**КФУ, Институт ИТИС, 1 курс, 2 семестр, 2021**

**Семестровая работа №1:**

**«B-дерево»**

**Выполнила:**

Галиева Илюза Ильдаровна

группа 11-203

**Преподаватель:**

Андреичев М.Д.

1. Дерево

B-дерево — структура данных, дерево поиска. С точки зрения внешнего логического представления — сбалансированное, сильно ветвистое дерево. Часто используется для хранения данных во внешней памяти.

В отличие от традиционных деревьев бинарного поиска, B-деревья характеризуются большим количеством ключей, которые они могут хранить в одном узле, вот почему они также известны как деревья ”большого ключа". Каждый узел в B-дереве может содержать несколько ключей, что позволяет дереву иметь больший коэффициент ветвления и, следовательно, меньшую высоту. Такая небольшая высота приводит к уменьшению объема дисковых операций ввода-вывода, что приводит к более быстрым операциям поиска и вставки. B-деревья особенно хорошо подходят для систем хранения данных с медленным и громоздким доступом к данным, таких как жесткие диски, флэш-память и компакт-диски.

B-деревья поддерживают баланс, гарантируя, что каждый узел имеет минимальное количество ключей, поэтому дерево всегда сбалансировано. Этот баланс гарантирует, что временная сложность для таких операций, как вставка, удаление и поиск, всегда равна O (log n), независимо от начальной формы дерева.

*Историческая справка*

Использование B-деревьев впервые было предложено Р. Бэйером (англ. R. Bayer) и Э. МакКрейтом (англ. E. McCreight) в 1970 году.

*Описание*

Свойства

1. Ключи в каждом узле x упорядочены по неубыванию.
2. В каждом узле есть логическое значение x.leaf. Оно истинно, если x — лист.
3. Каждый узел, кроме корня, содержит не менее t-1 ключей, а каждый внутренний узел имеет как минимум t дочерних узлов, где t — минимальная степень B-дерева.
4. Все листья находятся на одном уровне, т. е. обладают одинаковой глубиной, равной высоте дерева.
5. Корень имеет не менее 2 дочерних элементов и содержит не менее 1 ключа.

Операции над B-деревом

Над B-деревом можно проводить следующие операции:

-Поиск

-Вставка

-Удаление

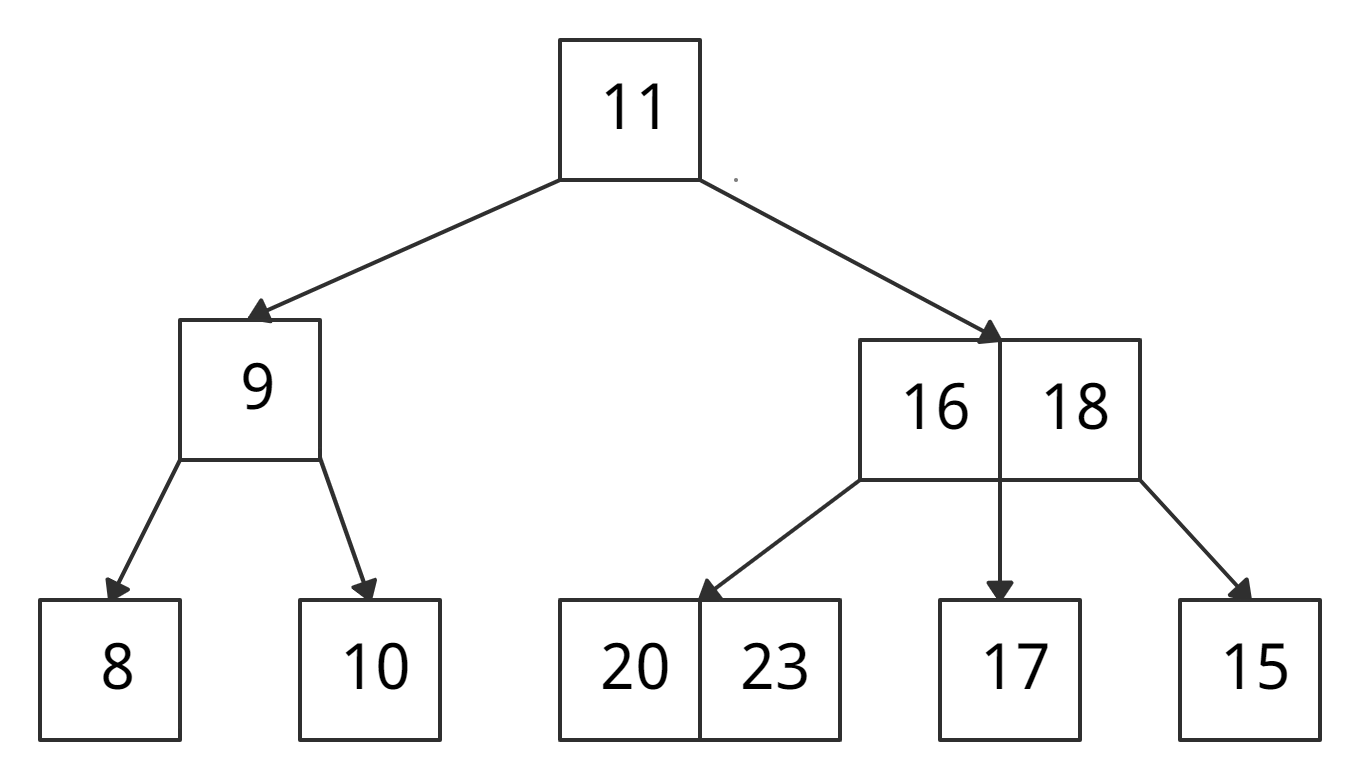
1. Поиск

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:

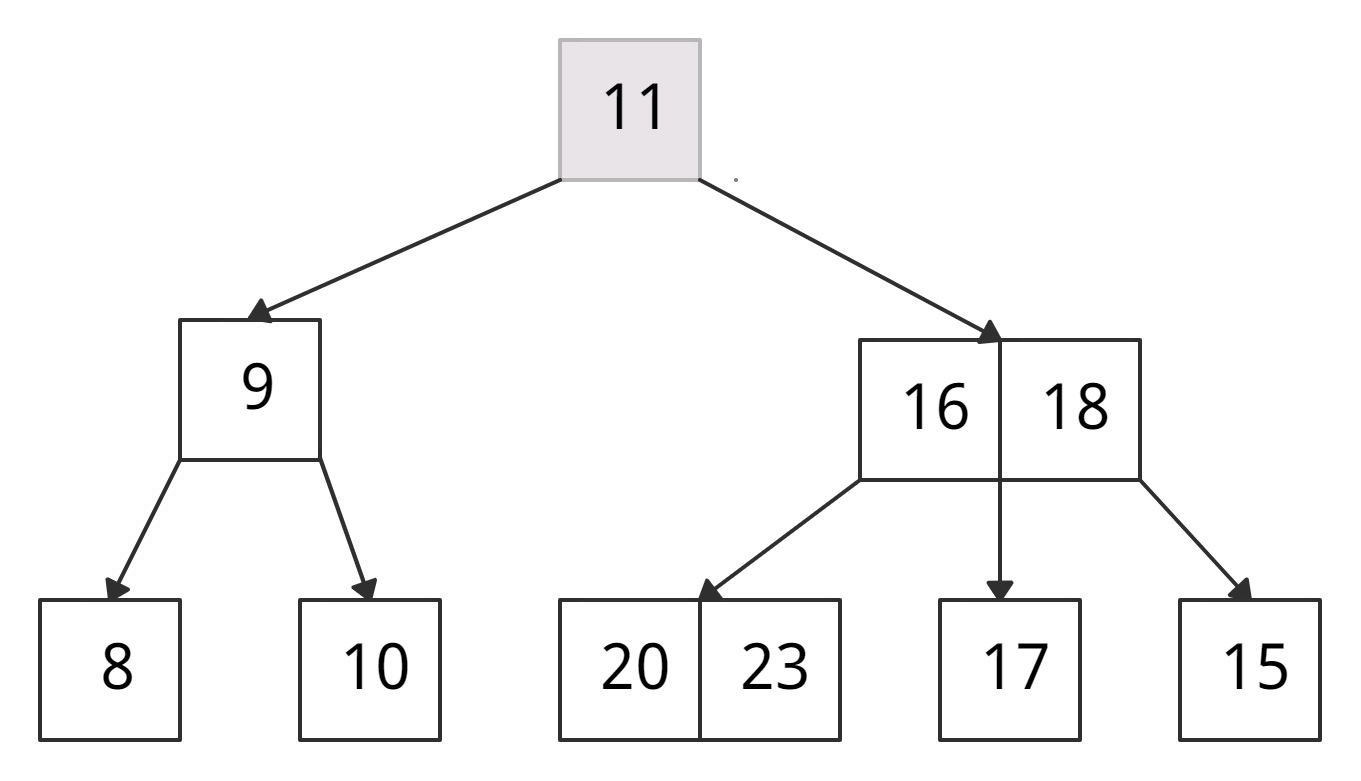
1. Сравниваем k с первым ключом узла, начиная с корня. Если k = первый ключ узла, возвращаем узел и индекс.
2. Если k.leaf = true, возвращаем NULL. Элемент не найден.
3. Если k < первый ключ корня, рекурсивно ищем левый дочерний элемент этого ключа.
4. Если в текущем узле более одного ключа и k > первый ключ, сравниваем k со следующим ключом в узле.  
   Если k < следующий ключ, ищем левый дочерний элемент этого ключа (k находится между первым и вторым ключами).  
   Иначе иначе ищем правый дочерний элемент ключа.
5. Повторяем шаги с 1 по 4, пока не дойдем до листа.

Пример:

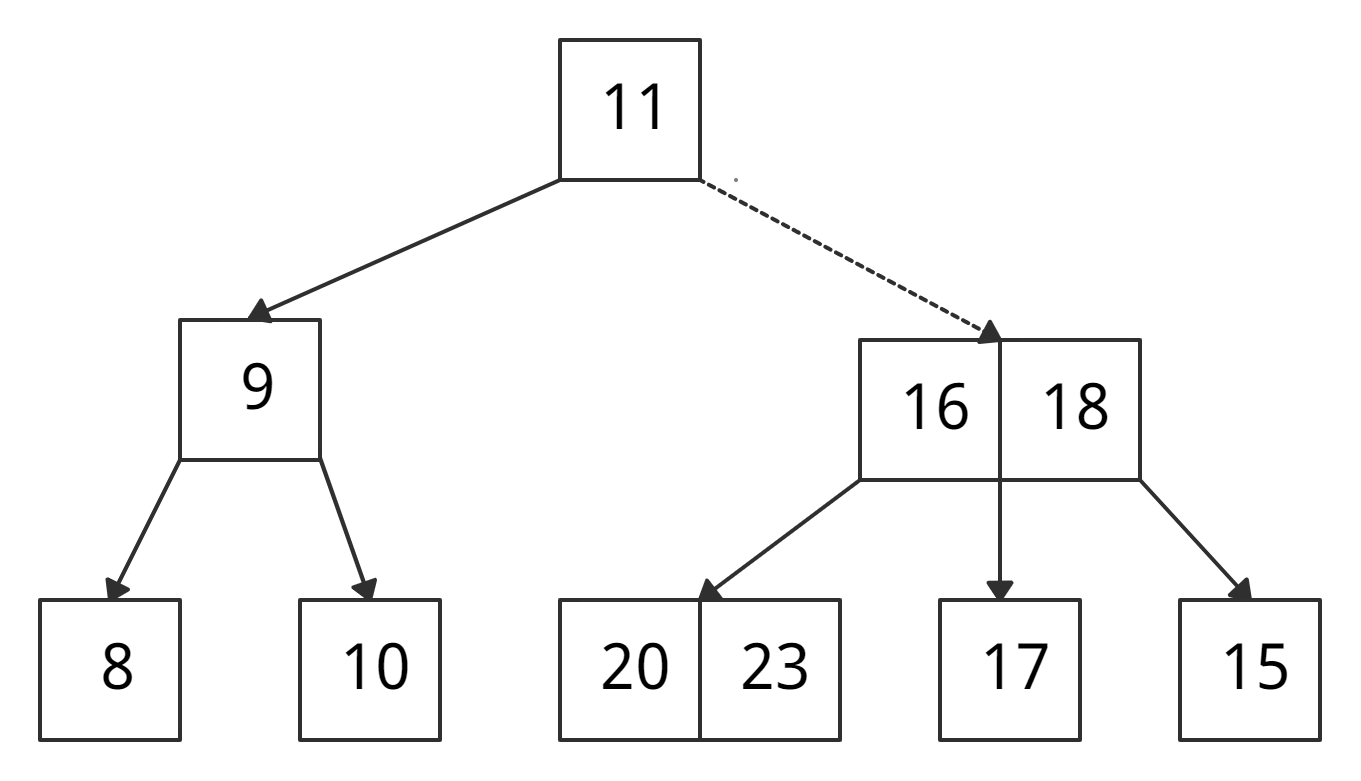
 Хотим найти ключ k = 17 в этом дереве.



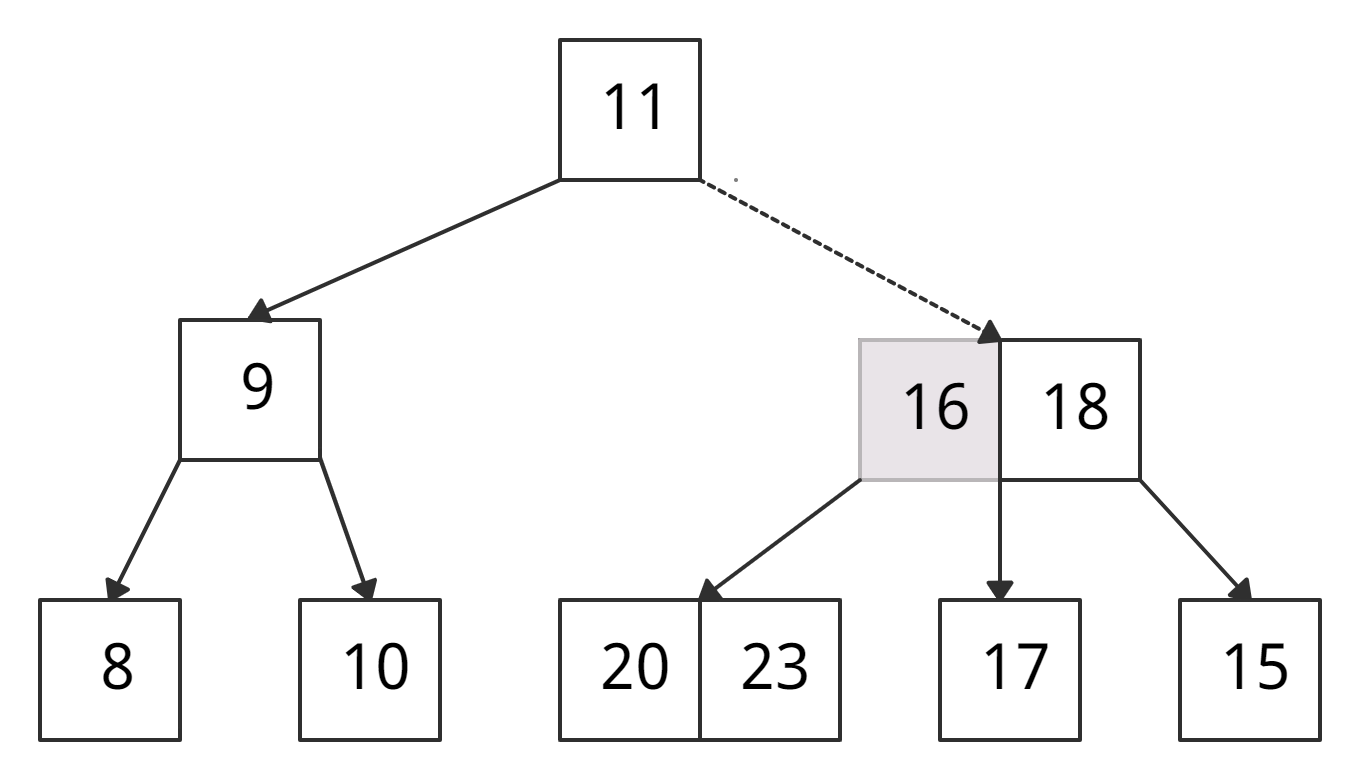
k нет в корне → сравниваем k с ключом корня.



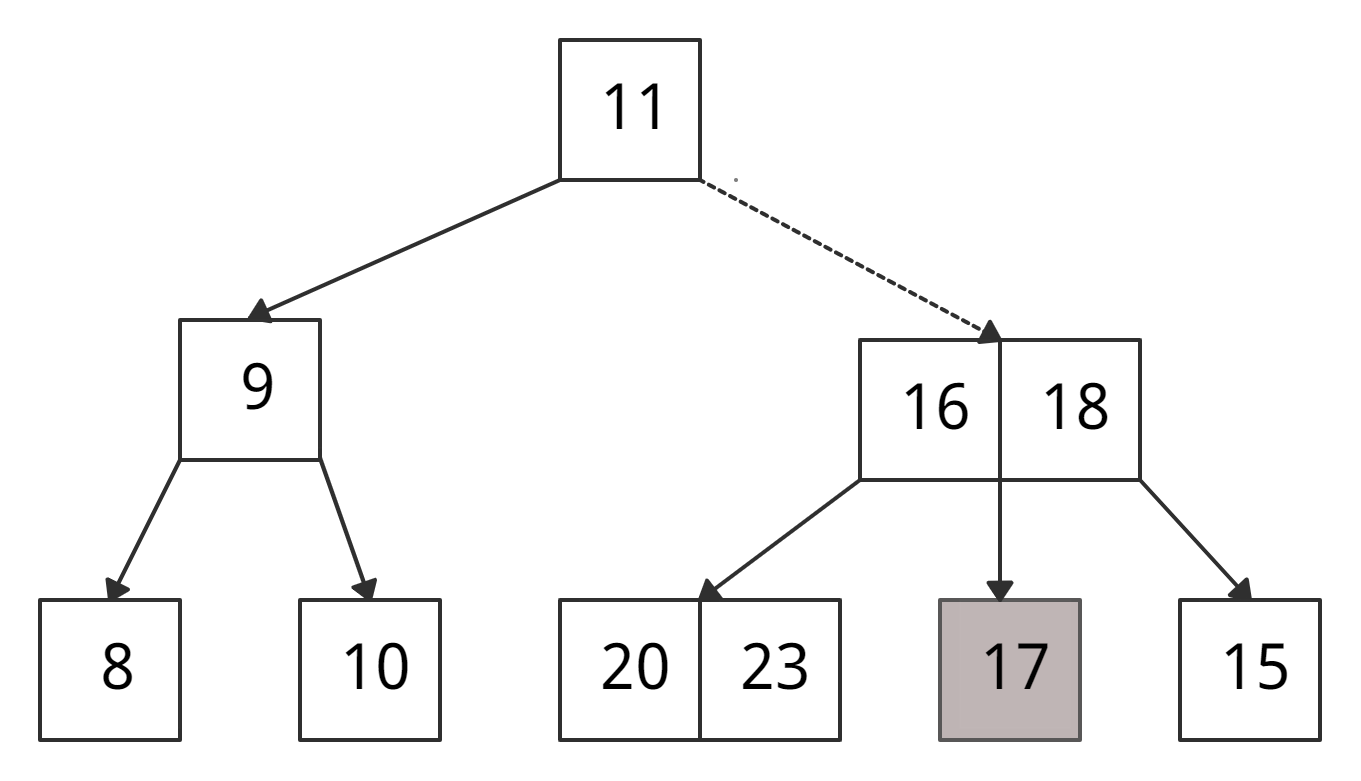
k > 11 → идем через правого «ребенка».



Сравниваем k с первым ключом узла: k > 16 → сравниваем k со вторым ключом узла.



k > 18 → k лежит между 16 и 18. Ищем либо в правом «ребенке» 16, либо в левом «ребенке» 18.



Нашли 17.

1. Вставка

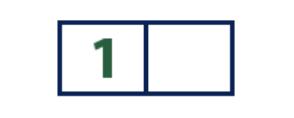
В В-дереве новый элемент может быть добавлен только в узел-лист. Это значит, что новая пара ключ-значение всегда добавляется только к узлу-листу. Вставка происходит следующим образом:

*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:

Давайте создадим B-дерево порядка 3, добавляя в него числа от 1 до 10.

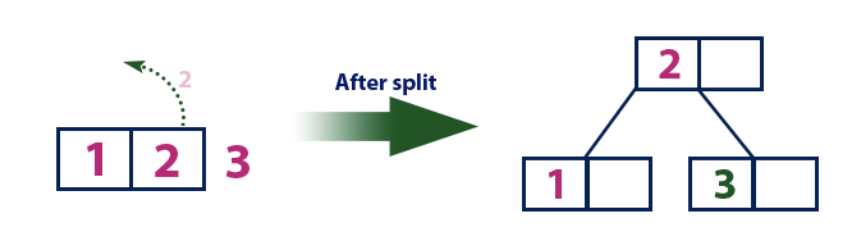
*Insert(1):*  
 Поскольку «1» — это первый элемент дерева – он вставляется в новый узел и этот узел становится корнем дерева.



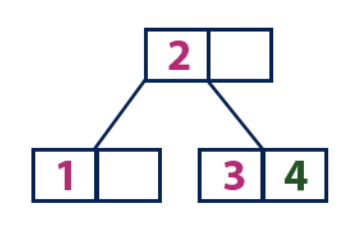
*Insert(2):*  
 Элемент «2» добавляется к существующему узлу-листу. Сейчас у нас всего один узел, следовательно он является и корнем и листом одновременно. В этом листе имеется пустая ячейка. Тогда «2» встает в эту пустую ячейку.



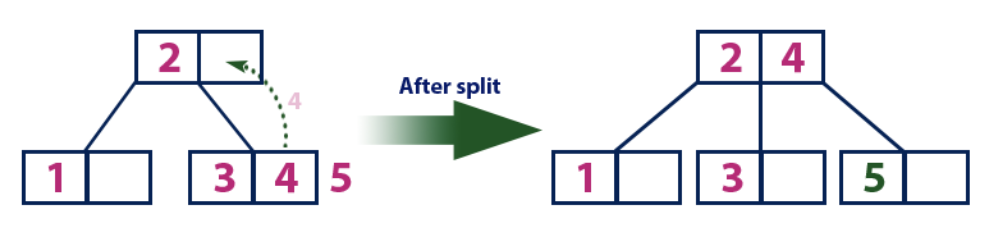
*Insert(3):*  
 Элемент «3» добавляется к существующему узлу-листу. Сейчас у нас только один узел, который одновременно является и корнем и листом. У этого листа нет пустой ячейки. Поэтому мы разделяем этот узел, отправляя среднее значение (2) в родительский узел. Однако у текущего узла родительского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



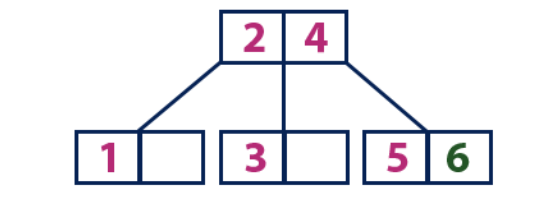
*Insert(4):*  
Элемент «4» больше корневого узла со значением «2», при этом корневой узел не является листом. Поэтому мы двигаемся по правому поддереву от «2». Мы приходим к узлу-листу со значением «3», у которого имеется пустая ячейка. Таким образом, мы можем вставить элемент «4» в эту пустую ячейку.



*Insert(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.

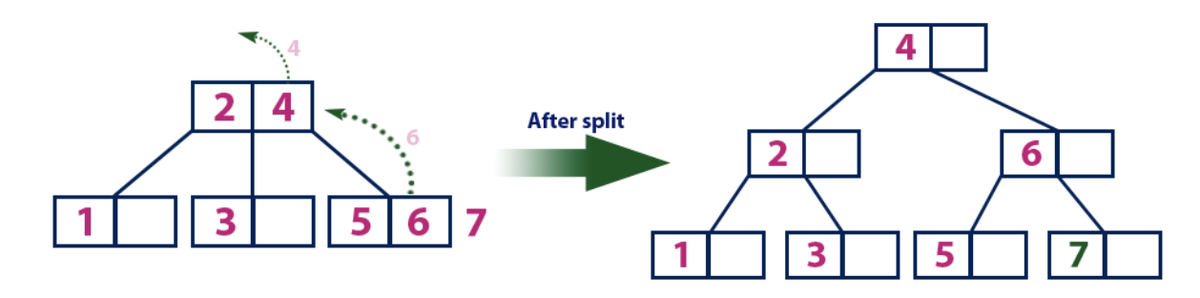


*Insert(6):*  
 Элемент «6» больше, чем элементы корня «2» и «4», который не является листом. Мы двигаемся по правому поддереву от элемента «4». Мы достигаем листа со значением «5», у которого есть пустая ячейка, поэтому элемент «6» помещаем как раз в нее.

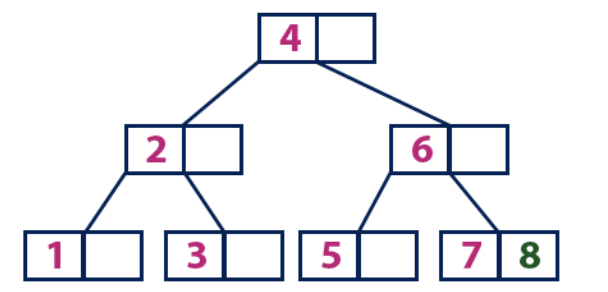


*Insert(7):*

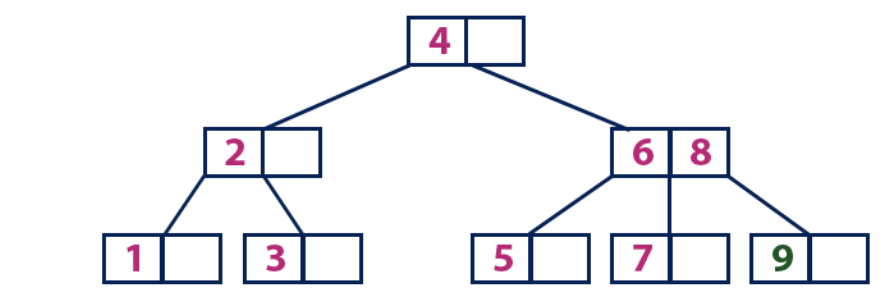
Элемент «7» больше, чем элементы корня «2» и «4», который не является листом. Мы двигаемся по правому поддереву от элемента «4». Мы достигаем узла-листа и видим, что он полон. Мы делим этот узел, отправляя среднее значение «6» вверх к родительскому узлу с элементами «2» и «4». Однако родительский узел 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



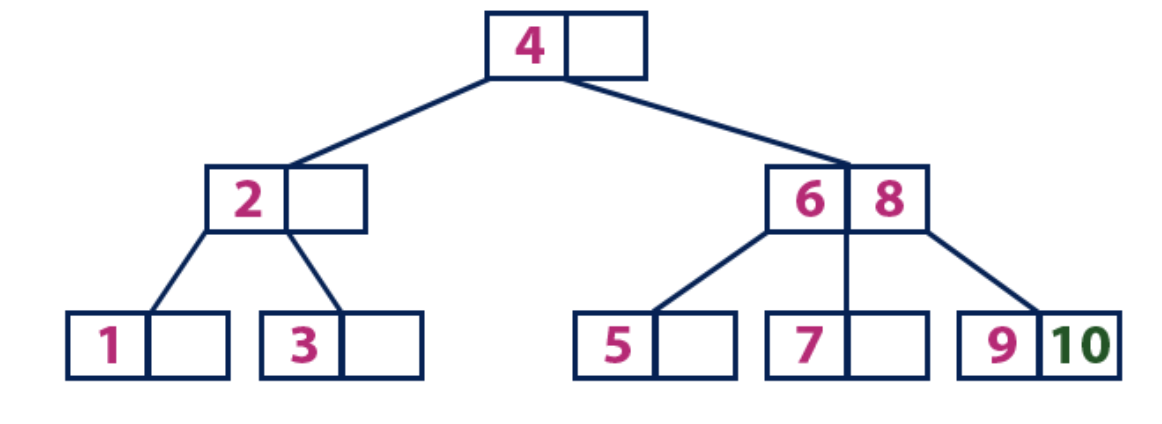
*Insert(8):*  
 Элемент «8» больше корневого узла со значением «4», при этом корневой узел не является листом. Мы двигаемся по правому поддереву от элемента «4» и приходим к узлу со значением «6». «8» больше «6» и узел с элементом «6» не является листом, поэтому двигаемся по правому поддереву от «6».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



*Insert(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.



Insert(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».

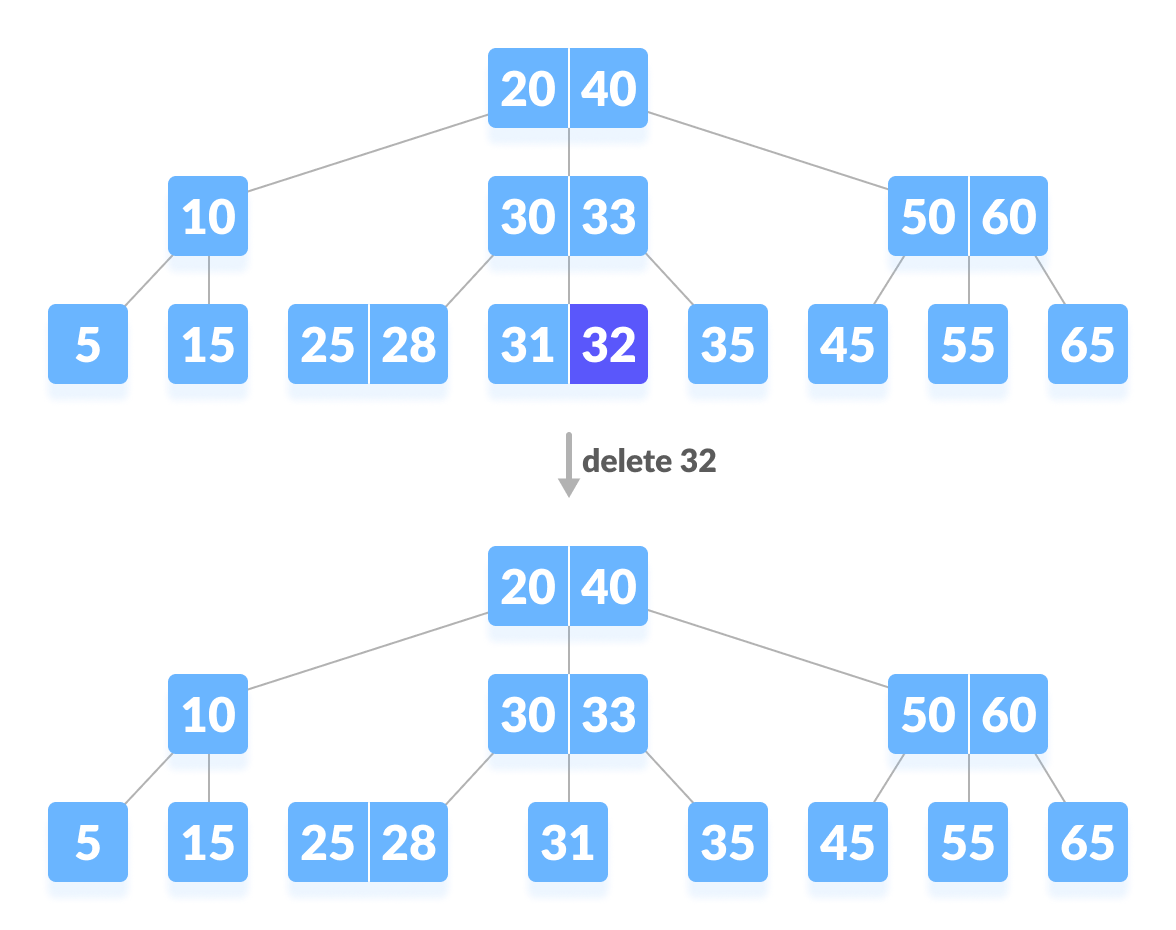


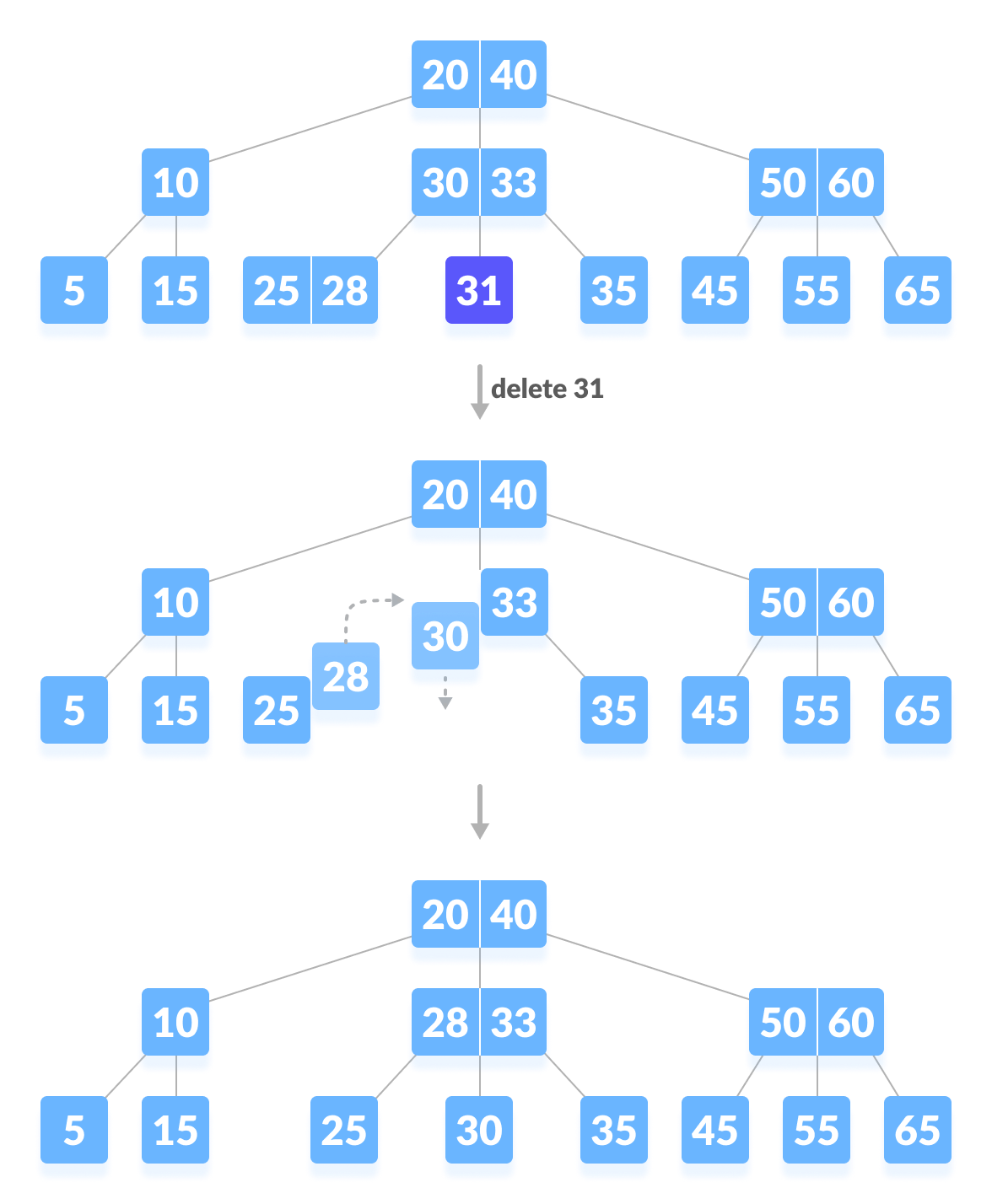
1. Удаление

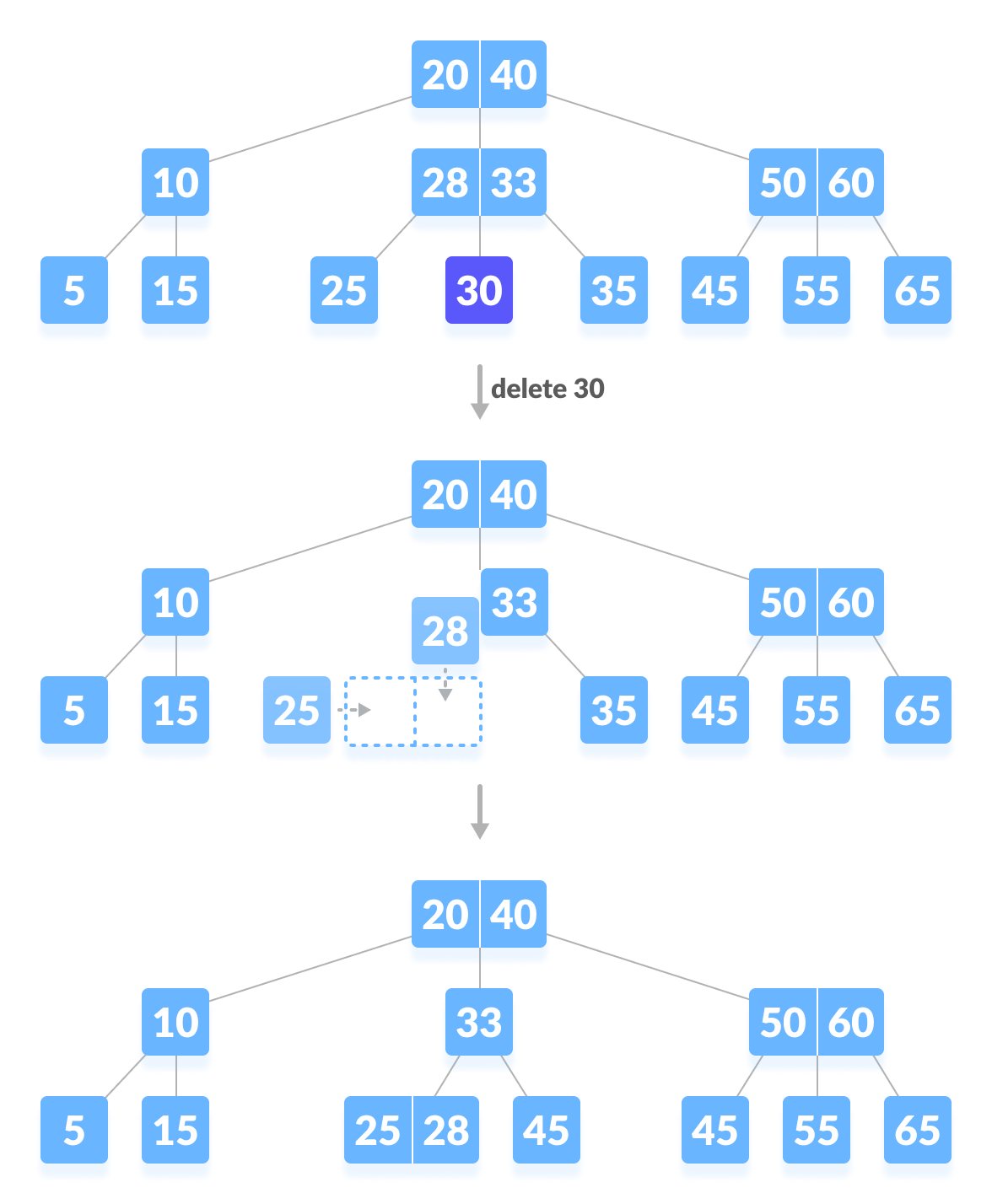
Существует три основных варианта операции удаления в B-дереве.

### **Случай I**

Ключ, подлежащий удалению, находится в листе. Для этого есть два варианта.

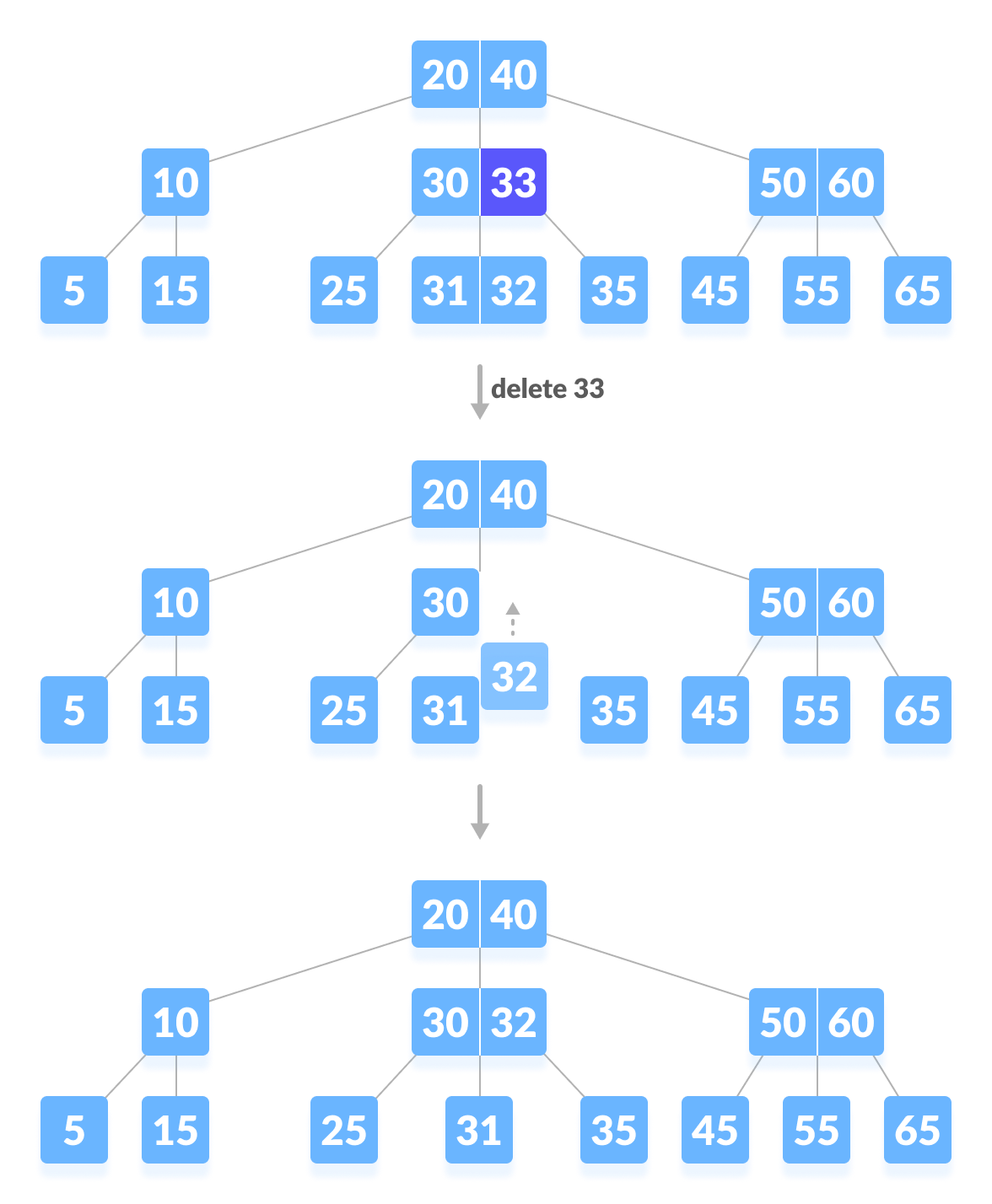
1.Удаление ключа не нарушает свойство минимального количества ключей, которое должен содержать узел.  
  
 В приведенном ниже дереве удаление 32 не нарушает вышеуказанные свойства.Удаление конечного ключа (32) из B-дерева

2.Удаление ключа нарушает свойство минимального количества ключей, которое должен содержать узел. В этом случае мы заимствуем ключ из его ближайшего соседнего узла-побратима в порядке слева направо.  
  
 Сначала посетите ближайшего левого родственника. Если у левого дочернего узла больше минимального количества ключей, то позаимствуйте ключ у этого узла.  
  
 В противном случае убедитесь, что оно заимствовано из ближайшего правого дочернего узла.  
  
 В дереве ниже удаление 31 приводит к вышеуказанному условию. Давайте позаимствуем ключ из левого дочернего узла.Удаление конечного ключа (31)

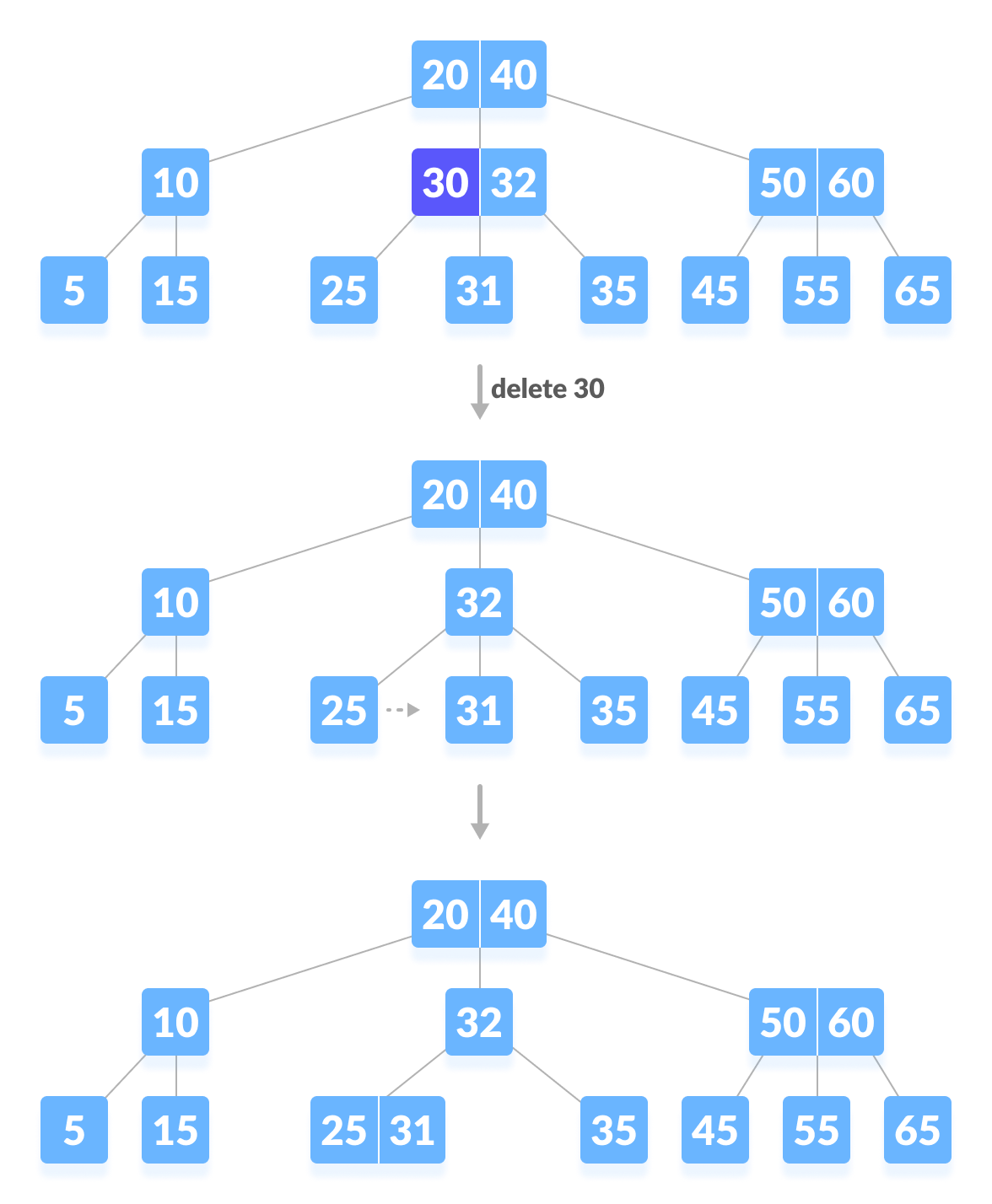
Если оба ближайших родственных узла уже имеют минимальное количество ключей, то объедините узел либо с левым родственным узлом, либо с правым родственным узлом. ****Это объединение выполняется через родительский узел.****  
  
Удаление 30 приводит к описанному выше случаю.  
 Удалить конечный ключ (30)

### **Случай II**

Если ключ, подлежащий удалению, находится во внутреннем узле, происходят следующие случаи.

1.Внутренний узел, который удаляется, заменяется на предыдущий по порядку, если у левого дочернего элемента больше минимального количества ключей.Удаление внутреннего узла (33)

2.Внутренний узел, который удаляется, заменяется преемником по порядку, если у нужного дочернего элемента больше минимального количества ключей.

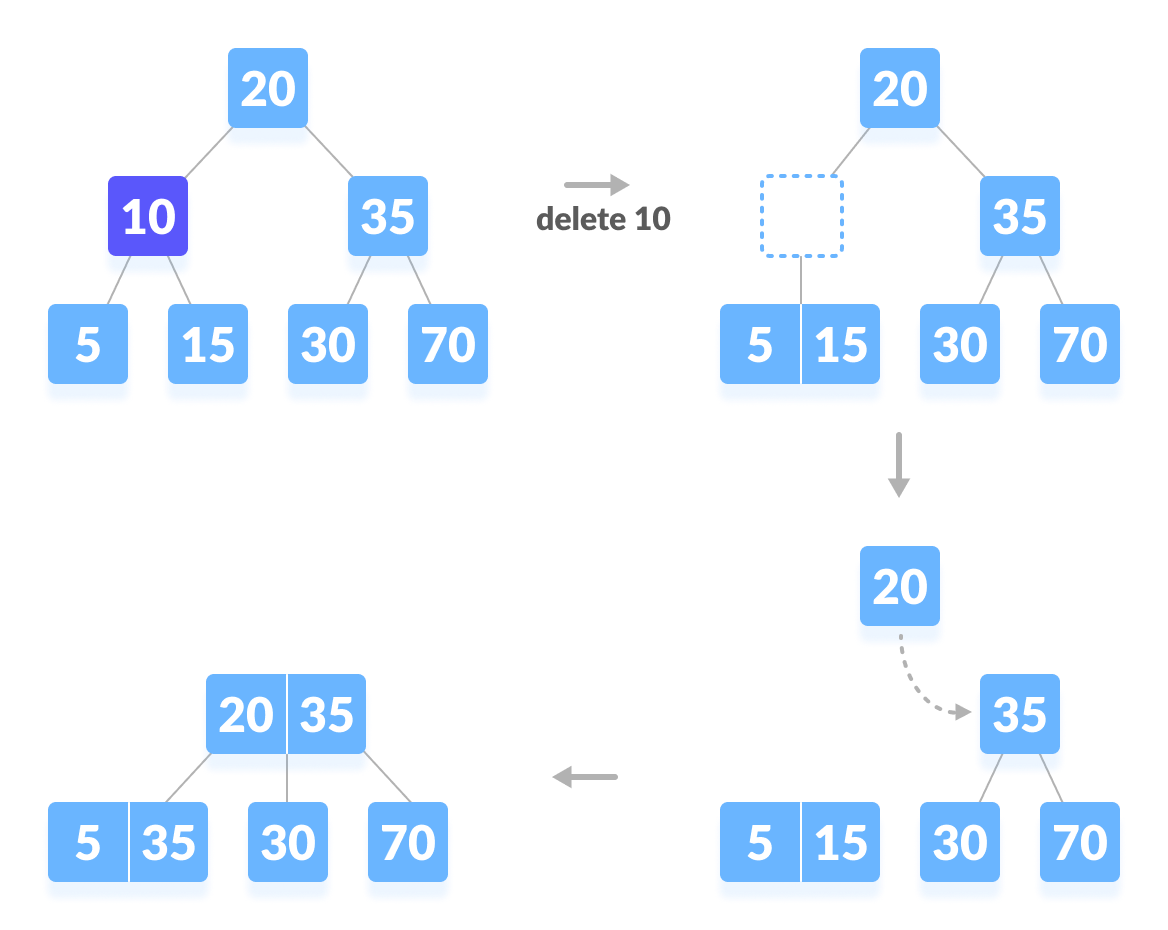
3.Если у любого из дочерних элементов есть точно минимальное количество ключей, тогда объедините левый и правый дочерние элементы.  
 Удаление внутреннего узла (30)

После объединения, если родительский узел имеет меньше минимального количества ключей, тогда найдите братьев и сестер, как в случае I.

**Пример III**

В этом случае высота дерева уменьшается. Если целевой ключ находится во внутреннем узле, и удаление ключа приводит к меньшему количеству ключей в узле (т.е. меньше минимально необходимого), тогда найдите предшественника inorder и преемника inorder. Если оба дочерних элемента содержат минимальное количество ключей, то заимствование невозможно. Это приводит к случаю II (3), то есть к объединению дочерних элементов.

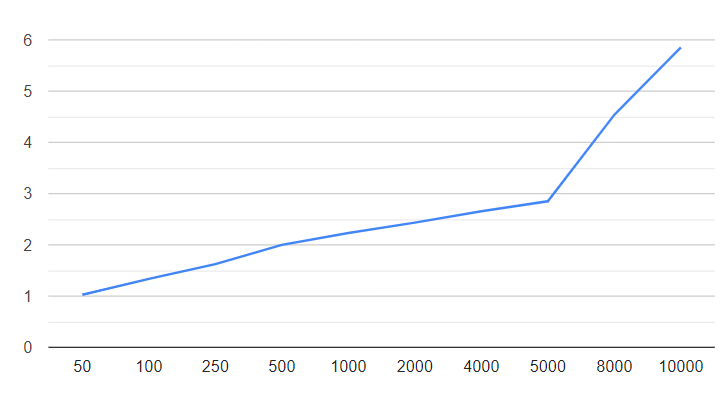
Опять же, поищите родственного, чтобы позаимствовать ключ. Но, если у родственного узла также есть только минимальное количество ключей, тогда объедините узел с родственным узлом вместе с родительским. Расположите дочерние элементы соответствующим образом (в порядке возрастания).

Удаление внутреннего узла (10)

Оценка времени сложности

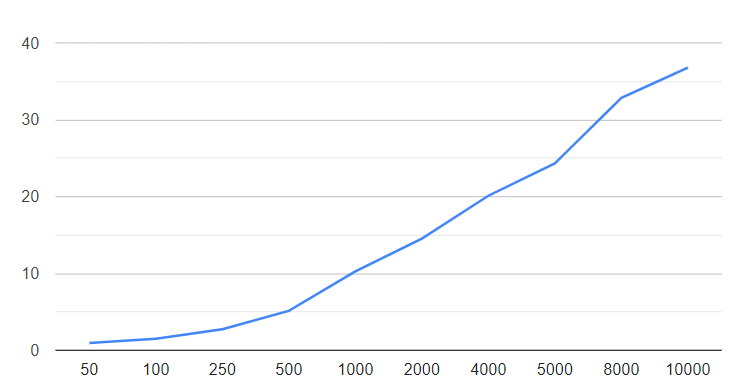
1)Поиск

**Средняя временная сложность:** Θ(log n)  
**Худшая временная сложность:** Θ(log n)



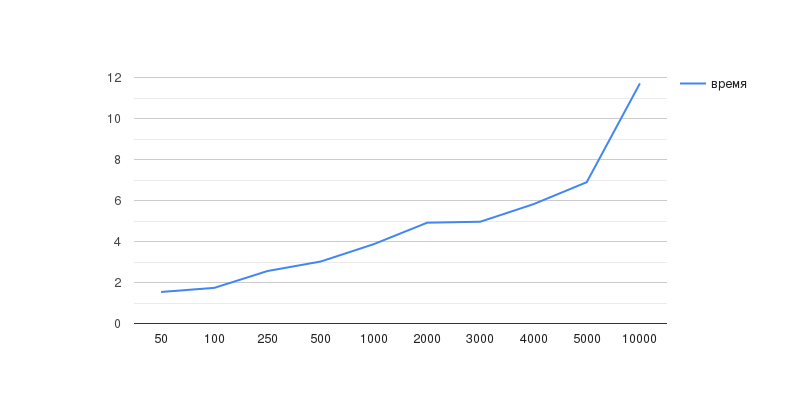
1. Удаление

**Временная сложность процесса удаления B-дерева** Предоставлено QuickLaTeX.com.



3)Вставка

**Временная сложность процесса вставки B-дерева** Предоставлено QuickLaTeX.com.



*Высота*

Количество обращений к диску, необходимое для выполнения большинства операций с В-деревом, пропорционально его высоте. Проанализируем высоту В-дерева в наихудшем случае.

Теорема:

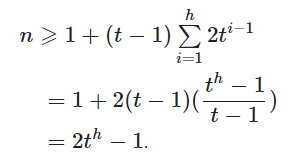
Если n⩾1, то для B-дерева T c n узлами и минимальной степенью t⩾2 имеется следующее неравенство:

h⩽logt((n+1)/2)

Доказательство:

▹

Корень B-дерева T содержит по меньшей мере один ключ, а все остальные узлы — хотя бы t−1 ключей. Так, T, высота которого h, имеет хотя бы 2 узла на глубине 1, хотя бы 2t узла на глубине 2, хотя бы 2t^2 узла на глубине 3, и так далее, до глубины h, оно имеет по меньшей мере 2t^h−1 узлов. Так, число ключей n удовлетворяет неравенству:



.

Простейшее преобразование дает нам неравенство t^h⩽(n+1)/2. Логарифмирование по основанию t обеих частей неравенства доказывает теорему

◃

Здесь мы видим преимущества B-деревьев над красно-черными деревьями. Хотя высота деревьев растет как O(logt) в обоих случаях (вспомним, что t — константа), в случае B-деревьев основание логарифмов имеет гораздо большее значение. Таким образом, В-деревья требуют исследования примерно в logt раз меньшего количества узлов по сравнению с красно-черными деревьями в большинстве операций. Поскольку исследование узла дерева обычно требует обращения к диску, количество дисковых операций при работе с В-деревьями оказывается существенно сниженным.

Достоинства и недостатки

-Во всех случаях полезное использование пространства вторичной памяти составляет свыше 50 %. С ростом степени полезного использования памяти не происходит снижения качества обслуживания.

-Произвольный доступ к записи реализуется посредством малого количества подопераций (обращения к физическим блокам).

-В среднем достаточно эффективно реализуются операции включения и удаления записей; при этом сохраняется естественный порядок ключей с целью последовательной обработки, а также соответствующий баланс дерева для обеспечения быстрой произвольной выборки.

-Неизменная упорядоченность по ключу обеспечивает возможность эффективной пакетной обработки.

Основной недостаток В-деревьев состоит в отсутствии для них эффективных средств выборки данных (то есть метода обхода дерева), упорядоченных по свойству, отличному от выбранного ключа.

Примение

Вторичные запоминающие устройства (жесткие диски, SSD) медленно работают с большим объемом данных. Людям захотелось сократить время доступа к физическим носителям информации, поэтому возникла потребность в таких структурах данных, которые способны это сделать.

Двоичное дерево поиска, АВЛ-дерево, красно-черное дерево и т. д. могут хранить только один ключ в одном узле. Если нужно хранить больше, высота деревьев резко начинает расти, из-за этого время доступа сильно увеличивается.

С B-деревом все не так. Оно позволяет хранить много ключей в одном узле и при этом может ссылаться на несколько дочерних узлов. Это значительно уменьшает высоту дерева и, соответственно, обеспечивает более быстрый доступ к диску.