

# Алгоритмы и структуры данных

Лекция 10

Хеш-таблицы.

Сергей Леонидович Бабичев

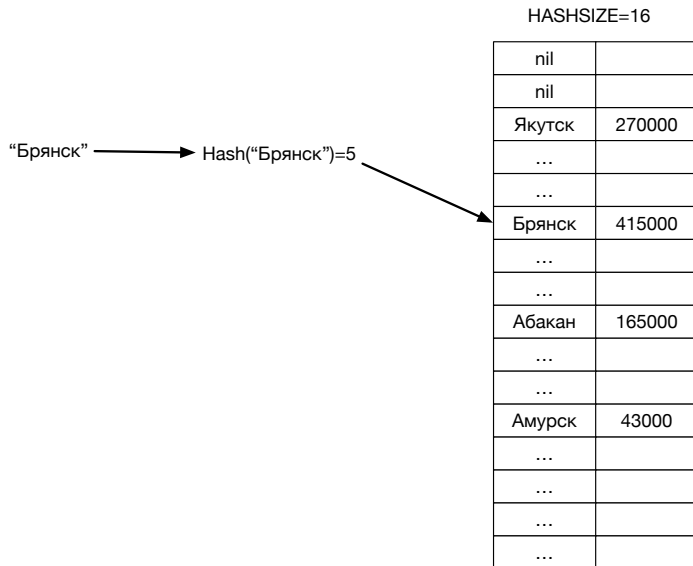
## План лекции

- 1 Хеш-таблицы
- 2 Сочетание хеш-таблиц и деревьев.
- 3 Хеш-таблицы во внешней памяти.

# Хеш-таблицы

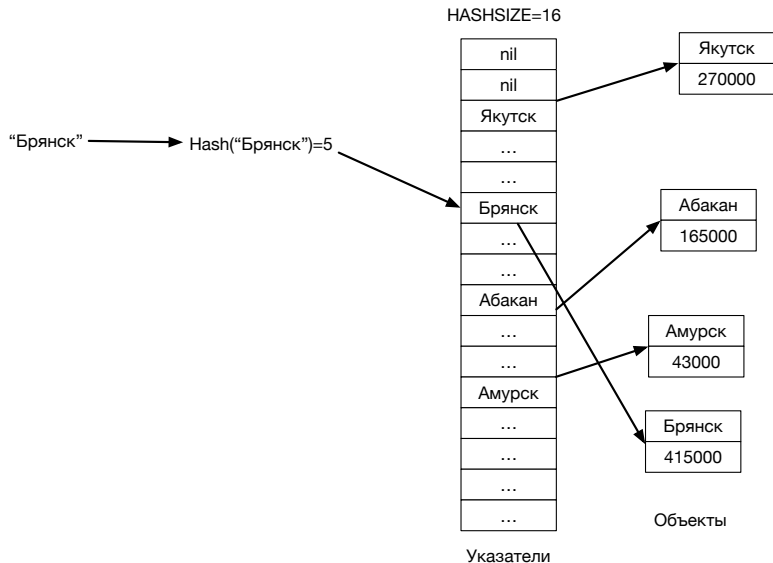
# Хеш-таблицы

- Простая хеш-таблица



# Хеш-таблицы

- Простая хеш-таблица, обычная реализация в виде массива указателей

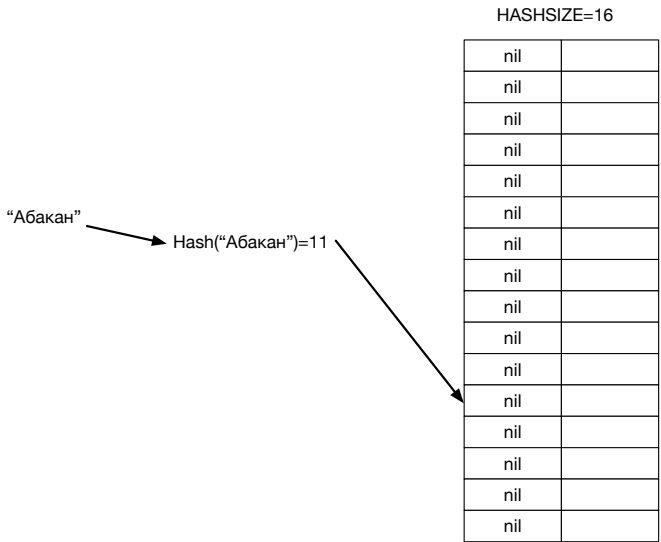


# Хеш-таблицы

- Известно количество элементов в контейнере  $C$
- Известен размер массива  $M$
- $\alpha = \frac{C}{M}$  — коэффициент заполнения, *fill-factor*, *load-factor*.
- $\alpha$  — главный показатель хеш-таблицы.

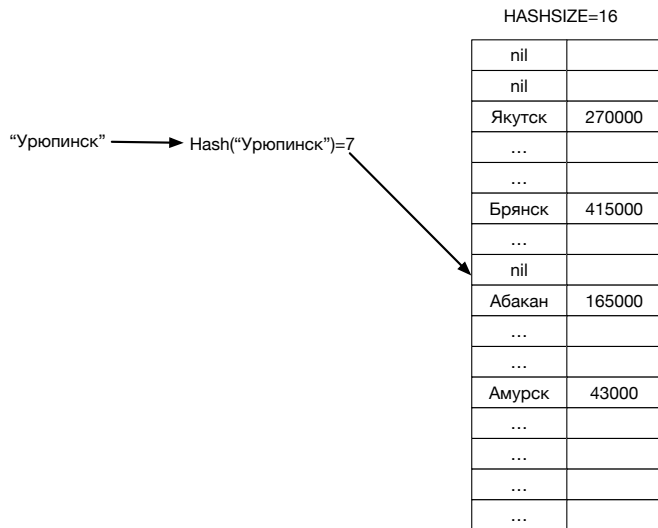
# Хеш-таблицы

- Операция создания хеш-таблицы



# Хеш-таблицы

- Операция создания хеш-таблицы требует операцию поиска.



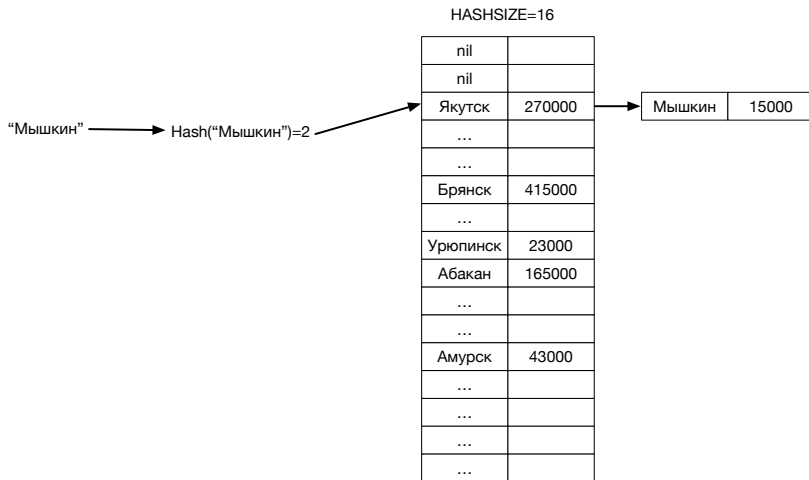


# Хеш-таблицы

- $\text{Hash}(\text{"Якутск"}) = 2$
- $\text{Hash}(\text{"Мышкин"}) = 2$
- Это — *коллизия*
- Коллизии — нежелательны.
- Без коллизий сложность операций поиска и вставки равна  $O(1)$
- Способы борьбы с коллизиями:
  - ▶ Прямая или закрытая адресация
  - ▶ Открытая адресация
  - ▶ Рехеширование

# Хеш-таблицы с прямой адресацией

- При коллизии во время создания элемента создаётся связный список конфликтующих.
- Можно использовать любую поисковую структуру данных.



# Хеш-таблицы с прямой адресацией

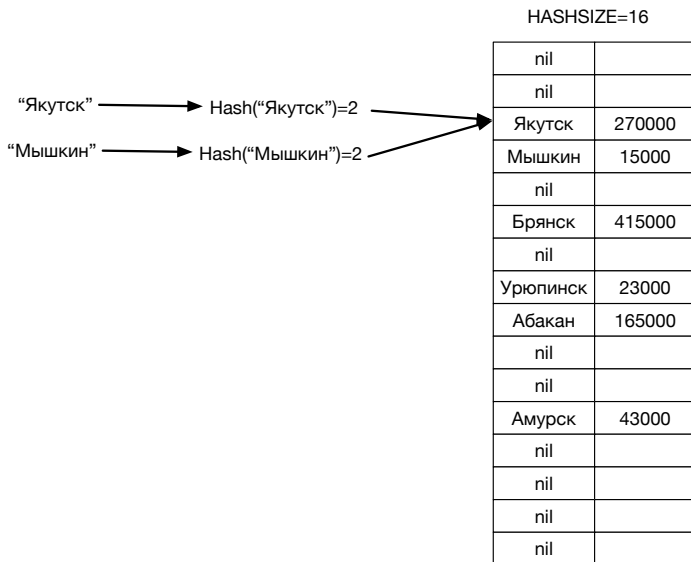
- 1 При поиске вычисляется хеш-функция.
- 2 Определяется место поиска — вторичная поисковая структура данных.
- 3 Если вторичной структуры нет, то нет и элемента.
- 4 Иначе элемент ищется во вторичной структуре.

# Хеш-таблицы с прямой адресацией

- 1 При удалении вычисляется хеш-функция.
- 2 Определяется место поиска — вторичная поисковая структура данных.
- 3 Если вторичной структуры нет, то нет и элемента.
- 4 Иначе элемент удаляется из вторичной структуре.
- 5 Если вторичная структура пуста, удаляет точку входа.

# Хеш-таблицы с открытой адресацией

- Другой способ поиска — искать в той же таблице повторно.



# Хеш-таблицы с открытой адресацией

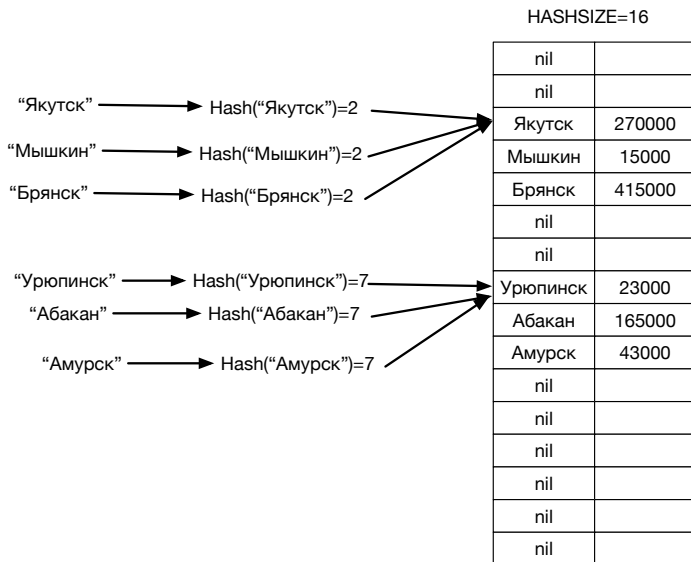
- 1 При поиске существующего вычисляется хеш-функция.
- 2 Определяется место поиска — индекс в хеш-таблице.
- 3 Если по индексу ничего нет, то нет и элемента.
- 4 Иначе по индексу — элемент с нашим ключом — элемент найден.
- 5 Если по индексу — элемент с другим ключом или элемент помечен удалённым, индекс увеличиваем на единицу и переходим к пункту 3.
- 6 Следующий индекс вычисляется по формуле  $(index + 1) \bmod M$ .

# Хеш-таблицы с открытой адресацией

- 1 При вставке вычисляется хеш-функция.
- 2 Определяется место поиска — индекс в хеш-таблице.
- 3 Если по индексу ничего нет или элемент помечен удалённым, то вставляем по индексу и выходим.
- 4 Если по индексу элемент с нашим ключом — меняем данные и выходим.
- 5 Если по индексу элемент с другим ключом то индекс увеличиваем на единицу и переходим к пункту 3.
- 6 Следующий индекс вычисляется по формуле  $(index + 1) \bmod M$ .

# Хеш-таблицы с открытой адресацией

- 1 Почему мы требуем свойства равномерности от хеш-функции.





# Хеш-таблицы с открытой адресацией

- 1 При удалении вычисляется хеш-функция.
- 2 Определяется место поиска — индекс в хеш-таблице.
- 3 Если по индексу ничего нет, то нет и элемента.
- 4 Иначе по индексу — элемент с нашим ключом — элемент найден.
- 5 Если по индексу — элемент с другим ключом, индекс увеличивается на единицу и переходим к пункту 3.
- 6 Следующий индекс вычисляется по формуле  $(index + 1) \bmod M$ .

# Расширение хеш-таблиц

Когда *fill-factor* начинает превосходить 0.5-0.6 таблицу расширяют.

- Создаётся другой массив указателей с нужным размером
- Из оригинального массива в порядке увеличения индексов извлекаются элементы и вставляются в новый массив (таблицу).
- Старый массив удаляется.

## Варианты хеш-таблиц: рехеширование

- В ряде случаев можно уменьшить кластеризацию ключей, используя *рехеширование*.
- При конфликте хеша вычисляется вторая хеш-функция  $S = H_2(key)$  от ключа или его части, которая тоже должна давать число в диапазоне  $[0 \dots M)$ .
- Все методы — вставки, поиска, удаления — немного изменяются.
- Следующие попытки поиска ключа производится по адресам  $(index + k \cdot S) \bmod M$ .
- Для эффективной работы рехеширования требуется, чтобы  $\gcd S, M = 1$ .
- Проще всего добиться этого выбором  $M$  как простого числа.

## Варианты хеш-таблиц: сискоо-хеширование

- Рассмотренный вариант организации хеш-таблиц всегда имеет ровно одно предпочтительное место для ключа.
- При сискоо-хешировании для каждого ключа имеется две таблицы и два места, определяемых двумя хеш-функциями —  $P_1 = H_1(key)$  и  $P_2 = H_2(key)$ .
- Если оказывается коллизия в позиции  $P_1$ , то производится попытка обратиться к  $P_2$ .
- Худший случай: обе позиции заняты.
- Тогда старый ключ *выталкивается*. Новый ключ вставляется на его место и ищется место вставки старого ключа.
- Начинается точно такая-же операция вставки в хеш-таблицу.
- Такая операция повторяется либо до нахождения свободного места для вытолкнутого ключа, либо до появления цикла.
- Появление цикла приводит к перестроению всей таблицы.
- Теоретическая сложность вставки —  $O(1)$  в худшем случае.

Подсчёт амортизационных расходов.

- Амортизационные расходы на закрытую адресацию. Теоретическая сложность в худшем случае  $\Theta(\log N / \log \log N)$ .
- Амортизационные расходы на открытую адресацию. Теоретическая сложность в худшем случае  $\Omega(\log N)$ .
- Амортизационные расходы на рехеширование.

Рехеширование уменьшает потребность в памяти.

Открытые обычно быстрее

# Хеш-таблицы с открытой адресацией

Рекомендации по использованию.

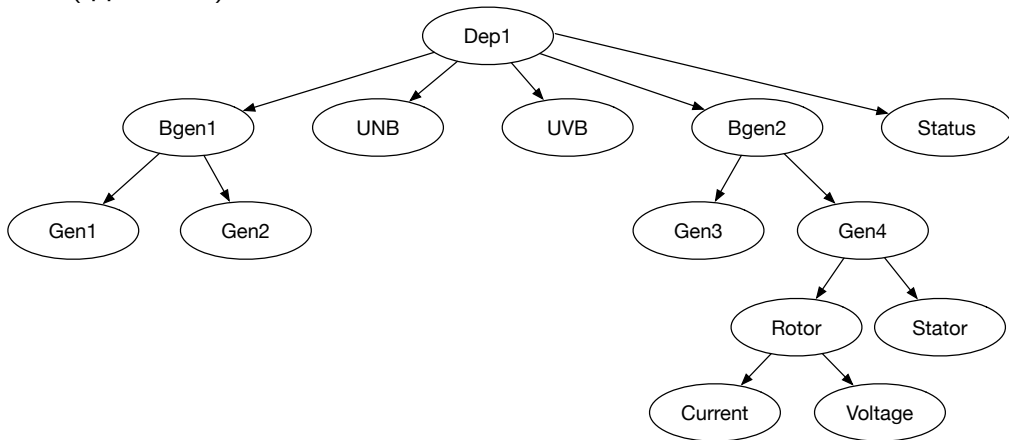
- 1 Всегда использовать хорошую хеш-функцию!
- 2 Использовать *fill-factor* не больше 0.5-0.6.

# Сочетание хеш-таблиц и деревьев

# Хеш-таблицы и деревья

Задача:

- Имеется набор объектов, представляющих иерархию гидроэлектростанции (фрагмент)



Dep1/Bgen/Gen4/Rotor/Voltage=127

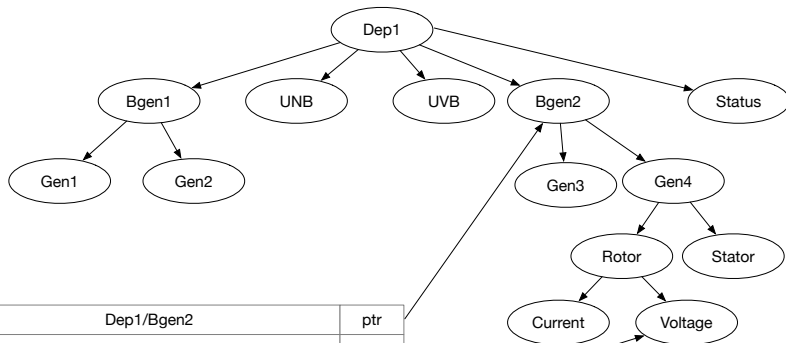


# Хеш-таблицы и деревья

- Дерево представляет объекты в виде указателей.
- Для функционирования важна структура дерева.
- Дерево не является деревом поиска, содержит разное число потомков.
- Поиск в дереве медленный.
- Доступ к любому объекту возможен через полное квалифицированное имя (FQN).
- Не все объекты одинаково часто используются.
- Ресурсы на компьютере сильно ограничены.

# Хеш-таблицы и деревья

- Доступ к элементу через кэш, реализованный в виде хеш-таблицы.



Dep1/Bgen2	ptr
...	
...	
Dep1/Bgen/Gen4/Rotor/Voltage	ptr
...	
...	
...	
Dep1/Bgen/Gen4/Rotor/Current	ptr
...	

# Хеш-таблицы и деревья

- Имеется хеш-таблица небольшого размера, содержащая FQN и указатель на узел дерева.
- При поиске FQN просматривается хеш-таблица. Если такая запись есть — возвращается указатель на объект.
- Если записи нет, то производится поиск по дереву и на место в хеш-таблице записывается новый ключ и найденный указатель.
- При незначительном расходе памяти удалось ускорить амортизированно типичные поиски в несколько раз.

# Хеш-таблицы во внешней памяти

# Хеш-таблицы во внешней памяти

Задача: имеется  $5 \cdot 10^9$  записей, состоящих из уникального ключа размером 129 байтов и данных, размером 260 байтов.

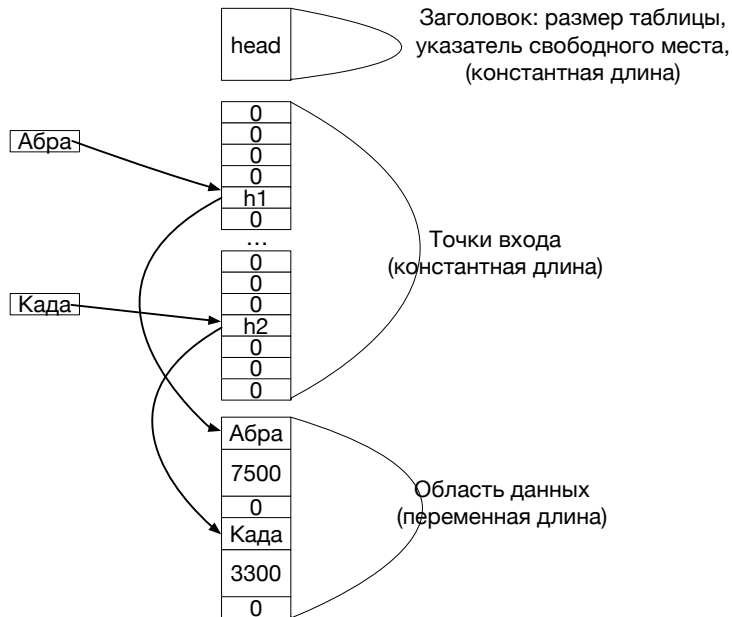
Данные располагаются в 5000000 файлах, в каждом из которых по 1000 строк.

Требуется организовать данные так, чтобы обеспечить быстрый поиск по ключу.

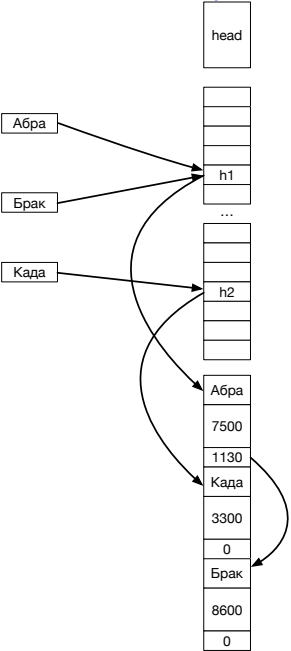
# Хеш-таблицы во внешней памяти

- Общий размер превышает 300GB.
- Поиск нужен будет в непредсказуемое время.
- Количество поисков велико, но много меньше общего числа записей.
- Допустимо хранение результатов преобразования данных на устройстве с произвольным доступом.

# Хеш-таблицы во внешней памяти



# Хеш-таблицы во внешней памяти





# Хеш-таблицы во внешней памяти

- Должны сохраняться при завершении программы (*persistent*)
- Минимизировать количество операций.
- Для оптимизации работы использовать кэширование.

Спасибо за внимание.

Следующая лекция —  
Динамическое программирование.