Алгоритмы и структуры данных-2

03

Вероятностные структуры данных
Фильтры и Skip-List

Нестеров Р.А., PhD, старший преподаватель
Департамент программной инженерии

ЯНВАРЬ 2024

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

План

01

Математическое ожидание и границы концентрации

02

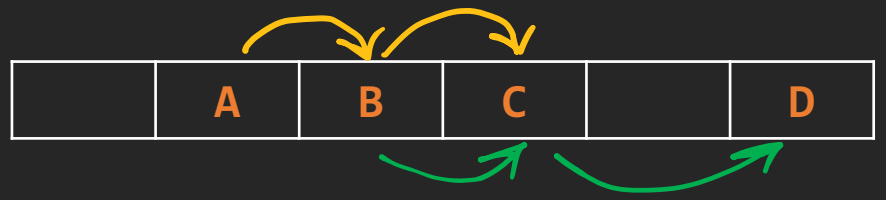
Фильтрация принадлежности объектов множеству

03

Связный список и логарифмическая сложность

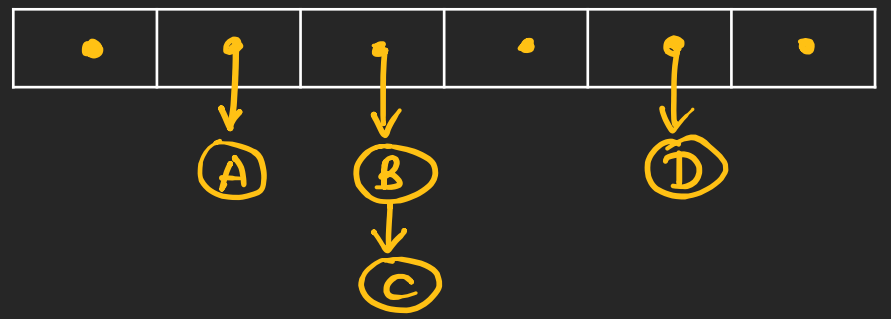
Об обработке коллизий в хеш-таблицах

ОТКРЫТАЯ АДРЕСАЦИЯ



- Все данные хранятся в **одной области** памяти
- Подходит для данных с **известными** свойствами

МЕТОД ЦЕПОЧЕК



- Простота и поддержка в **стандартных** библиотеках
- Подходит для данных с **любыми** свойствами

Давайте сыграем!

Игрок платит **1000 денег** за участие

С вероятностью $1:10^6$ игрок получит **10 миллиардов**, а иначе не получит ничего

Имеет ли смысл играть в такую игру?

Математическое ожидание суммы выигрыша составляет **9000 денег**...

$$E = \frac{10^{10}}{10^6} - 10^3 = 9000$$

Вместо того, чтобы оценивать среднюю [**ожидаемую**] результативность, будем искать **границы** вероятности **плохой** результативности.

Монету подбросили k раз.
В среднем, орел выпадает в $k/2$ случаев. Какова вероятность того, что орел никогда не выпадет?

$$P = \frac{1}{2^k}$$

QUICK SORT в среднем работает за $O(n \log n)$. Какова вероятность того, что его фактическое время работы превысит среднее?

$$P = O\left(\frac{1}{n}\right)$$

С высокой вероятностью [**w. h. p.**]

Случайное событие происходит с высокой вероятностью относительно некоторого n, если его вероятность составляет $1 - O(1/n)$.

- **QUICK SORT** работает за $O(n \log n)$ w. h. p.
- **Линейное пробирование** требует $O(\log n)$ времени w. h. p.
- **Хешировании кукушки** требует $O(\log n)$ обменов w. h. p.

Математическое ожидание против **w. h. p.**

Математическое ожидание дает **усредненную оценку** времени работы алгоритма

W. H. P. дает относительно **точные гарантии** того, как будет работать алгоритм

Фильтр принадлежности

Обеспечивает **«хранение»** множества объектов S размера n

Отвечает на запросы о **принадлежности** объекта исходному множеству S

Не предоставляет возможность **прямого доступа** к содержанию объектов

Гарантия №1 – без ошибок второго рода

Ложно-отрицательные срабатывания не допускаются!

Если объект q содержится в исходном множестве S, то фильтр всегда отвечает $q \in S$.

Детерминированные структуры данных подходят под этот критерий.

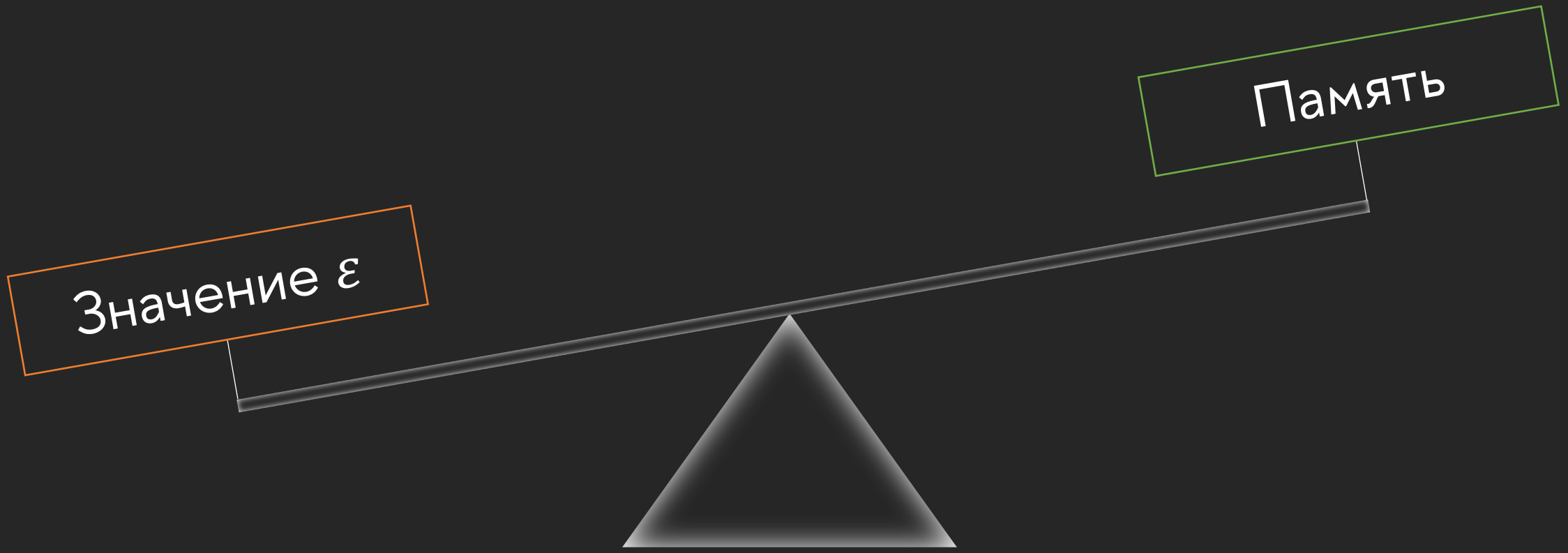
Гарантия №2 – ошибки первого рода

Ложно-положительные срабатывания возможны...

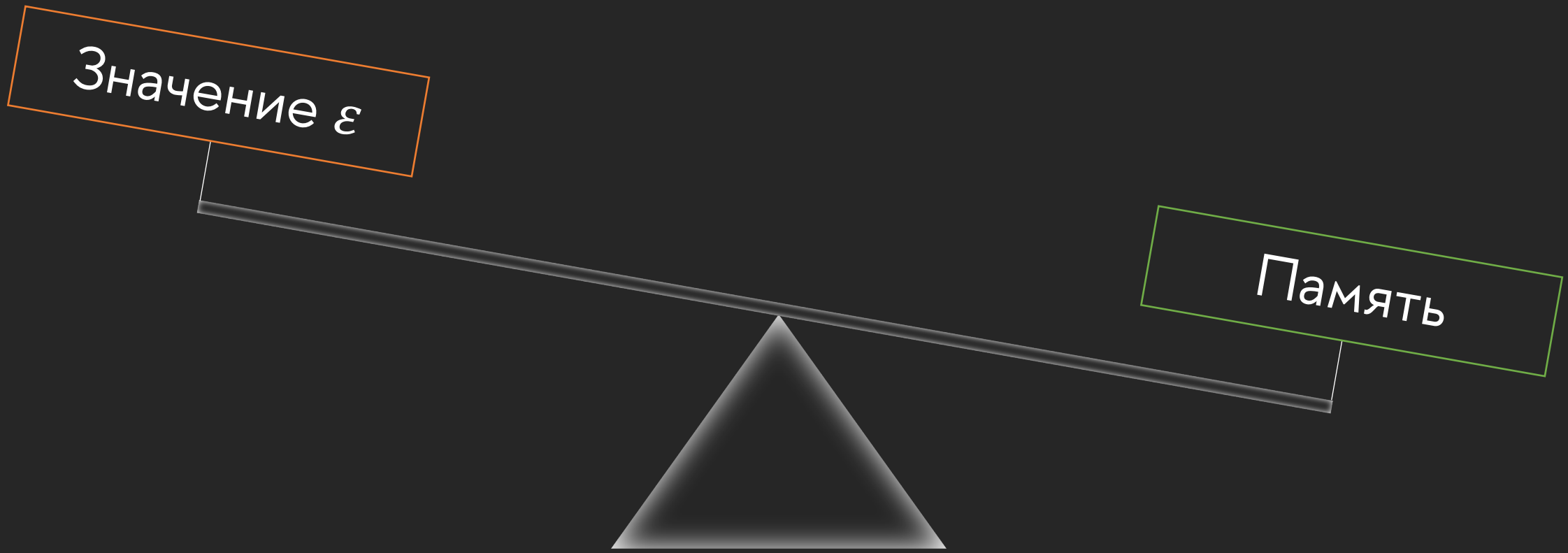
Если объект q НЕ содержится в исходном множестве S, то фильтр отвечает $q \in S$ с заданной вероятностью $\varepsilon < 1$.

Для детерминированных структур данных очевидно, что обеспечивается $\varepsilon = 0$.

Основной компромисс



Основной компромисс



Фильтр Блума

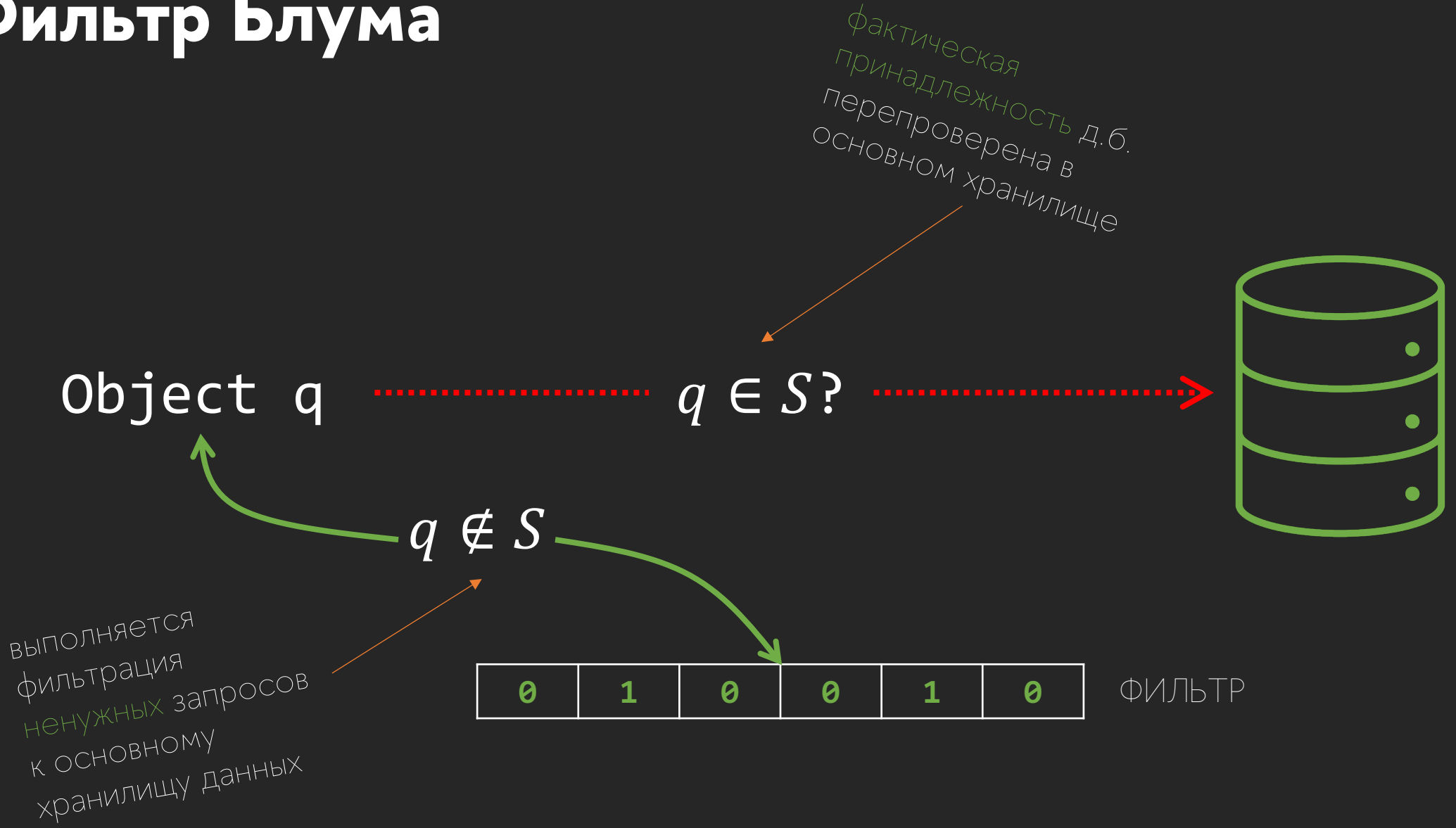
Представлен в 1970 г. Бёртоном Говардом Блумом

1. Теоретическая оценка требуемой памяти – $1.44 \cdot n \cdot \log_2(1/\varepsilon)$ бит
2. Затраты по памяти **никак не соотносятся** с размером объектов!

Фильтр Блума

Object q ····· SEARCH(q) ····>

Фильтр Блума



Фильтр Блума

Использование $O(n \log 1/\varepsilon)$ бит локальной памяти позволяет отфильтровать $1 - \varepsilon$ запросов $q \notin S$

Снижается общее число запросов к основной памяти, что улучшает общие временные затраты

Фильтр Блума – что внутри?

Пусть мощность множества S составляет n,
а вероятность ложно-положительного срабатывания ε.

1. Для идентификации объектов используется $k = \log_2 1/\varepsilon$ хеш-функций $h_1, h_2, \ldots, h_k$
2. Информация о принадлежности хранится в битовом массиве размера $m = nk \cdot \log_2 e = 1.44n \cdot \log_2 1/\varepsilon$

Фильтр Блума для вредоносных сайтов

Инициализируем фильтр нулевыми (ложными) значениями

InitializeBloomFilter.cpp

```cpp
std::vector<std::vector<bool>> table;

for (size_t i = 0; i < k; ++i) {
    for (size_t j = 0; j < m; ++i) {
        table[i][j] = 0;
    }
}
```

	0	1	2	3	4
t1	0	0	0	0	0
t2	0	0	0	0	0
t3	0	0	0	0	0

`std::vector<bool>` VERSUS
`std::bitset`

Фильтр Блума для вредоносных сайтов

Устанавливаем значения битов по вычисленным хеш-кодам

InsertBloomFilter.cpp

```cpp
// массив указаталей на хеш-функции h1, h2, h3
std::vector<size_t(*)(std::string)> funcVec;

void INSERT(const std::string& obj) {
    for (size_t i = 0; i < k; ++i) {
        table[i][funcVec[i](obj)] = 1;
    }
}
```

INSERT(“thisisavirus.com”)

	0	1	2	3	4
t1	0	0	1	0	0
t2	0	1	0	0	0
t3	0	0	0	0	1

h1(obj) = 2
h2(obj) = 1
h3(obj) = 4

Фильтр Блума для вредоносных сайтов

В случае коллизии значение бита не поменяется

InsertBloomFilter.cpp

```cpp
// массив указаталей на хеш-функции h1, h2, h3
std::vector<size_t(*)(std::string)> funcVec;

void INSERT(const std::string& obj) {
    for (size_t i = 0; i < k; ++i) {
        table[i][funcVec[i](obj)] = 1;
    }
}
```

INSERT(“goodwebsite.com”)

	0	1	2	3	4
t1	0	1	1	0	0
t2	1	1	0	0	0
t3	0	0	0	0	1

h1(obj) = 1
h2(obj) = 0
h3(obj) = 4

Фильтр Блума для вредоносных сайтов

CONTAINS(“google.com”) → NO!

InsertBloomFilter.cpp

```cpp
bool CONTAINS(const std::string& obj) {
    bool found = true;
    for (size_t i = 0; i < k; ++i) {
        if (!(table[i][funcVec[i](obj)])) {
            found = false;
            break;
        }
    }
    return found;
}
```

	0	1	2	3	4
t1	0	1	1	0	0
t2	1	1	0	0	0
t3	0	0	0	0	1

h1(obj) = 1
h2(obj) = 2
h3(obj) = 4

Фильтр Блума для вредоносных сайтов

CONTAINS(“verynormalsite.com”) → FALSE POSITIVE!

C++ InsertBloomFilter.cpp

```cpp
bool CONTAINS(const std::string& obj) {
    bool found = true;
    for (size_t i = 0; i < k; ++i) {
        if (!(table[i][funcVec[i](obj)])) {
            found = false;
            break;
        }
    }
    return found;
}
```

	0	1	2	3	4
t1	0	1	1	0	0
t2	1	1	0	0	0
t3	0	0	0	0	1

h1(obj) = 2
h2(obj) = 0
h3(obj) = 4

НА ДОСКЕ

ЭПИЗОД 1

Оценим FPP [**вероятность ложно-положительного срабатывания**] для фильтра Блума

Фильтр Блума для вредоносных сайтов

Можем ли мы удалить данные о сайте verynormalsite.com?

Коллизии в хеш-функциях не дают возможности удалять объекты из фильтра

	0	1	2	3	4
t1	0	1	1	0	0
t2	1	1	0	0	0
t3	0	0	0	0	1

а также удаление приведет к ложно-отрицательным срабатываниям по объектам, которые есть в множестве S, но еще не были добавлены в фильтр...

```
h1(obj) = 2
h2(obj) = 0
h3(obj) = 4
```

Анализ потоковых данных

размер потока данных
заранее неизвестен!

- где найти партнера для бала на ФКН
- как правильно вести соцсети кота
- бал на ФКН где арендовать смокинг
- КОМУ МОЛИТЬСЯ ЧТОБЫ ВСЕ УСПЕТЬ
- к чему снится доктор наук
- бал на ФКН 12.04 найти партнершу
- почему собака

Структура данных **Sketch** используется для **частотного** анализа потока данных

В этом случае нам **недостаточно** ответа на вопрос о [**не**]принадлежности

Count-Min Sketch

Матричная структура образована k хеш-функциями $h_1, h_2, \ldots, h_k$, которым соответствуют определенные строки

	0	1	2	3	4	5	…	m
h1	0	0	0	0	0	0	…	0
h2	0	0	0	0	0	0	…	0
…	…	…	…	…	…	…	…	…
hk	0	0	0	0	0	0	0	0

Count-Min Sketch

Значения в ячейках $a[i][j]$ матрицы A хранят не бинарные значения, а то количество раз, сколько i-ая хеш-функция давала значение j

	0	1	2	3	4	5	…	m
h1	0	6	7	8	5	1	…	3
h2	1	1	5	8	9	1	…	3
…	…	…	…	…	…	…	…	…
hk	6	7	14	1	4	5	41	1

Count-Min Sketch – Пример #1

```
h1 = k % 7
h2 = (k + 3 * (k % 2)) % 7
h3 = abs(k - 4) % 7
```

Определить состояние структуры после вставки объектов из множества $S = \{1, 3, 8, 16\}$

	0	1	2	3	4	5	6
h1	0	0	0	0	0	0	0
h2	0	0	0	0	0	0	0
h3	0	0	0	0	0	0	0

Count-Min Sketch – Пример #1

```
h1 = k % 7
h2 = (k + 3 * (k % 2)) % 7
h3 = abs(k - 4) % 7
```

Определить состояние структуры после вставки объектов из множества $S = \{1, 3, 8, 16\}$

	0	1	2	3	4	5	6
h1	0	1	0	0	0	0	0
h2	0	0	0	0	1	0	0
h3	0	0	0	1	0	0	0

```
h1(1) = 1
h2(1) = 4
h3(1) = 3
```

Count-Min Sketch – Пример #1

Определить состояние структуры после вставки объектов из множества $S = \{1, 3, 8, 16\}$

```
h1 = k % 7
h2 = (k + 3 * (k % 2)) % 7
h3 = abs(k - 4) % 7
```

	0	1	2	3	4	5	6
h1	0	1	0	1	0	0	0
h2	0	0	0	0	1	0	1
h3	0	1	0	1	0	0	0

```
h1(3) = 3
h2(3) = 6
h3(3) = 1
```

Count-Min Sketch – Пример #1

Определить состояние структуры после вставки объектов из множества $S = \{1, 3, 8, 16\}$

```
h1 = k % 7
h2 = (k + 3 * (k % 2)) % 7
h3 = abs(k - 4) % 7
```

	0	1	2	3	4	5	6
h1	0	2	0	1	0	0	0
h2	0	1	0	0	1	0	1
h3	0	1	0	1	1	0	0

```
h1(8) = 1
h2(8) = 1
h3(8) = 4
```

Count-Min Sketch – Пример #1

Определить состояние структуры после вставки объектов из множества $S = \{1, 3, 8, 16\}$

```
h1 = k % 7
h2 = (k + 3 * (k % 2)) % 7
h3 = abs(k - 4) % 7
```

	0	1	2	3	4	5	6
h1	0	2	1	1	0	0	0
h2	0	1	1	0	1	0	1
h3	0	1	0	1	1	1	0

```
h1(16) = 2
h2(16) = 2
h3(16) = 5
```

Count-Min Sketch – Пример #2

Определить состояние структуры после вставки объектов из **потока**

$S = \{\dots, 1, 3, 8, 16, \dots\}$

Оценить частоты появления этих элементов в потоке

```
h1 = k % 7
h2 = (k + 3 * (k % 2)) % 7
h3 = abs(k - 4) % 7
```

	0	1	2	3	4	5	6
h1	7	9	10	1	5	7	6
h2	4	8	7	5	4	10	7
h3	15	12	2	5	8	2	1

```
h1(…) = …
h2(…) = …
h3(…) = …
```

Count-Min Sketch – Пример #2

Определить состояние структуры после вставки объектов из **потока**

$S = \{\dots, 1, 3, 8, 16, \dots\}$

Оценить частоты появления этих элементов в потоке

```
h1 = k % 7
h2 = (k + 3 * (k % 2)) % 7
h3 = abs(k - 4) % 7
```

	0	1	2	3	4	5	6
h1	7	9	10	1	5	7	6
h2	4	8	7	5	4	10	7
h3	15	12	2	5	8	2	1

```
h1(1) = 1
h2(1) = 4
h3(1) = 3
```

Count-Min Sketch – Пример #2

Определить состояние структуры после вставки объектов из **потока**

$S = \{\dots, 1, 3, 8, 16, \dots\}$

Оценить частоты появления этих элементов в потоке

```
h1 = k % 7
h2 = (k + 3 * (k % 2)) % 7
h3 = abs(k - 4) % 7
```

	0	1	2	3	4	5	6
h1	7	9	10	1	5	7	6
h2	4	8	7	5	5	10	7
h3	15	12	2	5	8	2	1

```
h1(1) = 1
h2(1) = 4
h3(1) = 3
```

```
COUNT(1) = 5
```

Count-Min Sketch – Пример #2

Определить состояние структуры после вставки объектов из **потока**

$S = \{\dots, 1, 3, 8, 16, \dots\}$

Оценить частоты появления этих элементов в потоке

```
h1 = k % 7
h2 = (k + 3 * (k % 2)) % 7
h3 = abs(k - 4) % 7
```

	0	1	2	3	4	5	6
h1	7	9	10	2	5	7	6
h2	4	8	7	5	5	10	7
h3	15	12	2	5	8	2	1

```
h1(3) = 3
h2(3) = 6
h3(3) = 1
```

Count-Min Sketch – Пример #2

Определить состояние структуры после вставки объектов из **потока**

$S = \{\dots, 1, 3, 8, 16, \dots\}$

Оценить частоты появления этих элементов в потоке

```
h1 = k % 7
h2 = (k + 3 * (k % 2)) % 7
h3 = abs(k - 4) % 7
```

	0	1	2	3	4	5	6
h1	7	9	10	3	5	7	6
h2	4	8	7	5	5	10	7
h3	15	12	2	5	8	2	1

```
h1(3) = 3
h2(3) = 6
h3(3) = 1
```

```
COUNT(3) = 3
```

Count-Min Sketch – Пример #2

Определить состояние структуры после вставки объектов из **потока**

$S = \{\dots, 1, 3, 8, 16, \dots\}$

Оценить частоты появления этих элементов в потоке

```
h1 = k % 7
h2 = (k + 3 * (k % 2)) % 7
h3 = abs(k - 4) % 7
```

	0	1	2	3	4	5	6
h1	7	9	10	3	5	7	6
h2	4	8	7	5	5	10	7
h3	15	12	2	5	8	2	1

```
COUNT(25) = min(4, 5, 15) = 4
```

Да это же **фильтр Блума с подсчетом**! В нем доступна возможность удаления с некоторыми ограничениями...

	0	1	2	3	4	5	6
h1	7	9	10	3	5	7	6
h2	4	8	7	5	5	10	7
h3	15	12	2	5	8	2	1

Удаление объекта в этом случае возможно

Да это же **фильтр Блума с подсчетом**! В нем доступна возможность удаления с некоторыми ограничениями...

	0	1	2	3	4	5	6
h1	7	8	10	3	5	7	6
h2	4	8	7	4	5	10	7
h3	15	12	2	5	8	1	1

Удаление объекта в этом случае возможно

Да это же **фильтр Блума с подсчетом**! В нем доступна возможность удаления с некоторыми ограничениями...

	0	1	2	3	4	5	6
h1	7	8	10	3	5	7	6
h2	4	8	7	4	5	10	7
h3	15	12	2	5	8	1	1

Удаление объекта может привести к **ложно-отрицательному срабатыванию** фильтра

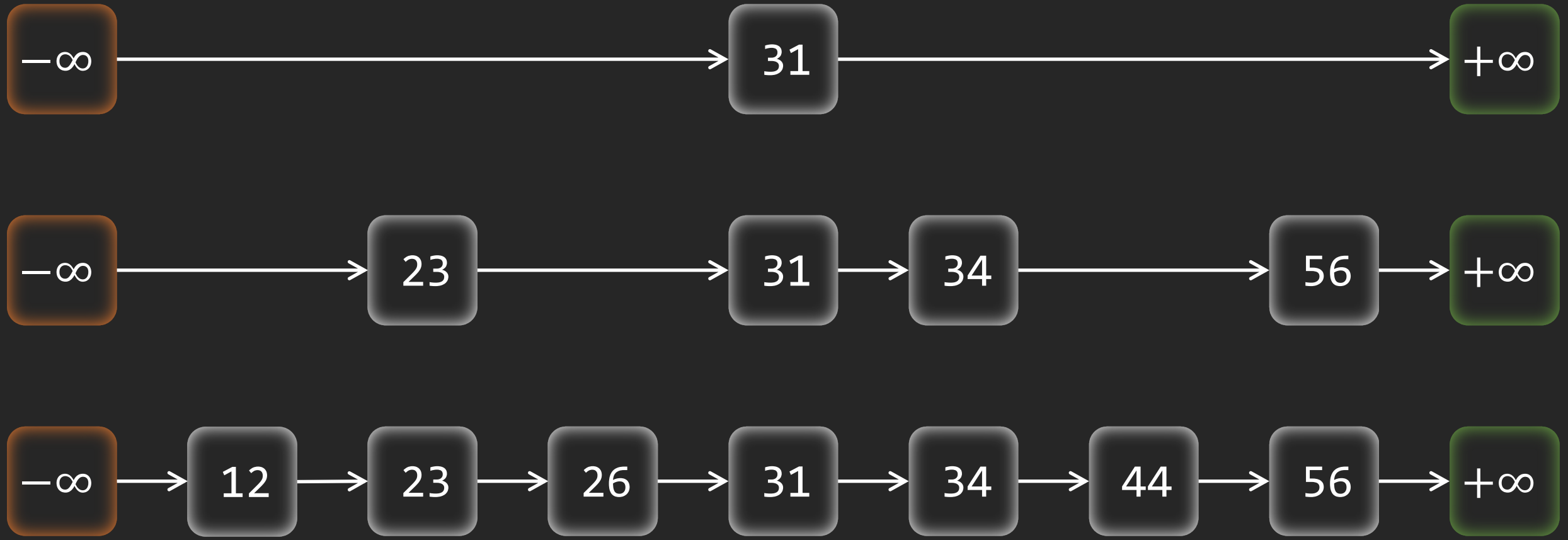
Список с пропусками [Skip-List]

Предложен Уильямом Пью в 1989 г.

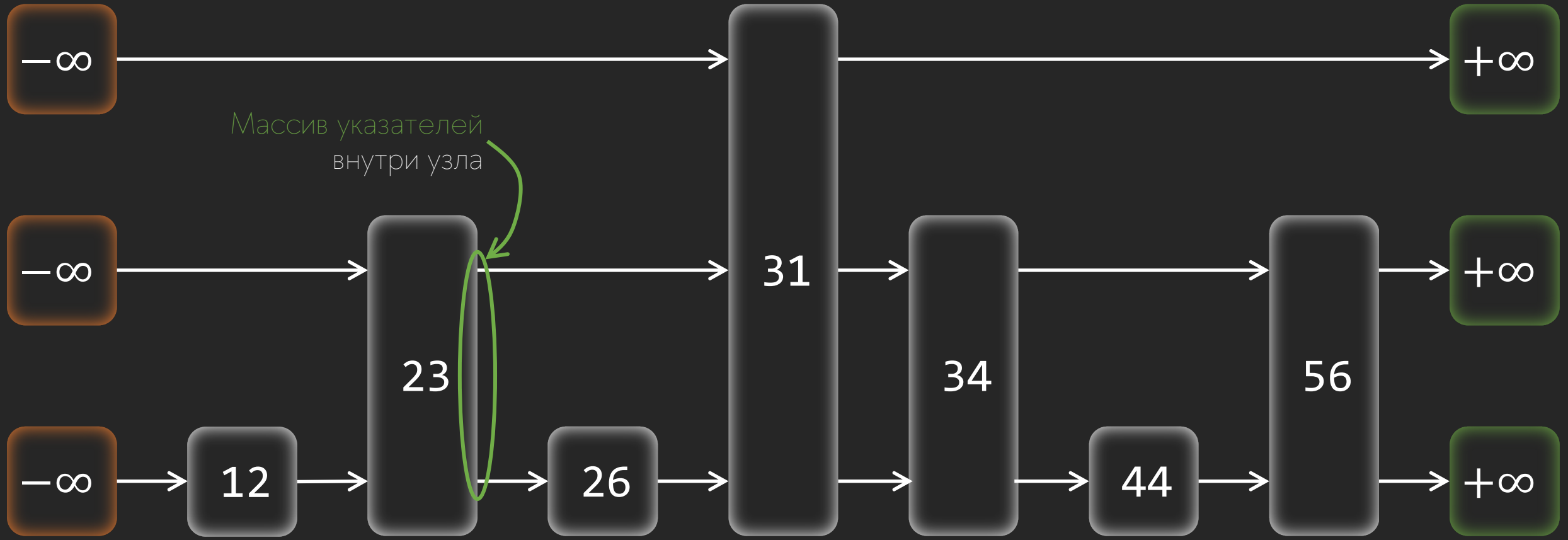
Представляет вероятностную реализацию **ADT Словарь** – INSERT, SEARCH, DELETE

Ожидаемое время работы основных операций составляет $O(\log n)$

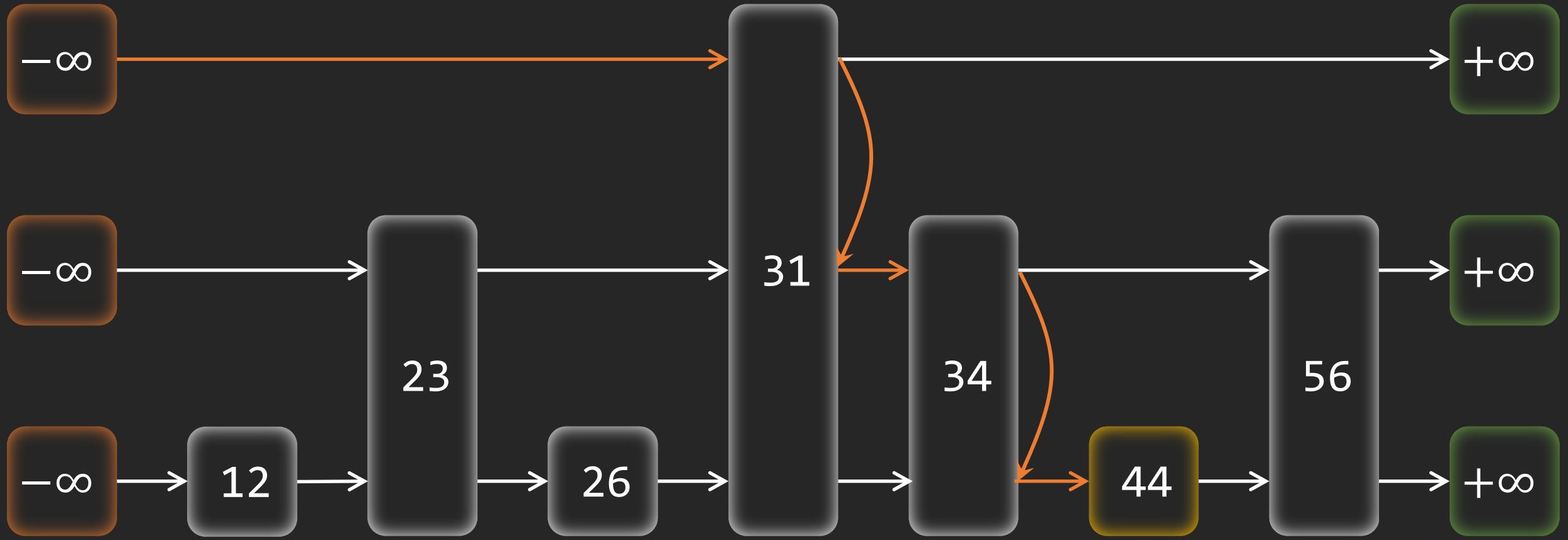
Список с пропусками [Skip-List]



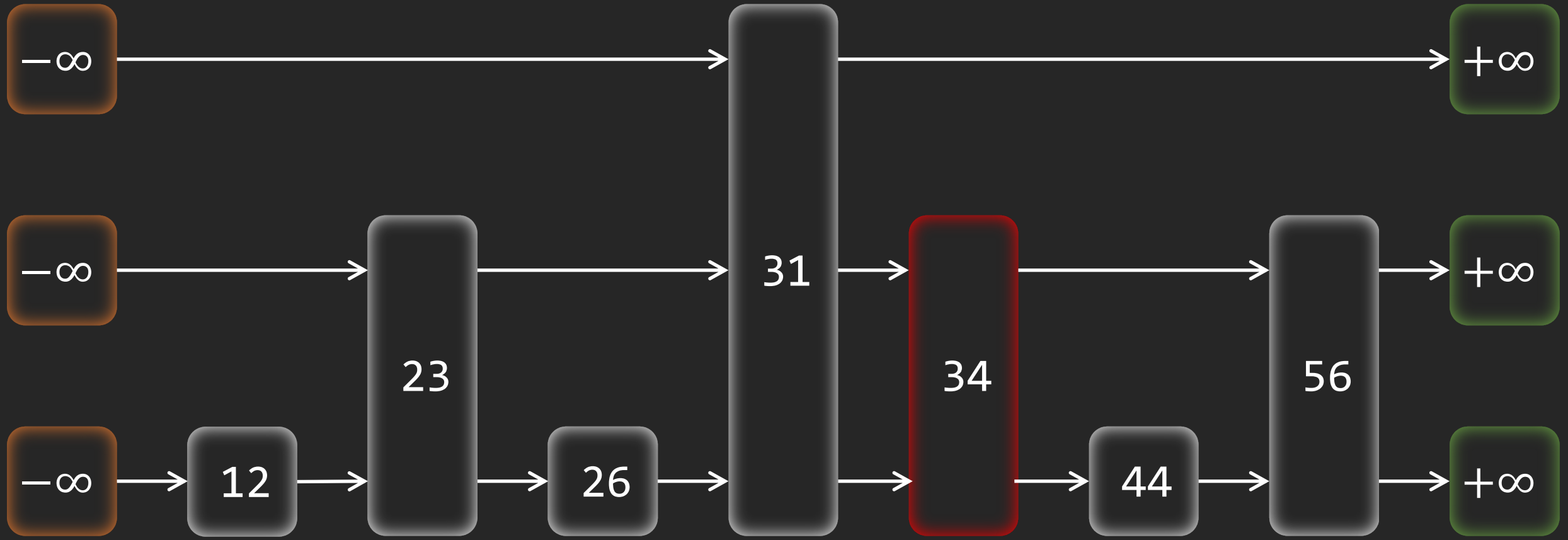
Об удобствах реализации



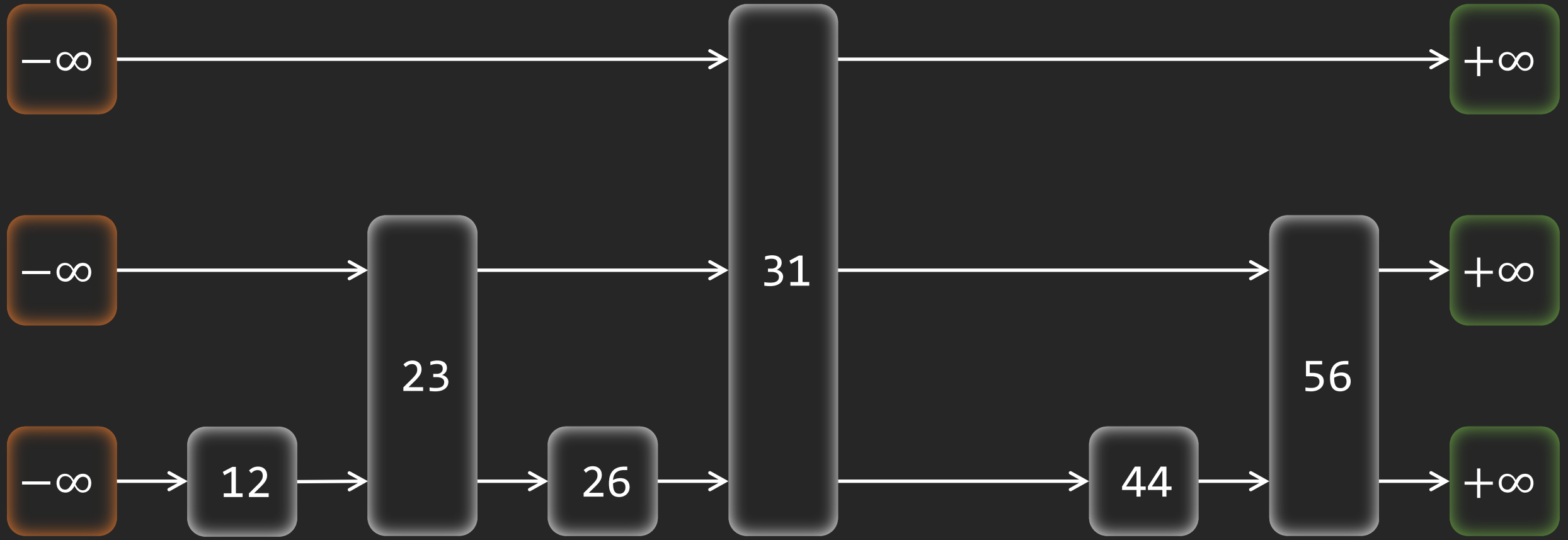
SEARCH(44)



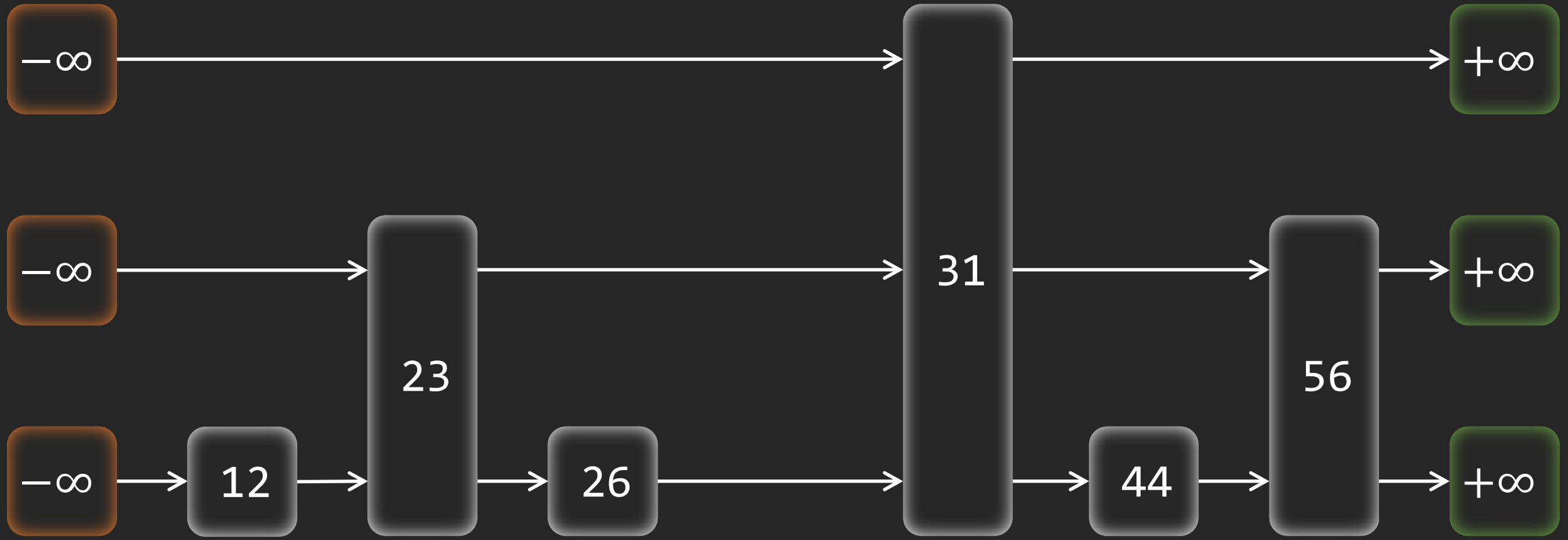
DELETE(34)



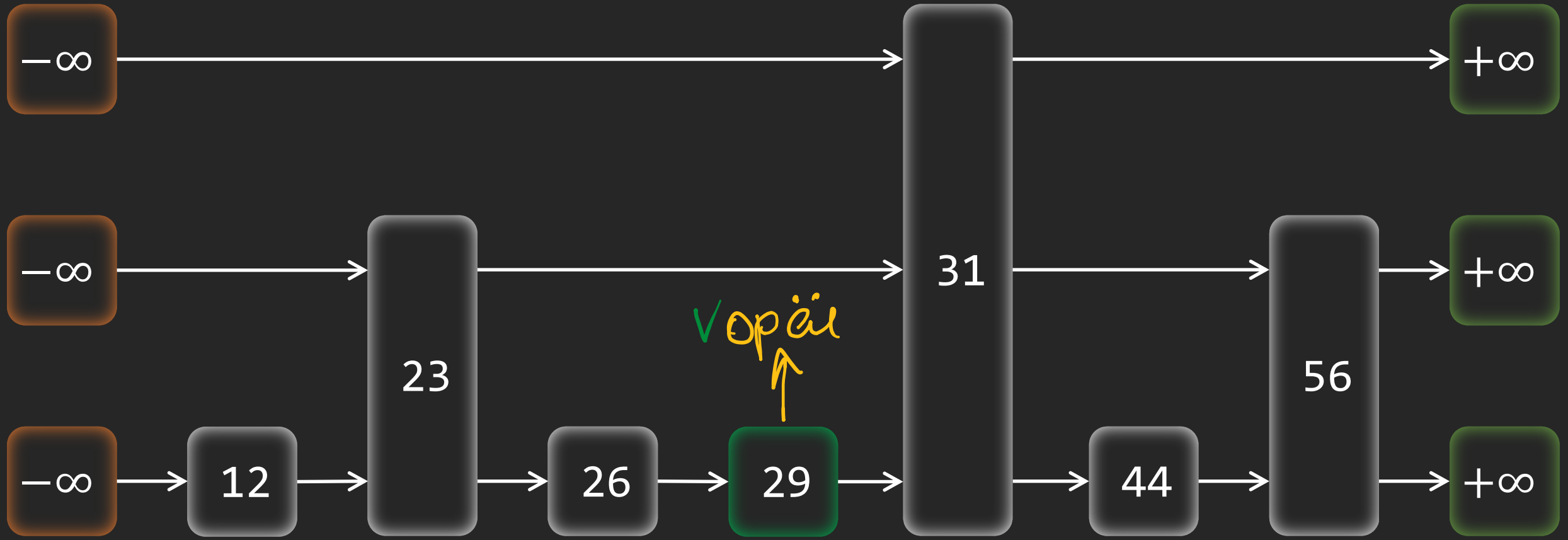
DELETE(34)



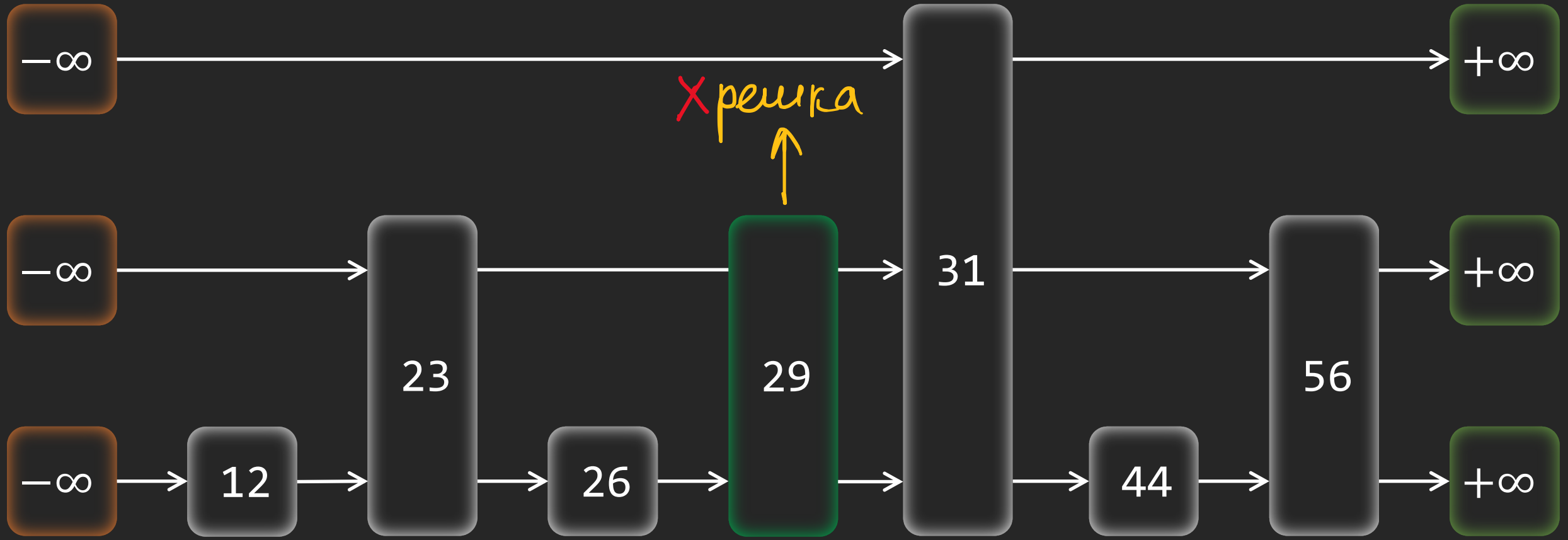
INSERT(29)



INSERT(29)

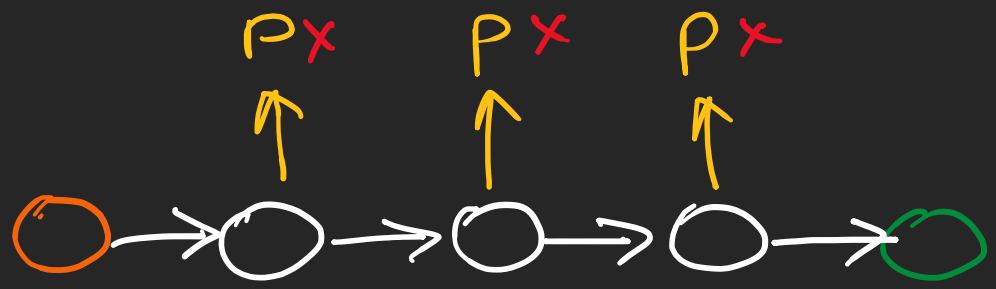


INSERT(29)

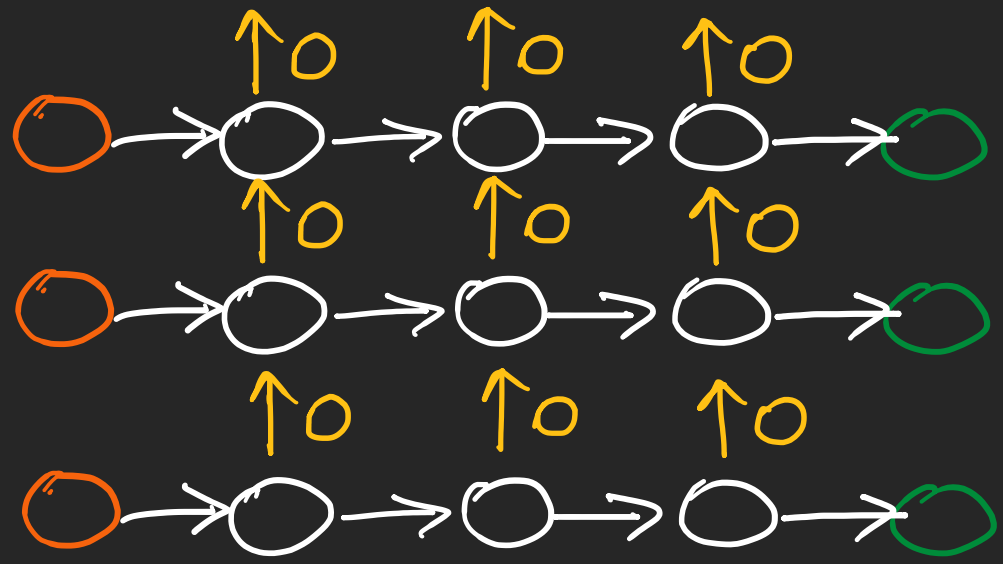


Вырожденные ситуации

ПРОСТОЙ СПИСОК



??? ПРОСТЫХ СПИСКОВ



Нам обязательно требуется параметр `maxHeight`,
ограничивающий рост высоты списка

НА ДОСКЕ

ЭПИЗОД 2

Оценим **сложностные характеристики** списка с пропусками

Истина слишком сложна,
чтобы допускать что-либо,
кроме **приближений**.

Джон фон Нейман