**АВЛ-дерево**

**АВЛ-дерево** — сбалансированное по высоте [двоичное дерево поиска](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0): для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

Особенностью АВЛ-дерева является то, что оно является сбалансированным в следующем смысле: для любого узла дерева высота его правого поддерева отличается от высоты левого поддерева не более чем на единицу. Доказано, что этого свойства достаточно для того, чтобы высота дерева логарифмически зависела от числа его узлов: высота h АВЛ-дерева с n ключами лежит в диапазоне от log2(n + 1) до 1.44 log2(n + 2) − 0.328. А так как основные операции над двоичными деревьями поиска (поиск, вставка и удаление узлов) линейно зависят от его высоты, то получаем *гарантированную* логарифмическую зависимость времени работы этих алгоритмов от числа ключей, хранимых в дереве. Напомним, что рандомизированные деревья поиска обеспечивают сбалансированность только в вероятностном смысле: вероятность получения сильно несбалансированного дерева при больших n хотя и является пренебрежимо малой, но остается *не равной нулю*.

Максимальная высота АВЛ-дерева при заданном числе узлов:

h <= |1.45 log\_2 (n + 2)|{\displaystyle h\;\leq \;\lfloor 1.45\log \_{2}(n+2)\rfloor \;}

Количество возможных высот на практике сильно ограничено (при 32-битной адресации максимальная высота равна 45, при 48-битной — 68), поэтому лучше заранее подсчитать все возможные значения минимального количества узлов для каждой высоты с помощью рекуррентной формулы для дерева Фибоначчи:

* n(0) = 0;
* n(1) = 1;
* n(h) = n(h - 2) + n(h - 1) + 1{\displaystyle n\_{h}=n\_{h-2}+n\_{h-1}+1}

Промежуточные значения количества узлов будут соответствовать предыдущей (меньшей) высоте.

**Основные операции**

Балансировка

Относительно АВЛ-дерева балансировкой вершины называется операция, которая в случае разницы высот левого и правого поддеревьев = 2, изменяет связи предок-потомок в поддереве данной вершины так, что разница становится <= 1, иначе ничего не меняет. Указанный результат получается вращениями поддерева данной вершины.

tree\* RotateR(tree\* y) { // Правый поворот

tree\* x = y->left;

tree\* r = x->right;

x->right = y; // Вращаем

y->left = r;

y->h = GetMax(Height(y->left), Height(y->right)) + 1; // Меняем значение высоты

x->h = GetMax(Height(x->left), Height(x->right)) + 1;

return x;

}

tree\* RotateL(tree\* x) { // Левый поворот

tree\* y = x->right;

tree\* l = y->left;

y->left = x;

x->right = l;

x->h = GetMax(Height(x->left), Height(x->right)) + 1;

y->h = GetMax(Height(y->left), Height(y->right)) + 1;

return y;

}

В каждом случае достаточно просто доказать то, что операция приводит к нужному результату и что полная высота уменьшается не более чем на 1 и не может увеличиться. Также можно заметить, что большое левое вращение — это композиция правого малого вращения и левого малого вращения. Из-за условия балансированности высота дерева О(log(N)), где N — количество вершин, поэтому добавление элемента требует O(log(N)) операций.

Добавление

tree\* Push(tree\* x, int i) { // Вставка

if (x == NULL) // Если дерево пустое

return(node(i));

if (i < x->i) // Если меньше родителя, левый

x->left = Push(x->left, i);

else if (i > x->i) // Если больше родителя, правый

x->right = Push(x->right, i);

else // Если такой же элемент, то не добавляем

return x;

x->h = 1 + GetMax(Height(x->left), Height(x->right)); // Присваиваем высоту родителю

int b = GetBal(x); // Находим разницу между левым и правым

if (b > 1 && i < x->left->i) // Если высота отличается больше, чем на 1, и значение левого больше элемента (левый, левый)

return RotateR(x); // Крутим узел

if (b < -1 && i > x->right->i) // (правый, правый)

return RotateL(x);

if (b > 1 && i > x->left->i) { // (левый, правый)

x->left = RotateL(x->left);

return RotateR(x);

}

if (b < -1 && i < x->right->i) { // (правый, левый)

x->right = RotateR(x->right);

return RotateL(x);

}

return x;

}

Показатель сбалансированности в дальнейшем будем интерпретировать как разность между высотой левого и правого поддерева. Непосредственно при вставке (листу) присваивается нулевой баланс. Процесс включения вершины состоит из трёх частей:

1. Находится место вставки и вершина, высота которой не изменится при вставке (это вершина, у которой высота левого поддерева не равна высоте правого; будем называть её Node)
2. Выполняется спуск от Node до места вставки с изменением балансов
3. Выполняется ребалансировка Node при наличии переполнения

Скорость вставки элемента равена O(log n)

Удаление

void Del(tree\* A, int i) { // Удаление элемента

if (A != NULL) {

if (A->i != i) // Если элемент не равен удаляемому, то переносим значение в дерево Х

X = Push(X, A->i);

Del(A->left, i);

Del(A->right, i);

}

}

Для простоты опишем рекурсивный алгоритм удаления. Если вершина — лист, то удалим её и вызовем балансировку всех её предков в порядке от родителя к корню. Иначе найдём самую близкую по значению вершину в поддереве наибольшей высоты (правом или левом) и переместим её на место удаляемой вершины, при этом вызвав процедуру её удаления.

Докажем, что данный алгоритм сохраняет балансировку. Для этого докажем по индукции по высоте дерева, что после удаления некоторой вершины из дерева и последующей балансировки высота дерева уменьшается не более, чем на 1. База индукции: Для листа очевидно верно. Шаг индукции: Либо условие балансированности в корне (после удаления корень может измениться) не нарушилось, тогда высота данного дерева не изменилась, либо уменьшилось строго меньшее из поддеревьев => высота до балансировки не изменилась => после уменьшится не более чем на 1.

Очевидно, что в результате указанных действий процедура удаления вызывается не более 3 раз, так как у вершины, удаляемой по второму вызову, нет одного из поддеревьев. Но поиск ближайшего каждый раз требует O(N) операций. Становится очевидной возможность оптимизации: поиск ближайшей вершины может быть выполнен по краю поддерева, что сокращает сложность до O(log(N)).

Высота АВЛ-дерева никогда не превысит высоту идеально сбалансированного дерева более, чем на 45 %. Для больших **n** имеет место оценка **1.04 log\_2 n**. Таким образом, выполнение основных операций требует порядка **log\_2 n** сравнений. Экспериментально выяснено, что одна балансировка приходится на каждые 2 включения и на каждые 5 исключений