Алгоритмы и структуры данных-2

Хеширование-3.
Вероятностные структуры данных

Практическое занятие 16 27.01–01.02.2024

2024-2025 учебный год

План

Обычное хеширование и **фильтр Блума**

Фильтр **кукушки**: хранение «отпечатков»

Многоуровневый **список с пропусками**:
логарифмическая сложность

Warm-up

FAQ по фильтрам

1. Что же **фильтрует** фильтр Блума/фильтр кукушки?
2. Какие операции **точно поддерживает** стандартная реализация фильтра Блума?
3. В чем состоит проблема **удаления** элементов из фильтра Блума стандартной реализации?
4. В чем разница между хеш-таблицей и фильтром?

Фильтр Блума

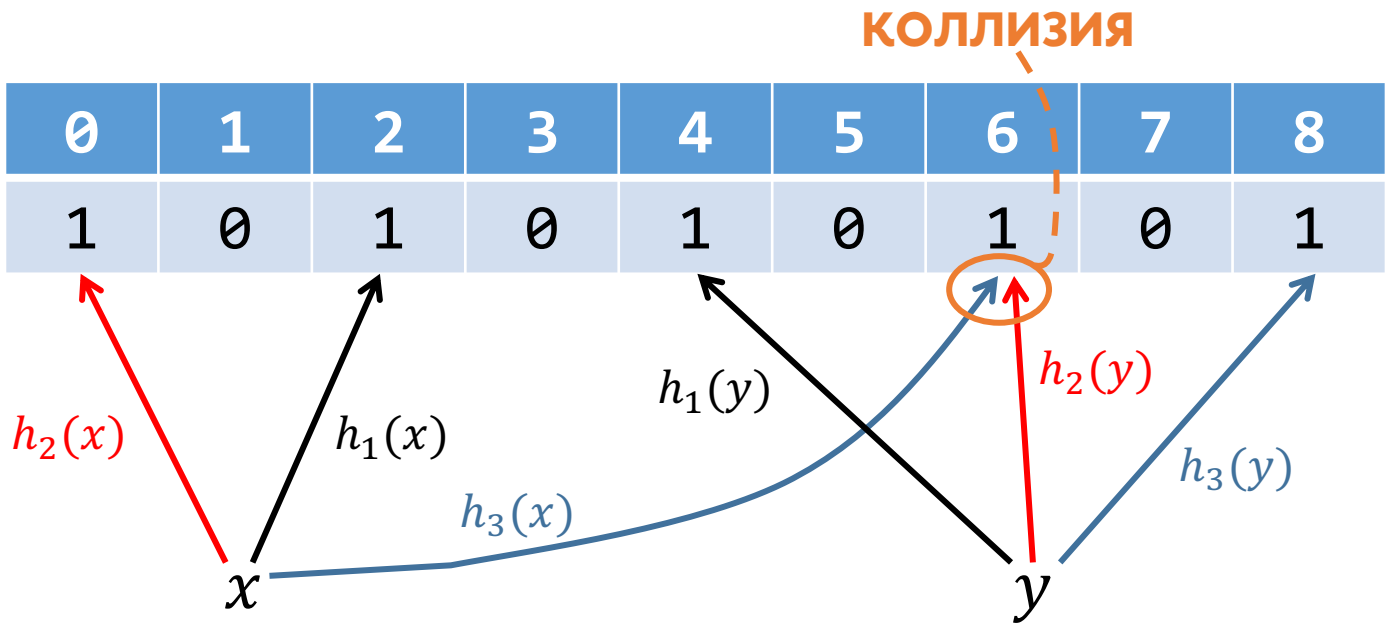
Устройство фильтра Блума

ε — доля ложно-положительных срабатываний

- k хеш-функций $h_1, h_2, \ldots, h_k$, где $k = \log(1/\varepsilon)$ и $h_i: U \to \{0, m-1\}$
- массив(-ы) из m битов, где $m = (\ln 2)nk = 1.44 \cdot nk$,
  и n — количество объектов в множестве U

ПРИМЕР КОНФИГУРАЦИИ

- $n = 2$ объекта и $\varepsilon = 1/8$
- $k = \log 8 = 3$ хеш-функции
- $m = 1.44 \cdot 3 \cdot 2 \approx 9$ бит

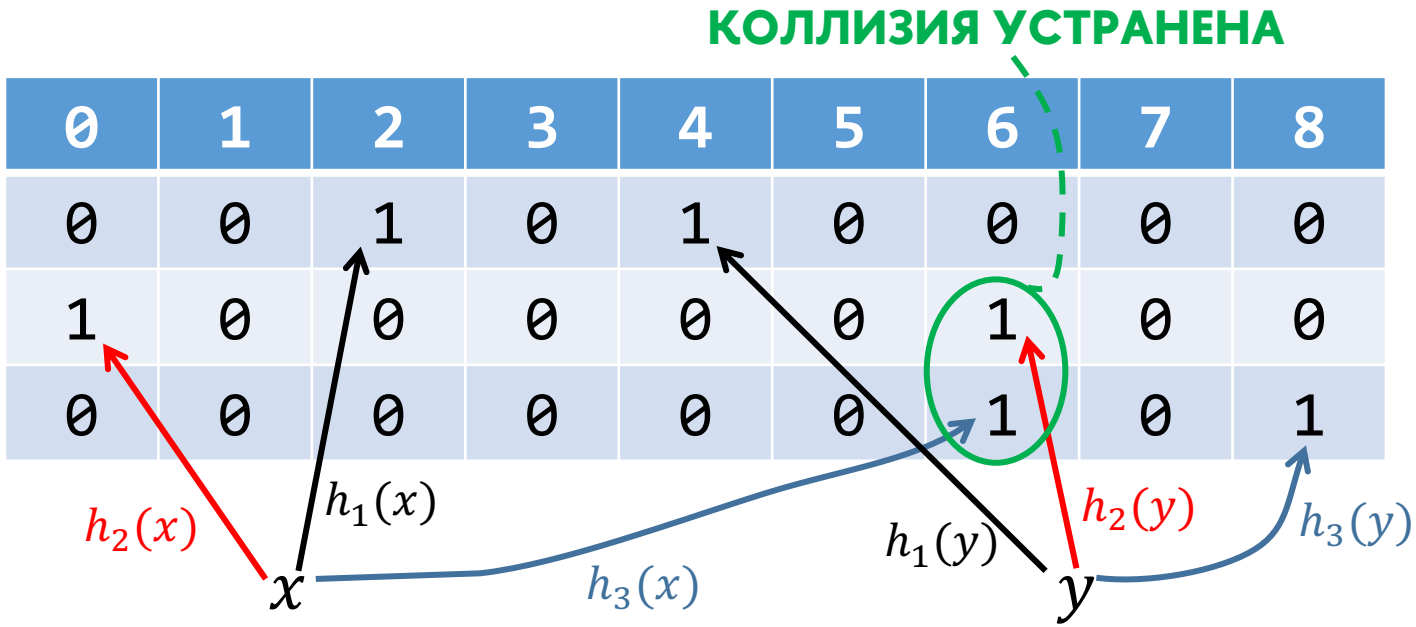


Устройство фильтра Блума

ε — доля ложно-положительных срабатываний

- k хеш-функций $h_1, h_2, \ldots, h_k$, где $k = \log(1/\varepsilon)$ и $h_i: U \to \{0, m-1\}$
- массив(-ы) из m битов, где $m = (\ln 2)nk = 1.44 \cdot nk$,
  и n — количество объектов в множестве U

- для каждой хеш-функции можно использовать **отдельный массив**
- снижение количества **коллизий** значений разных хеш-функций



как проверить принадлежность объекта множеству с помощью фильтра Блума?

0	1	2	3	4	5	6	7	8
1	0	1	0	1	0	1	0	1

параметры конфигурации m и k
у фильтра Блума можно задать **вручную**
и вычислять фактическое изменение
доли ложно-положительных
срабатываний фильтра

Фильтрация вредоносных сайтов-1

URL	Характеристика
br-icloud.com.br	фишинг
www.marketingbyinternet.com	фишинг
larcadelcarnevale.com	дефейс
www.vnic.co	дефейс
www.raci.it	дефейс
retajconsultancy.com	фишинг
…	…

1. Вставить данные URL в фильтр Блума размером $\boldsymbol{m = 12}$ бит с использованием $\boldsymbol{k = 3}$ следующих хеш-функций:
   $h_1(s) = 1237 \cdot s[0]$
   $h_2(s) = 1237 \cdot s[0] + 67 \cdot s[5]$
   $h_3(s) = 555 \cdot s[0] + 13 \cdot s[1] + 7 \cdot s[5]$
2. Оценить вероятность ложно-положительного срабатывания после вставки первого и последнего URL.
3. Выполнить поиск другого *хорошего* URL.

Фильтрация вредоносных сайтов-1

справочно: значения хеш-функций

URL	Характеристика	$h_1 \bmod 12$	$h_2 \bmod 12$	$h_3 \bmod 12$
br-icloud.com.br	фишинг	2	2	0
www.marketingbyinternet.com	фишинг	11	6	3
larcadelcarnevale.com	дефейс	0	4	5
www.vnic.co	дефейс	11	1	10
www.raci.it	дефейс	11	6	3
retajconsultancy.com	фишинг	6	3	8
…	…	…	…	…

Фильтр кукушки

Устройство фильтра кукушки

ε — доля ложно-положительных срабатываний

- k хеш-функций $h_1, h_2, \ldots, h_k$, где k не зависит от ε
- функция-отпечаток [fingerprint] $f: U \to \{1, 1/\varepsilon\}$

  предположив, что $1 + 1/\varepsilon = 2^f$, мы можем зарезервировать f бит для хранения отпечатка
- хеш-таблица на m слотов, каждый из которых хранит f бит

  m также не зависит от ε — обычно определяется как $m = C \cdot n$, где n — число объектов

Устройство фильтра кукушки

ε — доля ложно-положительных срабатываний

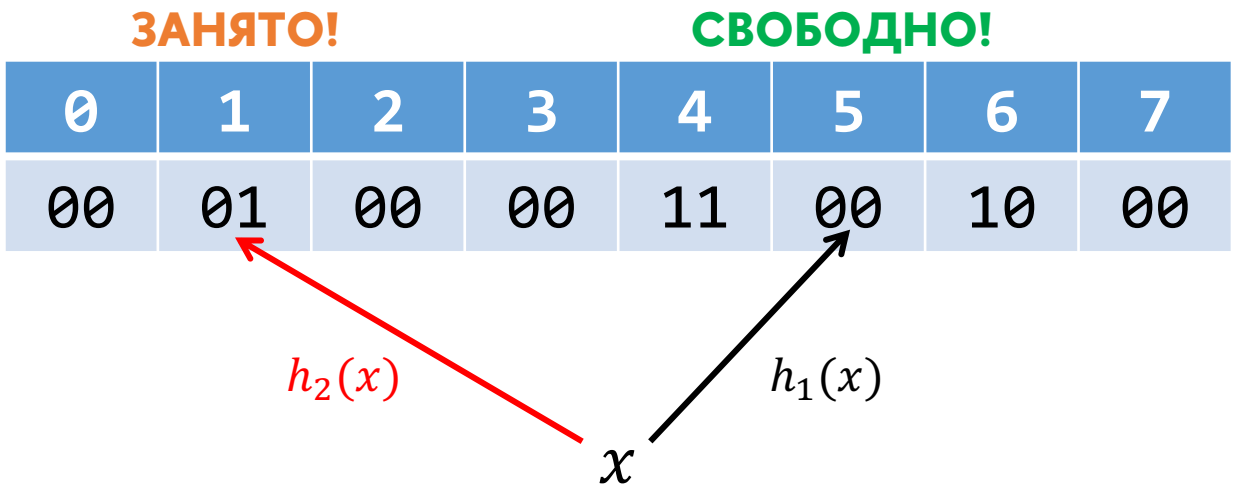
- k хеш-функций $h_1, h_2, \ldots, h_k$, где k не зависит от ε
- функция-отпечаток [fingerprint] $f: U \to \{1, 1/\varepsilon\}$

  предположив, что $1 + 1/\varepsilon = 2^f$, мы можем зарезервировать f бит для хранения отпечатка
- хеш-таблица на m слотов, каждый из которых хранит f бит

  m также не зависит от ε — обычно определяется как $m = C \cdot n$, где n — число объектов

ПРИМЕР КОНФИГУРАЦИИ

- $n = 4$ объекта и $m = 2n = 8$
- $\varepsilon = 1/3$ и $f = 2$ бита на слот
- $k = 2$ хеш-функции
- $f(x) = 2_{10} = 10_2$



Устройство фильтра кукушки

ε — доля ложно-положительных срабатываний

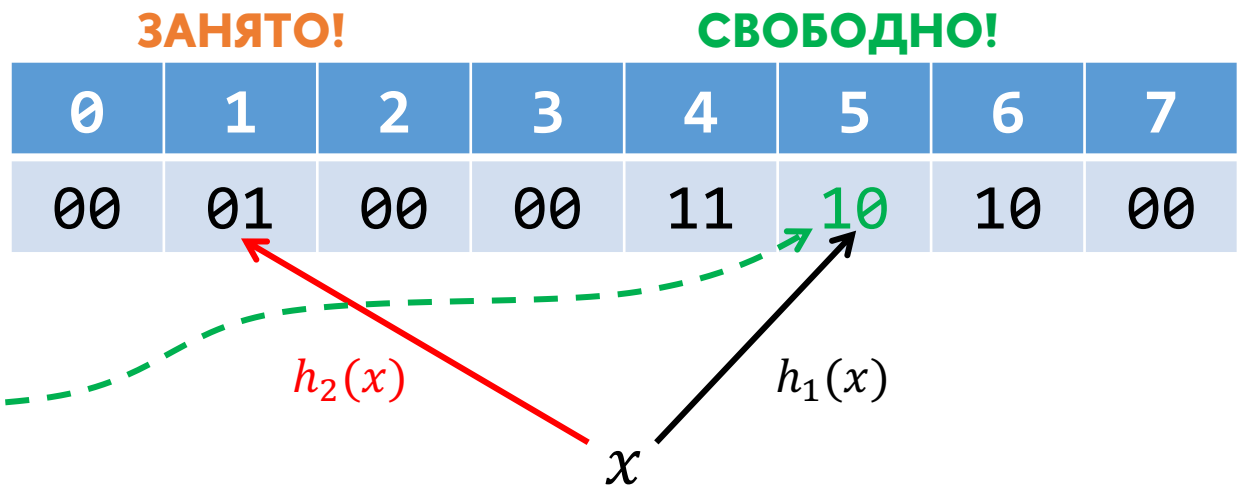
- k хеш-функций $h_1, h_2, \ldots, h_k$, где k не зависит от ε
- функция-отпечаток [fingerprint] $f: U \to \{1, 1/\varepsilon\}$

  предположив, что $1 + 1/\varepsilon = 2^f$, мы можем зарезервировать f бит для хранения отпечатка
- хеш-таблица на m слотов, каждый из которых хранит f бит

  m также не зависит от ε — обычно определяется как $m = C \cdot n$, где n — число объектов

ПРИМЕР КОНФИГУРАЦИИ

- $n = 4$ объекта и $m = 2n = 8$
- $\varepsilon = 1/3$ и $f = 2$ бита на слот
- $k = 2$ хеш-функции
- $f(x) = 2_{10} = 10_2$

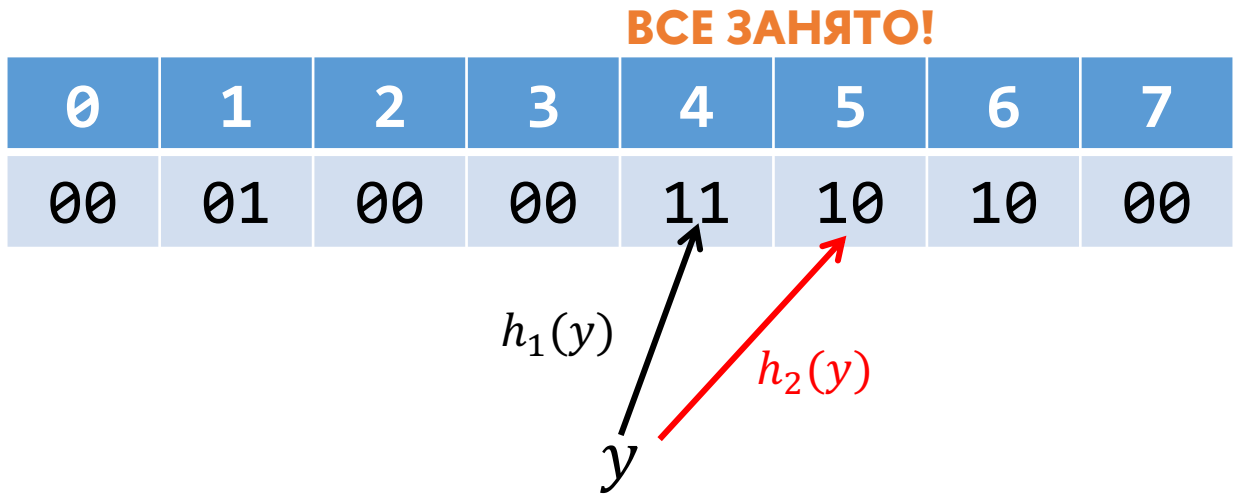


Устройство фильтра кукушки

ε — доля ложно-положительных срабатываний

ПРИМЕР КОНФИГУРАЦИИ

- $f(y) = 1_{10} = 01_2$

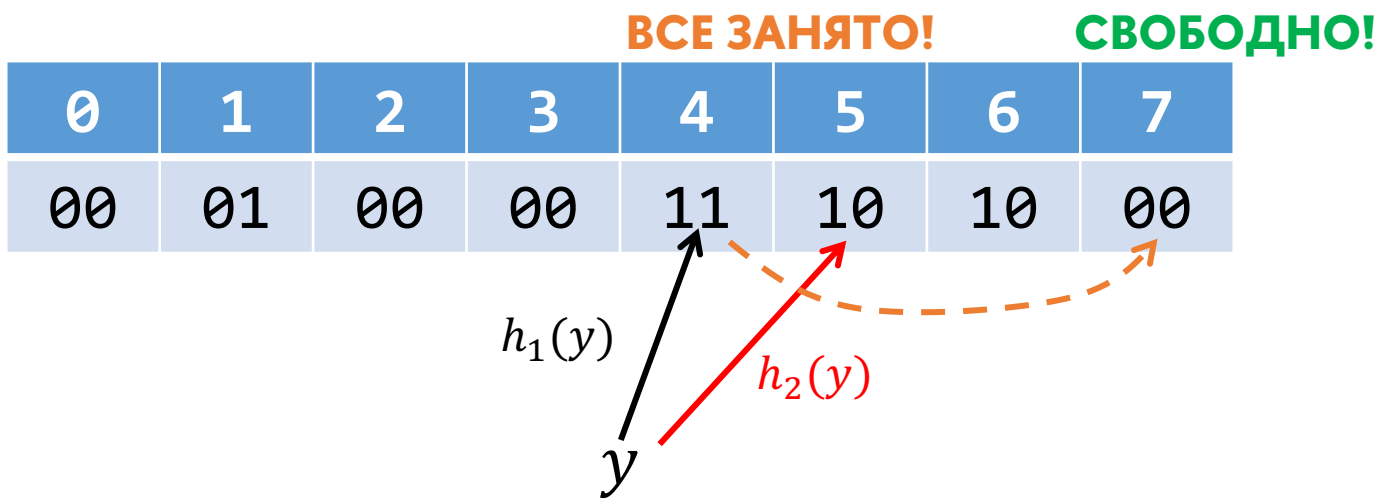


Устройство фильтра кукушки

ε — доля ложно-положительных срабатываний

ПРИМЕР КОНФИГУРАЦИИ

- $f(y) = 1_{10} = 01_2$
- вычисление нового места для отпечатка из первого слота 4:
$$4\ XOR\ h(11_2) = 7$$
- h — доп. хеш-функция для вычисления нового места:
$$h\big(f(\dots)\big): f(\dots) \to \{1, m-1\}$$
- пусть $h\big(f(y)\big) = 7$

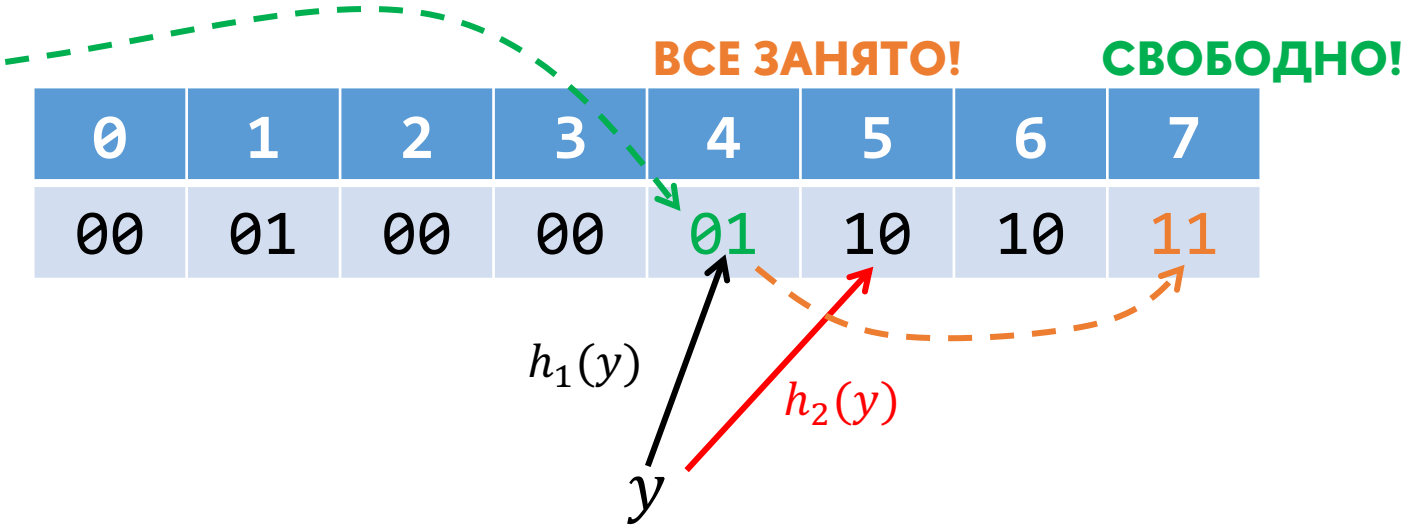


Устройство фильтра кукушки

ε — доля ложно-положительных срабатываний

ПРИМЕР КОНФИГУРАЦИИ

- $f(y) = 1_{10} = 01_2$
- вычисление нового места для отпечатка из слота 4:
  $$4\ XOR\ h(11_2) = 7$$
- h — доп. хеш-функция для вычисления нового места:
  $$h\big(f(\dots)\big): f(\dots) \to \{1, m-1\}$$
- пусть $h\big(f(y)\big) = 7$



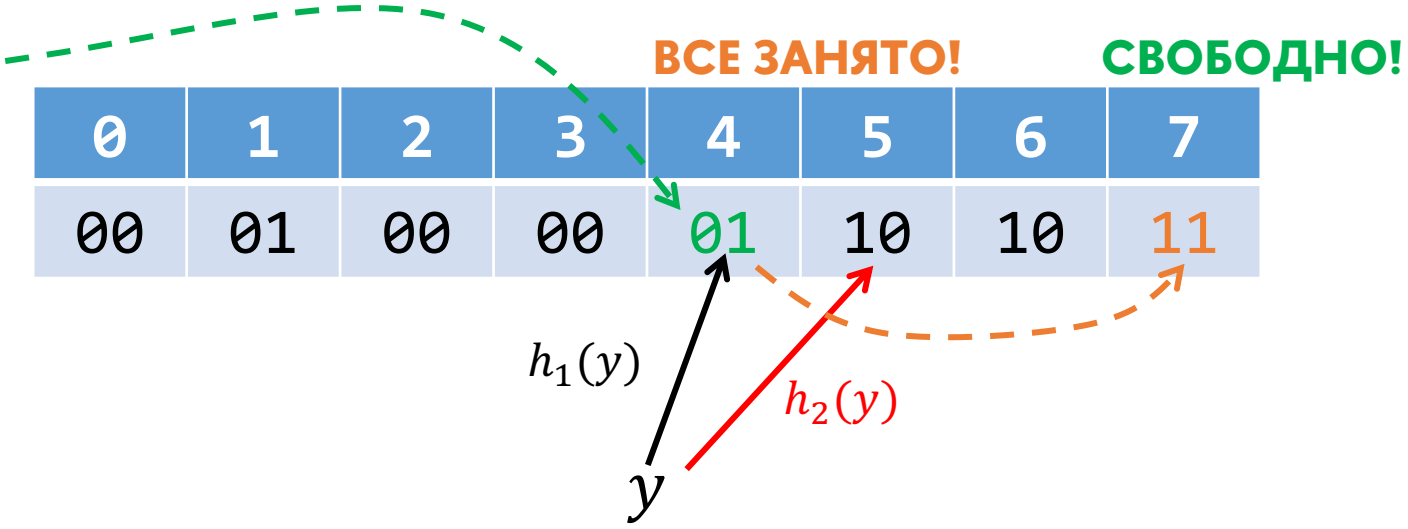
- если новое место также занято, **продолжаем по цепочке пытаться двигать** отпечатки..
- если пришлось передвинуть $O(\log n)$ отпечатков, то фильтр **нужно перестроить**!

Устройство фильтра кукушки

ε — доля ложно-положительных срабатываний

ПРИМЕР КОНФИГУРАЦИИ

- $f(y) = 1_{10} = 01_2$
- вычисление нового места для отпечатка из слота 4:
  $$4\ XOR\ h(11_2) = 7$$
- h — доп. хеш-функция для вычисления нового места:
  $h\big(f(\dots)\big)\colon f(\dots) \to \{1, m-1\}$
- пусть $h\big(f(y)\big) = 7$



такой процесс называется хешированием с **частичным ключом**, так об объекте ничего не известно, кроме его отпечатка $f(\dots)$

как и в фильтре Блума, хранение отпечатков объектов по каждой хеш-функции можно организовать в **отдельных массивах**, но это приведет к увеличению **затрат по памяти**

как проверить принадлежность объекта множеству с помощью фильтра кукушки?

0	1	2	3	4	5	6	7
00	01	00	00	01	10	10	11

Фильтрация вредоносных сайтов-2

URL	Характеристика
br-icloud.com.br	фишинг
www.marketingbyinternet.com	фишинг
larcadelcarnevale.com	дефейс
www.vnic.co	дефейс
www.raci.it	дефейс
retajconsultancy.com	фишинг
…	…

1. Вставить данные URL в фильтр кукушки размером **8** слотов на **3** бита, используя данные об отпечатках и потенциальных индексах, которые даны на следующем слайде.
2. Что необходимо учитывать при оценке вероятности **ложно-положительного ответа** фильтра кукушки?

Фильтрация вредоносных сайтов-2

URL	Характеристика	Отпечаток $f(URL)$	Потенциальные места		$h(f(URL))$
			$h_1(URL)$	$h_2(URL)$	
br-icloud.com.br	фишинг	001	1	4	6
www.marketingbyinternet.com	фишинг	000	4	7	5
larcadelcarnevale.com	дефейс	010	3	0	1
www.vnic.co	дефейс	010	7	5	2
www.raci.it	дефейс	100	1	0	3
retajconsultancy.com	фишинг	101	1	2	1
…	…	…	…	…	…

фильтр Блума VS фильтр кукушки

Фильтр Блума	Фильтр кукушки
вставка и проверка принадлежности объектов требует вычисления значений k **различных** хеш-функций	на практике используется особая схему с **двумя хеш-функциями**
время вставки остается неизменным вне зависимости от заполненности битового вектора(-ов)	с ростом заполненности хеш-таблицы хеширование кукушки потребует больше времени для поиска свободной ячейки – потребуется **перехеширование**
с ростом заполненности битового вектора(-ов) фильтра Блума значительно возрастает вероятность ложно-положительного срабатывания	целевой порог вероятности ложно-положительного ответа может остаться неизменным до заполнения на **95.5%** (практические данные)
не поддерживается **удаление** объектов, так как иначе возможны **ложно-отрицательные ответы**	объекты, о которых точно известно, что они были добавлены в фильтр, **могут быть удалены**

Список с пропусками

skip-list

Список с пропусками

Уильям Пью, 1989 г.
Статья "**Skip Lists: Probabilistic Alternative to Balanced Trees**"
в журнале *Communications of the ACM*.

- Обобщение отсортированных списков
- Ожидаемое время поиска - **O(log n)**
- *Вероятностная* структура данных

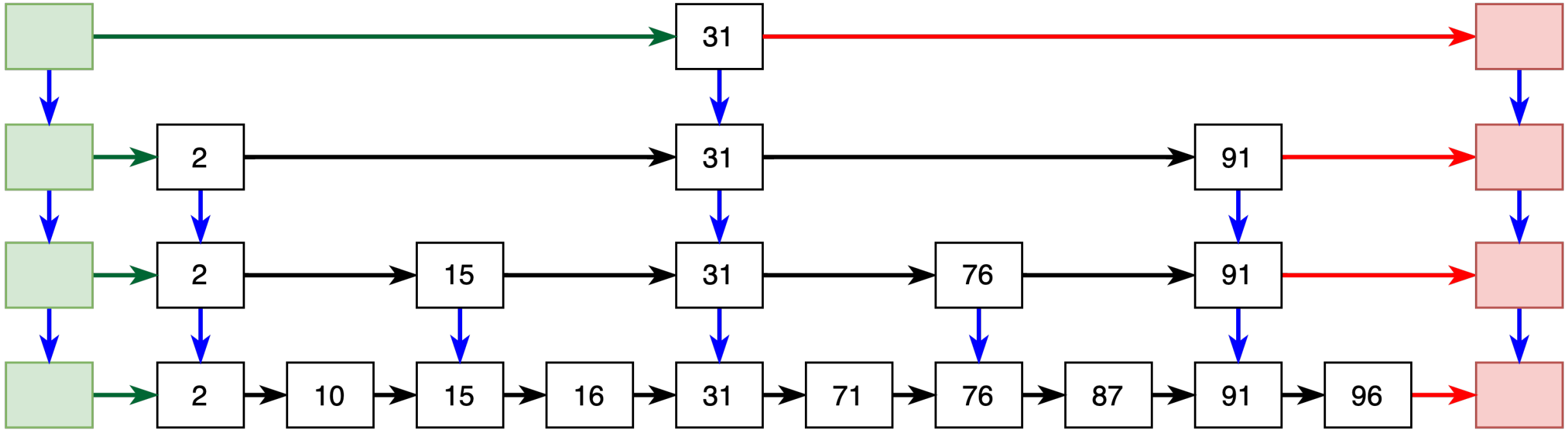
Идеальный список с пропусками

Ключи хранятся в отсортированном виде.

Содержит **O(log n)** уровней, каждый из которых
также является списком.

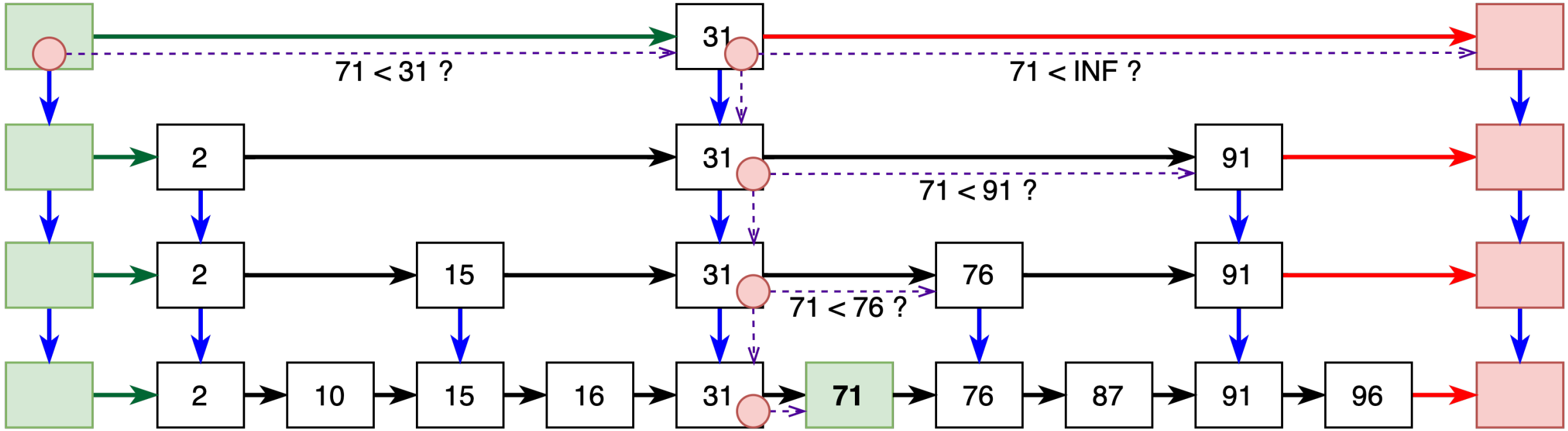
Все ключи хранятся только на последнем уровне, а каждый вышестоящий уровень содержит *половину* ключей, которые хранятся уровнем ниже.

Идеальный список с пропусками - Поиск



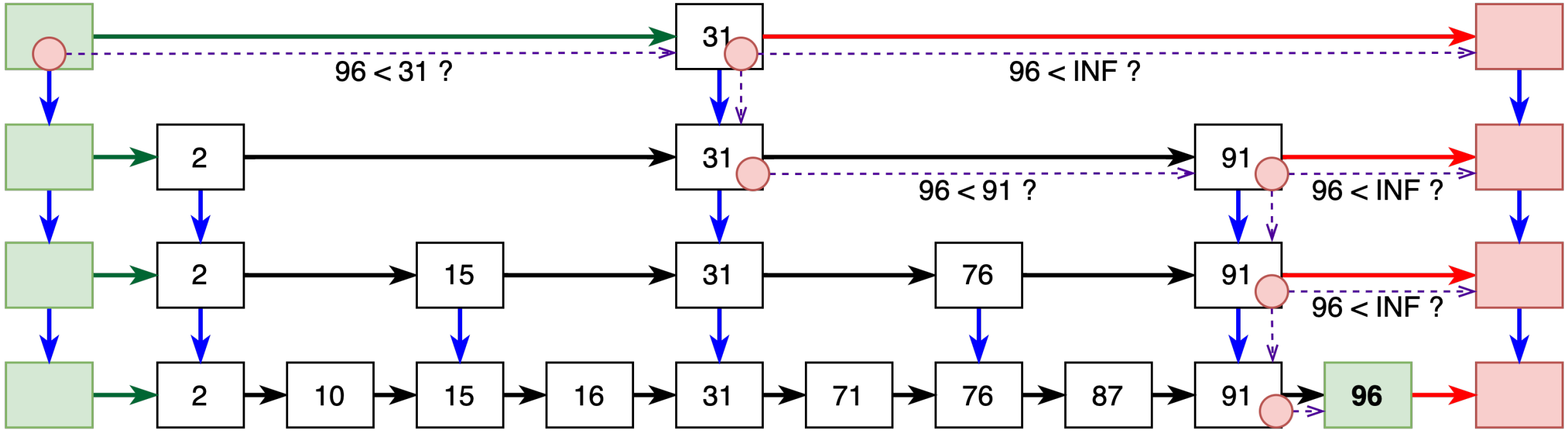
Идеальный список с пропусками - Поиск

`search(71)`



Идеальный список с пропусками - Поиск

`search(96)`



Идеальный список с пропусками - Поиск

```
Node *cur = head;

for (int i = MAX_LEVEL; i > 0; --i) {
    while (!cur->next[i] &&
            cur->next[i]->data < key)
{
        cur = cur->next[i];
    }

cur = cur->next[0];

if (cur->data = key) {
    return cur;
} else {
    return nullptr;
}
```

```
class Node {
    T data;
    Node **next;
    ...
}

class SkipList {
    Node *head;
    ...
}
```

Идеальный список с пропусками

проблемы со вставкой и удалением

Из-за фиксированных требований в результате вставки и удаления потребуется перестройка всего списка.

Решение: на каждом уровне *ожидается* половина ключей нижележащего уровня – используем рандомизацию

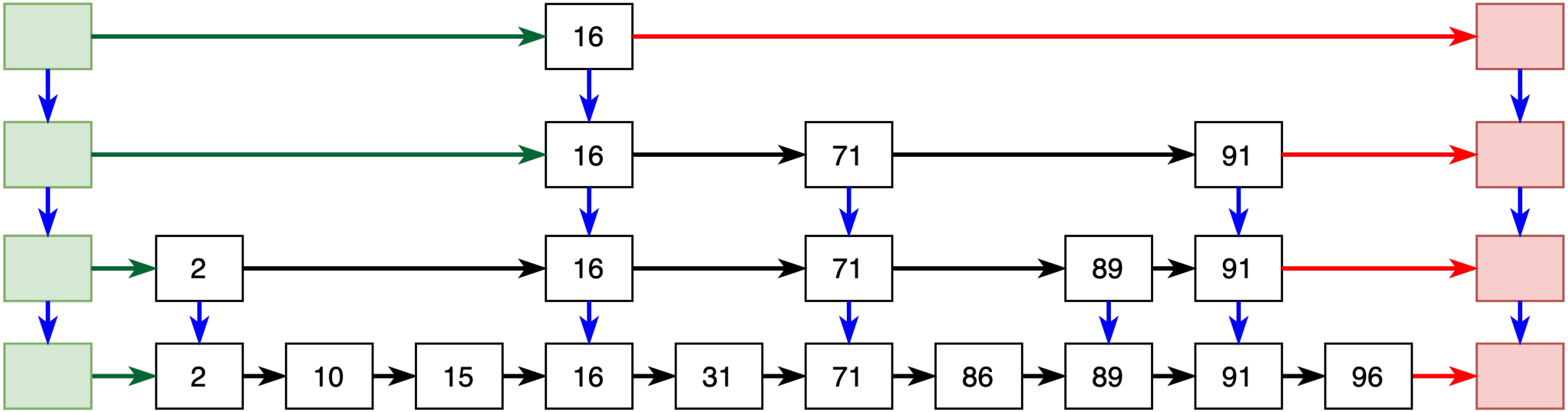
Случайный список с пропусками

Допускается не идеальная сбалансированность

Вероятность того, что ключ будет находиться на следующем (по высоте) уровне составляет ок. **0,5**:

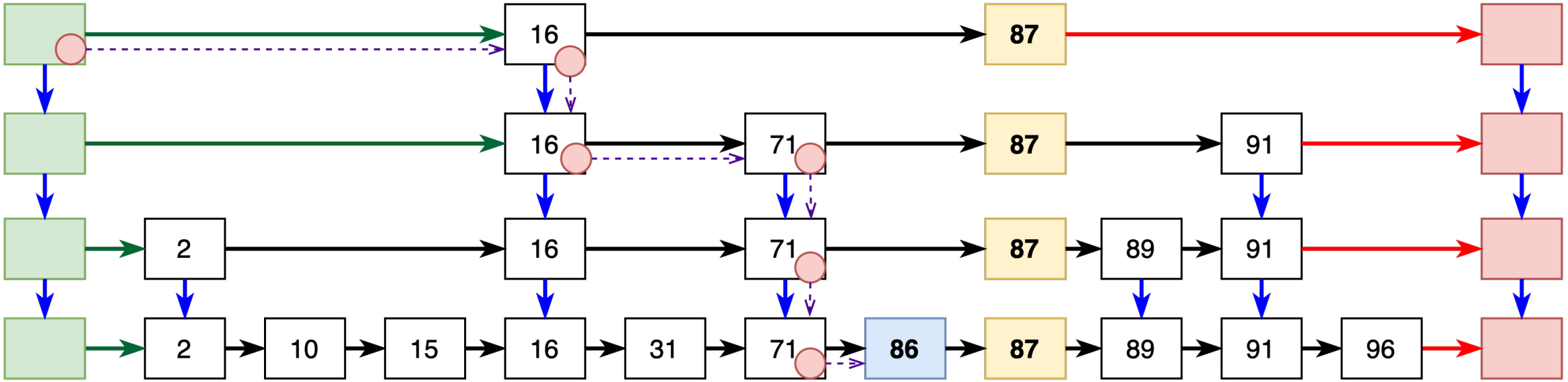
- *Ожидается* половина всех ключей на уровне 1
- *Ожидается* четверть всех ключей на уровне 2
- ...

Случайный список с пропусками



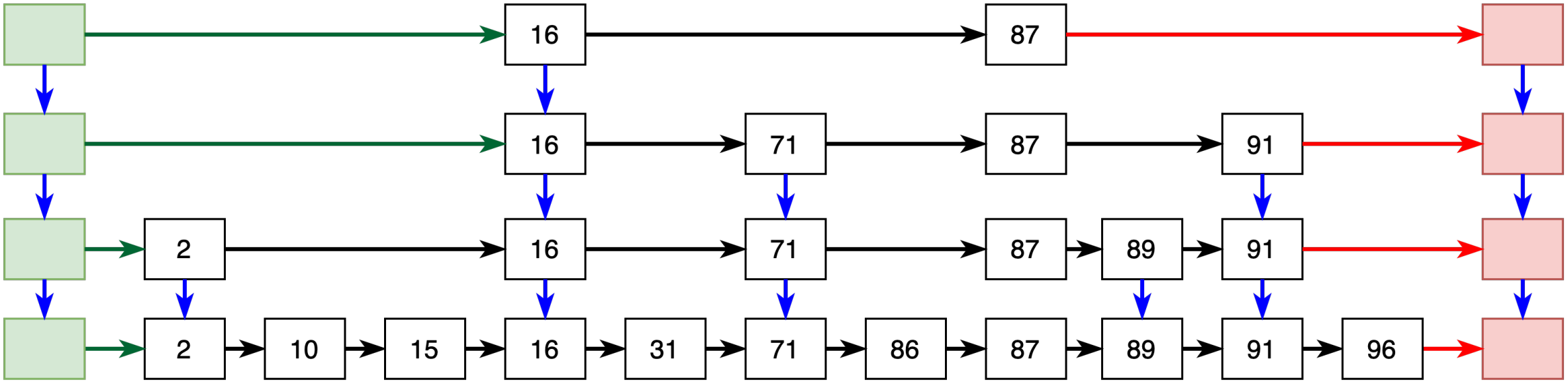
Случайный список с пропусками

`insert(87)` и 3 раза выпал орел



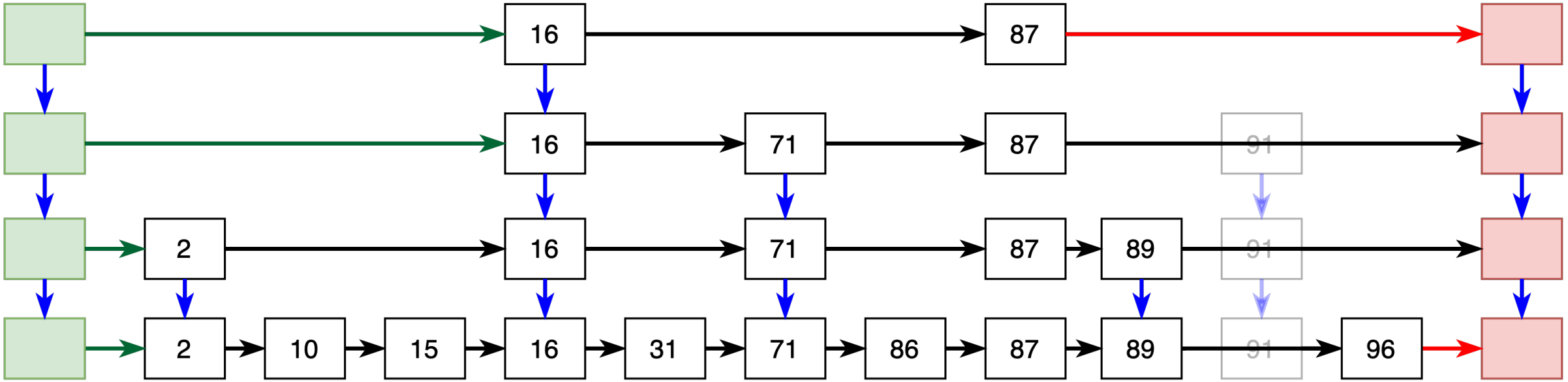
Случайный список с пропусками

`delete(91)` со всех уровней



Случайный список с пропусками

`delete(91)` со всех уровней



Случайный список с пропусками

Ожидаемая сложность выполнения основных операций совпадает с идеальным списком с пропусками

Вероятность *вырожденных* ситуация крайне мала:

1. Список с пропусками становится простым связным списком (всегда выпадают решки)
2. Каждый узел списка с пропусками будет находиться на каждом уровне (всегда выпадают орлы)

Упражнение на работу со `SkipList`

12 17 20 25 31 38 39 44 50 55

1. Создайте список с пропусками из приведенной выше последовательности ключей, используя результаты некоторого «физического эксперимента».
   Максимальное количество уровней – 5.
2. Выполните поиск каких-либо элементов в полученном списке. Сколько сравнений было выполнено?
3. Выполните удаление каких-либо элементов в полученном списке. Как может измениться сложность поиска?

Домашнее задание

опционально

К следующему семинару!

На выбор или вместе...

1. Реализуйте класс **`SkipList`** для хранения целочисленных данных
2. Сравните количество операций, которые потребуются для поиска в **`SkipList`** с **каким-либо сбалансированным деревом** на ваш выбор.