B-деревья

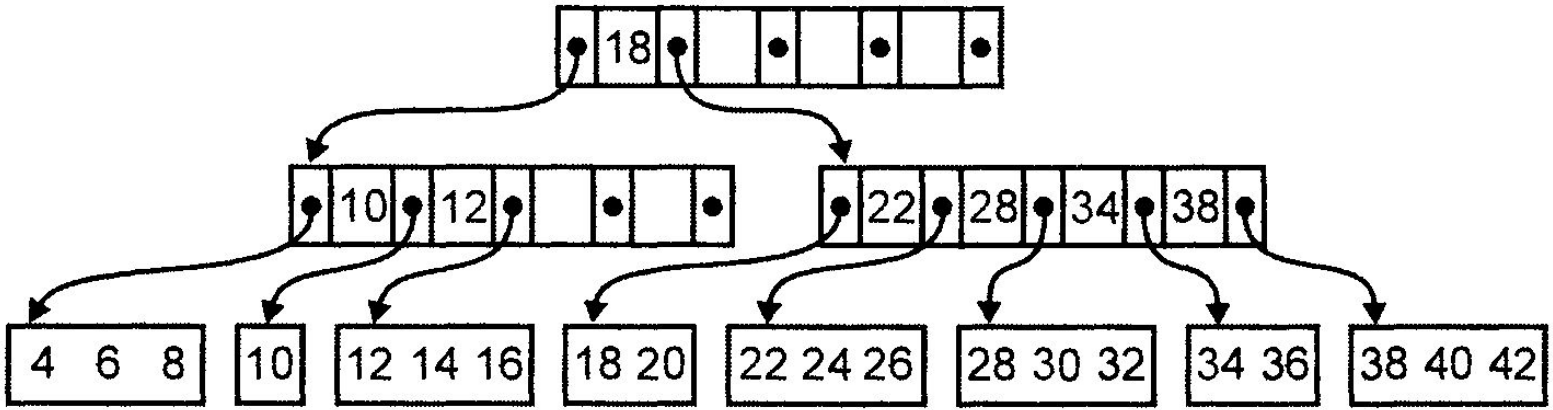
Java Developer Level 2

Содержание

- B деревья
- B* деревья
- B+ деревья

B-дерево

- Сбалансированное, сильно ветвистое дерево
- Придумано Бэйером и МакКрейтом в 1970 г.
- Используется в СУБД и в файловых системах



B-дерево

- Если данные хранятся на медленном устройстве HDD, …
- То выгоднее читать блоками (страницами)

Размер страницы

- Особенности физического хранения на HDD
- Настройки ОС
- Настойки СУБД

Размер страницы

- Oracle - DB BLOCK SIZE

  “ Размер блока базы данных Oracle всегда должен быть равен значению размера блока операционной системы”

N-арность дерева


```
CREATE TABLE Person (
   PersonID int,
   LastName char(80),
   FirstName char(80),
   Sex bit,
   BirthDate date,
   Address char(255),
);
```

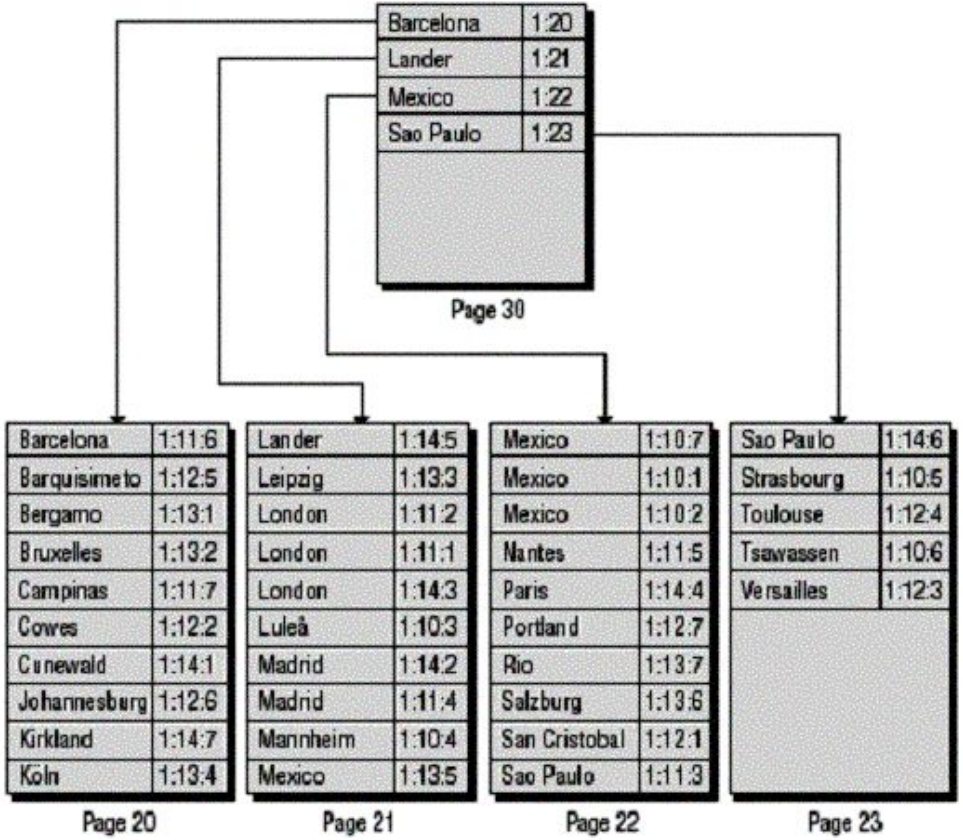
N-арность дерева

```
CREATE INDEX index1 ON Person (LastName);
```

B-дерево

- N-арность дерева = (page_size - header_size) / (sizeof(key) + sizeof(value)+sizeof(**struct**))

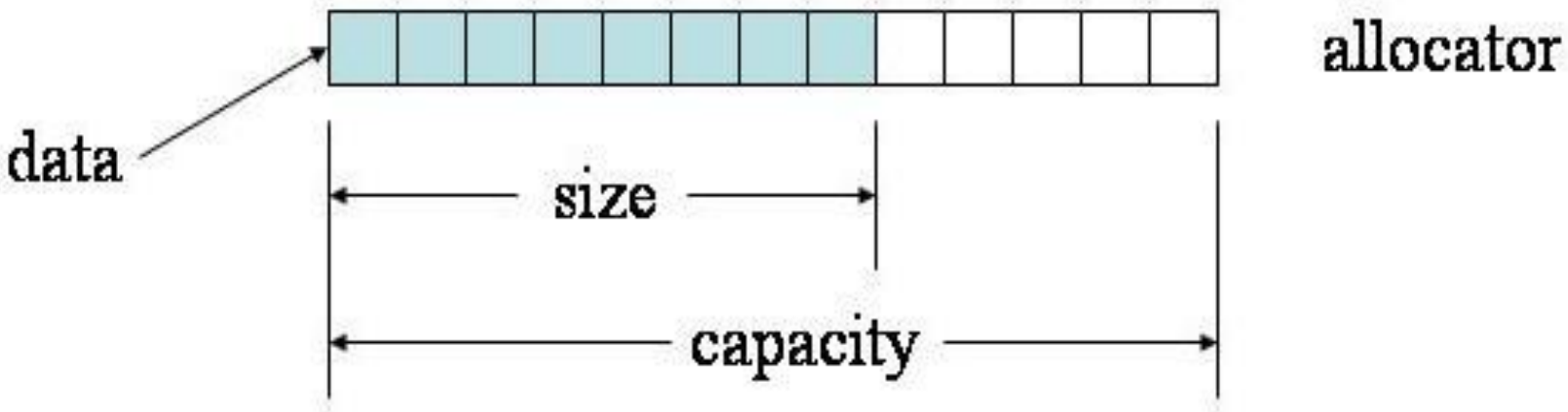
B-дерево


Barcelona 1:20
Lander 1:21
Mexico 1:22
Sao Paulo 1:23
Page 30
Barcelona 1:11:6
Barquisimeto 1:12:5
Bergamo 1:13:1
Bruxelles 1:13:2
Campinas 1:11:7
Cowes 1:12:2
Cunewald 1:14:1
Johannesburg 1:12:6
Kirkland 1:14:7
Köln 1:13:4
Page 20
Lander 1:14:5
Leipzig 1:13:3
London 1:11:2
London 1:11:1
London 1:14:3
Luleå 1:10:3
Madrid 1:14:2
Madrid 1:11:4
Mannheim 1:10:4
Mexico 1:13:5
Page 21
Mexico 1:10:7
Mexico 1:10:1
Mexico 1:10:2
Nantes 1:11:5
Paris 1:14:4
Portland 1:12:7
Rio 1:13:7
Salzburg 1:13:6
San Cristobal 1:12:1
Sao Paulo 1:11:3
Page 22
Sao Paulo 1:14:6
Strasbourg 1:10:5
Toulouse 1:12:4
Tsawassen 1:10:6
Versailles 1:12:3
Page 23

B-дерево

- Количество ключей в страницах файловых систем 50..2000
- При 2000 высота дерева на 1 миллиард записей = 2

B-дерево



B-дерево

- maxSize - максимально допустимое количество потомков
- keys - отсортированы
- Количество ключей на 1 меньше, чем size дерева

Структура данных

```
class Page<K extends Comparable<K>, V> {
  int size;
  V[] values;
  K[] keys;
  Page children[];
  Page parent;
}
```

Алгоритм поиска, B-дерево

```
Найти(ключ):
    индекс = НайтиКлюч(Ключ);
    если индекс < ключи.размер и
          ключи[индекс].ключ == ключ то
        вернуть значение
    если потомки[индекс] пусто то
        вернуть пусто
    иначе
        вернуть потомки[индекс].Найти(ключ)
```

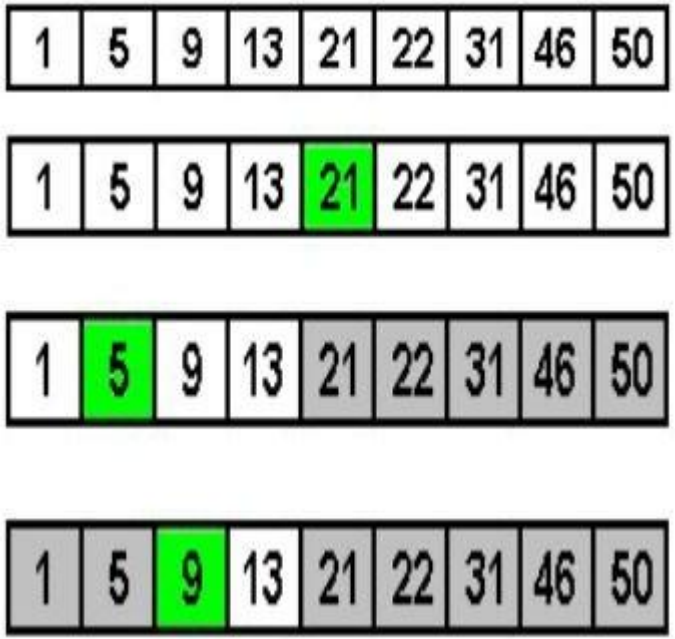
Дихотомия

```
НайтиКлюч(ключ):
    лев = 0
    прав = ключи.размер-1
    индекс  = (левый + правый) / 2;
    пока правый - левый > 1
       если ключи[индекс] <= ключ то
           левый = индекс;
       иначе
          правый = индекс;
       индекс  = (левый + правый) / 2;

  вернуть ключи[индекс] <= ключ ? индекс : индекс+1
```

Дихотомия

- По по степени роста дихотомия эквивалентна поиску в сбалансированном двоичном дереве O(log n)



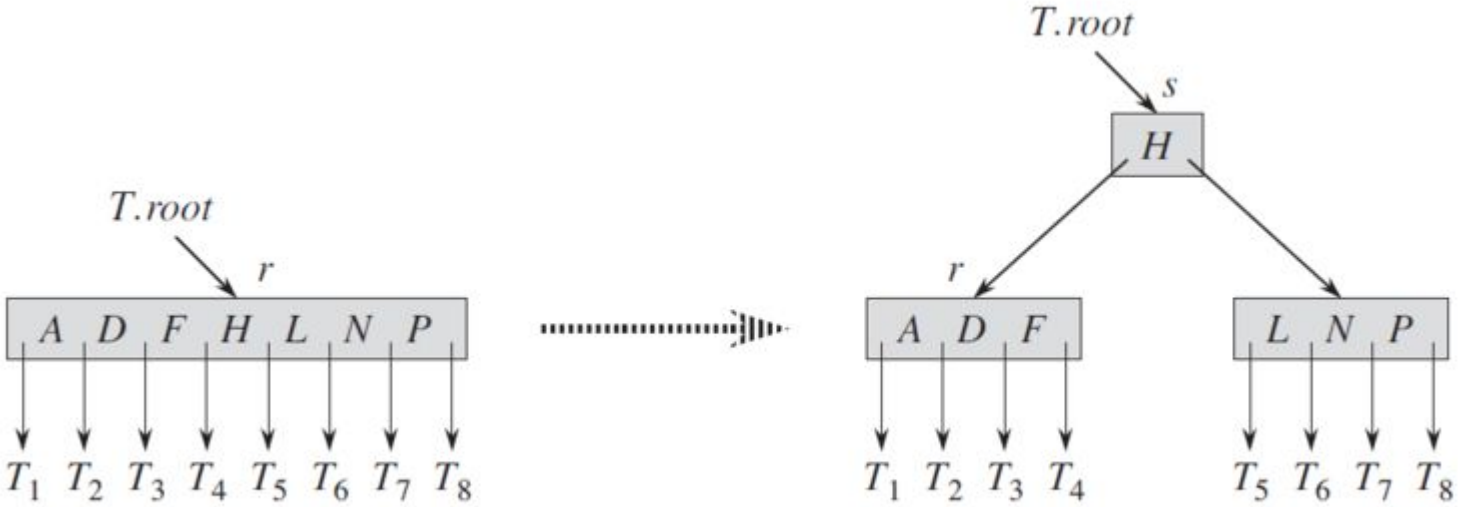
Вставка

- t-фактор - фактор заполненности страницы
- корень содержит 1..2t -1 ключей
- лист содержит t-1..2t -1 ключей
- tfactor = (maxSize+1)/2
- maxSize = 2*tfactor-1

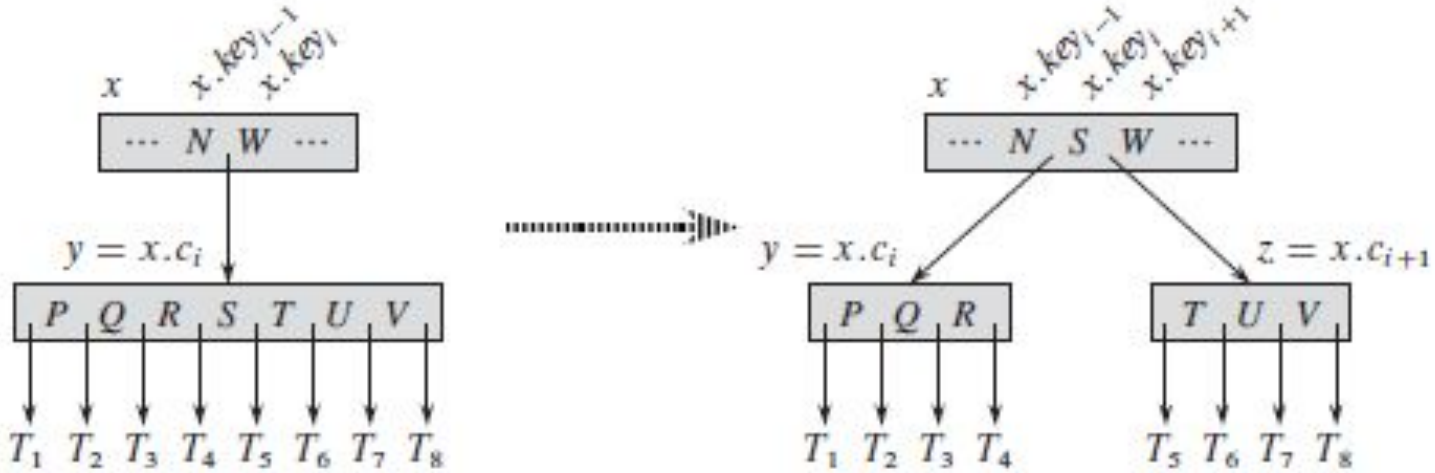
Вставка

- Вставка осуществляется всегда в листовой узел
- Если узел заполнен - он разбивается на 2, ссылка на него вставляется в родителя
- Чтобы это работало при поиске перед вставкой идет превентивное разбиение заполненных страниц

Разбиение корневой страницы



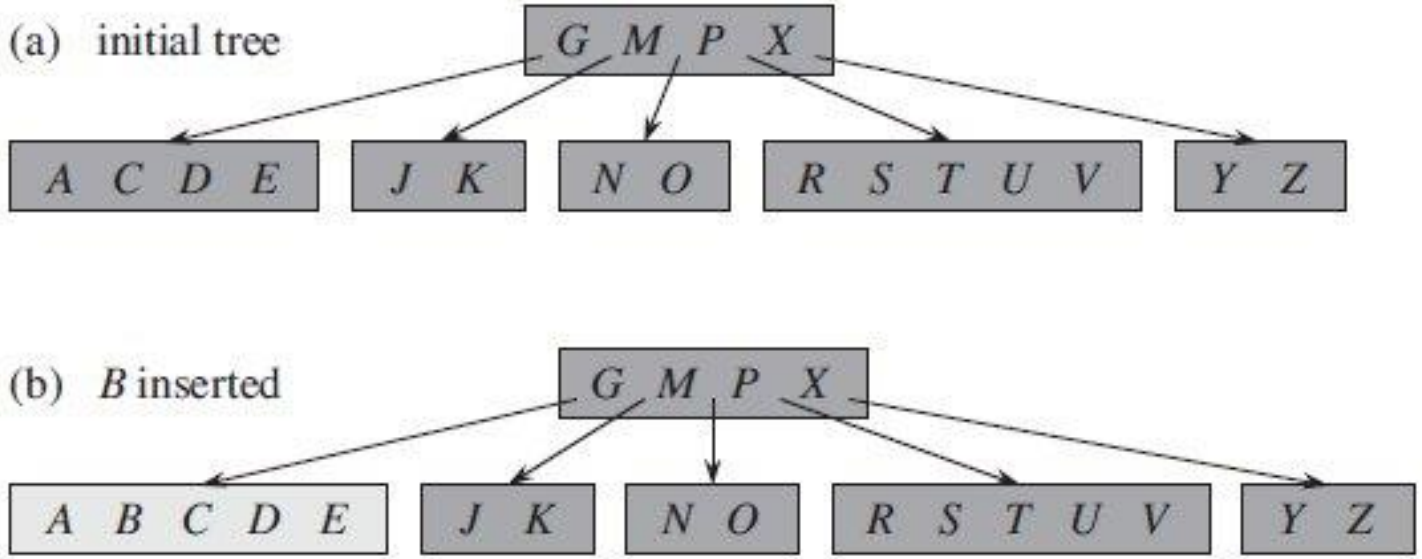
Разбиение листовой страницы



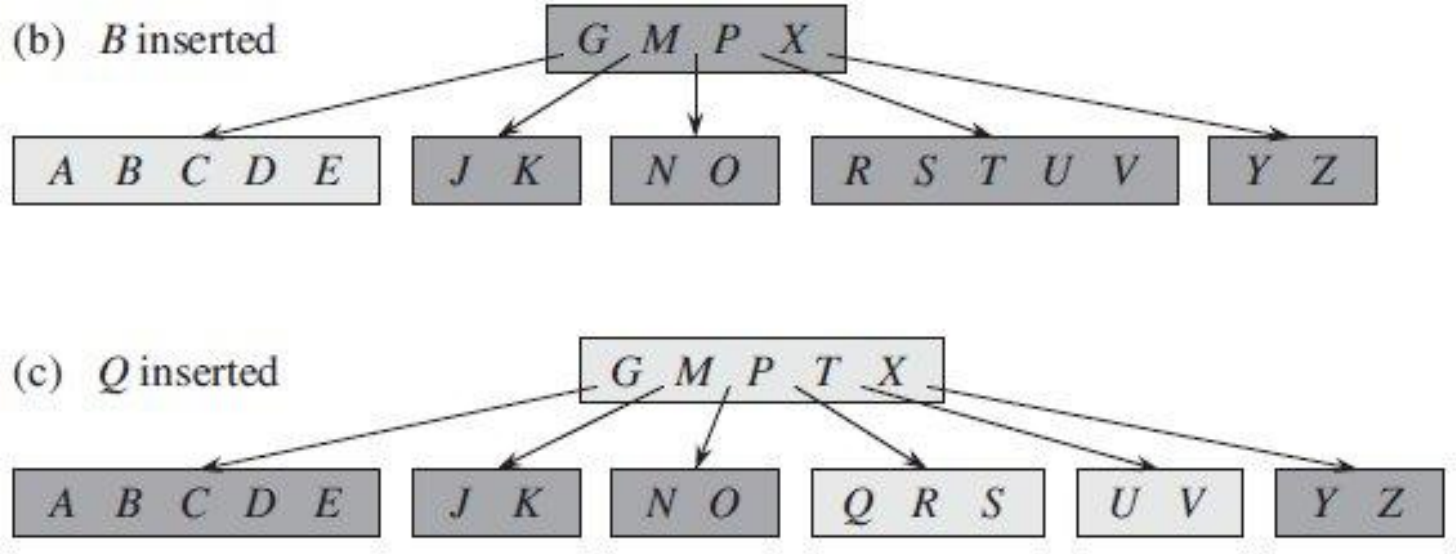
Разбить страницу

```
РазбитьСтраницу():
    середина = размер/2+1
    новая = новая Страница()
    новая.Перенести(текущая, середина+1, размер-1)
    ВставкаСтраницыВРодителя(
          ключи[середина], значения[середина], новая)
    ключи.удалить(середина)
    значения.удалить(середина)
```

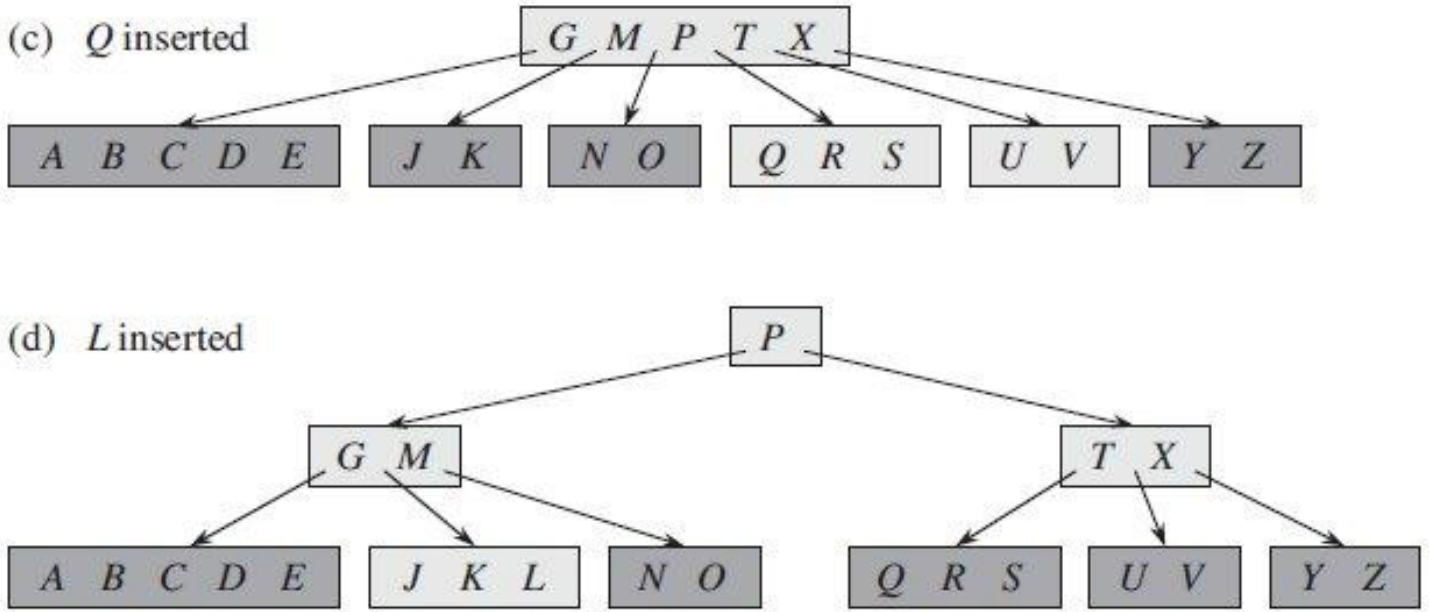
Вставка


(a) initial tree
G M P X
A C D E
J K
N O
R S T U V
Y Z
(b) B inserted
G M P X
A B C D E
J K
N O
R S T U V
Y Z

Вставка


(b) B inserted
G M P X
A B C D E
J K
N O
R S T U V
Y Z
(c) Q inserted
G M P T X
A B C D E
J K
N O
Q R S
U V
Y Z

Вставка


(c) Q inserted
G M P T X
A B C D E
J K
N O
Q R S
U V
Y Z
(d) L inserted
P
G M
T X
A B C D E
J K L
N O
Q R S
U V
Y Z

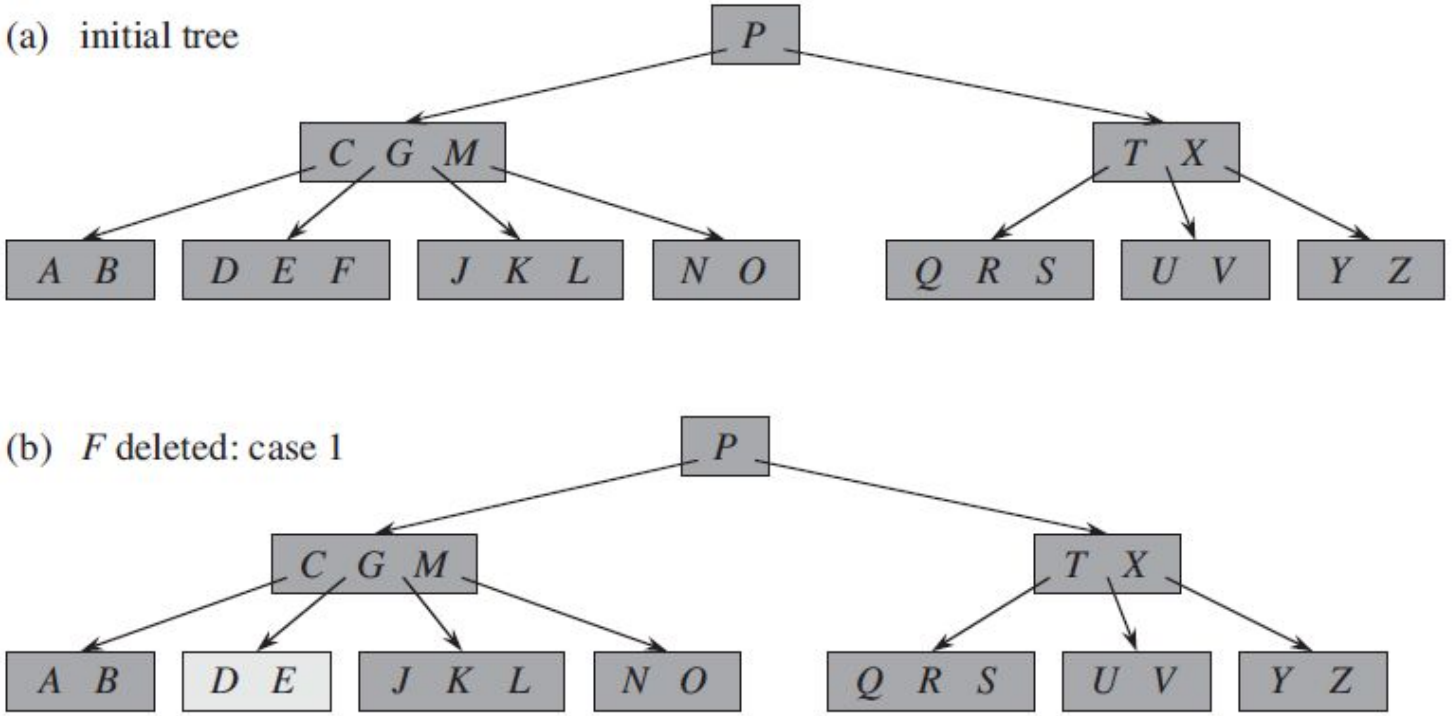
Удаление

- case 1

  если
  - узел - лист
  - количество ключей > t-1

  просто удаляем у себя

Удаление - case 1



Удаление

- case 2

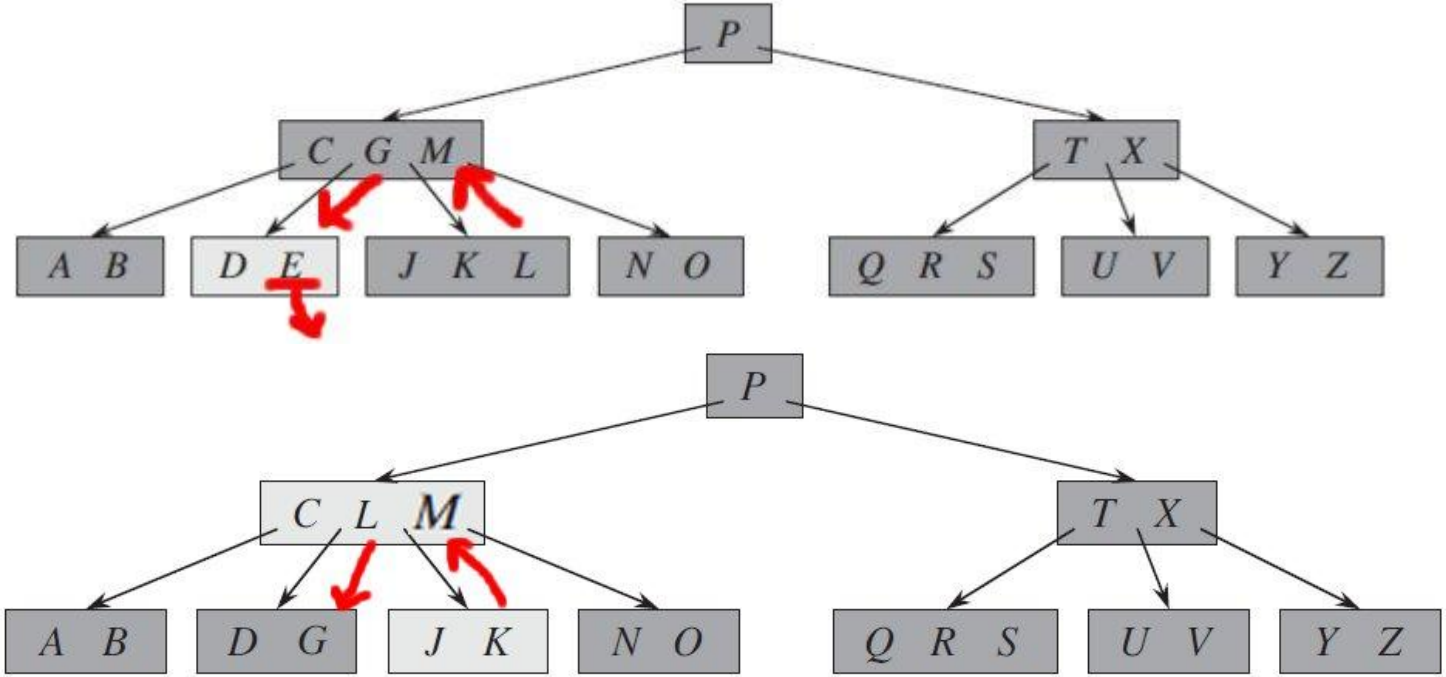
  если
  - узел - лист
  - количество ключей <= t-1
  - существует брат с количеством ключей > t-1

  берем разделительный ключ у брата k1

  заменяем ключ у родителя на на k1

  ключ родителя ставим вместо удаляемого

Удаление, case 2



Удаление

- case 3 - else

  если

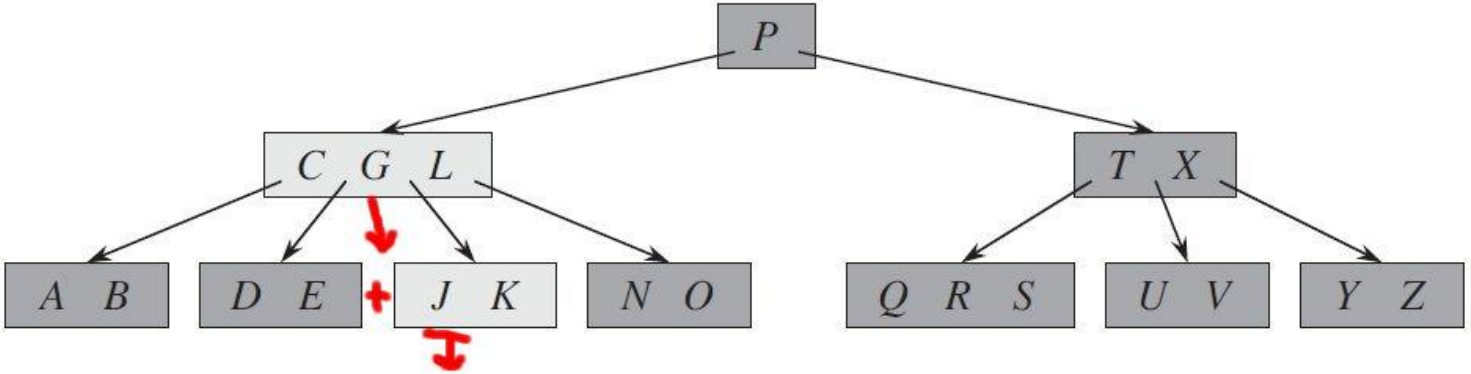
  - узел - лист
  - количество ключей <= t-1
  - не существует брата с количеством ключей > t-1

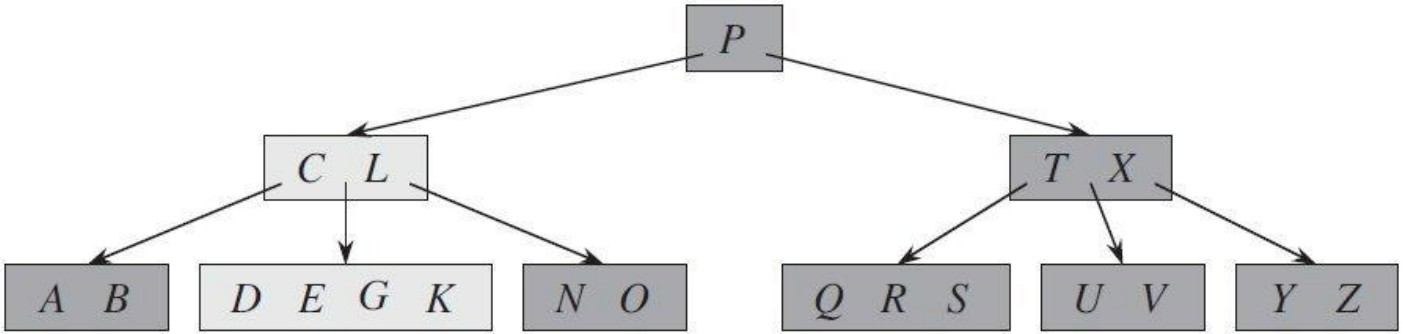
  удаляем нужный ключ

  объединяем 2 страницы потомков

  ключ родителя вставляем себе

Удаление - case 3





Удаление

- case 4

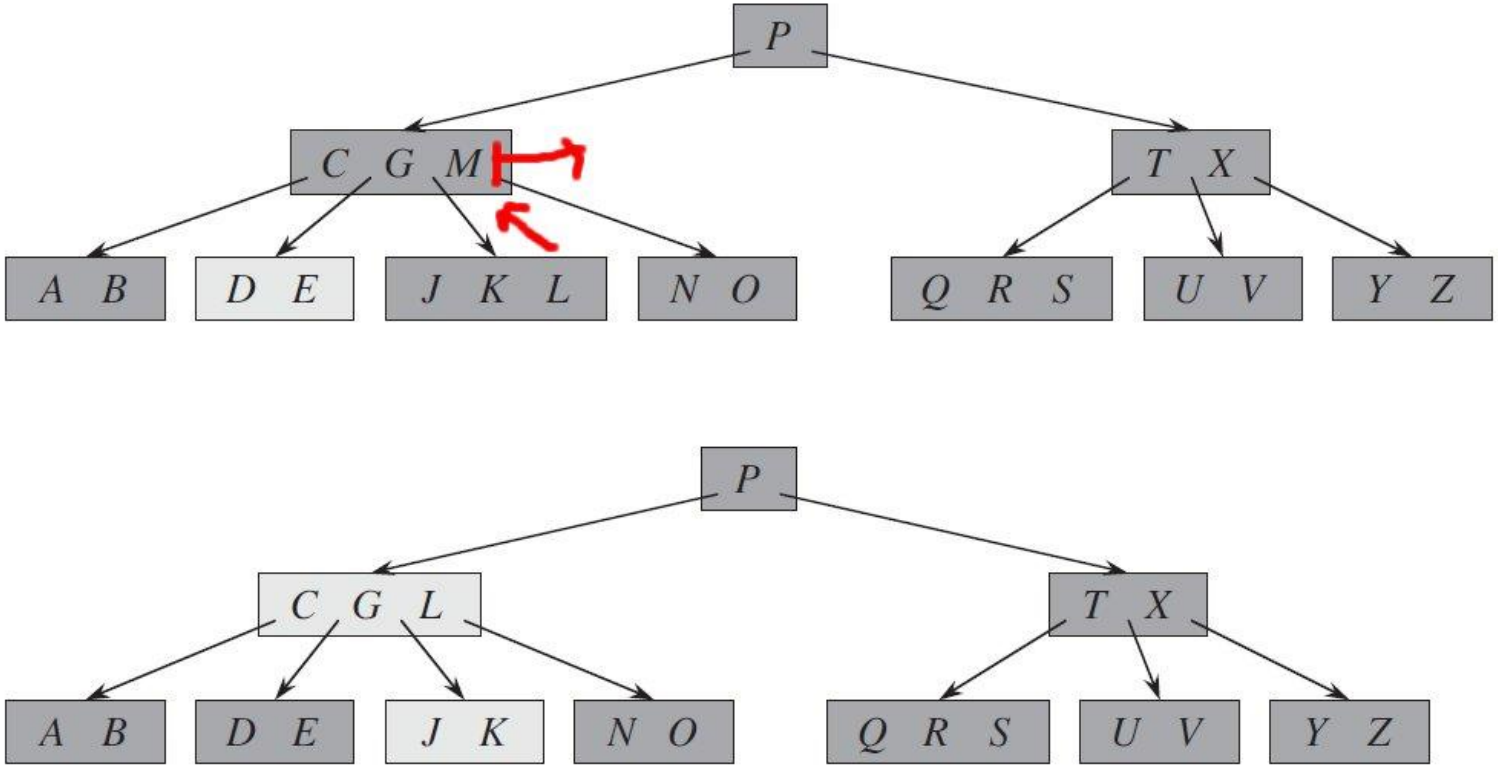
  если

  - узел - промежуточный
  - существует потомок с количеством ключей > t-1

  удаляем нужный ключ

  ключ поднимаем себе вставляем себе

Удаление - case 4



Удаление

- case 5

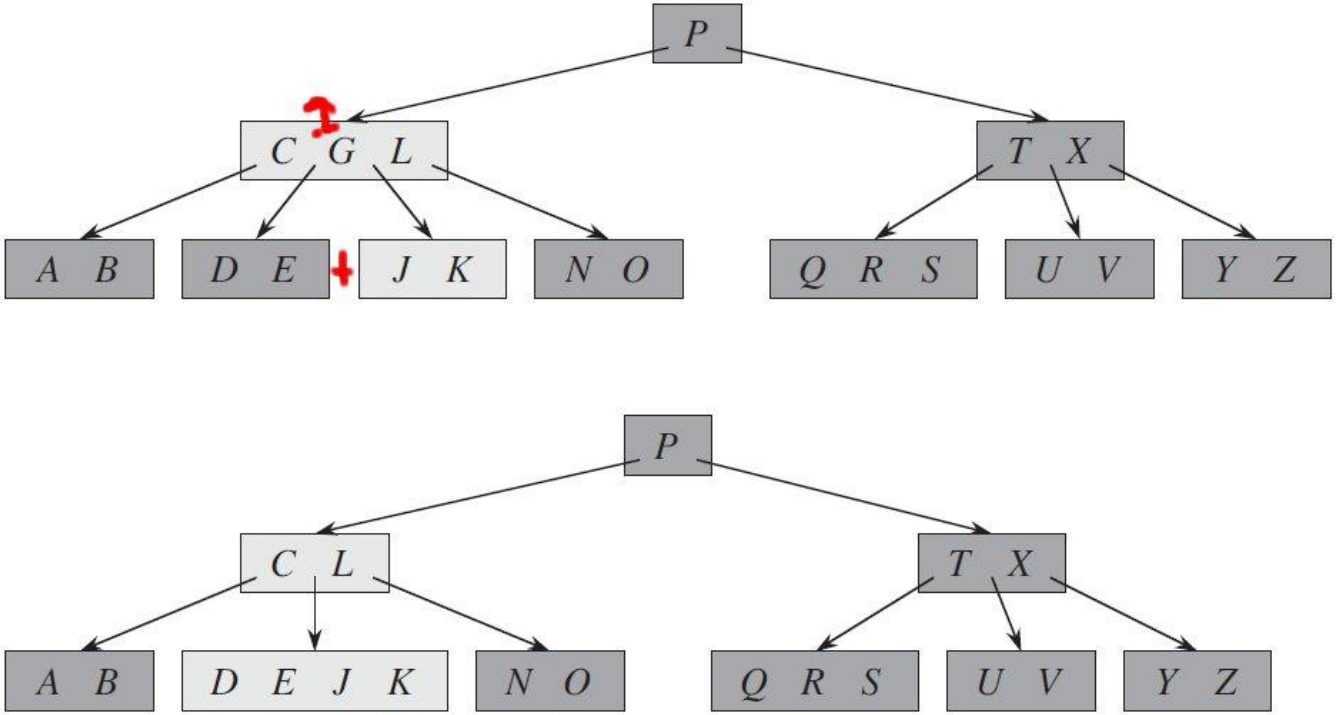
  если

  - узел - промежуточный
  - не существует потомок с количеством ключей > t-1

  удаляем нужный ключ

  объединяем страницы потомков

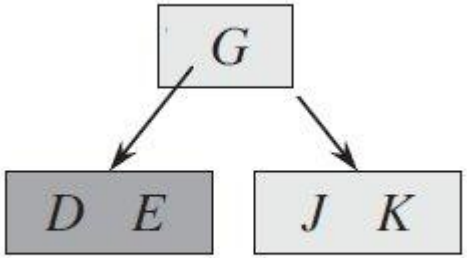
Удаление - case 5



Удаление

- при объединении крайних 2-х страниц, удаляем корень (переприсвоить root)
- при удалении корневой страницы нужно присвоить null в корень (root)

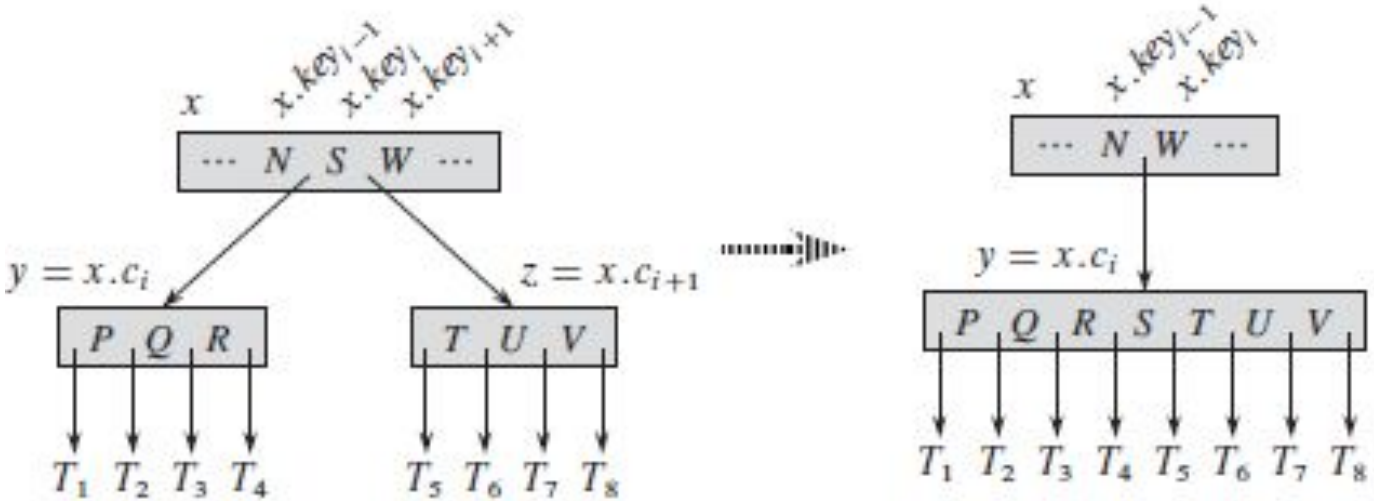
Удаление - финал



Операции

- удалить ключ-значение в странице
- вставить ключ-значение в страницу
- объединить 2 страницы

объединение страниц



Заключение

- B-дерево используется как структура данных при работе с более медленными устройствами
- Представляет из себя сильно ветвистое сбалансированное дерево
- Листовые страницы заполнены от t-1 до 2t-1
- Корень заполнен от 1 до 2t-1

B* дерево

- Вариант B дерева, заполненного на 2/3
- При вставке, если узел полностью заполнен то вначале заполняем брата, а потом делим 2 узла на 3
- Удаление как в B-дереве только держим заполненность 2/3

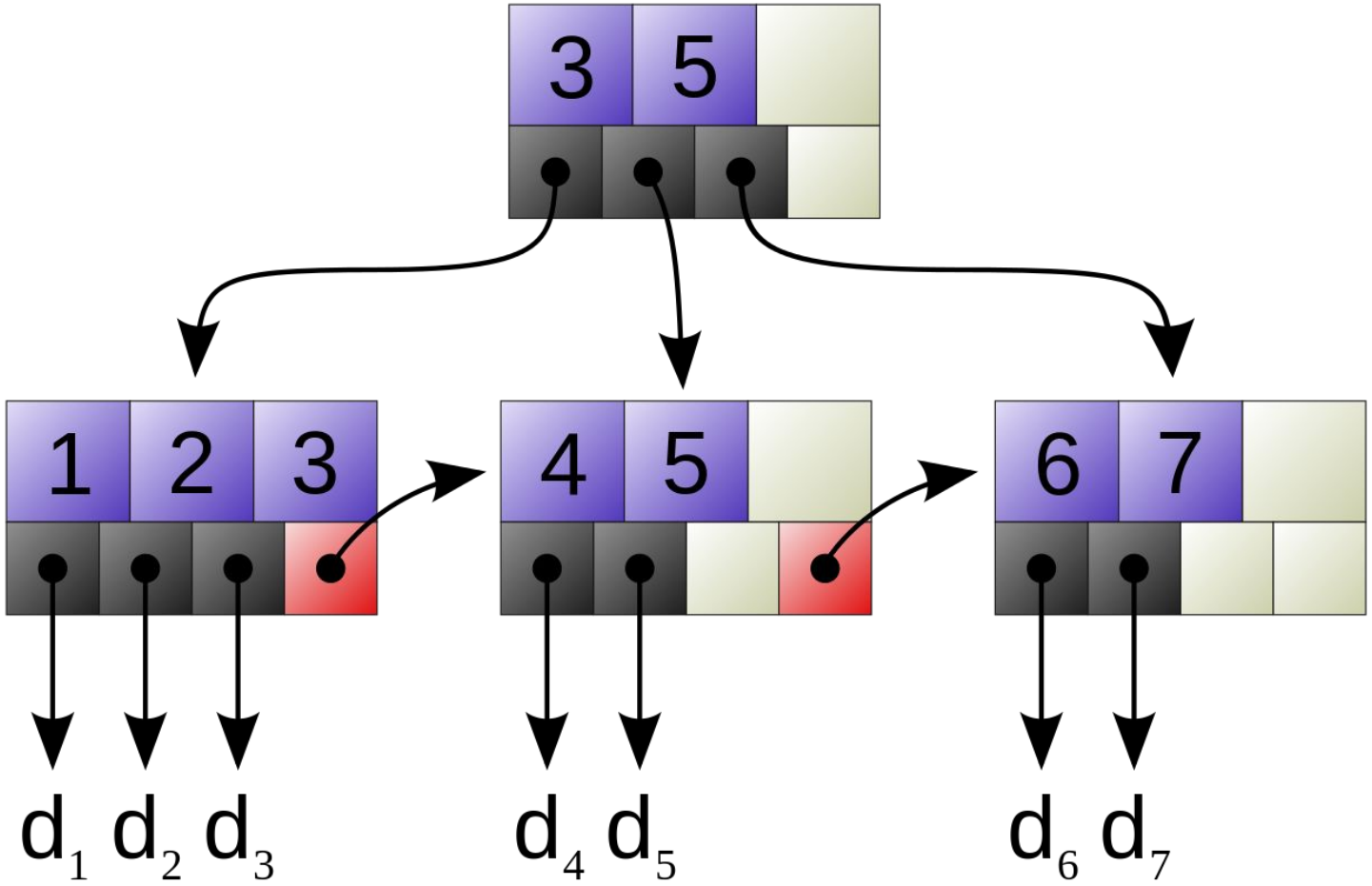
B* дерево

- Таким образом уменьшается % не используемой памяти в среднем с 25% до 16.5%
- Чуть сложнее алгоритмически

B+ дерево

- Вариант B дерева, в котором значения сохраняются только в листовых (терминальных) узлах
- Таким образом ключи дублируются от промежуточного узла вплоть до листового
- Все листовые узлы соеденены в связный список

B+ дерево



B+ дерево

- Требует больше памяти
- Поиск всегда заканчивается в листе
- Удаление тоже всегда происходит из листа
- Имеет возможность последовательного доступа к значениям без обхода дерева
- В остальном это обычное B-дерево

B+ дерево

PROGWARDS
Академия компьютерного мастерства

- Теоретически описано в 1978 г.
- Использовалось IBM в VSAM с 1973 г.
- Используется в файловых системах : NTFS, ReiserFS, NSS, XFS, JFS, ReFS и BFS
- Используется в СУБД: DB2, Informix, Microsoft SQL Server, Oracle Database, Adaptive Server Enterprise и SQLite