

# B Plus Tree

8 января 2023 г.

## 1 Вступление

### 1.1 Определение

Для того чтобы разобраться с концепцией B+-дерева, сначала стоит подробнее изучить понятие стандартного B-дерева.

B-дерево представляет собой строго сбалансированную древовидную структуру данных, которая поддерживает отсортированные данные и разрешает поиск, последовательный доступ, вставку и удаление в логарифмическом времени. B-дерево обобщает двоичное дерево поиска, позволяя использовать узлы с более чем двумя дочерними элементами.

Дерево является B-деревом порядка  $m$ , если обладает следующими свойствами:

- B-дерево является идеально сбалансированным, то есть глубина всех его листьев одинакова.
- Все узлы, кроме корня должны иметь как минимум  $\lceil m/2 \rceil - 1$  ключей и максимум  $m-1$  ключей.
- Все узлы без листьев, кроме корня (т.е. все внутренние узлы), должны иметь минимум  $\lceil m/2 \rceil$  потомков.
- Если корень – это узел не содержащий листьев, он должен иметь минимум 2 потомка.
- Узел без листьев с  $m-1$  ключами должен иметь  $m$  потомков.
- Левое поддерево узла будет иметь меньшие значения, чем правая сторона поддерева. Это означает, что узлы также сортируются в порядке возрастания слева направо.

B + -дерево является модификацией B-дерева. В B+-дереве настоящие ключи хранятся лишь в листьях дерева, а во внутренних узлах хранятся лишь ключи-маршрутизаторы, необходимые для поиска по дереву.

Кроме свойств, наследуемых от B-дерева, B+-дерево также имеет следующие характеристики, позволяющие эффективно выполнять операции над ним:

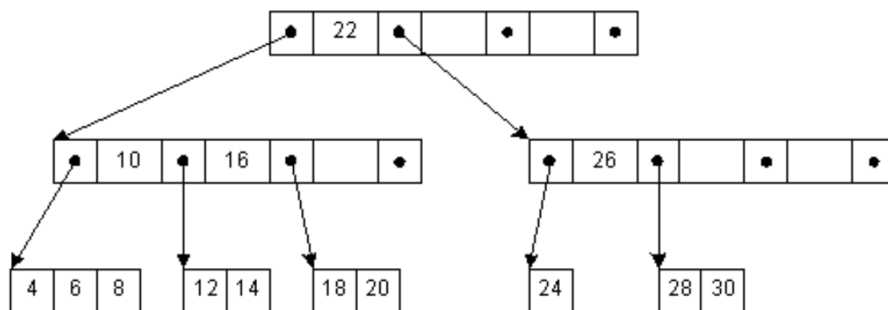


Рис. 1: Дерево B порядка  $t=4$

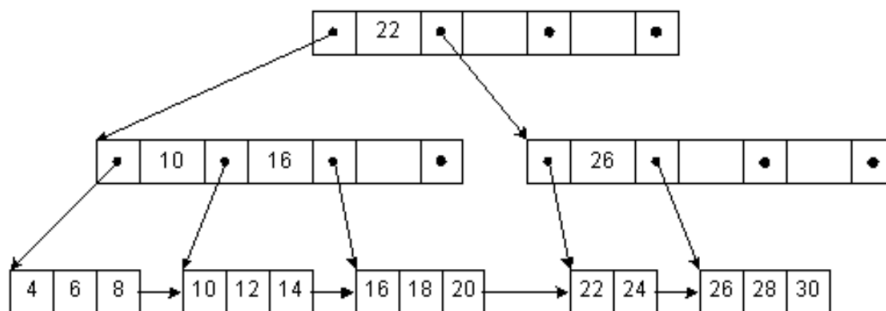


Рис. 2: Дерево B+ порядка  $t=4$

- В дереве B+ записи (указатели на данные) могут храниться только на листовых узлах, а внутренние узлы могут хранить только копии ключей (значения без указателя на данные). Здесь можно отметить, что, поскольку указатели данных присутствуют только в листовых узлах, конечные узлы обязательно должны хранить все значения ключей вместе с соответствующими указателями данных на блок дискового файла, чтобы получить к ним доступ.
- Листовые узлы дерева B+ связаны друг с другом в виде односвязных списков, чтобы сделать поисковые запросы более эффективными.

## 1.2 История

В-деревья были изобретены Рудольфом Байером и Эдвард МакКрейт во время работы в Boeing Research Labs, для цели эффективного управления страницей индексов для больших файлов с произвольным доступом. Основное предположение заключено в том, что индексы могут быть настолько объемными, что в основную память поместиться лишь небольшие фрагменты дерева. Статья Байера и МакКрайта «Организация и поддерж-

ка упорядоченных индексов» была впервые распространена в июле 1970 года и позже опубликована в *Acta Informatica*.

Байер и МакКрайт никогда не объясняли, что означает буква В: Boeing, сбалансированный, широкие, пушистые и байеровские. МакКрайт сказал, что «чем больше вы думаете о том, что означает В в В-деревьях, тем лучше вы понимаете В-деревья».

Одной из первых статей, в которой были подробно разобраны концепции модификаций В-дерева (в том числе и В+-дерева), является статья Дугласа Комера "Вездесущее В-дерево". В этой же работе он отмечает, что В+-дерево использовалось в программном обеспечении IBM для доступа к данным VSAM, и ссылается на опубликованную в журнале *IBM System Journal* статью 1973 года "Рекомендации по дизайну индексации".

### 1.3 Применение

Изначально структура В+-деревьев предназначалась для хранения данных в целях эффективного поиска в блочно-ориентированной среде хранения — в частности, для файловых систем; применение связано с тем, что в отличие от бинарных деревьев поиска, В+-деревья имеют очень высокий коэффициент ветвления (число указателей из родительского узла на дочерние, обычно порядка 100 или более), что снижает количество операций ввода-вывода, требующих поиска элемента в дереве.

Структура широко применяется в файловых системах — NTFS, ReiserFS, NSS, XFS, JFS, ReFS используют этот тип дерева для индексирования метаданных; BeFS также использует В+-деревья для хранения каталогов. Реляционные системы управления базами данных, такие как DB2, Informix, Microsoft SQL Server, Oracle Database (начиная с версии 8), Adaptive Server Enterprise и SQLite поддерживают этот тип деревьев для табличных индексов. Среди NoSQL-СУБД, работающих с моделью «ключ — значение», структура данных реализована для доступа к данным в CouchDB и Tokyo Cabinet.

## 2 Описание алгоритма

### 2.1 Асимптотическая сложность

В+-деревья являются сбалансированными, поэтому время выполнения стандартных операций в них пропорционально высоте, то есть  $O(\log n)$ . Однако стоит заметить, что так как степень дерева зачастую выбирается большой, константа при выполнении операций тоже большая. Это связано с большим количеством ключей в узлах, которые необходимо сравнить. Но из-за небольшой высоты дерева это не сильно сказывается на скорости работы

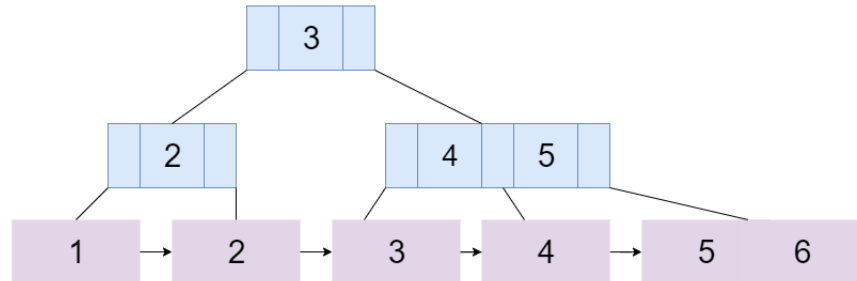


Рис. 3:

## 2.2 Поиск

*Поиск листа.* Для того, чтобы найти нужный ключ, необходимо сначала добраться до листового узла, который может его содержать. С помощью бинарного или линейного поиска ищется интервал содержащий значение ключа и осуществляется переход к соответствующему сыну. Операция повторяется пока текущий узел не окажется конечным(листовым). Функция возвращает лист, содержащий ключ. В противном случае пользователю показывается неудачная попытка.

*Поиск ключа.* Производится поиск листа и последующий поиск искомого ключа в нем. Программа выдает индекс искомого ключа и показывает неудачную попытку в противном случае.

Далее будет разобран пример поиска ключа  $k=4$  в B+-дереве порядка  $m=3$ .

Необходимо сравнить  $k=4$  с корневым узлом. Так как  $k > 3$ , осуществляется переход к правому потомку.

Необходимо найти нужный интервал. Так как  $k \geq 4$ , значение ключа сравнивается с следующим элементом:  $k \leq 5$ . Значит, осуществляется переход в подинтервал  $[4; 5]$  и производится поиск единственного ключа, соответствующего искомому. Операция завершена.

## 2.3 Добавление

Если дерево пустое, нужно создать новый корень и добавить первый ключ.

Иначе производится поиск листа, в который можно добавить ключ. Ключ добавляется в список в порядке неубывания. Если лист не заполнен, операция завершена. В случае, если лист заполнен, последующим действием после вставки выполняется *разбиение узла*.

Узел делится на две половины. Элемент  $m/2+1$  переносится в родительский узел. Если родительский узел оказывается заполненным, повторяется *разбиение узла*.

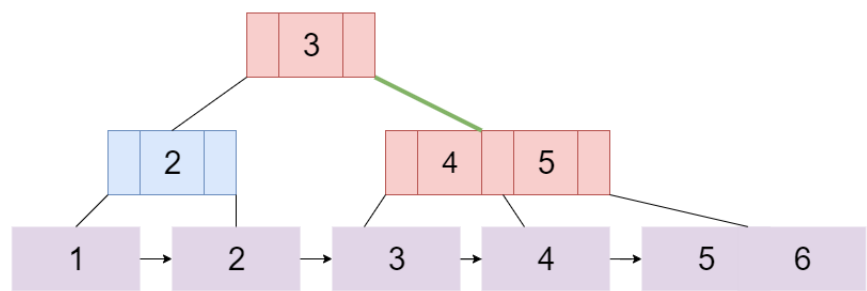


Рис. 4:

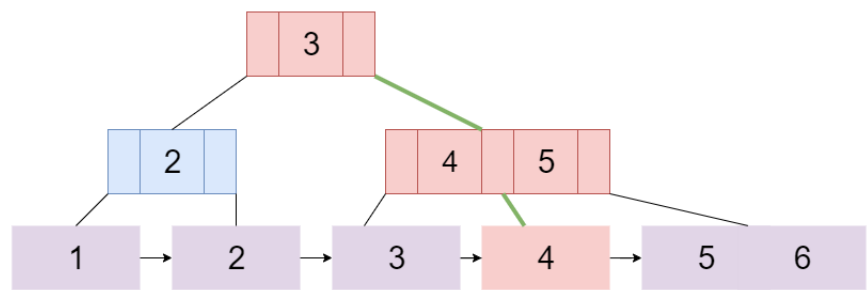


Рис. 5:

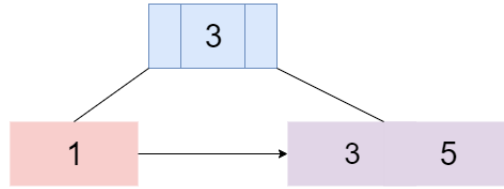


Рис. 6:

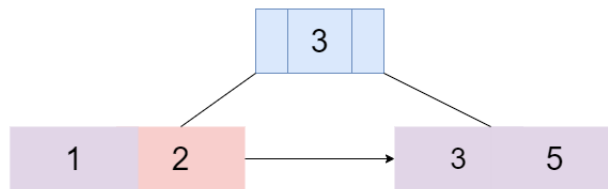


Рис. 7:

Далее на примере дерева порядка  $m=3$  будут рассмотрены два случая: когда количество элементов узла меньше  $m+1$  и когда происходит переполнение узлов.

*1 случай.* Производится поиск нужного листа и добавление  $k=2$  в порядке возрастания.

При добавлении количество элементов не превышает максимальное. Значит, операция завершается.

*2 случай.* Производится поиск нужного листа и добавление  $k=6$  в порядке возрастания.

Сейчас в текущем листе содержится слишком много ключей, поэтому необходимо разбить его на два. Берется ключ под номером  $\lfloor k/2 + 1 \rfloor$  со значением 5 и его копия помещается в родительский узел.

В левый узел помещается  $\lfloor n/2 \rfloor$ , а в правый  $\lfloor n/2 + 1 \rfloor$  ключей (где  $n$  - изначальное количество ключей).

В родительском узле количество ключей не превышает норму, поэтому операция завершается.

## 2.4 Удаление

Поскольку все ключи находятся в листах, для удаления в первую очередь необходимо найти листовой узел, в котором он находится, а затем и сам

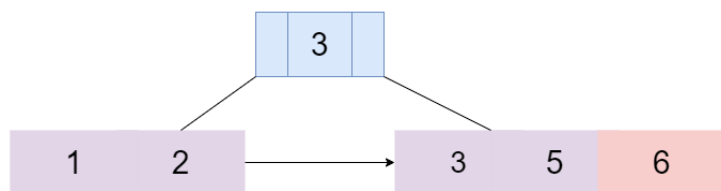


Рис. 8:

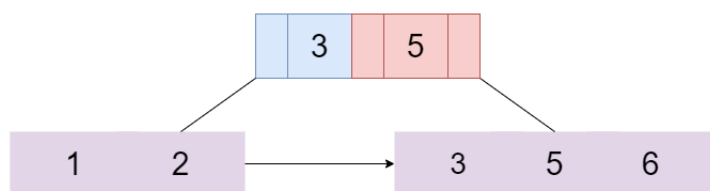


Рис. 9:

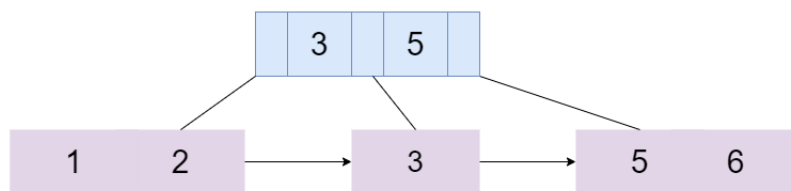


Рис. 10:

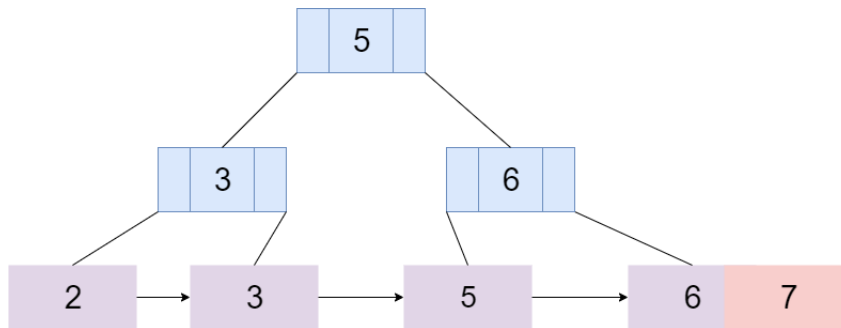


Рис. 11:

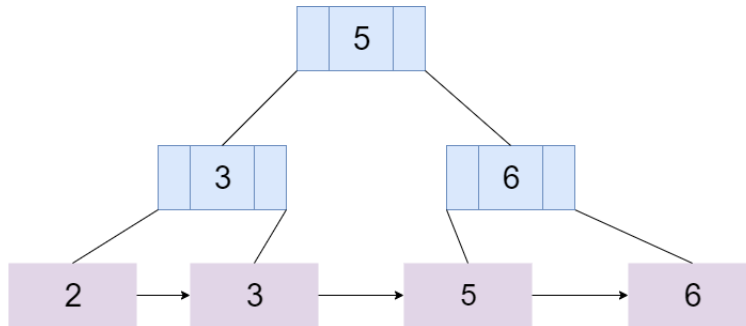


Рис. 12:

ключ. При нахождении нужного ключа он и связанная с ним ссылка удаляются. Если узел содержит минимальное количество ключей операция завершается. В противном случае нужно провести балансировку дерева.

Для этого производится попытка перераспределения элементов, то есть добавления в узел элемент из левого или правого брата (не забыв обновить информацию в родителе). Если это невозможно, необходимо выполнить слияние с братом и удалить ключ, который указывает на удалённый узел. Объединение может распространяться на корень, тогда происходит уменьшение высоты дерева.

Далее будут разобраны три случая.

*1 случай.* Ключ, который нужно удалить, присутствует только в листовом узле, а не во внутренних узлах. Для этого есть два случая:

1. Удаляется ключ со значением  $k=7$ .

Так количество ключей в узле соответствует минимальному, операция завершается.

2. Удаляется ключ  $k=5$ . При удалении в узле количество ключей становится меньше минимального.

Добавляется срединный ключ потомка к родителю.

Недостающий заимствуется ключ у брата.



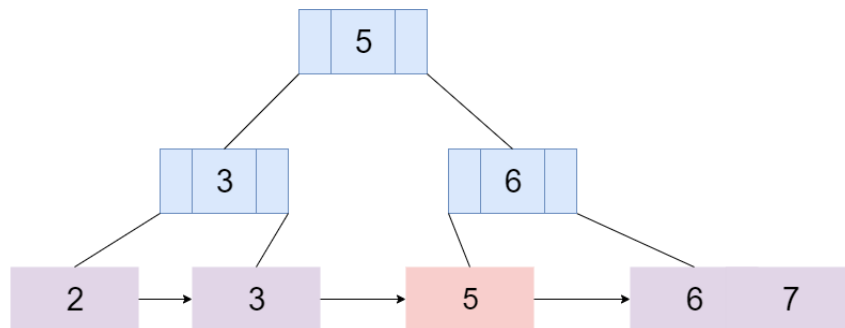


Рис. 13:

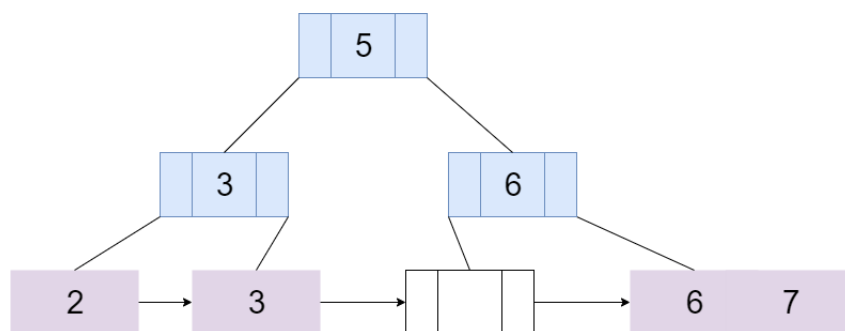


Рис. 14:

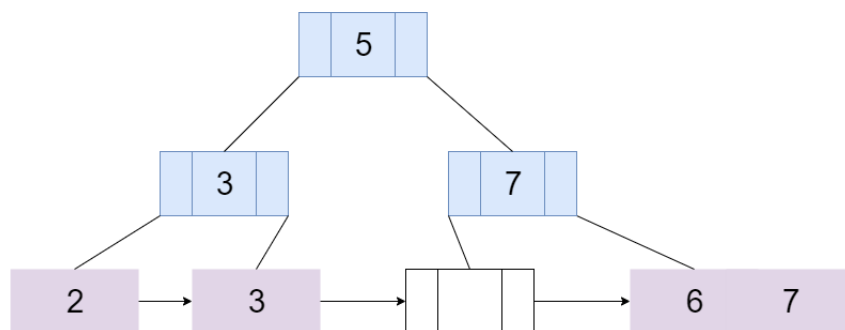


Рис. 15:

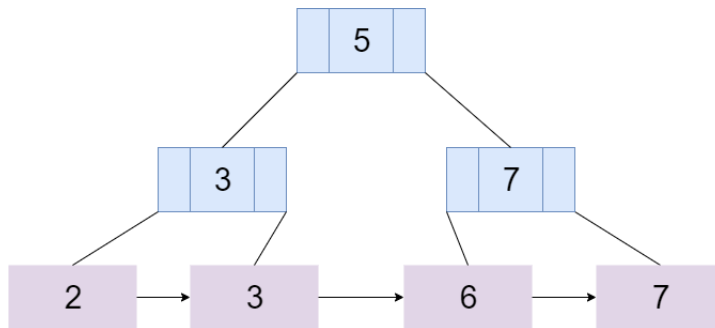


Рис. 16:

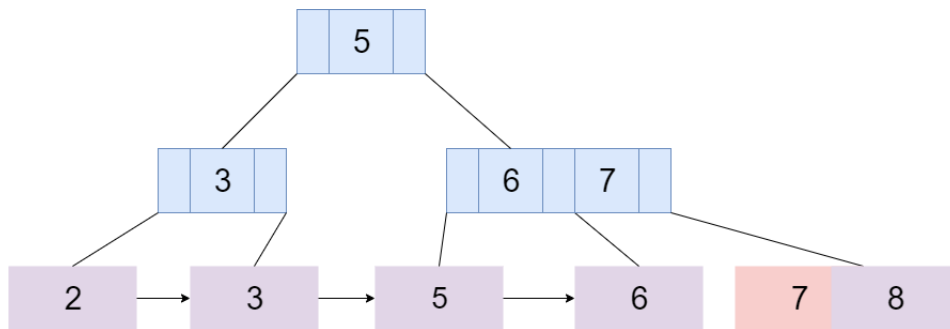


Рис. 17:

*2 случай.* Ключ, который нужно удалить, присутствует и во внутренних узлах. Имеются следующие случаи для этой ситуации.

1. Удаляется ключ  $k=7$ .

Если количество ключей в узле больше минимального, ключ просто удаляется из листового и внутреннего узла.

Пустое место во внутреннем узле заполняется преемником по порядку.

2. Удаляется ключ  $k=3$ .

Ключ удаляется и новый заимствуется у его родственного узла, если в нем после этого будет минимальное количество ключей.

Пустые узлы заполняются заимствованным ключом.

*3 случай.* Удаляется ключ  $k=8$ , который содержится в листовом и внутреннем узле. Позаимствовать ключи у соседних братьев невозможно, так как тогда их количество ключей будет меньше минимального.

Создается пустое пространство, ссылки на которое в последующем удаляются.

Далее продельвается слияние корня и ближайшего незаполненного ребенка.

Создаются новые ссылки, старые удаляются.

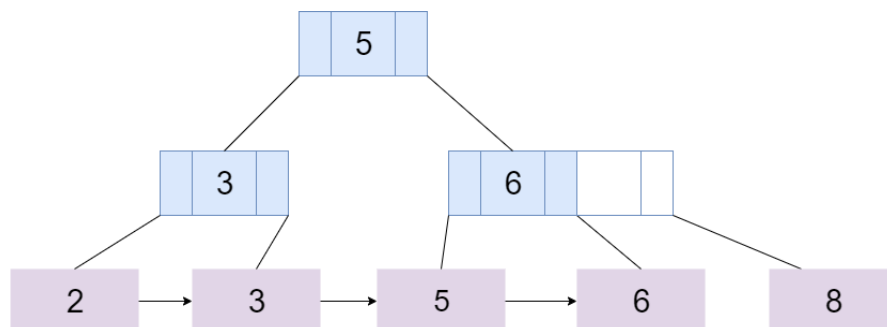


Рис. 18:

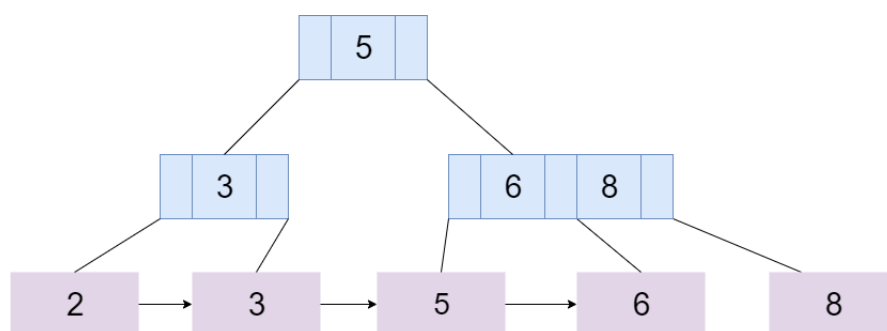


Рис. 19:

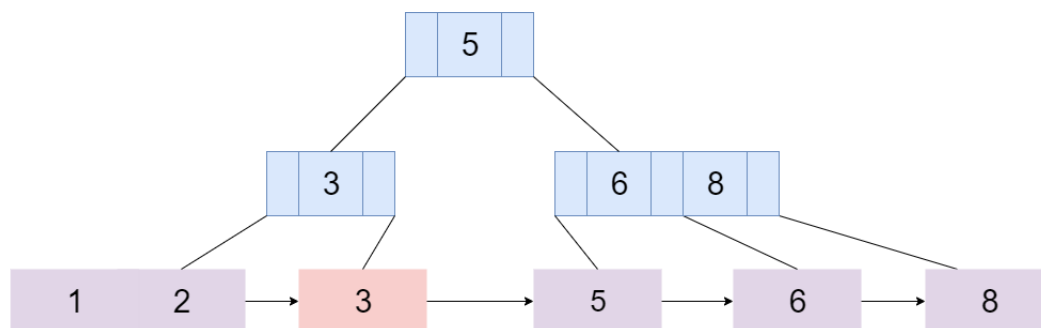


Рис. 20:

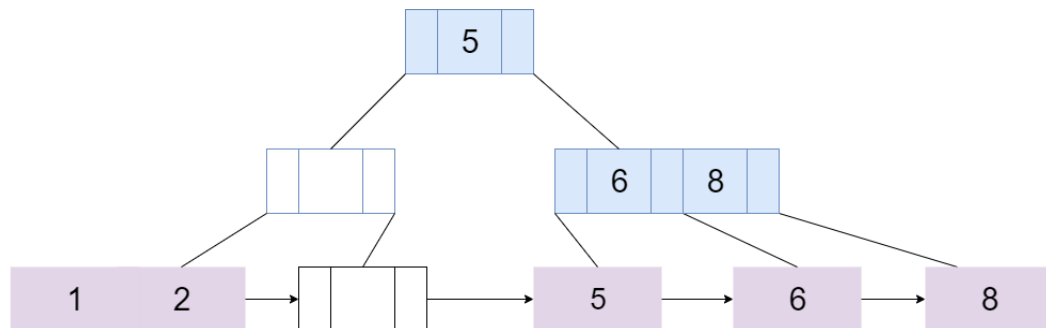


Рис. 21:

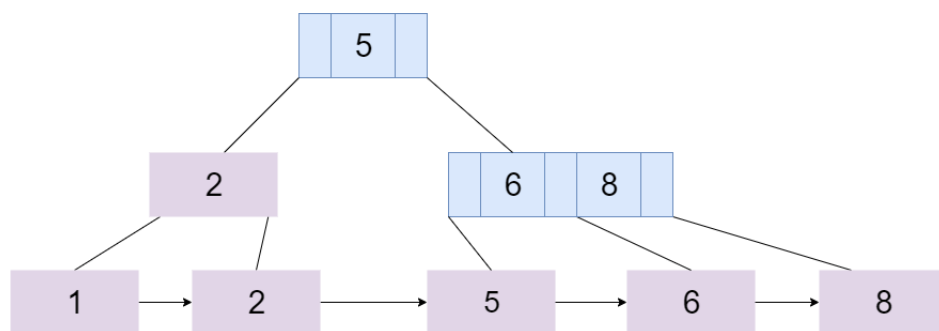


Рис. 22:

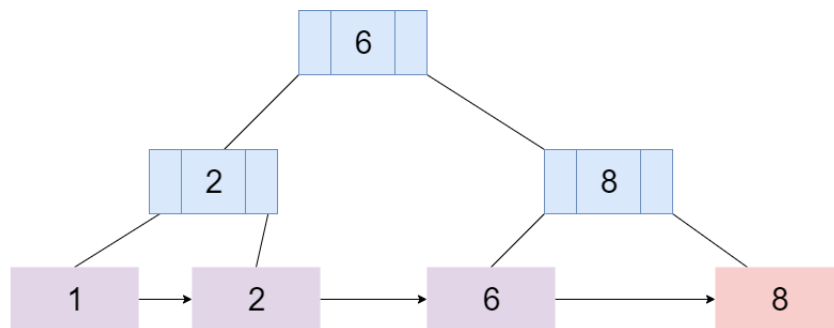


Рис. 23:

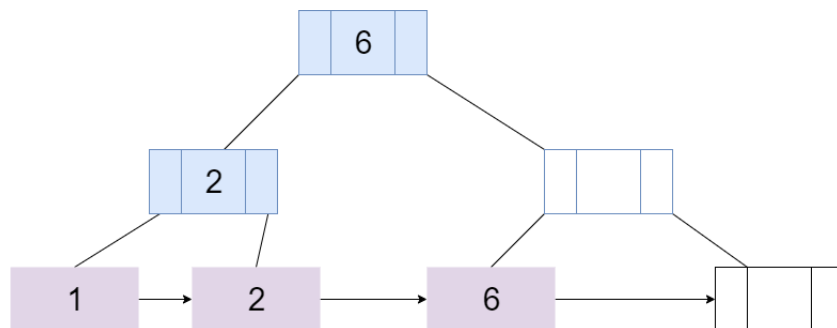


Рис. 24:

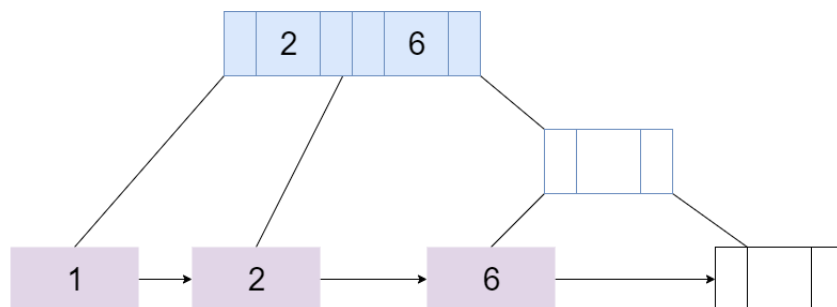


Рис. 25:

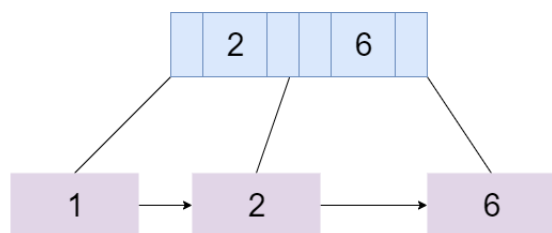


Рис. 26:

### 3 Постановка задачи

#### Условие задачи

Необходимо реализовать  $m$ -арное дерево поиска, удовлетворяющее следующим условиям:

- Дерево строго сбалансировано (каждая ветвь имеет одну высоту)
- Каждый узел имеет  $m$  дочерних элементов и  $m-1$  ключей или их копий
- Во внутренних узлах хранятся только копии ключей, сами ключи лежат в листьях. Листовые узлы представляют собой односвязный список

А также выполняемые следующие операции над собой:

- `insert(int k)` - добавляет ключ  $k$  в дерево
- `delete(int k)` - удаляет ключ  $k$  из дерева
- `search(int k)` - ищет ключ по значению и в случае успеха возвращает `True`, иначе - `False`
- `print()` - обходит каждый узел начиная с левого поддерева и выводит построчно списки ключей или их копий.

#### Формат входных данных:

В первой строке входные данные имеют число  $m$  (степень дерева). Далее пользователем в консоль вводятся команды в формате строки. Вызов операций вставки, удаления и поиска имеет вид `{"название команды" k}`, где  $k$  - ключ. Чтобы программа завершилась, необходимо ввести в консоль `"end"`.

#### Формат выходных данных:

После того как пользователь вызовет команду завершения, в консоли отобразятся ключи каждого узла, начиная с левого поддерева.

#### Ограничения:

$$2 \leq m \leq 100$$
$$-2^{31} < k < 2^{31}$$

### 4 Заключение

Таким образом,  $B^+$ -деревья являются актуальной структурой данных, с помощью которой можно эффективно и последовательно обрабатывать большие объемы данных с сохранением логарифмической сложности.

## 5 Список литературы

1. B.Bayer, E.McCreight "Organization and maintenance of large ordered indices"
2. WAGNER, R. "Indexing design considerations,"IBM Syst. J. 4, (1973)
3. "The Ubiquitous B-Tree "DOUGLAS COMER Computer Science Department, Purdue University,
4. Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. Алгоритмы: построение и анализ. 3-е изд.
5. DOUGLAS COMER: "The Ubiquitous B-Tree Computer Science Department, Purdue University
6. Дональд Кнут. Генерация всех деревьев. История комбинаторной генерации
7. Зубов В. С., Шевченко И. В. "Структуры и методы обработки данных: Практикум в среде Delphi"Глава 6. Поиск в не двоичных деревьях - В-деревьях
8. Freitas R, Wilcke W. Storage-class memory: The next storage system technology. IBM Journal of Research and Development, 2008
9. <https://ru.wikipedia.org/wiki/B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE>
10. [http://algorist.manual.ru/ds/s\\_btr.php](http://algorist.manual.ru/ds/s_btr.php)
11. <https://www.bluerwhite.org/btree/>
12. <https://neerc.ifmo.ru/wiki/index.php?title=B%2B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE>
13. KerttuPollari-Malmi "B+-trees"
14. <https://www.youtube.com/watch?v=DqcZLulVJOM>
15. <https://www.youtube.com/watch?v=pG0deCpuwpI>
16. <https://www.youtube.com/playlist?list=PLsEFMZUL5Ks0qKHxquVleVkM9LFLFS00>
17. [https://cs.hse.ru/data/2019/04/08/1176295299/%D0%A0%D0%B8%D0%B3%D0%B8%D0%BD\\_CoCoS\\_2019.pdf](https://cs.hse.ru/data/2019/04/08/1176295299/%D0%A0%D0%B8%D0%B3%D0%B8%D0%BD_CoCoS_2019.pdf)
18. <https://www.scaler.com/topics/data-structures/b-plus-trees/>
19. <https://stackoverflow.com/questions/870218/what-are-the-differences-between-b-trees-and>
20. [http://algorist.manual.ru/ds/s\\_btr.php](http://algorist.manual.ru/ds/s_btr.php)

21. [https://thodrek.github.io/cs564-fall17/lectures/lecture-14/Lecture\\_14\\_Hash.pdf](https://thodrek.github.io/cs564-fall17/lectures/lecture-14/Lecture_14_Hash.pdf)
22. <https://www.usenix.org/publications/loginonline/revisit-b-tree-vs-lsm-tree-upon-arrival>
23. <https://www.w3schools.blog/b-plus-tree-dbms>
24. <https://www.youtube.com/watch?v=TdtulzNC9iE>
25. <https://www.youtube.com/watch?v=KFcpDTpoixo&t=1s>
26. <https://www.programiz.com/dsa/insertion-on-a-b-plus-tree>
27. <https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html>