Algorytmy i Struktury Danych
Egzamin 1 (13.VII.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. modyfikowanie testów dostarczonych wraz z szablonem,
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, słownik, 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 egz1A.py`

Szablon rozwiązania:	`egz1A.py`
Złożoność akceptowalna (+2.0pkt):	$O(V^3 \log V)$
Złożoność wzorcowa (+2.0pkt):	$O(V^2 \log V)$
Gdzie V to liczba wierzchołków grafu.	

Złycerz (czyli zły rycerz) wędruje po średniowiecznym grafie $G = (V, E)$, gdzie waga każdej krawędzi to liczba sztabek złota, którą trzeba zapłacić za przejazd nią (myta, jedzenie, itp.). W każdym wierzchołku znajduje się zamek, który zawiera w skarbcu pewną daną liczbę sztabek złota. Złycerz może napaść na jeden zamek i zabrać całe jego złoto, ale od tego momentu zaczyna być ścigany i każdy przejazd po krawędzi jest dwa razy droższy, oraz dodatkowo na każdej drodze musi zapłacić r sztabek złota jako łapówkę (zatem od tej pory koszt przejazdu danej krawędzi jest równy dwukrotności wagi tej krawędzi plus wartość r). Co więcej, Złycerz nie może napaść więcej niż jednego zamku, bo jest trochę leniwy (oprócz tego, że zły). Proszę wskazać trasę Złycerza z zamku s do t o najmniejszym koszcie (lub największym zysku, jeśli to możliwe).

Uwaga. Złycerz może przejechać po danej krawędzi więcej niż raz (np. raz jadąc do zamku, który chce napaść, a potem z niego wracając).

Zadanie polega na implementacji funkcji:

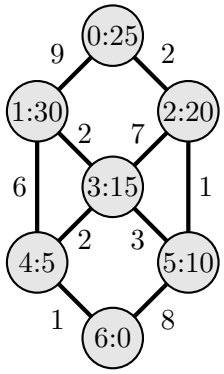
```
gold( G,V,s,t,r )
```

która na wejściu otrzymuje: graf G reprezentowany w postaci listowej, tablicę V zawierającą liczby sztabek złota w kolejnych zamkach, zamek początkowy s, zamek końcowy t oraz wysokość łapówki r. Funkcja powinna zwrócić najmniejszy koszt drogi uwzględniający ewentualny napad. Jeżeli zysk z napadu jest większy, od kosztu drogi należy, powstały zysk należy zwrócić jako liczbę ujemną (przykład).

Przykład. Dla wejścia:

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

V = [25,30,20,15,5,10,0]
s = 0, t = 6, r = 7
```



wynikiem jest 6.

Gdyby nie rabować żadnego z zamków, najmniejszy koszt wyniósłby $2 + 1 + 3 + 2 + 1 = 9$, droga $[0, 2, 5, 3, 4, 6]$. Jednak jeżeli obrabujemy zamek 1 (30 sztabek złota), koszt wyniesie $2 + 1 + 3 + 2 - 30 + 19 + 9 = 6$, droga $[0, 2, 5, 3, 1, 4, 6]$. Gdyby łapówka r wynosiła 10, żadna kradzież nie byłaby opłacalna, jednak gdyby łapówka r wynosiła 3, wtedy (po obrabowaniu zamku 1), funkcja powinna zwrócić wartość -3.

Algorytmy i Struktury Danych
Egzamin/Zaliczenie 1 (13.VII.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. modyfikowanie testów dostarczonych wraz z szablonem,
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, słownik, 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 egz1b.py`

Szablon rozwiązania:	`egz1b.py`
Złożoność akceptowalna (3.0pkt):	złożoność wielomianowa względem n i E (np. $O(nE^2)$)
Złożoność wzorcowa (+1.0pkt):	$O(nE)$
Gdzie n to liczba planet a E to pojemność zbiornika paliwa.	

Przedstawiciel Gwiezdnej Floty podróżuje z planety A do planety B. Po drodze będzie musiał wylądować na innych planetach w celu uzupełnienia paliwa. Cena paliwa na każdej planecie może być inna. Dodatkowo na każdej planecie jest teleport, który może przenieść go na jedną z kolejnych planet. Niestety, w tym celu jego statek kosmiczny musi mieć pusty zbiornik paliwa. Zaproponuj i zaimplementuj algorytm, który oblicza minimalny koszt pokonania trasy z planety A do B.

Na trasie przedstawiciela Gwiezdnej Floty znajduje się n planet opisanych w tablicach `D`, `C` i `T`:

- `D[i]` to odległość i-tej planety od planety `A` wyrażona w latach świetlnych wzdłuż trasy przelotu. Planety uporządkowane są w kolejności rosnącej odległości od A. Nie dopuszczamy sytuacji, w której dwie planety mają tą samą odległość od A.
- `C[i]` to cena jednej tony paliwa na planecie numer i,
- `T[i]` to para postaci (j, p), gdzie j to numer planety, na którą można dostać się teleportem z planety i (zawsze zachodzi $j \geq i$, gdzie $j = i$ oznacza, że teleport na tej planecie nie działa) a p to cena skorzystania z teleportu.

Planeta A ma numer 0 a planeta B ma numer $n - 1$. Statek kosmiczny potrzebuje tony paliwa na pokonanie każdego roku świetlnego. Pojemność zbiornika statku kosmicznego wynosi `E` ton. Nie ma obowiązku tankowania paliwa do pełna. Zakładamy, że `E` oraz wszystkie elementy tablic `D`, `C` i `T` to liczby naturalne. Można założyć, że rozwiązanie istnieje, t.j. da się przelecieć z A do B zgodnie z warunkami zadania.

Algorytm należy zaimplementować jako funkcję:

```
planets( D, C, T, E )
```

Proszę uzasadnić poprawność zaproponowanego algorytmu i oszacować jego złożoność obliczeniową.

Przykład. Dla wejścia:

```
#       0      1       2       3
D = [   0,     5,     10,     20]
C = [   2,     1,      3,      9]
T = [(2,3), (3,7), (2,10), (3,10)]
E = 10
```

wynikiem jest 17, co odpowiada zatankowaniu 5 ton paliwa na planecie A (czyli planecie 0, koszt 10), przelotowi na planetę 1 oraz skorzystaniu z teleportu z planety 1 na planetę 3 (czyli planetę B, koszt 7).

Podpowiedź. Proszę rozważyć funkcję $f(i, b)$, która daje minimalny koszt znalezienia się na planecie i mając b ton paliwa. Obliczanie tej funkcji w odpowiednio przemyślany sposób pozwala uzyskać złożoność wzorcową.

Algorytmy i Struktury Danych
Egzamin/Zaliczenie 2 (7.IX.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. modyfikowanie testów dostarczonych wraz z szablonem,
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, słownik, 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 (należy 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 egz2a.py`

Szablon rozwiązania:	`egz2a.py`
Złożoność akceptowalna (2pkt):	$O(n^2)$
Złożoność lepsza (+1pkt):	$O(n \log n)$
Złożoność wzorcowa (+1pkt):	$O(n)$

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:

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

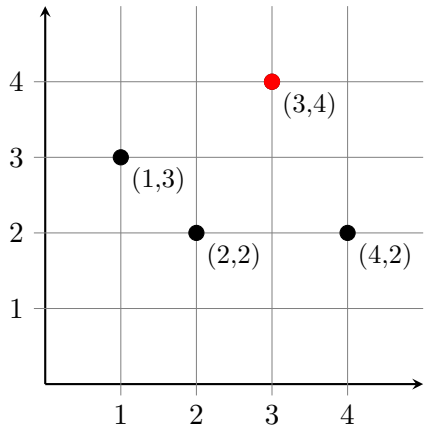
Siłą danego punktu jest to ile punktów dominuje. Zadanie polega na implementacji funkcji:

```
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 $(2, 2)$.

Podpowiedź. W realizacji algorytmu o złożoności $O(n)$ przydatne może być policzenie tablicy T takiej, że $T[i]$ to liczba punktów, których współrzędna y jest większa lub równa i.

Algorytmy i Struktury Danych
Egzamin/Zaliczenie 2 (7.IX.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. modyfikowanie testów dostarczonych wraz z szablonem,
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, słownik, 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 (należy 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 egz2b.py`

Szablon rozwiązania:	`egz2b.py`
Złożoność akceptowalna (2pkt):	dowolna złożoność wielomianowa
Złożoność lepsza (+1pkt):	$O(m^3)$
Złożoność wzorcowa (+1pkt):	$O(mn)$

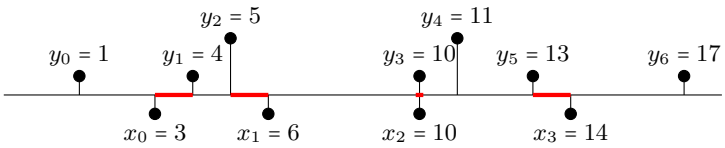
Szalony Inwestor wybudował po południowej stronie drogi n biurowców, na pozycjach $x_0 < \cdots < x_{n-1}$. Parkingi tych biurowców mają dopiero zostać wybudowane i dostępne jest w tym celu m działek ($m \geq n$), dostępnych na północnej stronie drogi, na pozycjach $y_0 < \cdots < y_{m-1}$. Inwestor 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 poycji $X[0] = 3$ dostaje parking na pozycji $Y[1] = 4$ (odlełgość 1),
2. Biurowiec z poycji $X[1] = 6$ dostaje parking na pozycji $Y[2] = 5$ (odlełgość 1),
3. Biurowiec z poycji $X[2] = 10$ dostaje parking na pozycji $Y[3] = 10$ (odlełgość 0),
4. Biurowiec z poycji $X[3] = 14$ 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 $Y[j]$.

Funkcja ta pozwala zarówno uzyskać algorytm o złożoności "lepszej" jak i "wzrocowej", choć to drugie wymaga pewnej optymalizacji obliczeń.

Algorytmy i Struktury Danych
Egzamin/Zaliczenie 3 (14.IX.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. modyfikowanie testów dostarczonych wraz z szablonem,
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, słownik, 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 (należy 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 egz3a.py`

Szablon rozwiązania:	`egz3a.py`
Złożoność akceptowalna (2pkt):	$O(n^3)$
Złożoność lepsza (+1pkt):	$O(n^2 \log n)$
Złożoność wzorcowa (+1pkt):	$O(n^2)$
Gdzie n to liczba zamków.	

Dobrycerz (czyli rycerz, który zawsze uprzejmie mówi "dzień dobry") chce się przedostać z zamku s do zamku t. Mapa zamków dana jest w postaci grafu nieskierowanego G, gdzie każda krawędź ma wagę oznaczającą ile godzin potrzeba, żeby ją przebyć. Wagi to liczby naturalne ze zbioru $\{1, 2, \ldots, 8\}$. Po najdalej 16 godzinach podróży Dobrycerz musi nocować w zamku. Warunki uprzejmości wymagają, żeby spędził w takim zamku 8 godzin (przejazd przez zamki, w których nie nocuje nie kosztuje dodatkowego czasu; szybko mówi "dzień dobry" strażnikom i jedzie dalej). Mapa z której korzysta Dobrycerz ma to do siebie, że liczba dróg jest proporcjonalna do liczby zamków. Czyli jeśli zamków jest n, to wiadomo, że dróg jest $O(n)$.

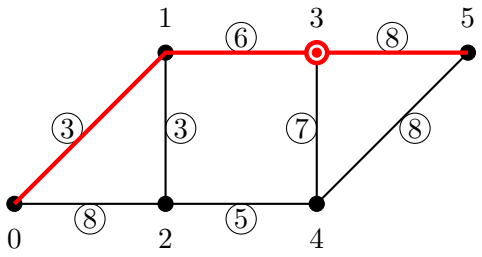
Zadanie polega na implementacji funkcji:

```
goodknight( G, s, t )
```

która na wejściu otrzymuje graf opisujący mapę zamków, reprezentowany w postaci macierzy sąsiedztwa (czyli $G[i][j]$ to liczba godzin, konieczna do przejechania bezpośrednio z zamku i do zamku j; w przypadku braku drogi $G[i][j] = -1$), zamek startowy s oraz zamek docelowy t, i zwraca minimalny czas (wyrażony w godzinach) potrzebny na przejazd z s do t (Dobrycerz nigdy nie musi nocować ani w zamku s ani w zamku t). Można założyć, że zawsze istnieje trasa z zamku s do t.

Przykład. Dla wejścia:

```
#         0  1  2  3  4  5
G = [ [ -1, 3, 8,-1,-1,-1 ],    # 0
      [  3,-1, 3, 6,-1,-1 ],    # 1
      [  8, 3,-1,-1, 5,-1 ],    # 2
      [ -1, 6,-1,-1, 7, 8 ],    # 3
      [ -1,-1, 5, 7,-1, 8 ],    # 4
      [ -1,-1,-1, 8, 8,-1 ]]    # 5
s = 0
t = 5
```



wynikiem jest 25. Dobrycerz pokonuje następującą trasę:

1. Jedzie z zamku 0 do zamku 1 (3 godziny).
2. Jedzie z zamku 1 do zamku 3 (6 godzin).
3. Nocuje w zamku 3 (8 godzin).
4. Jedzie z zamku 3 do zamku 5 (8 godzin)

Algorytmy i Struktury Danych
Egzamin/Zaliczenie 3 (14.IX.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. modyfikowanie testów dostarczonych wraz z szablonem,
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, słownik, 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 (należy 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 egz3b.py`

Szablon rozwiązania:	`egz3b.py`
Złożoność akceptowalna (2pkt):	$O(n^2)$
Złożoność wzorcowa (+2pkt):	$O(n \log n)$

Dany jest zbiór $P = \{[a_0, b_0], \ldots, [a_{n-1}, b_{n-1}]\}$ zawierający n parami różnych przedziałów domkniętych. Mówimy, że para przedziałów jest fajna jeśli albo jeden z nich zawiera się w drugim, albo są rozłączne. Jeśli para przedziałów nie jest fajna, to mówimy na nią niefajna.

Przykład. Rozważmy trzy przypadki:

1. Para przedziałów $[1, 3]$ i $[6, 7]$ jest fajna, bo przedziały są rozłączne.
2. Para przedziałów $[2, 6]$ i $[4, 6]$ jest fajna, bo drugi zawiera się w pierwszym.
3. Para przedziałów $[1, 8]$ i $[5, 10]$ jest niefajna, bo ani nie są rozłączne, ani jeden nie zawiera się w drugim.

Zadanie polega na implementacji funkcji:

```
uncool( P )
```

która na wejściu otrzymuje zbiór przedziałów P (w postaci listy, gdzie każdy element jest postaci $[a_i, b_i]$ i opisuje przedział domknięty) i zwraca parę indeksów (i, j) takich, że para przedziałów $P[i]$, $P[j]$ jest niefajna (można założyć, że taka para przedziałów zawsze istnieje).

Przykład. Dla wejścia:

```
#    0      1      2      3      4      5
P =[ [1,3], [6,7], [2,6], [4,6], [1,8], [5,10] ]
```

prawidłowym wynikami są, między innymi, $(2, 1)$ oraz $(4, 5)$.