Drzewa i słowniki - rozwiązania

Zadanie 1

Wpisujemy do tablicy zawartość drzewa, w porządku inorder. Otrzymujemy posortowaną tablicę. Teraz mamy rekurencyjną funkcję, która tworzy BST. Bierzemy medianę z tablicy, robimy z niej korzeń i teraz rekurencyjne wywołanie na lewo i prawo od mediany. Otrzymane drzewo jest idealnie zrównoważone, więc na pewno jest i RB i AVL.

Zadanie 2

W węźle należy trzymać sumę liczb w prawym i lewym poddrzewie. Teraz operacja: znajdujemy wspólnego roota dla węzłów o wartościach x oraz y (jest to pierwszy węzeł, z którego idziemy w inne strony szukając węzła x i y).

Aby znaleźć poszukiwaną sumę, trzeba przejść od węzła x rekurencyjnie w górę. Dla samego węzła x bierzemy sumę jego prawego poddrzewa. Gdy idziemy wyżej, to mogliśmy przyjść albo z lewego dziecka, albo z prawego dziecka:

- gdy przyszliśmy z lewego dziecka, to przyszliśmy z elementu mniejszego - dodajemy sumę prawego poddrzewa, elementu z aktualnego węzła i idziemy wyżej
- gdy przyszliśmy z prawego dziecka, to przyszliśmy z elementu mniejszego - po prostu idziemy wyżej

Przerywamy operację, gdy dojdziemy do naszego wspólnego roota. Dla y wykonujemy analogiczną operację, ale w lustrzanym odbiciu - gdy przyszliśmy z lewego dziecka, to po prostu idziemy wyżej, a gdy przyszliśmy z prawego dziecka, to dodajemy sumę lewego poddrzewa (z mniejszymi elementami), element z aktualnego węzła i idziemy wyżej.

Zadanie 3

Mamy tablicę o długości length, w której trzymamy drzewa mapujące int na int. Trzymamy również globalny licznik zrobionych snapów. Podczas operacji set(index, val) dodajemy do drzewa pod index klucz będący aktualnym licznikiem snapów i wartość val pod ten klucz równą wstawianej wartości. W czasie operacji get(index, snap_id) może się okazać, że w drzewie pod indeksem index nie ma klucza snap_id. Wtedy zwracamy wartość spod klucza będącego największym, nie większym niż snap_id. Po to właśnie nam w tym zadaniu drzewa, bo zwykłe słowniki nie mają takiej funkcjonalności.

Zadanie 4

Jest to problem LFU cache (Least Frequently Used).

Wersja mniej ambitna - O(log(n)):

Wartości pod kluczami trzymamy w zwykłym słowniku. Dodatkowo mamy jeszcze drzewo, gdzie kluczami są częstotliwości (liczby wykonanych get'ów), a wartościami listy kluczy ze słownika. Wykonując get wyciągamy klucz z listy pod odpowiednią częstotliwością w drzewie i dopinamy go na koniec listy pod częstotliwością o 1 większą. Ewentualne usuwanie przy put, to po prostu usunięcie pierwszego elementu z listy pod najmniejszym kluczem w drzewie. Żeby wszystko działało szybko, to w słowniku pod danym kluczem trzeba jeszcze trzymać częstotliwość, oraz wskazanie na węzeł listy. Warto dodać, że na kolosie słownik trzymając klucze powinien być drzewem, ponieważ nie zmienia to rzędu złożoności operacji, a pozwala uchronić się przed ćwiczeniowcami z ciemnogrodu.

Wersja bardziej ambitna - O(1) - chyba sobie odpuścimy, ale się da. Na kolosie na pewno by się nie pojawiło, bo w tej wersji już konieczne są słowniki.

Zadanie 5
Serializacja: utworzenie listy z wartości w drzewie w porządku preorder: korzeń, lewe poddrzewo, prawe poddrzewo.
Deserializacja: mamy daną listę wartości preorder, z których chcemy zrobić drzewo, pierwsza wartość to korzeń drzewa. Wykorzystamy do tego stos, żeby nie robić rekurencyjnie (ale tak też się da).

Algorytm:
1. Utwórz stos.
2. Pierwsza wartość to root - zrób z niej roota tworzonego drzewa i wrzuć wierzchołek na stos.
3. Tak długo, jak wartość w liście jest większa niż wartość ze szczytu stosu, usuwaj z wierzchołka stosu. Zrób z danej wartości wierzchołek i zrób z niego prawe dziecko ostatniego usuniętego wierzchołka. Wrzuć nowy wierzchołek na stos.
4. Jeżeli wartość w liście jest mniejsza niż wartość ze szczytu stosu, zrób z niej lewe dziecko wierzchołka ze szczytu stosu. Wrzuć nowy wierzchołek na stos.
5. Powtarzaj kroki 3 i 4 tak długo, jak jeszcze są rzeczy w liście.

Zadanie 6
Rozwiązanie 1:
Trzymamy w każdym węźle rozmiar lewego i prawego poddrzewa. Wtedy znajdujemy łatwo i-ty element w log(n).

Rozwiązanie 2:
Jak mamy wektor, to możemy trzymać jeszcze wektor wskazań na węzły drzewa i pobierać losowy element z wektora. Dodając węzeł dodajemy wskazanie na koniec wektora, a usuwając podmieniamy wartości pod odpowiednim indeksem w wektorze i ostatnim indeksem w wektorze i usuwamy ostatni indeks.