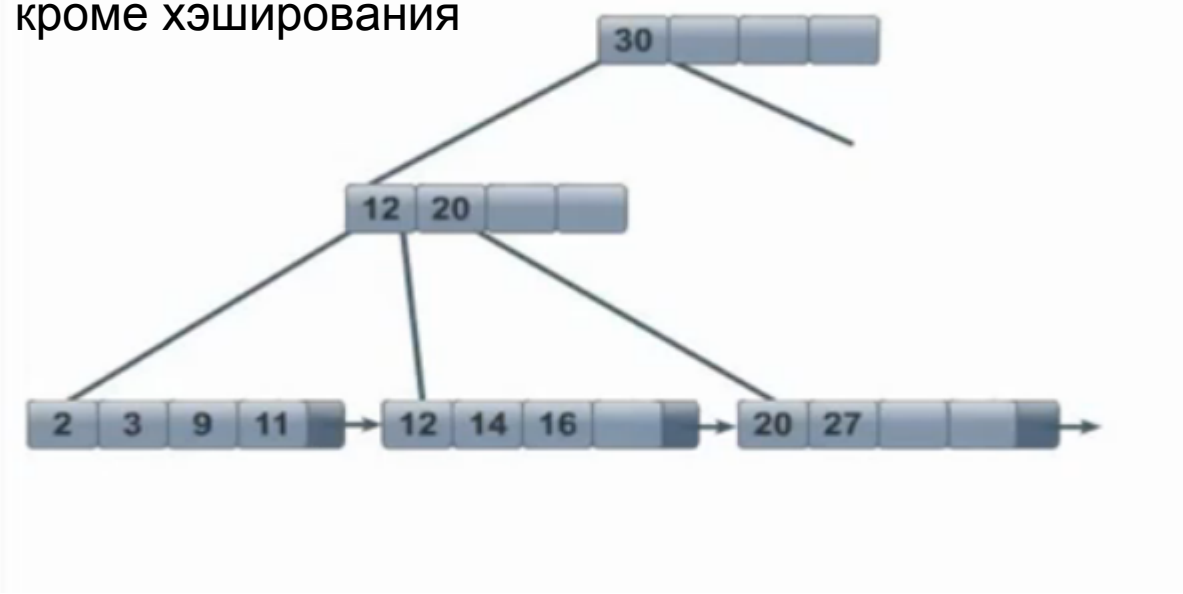
B+ tree

Работу выполнил студент группы ИВ-222
Гайдай А.В.
Дисциплина: САОД

B+ tree

B+ дерево — сбалансированное дерево поиска. Является модификацией B-дерева, истинные значения ключей которого содержатся только в листьях, а во внутренних узлах — ключи-разделители, содержащие диапазон изменения ключей для поддеревьев

Эти деревья были впервые предложены Bayer'ом и McCreight'ом в 1972 году и всего за несколько лет сменил почти все крупные методы доступа к файлам, кроме хэширования



Характеристика B+tree

1. B^+деревья сохраняют записи (или указатели на реальные записи) только на конечных узлах(листах)

2. Количество ключей в каждом узле (кроме корня) должно быть от *b* до 2*b* (*b* - порядок дерева)

3. Ключи хранятся в порядке убывания (т.е. отсортированы в лексикографическом порядке)

4. Конечные узлы из B^+-дерева связаны друг с другом, и тем самым формируют связанный список

Утверждение о высоте B+дерева

Утверждение.

Высота B+дерева с $n \geq 1$ ключами и минимальной степенью $b \geq 2$ в худшем случае не превышает

$$\log_b((n + 1) / 2).$$

Доказательство.

- Обозначим высоту B+дерева через h.
- Рассмотрим максимально высокое B+дерево: в корне такого дерева хранится 1 ключ, а в остальных узлах по $b - 1$ ключу (минимально возможное количество)
- На уровне 0 размещен один узел (корень) с 1 ключом
- На уровне 1: 2 узла (у каждого по $b - 1$ ключу)
- На уровне 2: $2b$ узлов
- На уровне 3: $2b^2$ узлов
- …
- На уровне h: $2b^{h-1}$ узлов

Утверждение о высоте B+дерева

- Тогда, общее количество ключей есть:

$$n = 1 + (b - 1)(2 + 2b + 2b^2 + 2b^3 + \cdots + 2b^{h-1}) =$$

$$= 1 + 2(b - 1)(1 + b + b^2 + \cdots + b^{h-1})$$

- Сумма h первых членов геометрической прогрессии:

$$S_h = d_1(q^h - 1)/(q - 1)$$

- Следовательно,

$$n = 1 + 2(b - 1)\,(b^h - 1)/(b - 1) = 2b^h - 1$$

$$n + 1 = 2b^h$$

$$(n + 1) / 2 = b^h$$

$$\boldsymbol{h = \log_b((n + 1) / 2)}$$

Утверждение доказано.

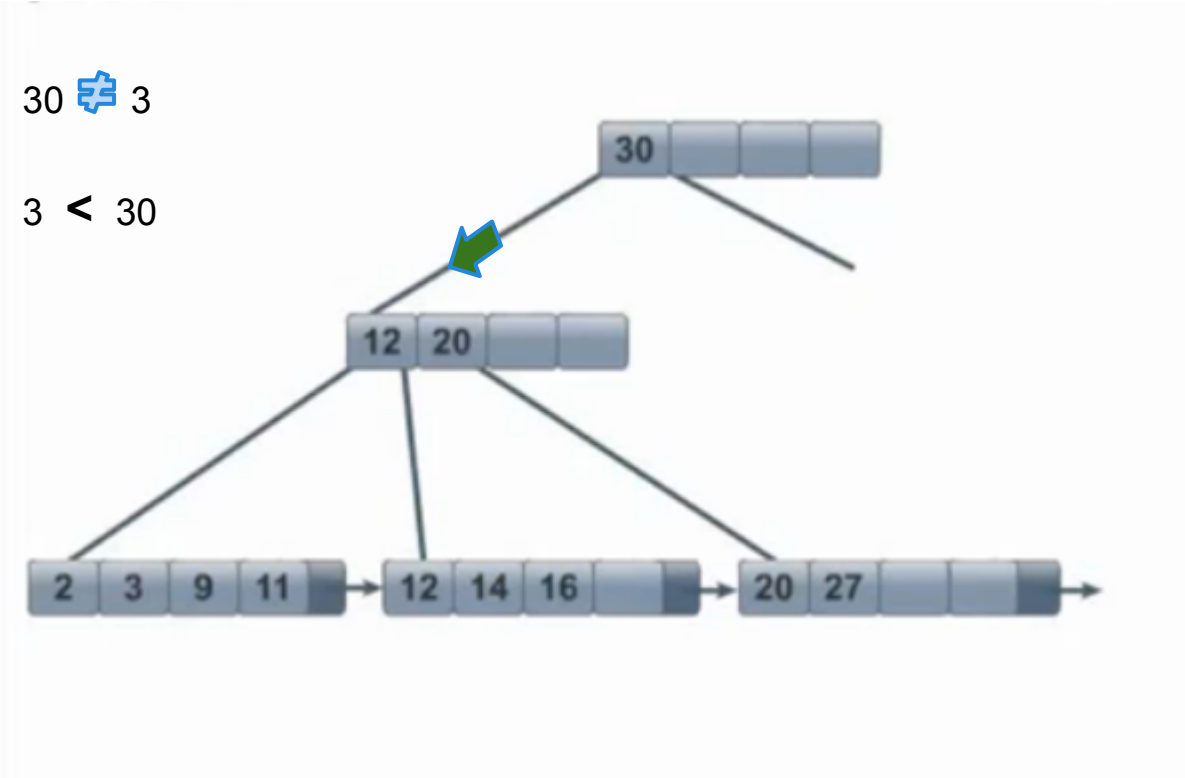
Алгоритм поиска элемента по ключу в B+дереве

Поиск в B^+-дереве является ступенчатым процессом, начиная с корневого узла:

1) Бинарный поиск ключа в текущем узле – стоит помнить, что поисковые значения сортируются. Ищем ключ K_i такой, что $K_i \leq K < K_{i1}$
2) Если текущий узел является внутренним, совершается переход на надлежащую ветвь, связанную с ключом K_i и повторяется пункт “1”
3) Если текущий узел является листом, то:

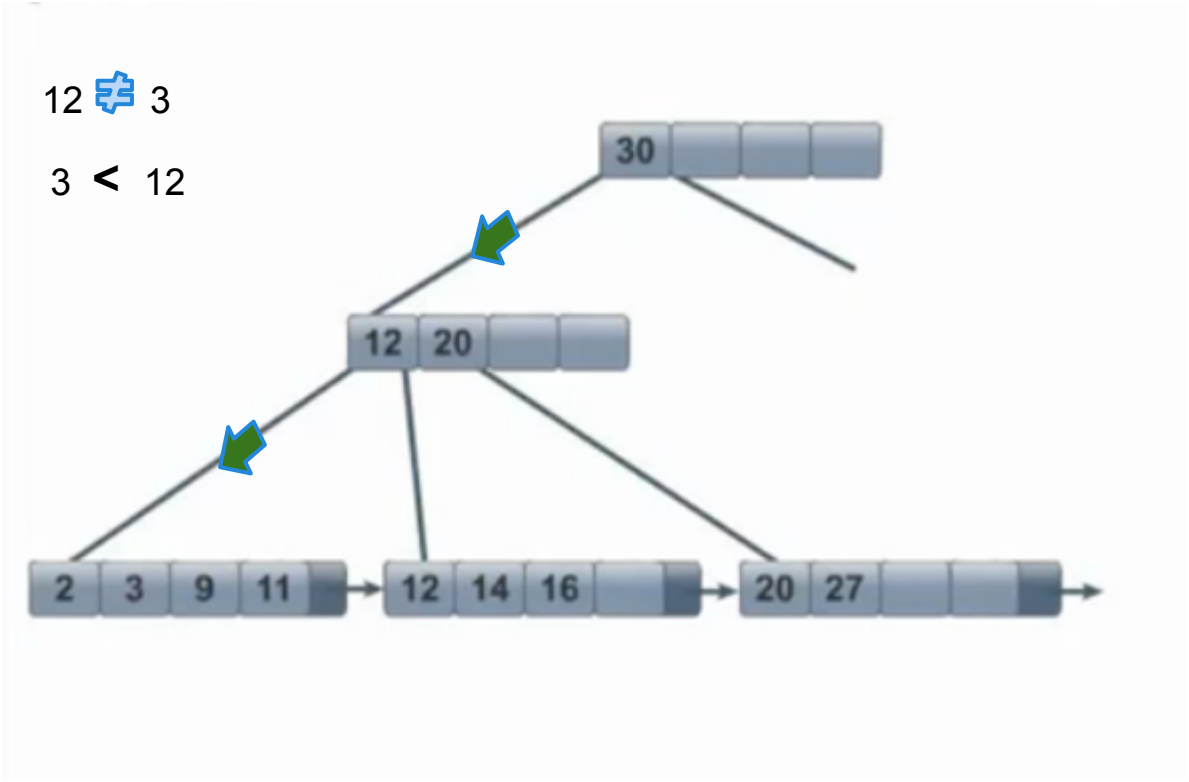
- Если $K = K_i$, то запись существует в таблице, и мы можем вернуть запись, связанную с K_i
- В противном случае, K не найден среди ключевых значений на листе, это значит, что в таблице нет ключей с нужным значением

Поиск записей, удовлетворяющих простому условию



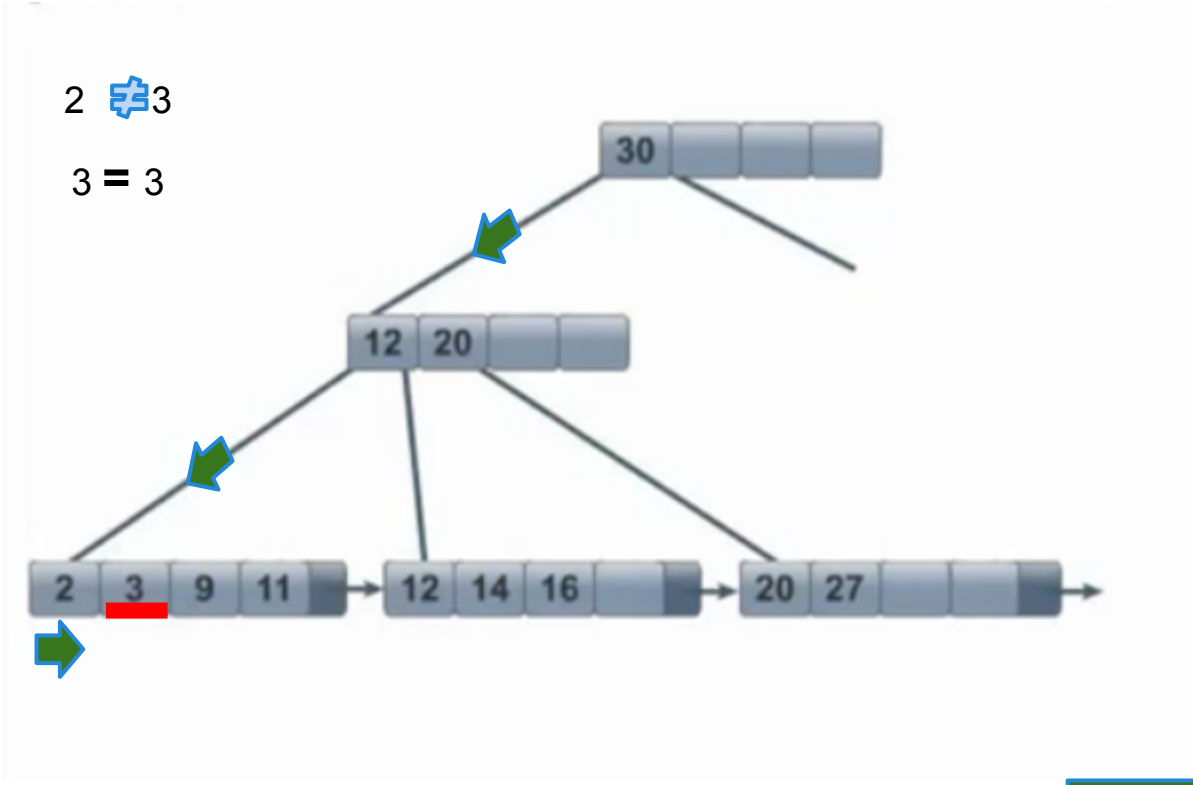
Find: 3

Искомая запись находится в листе, ищем нужный лист



Find: 3

Искомая запись находится в листе, ищем нужный лист



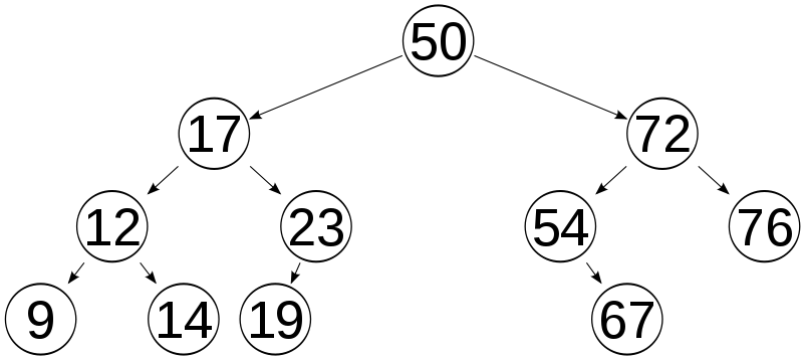
Find: 3

$T_{Find}=O(log_{b/2}n)$

b - порядок дерева

Доказательство вычислительной сложности

Поиск элемента в бинарном сбалансированном дереве имеет трудоёмкость равную $O(log_2 n)$



Основное отличие процесса поиска в B+дереве в том, что внутренний узел может иметь от $b/2$ до b дочерних узлов (а не ограничивается двумя)

Отсюда следует, что в худшем случае вычислительная сложность поиска в B+дереве будет равняться $T_{Find} = O\ (log_{b/2} n)$.

Алгоритм вставки в B^+-дерево

Шаги для вставки в B^+-дерева:

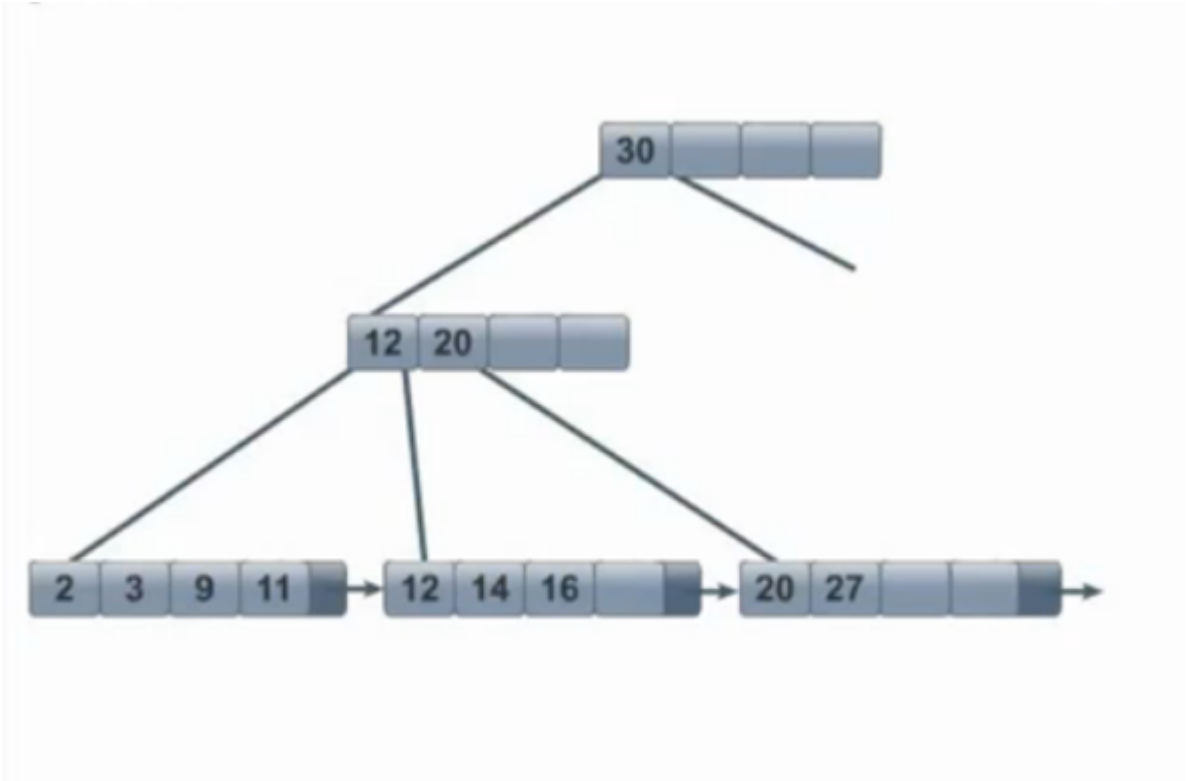
1) Поиск позиции для добавления нового элемента.
2) Вставка в найденный лист *L*

- Если *L* не полон, то запись просто создается
- Если *L* полон, то в B^+-дерево вводится новый листовой узел L_{new}, как правый брат *L*. Ключи в *L*, вместе с новым элементом, распределяются равномерно среди *L* и L_{new}. L_{new} вставляется в связанный список конечных узлов справа от L. Теперь мы должны связать L_{new} с деревом при этом L_{new} должен быть родным братом для *L*. Наименьший ключ из L_{new} будет **скопирован** и вставлен в родитель *L* - который также будет родителем L_{new}. Весь этот шаг известен как **разделение листового узла**

Алгоритм вставки в B^+-дерево

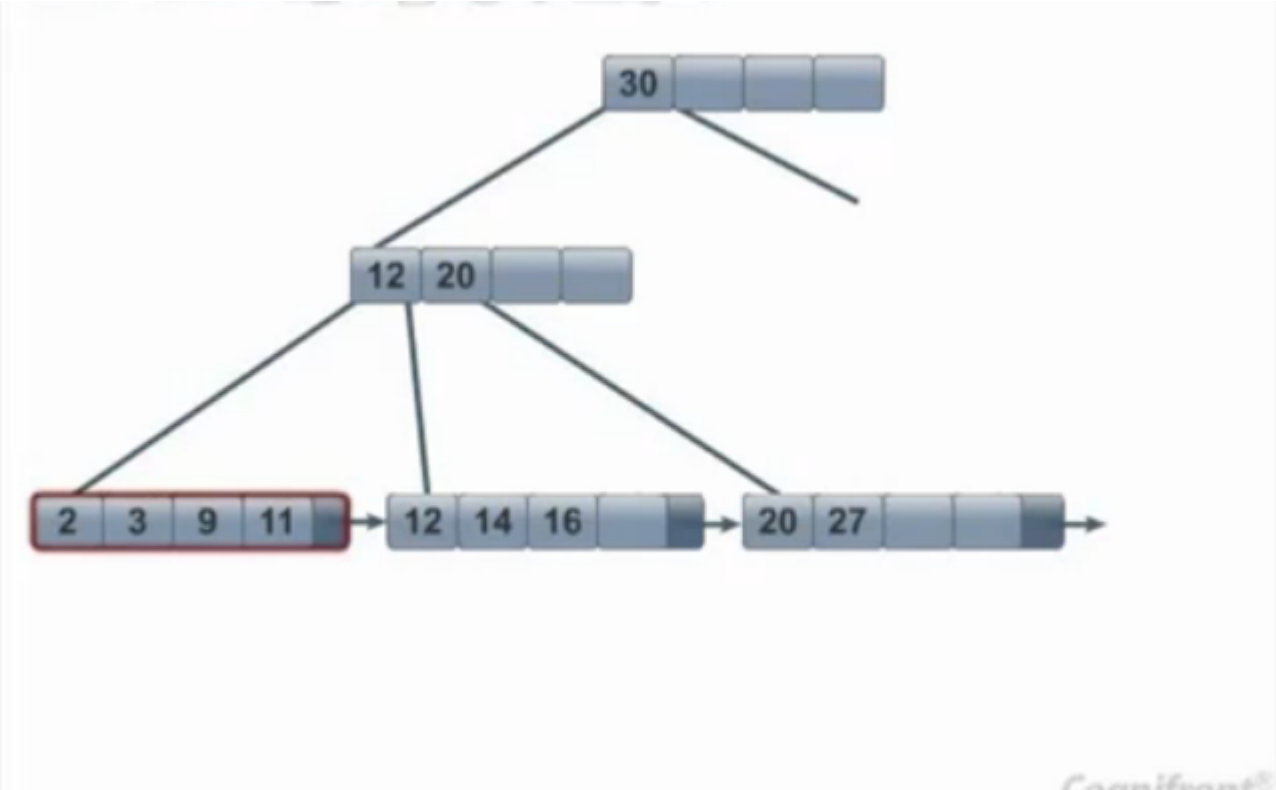
- Если родительский узел *P* заполнен, то он разделяется в свою очередь. Однако это **разделение внутреннего узла** немного отличается. Ключи узла *P* и новая запись должны быть равномерно распределены. Новой узел должен быть введен в качестве брата *P* при этом, **средний ключ перемещается выше узла(не копируется!)**. Это расщепление узлов может продолжаться вверх по дереву до корня
- Когда ключ добавляется в полный корень, то корень расщепляется на два, а средний ключ становится новым корнем. Это единственный способ для B^+-дерева расти в высоту

Вставка в B^+-дерево



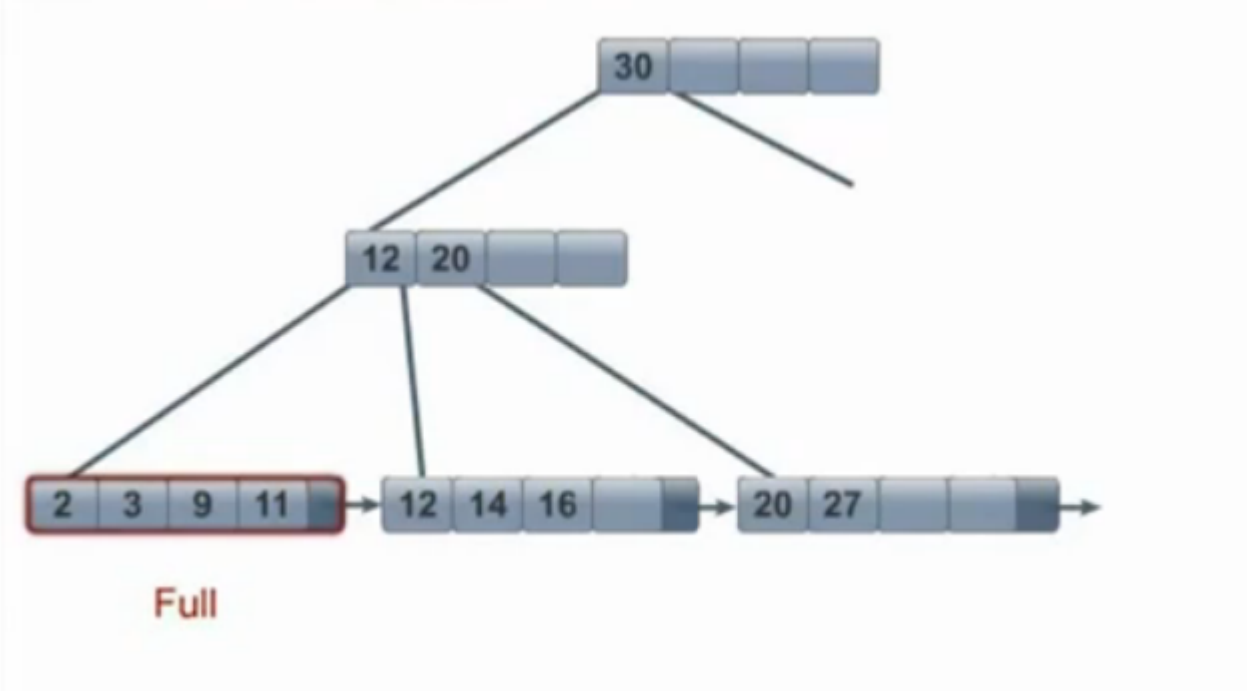
Insert: 7

Вставка в B^+-дерево



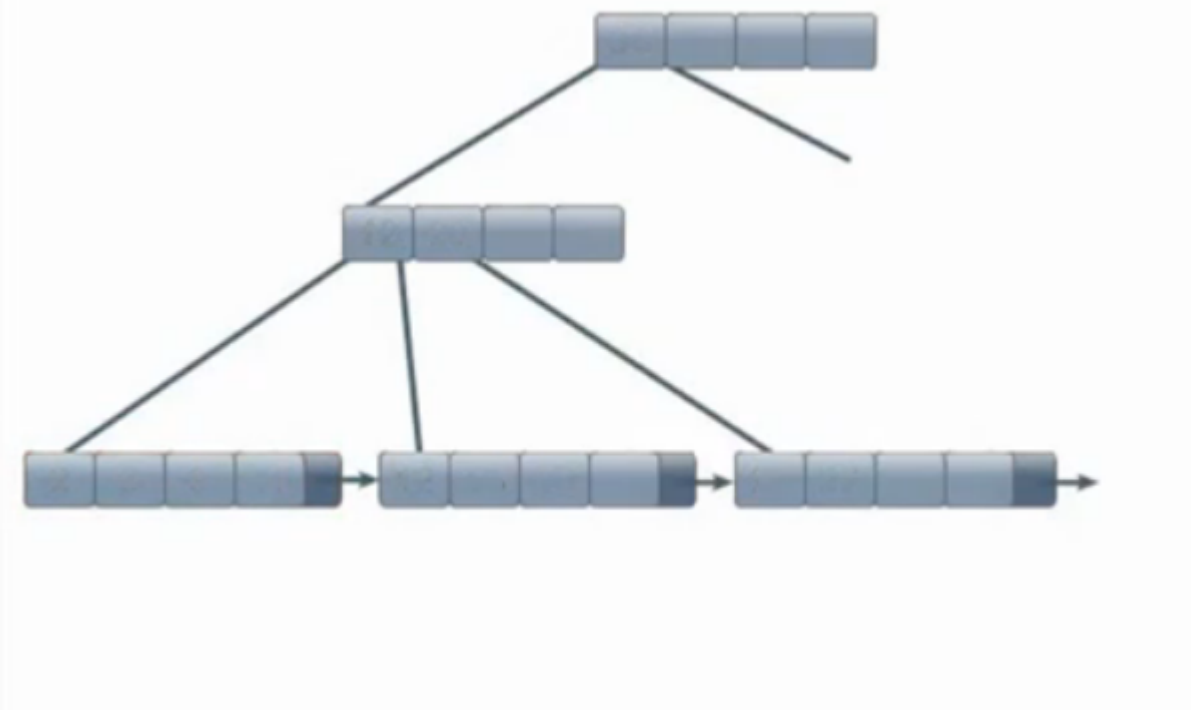
Insert: 7

Вставка в B^+-дерево



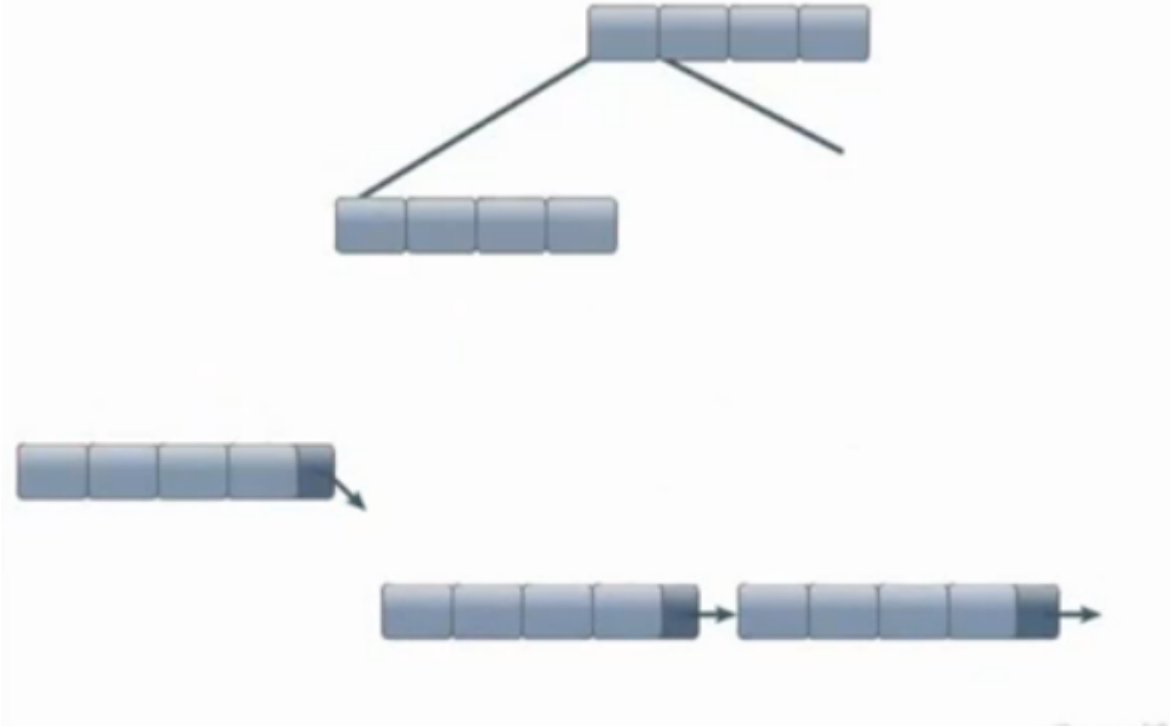
Insert: 7

Вставка в B^+-дерево



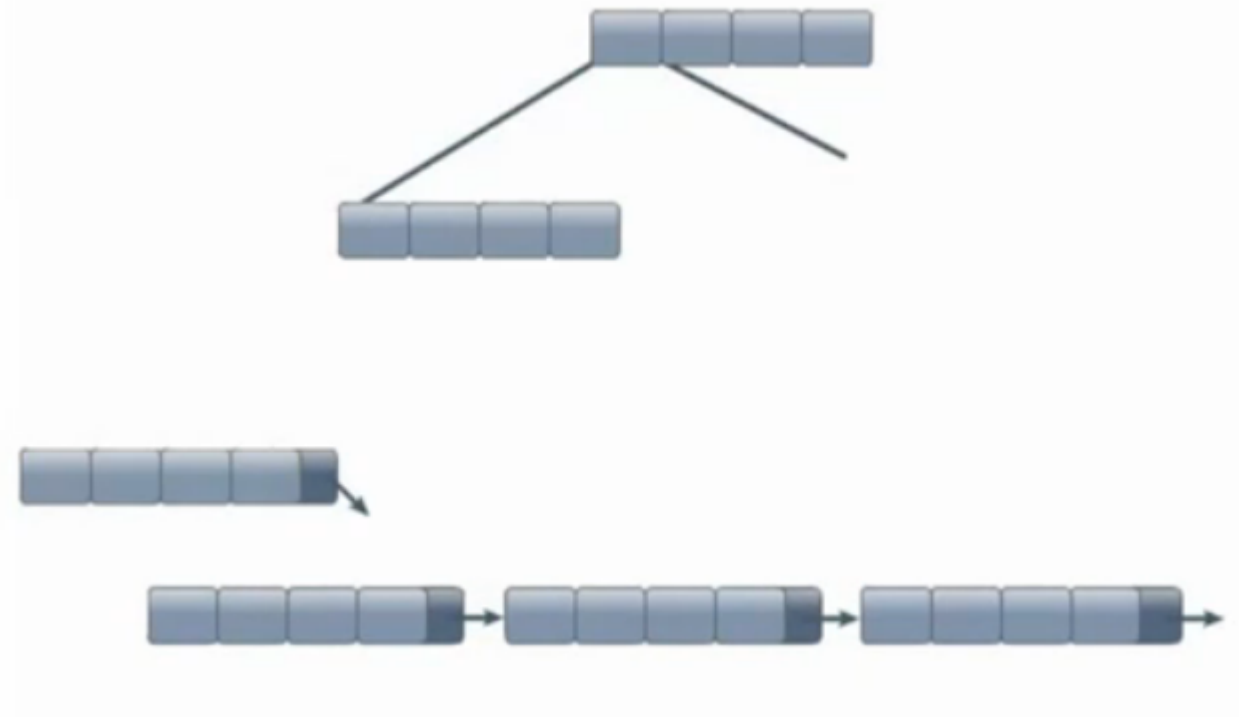
Insert: 7

Вставка в B^+-дерево



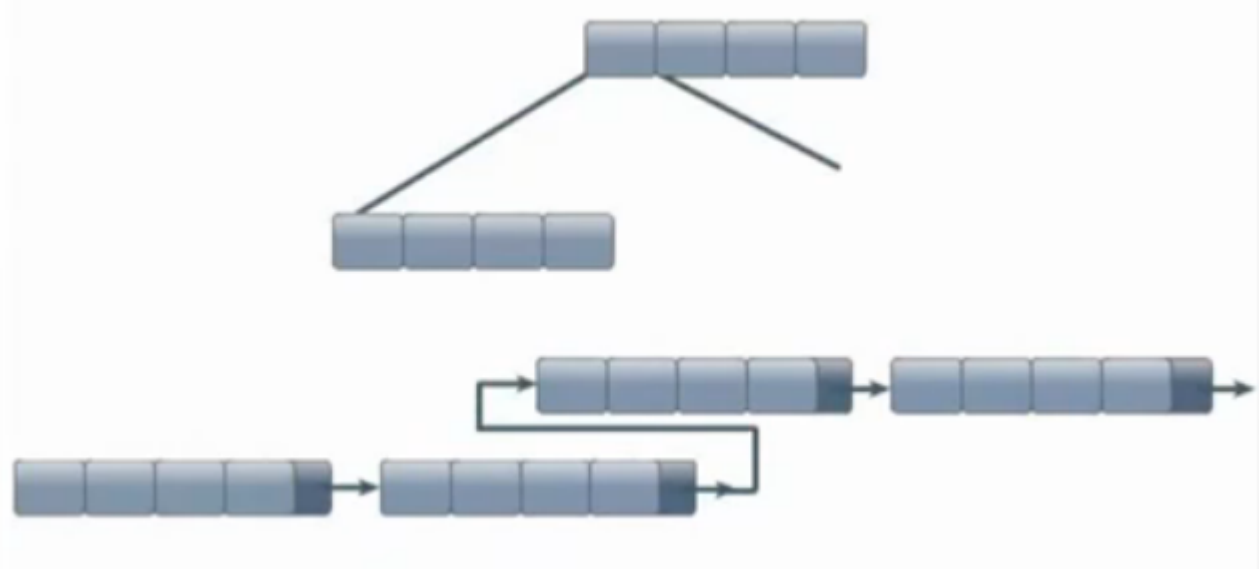
Insert: 7

Вставка в B^+-дерево



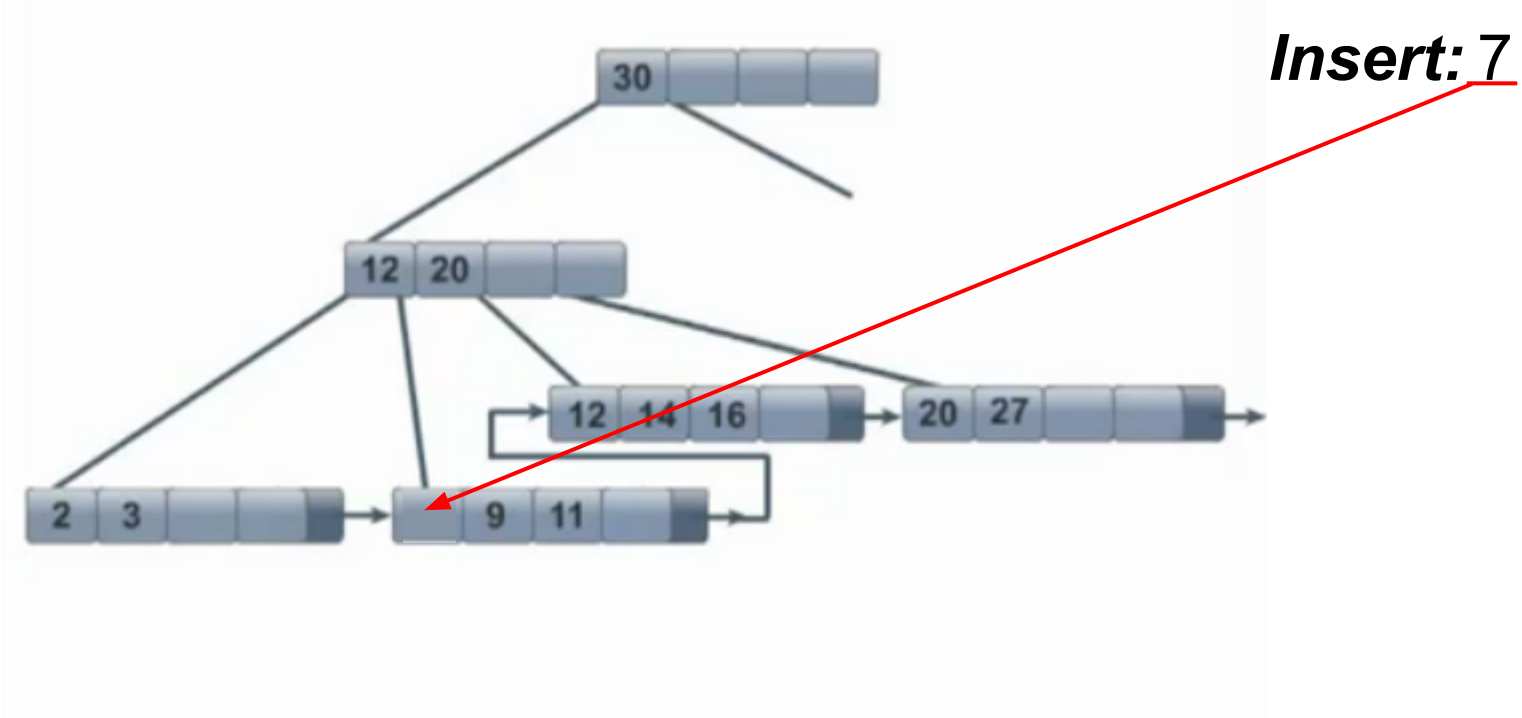
Insert: 7

Вставка в B^+-дерево

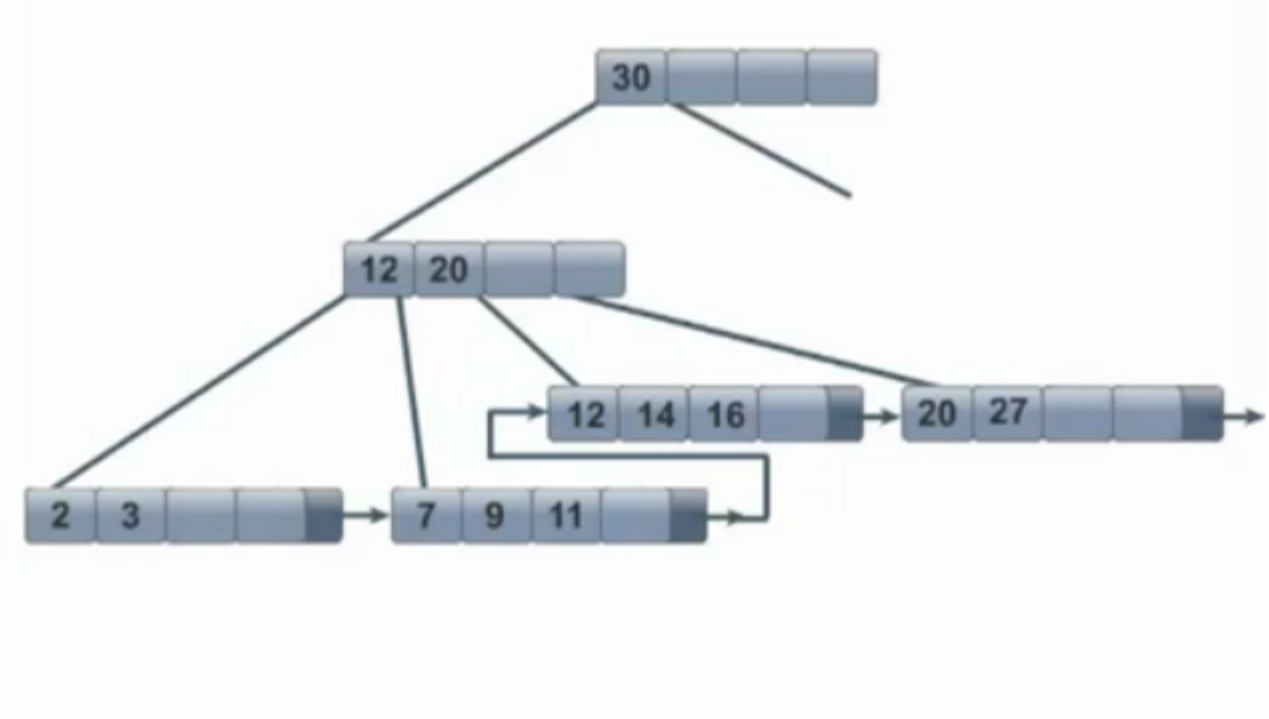


Insert: 7

Вставка в B^+-дерево

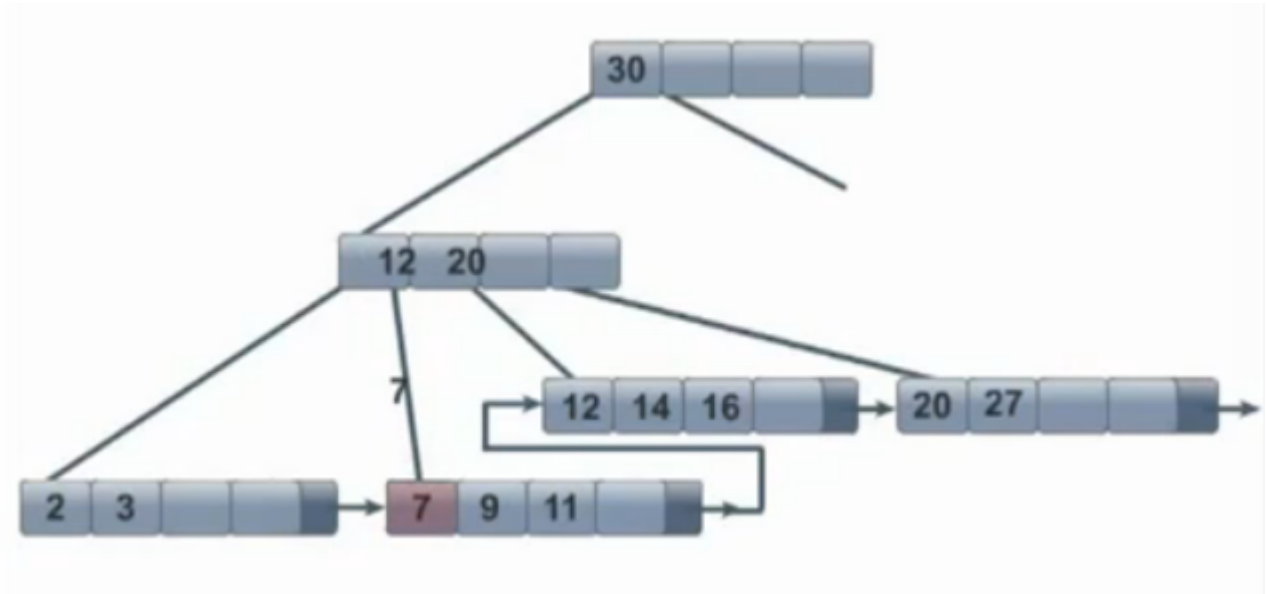


Вставка в B^+-дерево



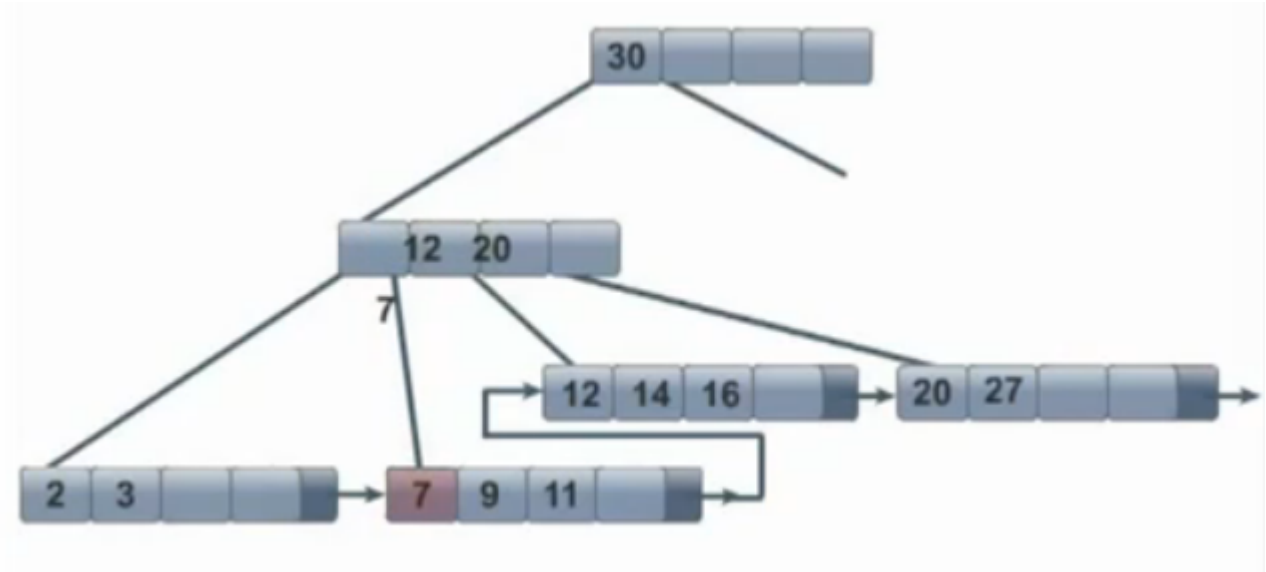
Insert: 7

Вставка в B^+-дерево



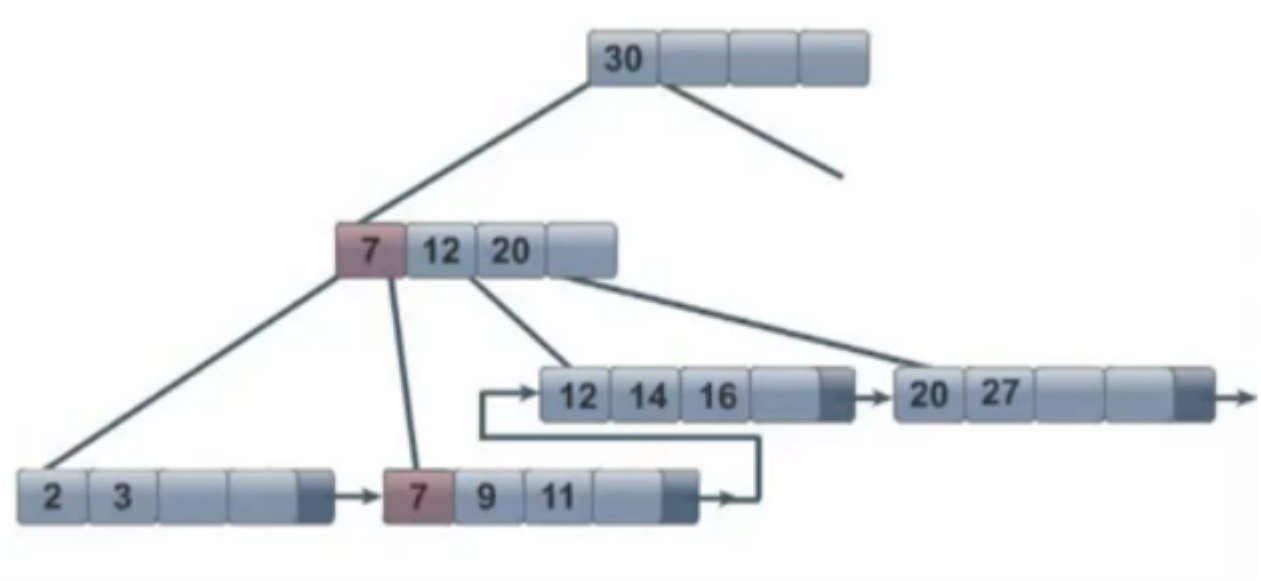
Insert: 7

Вставка в B^+-дерево



Insert: 7

Вставка в B^+-дерево



Insert: 7

$T_{Insert}=O(log_{b/2}n)$

b - порядок дерева

Доказательство вычислительной сложности

Поиск места для вставки имеет такую же вычислительную сложность как и просто процесс поиска ($O(\log_{b/2} n)$).
Любой возможный при вставке случай, при котором дерево подвергается изменению, имеет константную трудоёмкость

Следовательно $T_{Insert} = O((\log_{b/2} n) + 1) = O(\log_{b/2} n)$

Алгоритм удаления из B^+-дерево

Шаги для удаления элемента из B $^+$ -дерева:

1.) Поиск записи, которую необходимо удалить. Этот поиск завершится в листе L

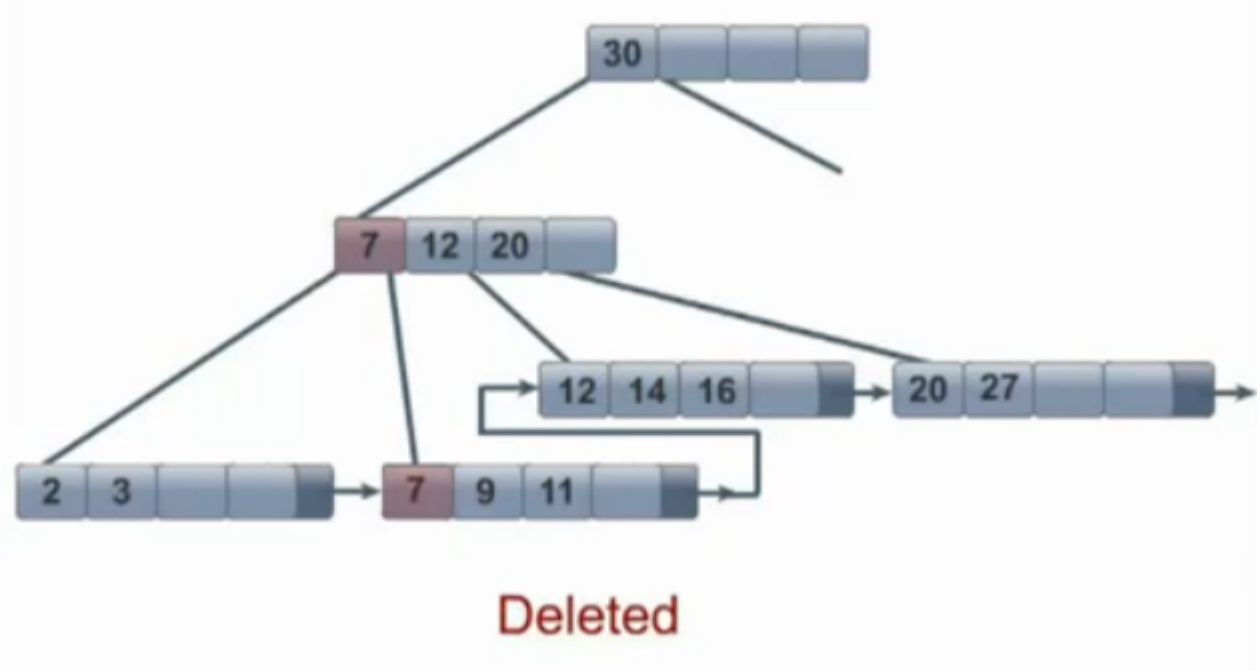
2.) Если лист L содержит более минимального числа элементов, то запись может быть безопасно удалена, без дальнейших действий

3.) Если лист содержит минимальное количество записей, то удаляемый элемент заменяется другим, таким при котором сохранится правильный порядок. Чтобы найти такие записи, мы проверяем узлы-братья L_{left} и L_{right}, один из них должен существовать

Алгоритм удаления из B^+-дерево

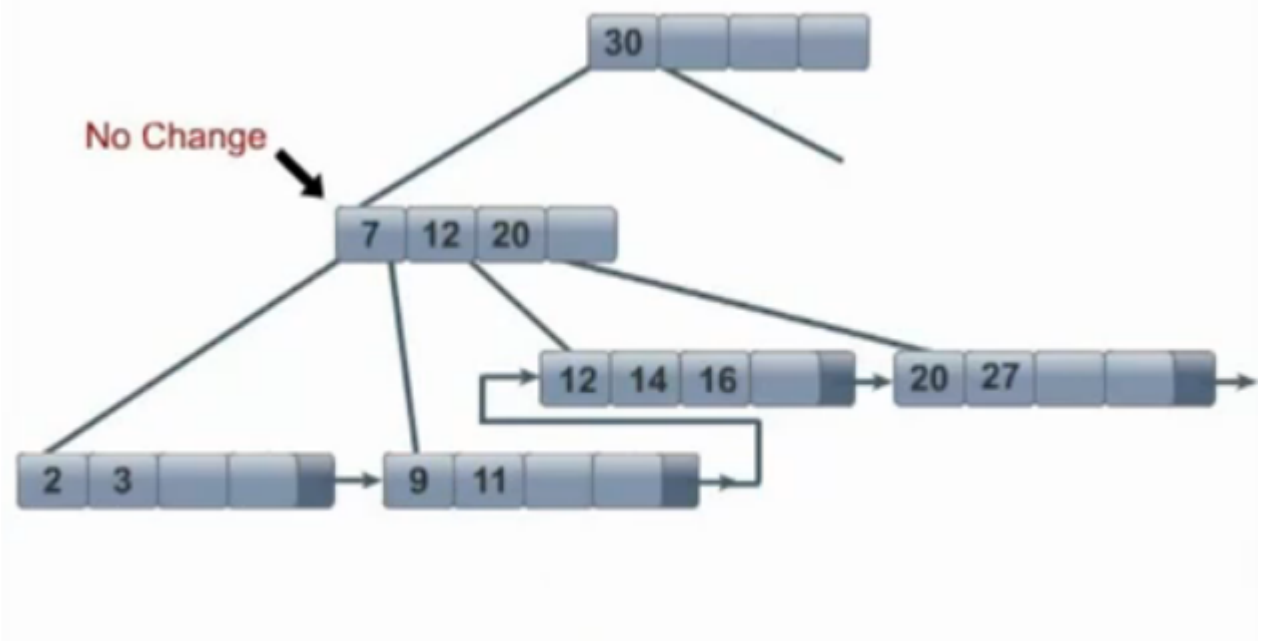
- Если один из этих конечных узлов имеет более чем минимальное количество записей, то записи передаются от этого брата, так что оба узла будут иметь приблизительно одинаковое количество элементов. Ключи из родительского узла, возможно, должны быть пересмотрены
- Если оба конечных узла L_{left} и L_{right} имеют только минимальное количество записей, то L передает свои элементы одному из братьев и удаляется из дерева. Новый лист будет содержать не более, чем максимальное число разрешенных записей
- Если последние два дочерних узла корня сливаются в один, то этот объединенный узел становится новым корневым и дерево теряет уровень

Удаление из B^+-дерево



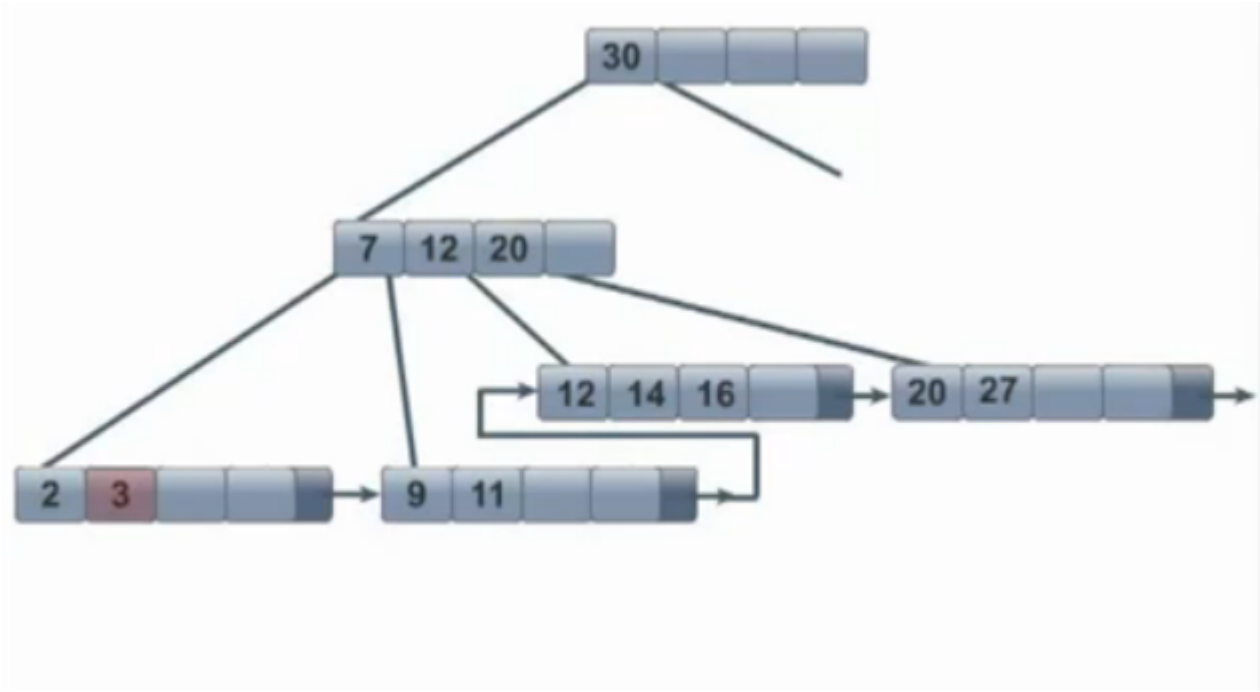
Delete: 7

Удаление из B^+-дерево



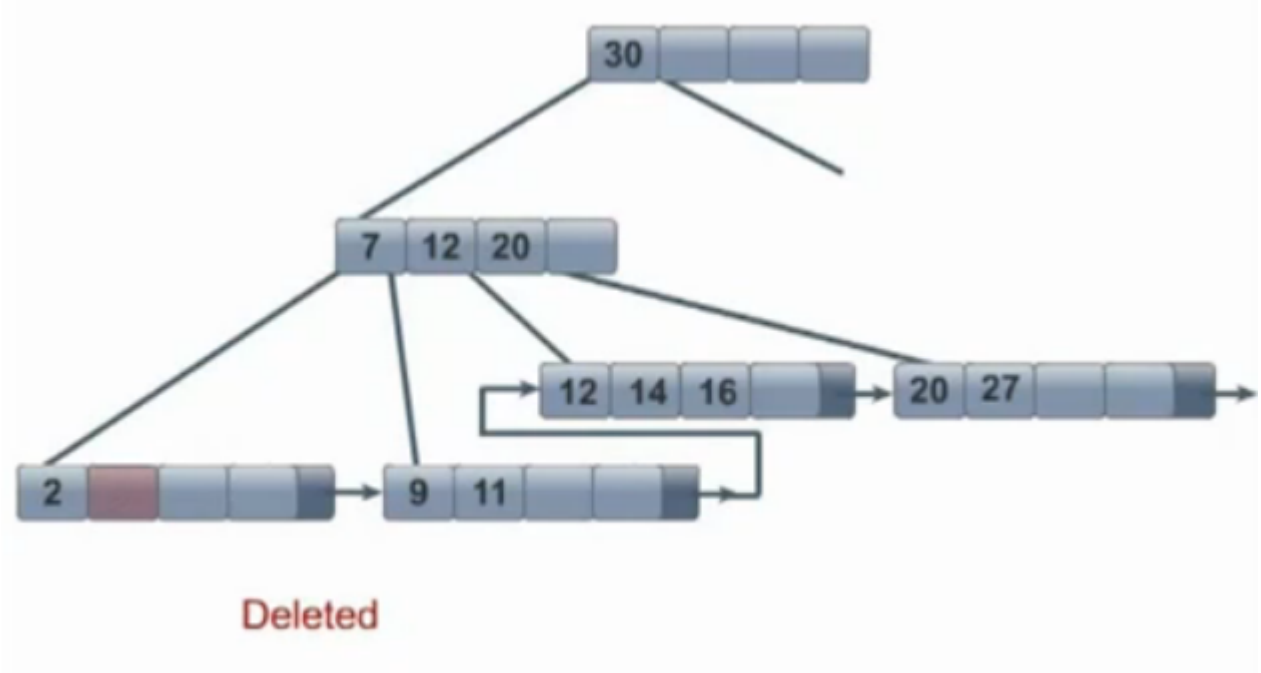
Delete: 7

Удаление из B^+-дерево



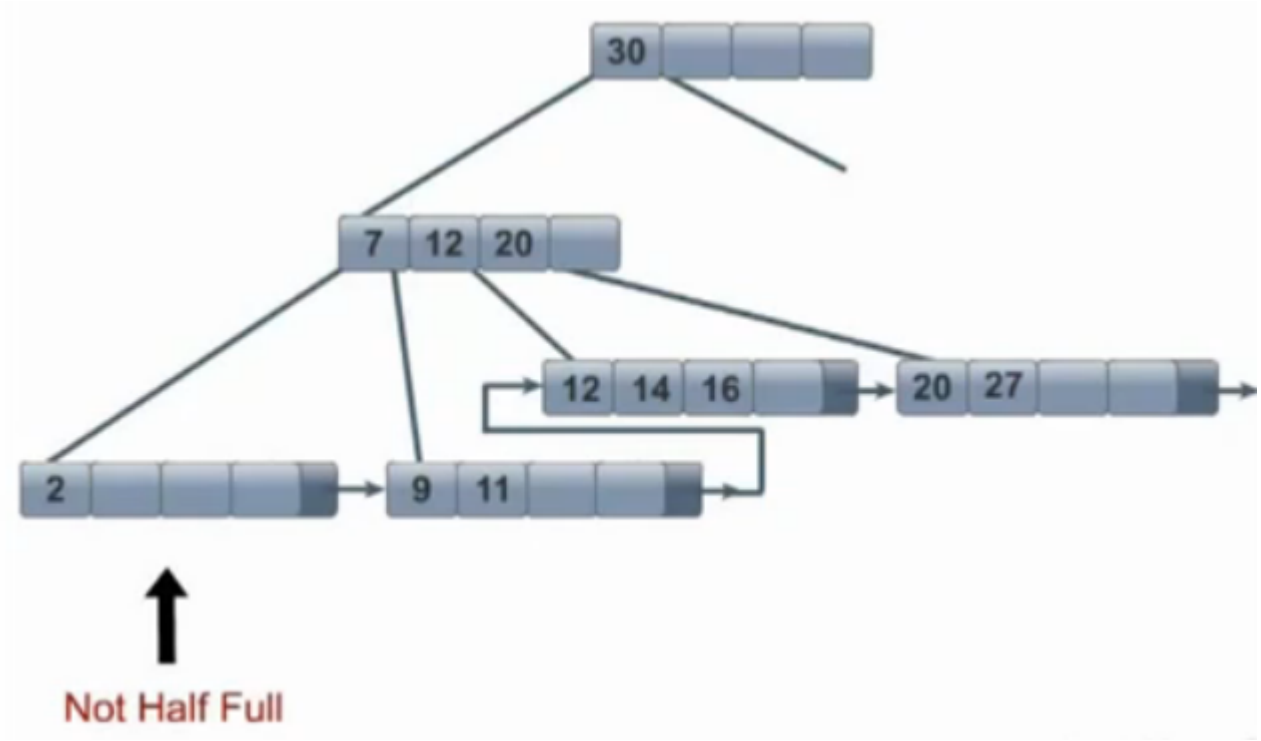
Delete: 3

Удаление из B^+-дерево



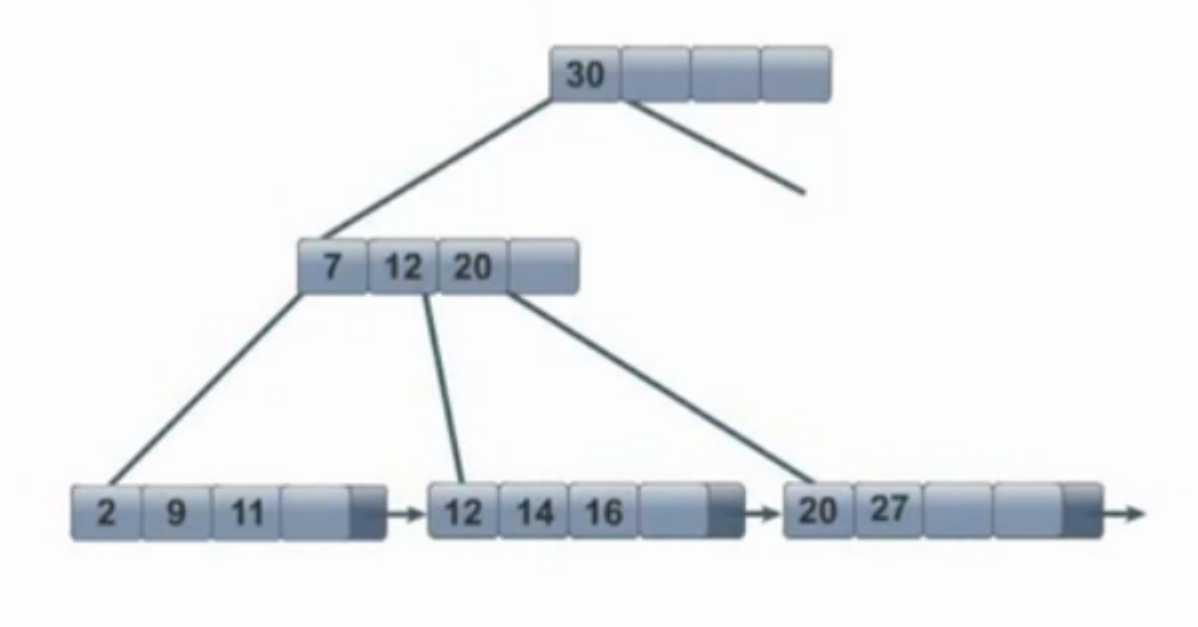
Delete: 3

Удаление из B^+-дерево



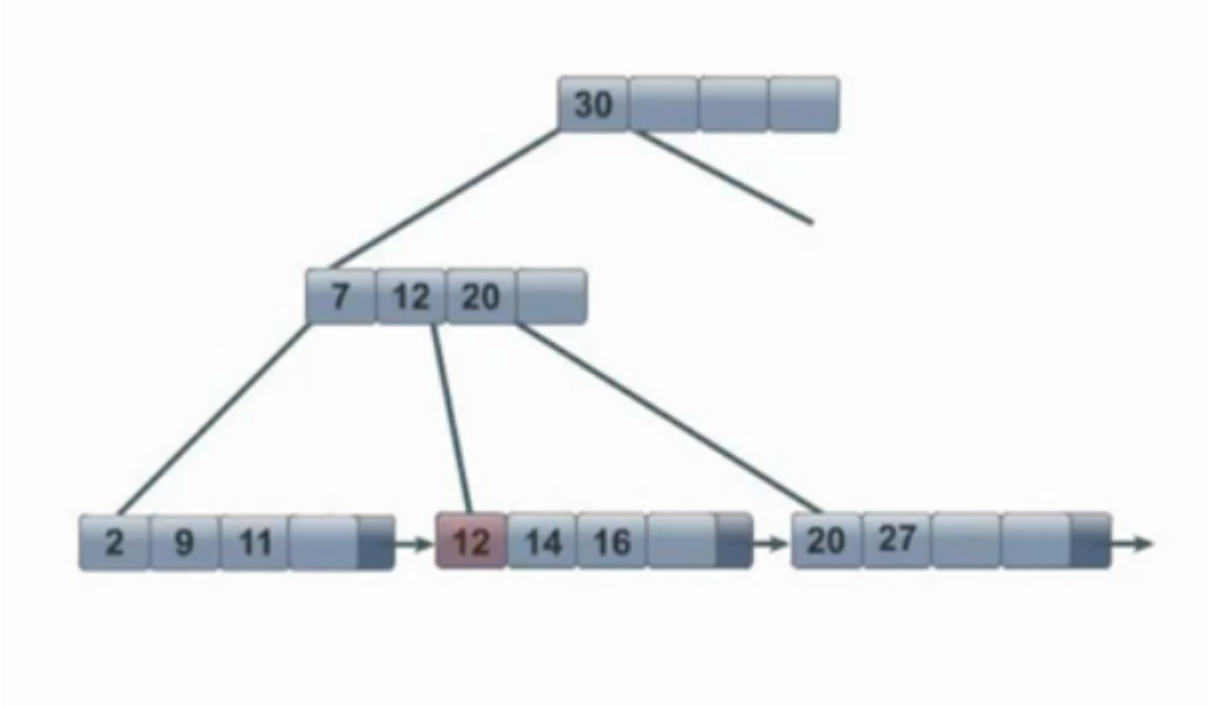
Delete: 3

Удаление из B^+-дерево



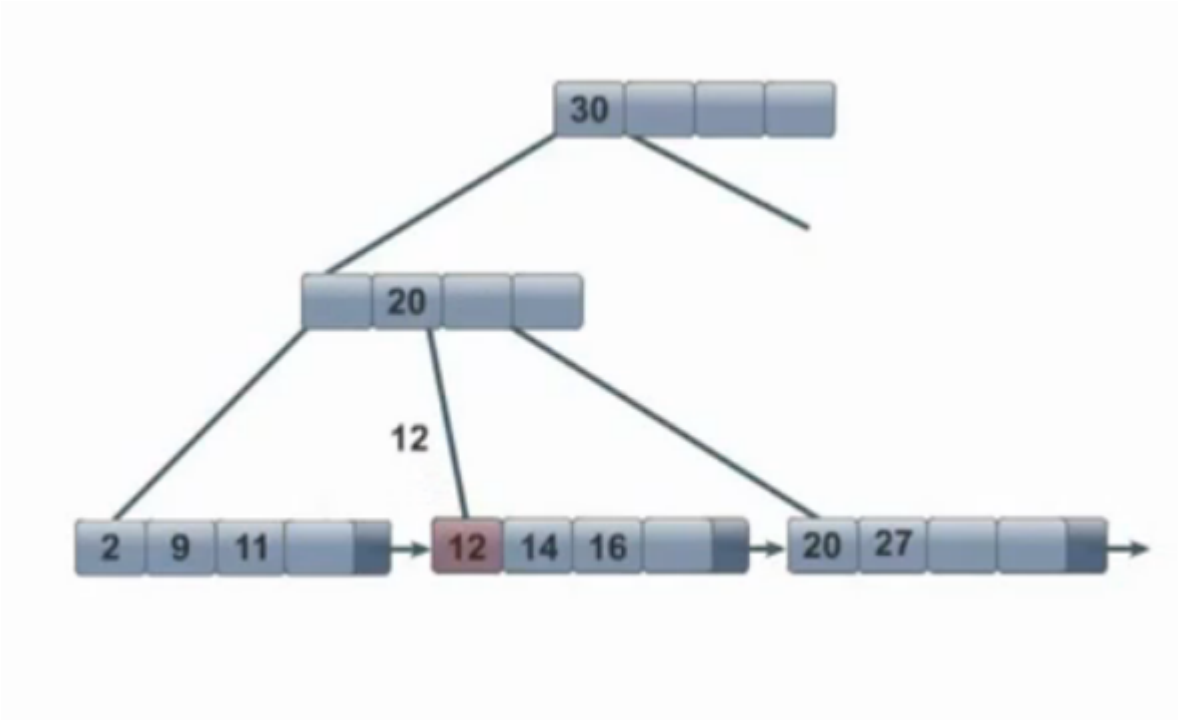
Delete: 3

Удаление из B^+-дерево



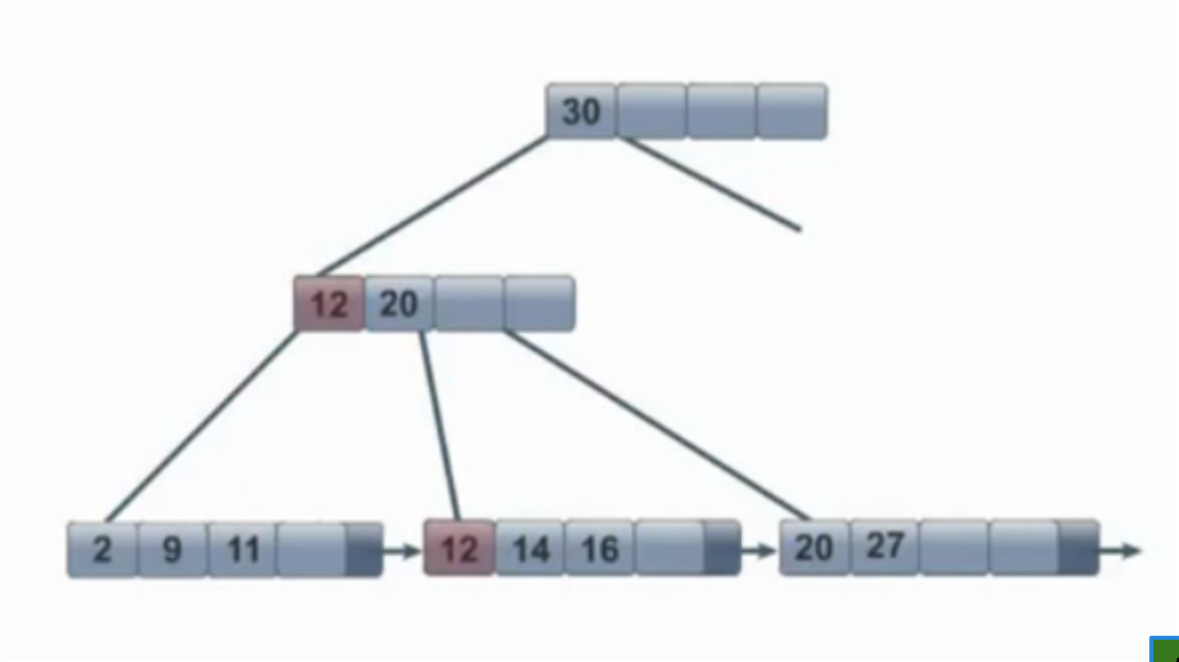
Delete: 3

Удаление из B^+-дерево



Delete: 3

Удаление из B^+-дерево



Delete: 3

$T_{Delete}=O(log_{b/2}n)$

b - порядок дерева

Доказательство вычислительной сложности

Поиск удаляемого элемента имеет такую же вычислительную сложность как и просто процесс поиска ($O(\log_{b/2} n)$).

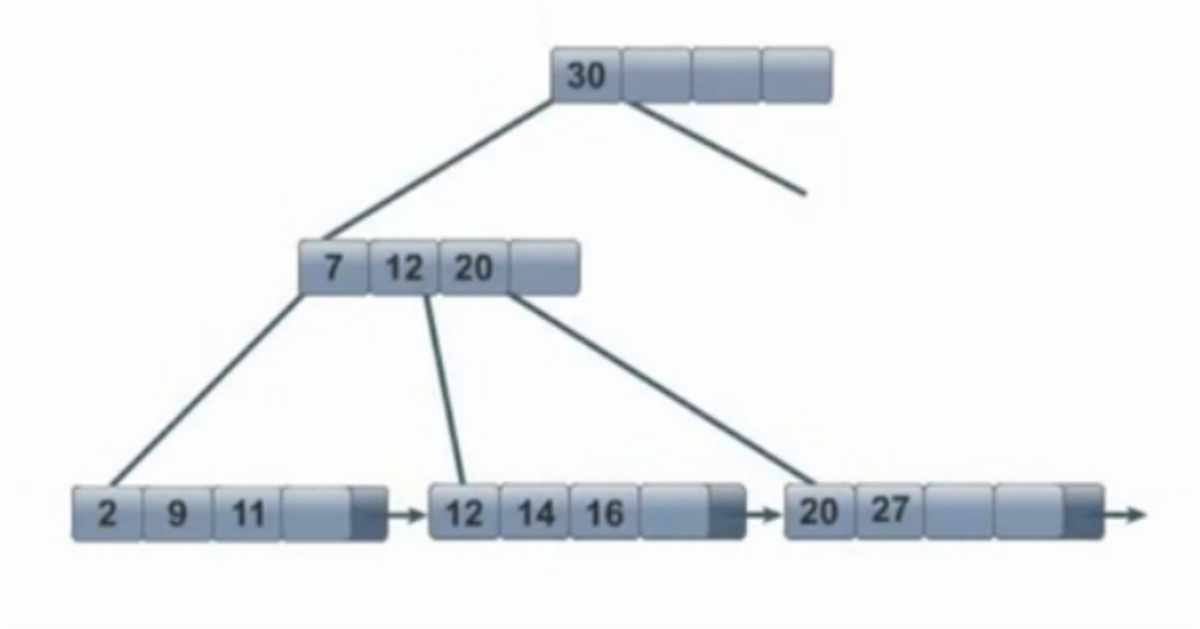
Любой возможный при удалении случай, при котором дерево подвергается изменению, имеет константную трудоёмкость.

Следовательно $T_{Delete} = O((\log_{b/2} n) + 1) = O(\log_{b/2} n)$.

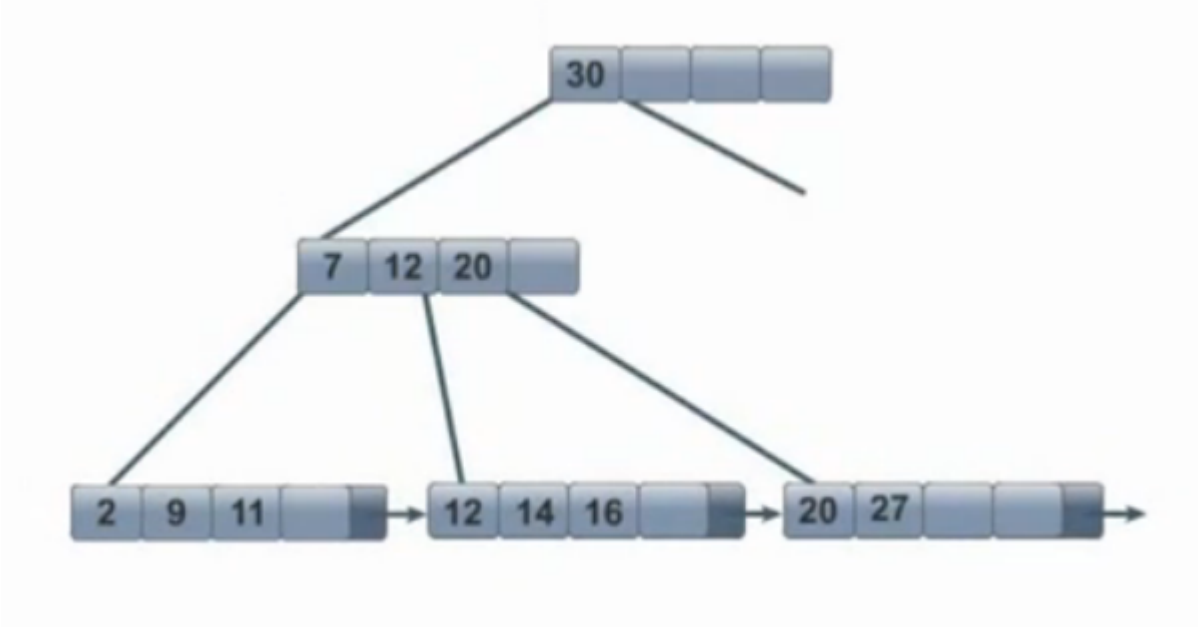
Алгоритм поиска по диапазону

1) Поиск первого элемента из диапазона в дереве

2) Проход по конечным узлам до тех пор, пока очередная запись удовлетворяет условиям диапазона или пока записи в дереве не закончатся

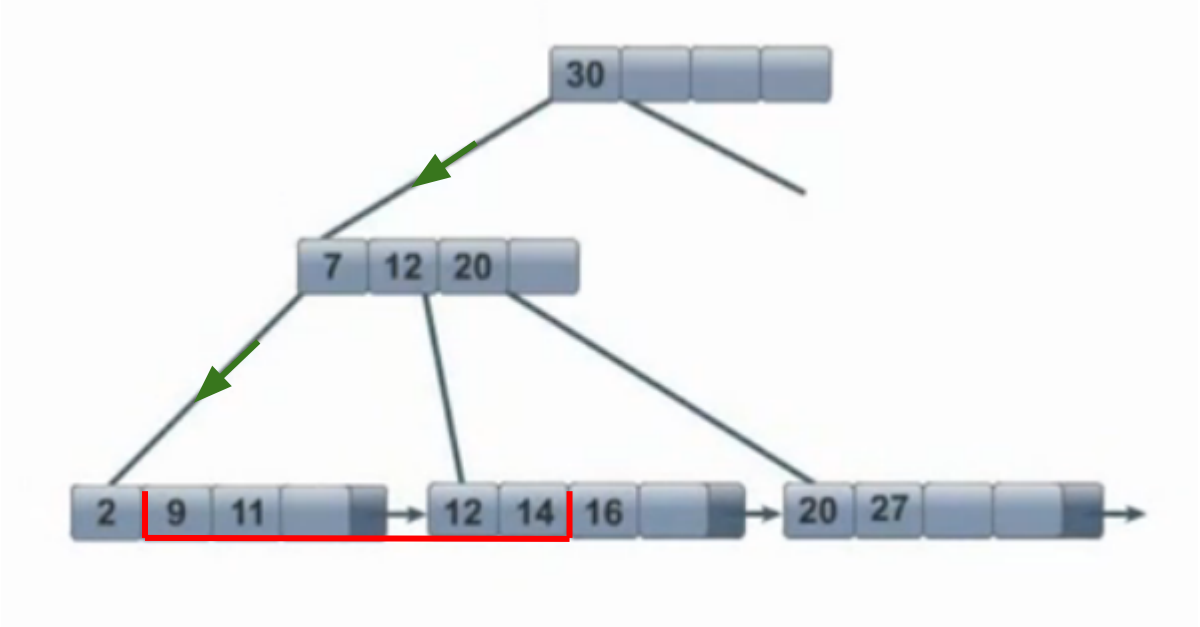


Поиск по диапазону



Find(x-y):
от 9 до 14

Поиск по диапазону



Find(x-y):
от 9 до 14

$T_{Find(x-y)} = O((\log_{b/2} n) + m)$

b - порядок дерева

m - длина связанного списка из листов

Доказательство вычислительной сложности

Поиск первого элемента из диапазона в дереве имеет трудоёмкость $O\ (log_{b/2}n)$

Чтобы найти все элементы диапазона, необходимо проходить по конечным узлам до тех пор, пока очередная запись удовлетворяет условиям диапазона или пока записи в дереве не закончатся

В худшем случае придётся пройти по всем листовым значениям, т.е. совершить m операций

Следовательно $T_{Find(x-y)} = O\ ((log_{b/2}n\ ) + m)$

Вычислительная сложность основных операций

Операция	Вычислительная сложность
Поиск	$T_{Find} = O(\log_b n)$
Вставка	$T_{Insert} = O(\log_b n)$
Удаление	$T_{Delete} = O(\log_b n)$
Поиск по диапазону	$T_{Find(x-y)} = O((\log_b n) + m)$

СПАСИБО ЗА ВНИМАНИЕ!