Algorytmy i Struktury Danych
Egzamin 1: Zadanie A (7.VII.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. 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 zaawansowanych struktur danych (np. słowników czy zbiorów),
2. korzystanie z następujących elementarnych struktur danych: krotka, lista, kolejka `collections.deque`, kolejka priorytetowa (`queue.PriorityQueue` lub `heapq`),
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 egz1a.py`

Szablon rozwiązania:	`egz1a.py`
Złożoność akceptowalna (1.5pkt):	$O(n^4)$, gdzie n to rozmiar wąwozu.
Złożoność wzorcowa (+2.5pkt):	$O(n \log n)$, gdzie n to rozmiar wąwozu.

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, wjeżdżając 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 $1m^3$).

Podpowiedź. Jak zmieniłby się wynik, gdyby wąwóz miał wyłącznie wjazd od zachodu? Co by się stało, gdybyśmy wiedzieli, że śnieg mamy zbierać dokładnie d dni?

Szablon rozwiązania:	`egz1b.py`
Złożoność akceptowalna (1.5pkt):	$O(n^2)$, gdzie n to rozmiar drzewa.
Złożoność wzorcowa (+2.5pkt):	$O(n)$, gdzie n to rozmiar drzewa.

Dane jest drzewo binarne opisane przez następujące klasy:

```
class Node:
  def __init__( self ):
    self.left = None    # lewe poddrzewo
    self.right = None   # prawe poddrzewo
    self.x = None       # pole do wykorzystania przez studentów
```

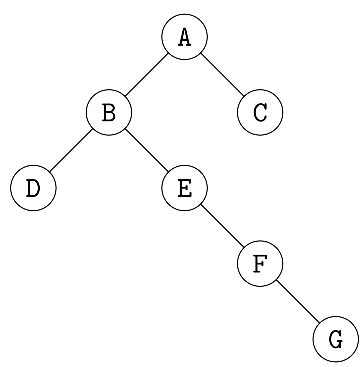
Mówimy, że takie drzewo jest *ładne* jeśli wszystkie jego liście znajdują się na jednym poziomie. Szerokością ładnego drzewa jest jego liczba liści a wysokością poziom, na którym te liście się znajdują (korzeń jest na poziomie 0, jego dzieci na poziomie 1, jego wnuki na poziomie 2 itd.). Zadanie polega na zaimplementowaniu funkcji:

```
def widentall( T )
```

która dla danego drzewa T zwraca minimalną liczbę krawędzi, które trzeba usunąć, żeby powstało ładne drzewo, którego szerokość jest jak największa i którego wysokość jest największa wśród drzew o maksymalnej szerokości. Usunięcie krawędzi odcina całe poddrzewo poniżej tej krawędzi.

Rozważmy następujące dane wejściowe:

```
A = Node()
B = Node()
C = Node()
A.left = B
A.right = C
D = Node()
E = Node()
B.left = D
B.right = E
F = Node()
E.right = F
G = Node()
F.right = G
```



Wywołanie `widentall(A)` powinno zwrócić wynik 2 (ucinamy krawędzie między `A` i `C` oraz między `E` i `F`. Ucięcie krawędzi między `B` i `D` oraz między `B` i `E` doprowadziłoby do ładnego drzewa o tej samej szerokości, ale mniejszej wysokości.

Szablon rozwiązania:	`egz2a.py`
Złożoność akceptowalna (1.5pkt):	$O(n^2)$, gdzie n to liczba transportów.
Złożoność wzorcowa (+2.5pkt):	$O(n \log n)$, gdzie n to liczba transportów.

Do elektrowni ma przyjechać seria długo oczekiwanych tranportów węgla. Każdy transport zawiera pewną liczbę ton węgla, które muszą być składowane w jednym z magazynów o numerach 0, 1, 2, ... (magazynów jest bardzo dużo i na pewno wystarczą na cały węgiel). Każdy magazyn ma pojemność T ton i węgiel z każdego transportu musi być przechowywany razem, w jednym magazynie (ale w danym magazynie może być węgiel z kilku różnych transportów). Dyrekcja elektrowni przyjęła zasadę, że gdy przyjeżdża kolejny transport, to węgiel umieszczany jest w magazynie o najniższym numerze, w którym się mieści (na szczęście żaden transport nie ma więcej niż T ton). Proszę napisać funkcję:

```
def coal( A, T )
```

która przyjmuje na wejściu tablicę $A = [a_0, \ldots, a_{n-1}]$ zawierającą ilości węgla w kolejnych transportach (wyrażoną w tonach, jako liczby naturalne) oraz liczbę naturalną T oznaczającą pojemność każdego z magazynów. Funkcja powinna zwrócić numer magazynu, w którym umieszczono ostatni transport. Funkcja powinna być możliwie jak najszybsza.

Przykład. Rozważmy następujące dane:

```
A = [1, 6, 2, 10, 8, 7, 1]
T = 10
```

Wywołanie `coal(A, T)` powinno zwrócić 0 (pierwsze trzy transporty zostaną umieszczone w magazynie nr 0, kolejny w magazynie nr 1, transport 8 ton zostanie umieszczony w magazynie nr 2, transport 7 ton w magazynie nr 3 i ostatni transport zmieści się jeszcze w magazynie nr 0).

Szablon rozwiązania:	egz2b.py
Złożoność akceptowalna (1.5pkt):	$O(n^2)$, gdzie n to komnat.
Złożoność wzorcowa (+2.5pkt):	$O(n)$, gdzie n to liczba komnat.

Magiczny Wojownik obudził się w komnacie 0 pewnej tajemniczej jaskini, mając w głowie jedynie instrukcje, jakie otrzymał od Złego Maga. Wie, że komnaty są ponumerowane od 0 do $n-1$ i w każdej komnacie znajduje się troje drzwi, z których każde pozwala przejść do komnaty o wyższym numerze (cofnięcie się do komnaty o niższym numerze grozi śmiercią Wojownika; co więcej, niektóre drzwi są zablokowane) oraz skrzynia z pewną liczbą sztabek złota. Wstępnie wszystkie drzwi są zamknięte, ale jeśli w skrzyni zostanie umieszczona odpowiednia liczba sztabek złota, to drzwi się otwierają i można nimi przejść. Z każdej skrzyni można zabrać najwyżej 10 sztabek złota, ale można też w niej zostawić dowolnie wiele sztabek. Na początku Wojownik nie ma ani jednej sztabki a jego celem (na zlecenie Złego Maga) jest dojść do komnaty $n-1$ mając jak najwięcej sztabek.

Zadanie polega na zaimplementowaniu funkcji:

```
def magic( C )
```

która otrzymuje na wejściu tablicę C opisującą jaskinię ($n = |C|$) i zwraca największą liczbę sztabek złota, z którymi Wojownik może dojść do komnaty $n-1$, lub -1 jeśli dotarcie do tej komnaty jest niemożliwe. Opis jaskini jest postaci $C = [R_0, \ldots, R_{n-1}]$, gdzie każde R_i to opis komnaty postaci:

$$[G, [K_0, W_0], [K_1, W_1], [K_2, W_2]]$$

gdzie G to liczba sztabek złota w skrzyni a każda para $[K_i, W_i]$ składa się z liczby K_i sztabek słota potrzebnych do otwarcia drzwi numer i prowadzących do komnaty W_i (gdzie $W_i > i$ lub $W_i = -1$ jeśli za tymi drzwiami jest lita skała i nie da się nimi przejść nawet jeśli się je otworzy). Funkcja powinna być możliwie jak najszybsza.

Przykład. Rozważmy następującą jaskinię:

```
C = [ [8,  [ 6, 3], [ 4, 2], [7, 1]],   # 0
      [22, [12, 2], [21, 3], [0,-1]],   # 1
      [9,  [11, 3], [ 0,-1], [7,-1]],   # 2
      [15, [ 0,-1], [ 1,-1], [0,-1]] ]  # 3
```

Optymalna trasa wojownika to:

1. Wziąć 1 sztabkę złota w komnacie 0 i przejść do komnaty 1.
2. Wziąć 10 sztabek złota w komnacie 1 i przejść do komnaty 2.
3. Zostawić 2 sztabki złota w komnacie 2 i przejść do komnaty 3.

Dzięki temu na koniec wędrówki Wojownik ma 9 sztabek złota.

Szablon rozwiązania:	`egz3a.py`
Złożoność akceptowalna (1.5pkt):	$O(n^2)$, gdzie n to liczba dni.
Złożoność wzorcowa (+2.5pkt):	$O(n \log n)$, gdzie n to liczba dni.

Autostrada Bajtocji to linia prosta o długości $T \in \mathbb{N}_+$, kilometrów. W zimie, w czasie kolejnych n dni na pewnych odcinkach autostrady pada śnieg. W i-tym dniu śnieg pada na odcinku $[a_i, b_i]$ (domkniętym obustronnie), gdzie $a_i, b_i \in \{0, 1, 2, ..., T-1\}$. W wyniku opadu grubość warstwy śnieniu [sic] na tym odcinku rośnie o 1mm. Proszę zaproponować i zaimplementować algorytm, który zwraca największą liczbę $H \in \mathbb{N}$ taką, że po n dniach na autostradzie jest punkt (lub punkty) na którym leży H milimetrów śniegu. Zakładamy, że w rozważanym okresie n dni śnieg nie topnieje. Algorytm powinien być możliwie jak najszybszy. Proszę uzasadnić jego poprawność i oszacować złożoność obliczeniową.

Algorytm należy zaimplementować jako funkcję:

```
def snow ( T, I )
```

która otrzymuje na wejściu długość autostrady T (liczba naturalna) oraz listę odcinków $I = [(a_0, b_0), (a_1, b_1), ..., (a_{n-1}, b_{n-1})]$ i zwraca maksymalną grubość śniegu H. Odcinki reprezentowane są jako krotki. Końce odcinków są liczbami naturalnymi.

Przykład. Dla danych wejściowych:

```
T = 100
I = [(3, 10), (0, 5), (20, 30), (25, 35), (26, 26)]
```

Funkcja powinna zwrócić wartość 3.

Szablon rozwiązania:	`egz3b.py`
Złożoność akceptowalna (1.5pkt):	$O(n^3)$, gdzie n to rozmiar labiryntu.
Złożoność wzorcowa (+2.5pkt):	$O(n^2)$, gdzie n to rozmiar labiryntu.

Magiczny Wojownik który poprzednio nie dotarł do ostatniej komnaty z maksymalną liczba sztabek złota otrzymał od Dobrego Maga jeszcze jedną 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 $L = [W_0, W_1, W_2, ..., W_{n-1}]$, gdzie każde W_i to napis o długości n znaków. Znak kropki '.' oznacza dostępną komnatę a znak '#' oznacza komnatę niedostępną.

Przykład. Dla danych wejściowych:

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

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