Algorytmy i Struktury Danych
Zadanie offline 1 (6.III.2023)

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`

Cesarzowa Bajtocji zgubiła w napisie s swój ulubiony palindrom. Cesarzowa nikomu nie mówiła jaki jest jej ulubiony palindrom i wiadomo jedynie, że jest bardzo długi oraz składa się z nieparzystej liczby liter alfabetu łacińskiego. Postanowiono odnaleźć zaginiony palindrom cesarzowej. W tym celu należy zaimplementować funkcję:

```
def ceasar( s )
```

która na wejściu otrzymuje słowo s (składające się wyłącznie z małych liter alfabetu łacińskiego) i zwraca długość najdłuższego spójnego podsłowa, które jest palindromem i którego długość jest nieparzysta. Użyty algorytm powinien być możliwie jak najszybszy. Proszę uzasadnić jego poprawność i oszacować złożoność obliczeniową.

Przykład. Dla słowa:

```
akontnoknonabcddcba
```

wynikiem jest 7 (`kontnok`; proszę zwrócić uwagę, że `abcddcba` jest dłuższym palindromem, ale jest długości parzystej więc na pewno nie jest zagubionym palindromem cesarzowej).

Algorytmy i Struktury Danych
Zadanie offline 2 (13.III.2023)

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`

System chłodzenia serwerów na pewnej uczelni wymaga stałych dostaw śniegu. Grupa zmotywowanych profesorów odnalazła w wysokich górach wąwóz, z którego można przywieźć śnieg. Wąwóz jest podzielony na n obszarów i ma wjazdy z zachodu i wschodu. Na każdym obszarze wąwozu znajduje się pewna ilość śniegu, opisana w tablicy S. W szczególności $S[0]$ to liczba metrów sześciennych śniegu bezpośrednio przy zachodnim wjeździe, $S[1]$ to liczba metrów sześciennych śniegu na kolejnym obszarze, a $S[n-1]$ to liczba metrów sześciennych śniegu przy wjeździe wschodnim (wiadomo, że zawartość tablicy S to liczby naturalne). Profesorowie dysponują maszyną, która danego dnia może zebrać śnieg ze wskazanego obszaru, wjeżdżając odpowiednio z zachodu lub wschodu. Niestety, są trzy komplikacje

1. Po drodze do danego obszaru maszyna topi cały śnieg na tych obszarach, po których przejeżdża (o ile nie został wcześniej zebrany). Na przykład jadąc z zachodu do obszaru 2 zeruje wartości $S[0]$ oraz $S[1]$ (bo po nich przejeżdża) oraz $S[2]$ (bo ten śnieg zbiera).

2. Każdego dnia maszyna może zebrać śnieg tylko z jednego, dowolnie wybranego obszaru, wjeżdzając albo z zachodu albo ze wschodu.

3. Ze względu na wysoką temperaturę, po każdym dniu na każdym obszarze topi się dokładnie jeden metr sześcienny śniegu.

Zadanie polega na zaimplementowaniu funkcji:

```
def snow( S )
```

która zwraca ile metrów sześciennych maksmalnie można zebrać z wąwozu (zebrany śnieg jest zabezpieczany i już się nie topi).

Rozważmy następujące dane:

```
S = [1,7,3,4,1]
```

wywołanie `snow(S)` powinno zwrócić liczbę 11. Możliwy plan zbierania śniegu to: zebranie $7m^3$ pierwszego dnia z obszaru 1 wjeżdżając z zachodu, zebranie $3m^3$ drugiego dnia z obszaru 3 wjeżdżając ze wschodu ($1m^3$ się stopił po pierwszym dniu), oraz zebranie $1m^3$ trzeciego dnia z obszaru 2 wjeżdżając z dowolnego kierunku (po dwóch dniach ilość śniegu na tym obszarze zmniejszy się z $3m^3$ do $1m^3$).

Zadanie można rozwiązać w czasie $O(n \log n)$, gdzie n to rozmiar wąwozu.

Algorytmy i Struktury Danych
Zadanie offline 3 (20.III.2023)

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`

Kilka uwag o Pythonie

```
x = "abcdefg"
y = x[::-1]     # y staje sie nowym napisem "gfedcba"
z = y           # z i y wskazują na ten sam napis (koszt tego przypisania to O(1)
                # niezaleznie od dlugosci napisu y)
ord("a")        # wynikiem jest kod litery 'a' (97)
chr(97)         # wynikiem jest litera o kodzie 97 ("a")
```

Zadanie offline 3.

Szablon rozwiązania: `zad3.py`

Mówimy, że dwa napisy są sobie równoważne, jeśli albo są identyczne, albo byłyby identyczne, gdyby jeden z nich zapisać od tyłu. Na przykład napisy "kot" oraz "tok" są sobie równoważne, podobnie jak napisy "pies" i "pies". Dana jest tablica T zawierająca n napisów o łącznej długości N (każdy napis zawiera co najmniej jeden znak, więc $N \geq n$; w praktyce można przyjąć, że $N \gg n$; każdy napis składa się wyłącznie z małych liter alfabetu łacińskiego). Siłą napisu $T[i]$ jest liczba indeksów j takich, że napisy $T[i]$ oraz $T[j]$ są sobie równoważne. Napis $T[i]$ jest najsilniejszy, jeśli żaden inny napis nie ma większej siły.

Proszę zaimplementować funkcję `strong_string(T)`, która zwraca siłę najsilniejszego napisu z tablicy T. Na przykład dla wejścia:

```
#     0       1       2      3        4      5       6
T = ["pies", "mysz", "kot", "kogut", "tok", "seip", "kot"]
```

wywołanie `strong_string(T)` powinno zwrócić 3. Algorytm powinien być możliwie jak najszybszy.

Zadanie można rozwiązać w czasie $O(N + n \log n)$, gdzie N to łączna długość napisów w tablicy wejściowej a n to liczba wyrazów.

Algorytmy i Struktury Danych
Zadanie offline 4 (17.IV.2023)

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 zad4.py`

Zadanie offline 4.

Szablon rozwiązania: `zad4.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 5 (24.IV.2023)

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 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.

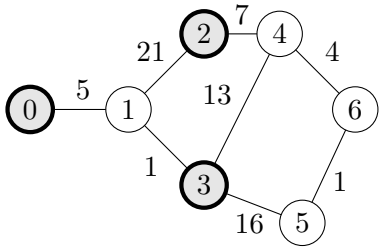
Zadanie polega na zaimplementowaniu funkcji:

```
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 (16.V.2023)

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 zad6.py`

Zadanie offline 6.

Szablon rozwiązania: `zad6.py`

Firma Binary Works zajmuje się produkcją kubełków do sortowania liczb. W firmie jest n pracowników oraz n maszyn pozwalających na tworzenie kubełków. Niestety praktycznie każde urządzenie pochodzi od innego wytwórcy i wymaga innych umiejętności. Stąd nie wszyscy pracownicy potrafią obsługiwać każde z nich. Zadanie polega na zaimplementowaniu funkcji: Zadanie polega na zaimplementowaniu funkcji:

```
def binworker( M )
```

gdzie M to tablica tablic o następującej interpretacji: Dla każdego $i \in \{0, \ldots, n-1\}$ $M[i]$ to lista numerów maszyn (ze zbioru $\{0, \ldots, n\}$ na których pracownik numer i potrafi pracować. Funkcja powinna zwrócić maksymalną liczbę pracowników, którzy mogą jednocześnie pracować (na danej maszynie może na raz pracować jednocześnie tylko jeden pracownik).

Przykład. Rozważmy następujące dane:

```
M = [ [ 0, 1, 3],   # 0
      [ 2, 4],      # 1
      [ 0, 2],      # 2
      [ 3 ],        # 3
      [ 3, 2] ]     # 4
```

Wynikiem wywołania `binworker(M)` powinno być 5. W szczególności:

- pracownik 0 może pracować na maszynie 1,
- pracownik 1 może pracować na maszynie 4,
- pracownik 2 może pracować na maszynie 0,
- pracownik 3 może pracować na maszynie 3,
- pracownik 4 może pracować na maszynie 2.

Algorytmy i Struktury Danych
Zadanie offline 7 (5.VI.2023)

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`

Magiczny Wojownik otrzymał od Dobrego Maga kolejną szansę. Musi przejść przez kwadratowy labirynt złożony z $N \times N$ komnat. 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. Wojownikowi 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. Zadanie postawione przez Maga polega na odwiedzeniu jak największej liczby komnat. Zadanie polega na zaimplementowaniu funkcji:

```
def maze ( L )
```

która otrzymuje na wejściu tablicę L opisującą labirynt i zwraca największą liczbę komnat, które może odwiedzić Wojownik 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 (12.VI.2023)

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`

W roku 2050 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.

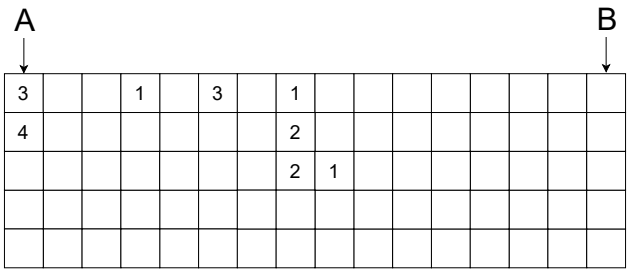
Dane wejściowe reprezentowane są jako dwuwymiarowa tablica liczb naturalnych `T`, w której wartość `T[u][v]` to objętość ropy na polu o współrzędnych (u, v) (objętość 0 oznacza brak ropy). Współrzędne u należą do zbioru $\{0, 1, \ldots, n-1\}$ a współrzędne v to zbioru $\{0, 1, \ldots, m-1\}$. Miasto A znajduje się na polu $(0, 0)$, zaś miasto B na polu $(0, m-1)$. Maksymilian porusza się jedynie po polach $(0, 0)$, $(0, 1), \ldots (0, m-1)$. Bok każdego pola ma długość 1 kilometra. Plamą ropy jest dowolny spójny obszar pól zawierających ropę. Dwa pola należą do spójnego obszaru jeśli mają wspólny bok lub są połączone sekwencją pól (zawierających ropę) o wspólnych bokach. Zakładamy, że początkowo cysterna jest pusta, ale pole $(0, 0)$ 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 zadania i zwraca minimalną liczbę zatrzymań cysterny potrzebną do przejechania trasy (cysterna porusza się tylko po polach $(0, v)$). Postój na polu $(0, 0)$ również jest częścią rozwiązania.

Przykład. Dla mapy (w pustych polach są wartości 0, a więc brak ropy):



wynikiem jest 3 (tankowania na polach 0, 5 i 7).

Algorytmy i Struktury Danych
Zadanie offline 9 (19. VI 2023)

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. 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` lub `heapq`),
2. korzystanie z zaawansowanych struktur danych (np. słowników czy zbiorów),
3. korzystanie ze struktur danych dostarczonych razem z zadaniem (jeśli takie są),
4. 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: `python zad9.py`

Szablon rozwiązania:	`zad9.py`
Złożoność akceptowalna:	$O(n^2)$, gdzie n to łączna liczba parkingów.
Złożoność wzorcowa:	$O(n \log n)$, gdzie n to łączna liczba parkingów.

Kierowca ciężarówki przewozi towary z miasta A do miasta B. W pewnych miejscach trasy przejazdu znajdują się parkingi. Przejeżdżając obok parkingu kierowca może (ale nie musi) się na nim zatrzymać i odpocząć. Przepisy transportowe narzucają jednak pewne ograniczenia związane z bezpieczeństwem:

1. Maksymalna liczba kilometrów, którą można przejechać bez zatrzymania wynosi T. Od zasady tej istnieje jeden wyjątek, opisany w punkcie 2.
2. W trakcie całego przejazdu z A do B kierowca może jeden raz przekroczyć limit T kilometrów jazdy bez zatrzymania. Może wówczas przejechać nie więcej niż $2T$ kilometrów bez zatrzymania.

Niestety, parkingi na trasie są płatne. Co więcej, opłaty za postój różnią się pomiędzy parkingami. Kierowca musi więc wybrać miejsca postoju w taki sposób, by przejechać trasę zgodnie z obowiązującymi przepisami i równocześnie zapłacić możliwie jak najmniej za postoje.

Zaproponuj i zaimplementuj algorytm, który wylicza minimalny koszt przejechania z miasta A do miasta B zgodnie z opisanymi przepisami transportu towarów. Koszt przejazdu z A do B definiujemy jako sumę opłat za parkowanie w miejscach, w których kierowca się zatrzymał (nie liczymy ceny paliwa; nie bierzemy pod uwagę czasu postoju na parkingu). W miastach A i B opłata nie jest pobierana. Uzasadnij poprawność zaproponowanego algorytmu i oszacuj jego złożoność obliczeniową.

Algorytm należy zaimplementować jako funkcję:

```
def min_cost( O, C, T, L )
```

Argumentami funkcji są:

- Tablica `O` zawierająca pozycje parkingów na trasie z A do B. `O[i]` to liczba kilometrów (wzdłuż trasy przejazdu) od A do i-go parkingu.
- Tablica `C` zawierająca ceny za postój na poszczególnych parkingach. `C[i]` to opłata za zatrzymanie na i-ym parkingu.
- Maksymalna liczba kilometrów `T`, którą można przejechać bez zatrzymywania (z zastrzeżeniem wyjątku opisanego powyżej).
- Długość `L` trasy (liczba kilometrów od A do B wzdłuż trasy przejazdu).

Wszystkie wartości przekazane w tablicach `O` i `C` oraz argumenty `T` i `L` to dodatnie liczby naturalne. Tablice nie muszą być posortowane. Funkcja `min_cost` powinna zwrócić jedną liczbę naturalną: minimalny koszt przejazdu z A do B. Można założyć, że parkingi są tak rozmieszczone, że da się przejechać z A do B zgodnie z obowiązującymi zasadami. Funkcja powinna być możliwie jak najszybsza.

Przykład. Dla danych:

```
O = [17, 20, 11, 5, 12]
C = [9,  7,  7,  7, 3]
T = 7
L = 25
```

wywołanie `min_cost(O, C, T, L)` powinno zwrócić wartość 10.

Podpowiedź. Zastanów się, jaki jest koszt dojechania do parkingu i mogąc lub nie mogąc wykorzystać wyjątek, o którym mowa w punkcie 2.