożoność czasową oraz pamięciową użytego algorytmu.

Przykład: Rozważmy następujący graf:



Najkrótszą ścieżką między wierzchołkami 0 i 4 wykorzystującą *dwumilowe buty* będzie ścieżka $[0, 1, 2, 4]$ o długości 10 (z krawędzią $(2, 4)$ będącą dwumilowym skokiem). Ścieżka złożona z dwóch dwumilowych skoków, $[0, 2, 4]$, byłaby krótsza, ale nie spełnia warunków zadania.

Algorytmy i Struktury Danych
Zadanie offline 7 (3.VI.2024)

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, kolejka `collections.deque`, kolejka priorytetowa (`queue.PriorityQueue, 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`

Indianin Jonasz wędruje po labiryncie szukając śladów dawnej cywilizacji. Labirynt jest kwadratowy i złożony z $n \times n$ komnat. Jonasz rozpoczyna wędrówkę w komnacie o współrzędnych $(0,0)$ znajdującej się na planie w lewym górnym rogu i musi dotrzeć do komnaty o współrzędnych $(n-1, n-1)$ w prawym dolnym rogu. Niektóre komnaty (zaznaczone na planie znakiem #) są niedostępne i nie można do nich się dostać. Jonaszowi wolno poruszać się tylko w trzech kierunkach, opisanych na planie jako Góra, Prawo i Dół oraz nie wolno mu wrócić do komnaty w której już był. Jonasz chce odwiedzić jak największej komnat, żeby potencjalnie znaleźć jak najwięcej interesujących artefaktów. Zadanie polega na zaimplementowaniu funkcji:

```
def maze ( L )
```

która otrzymuje na wejściu tablicę L opisującą labirynt i zwraca największą liczbę komnat, które może odwiedzić Jonasz na swojej drodze lub -1 jeśli dotarcie do końca drogi jest niemożliwe. Komnaty początkowej nie liczymy jako odwiedzonej. Funkcja powinna być możliwie jak najszybsza.

Labirynt opisuje lista $L = [W_0, W_1, W_2, \ldots, W_{n-1}]$, gdzie każde W_i to napis o długości n znaków. Znak kropki '.' oznacza dostępną komnatę a znak '#'oznacza komnatę niedostępną.

Przykład. Rozważmy następujący labirynt:

```
L = [ "....",
      "..#.",
      "..#.",
      "...." ]
```

Optymalna droga wojownika to: DDDPGGGPPDDD, podczas której Wojownik odwiedził 12 komnat.

Algorytmy i Struktury Danych
Zadanie offline 8 (10.VI.2024)

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, kolejka `collections.deque`, kolejka priorytetowa (`queue.PriorityQueue, 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`

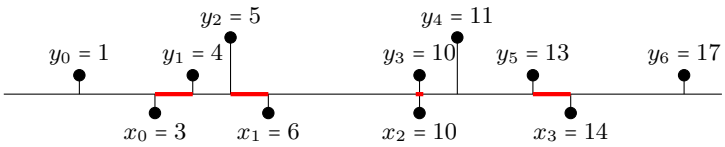
Szalony Inwestor wybudował po południowej stronie drogi n biurowców, na pozycjach $x_0 < \cdots < x_{n-1}$. Parkingi tych biurowców mają dopiero zostać wybudowane i dostępne jest w tym celu m działek ($m \geq n$), dostępnych na północnej stronie drogi, na pozycjach $y_0 < \cdots < y_{m-1}$. Inwestor chce wybudować dokładnie po jednym parkingu dla każdego biurowca (żadne dwa biurowce nie mogą dzielić tego samego parkingu). Zasady bezpiecznego ruchu wymagają, że i-ty biurowiec musi mieć parking na pozycji wcześniejszej niż $i+1$-szy. Inwestor chce wybudować parkingi na takich pozycjach, żeby suma odległości parkingów od biurowców była minimalna. Odległość i-go biurowca od j-ej działki to $|x_i - y_j|$. Zadanie polega na implementacji funkcji:

```
parking( X, Y )
```

która na wejściu otrzymuje listę `X` zawierającą n pozycji biurowców oraz listę `Y` zawierającą m pozycji działek na parkingi (listy `X` oraz `Y` zawierają nieujemne liczby całkowite). Funkcja powinna być możliwie jak najszybsza.

Przykład. Dla wejścia:

```
X = [3,6,10,14]
Y = [1,4,5,10,11,13,17]
```



wynikiem jest 3:

1. Biurowiec z pozycji $X[0] = 3$ dostaje parking na pozycji $Y[1] = 4$ (odlełgość 1),
2. Biurowiec z pozycji $X[1] = 6$ dostaje parking na pozycji $Y[2] = 5$ (odlełgość 1),
3. Biurowiec z pozycji $X[2] = 10$ dostaje parking na pozycji $Y[3] = 10$ (odlełgość 0),
4. Biurowiec z pozycji $X[3] = 14$ dostaje parking na pozycji $Y[5] = 13$ (odlełgość 1).

Podpowiedź. W realizacji algorytmu może pomóc obliczanie funkcji $f(i, j)$, zdefiniowanej jako:

> $f(i, j)$ = minimalna suma odległości biurowców z pozycji $X[0], \ldots, X[i]$ do przydzielonych im działek, przy założeniu że biurowiec z pozycji $X[i]$ ma przydzieloną działkę z pozycji $Y[j]$.

Algorytmy i Struktury Danych
Zadanie offline 9 (17.VI.2024)

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, kolejka `collections.deque`, kolejka priorytetowa (`queue.PriorityQueue, 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`

Mapa Gór Algorytmicznych to macierz o wymiarach $m \times n$, której każde pole to liczba naturalna stanowiąca wysokość danego obszaru (co ciekawe, każdy obszar ma inną wysokość). Student chce się wybrać na wycieczkę po tym paśmie górskim, ale stawia kilka warunków: Po pierwsze, będąc na danym polu może przejść wyłącznie na inne pole, które dzieli z nim jedną współrzędną (czyli może się poruszać o jedno pole na prawo, lewo, w górą, lub w dół). Po drugie, może przejść wyłącznie na pole o wyższej wysokości (wycieczka ma być wyzwaniem). Po trzecie, jego trasa ma być jak najdłuższa, czyli ma przejść jak najwięcej pól, wliczając w to pole startowe (w końcu im więcej czasu spędzie się w Górach Algorytmiki, tym lepiej). Student może wyruszyć z dowolnego wybranego przez siebie pola (po prostu jakoś się tam znajdzie w sobotę rano; nikt nie wie jak to się dzieje).

Zadanie polega na implementacji funkcji:

```
trip( M )
```

która na wejściu otrzymuje mapę Gór Algorytmicznych `M` i zwraca największą liczbę pól, które może odwiedzić student.

Przykład. Dla wejścia:

```
M = [ [7,6,5,12],
      [8,3,4,11],
      [9,1,2,10] ]
```

wynikiem jest 8 (jedną z tras o tej długości jest $1 \to 3 \to 4 \to 5 \to 6 \to 7 \to 8 \to 9$).

Podpowiedź. To zadanie łączy w sobie materiał z wszystkich części przedmiotu ASD (skojarzenie z acykliczynmi grafami skierowanymi nie jest niezbędne, ale pomaga w zrozumieniu poprawności rozwiązania).