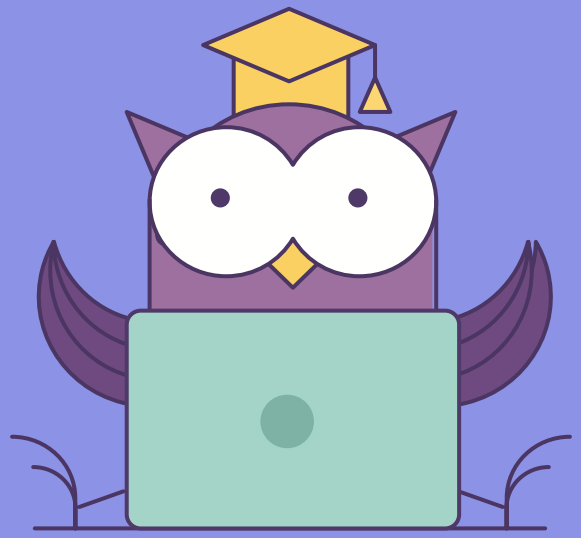
ОНЛАЙН-ОБРАЗОВАНИЕ

Не забыть включить запись!

Меня хорошо слышно && видно?



Напишите в чат, если есть проблемы!

Ставьте + если все хорошо

Поехали!

OTUS

B-деревья, деревья отрезков

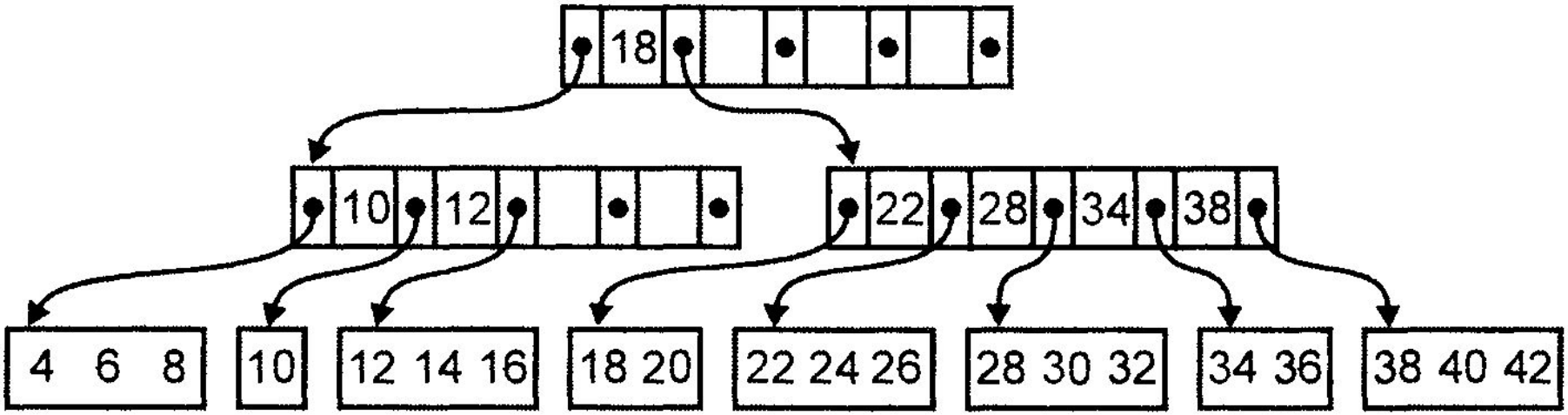
Мазнев Валерий

Содержание

- B деревья
- B+ деревья
- Деревья оптимального поиска
- Матричные деревья
- Деревья отрезков

B-дерево

- Сбалансированное, сильно ветвистое дерево
- Бэйером и МакКрейтом в 1970 г.
- Используется как индексы СУБД



B-дерево

- Данные хранятся на медленном устройстве HDD, SSD, …
- Выгоднее читать блоками (страницами)

B-дерево

- Особенности хранения на HDD
- Настройки ОС
- СУБД

- Oracle - DB BLOCK SIZE

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

B-дерево

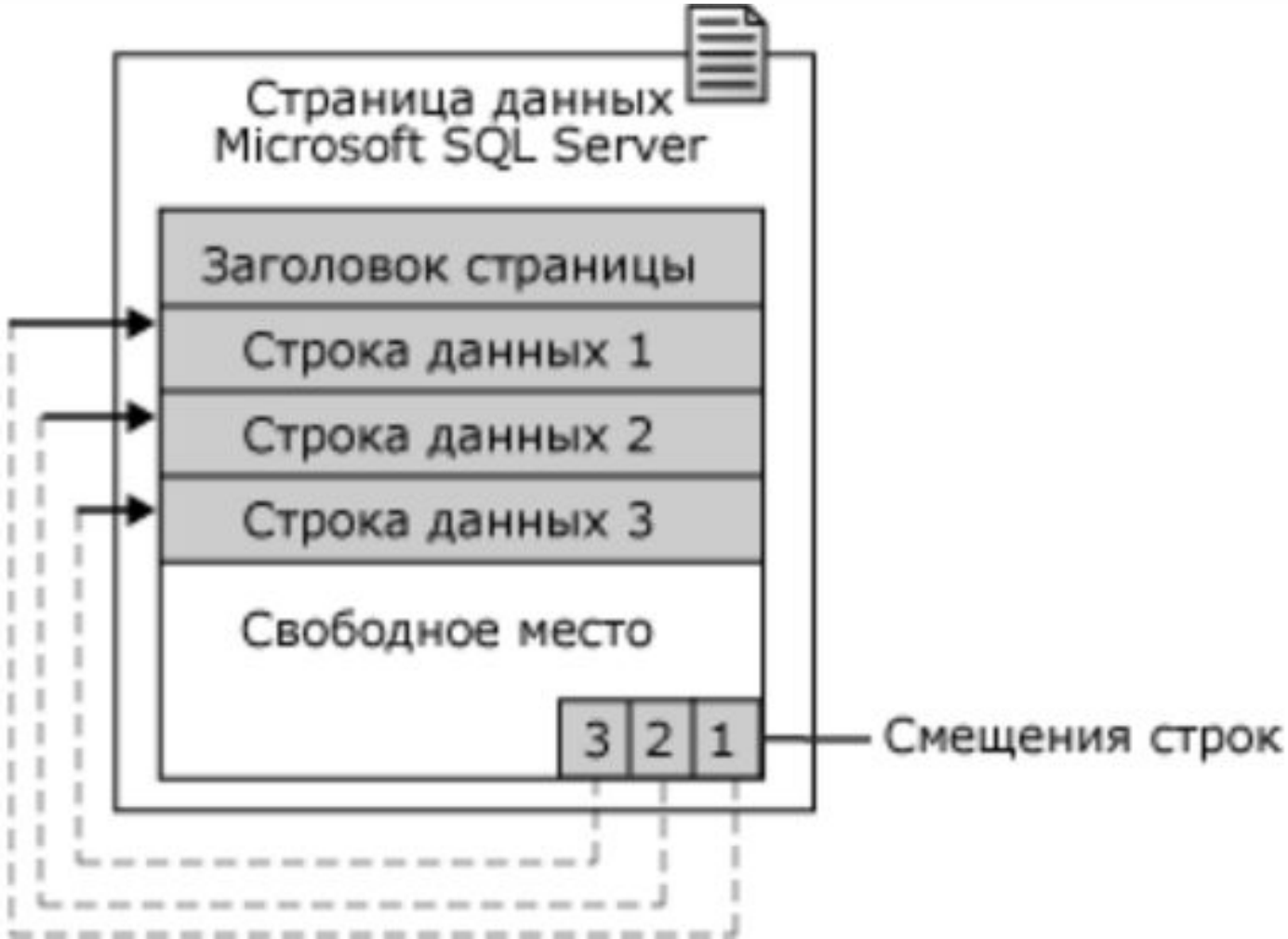
OTUS

```
CREATE TABLE Persons (
    PersonID int,
    LastName char(80),
    FirstName char(80),
    Address char(255),
    City char(80)
);
```

B-дерево

```
Struct Persons {
   int PersonID;
   char[80] LastName;
   char[80] FirstName;
   char[255] Address;
   char[80] City;
};
```

- Массив структур

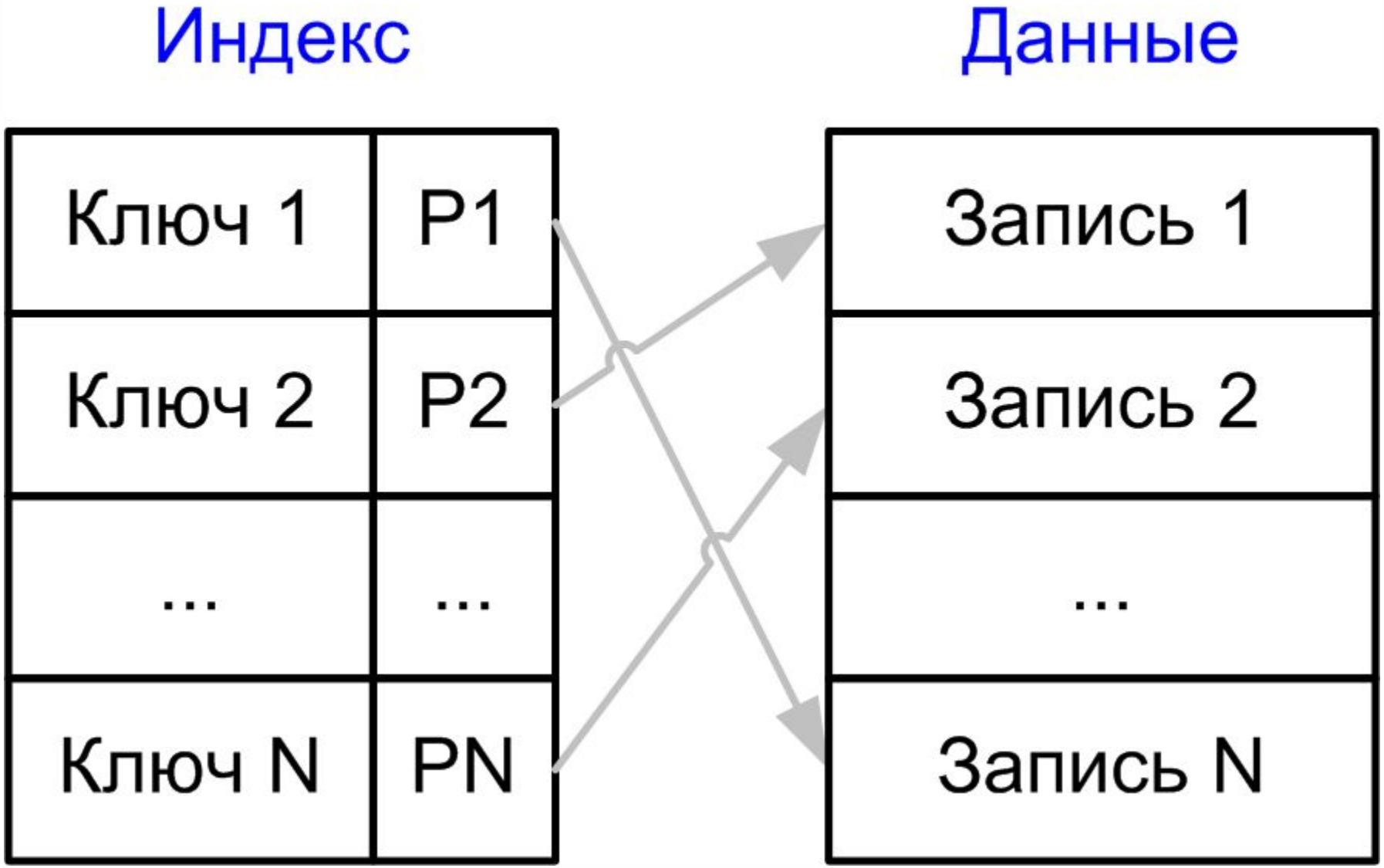


B-дерево

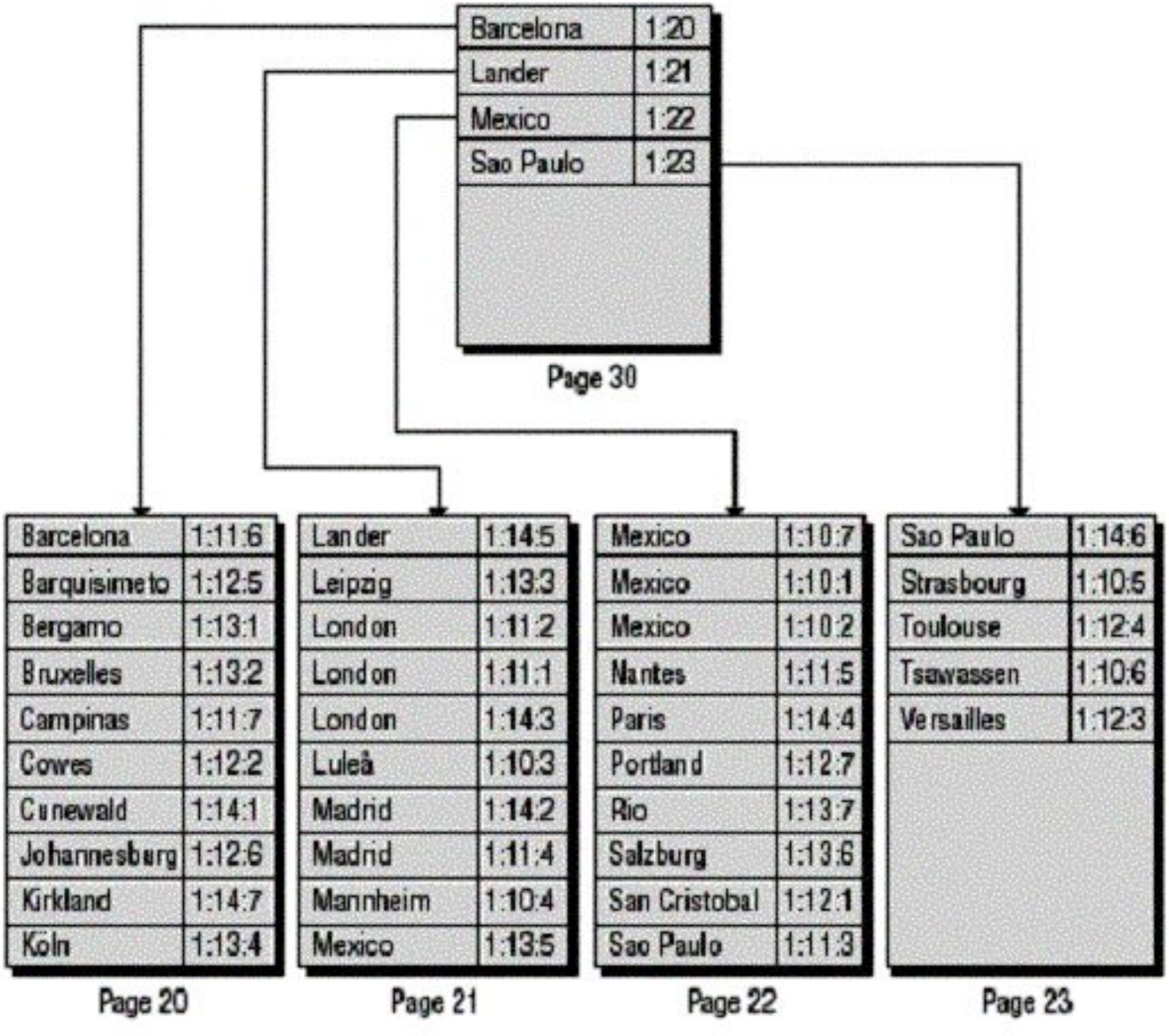
OTUS

```
CREATE INDEX index1 ON Persons (City);
```

B-дерево


Индекс
Данные
Ключ 1
P1
Ключ 2
P2
...
...
Ключ N
PN
Запись 1
Запись 2
...
Запись N

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
Koln 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

Структура данных двоичного дерева

```
class TreeLeaf<Key, Value> {
    Key key;
    Value value;
    TreeLeaf parent;
    TreeLead left;
    TreeLeaf right;
}
```

Структура данных B дерева

```
class BTreeLeaf<Key, Value> {
 int size;
 Value *values;
 Key[] keys;
 BTreeLeaf* Children[];
}
```

- size = (page_size - header_size) / (sizeof(Key) + sizeof(Value*))
- Keys - отсортированы
- Количество ключей на 1 меньше, чем size дерева

Алгоритм поиска, двоичное дерево

```
Найти(ключ):
    если текущий.ключ == ключ то
        вернуть значение
    если текущий.ключ > ключ то
        если левое пусто то
            вернуть пусто
        иначе
            вернуть левое.Найти(ключ)
    иначе
        если правое пусто то
            вернуть пусто
        иначе
            вернуть правое.Найти(ключ)
```

Алгоритм поиска, B-дерево, дихотомия

OTUS

```
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)
```

Алгоритм поиска, B-дерево, перебором ключей

OTUS

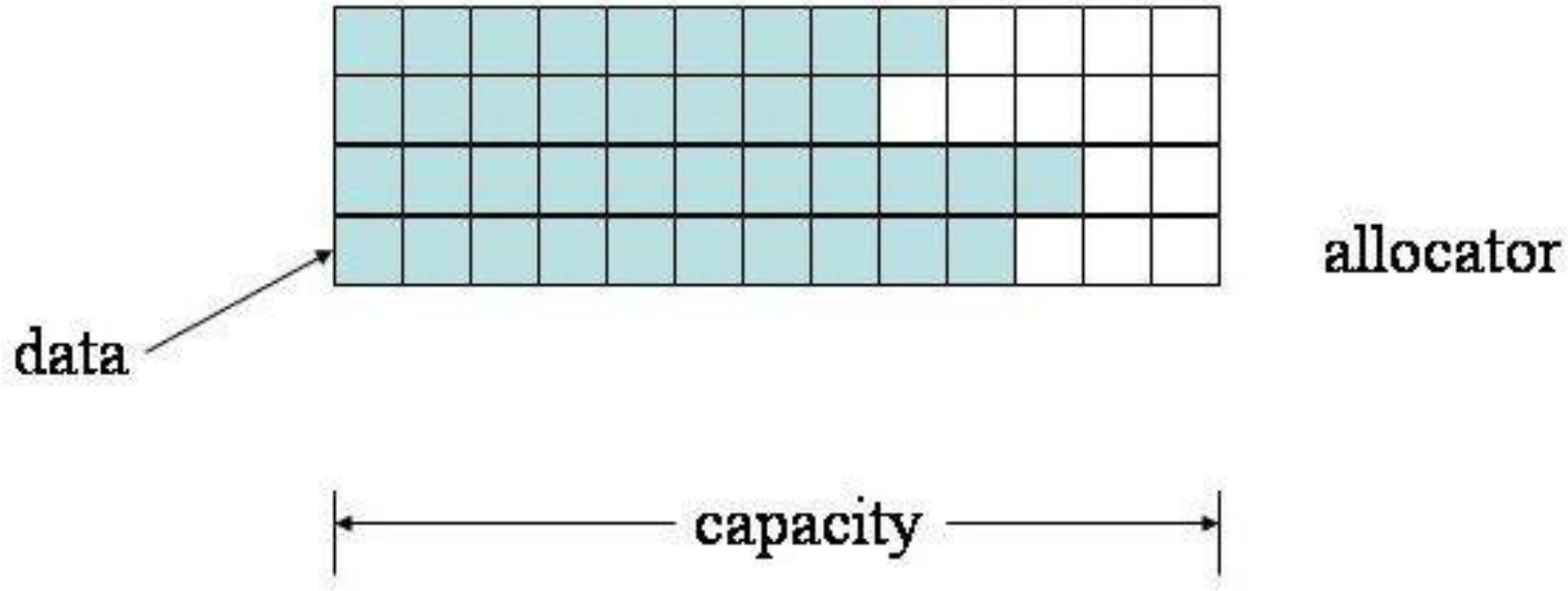
```
НайтиКлюч(ключ):
    для i от 0 до ключи.размер-1
        если ключ <= ключи[i] то
            вернуть i
    вернуть размер+1
```

Дихотомия

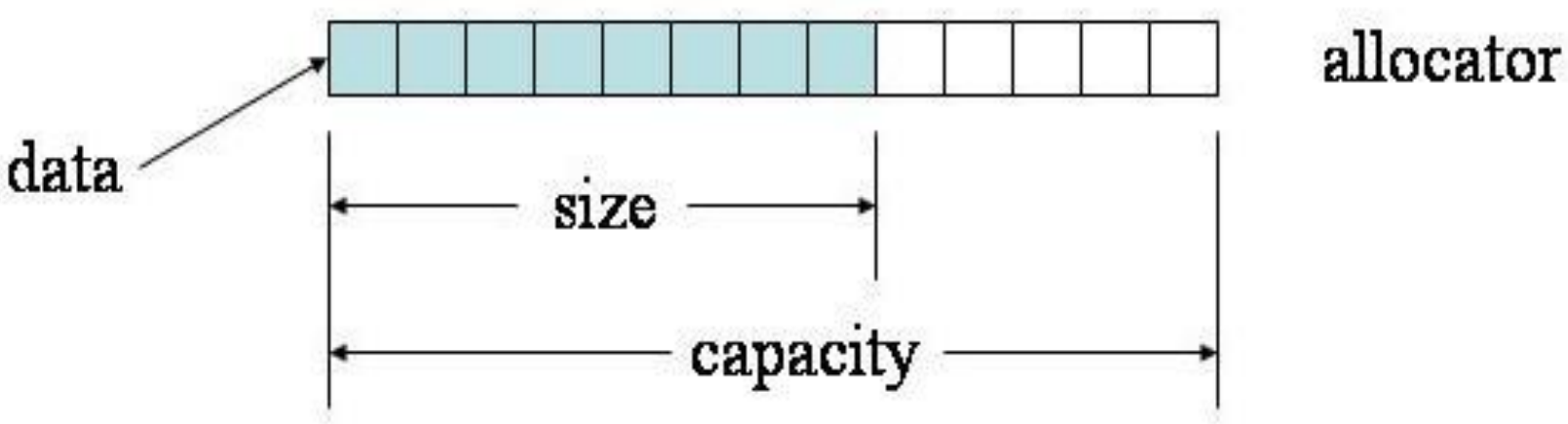
```
НайтиКлюч(ключ):
    лев = 0
    прав = ключи.размер-1
    индекс  = (левый + правый) / 2;
    пока правый - левый > 1
        если ключи[индекс] <= ключ то
            левый = индекс;
        иначе
            правый = индекс;
        индекс  = (левый + правый) / 2;

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

Сложность?

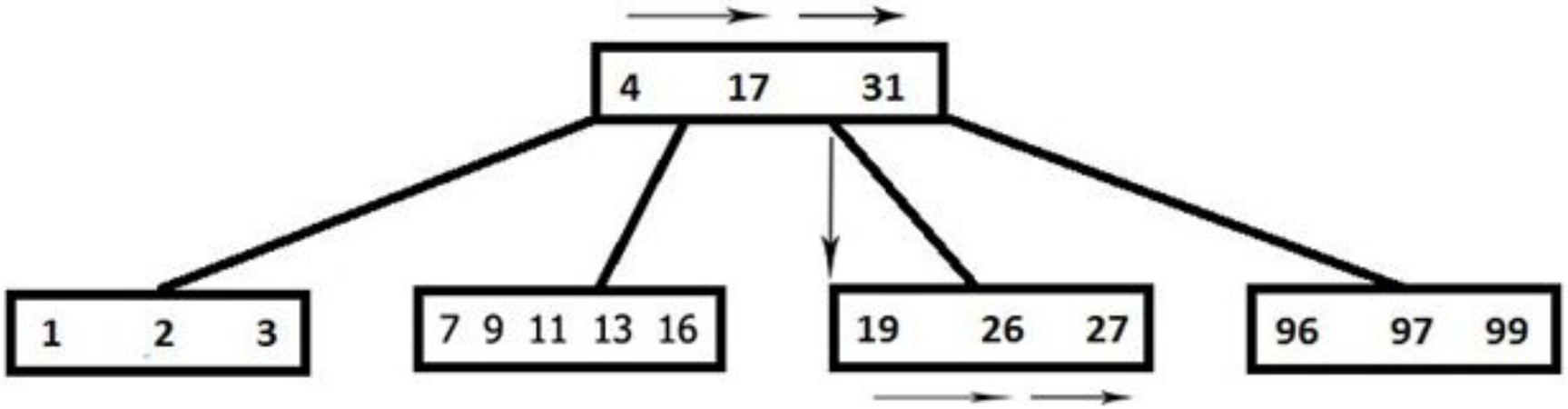
- Вставка в любое место массива одинаково эффективна



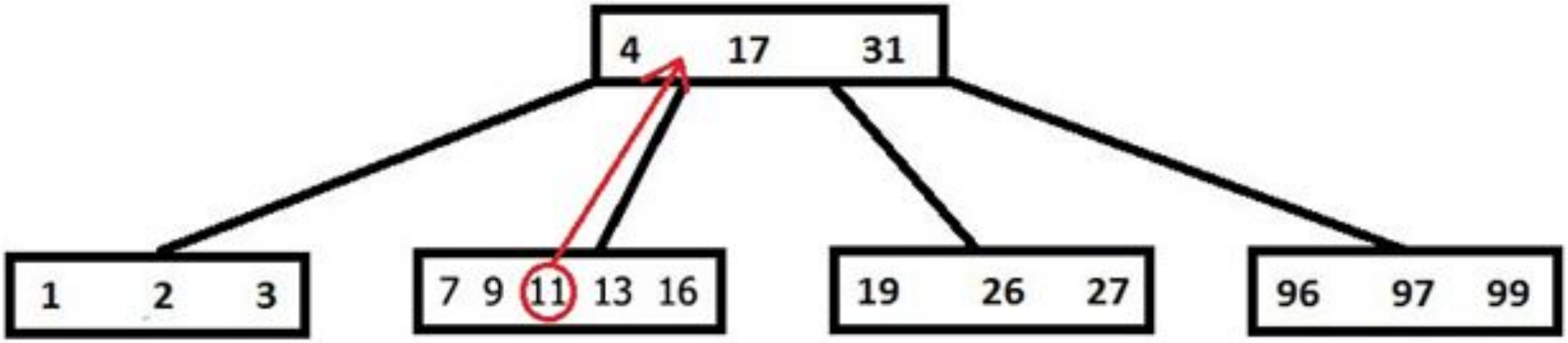
- Фактор t
- N = 2t +1



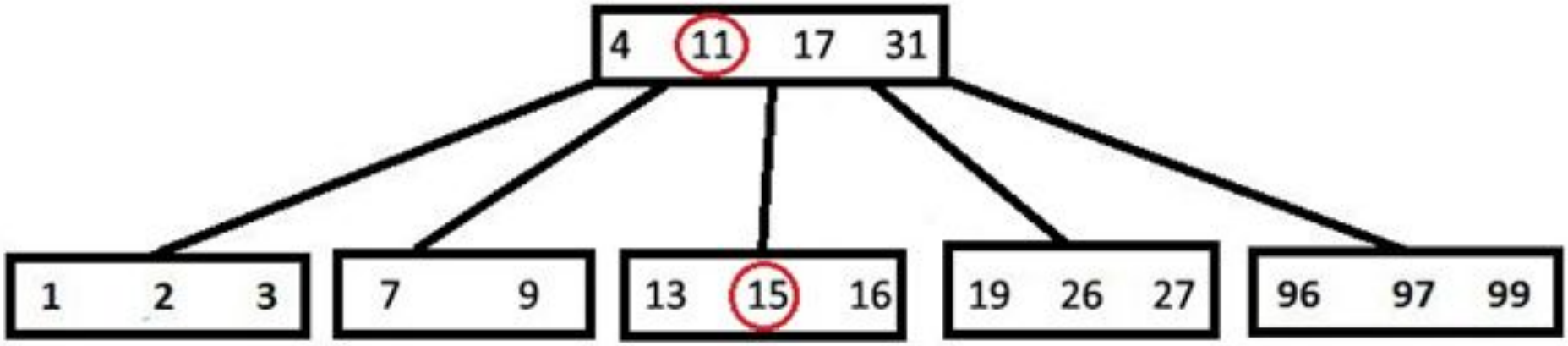
Вставка



Вставка



Вставка



Пример

- N = 10
- t фактор = 5

- если < 5 то
  вставляем в себя
  иначе
  вставляем в потомка

Вставка

```
Вставка(ключ):
    если ЛистовойУзел() то
        ВставкаВСебя()
    иначе
        индекс = НайтиКлюч(Ключ);
        потомки[индекс].Вставка(ключ, значение)
```

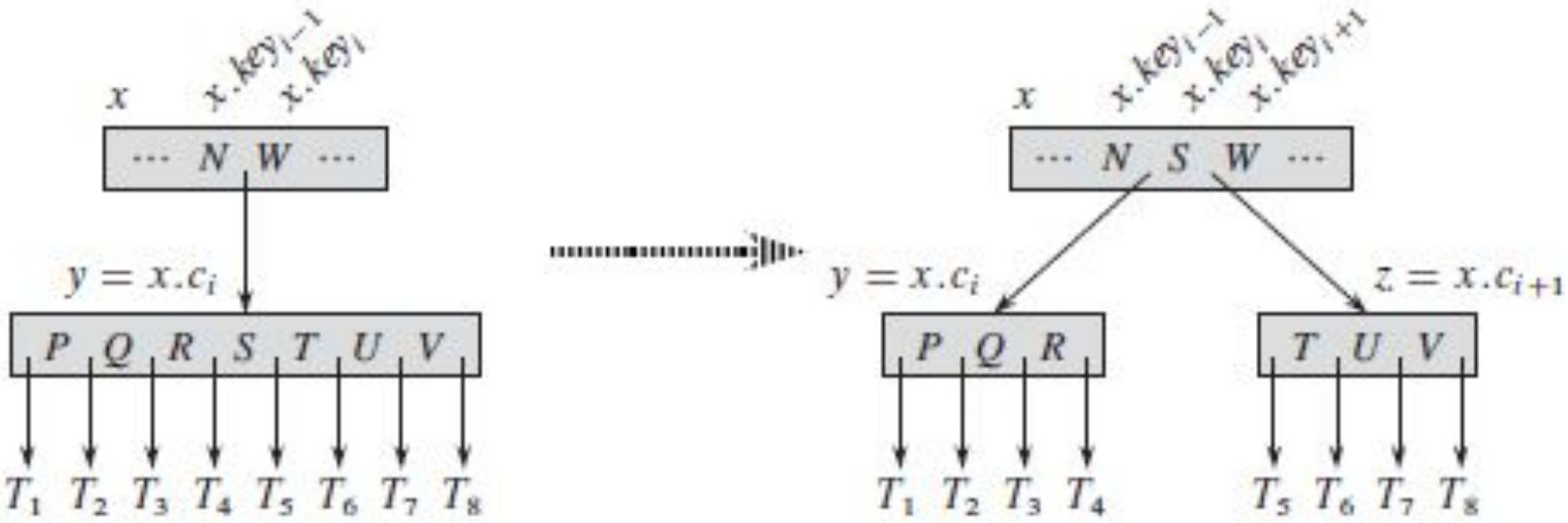
Проверка на лист

```
ЛистовойУзел():
    вернуть потомки.размер == 0
```

Вставка в себя

```
ВставкаВСебя(ключ):
    если ключи.размер == размер-1 то
        РазбитьСтраницу()

    индекс = страница.НайтиКлюч(Ключ);
    родитель.ключи.вставить(индекс, ключ)
    родитель.значения.вставить(индекс, значение)
    родитель.потомки.вставить(индекс, текущая)
```

Разбиение


x
x.key_{i-1}
x.key_i
... N W ...
y = x.c_i
P Q R S T U V
T1 T2 T3 T4 T5 T6 T7 T8
x
x.key_{i-1}
x.key_i
x.key_{i+1}
... N S W ...
y = x.c_i
z = x.c_{i+1}
P Q R
T U V
T1 T2 T3 T4
T5 T6 T7 T8

Разбить страницу

OTUS

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

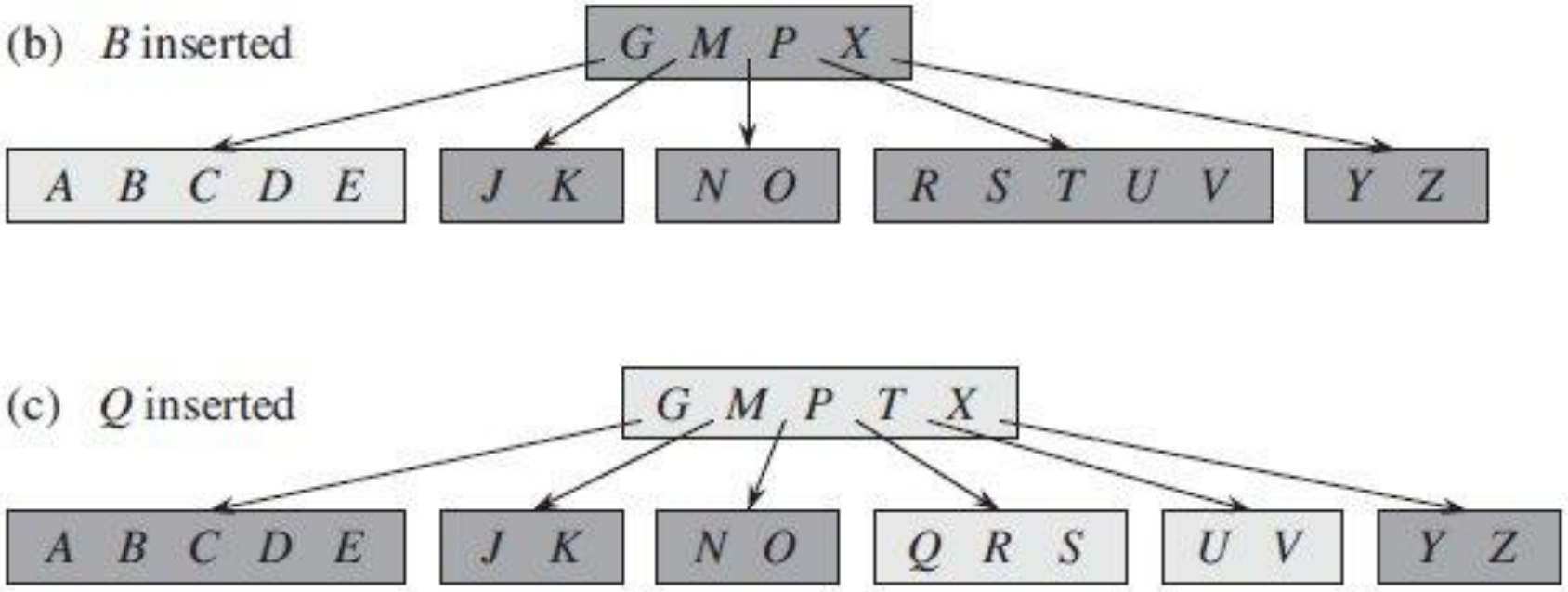
Перенести

```
Перенести(страница, начало, конец):
    для i=начало до конец
        ключи.добавить(страница.ключи[i])
        значения.добавить(страница.значения[i])
        если потомки.размер > 0 то
            потомки.добавить(страница.потомки[i+1])
        страница.ключи.удалить(i)
        страница.значения.удалить(i)
        если потомки.размер > 0 то
            страница.удалить(i+1)
```

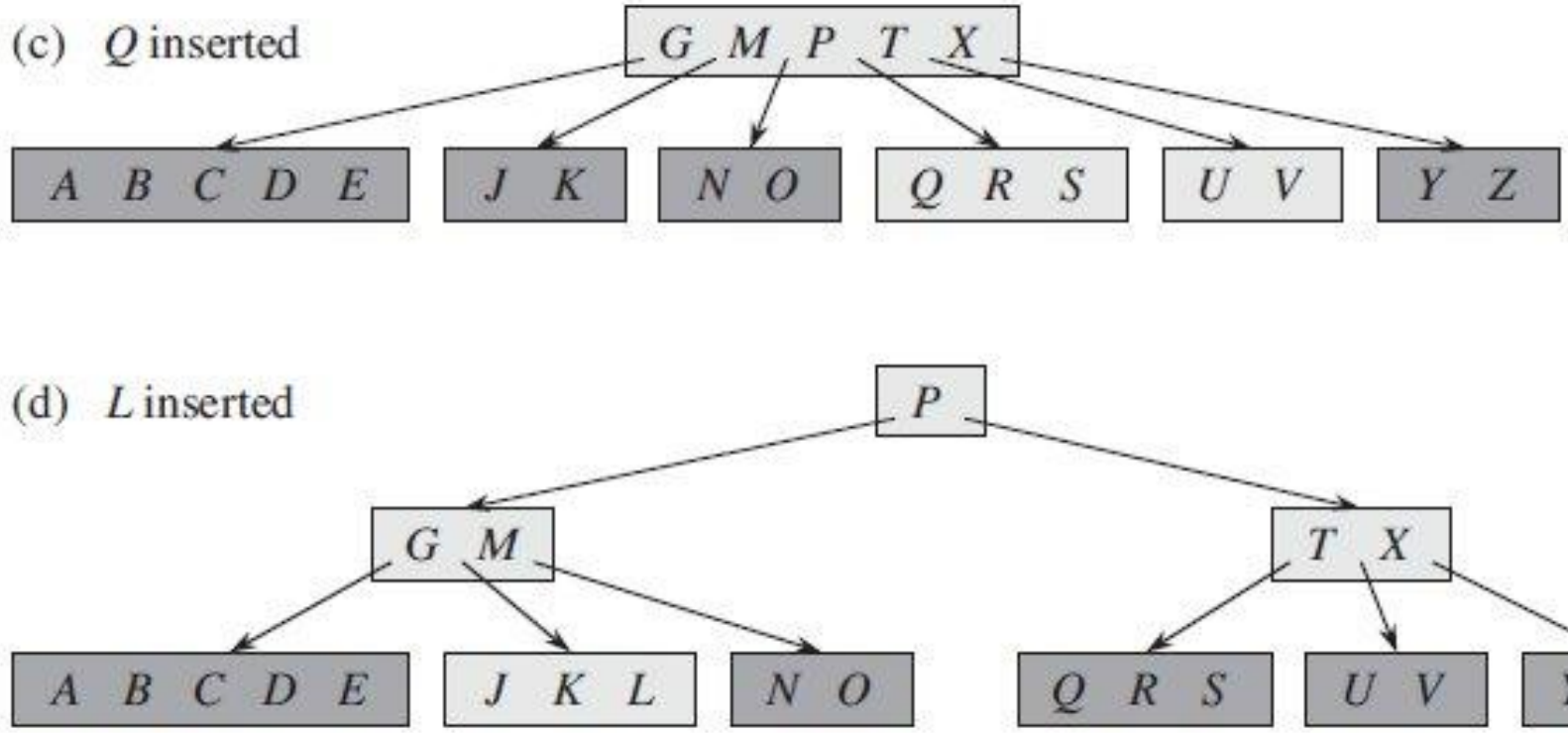
Вставка страницы в родителя

```
ВставкаСтраницыВРодителя(страница, ключ, значение):
    если родитель пусто то
        страница = новая Страница()
        страница.потомки.добавить(0, корень)
        корень = страница
    иначе
        если родитель.ключи.размер == размер-1 то
            родитель.РазбитьСтраницу()
        страница = родитель

    индекс = страница.НайтиКлюч(Ключ);
    страница.ключи.вставить(индекс, ключ)
    страница.значения.вставить(индекс, значение)
    страница.потомки.вставить(индекс, текущая)
```

Вставка

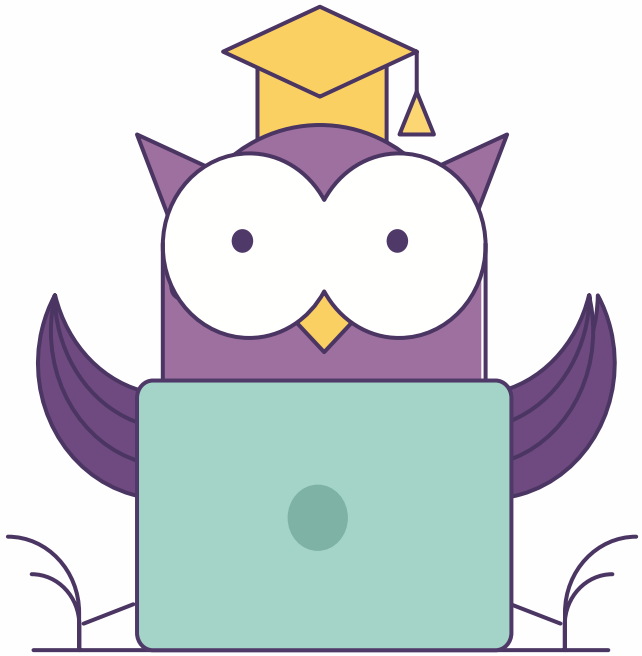

(b) B inserted
G M P X
A B C D E
J K
N O
R S T U V
Y Z
(c) Q inserted
G M P T X
A B C D E
J K
N O
Q R S
U V
Y Z

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

Вопросы?

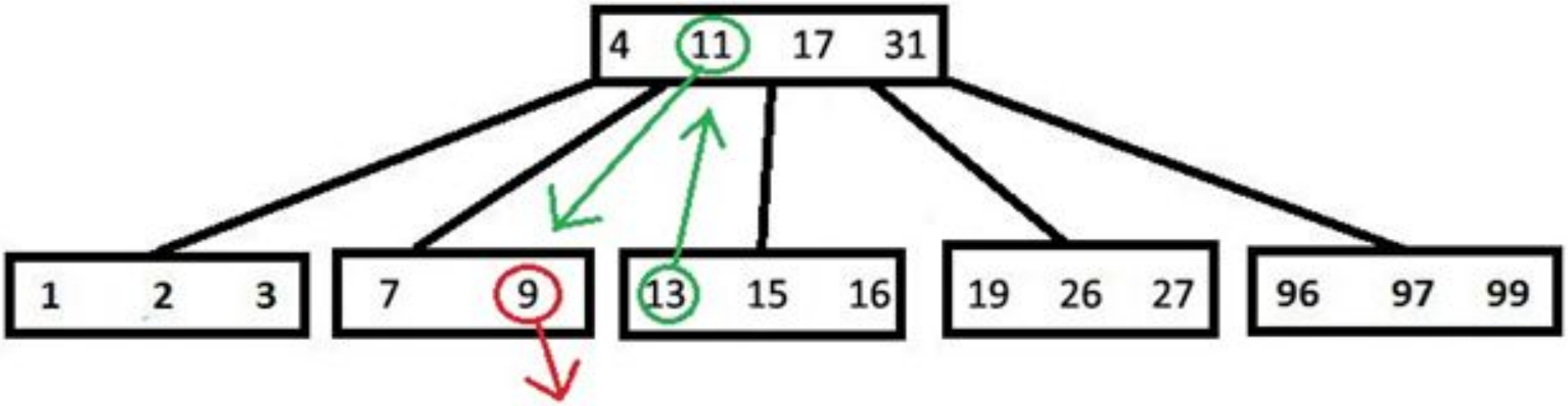
- Вопросы вставке в B дерево?



Удаление

```
узел = Найти(ключ)
если узел пусто то
    выход

если потомки[узел] лист то
    если потомки[узел].ключи.размер > t-1 то
        потомки[узел].УдалисьУСебя(ключ)
    иначе
        ...
```

Удаление



```
...
иначе
        брат1 = узел.НайтиПолногоБрата()
        если брат1 не пусто то
            ключ1 = брат1.ВыбратьКлюч1()
            ключ2 = родитель.ВыбратьКлюч2()
            родитель.ВставитьВСебя(ключ1)
            ЗаменитьКлюч(ключ, ключ2)
        иначе
    ...
```

```
...
иначе
        брат2 = узел.НайтиХудогоБрата()
        если брат2 не пусто то
            ключ1 = брат2.ВыбратьКлюч1()
            ключ2 = родитель.ВыбратьКлюч2(ключ)
            родитель.ВставитьВСебя(ключ1)
            УдлитьКлючУСебя(ключ)
            Объединить(брат2)
иначе // если не лист
```

Найти брата

- Полный брат - узел справа или слева, имеющий такого же родителя, который содержит больше t-1 ключей
- Худой брат - узел справа или слева, имеющий такого же родителя, который содержит меньше t-1 ключей

Выбрать ключ у брата:

если это правый брат **то**

узел = 0

иначе

узел = ключи.размер-1

результат = <ключ, значение>[узел]

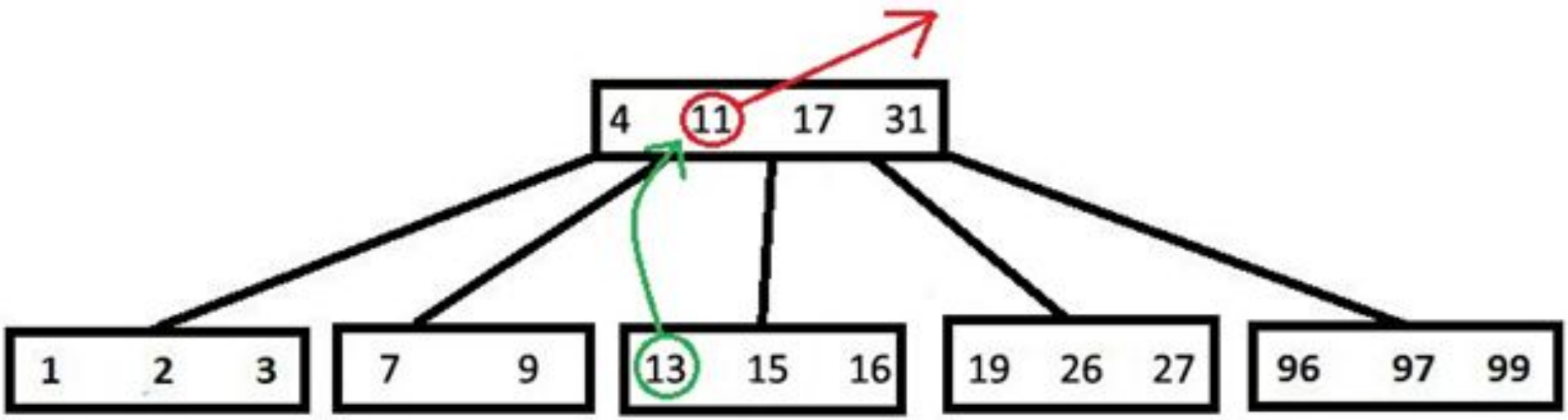
УдалитьУСебяПоИндексу(узел)

вернуть результат

Выбрать ключ у родителя

```
ВыбратьКлюч2(ключ):
    узел = НайтиКлюч(ключ)
    результат = <ключ, значение>[узел]
    УдалитьУСебяПоИндексу(узел)
    вернуть результат
```

Удаление из узла



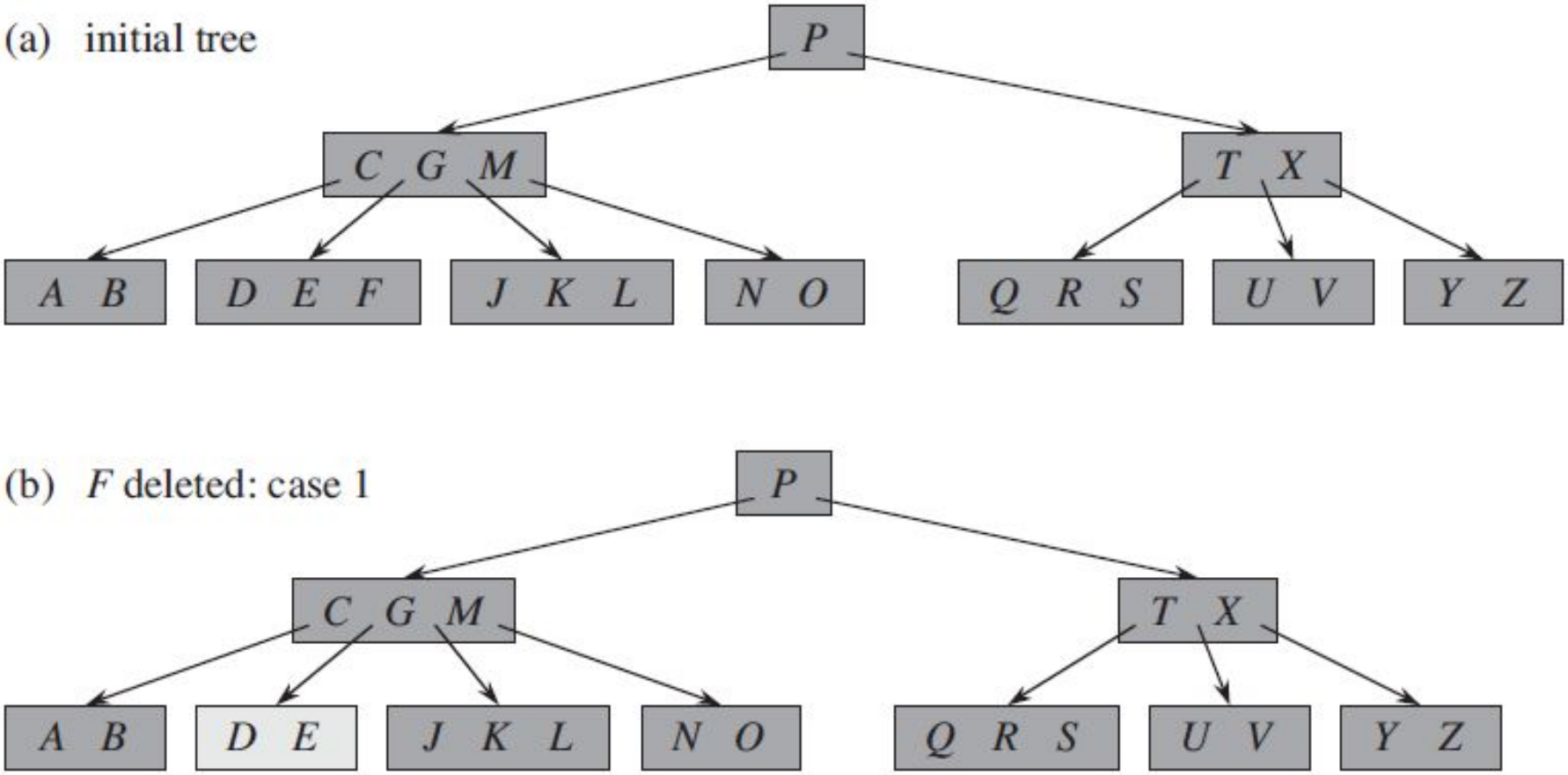
```
...
иначе // если не лист
    если потомки[узел].ключи > t-1  то
        потомок = потомки[узел]
        ключ1 = потомок.ВыбратьКлюч1()
        УдалитьКлючУСебя(ключ)
        ВставитьВСебя(ключ1)
    иначе
        ..
```

```
...
иначе
    брат = ПравыйБрат(потомки[узел])
    если потомки[брат].ключи > t-1  то
        потомок = потомки[узел]
        ключ1 = потомок.ВыбратьКлюч1()
        УдалитьКлючУСебя(ключ)
        ВставитьВСебя(ключ1)
    иначе
        ...
```

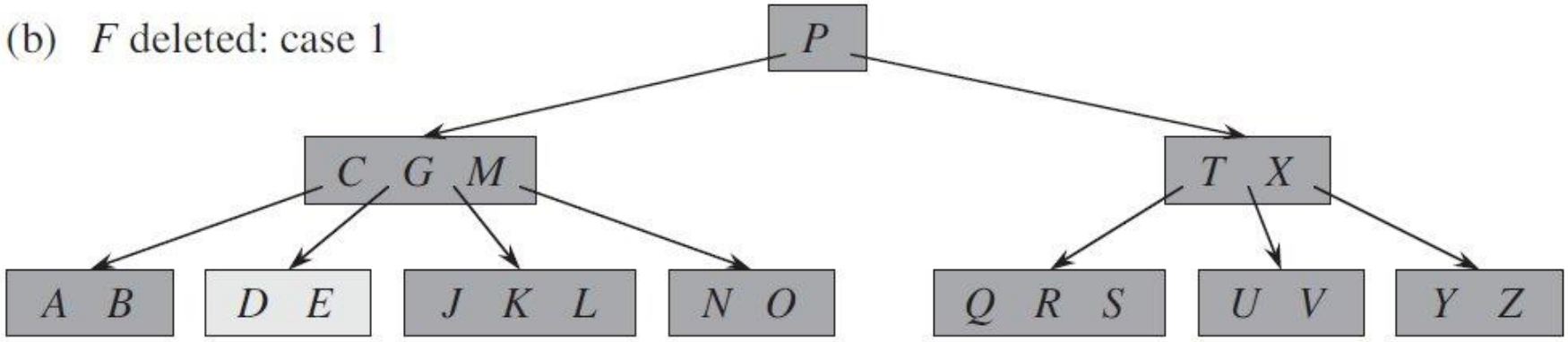
```
...
иначе
    брат = ПравыйБрат(потомки[узел])
    ОбъединитьСебя(брат)
    Удалить(ключ)
```

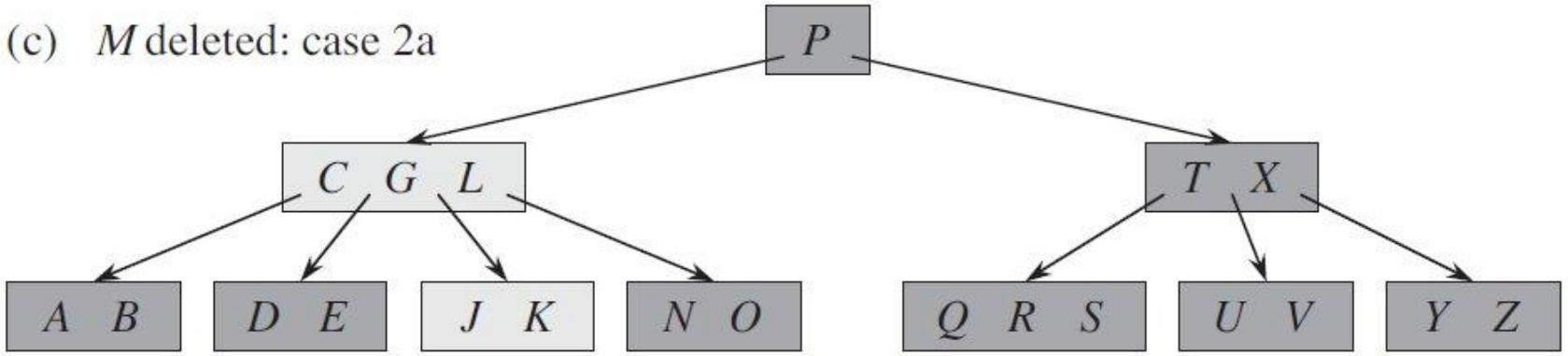
Рассмотреть случай, когда мы удалим последний ключ из последнего узла и надо будет занулить корень

Удаление

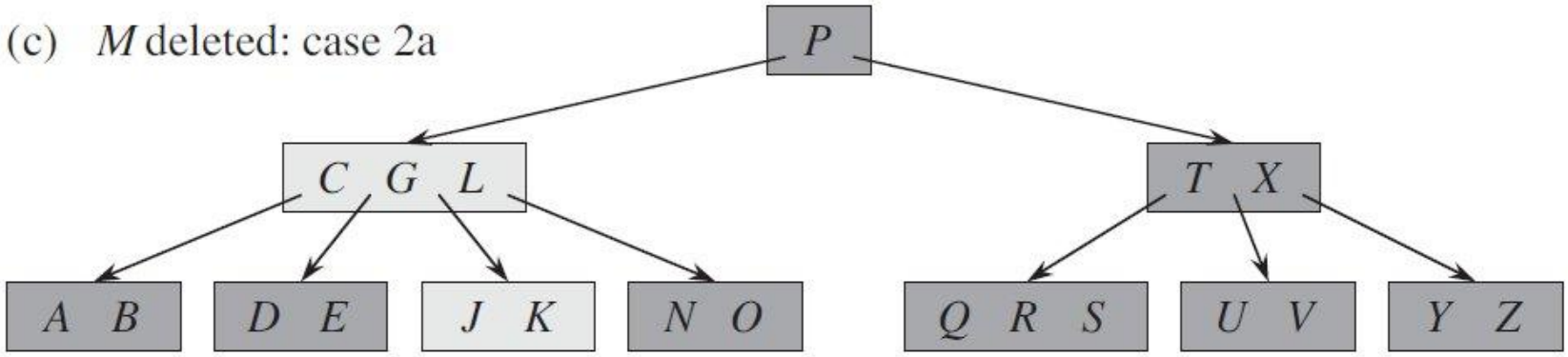

(a) initial tree
P
C G M
T X
A B
D E F
J K L
N O
Q R S
U V
Y Z
(b) F deleted: case 1
P
C G M
T X
A B
D E
J K L
N O
Q R S
U V
Y Z

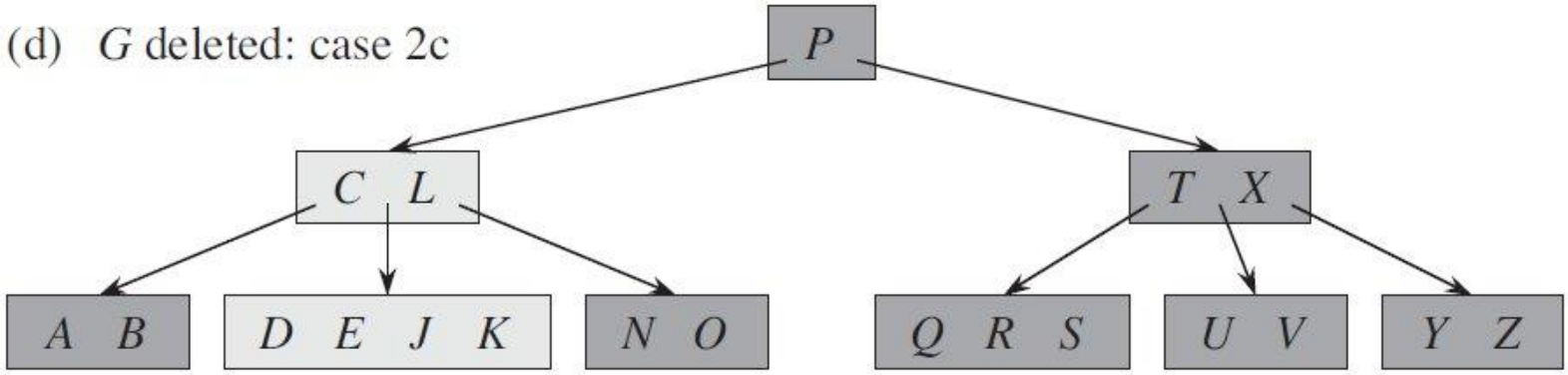
Удаление


(b) F deleted: case 1
P
C G M
T X
A B
D E
J K L
N O
Q R S
U V
Y Z


(c) M deleted: case 2a
P
C G L
T X
A B
D E
J K
N O
Q R S
U V
Y Z

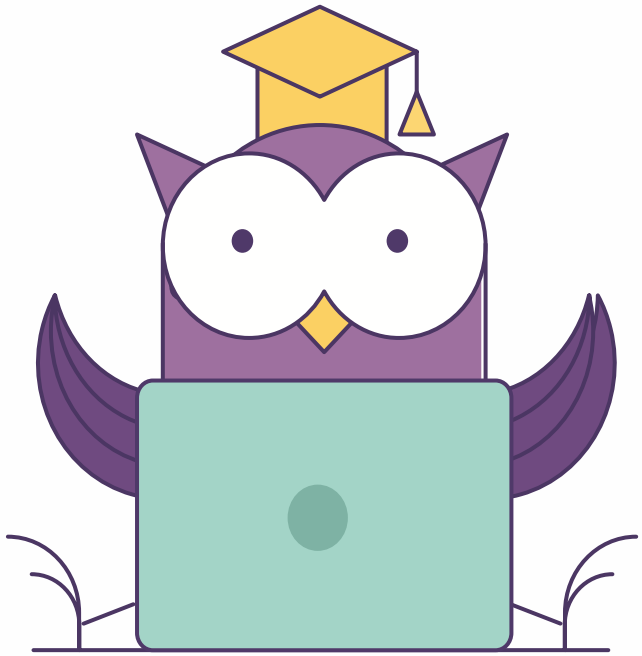
Удаление


(c) M deleted: case 2a
P
C G L
T X
A B
D E
J K
N O
Q R S
U V
Y Z


(d) G deleted: case 2c
P
C L
T X
A B
D E J K
N O
Q R S
U V
Y Z

Вопросы?

- Вопросы по B деревьям?

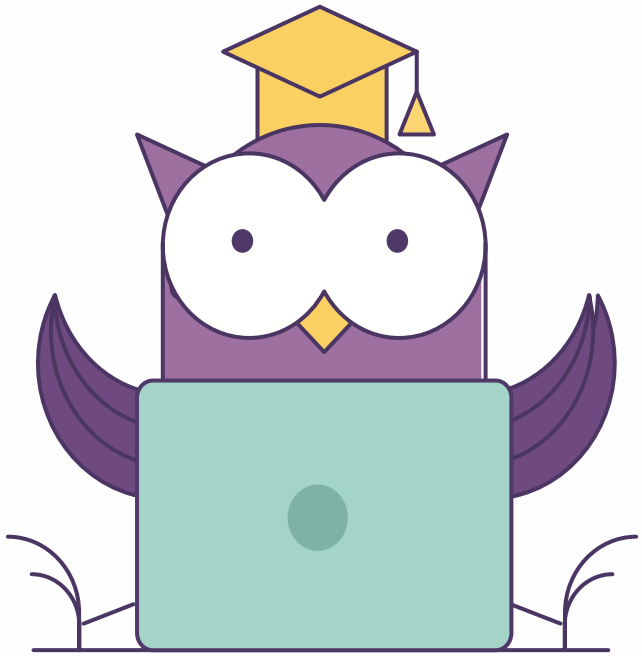


B* дерево

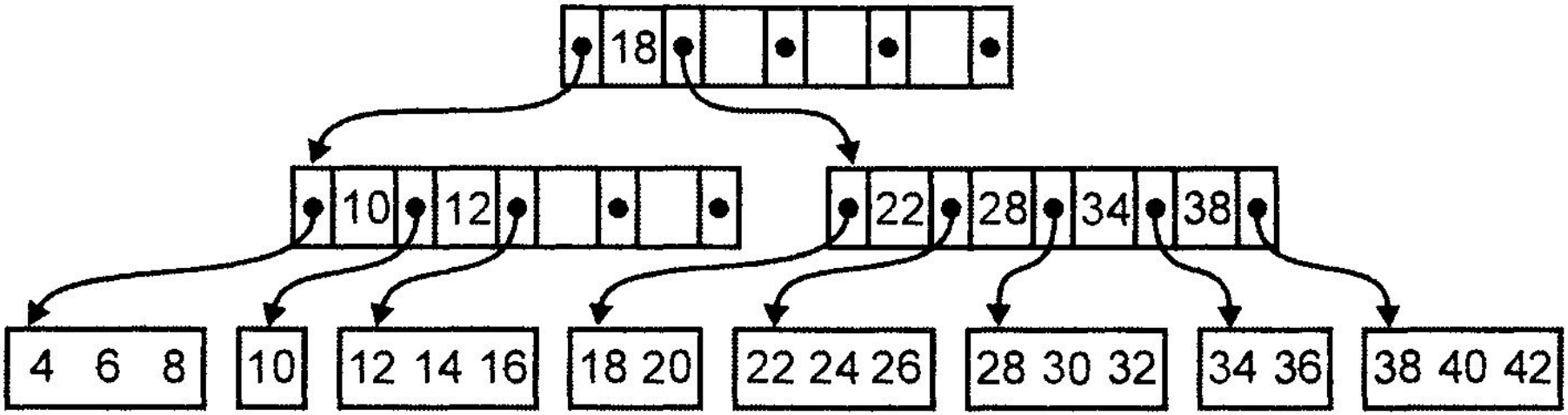
- Вариант B дерева, заполненного на 2/3
- При вставке, если узел полностью заполнен то вначале заполняем брата, а потом делим 2 узла на 3
- Удаление как в B-дереве только держим заполненность 2/3

Вопросы?

- Вопросы по B* деревьям?



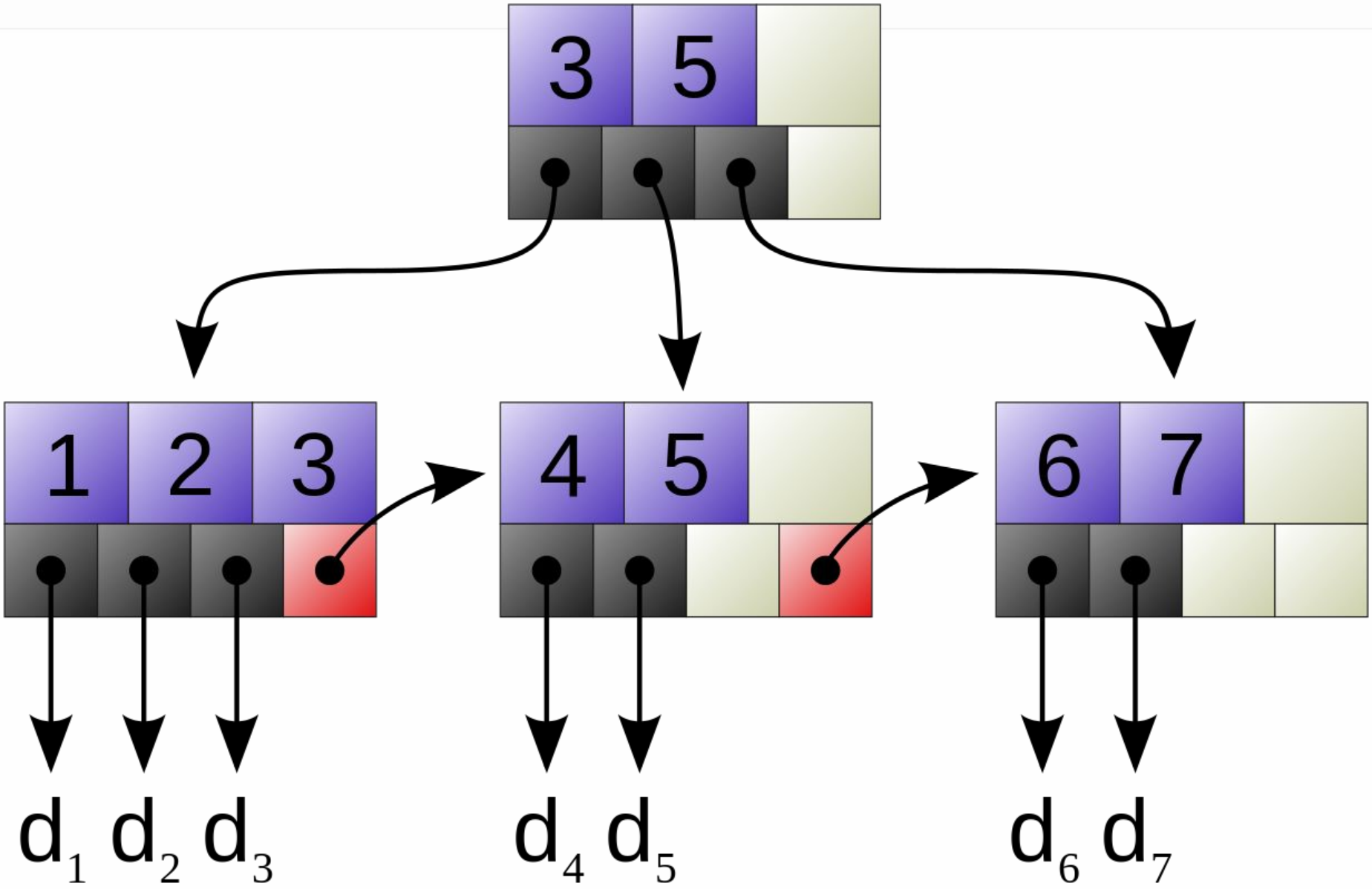
- Вариант B дерева, в котором значения сохраняются только в листовых (терминальных) узлах



B+ дерево

- 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

B+ дерево



Структура данных B+ дерева

```
class BTreeLeaf<Key, Value> {
  int size;
  bool isLeaf;
  Key[] keys;
  void* children[];
  BTreeLeaf *next;
}
```

Проверка на лист

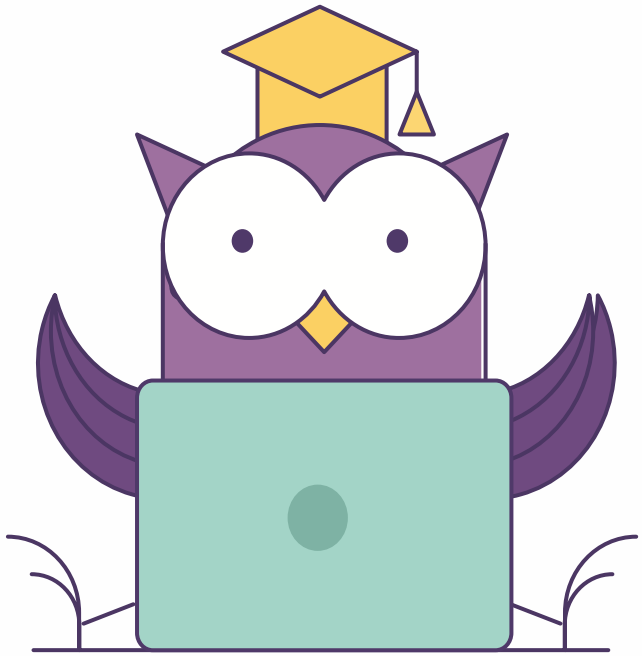
```
ЛистовойУзел():
    вернуть потомки.размер == 0

    вернуть являетсяЛистом
```

B+ дерево

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

Вопросы?

- Вопросы по B+ деревьям?



Сорри, дальше черновик

Домашнее задание

- Реализовать дерево оптимального поиска
- Алгоритм 1
- Алгоритм 2
- Опционально: Декартово дерево

Домашнее задание

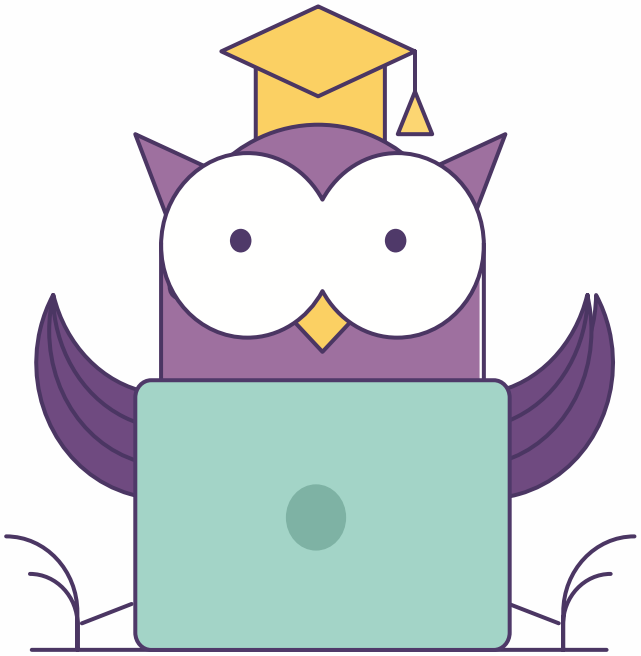
Сравнить производительность на dataset

- Построение
- Поиск

- Включить в общий тест

Total recall

- B дерево
- B* дерево
- B+ дерево
- Дерево оптимального поиска
- Матричное дерево
- Дерево отрезков



Заполните, пожалуйста, опрос о занятии

Спасибо
за внимание!