|  |  |  |  |
| --- | --- | --- | --- |
| Санкт-Петербургский политехнический университет Петра Великого  Институт машиностроения, материалов и транспорта  Высшая школа автоматизации и робототехники | | | |
| Курсовая работа  Дисциплина: Программирование на языках высокого уровня  Тема: АВЛ-дерево (добавление узла) | | | |
| Студент группы 3331506/80401  Преподаватель | |  | А. В. Пестов  М. С. Ананьевский  «\_\_» \_\_\_\_\_\_\_\_ 2021 г. |
|  | Санкт-Петербург  2021 г | |  |

# Оглавление

[Оглавление 2](#_Toc71209348)

[Введение 3](#_Toc71209349)

[Принцип работы 4](#_Toc71209350)

[Оценка скорости и памяти 6](#_Toc71209351)

[Применение алгоритма 6](#_Toc71209352)

[Список литературы 6](#_Toc71209353)

# Введение

АВЛ-дерево — сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

Дерево АВЛ названо в честь двух его советских изобретателей, Георгия Адельсона-Вельского и Евгения Ландиса, которые опубликовали его в своей статье 1962 года «Алгоритм организации информации».

В дереве АВЛ высота двух дочерних поддеревьев любого узла отличается не более чем на единицу; если в любой момент они отличаются более чем на единицу, выполняется перебалансировка для восстановления этого свойства. Поиск, вставка и удаление занимают время O (log n) как в среднем, так и в худшем случаях, где n - количество узлов в дереве до операции. Вставки и удаления могут потребовать перебалансировки дерева путем одного или нескольких вращений дерева.

Деревья АВЛ часто сравнивают с красно-черными деревьями, потому что оба поддерживают один и тот же набор операций и занимают O(log n) в для основных операций. Для приложений с интенсивным поиском деревья АВЛ быстрее, чем красно-черные деревья, потому что они более строго сбалансированы. Подобно красно-черным деревьям, деревья АВЛ сбалансированы по высоте.

В рамках данной курсовой работы будет рассмотрена реализация АВЛ дерева и алгоритм добавления узла на языке программирования *C++* с использованием методов объектно-ориентированного программирования.

# Принцип работы

Алгоритм был реализован при помощи языка программирования C++. Узел дерева представлен структурой node, полями которой являются значение ключа в узле, высота дерева, указатель на структуру node для левой и правой ветви.

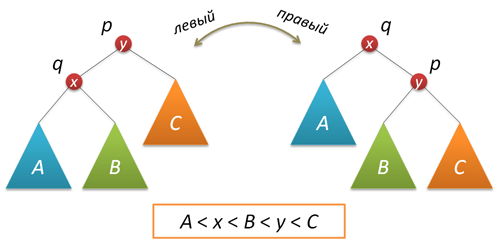
Отсутствие узлов слева или справа будем обнаруживать при помощи нулевого указателя в поле left и right соответственно. Для правильной работы программы необходимо реализовать следующую функцию:

Если на вход подан отсутствующий узел, то она возвращает 0, иначе в поле height узла записывает высоту дерева с корнем в узле.

Функция bfactor возвращает разницу между высотой правой и левой ветви. По свойству AVL дерева он может принимать значения -1, 0, 1. При добавлении и удалении узлов может возникать ситуация, когда это условие нарушится. Для этого в программе предусмотрена функция балансировки дерева.

В качестве вспомогательной функции также выступает функция realheight, которая возвращает наибольшее значение высоты правой и левой ветви узла.

Балансировка узлов может быть осуществлена с помощью поворота вокруг узлов дерева. Правый поворот в программе реализован следующим образом: в функцию передаётся указатель на узел p, узлу q выбирается за левый от узла p. Затем левому узлу от p присваивается значение узла, правого от q. Затем объявляется, что узел p – это узел справа от q. Вычисляются новые высоты узлов p и q и возвращается узел q в качестве указателя на текущий узел. Левый поворот осуществлён аналогичным образом, что показано на рисунке ниже.

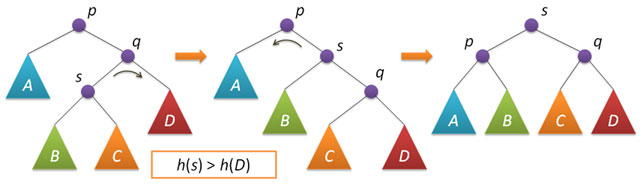


Непосредственно для исправления разбалансировки в узле p достаточно выполнить либо простой поворот влево вокруг p, либо большой поворот влево вокруг того же p. Простой поворот выполняется при условии, что высота левого поддерева узла q меньше или равна высоте его правого поддерева.

Изображение выглядит как текст, часы

Автоматически созданное описание

Большой поворот применяется при условии h(s)>h(D) и сводится в данном случае к двум простым — сначала правый поворот вокруг q и затем левый вокруг p.



Функция балансировки основана на проверке всех этих условий и возвращает тот же узел, который был подан на вход, но со сбалансированными ветвями.

Функция вставки ключа реализована с помощью прохода по дереву, сравнивая значение ключей в каждом из встретившихся узлов с тем, которое вставляется. В случае, если оно меньше, вызывается та же функция для левой ветви, иначе для правой. Если поданного узла не существует, то создаётся в куче новая структура, обозначающая узел, она связана с последним существующим узлом (указатель на неё хранится в поле right или left последнего узла перед тем, как она была создана. При возврате из рекурсии производится балансировка.

# Оценка скорости и памяти

Высота h АВЛ-дерева с n ключами лежит в диапазоне от log2(n + 1) до 1.44 log2(n + 2) − 0.328. Основные операции над двоичными деревьями поиска (поиск, вставка и удаление узлов) линейно зависят от его высоты, что гарантирует логарифмическую зависимость времени работы этих алгоритмов от числа ключей, хранимых в дереве.

Расход памяти O(n).

# Применение алгоритма

АВЛ-деревья могут быть применены для упорядоченного хранения элементов, вставки, поиска и удаления за время от log2(n + 1) до 1.44 log2(n + 2) − 0.328, что требуется, например, для баз данных.

# Список литературы

1. Вирт Н. Алгоритмы и структуры данных. — М.: Мир, 1989. — С. 272—286.
2. Адельсон-Вельский Г. М., Ландис Е. М. Один алгоритм организации информации // Доклады АН СССР. — 1962. — Т. 146, № 2. — С. 263—266.
3. Ben Pfaff. GNU libavl