[2pkt.] Zadanie 1.

Szablon rozwiązania: `zad1.py`
Drzewo przedziałowe: `inttree.py`

Dana jest lista `I` przedziałów domkniętych $[a_1, b_1]$, $[a_2, b_2], \ldots, [a_n, b_n]$. Napisz funkcję `intervals( I )`, która oblicza dla każdego $i \in \{1, 2, \ldots, n\}$ długość najdłuższego ciągłego przedziału, który można osiągnąć sumując wybrane przedziały spośród pierwszych i przedziałów z listy. Funkcja powinna być możliwie jak najszybsza.

Przedziały reprezentowane są w postaci listy par. Funkcja powinna zwrócić listę liczb, w której i-ty element to długość poszukiwanego najdłuższego przedziału zbudowanego z pierwszych i elementów wejścia. Na przykład, dla listy odcinków:

```
[(1,3),(5,6),(4,7),(6,9)]
```

rozwiązaniem jest lista `[2, 2, 3, 5]` zawierająca długości odcinków: `[1, 3]`, `[1, 3]`, `[4, 7]` oraz `[4, 9]`.

Drzewo przedziałowe

W pliku `inttree.py` została Państwu dostarczona elementarna implementacja drzewa przedziałowego, tak jak było ono opisane na wykładzie. Dostępne są następujące funkcje:

1. `tree(A)` – stwórz nowe drzewo przedziałowe; przechowywane przedziały muszą być postaci $[a, b]$, gdzie liczby a i b występują w tablicy `A`; tablica `A` musi być posortowana rosnąco i nie może zawierać powtórzeń. Funkcja zwraca korzeń drzewa `T`. Złożoność: $O(|A|)$.
2. `tree_insert(T, (a,b))` – wstaw do drzewa `T` (reprezentowanego przez korzeń) przedział $[a, b]$. Złożoność: $O(\log |A|)$.
3. `tree_remove(T, (a,b))` – usuń z drzewa `T` (reprezentowanego przez korzeń) przedział $[a, b]$ (jeśli przedziału nie było w drzewie, to nic nie robi). Złożoność: $O(\log |A|)$.
4. `tree_intersect( T, x )` – zwraca listę przedziałów z drzewa `T` (reprezentowanego przez korzeń), które zawierają punkt x (niektóre przedziały mogą występować na liście dwukrotnie). Złożoność: $O(k + \log |A|)$, gdzie k to liczba zwróconych przedziałów
5. `tree_print( T )` – funkcja pomocnicza wypisująca zawartość drzewa (proszę zajrzeć do kodu, żeby zobaczyć co wypisuje).

Nie ma obowiązku korzystać z tego drzewa—można zaimplementować własne, lub użyć innej struktury danych. Jeśli ktoś zaimplementuje **klasyczne drzewo BST** to może je analizować tak, jakby operacje na nim miały **złożoność $O(\log n)$**}.

Przykład wykorzystania drzewa przedziałowego

```
from inttree import *
T = tree([1, 2, 3, 4, 5])
tree_insert(T,(1, 4))
tree_insert(T,(2, 5))
tree_print(T)
tree_remove(T,(1, 4))
tree_print(T)
tree_insert(T,(1, 3))
print(tree_intersect(T, 3))
```