Сбалансированные деревья поиска

Дискретный анализ 2016/17

10 сентября 2016 г.

Литература

- Д. Кнут. Искусство программирования, том 3, Сортировка и поиск, 2-е издание, М.:Вильямс, 2003, стр. 492–509, п. 6.2.3. «Сбалансированные деревья».
- Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.. Алгоритмы: построение и анализ, 2-е издание, М.:Вильямс, 2005, стр. 336-359, глава 13, «Красно-черные деревья».
- Седжвик Р., Алгоритмы на C++, М.:Вильямс, 2014, стр. 487-527, глава 13, «Сбалансированные деревья».

Поиск

- Различные подходы
- Бинарные деревья поиска
- Операции с деревьями поиска

AVL-деревья

- Определение, высота дерева
- Вставка
- Удаление

Красно-черные деревья

- Определение, высота дерева
- Вставка
- Удаление

Сбалансированные деревья поиска

Где, что и как ищем

- Храним пары ключ-значение. Хотим найти значения или просто факт наличия для ...
  - точных значений ключей;
  - ключей в интервалах;
  - префиксов ключей;
  - ключей с подстроками;
  - ключей, удовлетворяющих маскам.
- Какова скорость доступа к данным?
  - равномерный (случайный) доступ;
  - скорость неравномерна, есть локальность доступа.
- Как часто ...
  - ищем?
  - добавляем?
  - изменяем?
  - удаляем?

Поиск

- Основные алгоритмы:
  - последовательный поиск — $O(n)$;
  - бинарный поиск в статическом отсортированном массиве — $O(\log n)$;
  - поиск в сбалансированных и сильноветвящихся деревьях — $O(\log n)$;
  - поиск с использованием хеширования — $O(1)$ (при наличии «хорошей» хеш-функции);
  - цифровой поиск — $O(|key|)$.
- Специализированные алгоритмы:
  - оптимальные деревья поиска;
  - string B-tree;
  - суффиксные деревья.

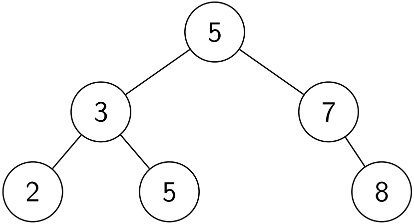
Сбалансированные деревья поиска

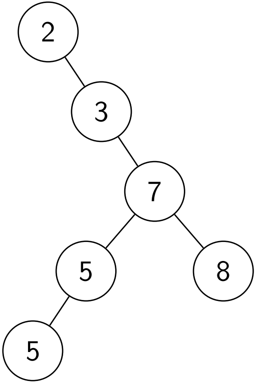
- Поиск
  - Различные подходы
  - Бинарные деревья поиска
  - Операции с деревьями поиска
- AVL-деревья
  - Определение, высота дерева
  - Вставка
  - Удаление
- Красно-черные деревья
  - Определение, высота дерева
  - Вставка
  - Удаление

Дерево поиска

- Левое поддерево содержит меньшие относительно корня ключи, правое — большие
- Может выродиться в линейный список
- Идеально сбалансированное дерево поиска — дерево наименьшей высоты $(\log_2 n)$
- Сбалансированное дерево — дерево, для которого выполняется какое-то условие баланса

Примеры деревьев поиска





Сбалансированные деревья поиска

Вывод ключей в упорядоченном порядке

```
PRINT-TREE(x)
if x ≠ NULL
    PRINT-TREE(left[x])
    print key[x]
    PRINT-TREE(right[x])
```

Поиск

```
Tree-Search(x, k)
if x = NULL || k = key[x]
        return x
if k < key[x]
        return Tree-Search(left[x], k)
else
        return Tree-Search(right[x], k)
```

Поиск

TREE-SEARCH-ITERATIVE(x, k)

```
while x ≠ NULL && k ≠ key[x]
    if k < key[x]
        x ← left[x]
    else
        x ← right[x]
return x
```

Минимум/максимум

Tree-Min(x)

while $left[x] \neq NULL$
    $x \leftarrow left[x]$
return x

Tree-Max(x)

while $right[x] \neq NULL$
    $x \leftarrow right[x]$
return x

Следующий/предшествующий элемент

$\textsc{Tree-Successor}(x)$

if $right[x] \neq NULL$
    return $\textsc{Tree-Min}(right[x])$
$y \leftarrow p[x]$
while $y \neq NULL \,\&\&\, x = right[y]$
    $x \leftarrow y$
    $y \leftarrow p[y]$
return y

Вставка

```
Tree-Insert(T, z)
y ← NULL
x ← root[T]
while x ≠ NULL
    y ← x
    if key[z] < key[x]
        x ← left[x]
    else
        x ← right[x]
p[z] ← y
if y = NULL // T – пустое
    root[T] = z
else
    if key[z] < key[y]
        left[y] ← z
    else
        right[y] ← z
```

Удаление

$\textsc{Tree-Delete}(T, z)$

if $left[z] = NULL \parallel right[z] = NULL$
    $y \leftarrow z$
else $y \leftarrow \textsc{Tree-Successor}(z)$
if $left[y] \neq NULL$
    $x \leftarrow left[y]$
else $x \leftarrow right[y]$
if $x \neq NULL$
    $p[x] \leftarrow p[y]$
if $p[y] = NULL$
    $root[T] \leftarrow x$
else if $y = left[p[y]]$
        $left[p[y]] \leftarrow x$
    else $right[p[y]] \leftarrow x$
if $y \neq z$
    $key[z] \leftarrow key[y]$ // Копируем содержимое y в z
return y

Поворот

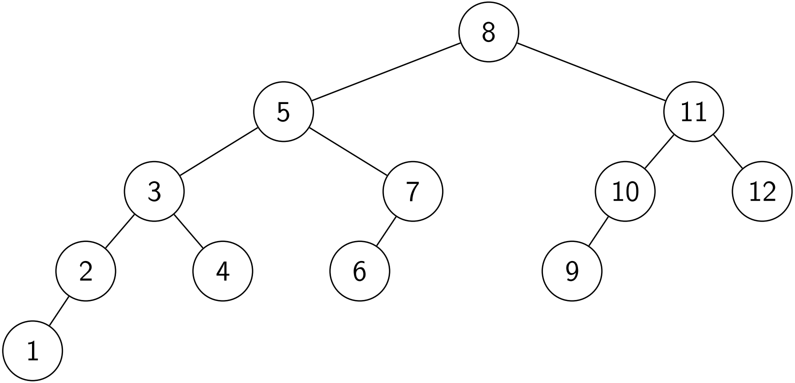
```
Left-Rotate(T, x)
y ← right[x]
right[x] ← left[y]
if left[y] ≠ NULL
        p[left[y]] ← x
p[y] ← p[x]
if p[x] = NULL
        root[T] ← y
else if x = left[p[x]]
                left[p[x]] ← y
        else right[p[x]] ← y
left[y] ← x
p[x] ← y
```

Сбалансированные деревья поиска

AVL-деревья

- Обычное дерево поиска при «плохих» данных вырождается в линейный список
- Высота дерева — длина самого длинного пути от корня к листу
- AVL-дерево — дерево, у которого высота левого поддерева любого узла отличается от высоты правого не более, чем на 1
- Удаление, добавление и поиск элемента в AVL-дереве производится за $O(\log n)$
- Дополнительно вместе с каждой вершиной храним разность высот левого и правого поддеревьев

Пример AVL-дерева



Высота AVL-дерева

Лемма

Пусть m_h — минимальное число вершин в AVL-дереве высоты h. Тогда $m_h = F_{h+2} - 1$, где F_h – h-ое число Фибоначчи.

Доказательство.

По индукции.

- База: $h = 1 : m_1 = F_3 - 1 = 1$
- Предположение: для m_h – верно
- $m_{h+1} = m_h + m_{h-1} + 1 = F_{h+2} - 1 + F_{h+1} - 1 + 1 = F_{h+3} - 1$

$\square$

Высота AVL-дерева

Теорема

Высота AVL-дерева есть $O(\log n)$, где n — количество узлов в дереве.

Доказательство.

$$F_n = \Omega(\phi^n) = \Omega\left(\left(\frac{\sqrt{5}+1}{2}\right)^n\right)$$

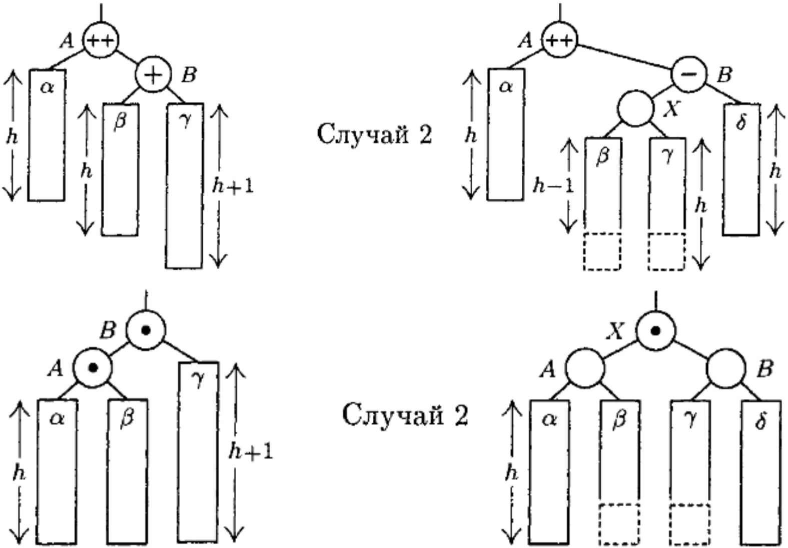
Поэтому $n \geq m_h = F_{h+2} - 1$, значит $\log n \geq \log \phi^{h+2}$, следовательно $h = O(\log n)$. $\square$

Сбалансированные деревья поиска

Вставка в AVL-дерево

1. Поиск ключа в дереве
2. Вставка нового ключа
3. Балансировка дерева при необходимости:
   - вычисляем новую разность высот поддеревьев по пути наверх;
   - если баланс нарушен, выполняем один из поворотов;
   - проверяем условие окончания, если не выполнилось – продолжаем подъем.

Балансировка при вставке



Вставка в AVL-дерево

- Вставляем элемент как в обычное дерево поиска
- Поднимаемся вверх и пересчитываем баланс: пришли из левого поддерева – прибавим 1 к балансу, из правого – вычтем 1
- Баланс $= 0$ – высота не изменилась – балансировка окончена
- Баланс $= \pm\, 1$ – высота изменилась, продолжаем подъем
- Баланс $= \pm\, 2$ – инвариант нарушен, делаем соответствующие повороты, если получаем новый баланс, равный 0, то останавливаемся, иначе ($\pm\, 1$) – продолжаем подъем

Сбалансированные деревья поиска

Удаление из AVL-дерева

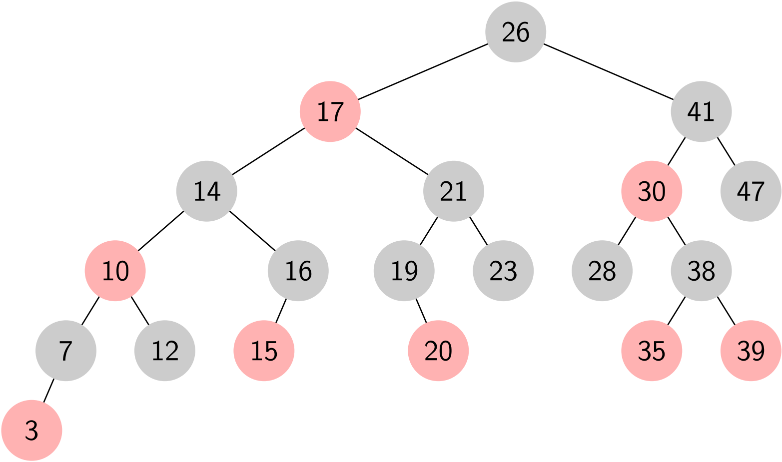
- Удаляем элемент как в обычном дереве поиска
- Поднимаемся вверх от удаленного элемента и пересчитываем баланс: пришли из левого поддерева – вычтем 1 из баланса, из правого – прибавим 1
- Баланс $= \pm 1$ – высота не изменилась – балансировка окончена
- Баланс $= 0$ – высота уменьшилась – продолжаем
- Баланс $= \pm 2$ – инвариант нарушен, делаем соответствующие повороты, если получаем новый баланс, равный 0, то продолжаем подъем, иначе – останавливаемся

Сбалансированные деревья поиска

Красно-черные деревья

1. Каждый узел является либо красным, либо черным
2. Корень дерева является черным
3. Каждый лист дерева (NIL) является черным
4. Если узел дерева красный, то оба дочерних узла — черные
5. Для каждого узла все пути от него до листьев-потомков содержат одинаковое количество черных узлов

Пример



Высота красно-черного дерева

Лемма

Поддерево любого узла x красно-черного дерева T содержит не менее $2^{bh(x)} - 1$ внутренних узлов.

Доказательство.

$bh(x)$ – черная высота узла x – количество черных узлов на пути от x к листу, не учитывая саму вершину x.
По индукции.

- База: высота x равна 0, значит x – фиктивный узел, поэтому $2^{bh(x)} - 1 = 2^0 - 1 = 0$ внутренних узлов – верно
- Рассмотрим узел x с черной высотой $bh(x)$; черная высота потомков либо $bh(x)$, либо $bh(x) - 1$, поэтому, исходя из предположения индукции:
  $\left(2^{bh(x)-1} - 1\right) + \left(2^{bh(x)-1} - 1\right) + 1 = 2^{bh(x)} - 1$

$\square$

Высота красно-черного дерева

Теорема

Красно-чёрное дерево с n внутренними узлами имеет высоту не более чем $2\log_2(n+1)$.

Доказательство.

- Согласно свойству 4, по крайней мере половина узлов на пути от корня к листу должны быть черными (без учета корня), значит $bh(x) \geq \frac{h}{2}$
- $n \geq 2^{bh(x)} - 1$
- Получаем: $n \geq 2^{\frac{h}{2}} - 1$, отсюда $h \leq 2\log_2(n+1)$
- Значит, $h = O(\log n)$

□

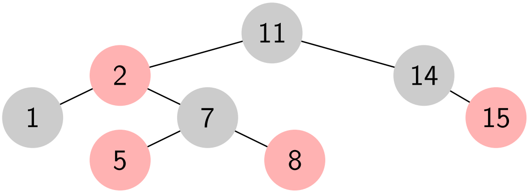
Сбалансированные деревья поиска

Вставка. Общая идея

- Выполняется вставка узла z в дерево T (как в обычное дерево поиска)
- Вставленный узел красится в красный цвет; возможно нарушено свойство 4
- Вызывается процедура, восстанавливающая инварианты красно-черного дерева; до тех пор, пока родитель узла z красный:
  - пусть y — «дядя» узла z; если дядя красный, то нужно перекрасить его и отца в черный цвет, а дедушку в красный, и переместить указатель z на дедушку;
  - если дядя черный, а узел z — правый сын, то нужно переместить указатель z на отца и повернуть их влево;
  - если дядя черный, а узел z — левый сын, то нужно покрасить отца в черный цвет, деда – в красный, а затем повернуть отца и деда вправо.

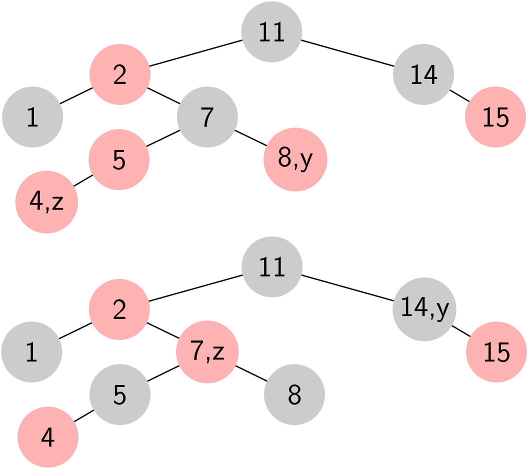
Вставка. Иллюстрация

Исходное дерево:

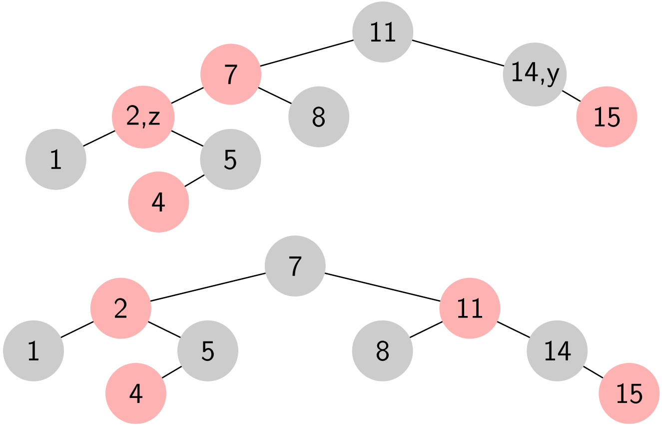


Вставляем элемент 4

Вставка. Иллюстрация



Вставка. Иллюстрация



Вставка. Алгоритм

$\textsc{RB-Insert-Fixup}(T, z)$

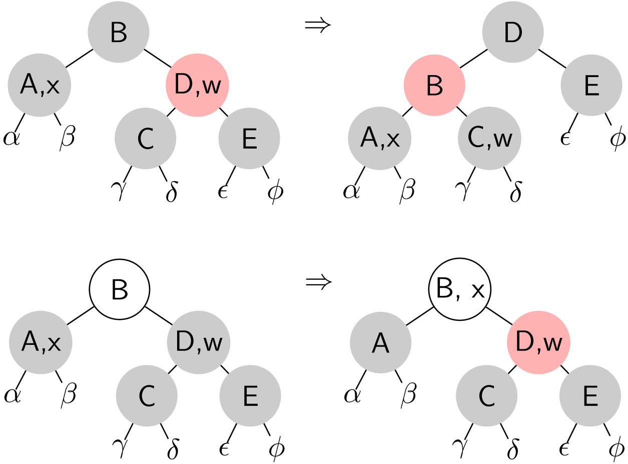
while $color[p[z]] = RED$
    if $p[z] = left[p[p[z]]]$
        $y \leftarrow right[p[p[z]]]$
        if $color[y] = RED$
            $color[p[z]] \leftarrow color[y] \leftarrow BLACK$
            $color[p[p[z]]] \leftarrow RED$, $z \leftarrow p[p[z]]$
        else if $z = right[p[z]]$
                $z = p[z]$
                $\textsc{Left-Rotate}(T, z)$
            $color[p[z]] \leftarrow BLACK$, $color[p[p[z]]] \leftarrow RED$
            $\textsc{Right-Rotate}(T, p[p[z]])$
    else (то же с заменой $left$ на $right$ и наоборот)
$color[root[T]] \leftarrow BLACK$

Сбалансированные деревья поиска

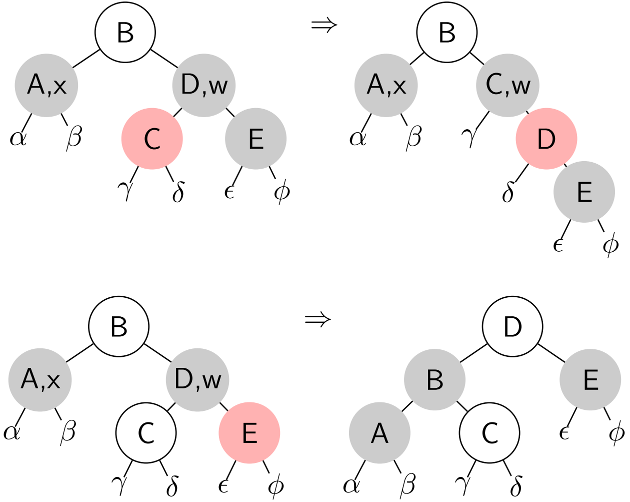
Удаление. Общая идея

- Выполняется удаление узла z из дерева T (как из обычного дерево поиска)
- Если у узла z есть оба сына, то находится узел y — следующий по порядку, у которого нет одного из сыновей
- Существующего сына узла y (или z) назовем x
- Если удалялся черный узел, то оказалось нарушено свойство 5
- Вызывается процедура, восстанавливающая инварианты красно-черного дерева; до тех пор, пока x не корень, и пока он черный (если x — красный, то можно перекрасить его, и дерево станет корректным), выполнять повороты и изменять окраску вершин

Удаление. Иллюстрация



Удаление. Иллюстрация



Удаление. Алгоритм

RB-Delete-Fixup(T, x)

while $x \neq root[T]$ и $color[x] = BLACK$
    if $x = left[p[x]]$
        $w \leftarrow right[p[x]]$
        if $color[w] = RED$
            $color[w] \leftarrow BLACK, color[p[x]] \leftarrow RED$
            Left-Rotate$(T, p[x])$
            $w \leftarrow right[p[x]]$
        if $color[left[w]] = BLACK$ и $color[right[w]] = BLACK$
            $color[w] \leftarrow RED$
            $x \leftarrow p[x]$

Удаление. Алгоритм

$\textsc{RB-Delete-Fixup}(T, x)$

while $x \neq root[T]$ и $color[x] = BLACK$
    if $x = left[p[x]]$
    else if $color[right[w]] = BLACK$
            $color[left[w]] \leftarrow BLACK, color[w] \leftarrow RED$
            $\textsc{Right-Rotate}(T, w)$
            $w \leftarrow right[p[x]]$
        $color[w] \leftarrow color[p[x]]$
        $color[p[x]] \leftarrow BLACK$
        $color[right[w]] \leftarrow BLACK$
        $\textsc{Left-Rotate}(T, p[x])$
        $x \leftarrow root[T]$
    else (то же с заменой $left$ на $right$ и наоборот)
$color[x] \leftarrow BLACK$