Ćwiczenia 13: Drzewa BST i tablice z haszowaniem

[dla osób z maksymalnie 2-ma plusami] Zadanie 1. (Implementacja drzew BST) Proszę zaimplementować następujące operacje na drzewie BST:

1. znalezienie minimum/maksimum
2. następnik/poprzednik

[dla osób z maksymalnie 2-ma plusami] Zadanie 2. (Implementacja tablicy haszującej) Proszę zaimplementować następujące operacje na tablicy haszującej z liniowym rozwiązywaniem konfliktów:

1. wyszukiwanie
2. wstawianie
3. usuwanie (omówić)

[dla osób z maksymalnie 2-ma plusami] Zadanie 3. (szukanie sumy w nieposortowanej tablicy) Dana jest nieposortowana tablica $A[1 \ldots n]$ oraz liczba x. Proszę podać algorytm sprawdzający czy istnieją indeksy i oraz j, takie ze $A[i] + A[j] = x$.

[dla osób z najwyżej 3-ma plusami] Zadanie 4. (geny) W pewnym laboratorium genetycznym powstał ciąg sekwencji DNA. Każda sekwencja to pewien napis składający się z symboli G, A, T, i C. Przed dalszymi badaniami konieczne jest upewnić się, że wszystkie sekwencje DNA są parami rózne. Proszę opisać algorytm, który sprawdza czy tak faktycznie jest.

Zadanie 5. (Indeksowane drzewa BST) Rozważmy drzewa BST, które dodatkowo w każdym węźle zawierają pole z liczbą węzłów w danym poddrzewie. Proszę opisać jak w takim drzewie wykonywać następujące operacje:

1. znalezienie i-go co do wielkości elementu,
2. wyznaczenie, którym co do wielkości w drzewie jest zadany węzeł

Proszę zaimplementować obie operacje.

Zadanie 6. Proszę zapropnować algorytm, który oblicza sumę wszystkich wartości w drzewie binarnym zdefiniowanym na węzłach typu:

```
class BNode:
  def __init__( self, value ):
    self.left   = None
    self.right  = None
    self.parent = None
    self.value  = val
```

Można korzystać wyłącznie ze stałej ilości pamięci (ale wolno zmieniać strukturę drzewa, pod warunkiem, że po zakończonych obliczeniach drzewo zostanie przywrócone do stanu początkowego.)

Zadanie 7. (przecięcie zbioru kluczy) Operacja *intersection* otrzymuje na wejściu dwa zbiory zrealizowane jako tablice asocjacyjne i zwraca ich liczbę elementów, które występują w obu. Proszę zaimplementować tę operację dla tablic asocjacyjnych realizowanych jako:

1. drzewa BST,
2. tablice z haszowaniem,

Można założyć, że wszystkie podstawowe operacje na tych strukturach danych są już zamiplementowane.

Zadanie 8. (suma kluczy z przedziału) Proszę opisać jak zmodyfikować drzewa czerwono-czarne (przechowujące liczby jako klucze) tak, by operacja `sum(T, x, y)` obliczająca sumę elementów z drzewa `T` o wartościach z przedziału $[x, y]$ działała w czasie $O(\log n)$ (gdzie n to liczba węzłów drzewa T). Pozostałe operacje na powstałym drzewie powinny miec taka sama złożoność jak w standardowym drzewie czerwono-czarnym.

Zadanie 9. (klocki) Dany jest ciąg klocków $(K_1, \ldots, K_n)$. Klocek K_i zaczyna sie na pozycji a_i i ciągnie się do pozycji b_i (wszystkie pozycje to nieujemne liczby naturalne) oraz ma wysokość 1. Klocki układane są po kolei–jeśli klocek nachodzi na któryś z poprzednich, to jest przymocowywany na szczycie poprzedzajacego klocka). Na przykład dla klocków o pozycjach (1, 3), (2, 5), (0, 3), (8, 9), (4, 6) powstaje konstrukcja o wysokosci trzech klocków. Proszę podac możliwie jak najszybszy algorytm, który oblicza wysokosc powstałej konstrukcji.