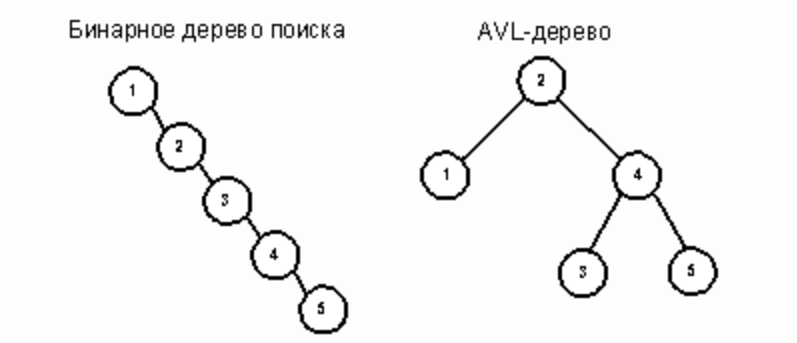
АВЛ-деревья

Бинарные деревья поиска предназначены для быстрого доступа к данным. В идеале разумно сбалансированное дерево имеет высоту порядка $O(\log_2 n)$. Однако при некотором стечении обстоятельств дерево может оказаться вырожденным. Тогда высота его будет O(n), и доступ к данным существенно замедлится.



Рассмотрим модифицированный класс деревьев, обладающих всеми преимуществами бинарных деревьев поиска и никогда не вырождающихся. Они называются сбалансированными. Под сбалансированностью будем понимать то, что для каждого узла дерева высоты обоих его поддеревьев различаются не более чем на 1. Разных реализаций сбалансированных деревьев много: 2-3-деревья, красно-чёрные деревья, АВЛ-деревья.

Будем представлять узел дерева следующей структурой:

```
struct Node
{
  int value;
  int height;
  Node* left;
  Node* right;
};
```

Также нам могут пригодиться три вспомогтельные функции:

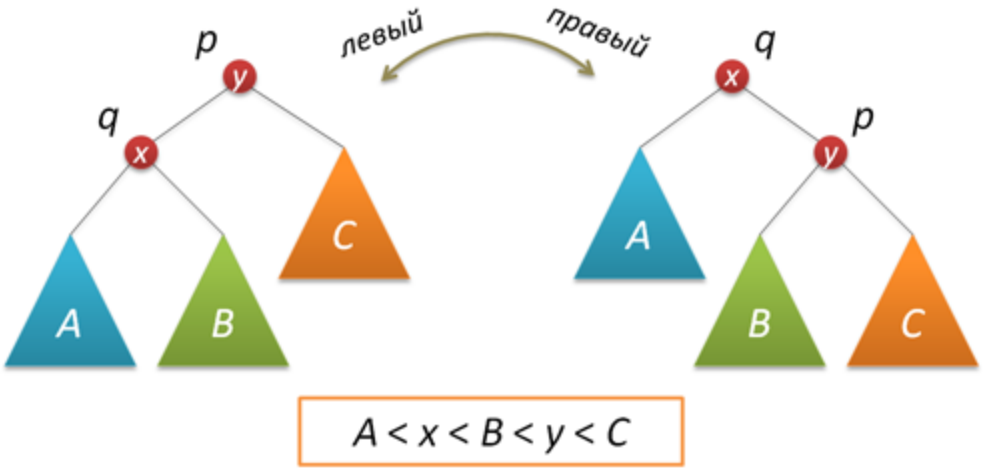
- height(), которая возвращает высоту (функция-обертка)
- balanceFactor() вычисляет balance factor (внезапно!)
- updateHeight() восстанавливает корректное значение высоты узла по значениям его потомков.

```
int height(Node *node)
{
  return node ? node->height : 0;
}

int balanceFactor(Node *node)
```

```
{
  return height(node->right) - height(node->left);
}

void updateHeight(Node *node)
{
  int heightLeft = height(node->left);
  int heightRight = height(node->right);
  node->height = ((heightLeft > heightRight) ? heightLeft : heightRight) + 1;
}
```

Балансировка

В процессе добавления или удаления узлов в АВЛ-дереве возможно возникновение ситуации, когда balance factor некоторых узлов оказывается равными 2 или -2, т.е. возникает расбалансировка поддерева. Для выправления ситуации применяются особые операции, называемые поворотами дерева. Основных поворотов два:

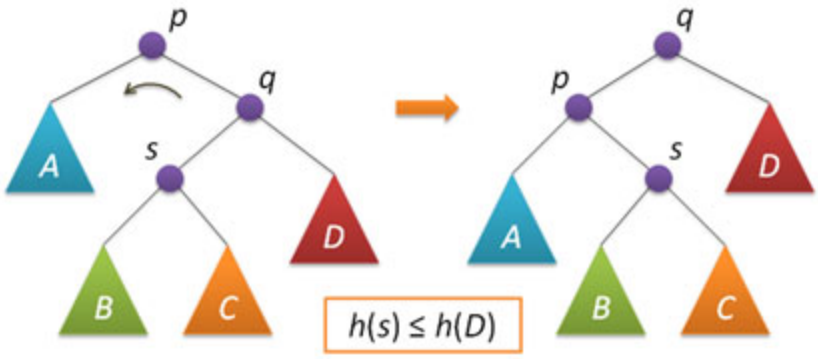


Код, реализующий правый поворот, может быть выглядеть, например, так:

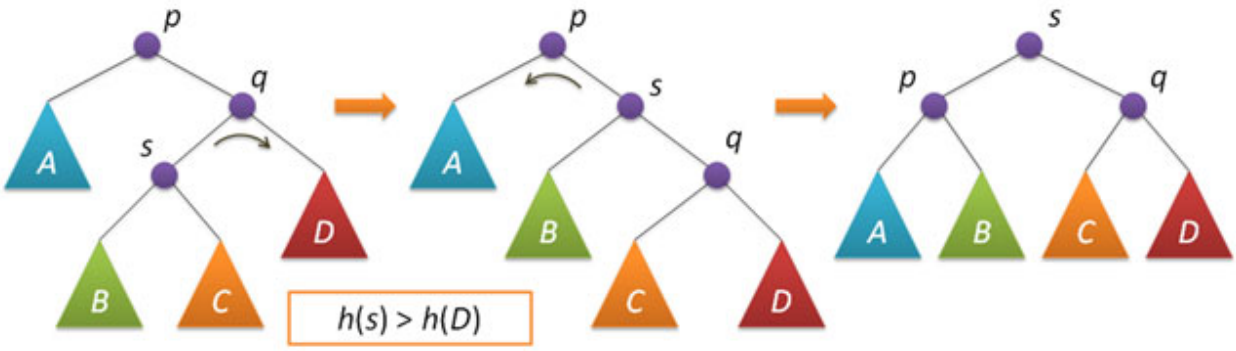
```
void rotateRight(Node* root)
{
  Node* pivot = root->left;
  root->left = pivot->right;
  pivot->right = root;
  updateHeight(root);
  updateHeight(pivot);
  return node;
}
```

Левый поворот реализуется симметрично.

Так вот, рассмотрим, какие бывают случаи при баланс факторе 2. Случаев тут два. Первый — когда высота левого поддерева узла q меньше высоты его правого поддерева: h(s)≤h(D). То есть balanceFactor(p) равен 2, balanceFactor(q) равен 0 или 1. В этом случае достаточно выполнить один левый поворот вокруг p.



Большой поворот применяется при условии h(s)>h(D), то есть balanceFactor(p) равен 2, а balanceFactor(q) равен -1. Такой поворот сводится к двум простым — сначала правый поворот вокруг q и затем левый вокруг p.



В соответствии с описанной схемой код, выполняющий балансировку, сводится к простой проверке условий и выполнению поворотов:

```
Node* balance(Node* p)
{
    updateHeight(p);

    if (balanceFactor(p) == 2)
    {
            if (balanceFactor(p->right) < 0)
                    p->right = rotateRight(p->right);

            return rotateleft(p);
    }
```

```
    if (balanceFactor(p) == -2)
    {
        if (balanceFactor(p->left) > 0)
            p->left = rotateLeft(p->left);

        return rotateRight(p);
    }

    return p;
}
```

Описанные функции поворотов и балансировки также не содержат ни циклов, ни рекурсии, а значит выполняются за постоянное время, не зависящее от размера АВЛ-дерева.

Вставка и удаление элементов

Вставка нового ключа в АВЛ-дерево выполняется практически так же, как это делается в простых деревьях поиска: спускаемся вниз по дереву, выбирая правое или левое направление движения в зависимости от результата сравнения ключа в текущем узле и вставляемого ключа. Единственное отличие заключается в том, что при возвращении из рекурсии (т.е. после того, как ключ вставлен либо в правое, либо в левое поддерево, и это дерево сбалансировано) выполняется балансировка текущего узла. Строго доказывается, что возникающий при такой вставке дисбаланс в любом узле по пути движения не превышает двух, а значит применение вышеописанной функции балансировки является корректным.

Удаление элементов происходит также аналогично удалению в BST, только нужно перебалансировать как сам удаляемый узел, так и дополнительно удаленный элемент в случае двух потомков.