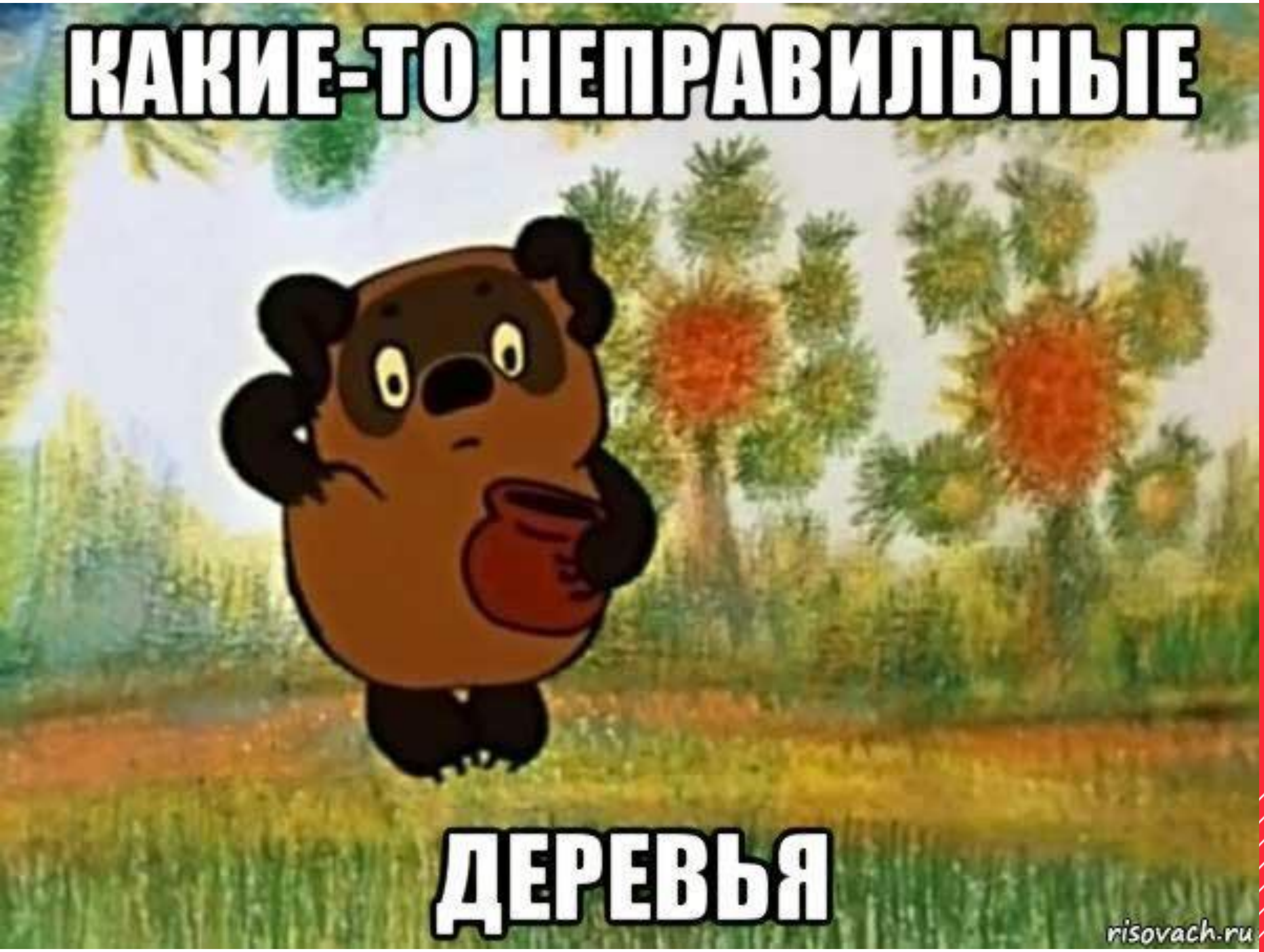
Деревья поиска 1

Шовкопляс Григорий

Введение в алгоритмы и структуры данных



Что такое дерево поиска и с чем его есть?

Дерево поиска

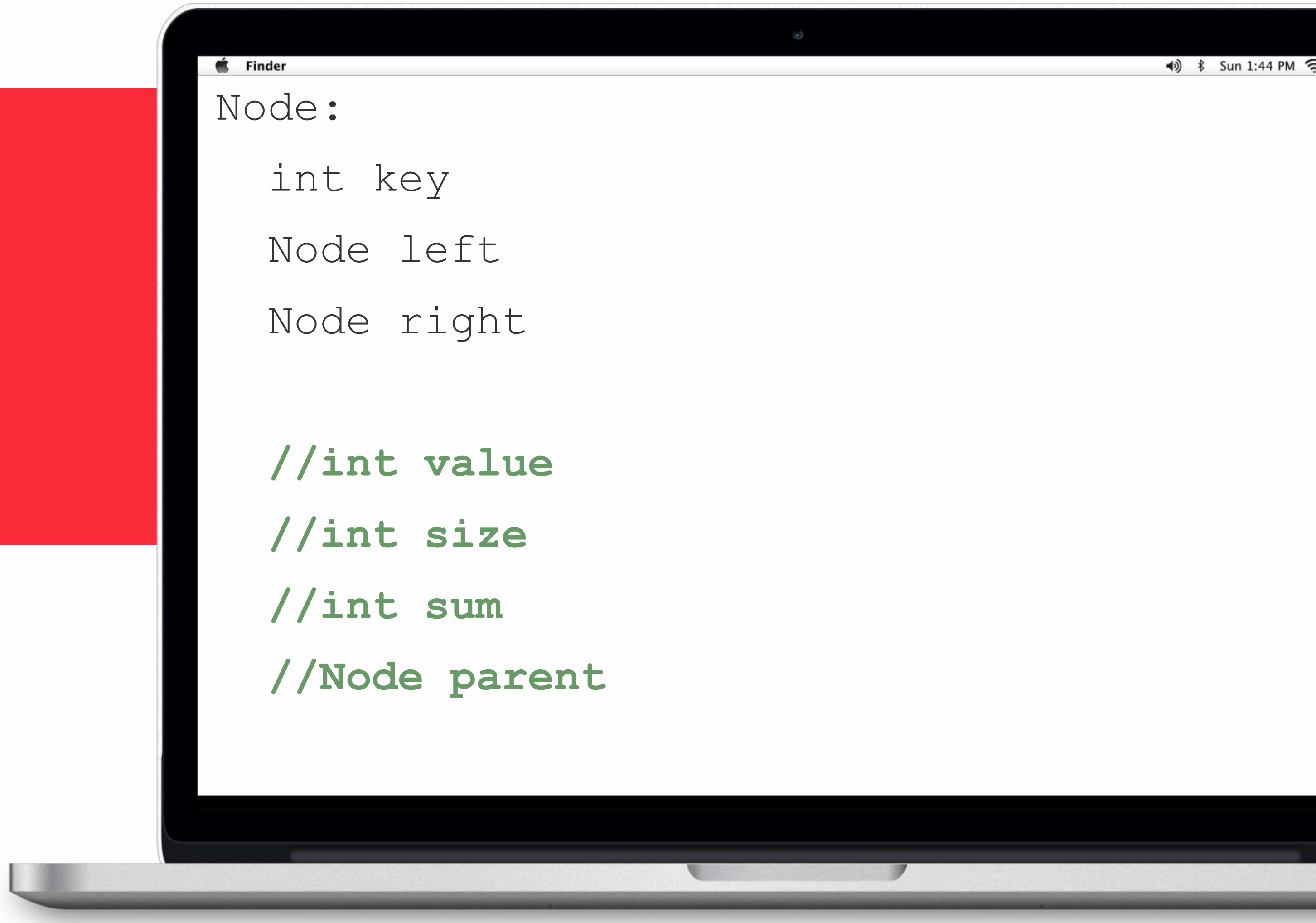
- Двоичное* дерево
- Если в текущей вершине значение X
- В левом поддереве все значения меньше X
- В правом больше X

Дерево поиска

- Операции:
    - Поиск элемента по ключу
    - Вставка
    - Удаление
    - Хм… чем-то похоже на set…
- Занятные свойства:
    - В готовое дерево поиска элемент вставляется единственным образом
    - Для одного множества ключей, существует несколько деревьев поиска

Дерево поиска

Структура



Дерево поиска

Поиск ключа

Возвращаем ссылку на узел

```
search(v, x):
  if v == null
    return null
  if v.key == x
    return v
  else if x < v.key
    return search(v.left, x)
  else
    return search(v.right, x)
```

Дерево поиска

Вставка ключа

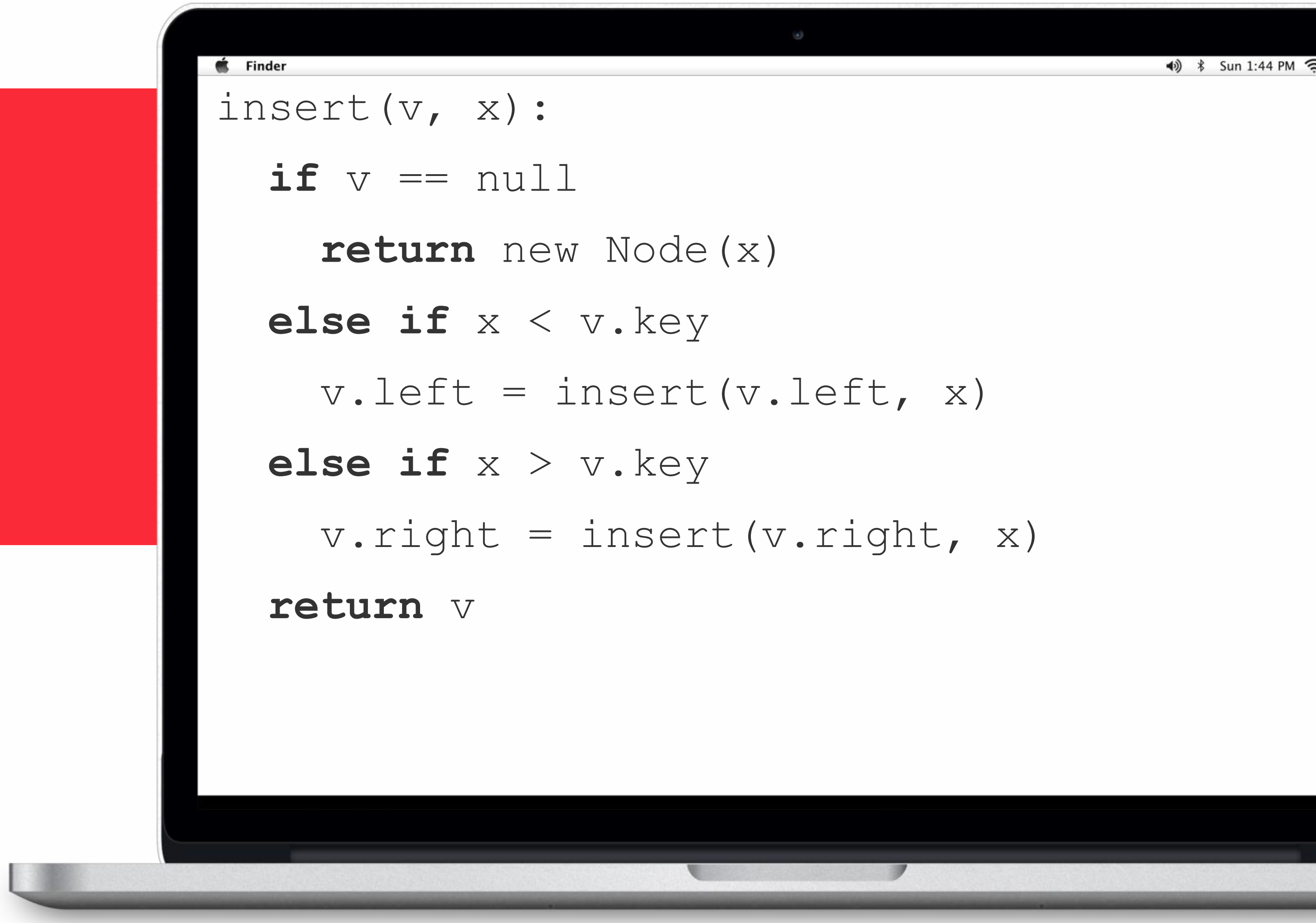
Возвращаем ссылку на дерево с уже вставленным элементом

```
insert(v, x):
  if v == null
    return new Node(x)
  if v.key == x
    return v
  else if x < v.key
    v.left = insert(v.left, x)
  else
    v.right = insert(v.right, x)
  return v
```

Дерево поиска

Вставка ключа

Немного причешем код



Дерево поиска

- А как удалять?
- Для начало нужно найти искомый узел
    - Если не нашли, то и удалять не надо
    - Если лист, то вроде понятно более менее
    - А иначе?
    - Если ребенок один, то тоже понятно
    - Давайте начнем!

Дерево поиска

Удаление элемента

Возвращаем ссылку на дерево с уже удаленным элементом

Дерево поиска

- Как удалить не лист?
- Вспоминаем: свойство вершины
- Когда удалим узел, останется два поддерева, кто может быть новым родителем?
  - Минимум правого поддерева
  - Максимум левого поддерева
- Как кого-нибудь из них найти?

Дерево поиска

Удаление элемента

Если лист или один ребенок

Заметим, что случай листа вырожден

```
delete(v, x):
  …
  if v.right == null and v.left == null
    v = null //почистить память!
  else if v.left == null
    v = v.right //и тут!
  else if v.right == null
    v = v.left //и даже тут!
  else ???
  return v
```

Дерево поиска

Удаление элемента

Последний случай

```
delete(v, x):
  …
  else if v.right == null
    v = v.left //и даже тут!
  else
    v.key = findMax(v.left).key
    v.left = delete(v.left, v.key)
  return v
```

Дерево поиска

Максимум в поддереве

Минимум аналогично

```
findMax(v):
  while v.right != null
       v = v.right
  return v
```

Дерево поиска

- Что еще бывает полезно?
- Вывести все элементы дерева в отсортированном порядке
- Искать следующий или предыдущий ключ

Дерево поиска

Вывод

```
printTree(v):
  if v != null
    printTree(v.left)
    write(v.key)
    printTree(v.right)
  return v
```

Дерево поиска

Поиск следующего (с информацией о родителе просто, сразу без этого)

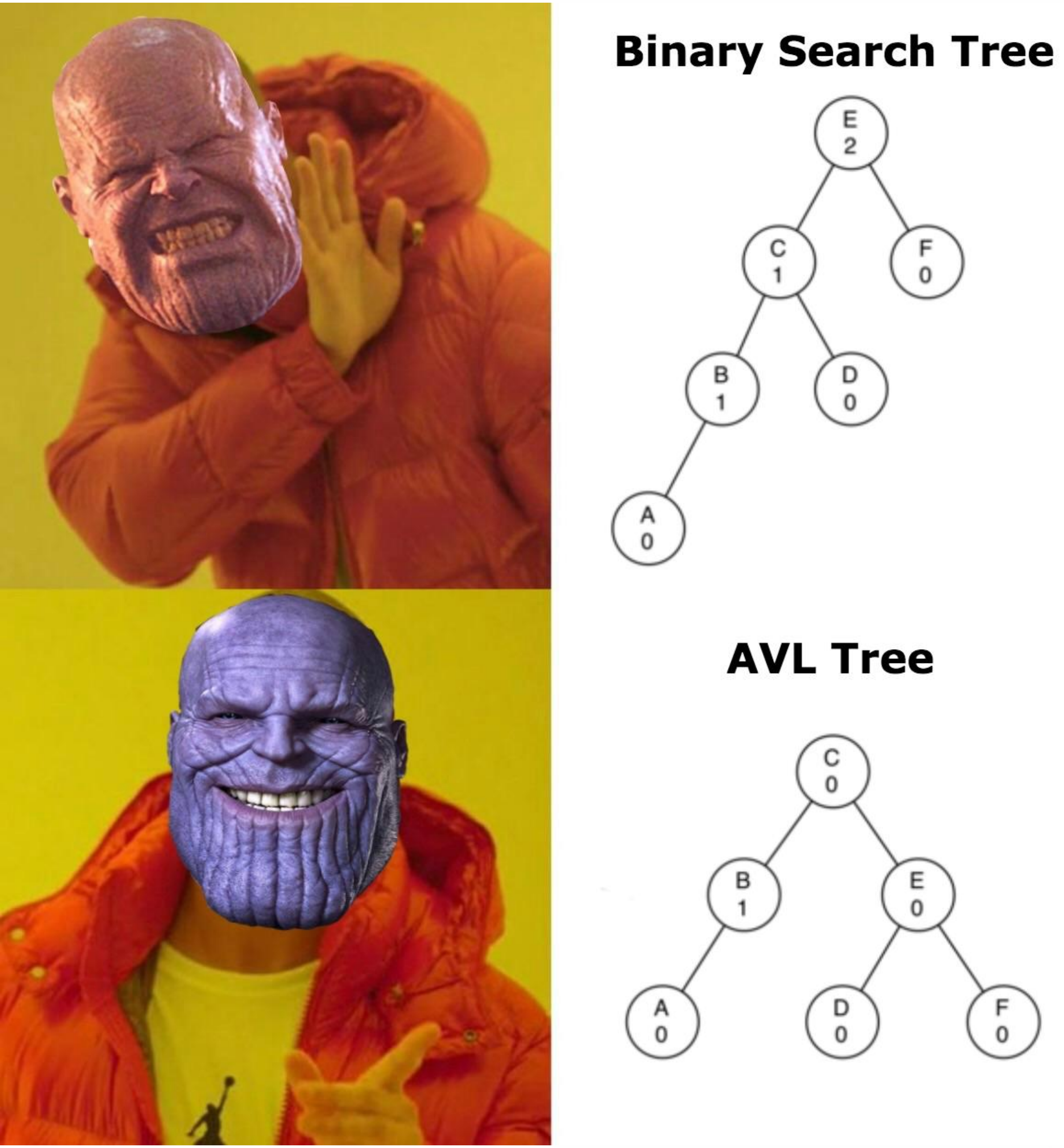
Какой инвариант?

Поиск предыдущего, аналогично

```
next(x):
  v = root, res = null
  while v != null
    if v.key > x
      res = v
      v = v.left
    else
      v = v.right
  return res
```

Дерево поиска

- Какое было время работы у всех рассмотренных операций?
- printTree за O(n)
- Остальные за O(h), где h высота дерева
- Давайте оценим высоту двоичного дерева поиска:
    - $\log_2 n \leq h \leq n$
- Как гарантировать маленькую высоту?



AVL-дерево

AVL-дерево

- По ощущениям, если глубины всех листов будут плюс-минус равны, то дерево будет логарифмической высоты
- AVL-дерево – двоичное дерево поиска с дополнительным свойством:
    - Для любой вершины высоты ее двух поддеревьев отличаются не больше, чем на единицу

AVL-дерево

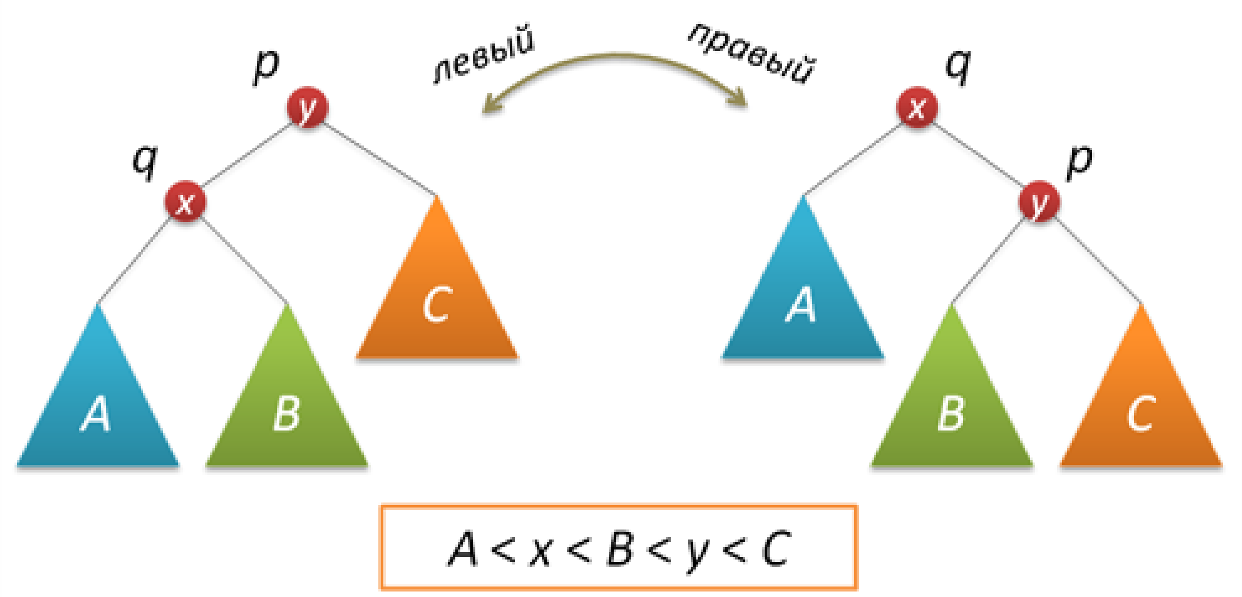
- Лемма: высота АВЛ-дерева O(log n)
- Пусть m_h – минимальное число вершин в АВЛ-дереве высоты h
- Заметим, что $m_h = m_{h-1} + m_{h-2} + 1$
- Похоже на числа Фибоначчи
- Докажем: $m_h = F_{h+2} - 1$
- $n \geq m_h = F_{h+2} - 1 \geq F_h \geq \left(\sqrt{2}\right)^h$
- $\log n \geq h \Rightarrow h = O(\log n)$
- Осталось понять, а как гарантировать выполнение такого свойства

AVL-дерево

- Балансировать надо, когда после удаления/добавления баланс стал ± 2
- Балансируем четырьмя видами поворотов:
    - Малое левое/правое вращение
    - Большое левое/правое вращение

AVL-дерево

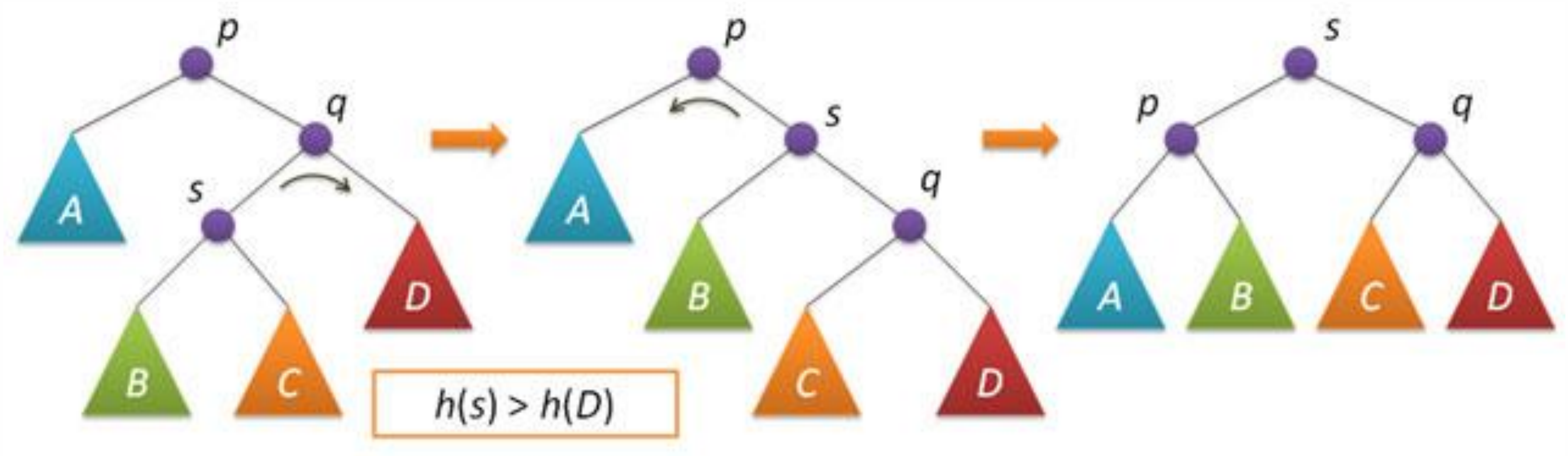
- Малое левое/правое вращение
- Почему этого вида поворотов недостаточно?



Картинка взята с https://habr.com/ru/post/150732/

AVL-дерево

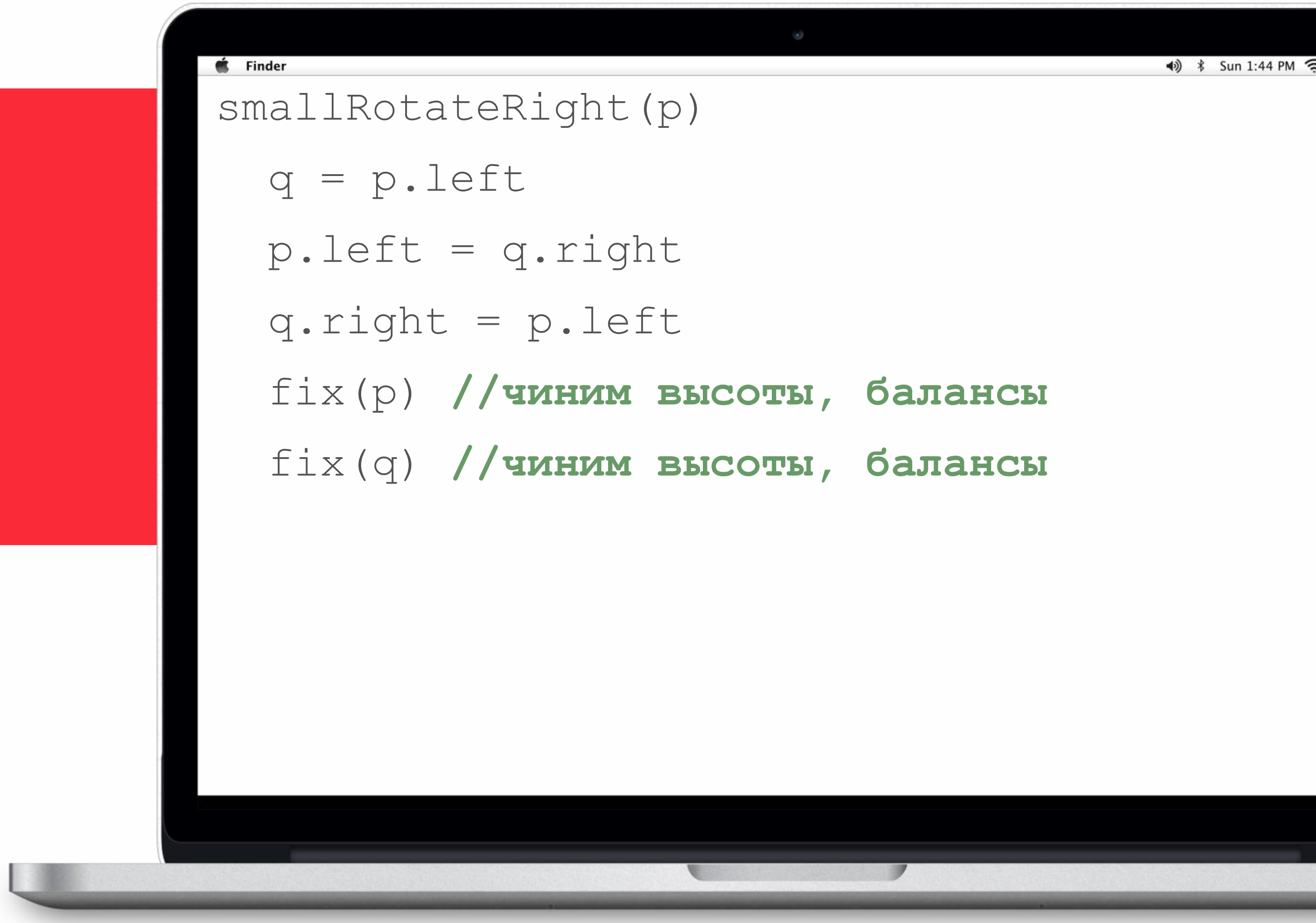
- Большое левое/правое вращение



Картинка взята с https://habr.com/ru/post/150732/

AVL-дерево

Повороты это просто!



AVL-дерево

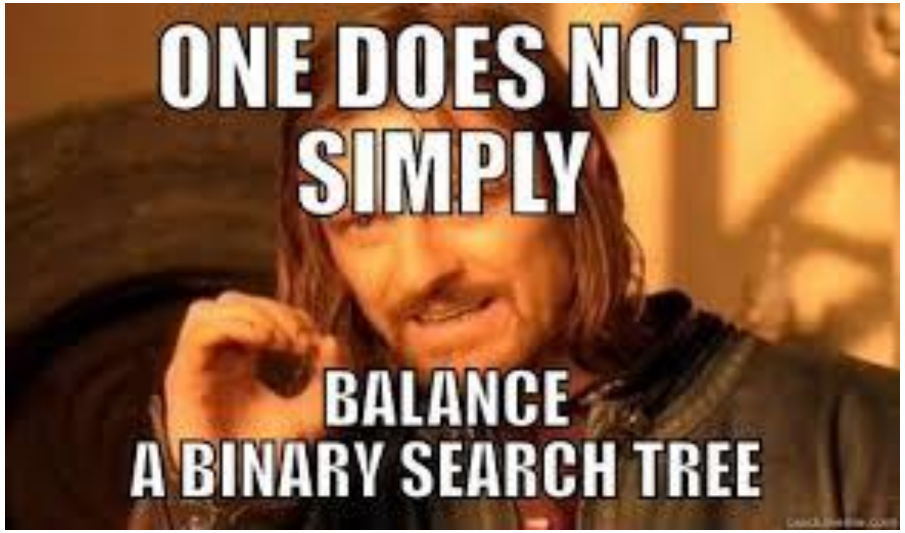
Повороты это просто!

AVL-дерево

Все остальное тоже просто!

С остальными операциями все также, поиск вообще, как в обычном BST

```
insert(v, x):
  if v == null
    return new Node(x)
  else if x < v.key
    v.left = insert(v.left, x)
  else if x > v.key
    v.right = insert(v.right, x)
  return balance(v)
```



Какие еще бывают сбалансированные деревья поиска?

Декартово дерево

- Обсудим и очень подробно на следующем занятии!

Красно-черное дерево

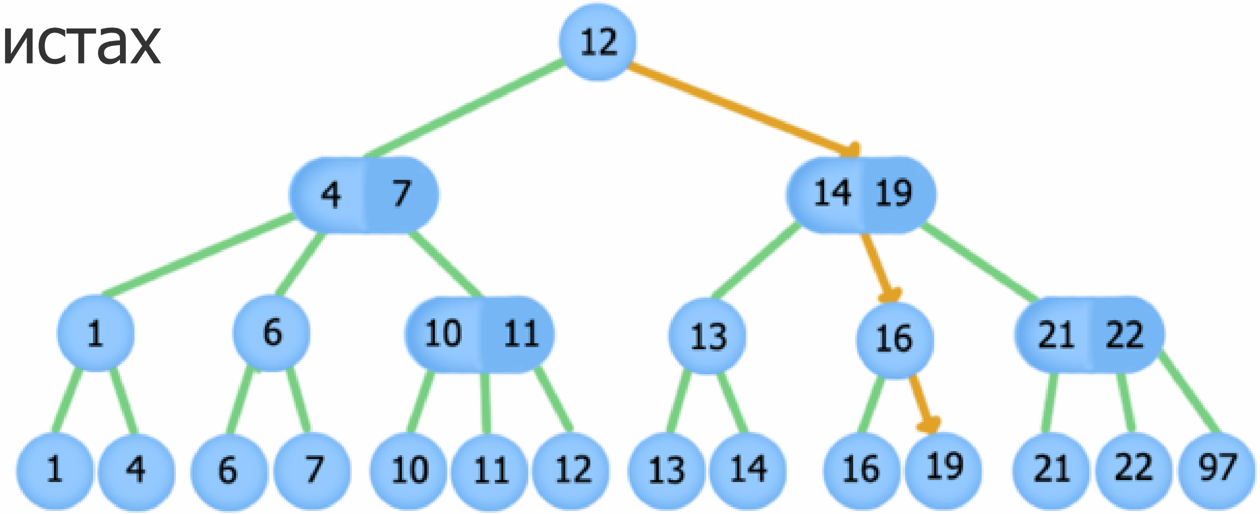
- Двоичное дерево поиска
- Теперь у каждого узла есть цвет красный или черный и куча свойств:
  - Корень и листья дерева — чёрные
  - У красного узла родительский узел — чёрный
  - Все простые пути из любого узла до листьев содержат одинаковое количество чёрных узлов
  - Чёрный узел может иметь чёрного родителя
- В C++ set и map реализованы как раз с помощью такого дерева

Красно-черное дерево

- Черная высота — число черных вершин на пути в лист
- Лемма: В красно-черном дереве с черной высотой *hb* количество внутренних вершин не менее $2^{hb-1}-1$
- Высота дерева *h* и у красной вершины, черные дети ⇒ черная высота такого дерева не меньше $\frac{h}{2}-1$
- Тогда $n \geq 2^{\frac{h}{2}}-1 \Rightarrow \log(n) \geq \frac{h}{2} \Rightarrow h \leq 2\log(n+1)$
- Т.о. $h = O(\log(n))$

2-3 дерево

- У каждого не листа 2 или 3 ребенка
- Глубины всех листов одинаковы
- Вся информация в листах



B дерево

- 2-3 дерево частный случай B+ дерева
- У каждого не листа от 2 до B детей
- Из памяти удобно читать блоки размера B!

Splay дерево

- Двоичное дерево поиска
- «магически» балансируется с помощью операции splay
- Splay «поднимает» узел в корень
- Амортизировано работает за O(log(n))

Все!