Алгоритмы и структуры данных

Лекция 6

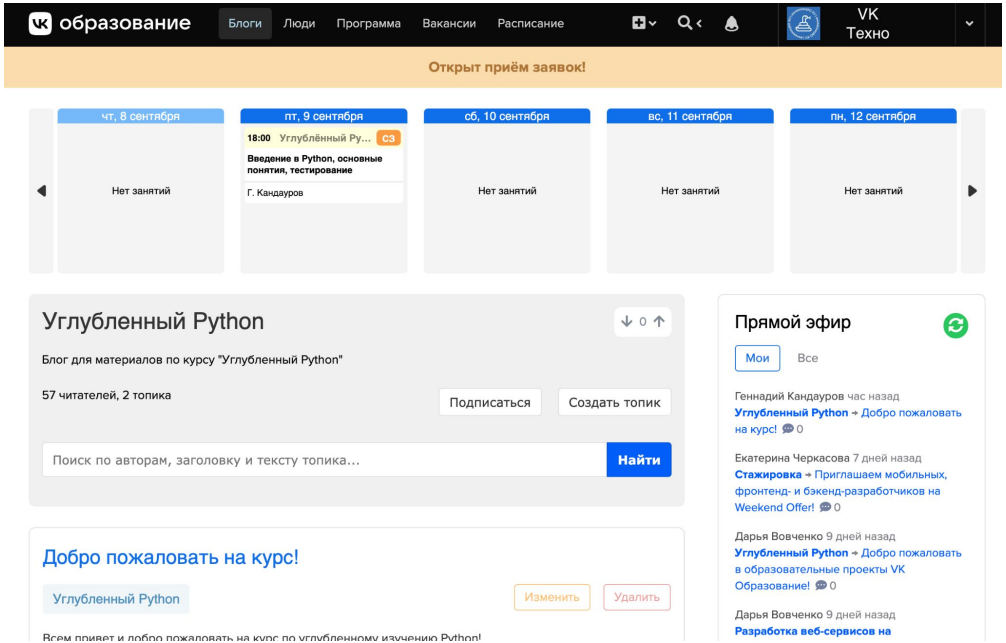
Хеш, хеширование

Кандауров Геннадий

education

Напоминание отметиться на портале

+ оставить отзыв



Квиз про прошлой лекции

Содержание занятия

1. Хеш
2. Коллизии
3. Что в Python?

Хеширование

Хеш: постановка задачи

Задача. Хранить ключи в контейнере с быстрыми операциями:

- быстро добавлять,
- быстро удалять,
- быстро проверять наличие.

Решение 1. Неупорядоченный массив:

- быстрое добавление – O(1),
- длительное удаление – O(n),
- длительный поиск – O(n).

Решение 2. Упорядоченный массив:

- длительное добавление – O(n),
- длительное удаление – O(n),
- быстрый поиск – O(log n).

Хеш

Частное решение 3.

Пусть ключи – неотрицательные целые числа в диапазоне [0, ..., n - 1].

Будем хранить A – массив bool.

A[i] == True, если *i* содержится:

- мгновенное добавление – O(1),
- мгновенное удаление – O(1),
- мгновенный поиск – O(1).

Хеш

Хеширование – преобразование ключей к числам.

Хеш-таблица – массив ключей с особой логикой, состоящей из:

1. Вычисления хеш-функции, которая преобразует ключ поиска в индекс.
2. Разрешения конфликтов, т.к. два и более различных ключа могут преобразовываться в один и тот же индекс массива.

Отношение порядка над ключами не требуется.

Хеш-функция

Хеш-функция — преобразование по детерминированному алгоритму входного массива данных произвольной длины (один ключ) в выходную битовую строку фиксированной длины (значение).

Результат вычисления хеш-функции называют *хешем*.

Коллизией хеш-функции *H* называется два различных входных блока данных X и Y таких, что *H(X) == H(Y)*.

Количество возможных значений хеш-функции не больше M и для любого ключа k:

$$0 \leq h(k) < M$$

Важно! Хорошая хеш-функция должна:

1. Быстро вычисляться.
2. Минимизировать количество коллизий.

HASH = рубить, перемешивать.

Хеш-функция

Качество хеш-функции зависит от задачи и предметной области.

Пример плохой хеш-функции.

h(k) = [последние [три] цифры k] = k % 1000.

Такая хеш-функция порождает много коллизий, если множество ключей – цены.

Частые значения:
000, 500, 999, 998, 990, 900

Хеш-функция: метод деления

h(n) = n mod M

M определяет размер диапазона значений: [0, .. , M−1].

Как выбрать M?

- Если M == 2 ** k, то значение хеш-функции не зависит от старших битов.
- Если M == 2 ** 8 - 1, то значение хеш-функции не зависит от перестановки байт.

Хорошо в качестве M брать простое число, далекое от степеней двойки.

Хеш-функция: метод деления

Сумма Флетчера - это остаток от деления интерпретируемого как длинное число потока данных на 255.

Пусть G - длинное число потока данных, $B=2^8=256$, D = B - 1

$$G \% D = (x_n * B^n + \ldots + x_1 * B + x_0) \% D =$$

$$= (x_n * (\ldots) * D + x_n + \ldots + x_1 * D + x_1 + x_0) \% D =$$

$$= ((\ldots) * D \% D + (x_n + \ldots + x_1 + x_0) \% D) \% D =$$

$$= (x_n + \ldots + x_1 + x_0) \% D$$

- $(D+1)^n = D^n + \ldots + D + 1 = (\ldots) * D + 1$
- $(a + b) \% d = (a \% d + b \% d) \% d$

Хеш-функция: метод умножения

$$h(k) = [M \cdot \{k \cdot A\}],$$

{} – дробная часть
[] – целая часть
A – действительное число, 0 < A < 1

M определяет диапазон значений: [0, .. , M−1].

Кнут предложил в качестве A использовать число, обратное к золотому сечению:

$$A = \phi^{-1} = \left(\frac{\sqrt{5}-1}{2}\right) = 0.6180339887\ldots$$

Такой выбор A дает хорошие результаты хеширования.

Хеш-функция: метод умножения

Хеш-функцию *h(k) = [M * {k * A}]* вычисляют без использования операций с числами с плавающими точками.

Пусть M – степень двойки, M = 2 ** p, p <= 32.

Вместо действительного числа A берут близкое к нему

$$A = \frac{s}{2^{32}} = \frac{2654435769}{2^{32}}$$

то есть s = 2654435769

Тогда $h(k) = \left[2^{p} \cdot \left\{k \cdot \frac{s}{2^{32}}\right\}\right] = \left[2^{p} \cdot \left\{\frac{r_1 2^{32} + r_0}{2^{32}}\right\}\right] = \left[2^{p} \cdot \frac{r_0}{2^{32}}\right]$

$$= \left[\frac{r_0}{2^{32-p}}\right] = \left[\frac{r_{01} 2^{32-p} + r_{00}}{2^{32-p}}\right] = r_{01}$$

Старшие p бит r_0

Итого: `h(k) = (k * s mod 2 ** 32) >> (32 - p)`

Хеш-функция: строки

Строка $s = s_0, s_1, \ldots, s_{n-1}$.

Вариант 1.

$$h_1(s) = (s_0 + s_1 a + s_2 a^2 + \ldots + s_{n-1} a^{n-1}) \bmod M$$

Вариант 2.

$$h_2(s) = (s_0 a^{n-1} + s_1 a^{n-2} + \ldots + s_{n-2} a + \ldots + s_{n-1}) \bmod M$$

Число M – степень двойки.

Важно правильно выбрать константу *a*.

Хотим, чтобы при изменении одного символа, хеш-функция изменялась.
То есть, чтобы все значения s * a mod M, 0 <= s < M были различны.
Для этого достаточно, чтобы *a* и *M* были **взаимно простыми**.

Хеш-функция: строки

$h_2(s)$ вычисляется эффективнее, если использовать метод Горнера:

$$h_2(s) = (((s_0a + s_1)a + s_2)a + \ldots + s_{n-2})a + \ldots + s_{n-1}$$

$h_1(s)$ можно вычислять аналогично, но начиная с конца строки.

Примечание

В с-строках известен только указатель на начало строки, но не размер, поэтому удобнее вычисля $h_2(s)$.

Хеш: коллизии

Парадокс дней рождений.

Сколько необходимо взять человек, чтобы вероятность совпадения дней рождения (число и месяц) хотя бы у двух людей превышала 50 %?

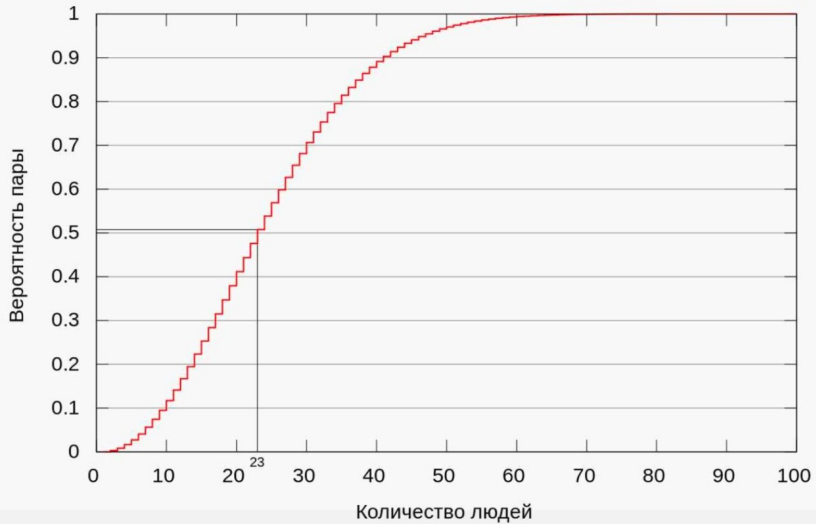
Хеш: коллизии

Вычислим вероятность того, что все дни рождения в группе будут различными:

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

Тогда вероятность того, что хотя бы у двух человек из *n* дни рождения совпадут:

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



Ответ: 23.

Хеш: коллизии

При вставке в хеш-таблицу размером 365 ячеек всего лишь 23-х элементов вероятность коллизии уже превысит 50 %, при вставке 50 элементов вероятность превысит 97% (если каждый элемент может равновероятно попасть в любую ячейку).

Хеш-таблицы различаются по методу разрешения коллизий.

Основные **методы** разрешения коллизий:

1. Метод цепочек.
2. Метод открытой адресации.

Хеш: коллизии

Хеш-таблица – структура данных, хранящая ключи в таблице. Индекс ключа вычисляется с помощью хеш-функции. Операции: добавление, удаление, поиск.

Пусть хеш-таблица имеет размер M, количество элементов в хеш-таблице – N.

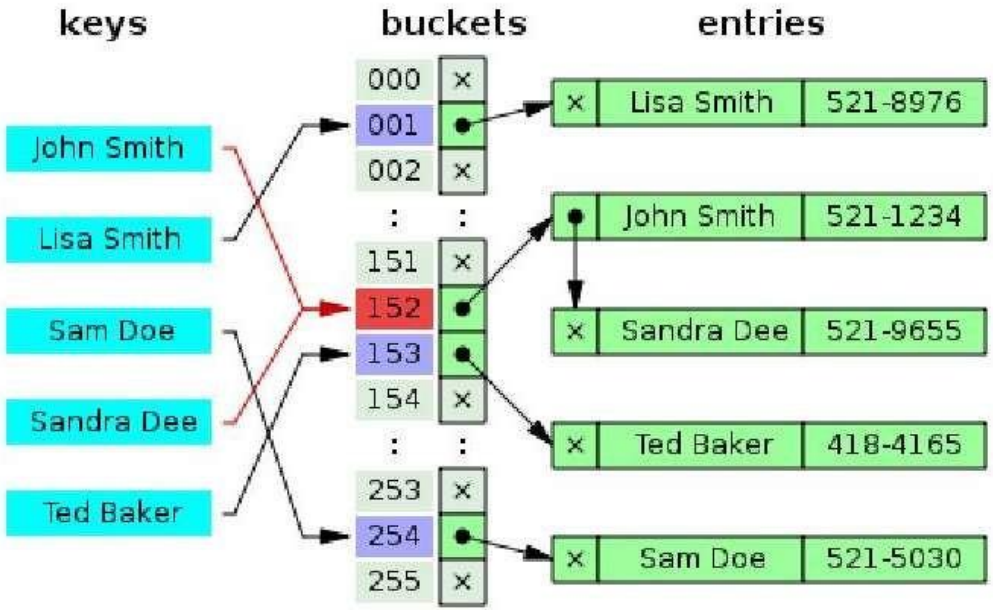
Число хранимых элементов, деленное на размер массива (число возможных значений хеш-функции), называется **коэффициентом заполнения хеш-таблицы** (load factor). Обозначим его $\alpha = N / M$.

Этот коэффициент является важным параметром, от которого зависит среднее время выполнения операций.

Хеш: метод цепочек

Каждая ячейка массива является указателем на связный список (цепочку).

Коллизии приводят к тому, что появляются цепочки длиной более одного элемента.



Хеш: метод цепочек

Добавление ключа.

1. Вычисляем значение хеш-функции добавляемого ключа – h.
2. Находим A[h] – указатель на список ключей.
3. Вставляем в начало списка (в конец списка дольше). Если запрещено дублировать ключи, то придется просмотреть весь список.

Время работы:

В лучшем случае – **O(1)**.

В худшем случае

- если не требуется проверять наличие дубля, то **O(1)**,
- иначе – **O(N)**.

Хеш: метод цепочек

Удаление ключа.

1. Вычисляем значение хеш-функции удаляемого ключа – h.
2. Находим A[h] – указатель на список ключей.
3. Ищем в списке удаляемый ключ и удаляем его.

Время работы:

В лучшем случае – **O(1)**.
В худшем случае – **O(N)**.

Хеш: метод цепочек

Поиск ключа.

1. Вычисляем значение хеш-функции ключа – h.
2. Находим A[h] – указатель на список ключей.
3. Ищем его в списке.

Время работы:
В лучшем случае – **O(1)**.
В худшем случае – **O(N)**.

Хеш: метод цепочек

Среднее время работы.

Теорема. Среднее время работы операций поиска, вставки (с проверкой на дубликаты) и удаления в хеш-таблице, реализованной методом цепочек – $O(1+\alpha)$, где α – коэффициент заполнения таблицы.

Доказательство. Среднее время работы – математическое ожидание времени работы в зависимости от исходного ключа.

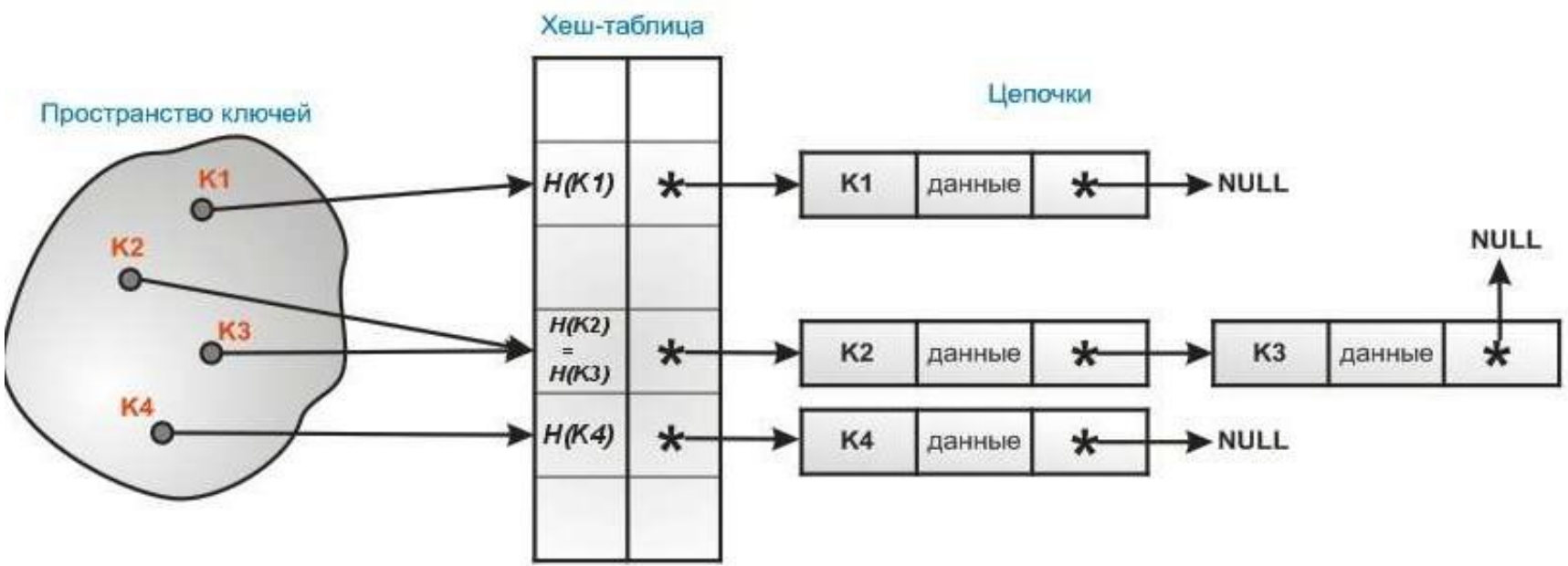
Время работы для обработки одного ключа $T(k)$ зависит от длины цепочки и равно $O(1+N_{h(k)})$, где N_i – длина i-ой цепочки. Предполагаем, что хеш-функция равномерна, а ключи равновероятны.

Среднее время работы

$$T_{ср}(M,N) = M\big(T(k)\big) = \sum_{i=0}^{M-1} \frac{1}{M}(1+N_i) = \frac{1}{M}\sum_{i=0}^{M-1}(1+N_i) = \frac{M+N}{M}$$

$$= 1+\alpha$$

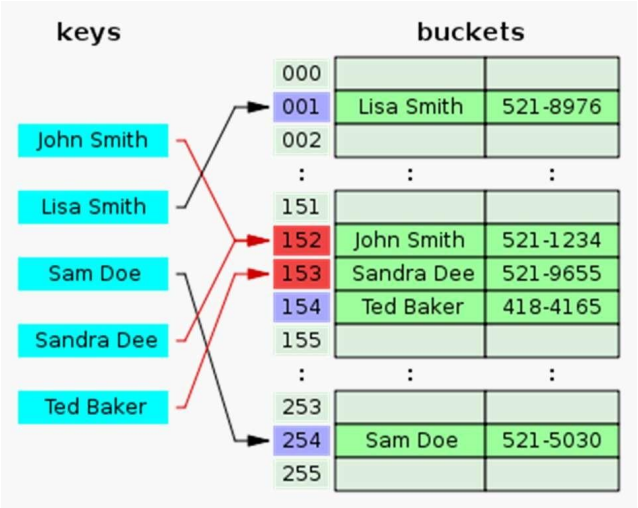
Хеш: метод цепочек



Хеш: открытая адресация

Все элементы хранятся непосредственно в массиве. Каждая запись в массиве содержит либо элемент, либо NIL.

При поиске элемента систематически проверяем ячейки до тех пор, пока не найдем искомый элемент или не убедимся в его отсутствии.



Хеш: открытая адресация

Вставка ключа.

1. Вычисляем значение хеш-функции ключа – h.
2. Систематически проверяем ячейки, начиная от A[h], до тех пор, пока не находим пустую ячейку.
3. Помещаем вставляемый ключ в найденную ячейку.

В п.2 поиск пустой ячейки выполняется в некоторой последовательности. Такая последовательность называется **«последовательностью проб».**

Хеш: открытая адресация

Последовательность проб зависит от вставляемого в таблицу ключа. Для определения исследуемых ячеек расширим хеш-функцию, включив в нее номер пробы (от 0).

$$h : U \times \{0, 1, \dots, M-1\} \to \{0, 1, \dots, M-1\}.$$

Важно, чтобы для каждого ключа k последовательность проб

$$\langle h(k, 0), h(k, 1), \dots, h(k, M-1) \rangle$$

представляла собой перестановку множества $\langle 0,1, \dots, M-1 \rangle$, чтобы могли быть просмотрены все ячейки таблицы.

Хеш: открытая адресация

Поиск ключа.

Исследуется та же последовательность, что и в алгоритме вставки ключа.

Если при поиске встречается пустая ячейка, поиск завершается неуспешно, поскольку искомый ключ должен был бы быть вставлен в эту ячейку в последовательности проб, и никак не позже нее.

Хеш: открытая адресация

Удаление ключа.

Алгоритм удаления достаточен сложен.

Нельзя при удалении ключа из ячейки i просто пометить ее значением NIL. Иначе в последовательности проб для некоторого ключа (или некоторых) возникнет пустая ячейка, что приведет к неправильной работе алгоритма поиска.

Решение. Помечать удаляемые ячейки спец. значением «Deleted».

Нужно изменить методы поиска и вставки.

В методе вставки проверять «Deleted», вставлять на его место.
В методе поиска продолжать поиск при обнаружении «Deleted».

Хеш: открытая адресация

Вычисление последовательности проб

Желательно, чтобы для различных ключей **k** последовательность проб
h(k, 0), h(k, 1), ..., h(k, M - 1)
давала большое количество последовательностей - перестановок множества {0, 1, ..., M - 1}

Обычно используется три метода построения h(k, i):

1. Линейное пробирование
2. Квадратичное пробирование
3. Двойное хеширование

Хеш: открытая адресация

Линейное пробирование

$$h(k, i) = (h'(k) + c\ i) \bmod M$$

Основная проблема - кластеризация.
Последовательность подряд идущих занятых элементов таблицы быстро увеличивается, образуя кластер.
Попадание в элемент кластера при добавление гарантирует “одинаковую прогулку” для различных ключей и проб.
Новый элемент будет добавлен в конец кластера, увеличивая его.

Если $h(k_1, i) = h(k_2, j)$, то $h(k_1, i + r) = h(k_2, j + r)$ для всех r.

Хеш: открытая адресация

Квадратичное пробирование

$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \bmod M$$

Требуется, чтобы последовательность проб содержала все индексы 0, ..., M-1.

Требуется подбирать c_1 и c_2.

При $c_1 = c_2 = 1/2$ проба вычисляется рекуррентно:

$$h(k, i + 1) = (h(k, i) + i + 1 \pmod M)$$

Возникает вторичная кластеризация. Проявляется на ключах с одинаковым хеш-значением $h'(.)$.

Если $h(k_1, 0) = h(k_2, 0)$, то $h(k_1, i) = h(k_2, i)$ для всех i.

Соответствует цепочкам в методе цепочек. Разница лишь в том, что в методе открытой адресации эти цепочки могут еще пересекаться.

Хеш: открытая адресация

Квадратичное пробирование

Утверждение.

Если $c_1 = c_2 = 1/2$, а $M = 2^p$, то квадратичное пробирование выдает перестановку {0, 1, 2, … M - 1}.

Доказательство.

От противного. Пусть существуют i и j, $0 \le i, j \le M - 1$, для которых

$$\frac{i(i+1)}{2} \equiv \frac{j(j+1)}{2} (mod\ 2\ ).$$

Тогда $i^2 + i - j^2 - j = 2^{p+1}D$,

$(i - j)(i + j + 1) = 2^{p+1}D$,

Если i и j одинаковой четности, то i + j + 1 нечетна, но i - j не может делиться на 2^{p+1}.

Если i и j разной четности, то i - j нечетна, но i + j + 1 не может делиться на 2^{p+1},

тк $0 < i + j + 1 < 2^{p+1}$. Противоречие.

Хеш: открытая адресация

Двойное хеширование

$h(k, i) = (h_1(k) + i\, h_2(k)) \bmod M$

Требуется, чтобы последовательность проб содержала все индексы 0, 1, ..., M - 1.

Для этого все значения $h_2(k)$ должны быть взаимно простыми с M.

- M может быть степенью двойки, а $h_2(k)$ всегда возвращать нечетные числа.
- M простое, а $h_2(k)$ меньше M.

Общее количество последовательностей проб $O(M^2)$.

Хеш: открытая адресация

Анализ хеш-таблиц с открытой адресацией

Теорема.
Математическое ожидание количества проб при неуспешном поиске в хеш-таблице с открытой адресацией и коэффициентом заполнения $\alpha = n / m < 1$ в предположении равномерного хеширования не превышает $1 / (1 - \alpha)$.
Без доказательства.

Время работы методов поиска, добавления и удаления:

В лучше случае - O(1)

В худшем случае - O(N)

В среднем - $O(1 / (1 - \alpha))$

Хеш: открытая адресация

Плюсы

+ Основное преимущество метода открытой адресации – не тратится память на хранение указателей списка.
+ Нет элементов, хранящихся вне таблицы.

Минусы

– Хеш-таблица может оказаться заполненной. Коэффициент заполнения α не может быть больше 1.
– При приближении коэффициента заполнения α к 1 среднее время работы поиска, добавления и удаления стремится к N.
– Сложное удаление.

Хеш

	Лучший случай.	В среднем. Метод цепочек.	В среднем. Метод открытой адресации.	Худший случай.
Поиск	$O(1)$	$O(1+\alpha)$	$O\left(\frac{1}{1-\alpha}\right)$	$O(N)$
Вставка	$O(1)$	$O(1+\alpha)$	$O\left(\frac{1}{1-\alpha}\right)$	$O(N)$
Удаление	$O(1)$	$O(1+\alpha)$	$O\left(\frac{1}{1-\alpha}\right)$	$O(N)$

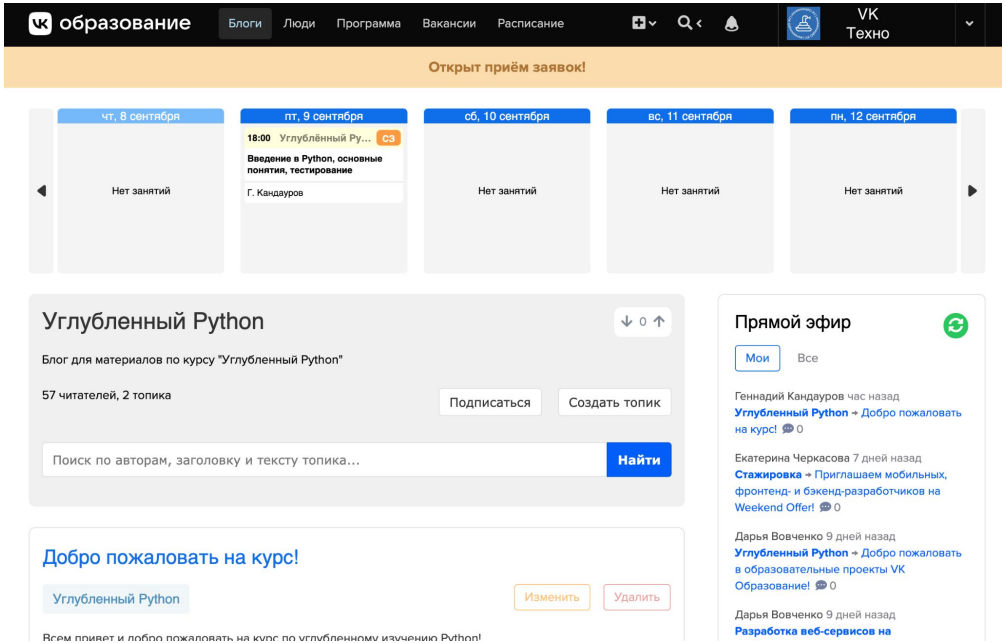
Хеш

Использование:

- checksum
- CRC (циклический избыточный код)
- Криптографические хеш-функции (MD5, SHA-1, SHA-256)
- Цифровая подпись DSA
- Блокчейн

Напоминание отметиться на портале Vol 2

+ оставить отзыв после лекции



Спасибо за внимание

education