Metody programowania - Baca - Zadanie 2 - Drzewo Histogramowe

Treść

Po wykonaniu ostatniego zadania awansowałeś do działu analityki danych. Otrzymałeś bardziej sensowne zadanie zaimplementowania specjalnej struktury danych nazwanej "Drzewem Histogramowym". Jest to zmodyfikowane drzewo BST, które oprócz standardowych operacji umożliwia szybkie określenie, ile elementów znajduje się w danym zakresie wartości.

Drzewo Histogramowe to drzewo BST, w którym każdy węzeł przechowuje nie tylko swoją wartość klucza, ale również liczbę elementów w poddrzewie, którego jest korzeniem. Dzięki temu możliwe jest efektywne odpowiadanie na zapytania typu "Ile elementów jest większych od X a mniejszych od Y?" bez przechodzenia przez wszystkie węzły.

Twoim zadaniem jest zaimplementowanie Drzewa Histogramowego z zachowaniem wymaganych złożoności czasowych dla wszystkich operacji.

Fragment kodu

```
class HistogramNode:
    def __init__(self, key):
        self.key = key
        self.count = 1  # liczba węzłów w poddrzewie (na początku tylko ten węzeł)
        self.left = None
        self.right = None

class HistogramTree:
    def __init__(self):
        self.root = None

    def insert(self, key):
        """Wstawia element do drzewa. Złożoność: O(h)"""
        pass

    def delete(self, key):
        """Usuwa element z drzewa. Złożoność: O(h)"""
        pass

    def search(self, key):
        """Sprawdza czy element jest w drzewie. Złożoność: O(h)"""
        pass

    def count_in_range(self, start, end):
        """Liczy elementy w zakresie [start, end]. Złożoność: O(h)"""
        pass

    def count_less_than(self, key):
```

```
        """Liczy elementy mniejsze od klucza. Złożoność: O(h)"""
        pass

    def count_greater_than(self, key):
        """Liczy elementy większe od klucza. Złożoność: O(h)"""
        pass

    def find_kth_smallest(self, k):
        """Znajduje k-ty najmniejszy element w drzewie. Złożoność: O(h)"""
        pass

    def inorder_traversal(self):
        """Zwraca listę elementów w porządku in-order. Złożoność: O(n)"""
        pass
```

Gdzie h oznacza wysokość drzewa.

Wejście

Na wejściu program otrzymuje serię poleceń:

- `INSERT x` - wstawia wartość x do drzewa
- `DELETE x` - usuwa wartość x z drzewa
- `SEARCH x` - sprawdza czy x jest w drzewie
- `COUNT_RANGE x y` - liczy elementy w zakresie [x, y] (włącznie)
- `COUNT_LESS x` - liczy elementy mniejsze od x
- `COUNT_GREATER x` - liczy elementy większe od x
- `FIND_KTH k` - znajduje k-ty najmniejszy element (numeracja od 1)
- `INORDER` - wyświetla elementy w porządku in-order w formacie `[ elem1, elem2, ... ]`
- `EXIT` - kończy działanie programu

Polecenia są wprowadzane do momentu napotkania polecenia EXIT.

Wyjście

Na wyjściu powinny być wyświetlane:

- Dla polecenia INSERT - `Added: x` lub `Element already exists`
- Dla polecenia DELETE - `Deleted: x` lub `Element does not exist`
- Dla polecenia SEARCH - `YES` jeśli element jest w drzewie, `NO` jeśli nie jest
- Dla polecenia COUNT_RANGE - `Elements in range [x, y]: z` (gdzie z to wynik)
- Dla polecenia COUNT_LESS - `Elements less than x: z`
- Dla polecenia COUNT_GREATER - `Elements greater than x: z`
- Dla polecenia FIND_KTH - k-ty najmniejszy element lub `Invalid index`
- Dla polecenia INORDER - elementy w porządku in-order w formacie `[ elem1, elem2, elem3, ... ]`

Wymagania implementacyjne

1. Na Bacy dostępny jest Python w wersji 2.7.

2. Należy wykorzystać strukturę przedstawioną we fragmencie kodu, dopisując wyłącznie implementacje metod.
3. **Wszystkie operacje z wyjątkiem INORDER muszą mieć złożoność czasową O(h), gdzie h to wysokość drzewa** (w szczególności metoda `count_in_range` nie może przechodzić przez wszystkie elementy w wybranym zakresie).
4. **Wszystkie operacje z wyjątkiem INORDER muszą mieć złożoność pamięciową O(1)** (w szczególności metody inne niż `inorder_traversal` nie mogą wykorzystywać dodatkowych struktur danych).
5. Pole `count` w każdym węźle musi być prawidłowo aktualizowane przy każdej operacji modyfikującej drzewo.
6. Nie można importować zewnętrznych bibliotek.
7. Przy operacji DELETE należy rozważyć wszystkie przypadki.

Przykładowe dane

Wejście:

```
INSERT 50
INSERT 30
INSERT 70
INSERT 20
INSERT 40
INSERT 60
INSERT 80
INORDER
COUNT_RANGE 30 60
COUNT_LESS 50
COUNT_GREATER 50
FIND_KTH 3
DELETE 30
INORDER
SEARCH 30
SEARCH 40
COUNT_LESS 50
FIND_KTH 3
EXIT
```

Wyjście:

```
Added: 50
Added: 30
Added: 70
Added: 20
Added: 40
Added: 60
Added: 80
[ 20, 30, 40, 50, 60, 70, 80 ]
Elements in range [30, 60]: 4
```

```
Elements less than 50: 3
Elements greater than 50: 3
40
Deleted: 30
[ 20, 40, 50, 60, 70, 80 ]
NO
YES
Elements less than 50: 2
50
```