Хеш-таблица

education

Сравнение с уже известными структурами данных

- Структура данных, позволяющая хранить все элементы в виде пары «ключ-значение».
- Сложность O(1) на операции вставки/удаления/выборки.

	Хеш-таблица	Массив	Список
Вставка	O(1)	O(n)	O(1)
Удаление	O(1)	O(n)	O(1)
Выборка	O(1)	O(1)	O(n)

education

Основные операции и применение

- insert(k, v)
- get(k)
- delete(k)
- contains(k) — для множества

```
insert(k, v) {
        T[k] = v
}

get(k) {
        return T[k]
}
```

Когда может понадобиться хеш-таблица

- Если мы хотим хранить большие ключи — например, k будут иметь диапазон от 0 до 10 в 9-й степени, при этом не весь этот диапазон будет занят.
- Нам бы в этом случае пригодилась функция, которая приводила бы наш большой диапазон к маленькому диапазону.
- Например, мы можем брать остаток от деления h = k%M, где M — это некоторая константа.
- Тогда h(k), где k может быть сколь угодно большим, приводит k-принадлежащие диапазону 0 до L – 1, где L — большое число, к диапазону 0...n – 1, где n значительно меньше L.
- Появляется вероятность возникновения индексов в таблицах i1 и i2, имеющих равные значения при разных ключах.

```
insert(k, v) {
        T[h(k)] = v
}

get(k) {
        return T(h(k))
}

delete(k) {
        T(h(k)) = null
}
```

Когда может понадобиться хеш-таблица

- Можно хранить в списке, но нужен более быстрый доступ.
- Когда нет возможности получать по целочисленному индексу — например, речь идёт о строках или об объектах.

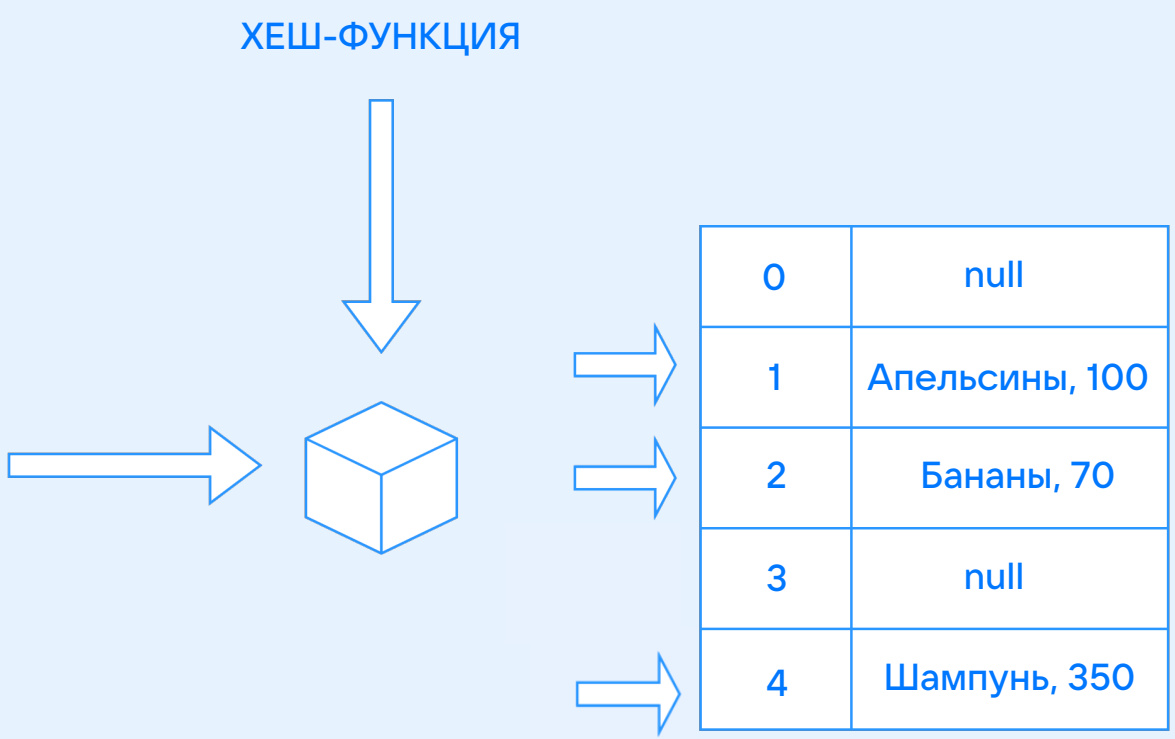
```
insert(k, v) {
        T[h(k)] = v
}

get(k) {
        return T(h(k))
}

delete(k) {
        T(h(k)) = null
}
```

Хеш-функция

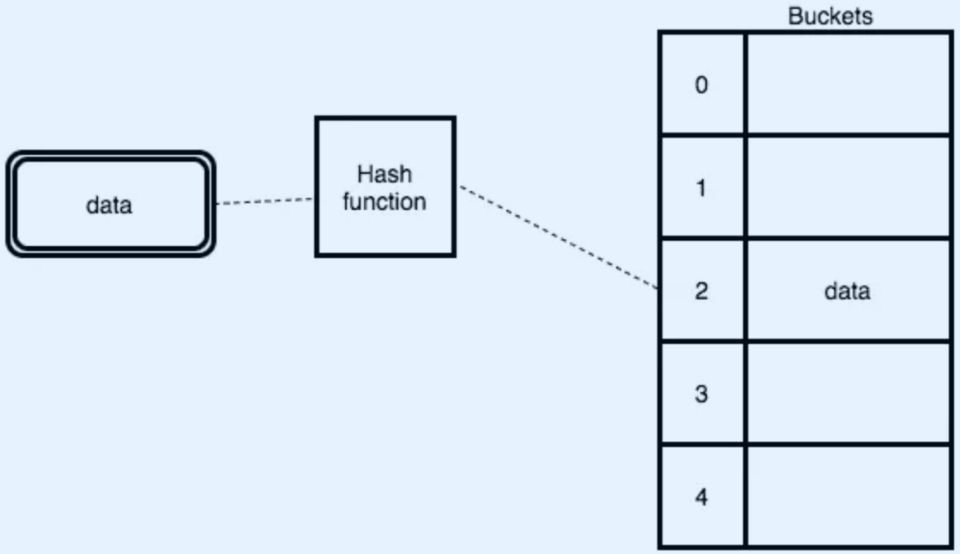
- Каждый ключ преобразовываем с помощью хеш-функции в индекс таблицы.
- Вне зависимости от схожести ключей результирующие индексы должны быть разными.



education

Хеш-функция

- Определяет, как и с каким индексом будет храниться значение.
- Она должна быть последовательна, то есть для одного и того же ключа возвращать одно и то же значение.
- Равномерно распределённой (отсутствует связь между ключом и итоговым индексом) для конкретного размера.
  В нашем случае — от 0 до n – 1.
- Должна быть простой для вычисления.
- Выходные значения не должны выходить за заданный диапазон.



education

Переполнение таблицы

- **Load factor n/m** — соотношение количества элементов и размера таблицы. Если значение стремится к 1 (~0,75), то нам нужна новая аллокация памяти.
- Оптимизируем хеш-функцию под новый объём данных.
- Перехешируем все ключи, заново переписывая все индексы.

0	null
1	Апельсины, 100
2	Бананы, 70
3	null
4	Шампунь, 350

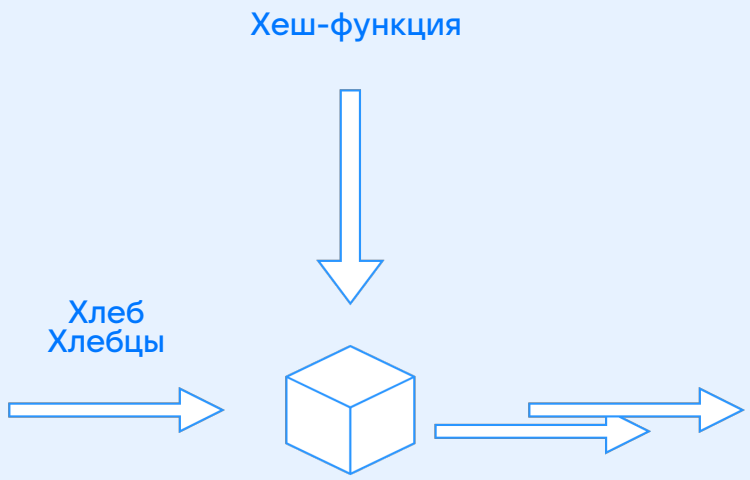
0	null
1	Апельсины, 100
2	Бананы, 70
3	null
4	null
5	Сахар
6	null
7	null
8	null
9	Шампунь, 350

Проблемы хеширования

- Мы не можем всем ключам выдавать уникальные индексы.
- Мы можем лишь подобрать хорошую хеш-функцию, которая бы минимизировала такую вероятность.
- Равномерность распределения для избежания дублирования индексов.

Коллизии

- Разным ключам наша хеш-функция выдаёт одинаковые значения (парадокс дней рождений).
- Для каждой h(k) можно подобрать такие k, что индексы на выходе будут идентичными.
- Теперь нам надо хранить по хешу пару «ключ-значение», чтобы каждый раз иметь возможность удостовериться, что мы выбрали нужное нам значение.



0	null
1	Апельсины, 100
2	Бананы, 70
3	Баклажан, 90
4	null
5	null
6	null
7	null
8	null
9	Шампунь, 350

Парадокс дней рождений

Для группы в 23 человека вероятность совпадения дней рождений больше 50%.

$$\bar{p}(n) = 1 \cdot \left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right) \cdots \left(1 - \frac{n-1}{365}\right) = \frac{365 \cdot 364 \cdots (365 - n + 1)}{365^n} = \frac{365!}{365^n (365 - n)!}$$

$$p(n) = 1 - \bar{p}(n)$$

n	$p(n)$
10	12 %
20	41 %
30	70 %
50	97 %
100	99.99996 %
200	99.9999999999999999999999998 %
300	$(1 - 7 \times 10^{-73}) \times 100$ %
350	$(1 - 3 \times 10^{-131}) \times 100$ %
366	100 %

education

Пример хеширования

- Любой ключ можем перевести через таблицу ASCII.

  tap — 116 97 112.
- Далее можем провести любую операцию с этими числами, например сложение.
- Берём остаток от деления результата h(325).
- Как быть со словом pat?

Построение хеш-функции

Для диапазона от [0, m – 1]:

Метод остатка от деления

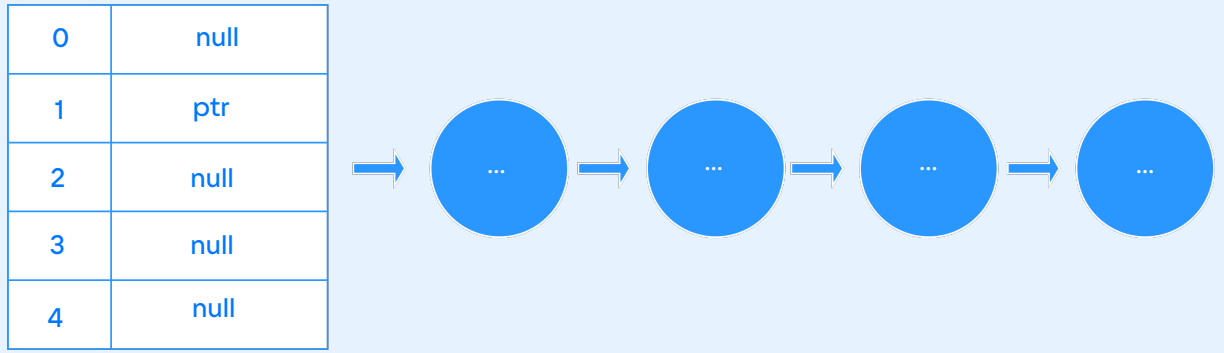
- h(k) = k % m (остаток от деления). Индекс, который мы получим в результате такого преобразования, не выйдет за пределы m – 1.
- m — простое число, отличное от степени 2.
  В таком случае у нас будет наилучшее распределение значений индексов. Но в то же время m — размер нашей хеш-функции,
  и подобного рода ограничения могут быть
  не очень удобны.

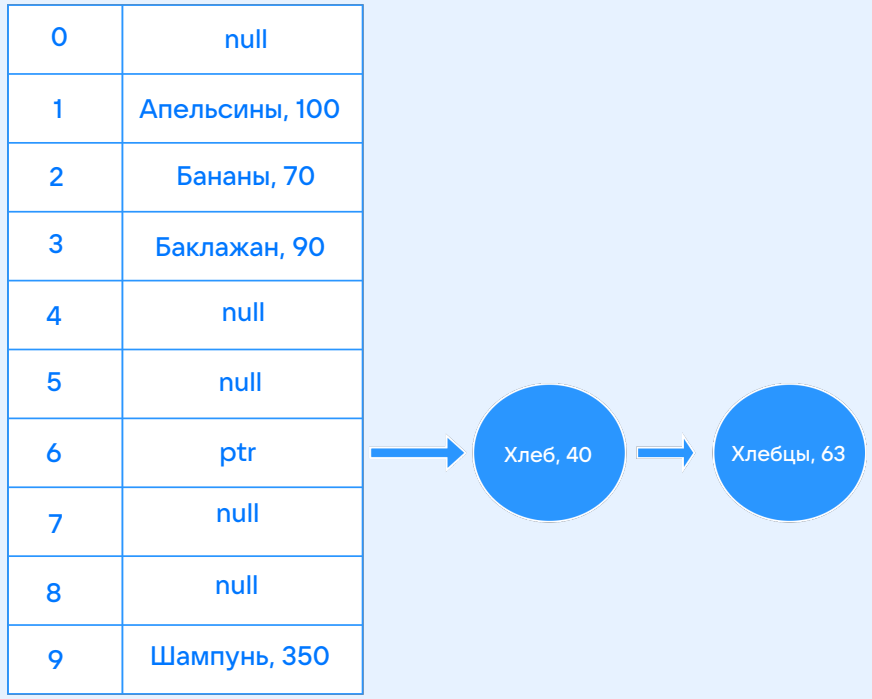
Метод умножения

- h(k) = ⌊m * ((k * A) mod 1)⌋ ⌊⌋ — эти скобки — округление до наименьшего целого.
- A — вещественное число, константа. 0 < A < 1; A * k — тоже вещественное; mod 1 выделяет вещественную часть.
- Необходимо подобрать A между 0 и 1 так, чтобы было максимально равномерное распределение между 0 и m – 1, и округлить в наименьшую сторону.
- Кнут предложил A считать A = $(\sqrt{5} - 1)/2 = 0{,}618$. Такое значение даёт равномерное распределение значений хеш-функции.
- Этот метод даёт возможность убрать ограничения на возможные значения m.
- Всегда существует набор хеш-функций, которые присваивают таблице при создании.

Метод цепочек

- На каждое одинаковое значение создаётся связанный список.
- Если список уже есть, то значение дописываем в конец.
- Простота реализации.
- В худшем варианте выборка будет стремиться к O(n).
- Теперь каждая ячейка должна хранить хеш-таблицы.





education

Метод цепочек. Вставка, удаление и поиск

Как будет работать вставка при методе цепочек?

- У нас есть две пары «ключ-значение»: K1 V1 и K2 V2.
- Пытаемся вставить первую пару K1 V1. Предположим, что наша хеш-функция по ключу K1 присвоила индекс 3, следующей мы пытаемся вставить вторую пару K2 V2.
- Ключу K2 также присваивают индекс 3, в данной ситуации мы просто записываем вторую пару в конец списка.
- Аналогичным образом будут работать поиск и удаление.

- Мы пропускаем через хеш-функцию наш ключ, получаем индекс три, находим по этому индексу нужный список и уже по ключу будем искать в списке нашу пару.

Открытая адресация

1

В случае коллизии ключ храним в соседней ячейке.

2

Значения всегда хранятся в ячейках таблицы.

3

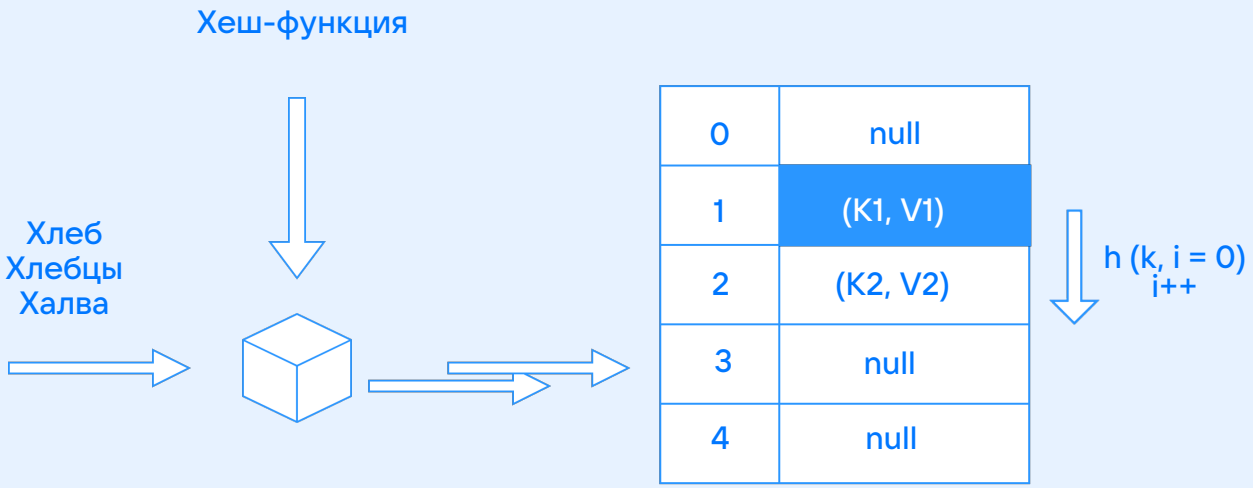
При выборке в случае коллизии мы последовательно перебираем все значения с заданным ключом.

4

Наиболее эффективно, когда заранее известны максимальные размеры входящих данных.

Открытая адресация

- Допустим, у нас есть две пары «ключ-значение» (k1, v1) и (k2, v2).
- У первой пары по ключу вычислили индекс, равный 1. Далее мы хотим вставить вторую пару и по k2 получаем индекс, тоже равный 1.
- В этой ситуации самым простым будет просто увеличивать индекс на 1.
- Подобного рода разрешение коллизий, когда мы берём соседнюю ячейку, называется линейным пробированием.



education

Вставка

- Добавим в нашу хеш-функцию новый параметр i — количество попыток найти подходящую ячейку.
- Учитываем её каждый раз в случае коллизии:

  $h(k,i) = (h(k) + i) \bmod m$.
- Важно отметить, что при вставке, если мы дошли до конца таблицы, а место так и не нашли, мы переходим к её началу. Таким образом, мы равномерно распределяем все наши значения, но при этом и сложность операций тоже увеличивается.
- Возможно формирование длинных последовательностей занятых ячеек, что негативно сказывается на скорости выборки.
- При методе цепочек сложность ухудшается, когда у нас начинают увеличиваться списки, тут сложность ухудшается, если нам приходится проходить много ячеек, стоящих следом за значением, вызывающим коллизию.

Выборка

- Допустим, у нас есть три значения, давших коллизии. Нужный нам ключ — «Халва».
- Вычисляем индекс.
- Проходимся от нашего индекса, инкрементируя количество попыток.
- Если ключ не нашли и попали на пустую ячейку, значит, нашего ключа нет в таблице.
- Пустое значение может быть любым — это может быть как null-, nil-, None-значение и др., так и любая константа, которую вы выберете.

Удаление

- Так же, как и в случае выборки, ищем нужное значение.
- При удалении записываем в освободившуюся ячейку предопределённую константу, например removed.
- В случае, если при чтении данных мы попадаем на removed, мы просто продолжаем итерироваться, а не прекращаем поиск, как это было бы, окажись там null.
- В случае большого количества removed-ячеек мы начнём терять время на их перебор.
- Если появится очередная коллизия с ключом «Хлеб», то мы сможем вставить новую пару «ключ-значение» на место removed.

0	null
1	Хлеб, 40
2	REMOVED
3	Халва, 70
4	null

Реализация в коде

- h(v) для любых k находится в диапазоне [0, m – 1].
- (h(v) + i) mod m не превышает m.

```
insert(key, value) {
    for i < m {
        hash = (h(v) + i) mod m
            if T[hash] == null || T[hash] == REMOVED {
                T[hash] = value
            return
        }
    }
}
```

```
get(key) {
    for i < m {
        hash = (h(v) + i) mod m
      if T[hash] != null {
          if T[hash].key == key
            return T[hash]
      } else {
        return null
      }
    }
    return null
}
```

```
delete(key){
    for i < m {
      hash = (h(v) + i) mod m
     if T[hash] != null and T[hash] != REMOVED {
      if T[hash].key == key
            T[hash] = REMOVED
      }
    }
}
```

education

Двойное хеширование

- Метод открытой адресации тоже не идеален.
- Для диапазона от 0 до m – 1 добавляем вспомогательную хеш-функцию.
- В случае если ячейка h(k) занята i (номер попытки) = 0, то рассматриваем ячейку (h(k) + h1(k)) mod m, затем (h(k) + 2 * h1(k)) mod m и h(k, i) = (h(k) + i * h1(k)) mod m.
- h1(k) в данном случае — просто вспомогательная хеш-функция. Как видно из примера, результат её работы никогда не должен быть равен 0.
- Чтобы обойти все ячейки таблицы, h1(k) и m должны быть взаимно простыми, то есть у них не должно быть общих делителей, кроме 1.
- Значения, которые возвращает вспомогательная функция, также должны гарантировать эту возможность.
- Пример такой функции может быть: h1(k) = 1 + k mod (m – 1).

Двойное хеширование

- Вне зависимости от метода разрешения коллизий, мы должны ограничить длину поиска — перебор минимального значения при выборке и при вставке.
- 3–4 сравнения для каждого ключа говорят о неэффективности хеш-функции.
- В идеальном случае все ячейки будут заняты.

Будем ВКонтакте!