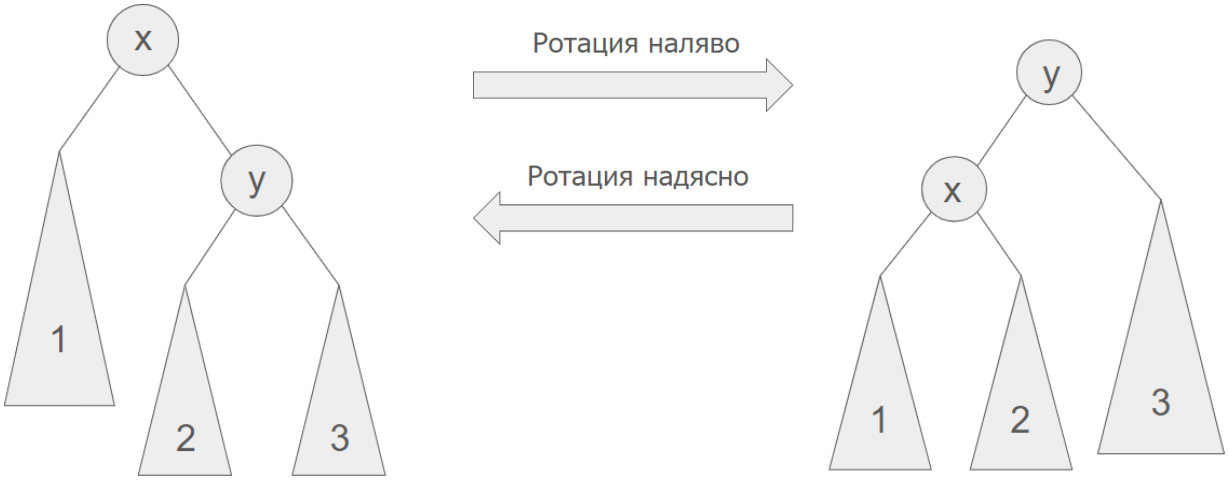
AVL дърво

AVL-дървото е балансирано двоично дърво за търсене, което след всяко добавяне или изтриване на елемент възстановява балансираността си чрез ротации (при необходимост). Кръстено е на откривателите си - Аделсон-Велски и Ландис, които го представят за първи път през 1962 година.

Казваме, че едно дърво е балансирано, ако за всеки негов връх е изпълнено, че разликата във височините на лявото и дясното му поддърво е не по-голяма от едно. Затова за всеки връх x въвеждаме коефициент на баланса, който пази информация за разликата във височините на дясното и лявото поддърво на върха : b(x) = height (rightSubtreeOfX) - height(leftSubtreeOfX).

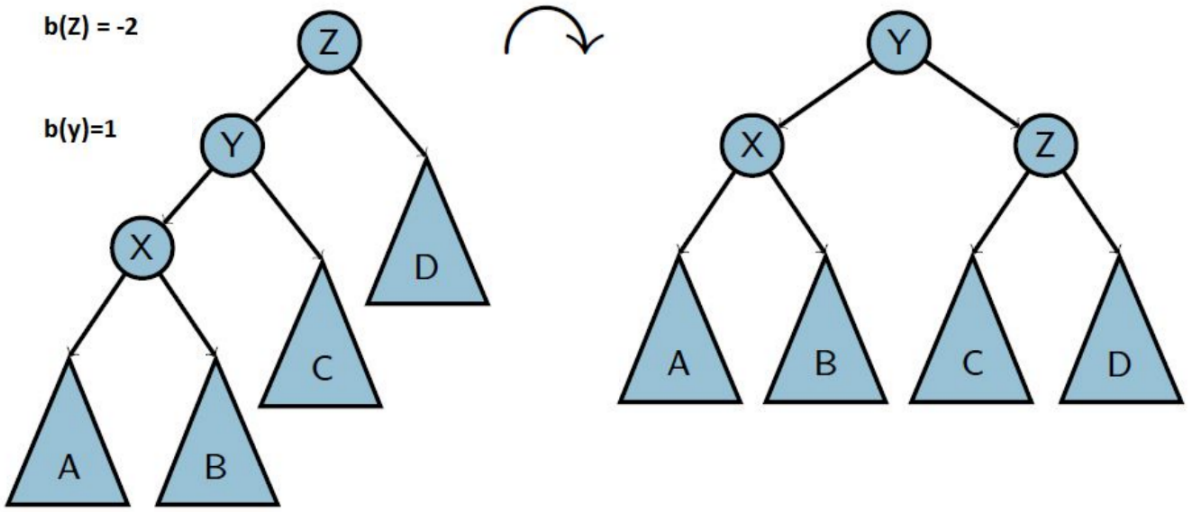
Във всеки един момент в AVL дървото всеки връх има коефициент на баланса -1, 0, или 1. При търсене на елемент в дървото не правим промени по него, затова методът се реализира както при обикновено двоично дърво, като заради балансираността сложността е O(logN). При добавяне на елемент обаче това свойство може да се наруши. Затова, след като добавим елемента по обичайния за BST начин, актуализираме коефициентите на баланса. Тъй като след едно добавяне височината на което и да е поддърво се променя най-много с едно, то ако има проблемен връх, той ще е с коефициент -2 или 2. Проблем може да възникне само при "предшественик" на новодобавения връх, затова трябва да обходим предшествениците отзад напред и да намерим първия връх, в който коефициентът излиза от разрешените граници. След това прилагаме една от четири ротации в зависимост от условията, като след това дървото отново е балансирано.

Ротациите са така измислени, че да запазват основното свойство на двоичното дърво, а именно - за всеки връх стойностите във всички върхове от лявото му поддърво да са по-малки от него, а стойностите от дясното му поддърво да са по-големи или равни на него. На картинката виждаме обща схема на ротация наляво. В първоначалното дърво всички елементи от поддървото 1 са по-малки от x, а y, елементите на 2 и на 3 са по-големи от него. Освен това елементите на 2 са по-малки от y, който от своя страна е по-малък от елементите на 3. Т.е. можем да ги подредим по следния начин: 1 < x < 2 < y < 3. Вътрешната структура на поддърветата 1, 2 и 3 не ни интересува, защото нищо в нея не се променя от ротацията. Виждаме, че и за новото дърво е изпълнено 1 < x < 2< y < 3. Следователно основното свойство на двоичните дървета се запазва при ротация наляво, а аналогично и при ротация надясно.



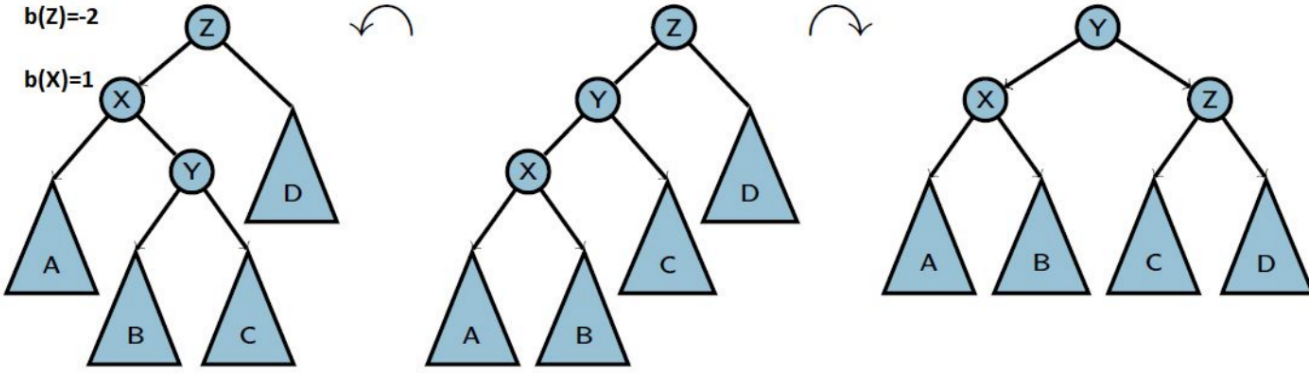
Балансиране с ротация надясно

Нека Z е първият връх (движейки се от новия връх към корена), за който коефициентът на баланса става -2. Тогава трябва да направим ротация надясно. Ако новият връх е добавен към лявото поддърво на лявото дете на Z (на картинката b(Y) = -1), то е достатъчно да направим една ротация надясно върху поддървото с корен Z. Така ще възстановим баланса, като новото поддърво ще има същата височина, която старото е имало преди добавянето.



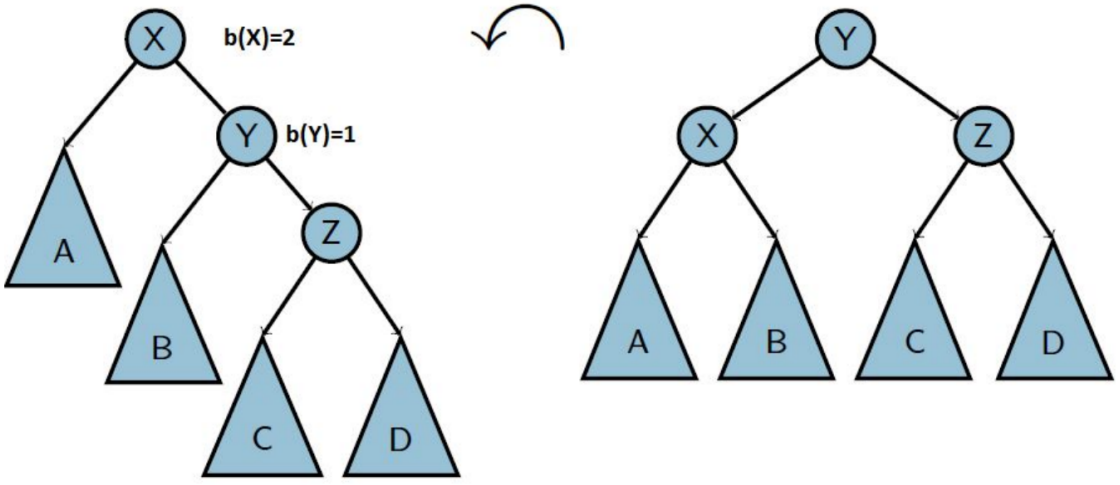
Балансиране с ротация първо наляво и после надясно

Ако обаче новият елемент е добавен към дясното поддърво на лявото дете на Z (на картинката b(X) = 1), то само ротация надясно няма да реши проблема - дървото ще остане небалансирано. Затова първо се прави ротация наляво при X, а чак след това надясно при Z. По този начин вече дървото става балансирано. При това отново се запазва височината отпреди добавянето.



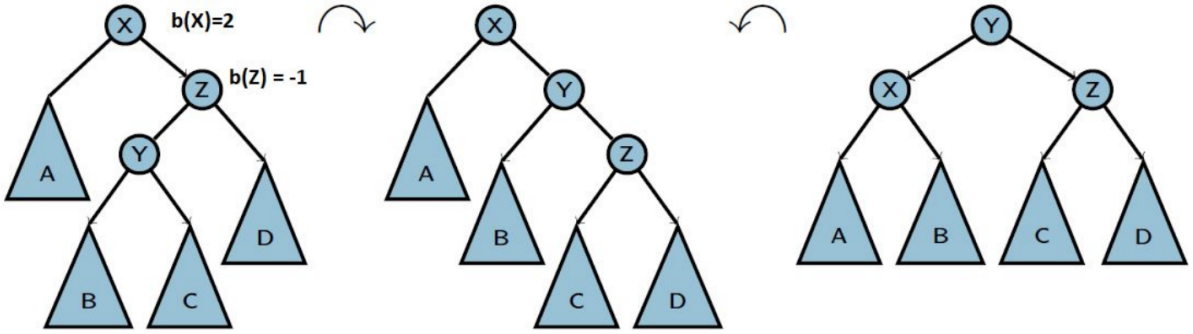
Балансиране с ротация наляво

Ако намерим връх X, за който новият коефициент е 2, логиката е подобна, но ротацията трябва да е наляво. Ако новият връх е добавен към дясното поддърво на дясното дете на X (на картинката b(Y) = 1), то една ротация наляво е достатъчна.



Балансиране с ротация първо надясно и после наляво

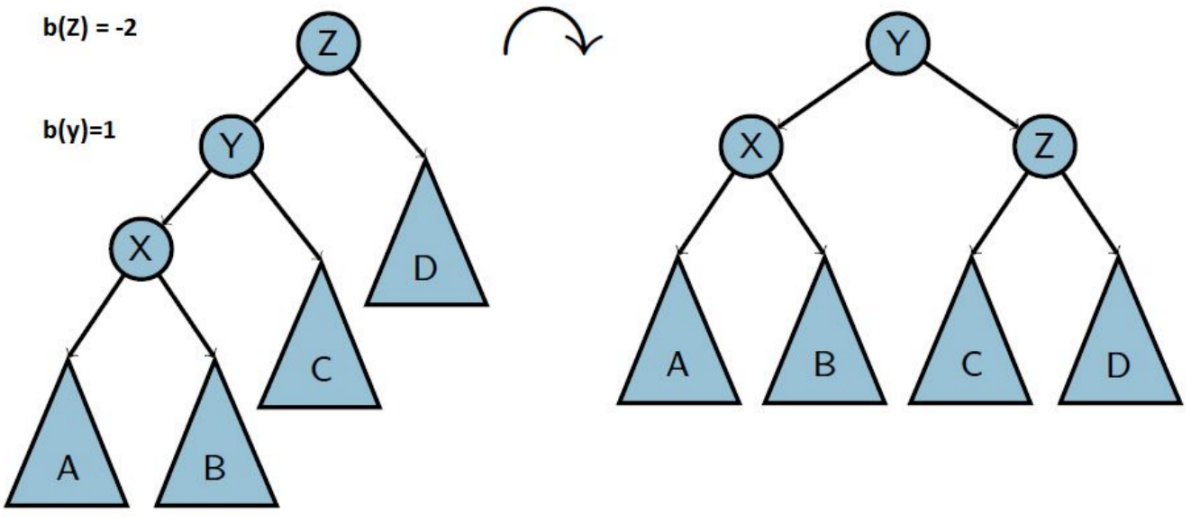
Ако обаче новият връх е добавен към лявото поддърво на дясното дете на X (на картинката b(Z) = -1), то ротация наляво няма да е достатъчна да балансира дървото. Затова правим първо ротация надясно при Z, а след това наляво при X.



За намирането на първоначалната позиция на новия елемент сложността е O(logN), за актуализирането на коефициентите на баланса също е O(logN) (обхождаме само "предшествениците" на новия връх), а сложността на самата ротация е константна. Следователно сложността на добавянето на елемент в AVL дърво е O(logN).

Изтриване на елементи

При изтриването на елементи има някои особености. Първо трием елемента по стандартния за двоично дърво начин, като по този начин височината на дадено поддърво намалява най-много с 1. Нека отново да разгледаме картинката за ротация надясно.



За да стигнем до b(Z) = -2, сме изтрили елемента от поддървото D. Нека височината на D е h. Тук, за разлика от добавянето, може височината и на двете поддървета на Y да е h+1. В този случай след ротацията височината на поддървото е същата, каквато е била преди изтриването. В случая, в който обаче само едното поддърво на Y има височина h+1, а другото - h, след прилагането на подходящата ротация височината на поддървото намалява. Тогава не сме сигурни, че дървото е балансирано, и трябва да продължим да обновяваме коефициентите на баланса нагоре по дървото.

Сложността отново е O(logN), тъй като самото изтриване има сложност O(logN), актуализирането на коефициентите също е със сложност O(logN), а всяка ротация е с константна сложност.

Примерна имплементация

- https://github.com/StefankaManahova/SDP/blob/main/avl_tree.cpp
- https://github.com/MariaGrozdeva/Data_structures_and_algorithms_FMI/blob/main/AdvancedDS-AVL_DoS_Tree/AVL.hpp

Изготвено от Стефанка Манахова, КН 2 курс, 4 група

Източници:

- *https://learn.fmi.uni-sofia.bg/pluginfile.php/187457/mod_resource/content/2/SDA_2018-2019_lecture_8.pdf*
- *https://en.wikipedia.org/wiki/AVL_tree*
- *https://www.programiz.com/dsa/avl-tree*