Лекция 5

Хеш-таблицы. Поддерживаемые операции. Коллизии. Хеш-функции. Фильтры Блума.

Хеш-таблица – это структура данных, позволяющая быстро получать информацию по ключу вне зависимости от имеющихся данных

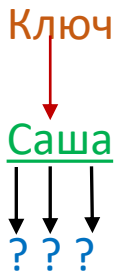
Концепция Х-Т очень проста. Х-Т присутствуют в любом языке программирования. Х-Т используются

- при построении кэша;
- индексов баз данных;
- в языковых процессорах;
- в ассоциативных массивах

Х-Т можно рассматривать как массив. Массивы обеспечивают немедленный случайный доступ к ячейке за O(1)

Пример: телефонная книга (Имя-номер телефона)

Мы хотим организовать поиск по ключу: по имени человека (ключу) найти его номер телефона



Имя	Номер
Петя	32-32-32
Вася	44-22-11
Маша	11-11-11
Саша	22-33-88
Коля	44-44-44
Кися	88-88-88

1) Можно действовать простым перебором. 4-ая попытка оказалась удачной. Очень медленно! (O(n))
2) Если знать индекс ind искомой записи (индекс), то ее можно найти практически мгновенно. ind=3 O(1)

Но где взять эти индексы?

ind	Имя	Номер
0	Петя	32-32-32
1	Вася	44-22-11
2	Маша	11-11-11
3	Саша	22-33-88
4	Коля	44-44-44
5	Кися	88-88-88

Заполним таблицу так, чтобы индекс каждого элемента вычислялся через его значение

ind	Имя	Номер
0		
1		
2		
3		
4		
5		

Пусть есть пустая таблица и элемент, который мы хотим в нее сохранить.

ind ??
Саша 22-33-88

Вычислим ind, исходя из имени (ключа). Тогда в будущем мы по ключу также сможем вычислить ind и сразу получить значение (номер телефона). Как вычислить ind?
Например: можно использовать кодировку Windows-1251:

С- 209+
а - 224+
ш- 248+
а- 224

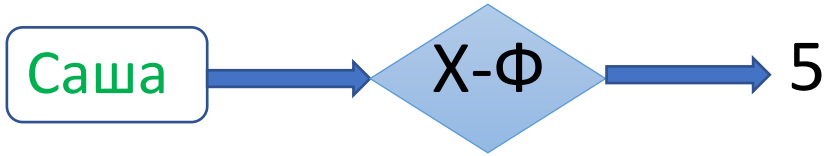
905 ???

905 mod 6 =5

ind	Имя	Номер
0		
1		
2		
3		
4		
5	Саша	22-33-88

Хеш-функция

Хеш-функция преобразует ключ в индекс.



Строковое значение преобразовано в число.

Если бы ключом являлся номер телефона, то можно было бы просто сложить все цифры номера и взять остаток по модулю.

Существует множество хеш-функций (хороших и не очень). Разработка хороших хеш-функций является нетривиальной задачей.

Размер хеш-таблицы

- В примере с телефонной книгой отведем 15 позиций для букв имени. Всего 33 буквы. Получим $n = 33^{15}$ - столько строк в таблице.
- Невозможно и не имеет смысла выделять такую память. Размер Х-Т значительно меньше. При хорошей организации, хорошей хеш-функции, непатологических данных имеем следующее время на осуществление основных поддерживаемых операций

Операции	Типичное время выполнения
1. Просмотреть (lookup): по ключу *k* вернуть указатель на объект в Х-Т с ключом *k* (либо сообщить, что такого объекта не существует)	O(1)*
2. **Вставить**: с учетом нового объекта *x* добавить в Х-Т	O(1)
3. **Удалить**: по ключу удалить объект с ключом k из Х-Т, если он существует	O(1)*

Применение хеш-таблиц

1) Устранение дублирования.

Пример: вы наблюдаете за посетителями вашего веб-сайта. Отслеживается число несовпадающих IP-адресов, которые обращались к вашему веб-сайту, в дополнение к суммарному числу посещений.

При поступлении нового объекта *x* (IP-адреса) с ключом *k*:

- Применить операцию «**Просмотреть**»
- Если нет, то применить операцию «**Вставить**», для того, чтобы поместить *x* в X-T

2) Задача о сумме двух чисел.

Вход: неотсортированный массив A из n целых чисел и целевое целое t.

Цель: определить, существует ли 2 числа $x, y \epsilon A$: $x + y = t$

Задача о сумме двух чисел (варианты решения)

- Попытка №1: перебрать все пары x и y и сравнить их сумму с t. Сложность $O(n^2)$.
- Попытка №2: $(\forall x\ \exists!\ y = t - x)$

Вход: массив A из n целых чисел и целевое целое число t.

Выход: «да», если $A[i] + A[j] = t$ для некоторых $i, j \in \{1, 2, 3, ..., n\}$, «нет» — в противном случае.

```
for i = 1 to n do
    y := t − A[i]
    if A содержит y then      // линейный поиск
        return «да»
return «нет»
```

Сложность $O(n^2)$.

Задача о сумме двух чисел - продолжение

- Попытка №3: отсортируем массив

СУММА ДВУХ ЧИСЕЛ (РЕШЕНИЕ НА ОСНОВЕ ОТСОРТИРОВАННОГО МАССИВА)

Вход: массив A из n целых чисел и целевое целое число t.

Выход: «да», если $A[i] + A[j] = t$ для некоторых $i, j \in \{1, 2, 3, ..., n\}$, «нет» — в противном случае.

```
sort A     // используя подпрограмму сортировки
for i = 1 to n do
   y := t − A[i]
   if A содержит y then     // двоичный поиск
      return «да»
return «нет»
```

Сложность $O(n \log n + \log n) = O(n \log n)$.

- Попытка №4: решение на основе Х-Т

СУММА ДВУХ ЧИСЕЛ (РЕШЕНИЕ НА ОСНОВЕ ХЕШ-ТАБЛИЦЫ)

Вход: массив A из n целых чисел и целевое целое число t.

Выход: «да», если $A[i] + A[j] = t$ для некоторых $i, j \in \{1, 2, 3, ..., n\}$, «нет» — в противном случае.

```
H := пустая хеш-таблица
for i = 1 to n do
   Вставить A[i] в H
for i = 1 to n do
   y := t − A[i]
   if H содержит y then     // используя операцию «Просмотреть»
      return «да»
return «нет»
```

Сложность $O(n)$.

3) Применение – поиск в огромном пространстве состояний

- Компьютерные игры, задачи планирования. Например, шахматная программа, разведывающая последствия различных ходов. Имеем огромный орграф: вершины – позиции, дуги – ходы. Размер графа $\backsim 10^{100}$.

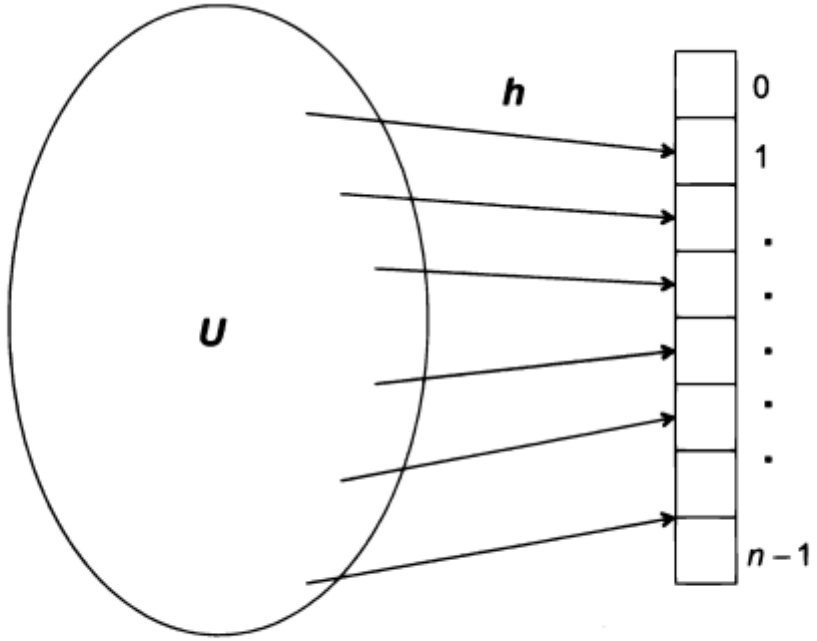
Можно выполнить поиск в ширину, начиная с текущей позиции, и разведать краткосрочные последствия разных ходов до достижения ограничения по времени. С помощью Х-Т отслеживаются посещенные вершины. Если вершина посещена (внесена в Х-Т), алгоритм ее пропускает и возвращается назад. В противном случае он вставляет ее в Х-Т.

Реализация хеш-таблиц

- Хеш-таблица хранит множество S ассоциированных ключей, взятых из универсума U. Если U невелико, то решение на основе массива является хорошим. Операции «Просмотреть», «Вставить», «Удалить» выполняются за постоянное время. Но обычно U очень велико. Реально рассматривать структуры данных, которые требуют пространства пропорционально $|S|$.
- Можно хранить объекты в связном списке. Тогда пространство пропорционально $|S|$. Но при этом время на «Просмотреть» и «Удалить» масштабируется с $|S|$, что хуже, чем постоянное время.

Структура данных	**Пространство**	**Типичное время выполнения операции** Просмотреть
Массив	$\Theta\|U\|$	$\Theta(1)$
Связный список	$\Theta\|S\|$	$\Theta\|S\|$
Хеш-таблица	$\Theta\|S\|$	$\Theta(1)$*

Как действует хеш-функция (*h*)



Хеш-функция отображает каждый возможный ключ из универсума *U* в позицию в {0, 1, 2, ..., *n* – 1}. При $|U| > n$ два разных ключа будут отображаться в одинаковую позицию

Коллизии

Продолжение примера с телефонной книгой

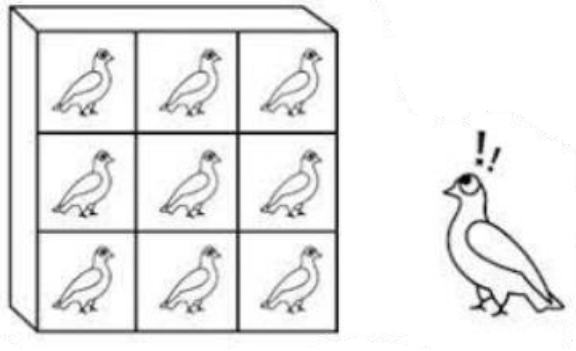
Петя 207+229+242+255=933	933 mod 6=3
Вася 194+224+241+255=914	914 mod 6=2
Маша 204+224+248+224=900	900 mod 6=0
Коля 202+238+235+255=930	930 mod 6=0
Кися 202+232+241+255=930	930 mod 6=0

??

ind	Имя	Номер
0	Маша	11-11-11
1		
2	Вася	44-22-11
3	Петя	32-32-32
4		
5	Саша	22-33-88

Две разные записи попадают в одну ячейку. Имеем коллизию.

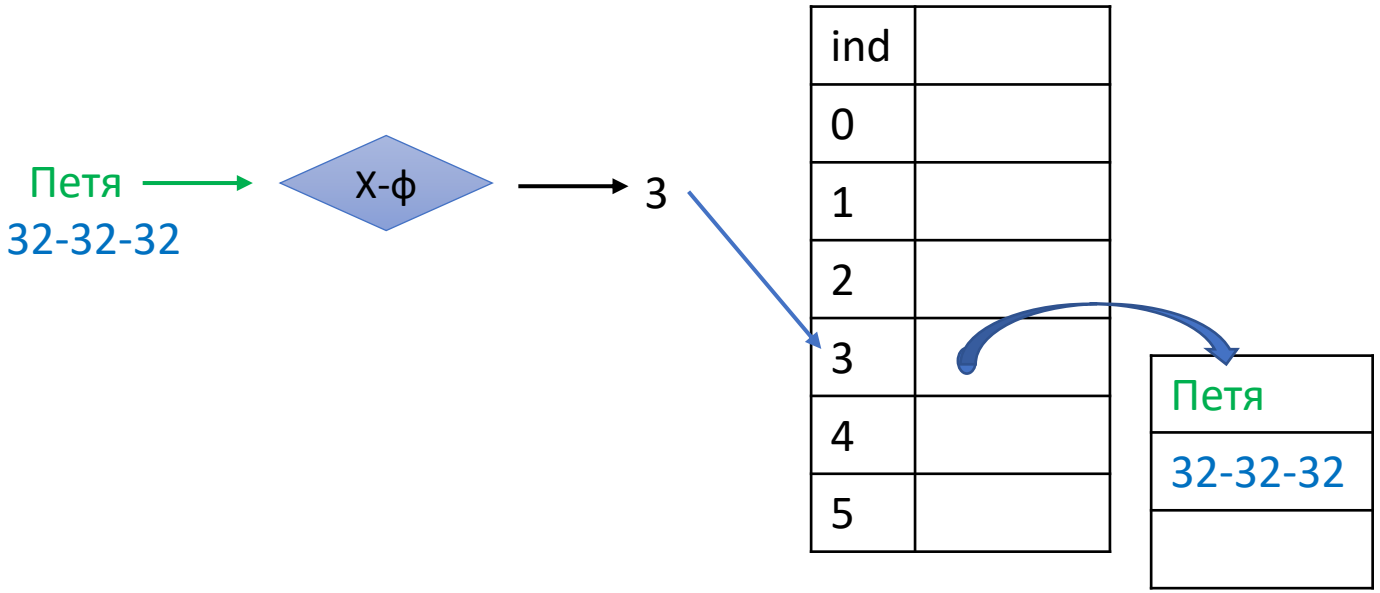
Т.к. обычно $|U| \gg n$, то коллизии неизбежны. Принцип Дирихле.



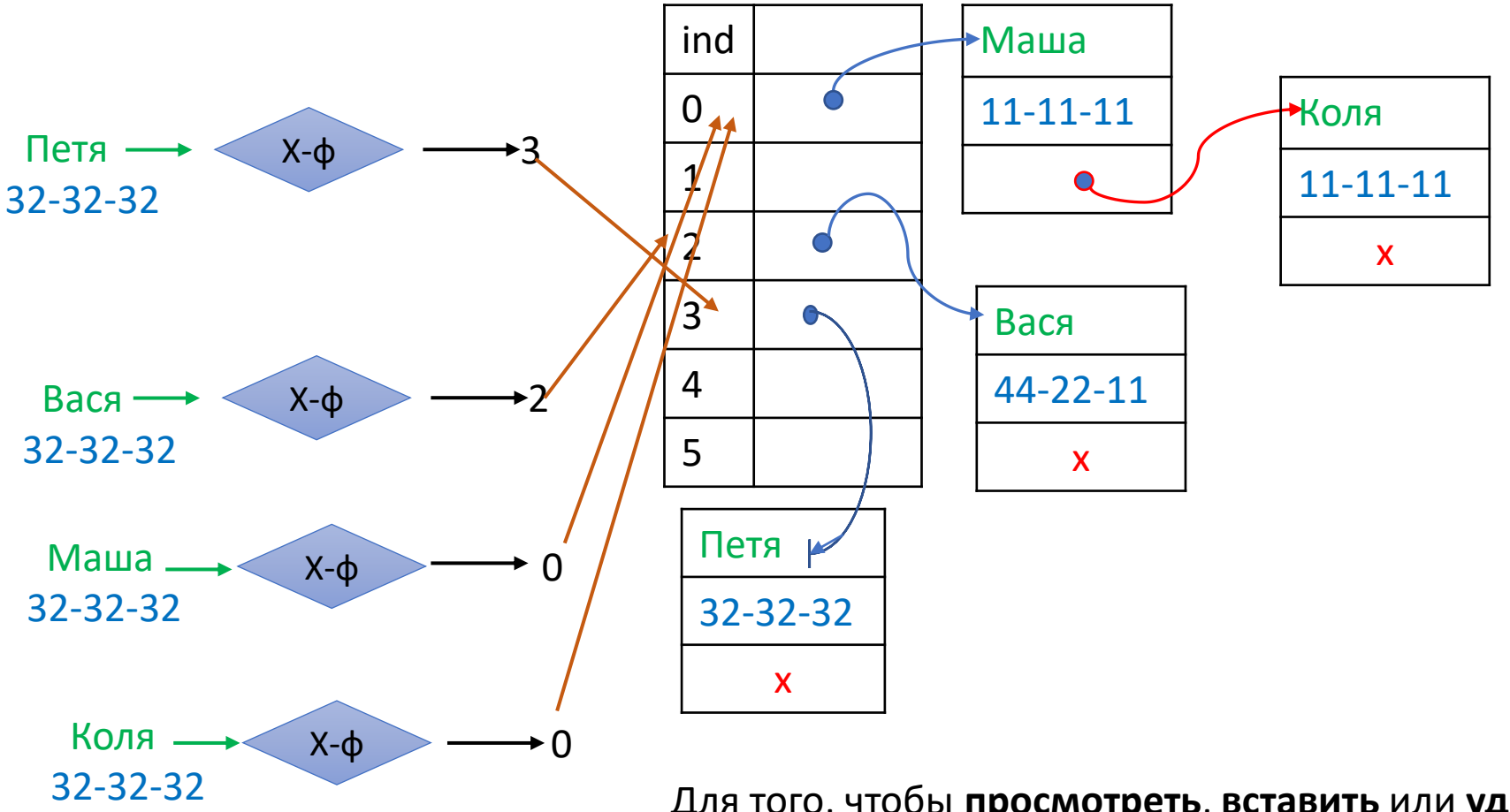
Разрешение коллизий

Коллизия: два различных ключа k_1 и k_2 из U конфликтуют в рамках хеш-функции h, если $h(k_1) = h(k_2)$

1) Метод цепочек (или раздельное сцепление). В Х-Т хранятся не значения, а ссылки на связный список, в котором хранятся значения. В случае коллизии мы ищем значения в этом списке, переходя от одной записи к другой.



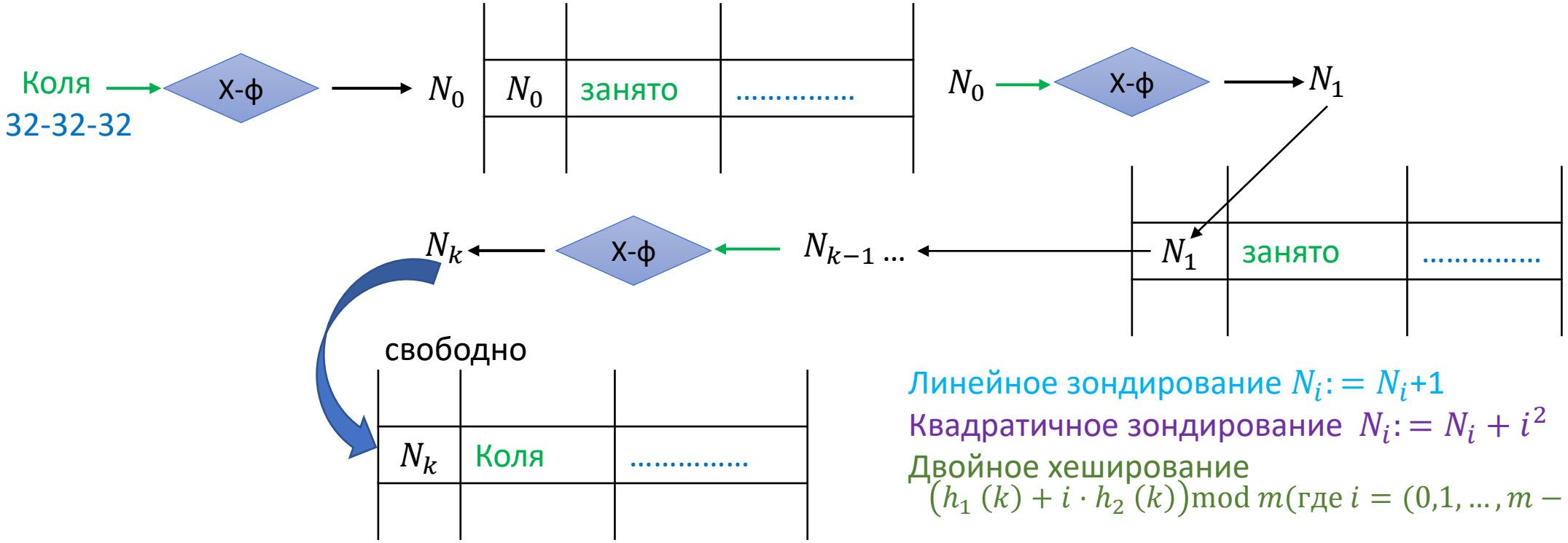
Метод цепочек (продолжение)



Для того, чтобы **просмотреть**, **вставить** или **удалить** объект с ключом k, необходимо выполнить эти операции в связном списке из $A[h[k]]$.

2) Открытая адресация

- При попытке вставить объект в занятую позицию Х-Т, каким-то образом ищется следующая позиция (например, применяется хеш-функция к найденному индексу). Если там пусто, то объект вставляется, иначе, ищется следующая позиция. Получаем зондажную последовательность (процесс пробирования).



Линейное зондирование $N_i := N_i + 1$

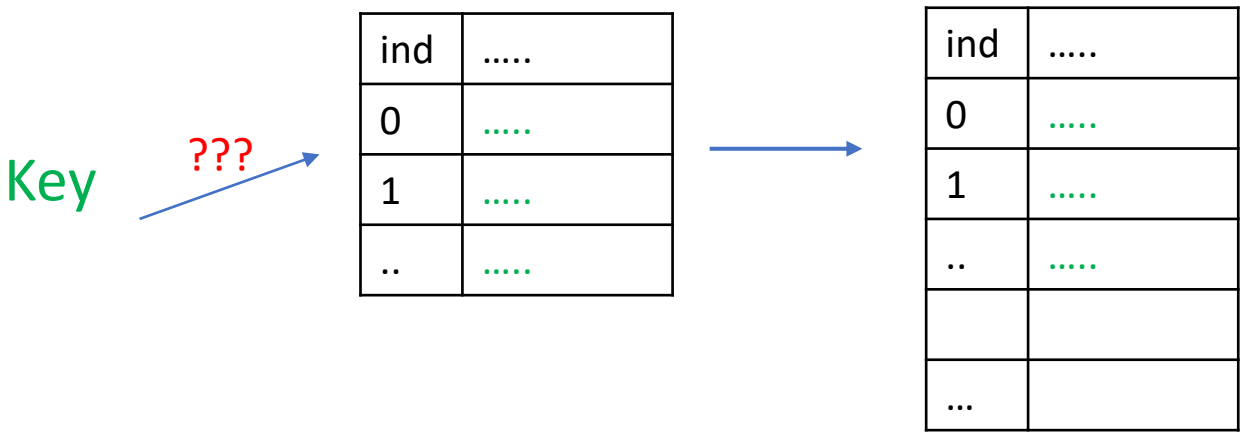
Квадратичное зондирование $N_i := N_i + i^2$

Двойное хеширование

$(h_1(k) + i \cdot h_2(k)) \bmod m$ (где $i = (0,1,\dots,m-1)$

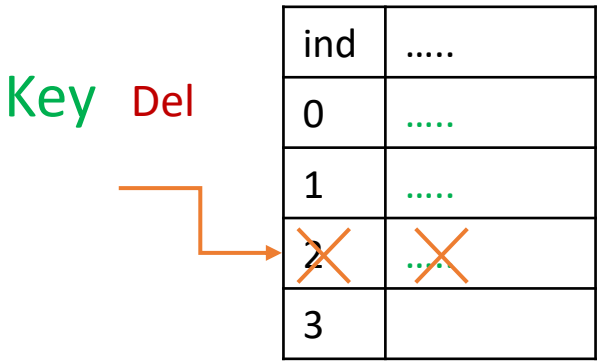
Проблемы открытой адресации и их разрешение

- При попытке вставить выясняется, что вся Х-Т заполнена. Можно создать новую таблицу (большего размера) и скопировать все значения в нее.



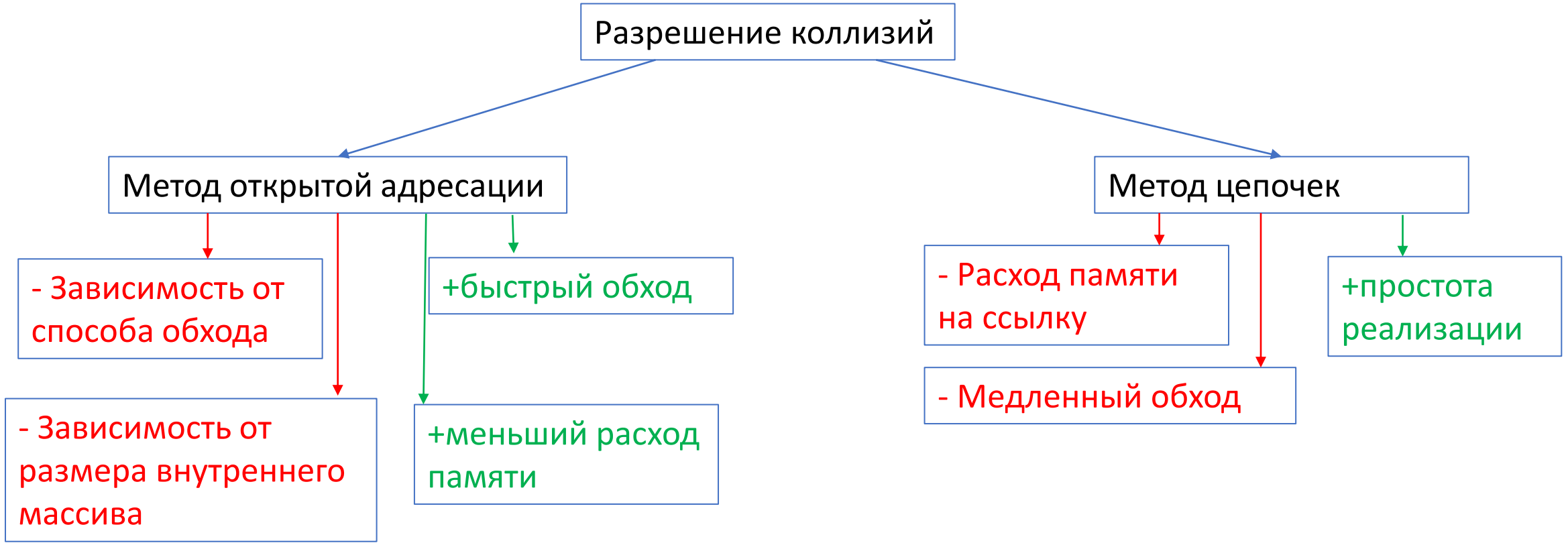
Рехеширование

- Если нужно удалить значение, то можно помечать ячейки Х-Т как *удаленные*



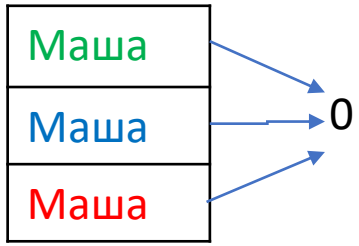
При поиске объектов удаленные ячейки будем пропускать, при вставке объектов использовать для записи. Если накопилось много *удаленных* ячеек, поиск (из-за пропуска *удаленных)* сильно замедляется. Тогда собирают оставшиеся заполненные ячейки и переносят в новую Х-Т.

Способы разрешения коллизий (плюсы и минусы)

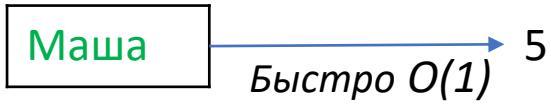


Хорошая хеш-функция

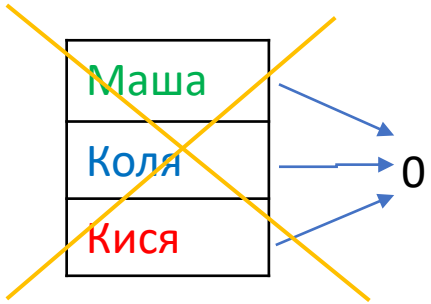
Детерминизм



Эффективность



Равномерность



Ограниченность

0<N<размер таблицы

Загрузка X-T

$$\text{Загрузка X} - \text{T} = \frac{\text{число хранимых объектов}}{\text{длина массива}} \qquad (\alpha = \frac{|S|}{n})$$

При открытой адресации часть $1 - \alpha$ – пустая. $1 - \alpha$ – шанс найти пустую ячейку в идеальном случае.

Мат.ожидание (среднее число зондирований) $\frac{1}{1-\alpha}$ (геометрическое распред.)

Стратегия разрешения коллизий	Идеализированное время выполнения операций
Метод цепочек	$O(\lceil\alpha\rceil)$
Двойное хеширование	$O(1/_{1-\alpha})$
Линейное зондирование	$O(1/_{(1-\alpha)^2})$

Размер X-T должен периодически изменяться с тем, чтобы ее загрузка не превышала 70%

Фильтры Блума

- Используются для тех же целей, что и хеш-таблицы.
- Операции «Просмотреть» и «Вставить», размер памяти критичен, а редкие ошибки не играют роли. Работают быстро, но могут иногда ошибаться (ключ не был вставлен, а фильтр Блума отвечает, что был).

Операции	Время выполнения
1. Просмотреть (lookup): по ключу *k* вернуть указатель на объект в фильтре Блума с ключом *k* (либо сообщить, что такого объекта не существует)	$O(1)^{\dagger}$ († -управляемая, но ненулевая вероятность ложных утверждений)
2. **Вставить**: с учетом нового объекта *x* добавить в Х-Т	O(1)

Применения фильтров Блума

- Спеллчекеры (проверка орфографии). Каждое слово вставляется в фильтр, помечаются слова, которых нет.
- Запрещенные слова (проверка допустимых паролей).
- Интернет-маршрутизаторы (пакеты данных быстро проходят маршрут с потоковой скоростью.

Реализация фильтров Блума

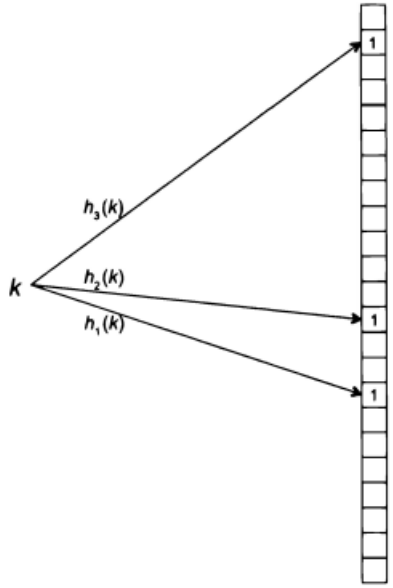
- *n*-битовая строка (первоначально, из нулей)

ФИЛЬТР БЛУМА: ВСТАВИТЬ (ПО КЛЮЧУ *k*)

for $i = 1$ to m **do**

$A[h_i(k)] := 1$

Например, если $m = 3$ и $h_1(k) = 23$, $h_2(k) = 17$ и $h_3(k) = 5$, вставка k приводит к тому, что 5-й, 17-й и 23-й биты массива устанавливаются равными 1



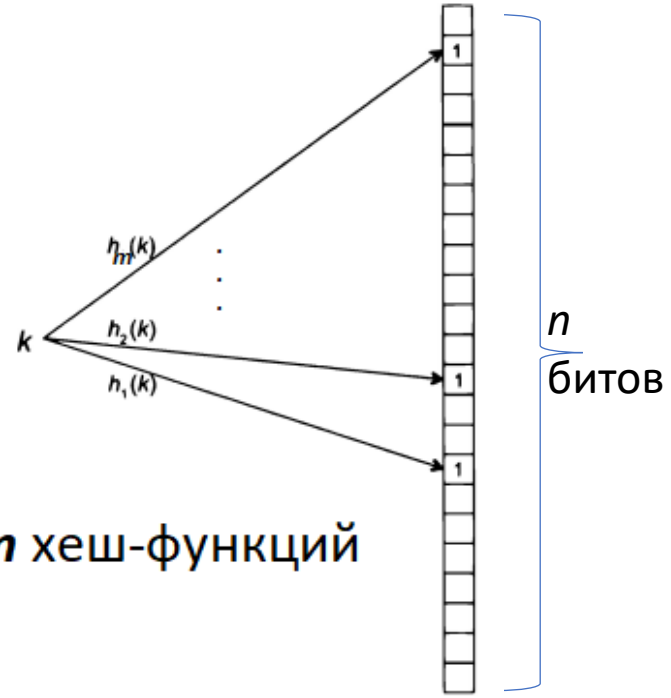
ФИЛЬТР БЛУМА: ПРОСМОТРЕТЬ (ПО КЛЮЧУ *k*)

for $i = 1$ to m **do**

if $A[h_i(k)] = 0$ **then**

return «нет»

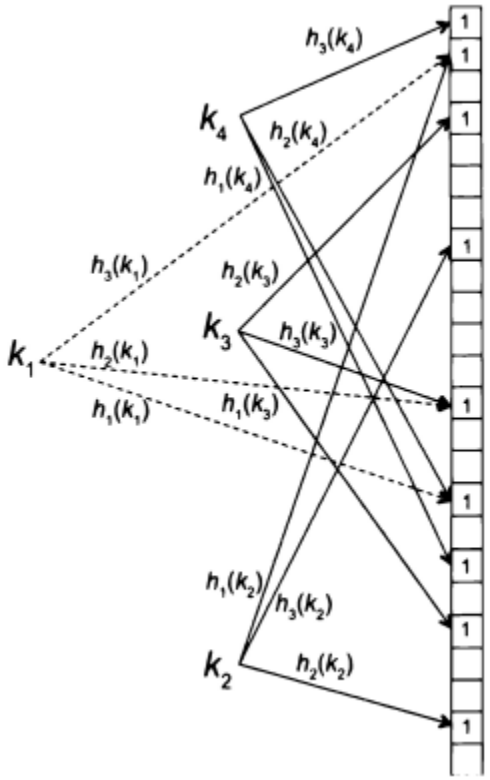
return «да»



m хеш-функций

Пример Фильтра Блума

4 ключа k_1, k_2, k_3, k_4. 3 хеш-функции.



Ключ	Значение h_1	Значение h_2	Значение h_3
k_1	23	17	5
k_2	5	48	12
k_3	37	8	17
k_4	32	23	2

9 бит было установлено в 1. (Операция «Вставить»)
Операция «Просмотреть» ответит «Да» на ключ k_1, даже если он не был вставлен.
Т.е. ложное утверждение «Да» возможно, но ложное отрицание – нет!
С помощью эвристического анализа можно свести к минимуму вероятность ложных утверждений.
Если $b = \frac{n}{|S|}$ битов на ключ, то при $m = \ln 2 \cdot b$
достигается минимум вероятности=$\left(\frac{1}{2}\right)^{\ln 2 \cdot b}$