Что такое b-дерево

B-дерево (читается как Би-дерево) — это особый тип сбалансированного дерева поиска, в котором каждый узел может содержать более одного ключа и иметь более двух дочерних элементов. Из-за этого свойства B-дерево называют сильноветвящимся.

B-дерево порядка m обладает следующими свойствами:

* Каждый узел, кроме корня, содержит не менее t−1 ключей, и каждый внутренний узел имеет по меньшей мере t дочерних узлов. Если дерево не является пустым, корень должен содержать как минимум один ключ.
* Каждый узел, кроме корня, содержит не более 2t−1ключей и не более чем 2t детей во внутренних узлах
* Корень содержит от 1 до 2t−1 ключей, если дерево не пусто и от 2 до 2t детей при высоте большей 0.
* Каждый узел дерева, кроме листьев, содержащий ключи k1,...,kn имеет n+1 сына. i-й сын содержит ключи из отрезка [ki−1;ki],k0=−∞,kn+1= +∞
* Ключи в каждом узле упорядочены по неубыванию.
* Все листья находятся на одном уровне.

**Зачем нужно**

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.

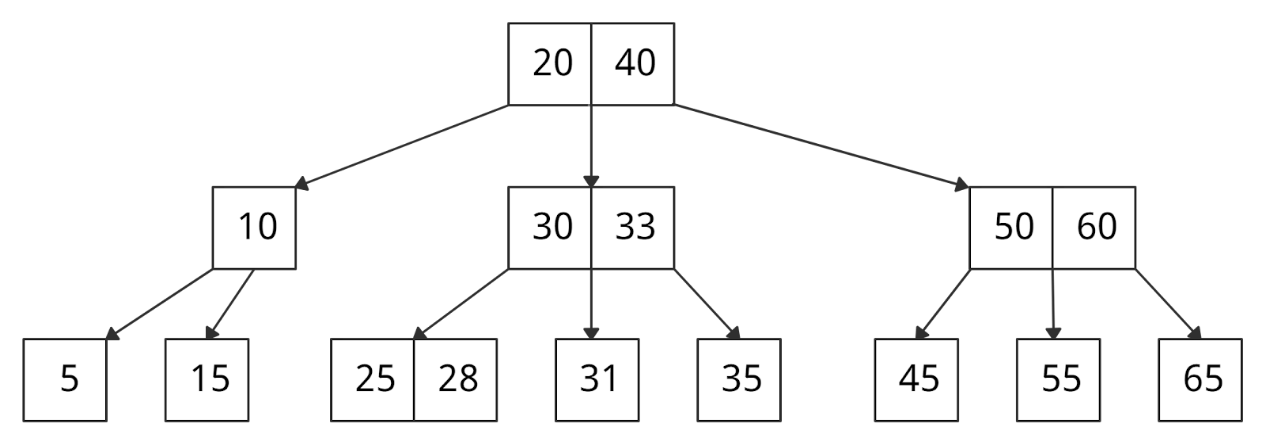
Двоичное дерево поиска, АВЛ-дерево, красно-черное дерево и т. д. могут хранить только один ключ в одном узле. Если нужно хранить больше, высота деревьев резко начинает расти, из-за этого время доступа сильно увеличивается.

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.

# B-дерево

###### В этом руководстве вы узнаете, что такое B-дерево и как его можно реализовать на C, C ++, Java и Python.

**B-дерево** (читается как Би-дерево) — это особый тип сбалансированного дерева поиска, в котором каждый узел может содержать более одного ключа и иметь более двух дочерних элементов. Из-за этого свойства B-дерево называют сильноветвящимся.



#### Зачем нужно

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.

#### 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.

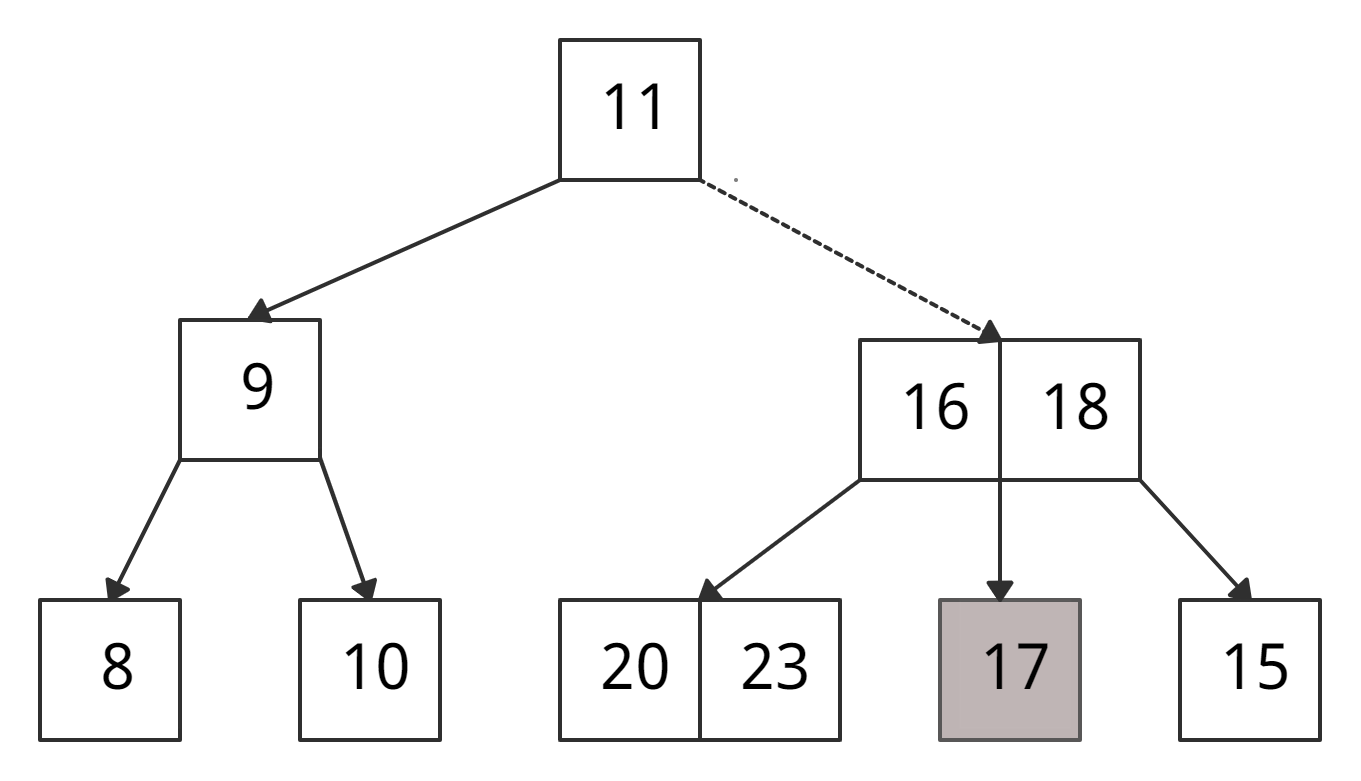
## Операции с B-деревом

### Поиск элемента

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

Поиск ключа в B-дереве работает так же, как и в двоичном дереве поиска.

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.



**Операция вставки в B-дерево**

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

1. 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:
2. *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.

## Операция удаления

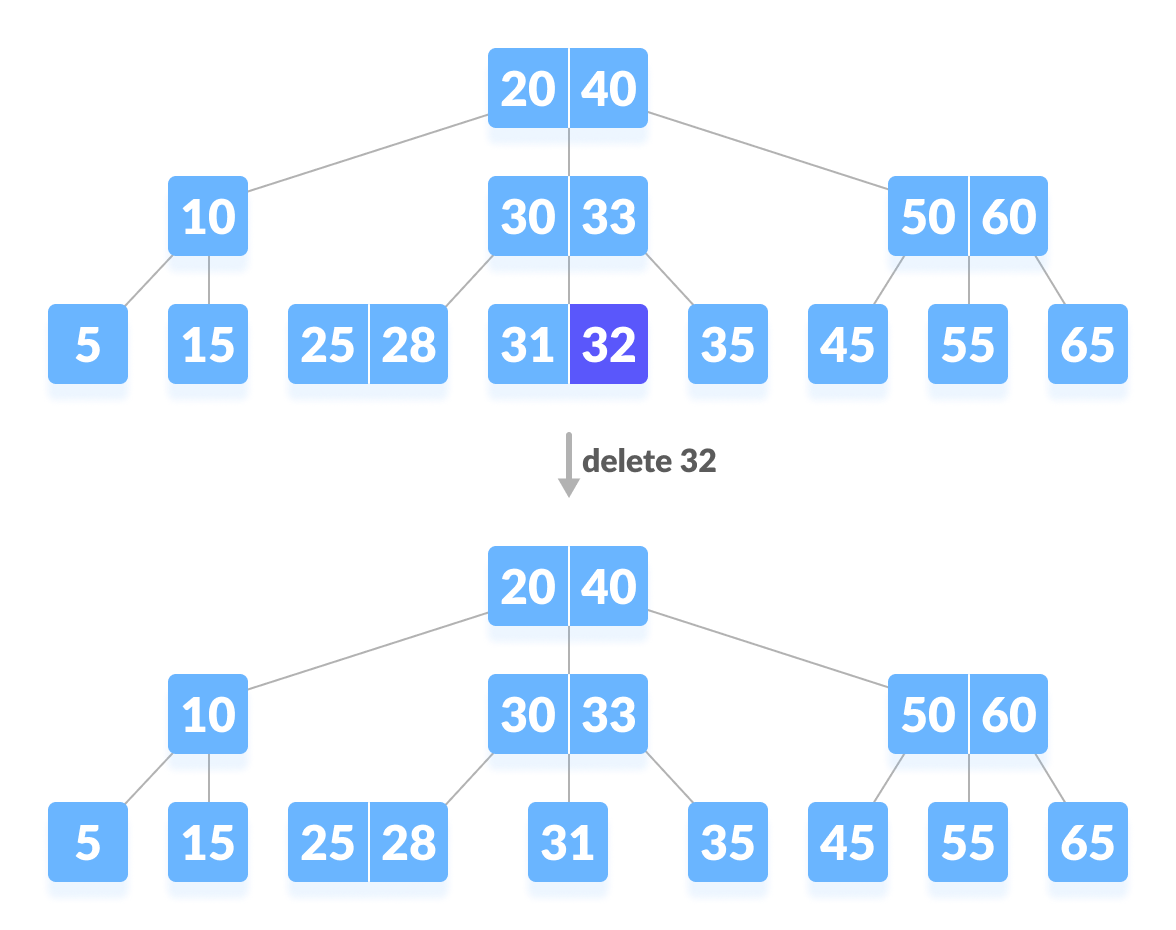
**Средняя временная сложность:** Θ(log n)  
**Худшая временная сложность:** Θ(log n)

Существует три основных варианта операции удаления в B- дереве.

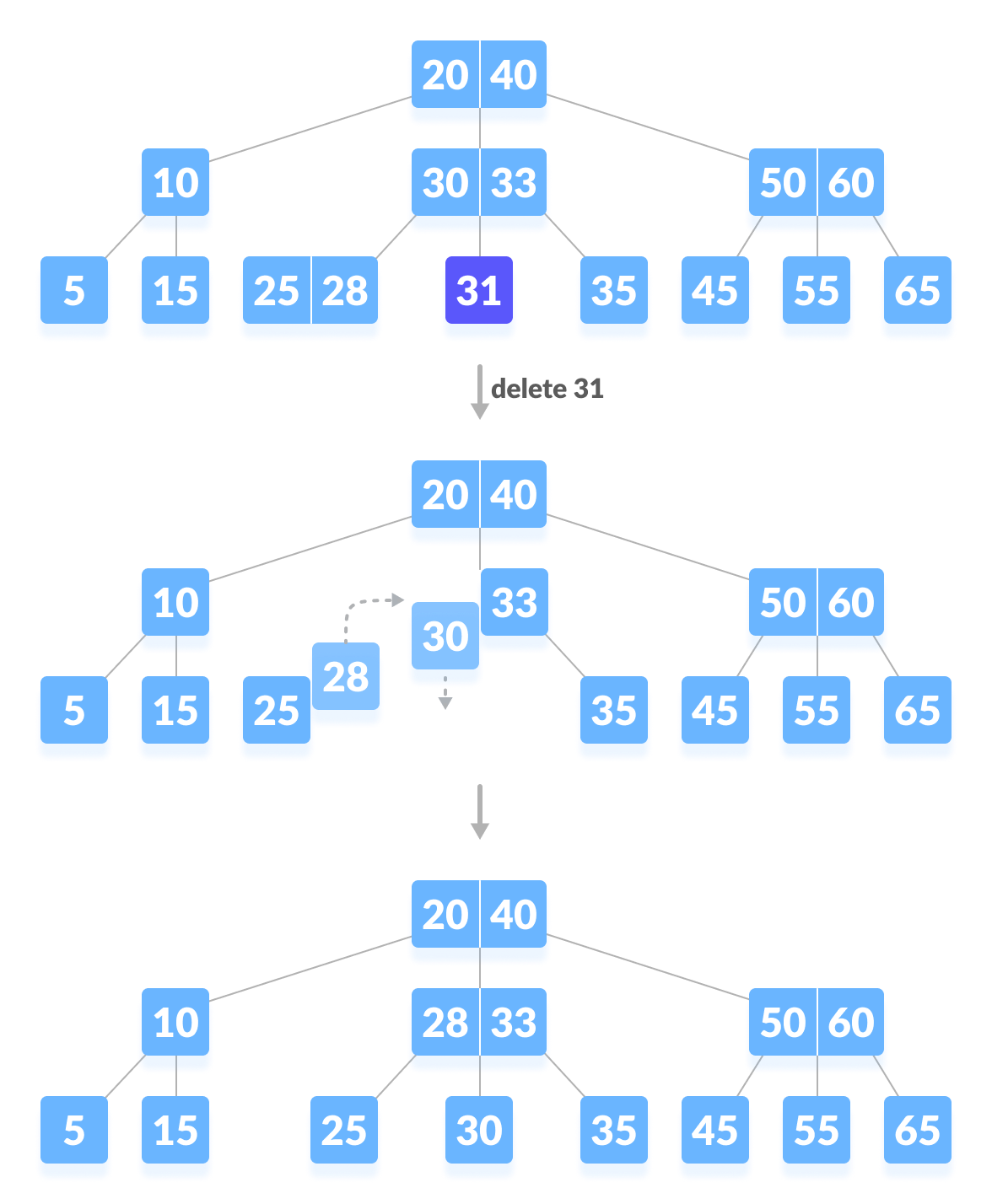
### Случай I

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

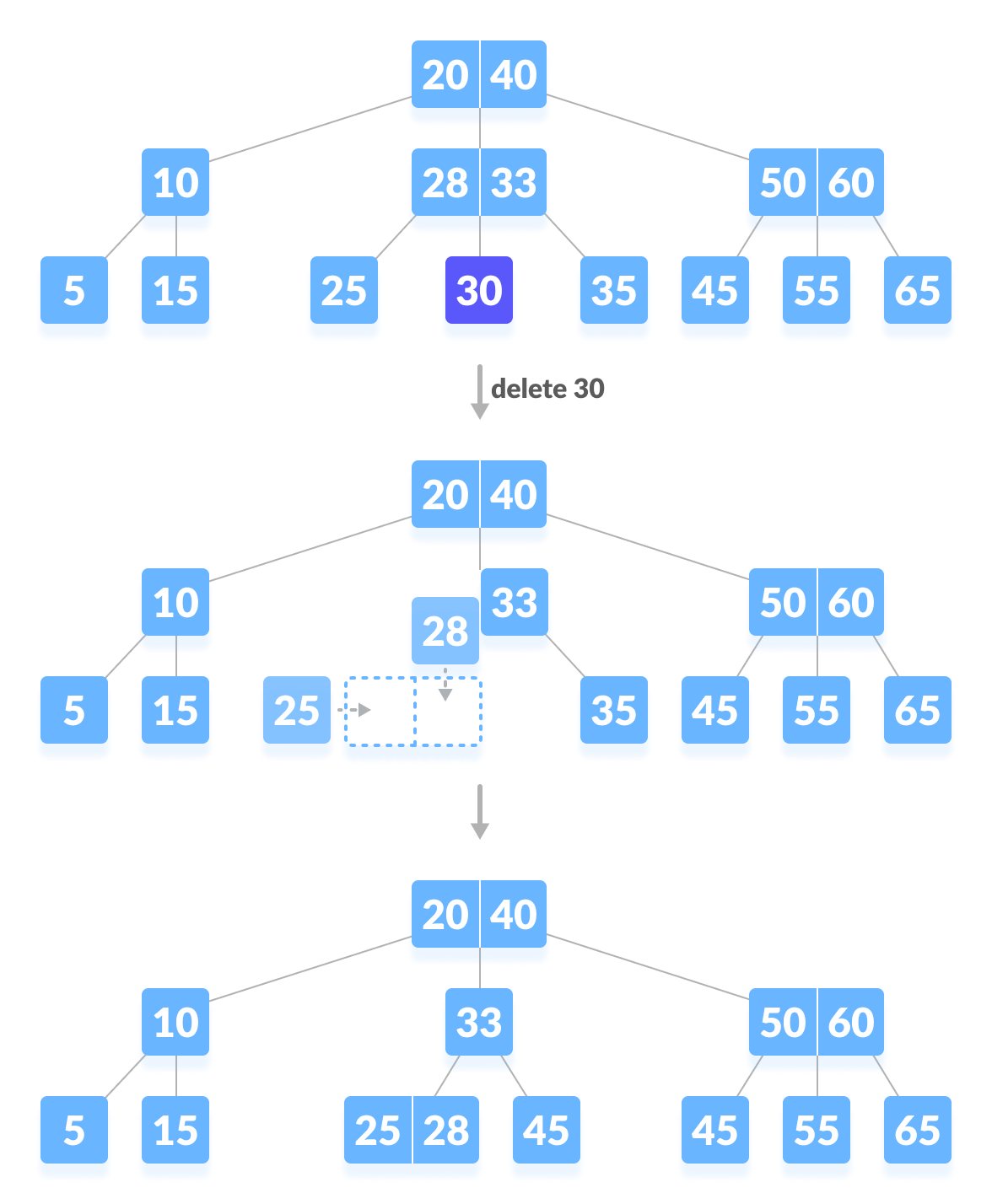
1. Удаление ключа не нарушает свойство минимального количества ключей, которое должен содержать узел.



1. Удаление ключа нарушает свойство минимального количества ключей, которое должен содержать узел. В этом случае мы заимствуем ключ из его ближайшего соседнего дочернего узла в порядке слева направо.  
     
   Сначала посетите ближайший левый дочерний узел. Если у левого дочернего узла больше минимального количества ключей, то позаимствуйте ключ у этого узла.  
     
   В противном случае проверьте, чтобы оно заимствовалось из ближайшего правого дочернего узла.



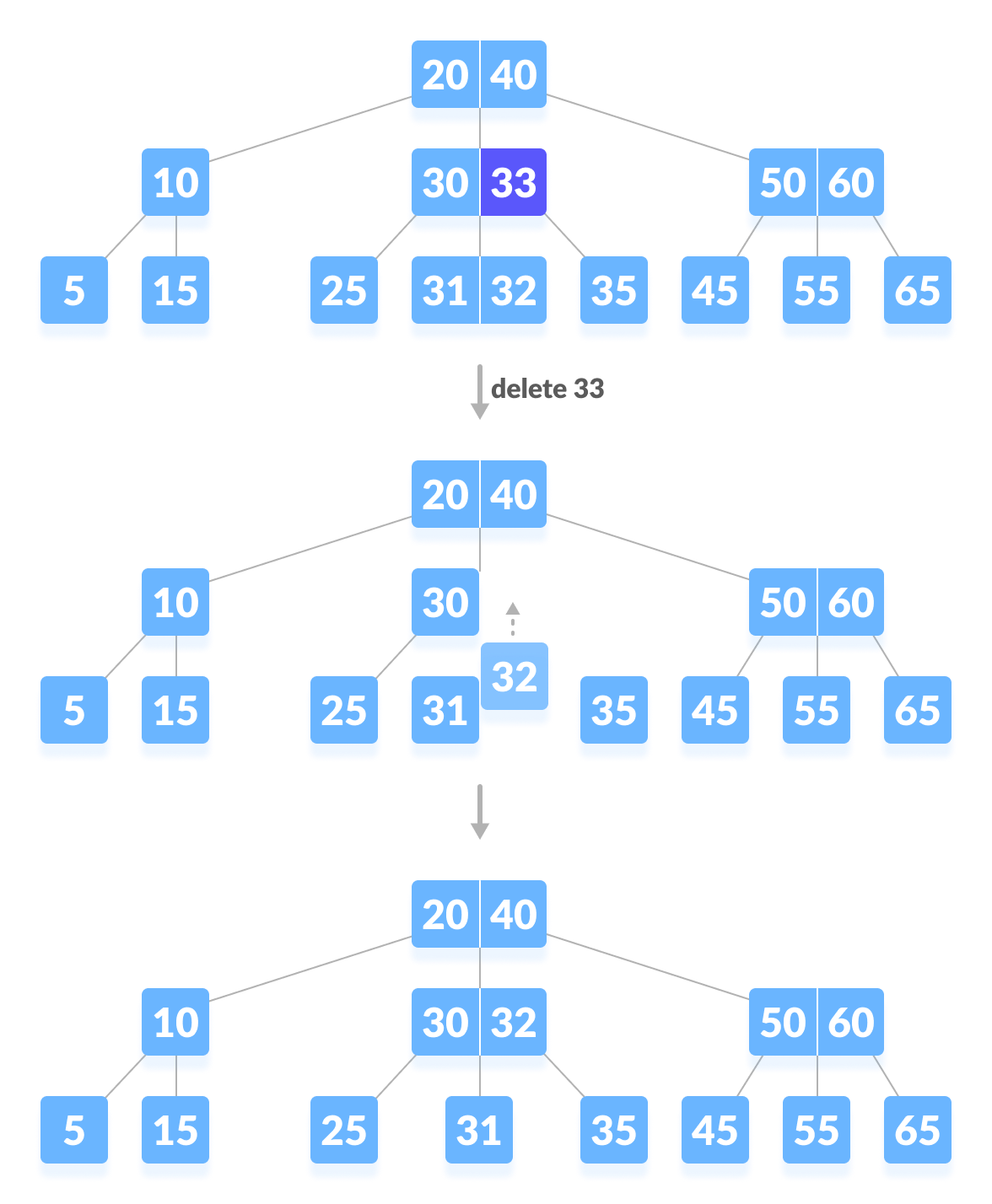
Если оба ближайших дочерних узла уже имеют минимальное количество ключей, то объедините узел либо с левым дочерним узлом, либо с правым дочерним узлом. **Это объединение выполняется через родительский узел.**



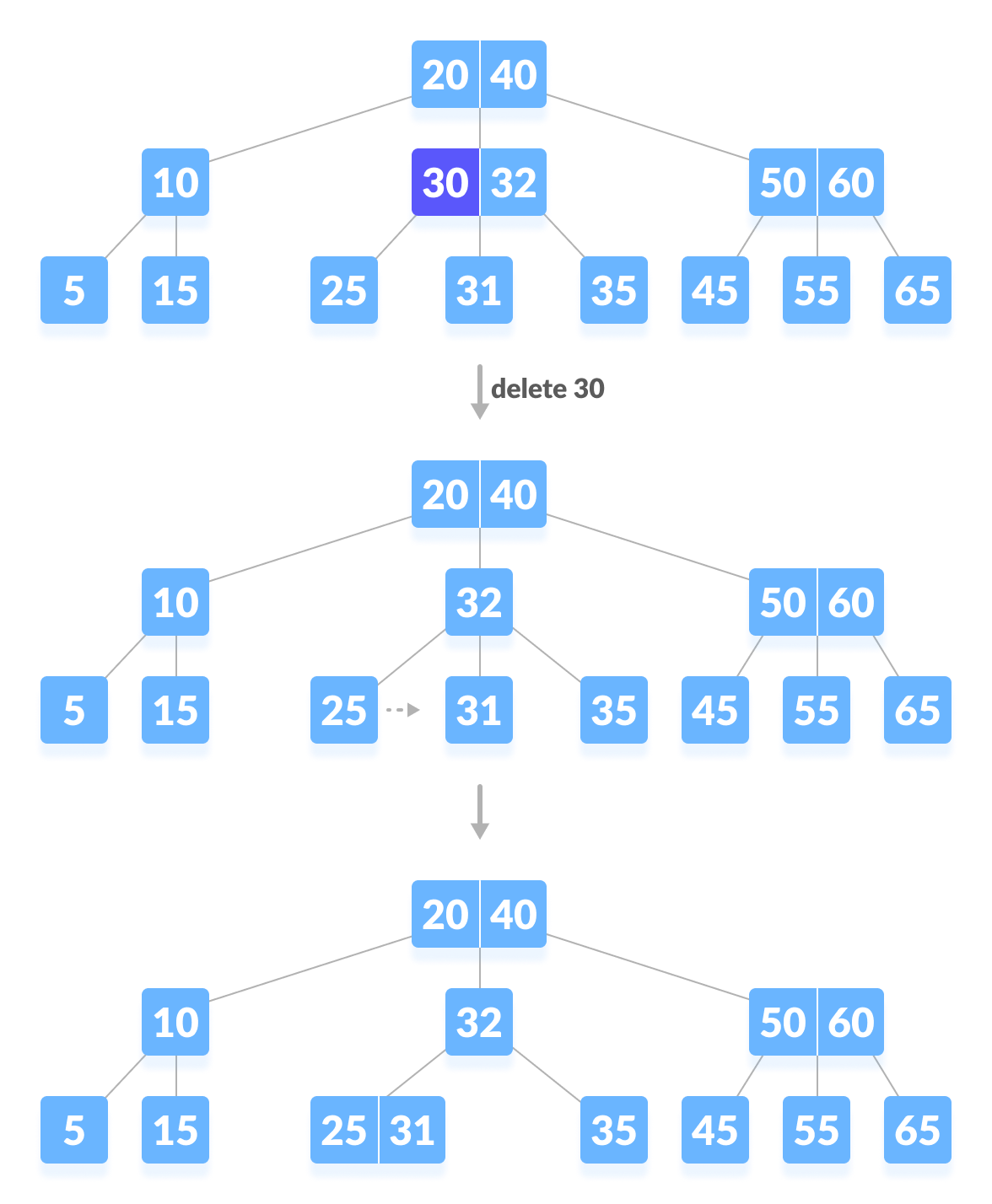
### Случай II

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

1. Внутренний узел, который удаляется, заменяется предшественником по порядку, если у левого дочернего узла больше минимального количества ключей.



1. Внутренний узел, который удаляется, заменяется преемником по порядку, если у нужного дочернего узла больше минимального количества ключей.
2. Если у любого из дочерних элементов есть точно минимальное количество ключей, объедините левый и правый дочерние элементы.

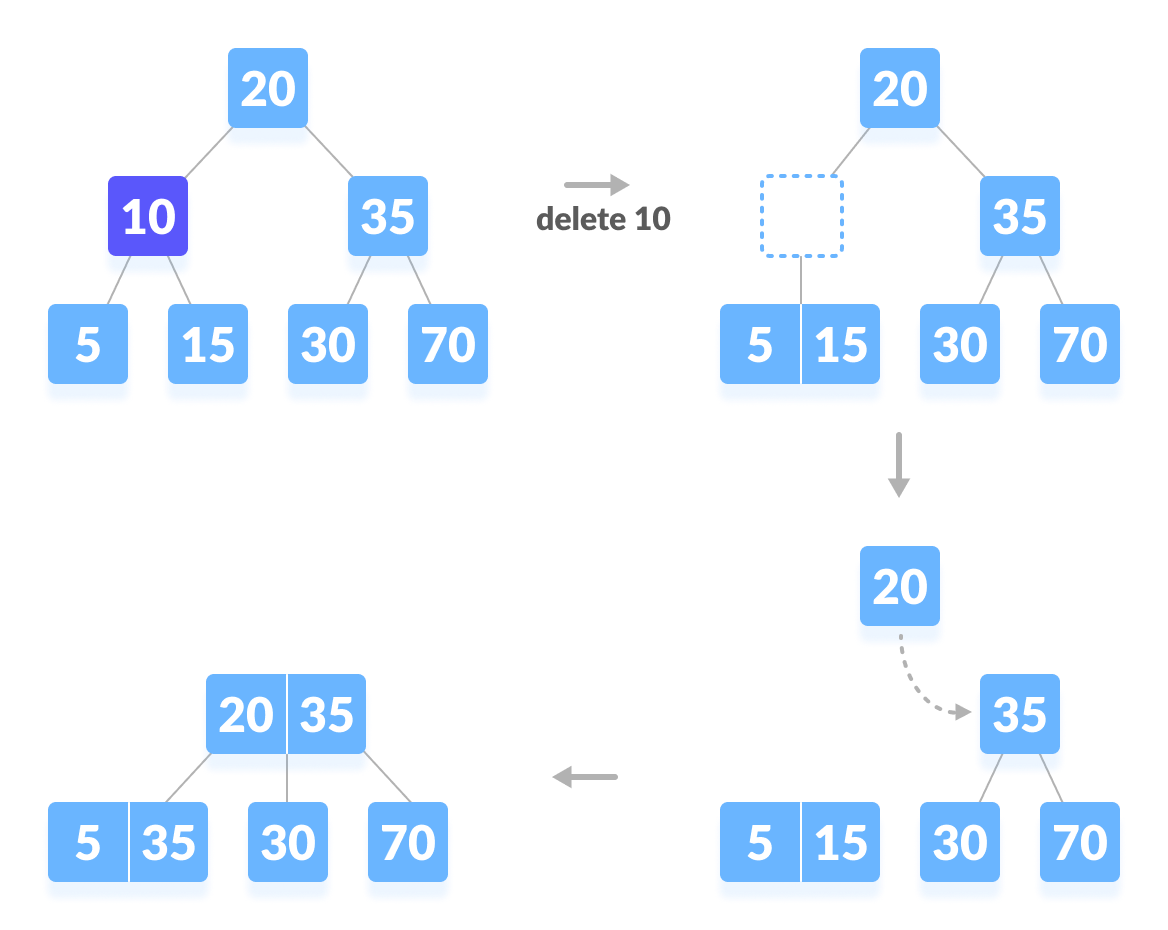


1. После слияния, если родительский узел имеет меньше минимального количества ключей, тогда ищите братьев и сестер, как в случае I.

### Пример III

В этом случае высота дерева уменьшается. Если целевой ключ находится во внутреннем узле, и удаление ключа приводит к уменьшению количества ключей в узле (т.е. Меньше минимально необходимого), тогда ищите предшественника по порядку и преемника по порядку. Если оба дочерних элемента содержат минимальное количество ключей, то заимствование не может иметь места. Это приводит к случаю II (3), т.е. объединению дочерних элементов.

Опять же, ищите родственного узла, 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).



## **Вариации B-дерева**

### B+-дерево

### B\*-дерево

### 2-3 дерево