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, 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 $1m^3$).

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?

Algorytmy i Struktury Danych
Egzamin 1: Zadanie B (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 egz1b.py`

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
```

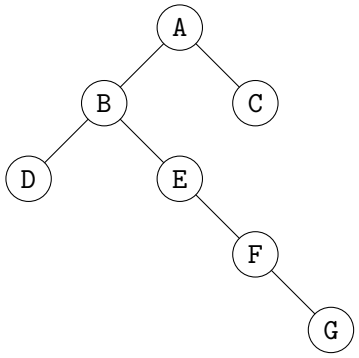
Mówimy, że takie drzewo jest *ładne* jeśli wszystkie jego liście znajdują się na jednym poziomie. Szerokością ładnego drzewa jest jego liczba liści a wysokością poziom, na którym te liście się znajdują (korzeń jest na poziomie 0, jego dzieci na poziomie 1, jego wnuki na poziomie 2 itd.). Zadanie polega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