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
```

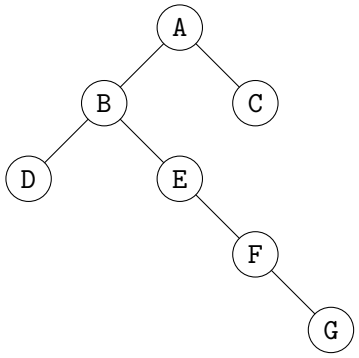
Mówimy, że takie drzewo jest *ładne* jeśli wszystkie jego liście znajdują się na jednym poziomie. Szerokością ładnego drzewa jest jego liczba liści a wysokością poziom, na którym te liście się znajdują (korzeń jest na poziomie 0, jego dzieci na poziomie 1, jego wnuki na poziomie 2 itd.). Zadanie polega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