АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

ЛЕКЦИЯ 5

Хеш-функции

Хеш указывает, где искать ключ. Равенство ключей проверяется отдельно.

05

ИЗ 15 ЛЕКЦИЙ



МОДУЛЬ 2 Хеш-таблицы. Графы

1 КУРС / 2 АКАДЕМИЧЕСКИХ ЧАСА / PYTHON 3

Цель и содержание лекции

ЦЕЛЬ И РЕЗУЛЬТАТЫ ОБУЧЕНИЯ

понять связь между ключом, хешем и ячейкой таблицы, применять `dict` и `set` для поиска и исключения дубликатов и объяснять, как коллизии влияют на сложность операций.

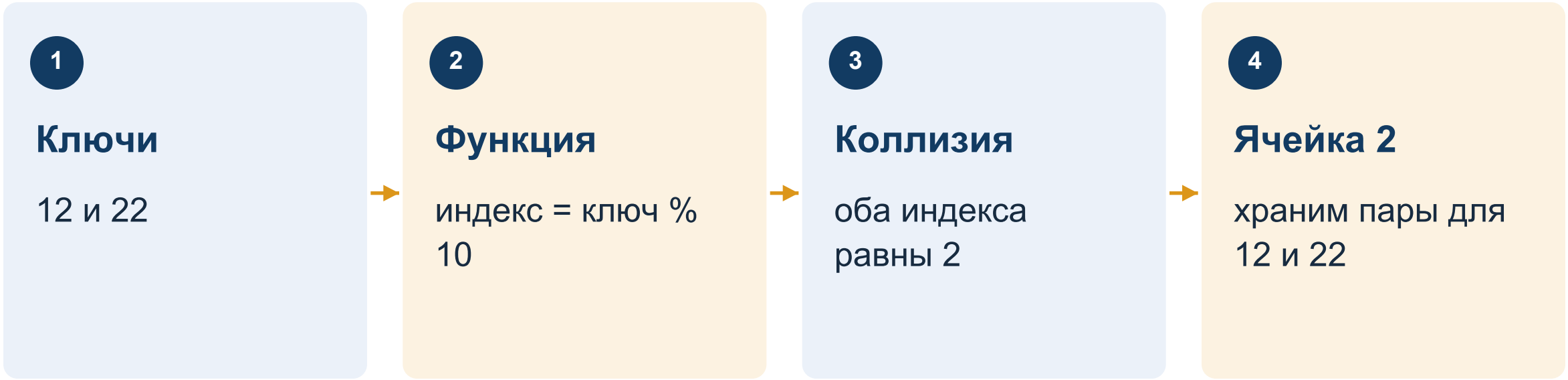
- Различать хеш, индекс ячейки и ключ.
- Объяснять коллизии и способы их обработки.
- Использовать dict и set для поиска и устранения повторов.

ПЛАН ЛЕКЦИИ

1. **Ключ, хеш-функция и адрес ячейки**
2. **Поиск по ключу и исключение дубликатов в Python**
3. **Коллизии и метод цепочек**
4. **Трассировка и реализация учебной хеш-таблицы**
5. **Коэффициент заполнения и стоимость операций**

Разные ключи могут попасть в одну ячейку

СХЕМА



При поиске сравниваем сам ключ с ключами в ячейке, чтобы различить 12 и 22.

От ключа к ячейке

01 ТЕМА

Пусть в лаборатории каждому прибору присвоен целочисленный код, а по коду требуется получить название прибора. Список пар `(код, название)` позволяет решить задачу последовательным просмотром. **Хеш-таблица** хранит пары ключ–значение и вычисляет, в какой части таблицы искать пару. Ключом здесь служит код, значением — название. Операция записи для уже существующего ключа обновляет связанное с ним значение.

Хеш-функция преобразует ключ в целое число, называемое хешем. Если таблица состоит из `m` ячеек, можно взять остаток от деления хеша на `m` и получить индекс в диапазоне от 0 до `m - 1`. Для учебной модели с целыми ключами возьмём сам ключ как хеш. Тогда ключ 12 при пяти ячейках отправляется в ячейку 2, поскольку `12 % 5 == 2`. Ключ 17 даёт тот же индекс.

От ключа к ячейке

01 ПРОДОЛЖЕНИЕ

Равные ключи должны приводить к одному и тому же хешу, иначе поиск мог бы обратиться в другую ячейку, чем вставка. Разные ключи могут иметь одинаковый хеш или получить один индекс после взятия остатка. При большем числе возможных ключей, чем ячеек, совпадения индексов неизбежны. Хорошая хеш-функция помогает распределять типичные ключи равномерно. Например, остаток по модулю 5 даёт плохое распределение, если все коды кратны 5.

Словарь и множество в Python

02 ТЕМА

В Python словарь `dict` связывает уникальные ключи со значениями. Запись `in` проверяет наличие ключа, а `get` позволяет указать результат для отсутствующего ключа. Прямое обращение `devices[31]` к отсутствующему ключу вызовет `KeyError`. Если `None` тоже хранится как значение, для различения отсутствующего ключа и существующего ключа со значением `None` используйте проверку `key in devices`.

Множество `set` хранит уникальные элементы и поддерживает проверку принадлежности. Чтобы убрать дубликаты и сохранить порядок первых появлений, проходим исходную последовательность слева направо. Множество `seen` хранит уже встреченные коды, а список `unique_codes` — результат. После обработки каждой позиции множество содержит все коды обработанной части, а список содержит первые появления этих кодов в исходном порядке. Повторный код оставляет оба результата прежними.

Словарь и множество в Python

02 ПРИМЕР PYTHON

05_01.py

```
devices = {12: "весы", 17: "термометр", 9: "таймер"}
print(devices.get(17, "код не найден"))
print(devices.get(31, "код не найден"))

codes = [17, 12, 17, 9, 12, 22]
seen = set()
unique_codes = []
for code in codes:
    if code not in seen:
        seen.add(code)
        unique_codes.append(code)
print(unique_codes)
```

РЕЗУЛЬТАТ ЗАПУСКА

```
термометр
код не найден
[17, 12, 9, 22]
```

Словарь и множество в Python

02 ПРОДОЛЖЕНИЕ

Ключи словаря и элементы множества должны быть хешируемыми. К числу подходящих ключей относятся целые числа, строки и кортежи из хешируемых элементов. Изменяемый список нельзя напрямую использовать как ключ. Если ключом служит код вместе с номером кабинета, можно использовать кортеж `(код, номер_кабинета)`. Выбор ключа определяет, какие записи считаются одинаковыми. Для подсчёта уникальных приборов название может быть слишком общим, поскольку несколько приборов имеют один тип.

Как разрешить коллизию

Коллизией в хеш-таблице называют ситуацию, когда разные ключи претендуют на одну ячейку. В методе цепочек каждая ячейка хранит набор пар, относящихся к этому индексу. В учебном коде цепочка представлена списком. При поиске вычисляем индекс, затем последовательно сравниваем ключи внутри этой цепочки. Совпадение индекса позволяет найти нужную цепочку, а совпадение ключа определяет пару.

Добавим ключи 12, 17, 9 и 22 в таблицу с пятью ячейками.

Как разрешить коллизию

03 ПРОДОЛЖЕНИЕ

Индекс	Ключи в цепочке	Как получен индекс
0	Пусто	—
1	Пусто	—
2	12, 17, 22	Остаток 2 у всех трёх ключей
3	Пусто	—
4	9	`9 % 5 = 4`

Поиск ключа 17 проверяет в ячейке 2 сначала 12, затем 17 и возвращает его значение. Поиск ключа 32 выбирает ту же ячейку, проверяет все три пары и сообщает об отсутствии. Чтобы обновить значение для 17, меняем соответствующую пару. Если добавить ещё одну пару с ключом 17, в цепочке появятся две конкурирующие записи.

Учебная реализация

04 ПРИМЕР PYTHON

05_02.py / фрагмент 1 из 2

```
class ChainedTable:
    def __init__(self, capacity):
        if capacity <= 0:
            raise ValueError("capacity must be positive")
        self.buckets = [[] for _ in range(capacity)]

    def put(self, key, value):
        bucket = self.buckets[key % len(self.buckets)]
        for index, (stored_key, _) in enumerate(bucket):
            if stored_key == key:
                bucket[index] = (key, value)
                return
        bucket.append((key, value))

    def get(self, key):
        bucket = self.buckets[key % len(self.buckets)]
        for stored_key, value in bucket:
            if stored_key == key:
                return value
        raise KeyError(key)
```

Учебная реализация

04 ПРИМЕР PYTHON

05_02.py / фрагмент 2 из 2

```
table = ChainedTable(5)
table.put(12, "весы")
table.put(17, "термометр")
table.put(9, "таймер")
table.put(22, "гигрометр")
print(table.get(17))
table.put(17, "термометр резервный")
print(table.get(17))
try:
    table.get(32)
except KeyError:
    print("код не найден")
```

РЕЗУЛЬТАТ ЗАПУСКА

```
термометр
термометр резервный
код не найден
```

Учебная реализация

04 ПРОДОЛЖЕНИЕ

Модель принимает целочисленные ключи и фиксированное положительное целое число ячеек. Списки для ячеек создаются по отдельности. Выражение `[[]] * capacity` создало бы несколько ссылок на один общий список, и запись в одну ячейку отразилась бы в остальных. В `put` и `get` используется одно правило вычисления индекса. Каждая записанная пара остаётся в своей цепочке, а проверка ключа гарантирует, что возвращается значение нужной пары.

Заполнение, время и память

Для `n` хранимых пар и `m` ячеек **коэффициент заполнения** равен `n / m`. В таблице с цепочками это средняя длина цепочки по всем ячейкам. В примере с четырьмя парами и пятью ячейками среднее равно 0,8, хотя длина одной цепочки равна 3. При равномерном распределении ожидаемая стоимость поиска с цепочками составляет `O(1 + n/m)` в модели, где вычисление хеша и сравнение ключей стоят `O(1)`. При ограниченном коэффициенте заполнения это `O(1)`. Если все ключи попали в одну цепочку, поиск может стоить `O(n)`.

Таблица с изменяемой ёмкостью может увеличить число ячеек при росте заполнения. При этом пары нужно распределить заново, поскольку остаток зависит от нового `m`. Отдельное расширение требует переноса всех пар. При равномерном хешировании и геометрическом росте ёмкости ожидаемая амортизированная стоимость вставки составляет `O(1)`. Учебная таблица занимает `O(m + n)` памяти и показывает, как с ростом числа записей меняется длина цепочек. Для исключения дубликатов с `set` ожидаемое время составляет `O(n)` в той же модели стоимости, а дополнительная память составляет `O(k)` для `k` разных ключей, включая список результата.

Заполнение, время и память

В примере можно сравнить ёмкости 5 и 7 для одного набора ключей. При пяти ячейках 12, 17 и 22 попадают в одну цепочку, а при семи получают индексы 5, 3 и 1. Ключ 9 получает индекс 2. Если изменить только размер списка ячеек и оставить старые пары на прежних позициях, поиск начнёт обращаться по новым индексам и потеряет часть записей. После расширения полезно проверить поиск всех старых ключей и убедиться, что число пар осталось прежним.

Что нужно уметь после лекции

ИТОГИ

Хеш-функция помогает найти ячейку по ключу. Метод цепочек хранит все пары, попавшие в одну ячейку, и различает их сравнением ключей. Для типичных задач Python использует `dict` для связи ключа со значением и `set` для проверки принадлежности. Стоимость поиска зависит от распределения ключей и заполнения таблицы.

> Хеш указывает, где искать ключ. Равенство ключей проверяется отдельно.

Проверьте понимание

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Чем ключ отличается от хеша и индекса ячейки?
2. Почему при совпадении индексов нужно сравнивать ключи?
3. Что происходит при повторной записи уже существующего ключа?
4. Как множество помогает удалить дубликаты с сохранением порядка?
5. Почему увеличение числа ячеек требует пересчитать индексы существующих пар?

Самостоятельная работа

ПРАКТИКА

1. Разместите ключи `[6, 14, 22, 23, 30, 9]` по методу цепочек при `m = 8`. Покажите поиск 22 и отсутствующего 38 и вычислите коэффициент заполнения.
2. Уберите повторы из списка `['K-2', 'K-1', 'K-2', 'K-3', 'K-1']`, сохранив порядок первых появлений. Постройте через `dict` ещё и таблицу частот, чтобы сверить результат.
3. Добавьте к `ChainedTable.get` счётчик сравнений. Сравните поиск отсутствующего ключа после вставки 0, 5, 10, 15 и после вставки 0, 1, 2, 3 при пяти ячейках.

Источники и документация

ЧТЕНИЕ

ОСНОВНОЕ ЧТЕНИЕ

А. Бхаргава. Грокаем алгоритмы

СПб.: Питер, 2017. Глава 5. Хеш-таблицы. С. 100–126.

> При чтении сначала вручную пройдите один пример, затем запустите код и сравните результат.