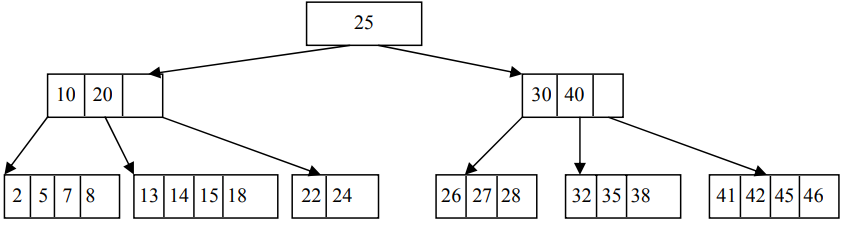
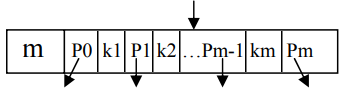
**8 Внешняя сортировка (часть 2). B-деревья**

**Пример 8.1**.

Рассмотрим пример B-дерева **порядка 2**. Все страницы кроме корневой могут содержать от 2 до 4 элементов, от 3 до 5 потомков [1].



В общем виде страница B-дерева выглядит следующим образом:



В узлах располагается несколько ключей в порядке возрастания слева направо.

Дочерние узлы содержат ключи, которые находятся в промежутке между значениями родительского узла:

Здесь

– m – число элементов на странице,

– k1, …km – элементы, расположенные на странице,

– P0 – указатель на страницу-потомка с элементами, меньшими k1

– P1 - указатель на страницу-потомка с элементами, большими k1 и меньшими k2

…

– Pm – указатель на страницу-потомка с элементами, большими km

|  |  |
| --- | --- |
| **!** | На рисунках страниц элемент *m* не показывается, он используется в программном коде при реализации B-дерева |

**Пример 8.2**.

У B-дерева примера 8.. не более **2 X 2 = 4** элементов на странице. На левой странице первого уровня (следующего за корнем на нулевом уровне) есть три элемента:

1. элемент с ключом **10** – он указывает на страницу с элементами, **меньшими 10** (на соответствующей странице-листе второго уровня размещены элементы **2, 5, 7, 8**;

2. элемент с ключом **20** – он указывает на страницу с элементами, **большими 10, но меньшими 20**;

3. последний элемент указывает на страницу с элементами, **большими 20**.

**Алгоритм вставки нового элемента**

Сначала необходимо найти место для вставки. Задача сводится к поиску подходящей страницы для элемента.

Если страница не заполнена, то элемент добавляется в нее c cохранением упорядоченности элементов на странице.

Если страница заполнена, то на ней находится 2n элементов. Добавление еще одного элемента приводит к расщеплению страницы на две равные части, а средний элемент переходит на верхнюю страницу. Две страницы имеют минимальное значение элементов (n).

Элемент, перешедший наверх, в свою очередь может вызвать переполнение и расщепление страницы.

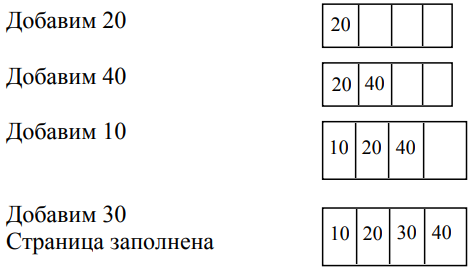
В конечном итоге некоторый элемент может быть выдавлен на самый верхний уровень, образовав новый корень.

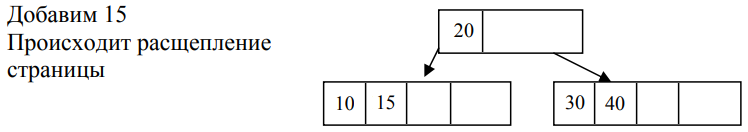
Таким образом, **B-деревья растут от листьев к корням.**

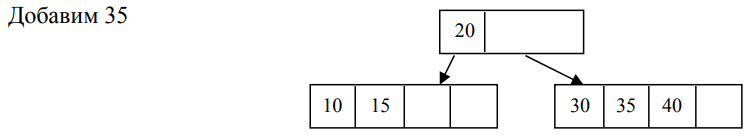
Рассмотрим процедуру последовательного добавления элементов

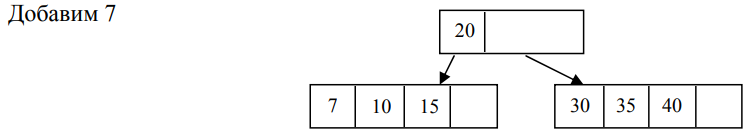
|  |
| --- |
| **20, 40, 10, 30, 15, 35, 7, 26, 18, 22, 5, 42, 13, 46, 27, 8, 32, 38, 24, 45, 25** |

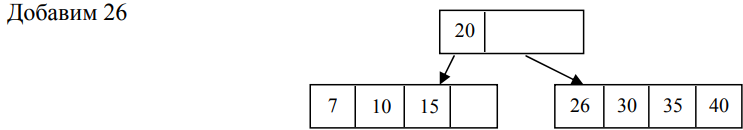
в B-дерево **порядка 2**:

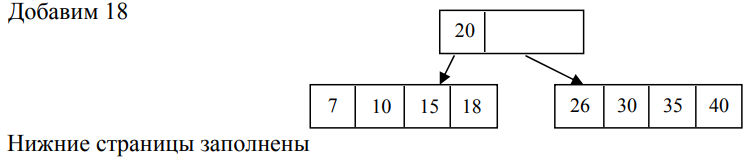


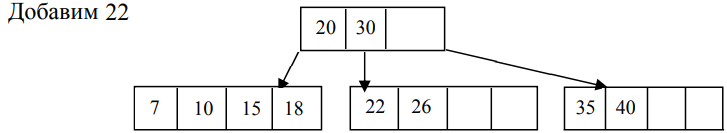


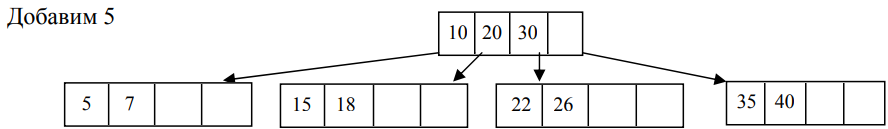


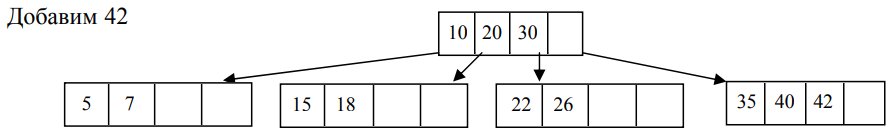


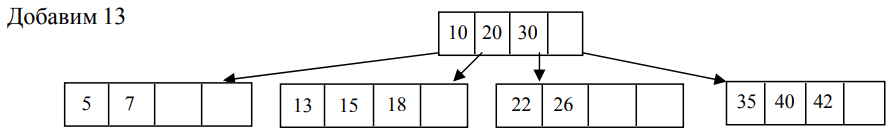


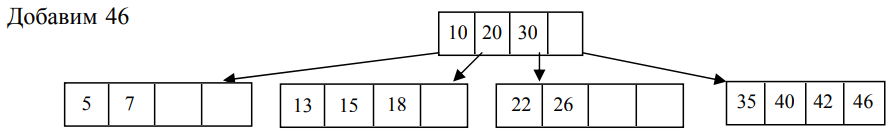


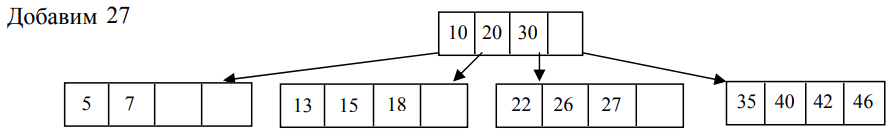


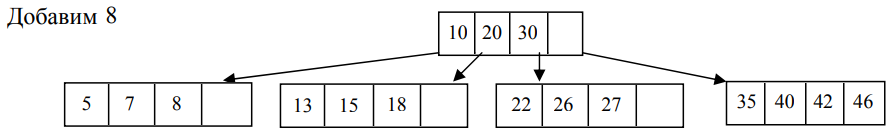


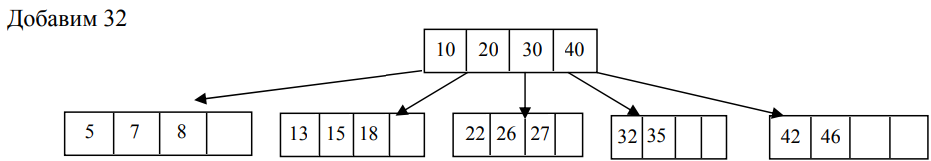


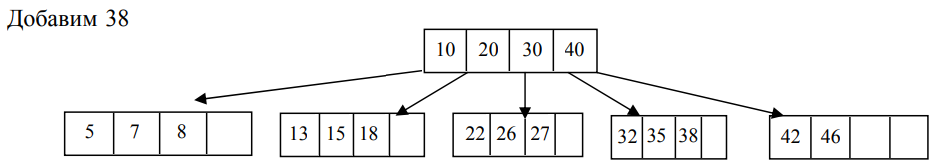


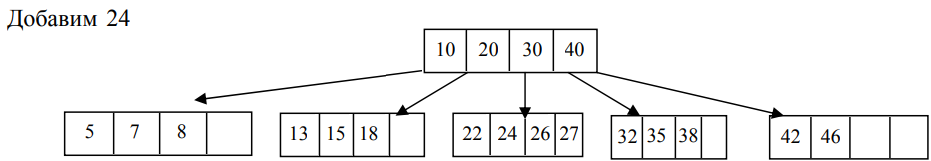


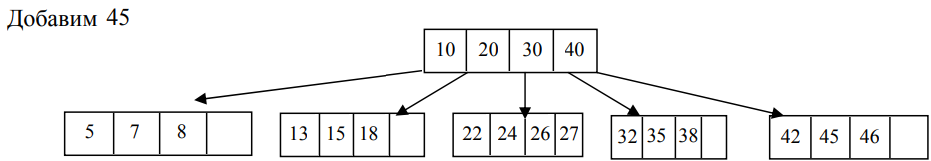


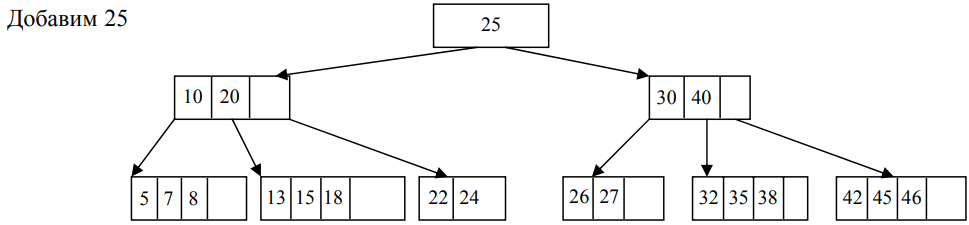












**Операция вставки элемента в B-дерево (создание дерева с упорядоченными данными)** [2]

В В-дереве новый элемент может быть добавлен только в узел-лист. Это значит, что новая пара ключ-значение всегда добавляется только к узлу-листу. Вставка происходит следующим образом:

Шаг 1: Проверить пустое ли дерево.

Шаг 2: Если дерево пустое, создать новый узел с новым значением ключа и его принять за корневой узел.

Шаг 3: Если дерево не пустое, найти подходящий узел-лист, к которому будет добавлено новое значение, используя логику дерева двоичного поиска.

Шаг 4: Если в текущем узле-листе есть незанятая ячейка, добавить новый ключ-значение к текущему узлу-листу, следуя возрастающему порядку значений ключей внутри узла.

Шаг 5: Если текущий узел полон и не имеет свободных ячеек, разделите узел-лист, отправив среднее значение родительскому узлу. Повторяйте шаг, пока отправляемое значение не будет зафиксировано в узле.

Шаг 6: Если разделение происходит с корнем дерева, тогда среднее значение становится новым корнем дерева и высота дерева увеличивается на единицу.

**Пример 8.3**:

Создадим B-дерево **порядка 3**, добавляя в него **числа от 1 до 10**. Новый элемент показан оттенками **зеленого** цвета, уже существующие – **лилового**.

Добавим 1:

Поскольку «1» — это первый элемент дерева – он вставляется в новый узел и этот узел становится корнем дерева.



Добавим 2:

Элемент «2» добавляется к существующему узлу-листу. Сейчас у нас всего один узел, следовательно, он является и корнем и листом одновременно. В этом листе имеется пустая ячейка. Тогда «2» встает в эту пустую ячейку.



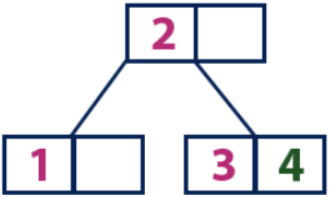
Добавим 3:

Элемент «3» добавляется к существующему узлу-листу. Сейчас у нас только один узел, который одновременно является и корнем и листом. У этого листа нет пустой ячейки. Поэтому мы разделяем этот узел, отправляя среднее значение (2) в родительский узел. Однако у текущего узла родительского узла нет. Поэтому среднее значение становится корневым узлом дерева.



Добавим 4:

Элемент «4» больше корневого узла со значением «2», при этом корневой узел не является листом. Поэтому мы двигаемся по правому поддереву от «2». Мы приходим к узлу-листу со значением «3», у которого имеется пустая ячейка. Таким образом, мы можем вставить элемент «4» в эту пустую ячейку.



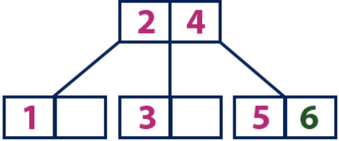
Добавим 5:

Элемент «5» больше корневого узла со значением «2», при этом корневой узел не является листом. Поэтому мы двигаемся по правому поддереву от «2». Мы приходим к узлу-листу и обнаруживаем, что он уже полон и не имеет пустых ячеек. Тогда мы делим этот узел, отправляя среднее значение (4) в родительский узел (2). В родительском узле есть для него пустая ячейка, поэтому значение «4» добавляется к узлу, в котором уже есть значение «2», а новый элемент «5» добавляется в качестве нового листа.



Добавим 6:

Элемент «6» больше, чем элементы корня «2» и «4», который не является листом. Мы двигаемся по правому поддереву от элемента «4». Мы достигаем листа со значением «5», у которого есть пустая ячейка, поэтому элемент «6» помещаем как раз в нее.



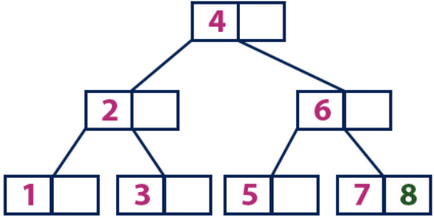
Добавим 7:

Элемент «7» больше, чем элементы корня «2» и «4», который не является листом. Мы двигаемся по правому поддереву от элемента «4». Мы достигаем узла-листа и видим, что он полон. Мы делим этот узел, отправляя среднее значение «6» вверх к родительскому узлу с элементами «2» и «4». Однако родительский узел тоже полон, поэтому мы делим узел с элементами «2» и «4», отправляя значение «4» родительскому узлу. Только вот этого узла еще нет. В таком случае узел с элементом «4» становится новым корнем дерева.



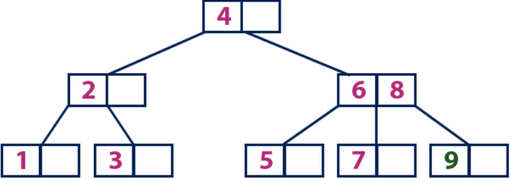
Добавим 8:

Элемент «8» больше корневого узла со значением «4», при этом корневой узел не является листом. Мы двигаемся по правому поддереву от элемента «4» и приходим к узлу со значением «6». «8» больше «6» и узел с элементом «6» не является листом, поэтому двигаемся по правому поддереву от «6». Мы достигаем узла-листа с «7», у которого есть пустая ячейка, поэтому в нее мы помещаем «8».



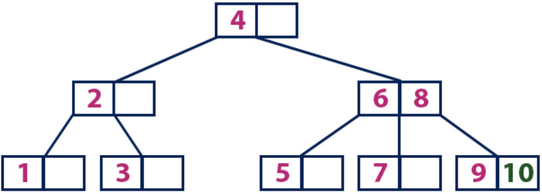
Добавим 9:

Элемент «9» больше корневого узла со значением «4», при этом корневой узел не является листом. Мы двигаемся по правому поддереву от элемента «4» и приходим к узлу со значением «6». «9» больше «6» и узел с элементом «6» не является листом, поэтому двигаемся по правому поддереву от «6». Мы достигаем узла-листа со значениями «7» и «8». Он полон. Мы делим этот узел, отправляя среднее значение (8) родительскому узлу. Родительский узел «6» имеет пустую ячейку, поэтому мы помещаем «8» в нее. При этом новый элемент «9» добавляется в узел-лист.



Добавим 10:

Элемент «10» больше корневого узла со значением «4», при этом корневой узел не является листом. Мы двигаемся по правому поддереву от элемента «4» и приходим к узлу со значениями «6» и «8». «10» больше «6» и «8» и узел с этими элементами не является листом, поэтому двигаемся по правому поддереву от «8». Мы достигаем узла-листа со значением «9». У него есть пустая ячейка, поэтому туда мы помещаем «10».



**Поиск элемента в B-дереве**

Поиск по B-дереву аналогичен поиску по двоичному дереву поиска. В двоичном дереве поиска поиск начинается с корня и каждый раз принимается двустороннее решение (пойти по левому поддереву или по правому). В В-дереве поиск также начинается с корневого узла, но на каждом шаге принимается n-стороннее решение, где n – это общее количество потомков рассматриваемого узла. В В-дереве сложность поиска составляет O(log n). Поиск происходит следующим образом:

Шаг 1: Считать элемент для поиска.

Шаг 2: Сравнить искомый элемент с первым значением ключа в корневом узле дерева.

Шаг 3: Если они совпадают, вывести: «Искомый узел найден!» (или что-то подоьное) и завершить поиск.

Шаг 4: Если они не совпадают, проверить больше или меньше значение элемента, чем текущее значение ключа.

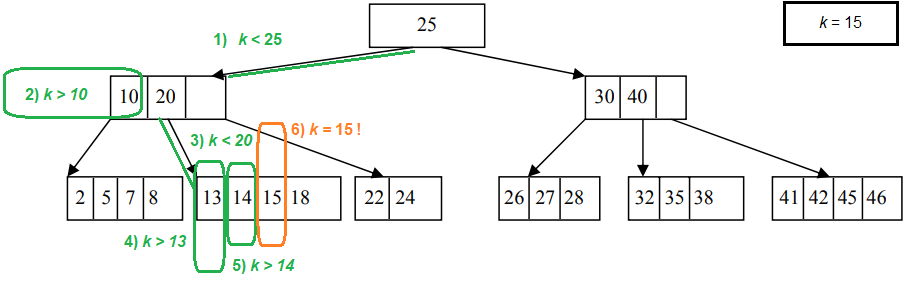
Шаг 5: Если искомый элемент меньше, продолжить поиск по левому поддереву.

Шаг 6: Если искомый элемент больше, сравнить элемент со следующим значением ключа в узле и повторять Шаги 3, 4, 5 и 6 пока не будет найдено совпадение или пока искомый элемент не будет сравнен с последним значением ключа в узле-листе.

Шаг 7: Если последнее значение ключа в узле-листе не совпало с искомым, вывести «Элемент не найден!» и завершить поиск.

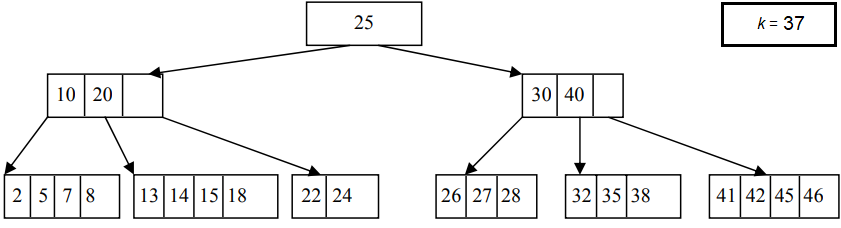
**Пример 8.4**.

Найти в дереве **Примера 8.1** элемент с ключом 15.



**Пример 8.5**.

Найти в дереве **Примера 8.1** элемент с ключом 37.



**Выполняется самостоятельно**

**Удаление элемента из B-дерева**

Изменение структуры узлов может возникнуть, когда узел удаляется.

Если в соседних узлах слишком мало ключей – занятость узлов падает ниже значения m/2 – и эти узлы находятся на одном родственном уровне, то такие узлы необходимо слить. Данная ситуация называется **антипереполнением**.

Существует **два способа объединения узлов**:

1. если у двух смежных узлов общий предок, и их содержимые помещаются в один узел, их следует объединить;

2. если содержимое смежных узлов не помещается в одном узле, то ключи перераспределяются между ними, чтобы восстановить баланс

Рассмотрим задачу удаления узлов из B-дерева [1]. Различаются два случая:

1. **элемент находится на листе** – удаление выполняется просто в массиве соответствующего листа;

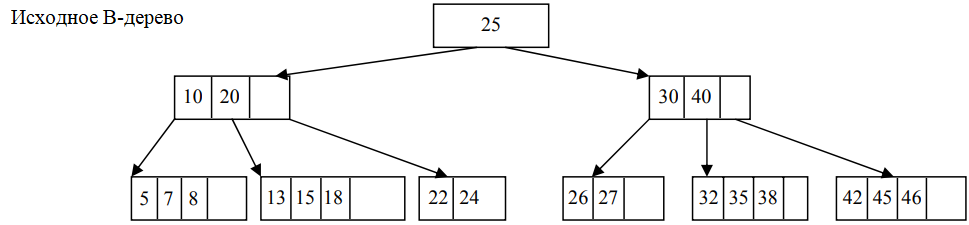
2. **элемент находится не на листе**. Тогда он заменяется на самый правый элемент самого правого листа, а у листа удаляется элемент. После удаления необходимо проверить, что на листе осталось элементов не меньше n, иначе будет нарушено основное условие B-деревьев. В этом случае забираются элементы с одной из соседних страниц. Если это невозможно (в случае, когда соседняя страница тоже достигла своего минимального уровня), то страницы сливают, добавляя средний элемент со страницы-предка.

**Пример 8.6**.

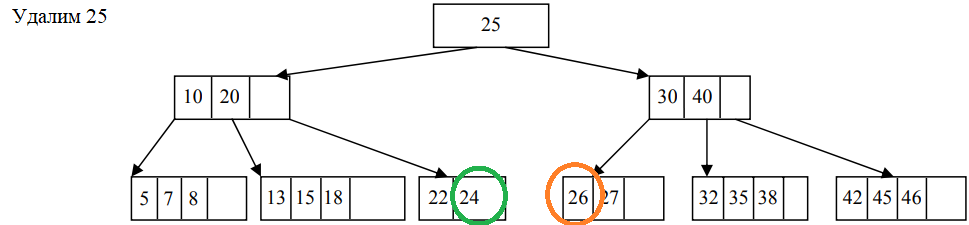
Удалим у дерева, построенного в **примере 8.2**, узлы в порядке обратном их поступлению:

|  |
| --- |
| **25, 45, 24, 38, 32, 8, 27, 46, 13, 42, 5, 22, 18, 26, 7, 35, 15**. |

Исходное B-дерево



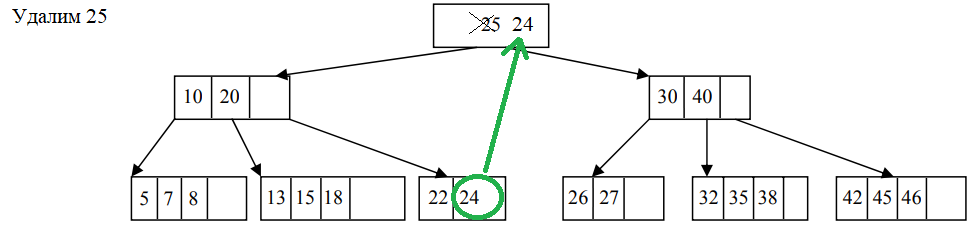
Удаляемый ключ можно заменить только на самый большой из меньших или самый маленький из больших [4]. На рисунке они обведены зеленым и оранжевым цветом соответственно. Первый — это последний ключ правого листа слева, второй – первый ключ левого листа справа.

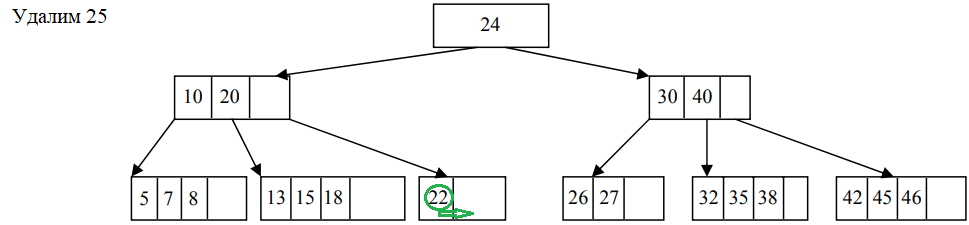


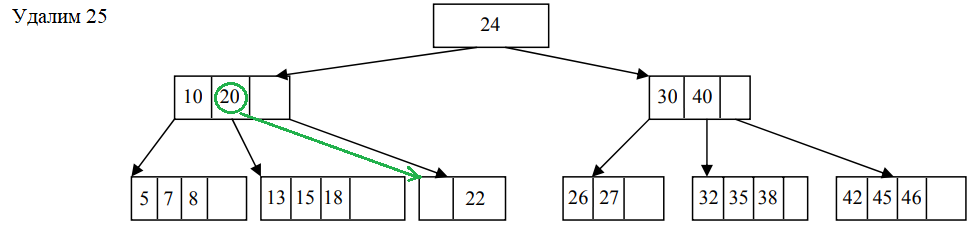
**Два претендента на роль заменяющего элемента показаны цветом.**

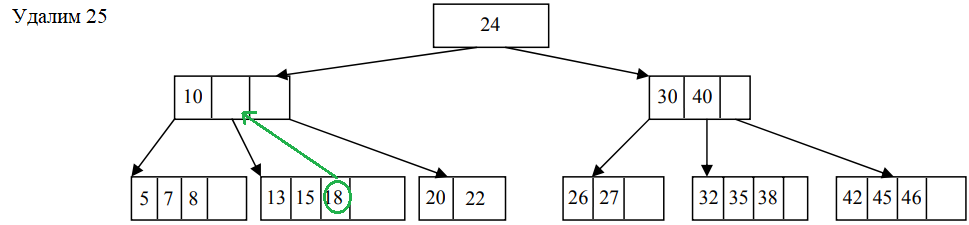
Если один из них не пуст, просто меняем удаляемый ключ на один из них и удаляем заменяющий ключ из исходного листа, если нет, возвращаемся к задаче, как сделать листовой узел непустым.

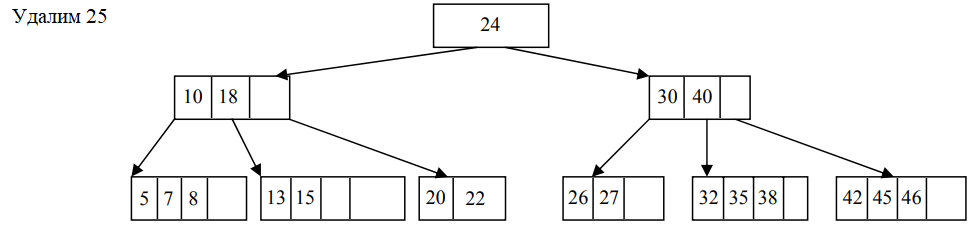
**Выбираем для замены ключа 25 ключ 24.**

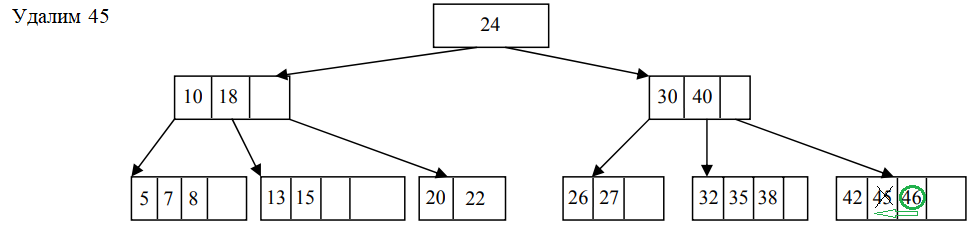


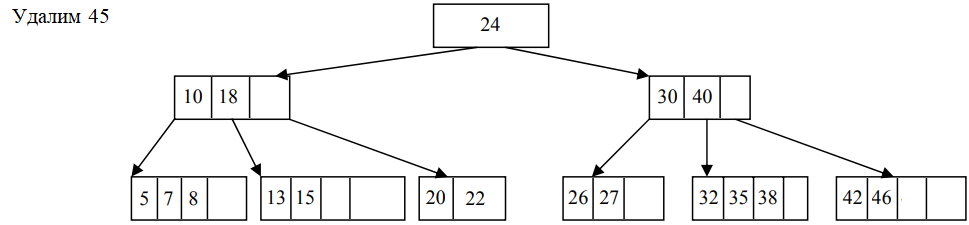


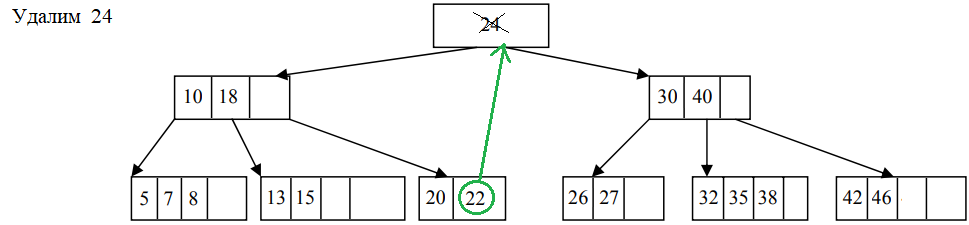


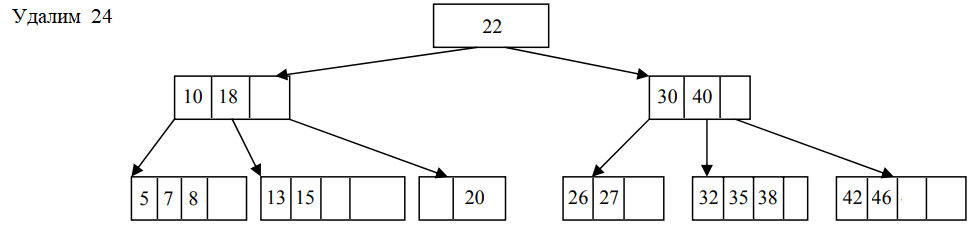


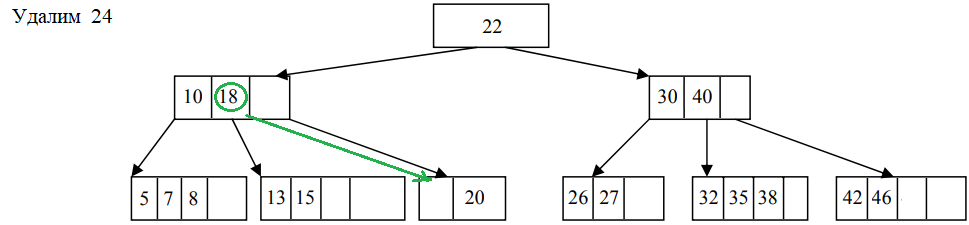


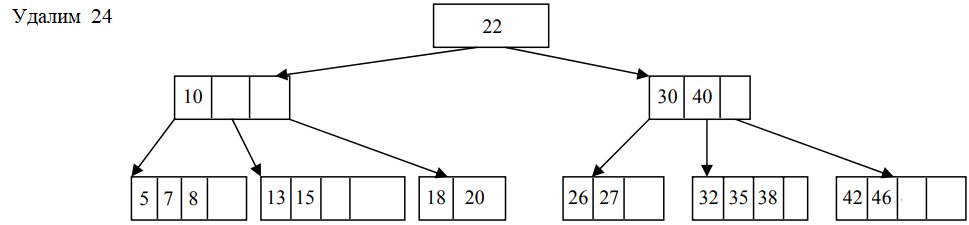




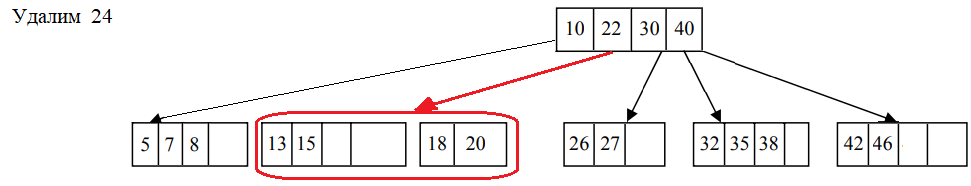


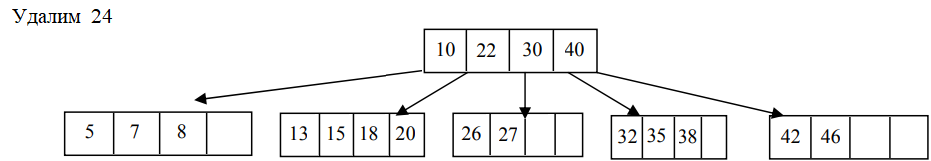


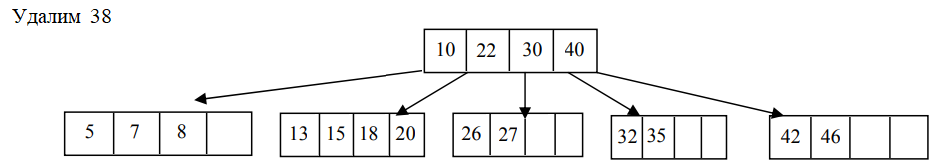


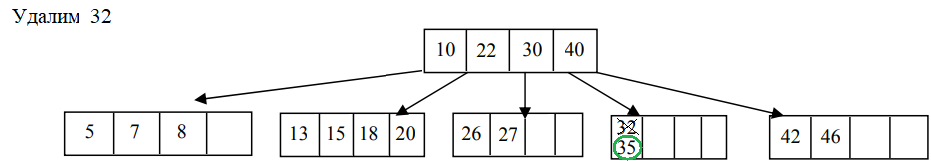


Теперь проверим узел с ключом **10**, из которого был удален элемент, на предмет антипереполнения. Число элементов в узле равно **единице**. Это меньше половины размера узла (**m/2 = 4/2 = 2**). Следовательно, возникло **антипереполнение**. Выберем два соседних узла: один - из которого было удалено **18,** второй - соседний узел с ключами **30, 40**. Суммарное число элементов в этих узлах не превышает максимального размера узла (m=4), значит, нам следует произвести слияние узлов. При слиянии элемент **22** родителя попадает в результирующий узел и удаляется из родителя.

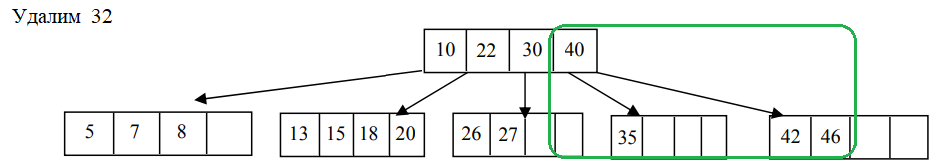


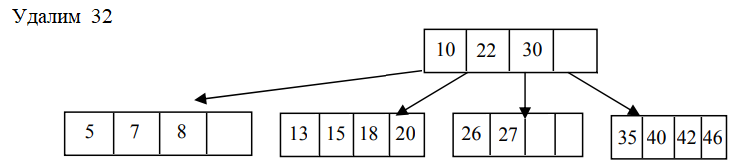


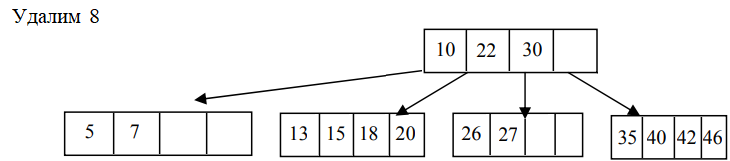


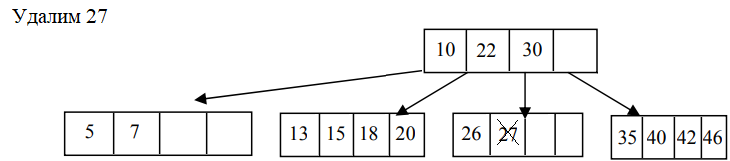


При удалении ключа **32** возникло **антипереполнение,** следует произвести слияние узлов.

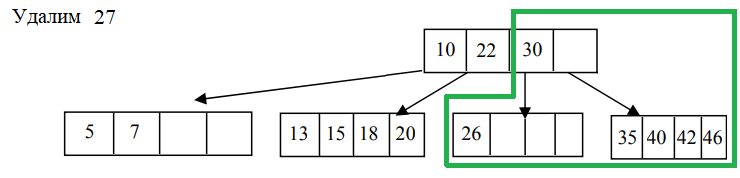


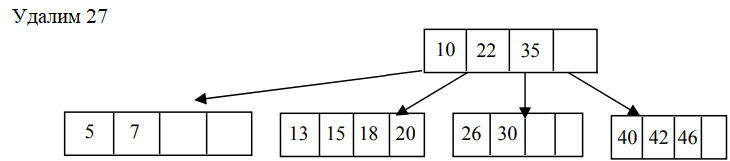


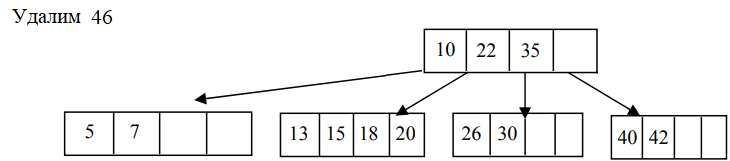


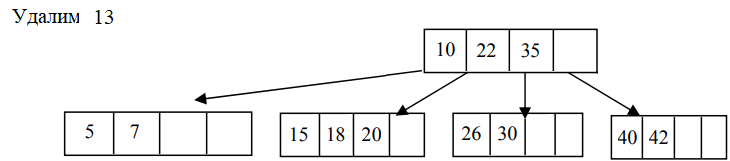


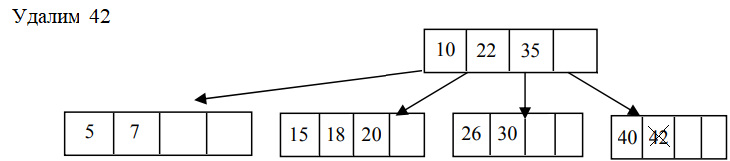
При удалении ключа **27** возникло **антипереполнение,** следует произвести слияние узлов.



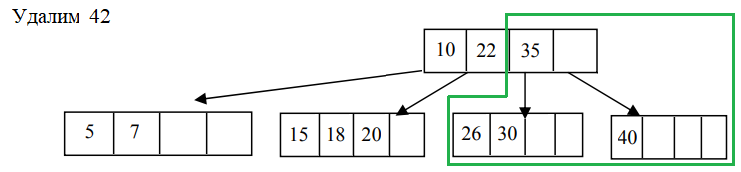


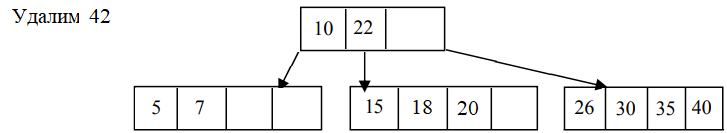


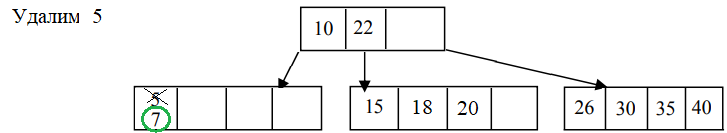




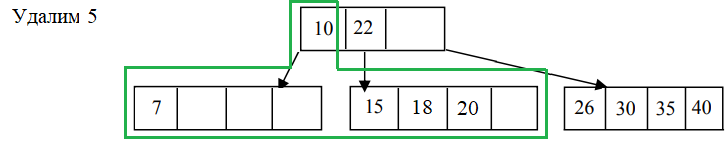
При удалении ключа **42** возникло **антипереполнение,** следует произвести слияние узлов.



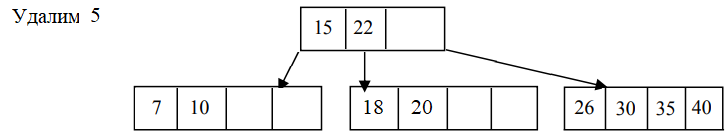


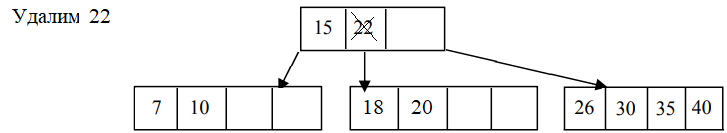


При удалении ключа **5** возникло **антипереполнение,** следует произвести слияние узлов.

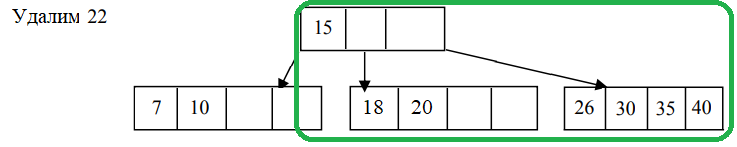


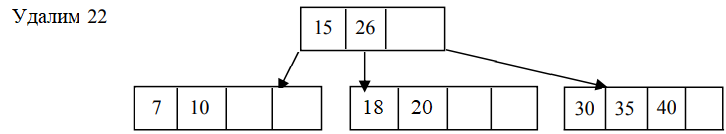
При этом содержимое смежных узлов (с ключами 7 и {15, 18, 20}) не помещается в одном узле, то ключи перераспределяются между ними, чтобы восстановить баланс.

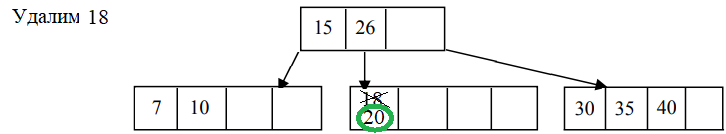




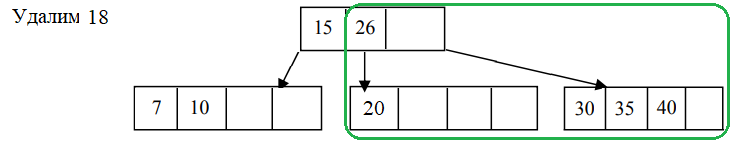
При удалении ключа **22** возникло **антипереполнение,** следует произвести слияние узлов с перераспределением ключей.

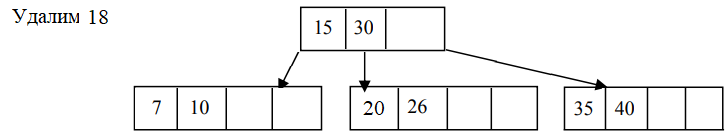


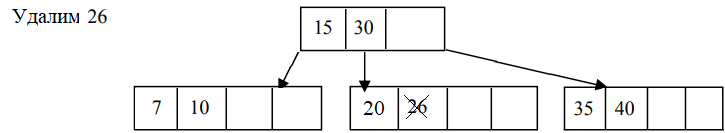




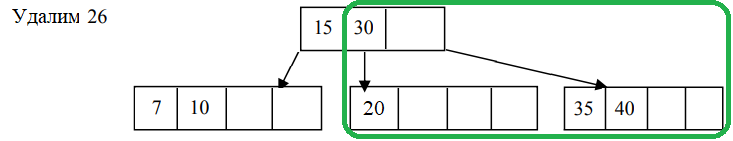
При удалении ключа **18** возникло **антипереполнение,** следует произвести слияние узлов с перераспределением ключей.

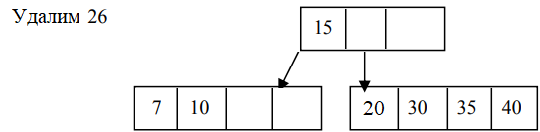


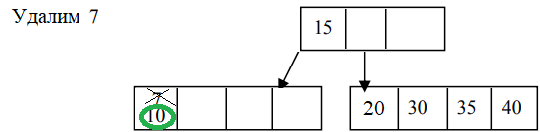




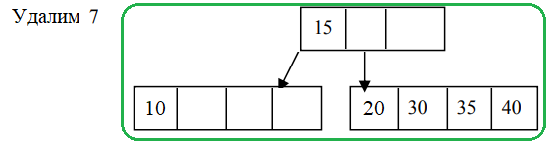
При удалении ключа **26** возникло **антипереполнение,** следует произвести слияние узлов.

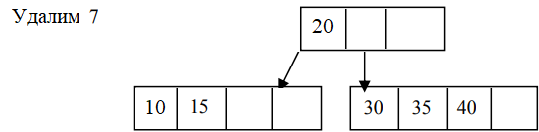


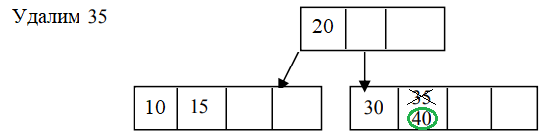


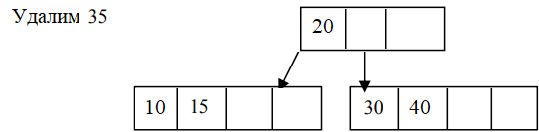


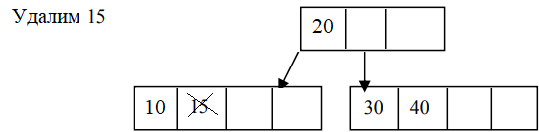
При удалении ключа **7** возникло **антипереполнение,** следует произвести слияние узлов с перераспределением ключей.



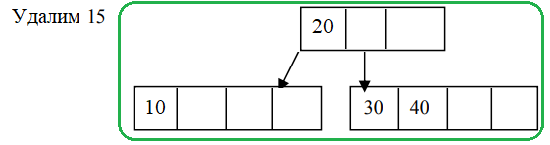


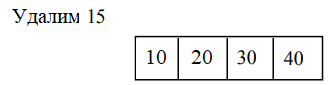






При удалении ключа **7** возникло **антипереполнение,** следует произвести слияние узлов.





СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://dispace.edu.nstu.ru/didesk/file/get/192354&ved=2ahUKEwiVvu6146WLAxUBksMKHR2iKNkQFnoECDkQAQ&usg=AOvVaw3TTkdhsLid5SZTu06Jm4KT

2 Структура данных B-дерево. URL: https://habr.com/ru/companies/otus/articles/459216/

3 B-деревья. URL: https://ru.hexlet.io/courses/algorithms-trees/lessons/btrees/theory\_unit

4 Алгоритм удаления узла из btree. URL: https://habr.com/ru/articles/242411/